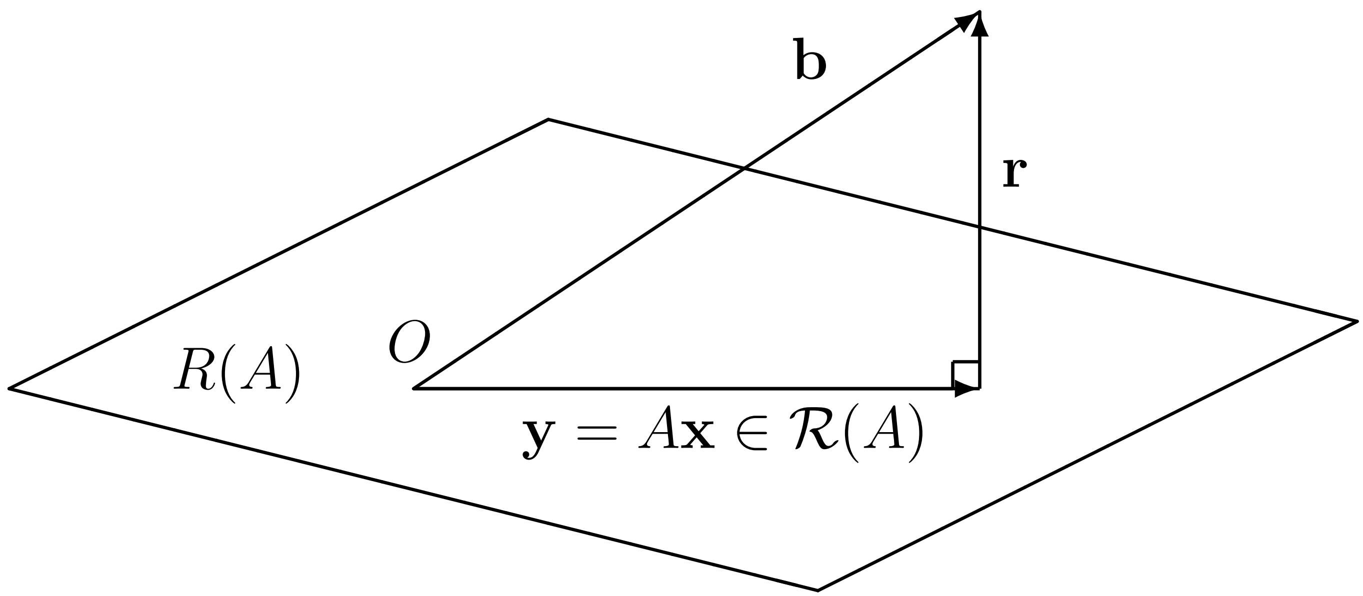
Chapter 3: Linear Least Squares

Outline:

0. Introduction to Projection
1. Least Squares Data Fitting
2. Existence, Uniqueness, and Conditioning
3. Solving Linear Least Squares Problems

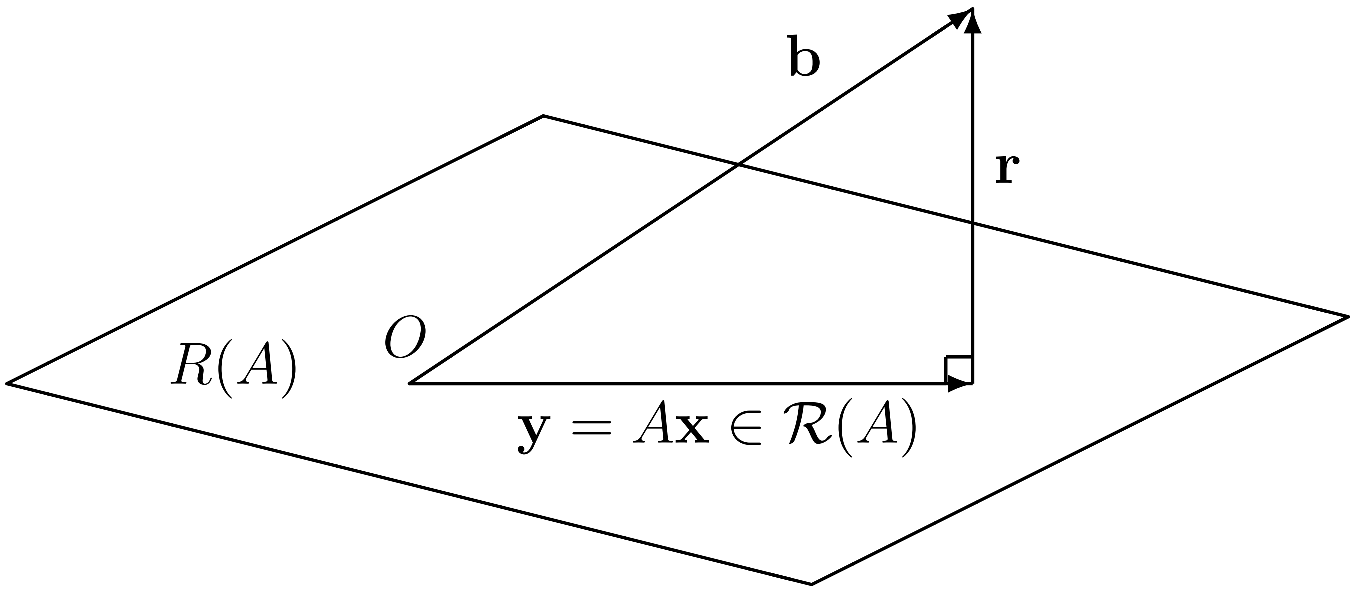
Projection

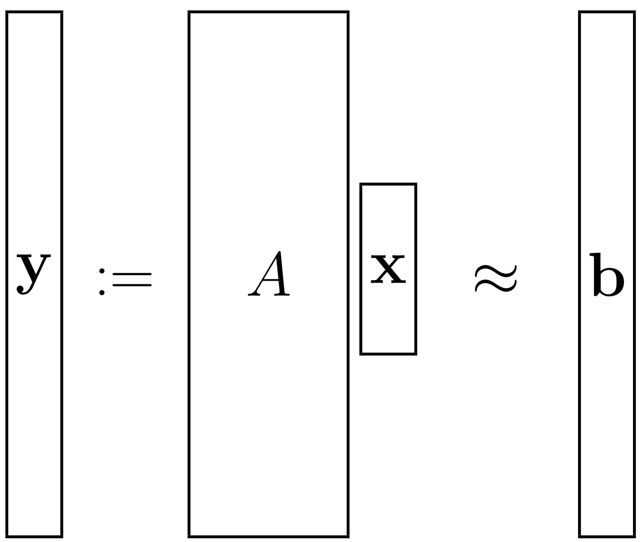


- $\mathbf{b} = \mathbf{r} + A\mathbf{x} \notin \mathcal{R}(A)$
- $\mathbf{y} = A\mathbf{x}$ – *projection of* $\mathbf{b}$ *onto* $\mathcal{R}(A)$
- $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ – *residual vector,* $\perp \mathcal{R}(A)$

Projection, $\mathbf{r} \perp \mathcal{R}(A)$, happens only for a very special choice of $\mathbf{x}$.

Projection



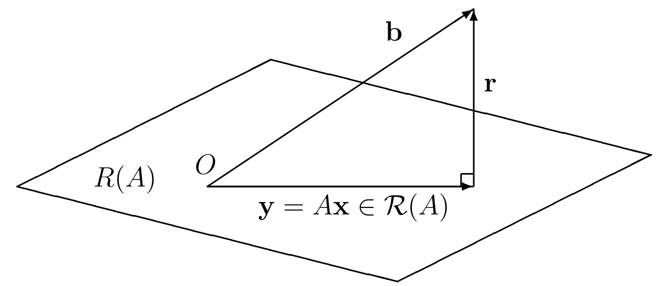


$\mathbf{y}$ is a linear combination of the columns of A:

$$\mathbf{y} \;:=\; A\mathbf{x} \;=\; \mathbf{a}_1 x_1 + \mathbf{a}_2 x_2 + \cdots + \mathbf{a}_n x_n \approx \mathbf{b}$$

$$\mathbf{r} \;:=\; \mathbf{b} - A\mathbf{x} = \mathbf{b} - \mathbf{y}$$

Projection



- With $m > n$, we have:
  - $A = m \times n$ matrix
  - $\mathbf{b}, \mathbf{y} \in \mathbb{R}^m$
  - $\mathbf{x} \in \mathbb{R}^n$ – coefficient set (“*model coefficients*”)
  - $\mathbf{y} = \sum_{j=1}^{m} \mathbf{a}_j x_j$ – best approximation (in least-squares sense)
  - $\mathbf{a}_j$ – model or model basis (*user-prescribed*)

- Remarkably, this chapter focuses on finding $\mathbf{x}$,

$$\mathbf{x} \;=\; \underset{\mathbf{x}' \in \mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{b} - A\mathbf{x}'\|_2,$$

  not on choice of columns of A.

- Both are important.

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

- Effectively, high-dimensional data are projected onto a low-dimensional space to suppress irrelevant detail

- Such projection is conveniently accomplished by the method of *least squares*

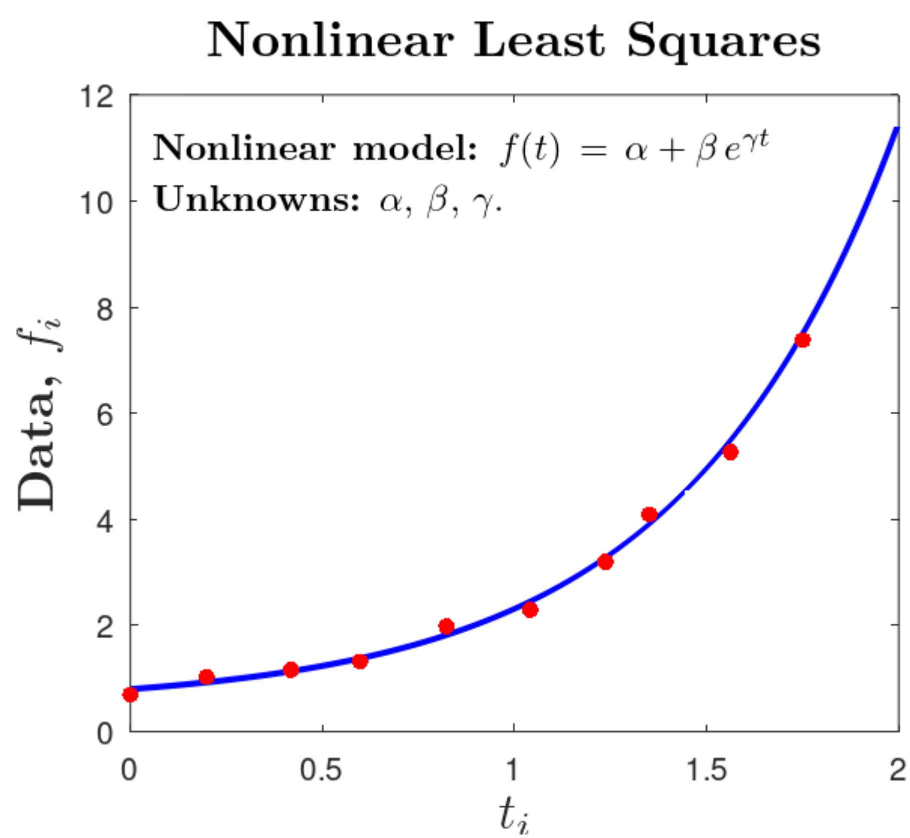
Nonlinear vs. Linear Least Squares

- Starting with some data, f_i, taken at timepoints (say), t_i, $i = 1, \ldots, m$, we might have some physical insight that says we expect f behaves as an exponential in time, such as

$$f(t) = \alpha + \beta e^{\gamma t}.$$

- Such a model is *nonlinear* in at least one of the unknown model parameters (α, β, γ), which makes this a nonlinear least squares problem, to be studied in Chapter 6.

t_i	f_i
0.036650	0.960495
0.218031	0.939770
0.405460	1.213982
0.593674	1.156828
0.832617	1.636737
0.956528	2.425123
1.163127	2.791084
1.410997	4.451842
1.553994	5.522619
1.826442	8.519962



Linear Least Squares

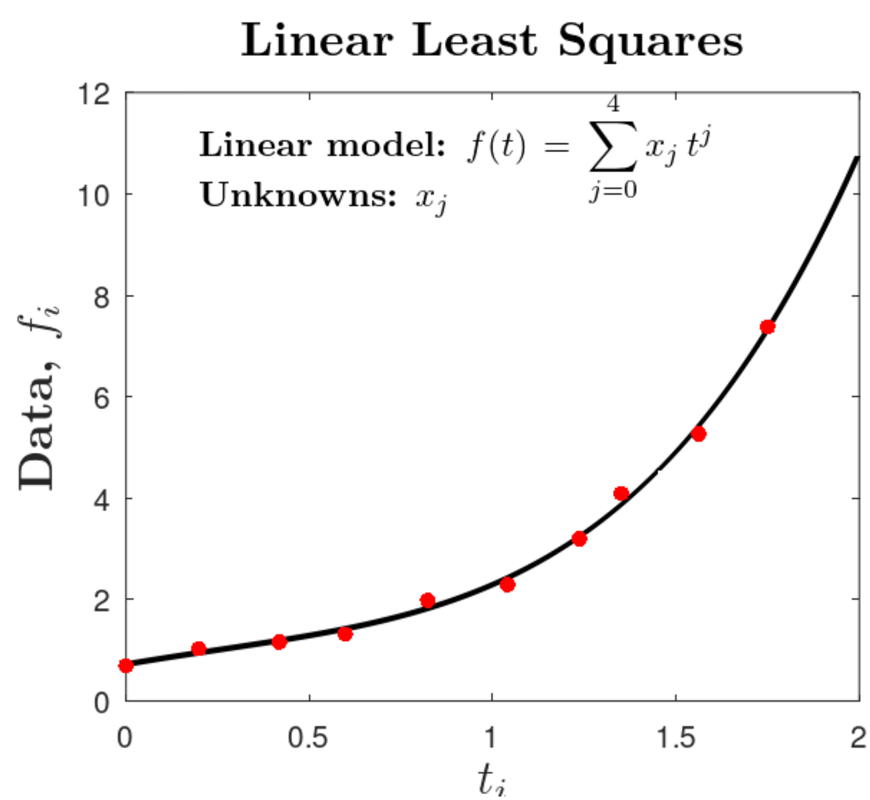
- Alternatively, we can consider a model in which the dependency of $f(t)$ is *linear* in the unknown basis coefficients (i.e., model parameters).

- An example is the polynomial in t given by

$$f(t) = x_0 + x_1 t + x_2 t^2 + x_3 t^3 + x_4 t^4$$

- In this example, we have ten data points (t_i, f_i), $i = 1, \ldots, m$ $(m = 10)$ and only five unknown model parameters, x_j, $j = 0, \ldots, n-1$, with $n = 5$.

t_i	f_i
0.036650	0.960495
0.218031	0.939770
0.405460	1.213982
0.593674	1.156828
0.832617	1.636737
0.956528	2.425123
1.163127	2.791084
1.410997	4.451842
1.553994	5.522619
1.826442	8.519962



Linear Least Squares Example

- To set up the LLSQ (linear least-squares) system, evaluate the basis functions (here, t^j) at timepoints t_i, $i = 1, \ldots, m$ and write down the system we'd like to solve (approximately)

- So, for our polynomial model we'd have

$$x_0 \cdot 1 + x_1 \cdot t_i + x_2 \cdot t_i^2 + x_3 \cdot t_i^3 + x_4 \cdot t_i^4 \approx f_i, \quad i = 1, \ldots, m$$

- For $j = 0, \ldots, 4$, define the jth column of the system matrix $\mathbf{A}$ as t_i^j.

- The resultant system with unknown model coefficients $\mathbf{x} = [x_0,\, x_1, \ldots, x_4]^T$ is,

$$\mathbf{y} \;=\; \mathbf{A}\mathbf{x} \;=\; \underbrace{\begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^4 \\ 1 & t_2 & t_2^2 & \cdots & t_2^4 \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & t_m & t_m^2 & \cdots & t_m^4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_4 \end{bmatrix}}_{\text{model: } \mathbf{y} = \mathbf{A}\mathbf{x} \in \mathbb{P}_4(t_i)} \;\approx\; \underbrace{\begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ \vdots \\ b_m \end{bmatrix}}_{\text{data}}$$

- Then the LLSQ system is $\mathbf{A}\mathbf{x} \;\approx\; \mathbf{b}$, with *data vector* $\mathbf{b} = [b_1,\, b_2, \ldots, b_m]^T$

Linear Least Squares Example, continued

- When the basis functions are *monomials* (i.e., t^j), $\mathbf{A}$ is known as a *Vandermonde matrix.*

- We could also consider a system based on Chebyshev polynomials, defined recursively as

$$T_0(\xi) = 1, \quad T_1(\xi) = \xi, \quad T_k(\xi) = 2\xi T_{k-1}(\xi) - T_{k-2}(\xi),\; k \geq 2$$

- Chebyshev polynomials are orthogonal with respect to a *weighted inner product* on [-1,1].

- For our example, shift $t \in [0,2]$ to $[-1,1]$ by defining $\xi = t - 1$.

- With $\tilde{T}_k(t) := T_k(t-1)$, define the new system as

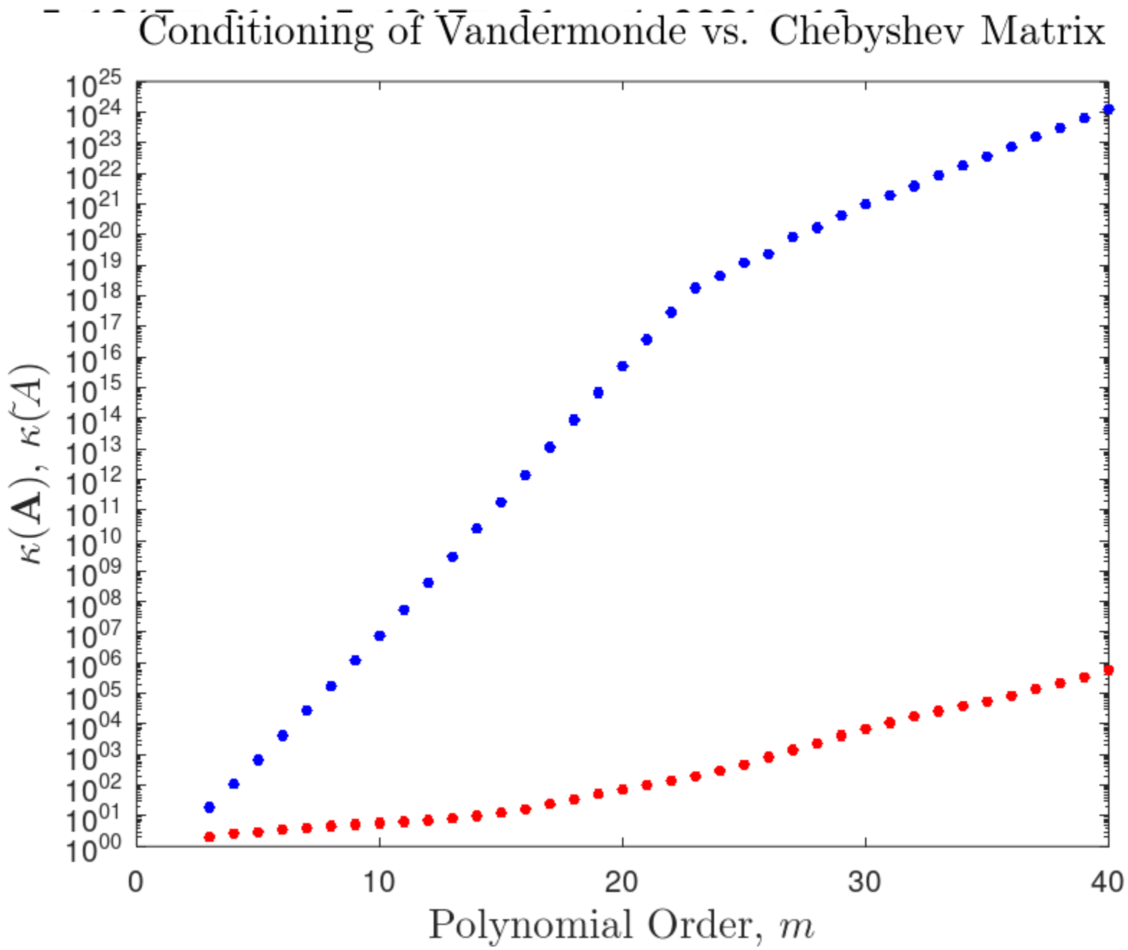
$$\mathbf{y} \;=\; \tilde{\mathbf{A}}\tilde{\mathbf{x}} \;=\; \underbrace{\begin{bmatrix} 1 & \tilde{T}_1(t_1) & \tilde{T}_2(t_1) & \cdots & \tilde{T}_4(t_1) \\ 1 & \tilde{T}_1(t_2) & \tilde{T}_2(t_2) & \cdots & \tilde{T}_4(t_2) \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \tilde{T}_1(t_m) & \tilde{T}_2(t_m) & \cdots & \tilde{T}_4(t_m) \end{bmatrix} \begin{bmatrix} \tilde{x}_0 \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_4 \end{bmatrix}}_{\text{model: } \mathbf{y} = \tilde{\mathbf{A}}\tilde{\mathbf{x}} \in \mathbb{P}_4(t_i)} \;\approx\; \underbrace{\begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ \vdots \\ b_m \end{bmatrix}}_{\text{data}}$$

- Advantage of this approach is that $\tilde{\mathbf{A}}$ generally has a lower condition number than $\mathbf{A}$ because columns of $\tilde{\mathbf{A}}$ are "close" to being orthogonal

- In exact arithmetic, both systems should return the same projection, $\mathbf{y}$.

- They could differ, however, because of potential ill-conditioning of the Vandermonde matrix

k	cond(A)	cond(C)	norm(ra)	norm(rc)	
4.0000e+01	3.7697e+00	1.7388e+00			
3.0000e+00	1.8825e+01	2.0122e+00	2.9048e+00	2.9048e+00	3.7052e-16
4.0000e+00	1.0886e+02	2.6451e+00	8.7323e-01	8.7323e-01	1.3052e-15
5.0000e+00	6.7294e+02	2.9171e+00	5.4034e-01	5.4034e-01	1.1065e-15
6.0000e+00	4.2028e+03	3.5822e+00	5.2602e-01	5.2602e-01	1.5903e-14
7.0000e+00	2.7121e+04	4.0197e+00	5.2579e-01	5.2579e-01	4.4941e-14
8.0000e+00	1.7079e+05	4.6311e+00	5.2579e-01	5.2579e-01	4.2277e-15
9.0000e+00	1.2077e+06	5.2146e+00	5.1571e-01	5.1571e-01	8.8356e-14
1.0000e+01	7.7462e+06	5.7839e+00	5.1360e-01	5.1360e-01	3.5073e-13
1.1000e+01	5.5237e+07	6.4176e+00	5.1290e-01	5.1290e-01	7.6569e-13
1.2000e+01	4.0663e+08	7.1570e+00			
1.3000e+01	2.9783e+09	8.1382e+00			
1.4000e+01	2.4464e+10	9.9329e+00			
1.5000e+01	1.7558e+11	1.2590e+01			
1.6000e+01	1.3627e+12	1.6773e+01			
1.7000e+01	1.1033e+13	2.3955e+01			
1.8000e+01	8.6966e+13	3.4729e+01			
1.9000e+01	6.7061e+14	5.1024e+01			
2.0000e+01	4.9394e+15	7.1866e+01			



demo4/lsq2_test.m

```
hdr; format shorte;

m=90;
r=rand(m,1);
t=2*[0:m-1]'/m;
t=t+.2*(r-.5);
t=max(t,0); t=min(t,2);  t=unique(t);
m=length(t);
xi=(t-1);

b=.5+.3*exp(1.8*t);
r=rand(m,1);
b=b + 0.2*(r-.5);

A=ones(m,2); A(:,2)=t;
C=ones(m,2); C(:,2)=xi;
n=40;

disp(' ')
disp('      k         cond(A)      cond(C)      norm(ra)     norm(rc)')
ca = cond(A); cc = cond(C); disp([k ca cc])

hold off
for k=3:n;
  A  = [A t.^(k-1)];
  C  = [C 2*xi.*C(:,k-1)-C(:,k-2)];

  xa = A\b; ya = A*xa; ra=b-ya; na=norm(ra);
  xc = C\b; yc = C*xc; rc=b-yc; nc=norm(rc);
  dc = yc-ya;                   nd=norm(dc)/norm(b);

  ca = cond(A); cc = cond(C); disp([k ca cc na nc nd])
  semilogy(k,ca,'b.',ms,12,k,cc,'r.',ms,12); hold on
end;
xlabel('Polynomial Order, $m$',intp,ltx,fs,14);
ylabel('$\kappa({\bf A}),$ $\kappa({\tilde \bf A})$',intp,ltx,fs,14);
title('Conditioning of Vandermonde vs. Chebyshev Matrix',intp,ltx,fs,14);
```

Solving the Linear Least Squares System

- Here, we have an *overdetermined* linear system,

$$\mathbf{A}\mathbf{x} \approx \mathbf{b}$$

  with an $m \times n$ matrix, $\mathbf{A}$, $m > n$.

- We have more equations than we do unknowns and in general cannot hope to solve all of them.

- The *least squares* idea is to find $\mathbf{x}$ such that we minimizes the Euclidean norm of the *residual vector*, $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$,

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

- As mentioned earlier, minimizing the 2-norm is equivalent to finding an orthogonal projection, which is in fact the way we typically formulate and solve the LLSQ systems.

- Let's proceed with formulating the question as a minimization problem.

Residual Minimization

- Consider LLSQ $\mathbf{A}\mathbf{x} \approx \mathbf{b}$ and associated *objective function*, $\phi(\mathbf{x}) := \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2$. Assume that $\mathbf{A}$ has full rank. Does this always have a solution?

- Yes. $\phi \geq 0$, $\phi \longrightarrow \infty$ as $\|\mathbf{x}\| \longrightarrow \infty$, ϕ is continuous, $\Longrightarrow \phi$ has a mininum.

- Is it always unique? Yes (again, assuming full rank)

- What happens if $\mathbf{A}$ does not have full rank? Then there is a nullspace such that $A\mathbf{n} = 0$ for any vector $\mathbf{n}$ in the nullspace. Thus, if $\mathbf{x}$ is a solution, then $\|\mathbf{b}-\mathbf{A}(\mathbf{x}+\mathbf{n})\|_2 = \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2$

- Note that the *projection*, $\mathbf{y} = \mathbf{A}(\mathbf{x} + \mathbf{n}) = \mathbf{A}\mathbf{x}$, is unchanged (i.e., it *is* unique)

Residual Minimization, continued

- To find the minimize, $\mathbf{x}$, evaluate the objective function and its gradient:

$$\begin{aligned}\phi(\mathbf{x}) &= \|\mathbf{r}\|_2^2 = \|\mathbf{b}-\mathbf{y}\|_2^2 = \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2^2 \\ &= (\mathbf{b}-\mathbf{A}\mathbf{x})^T(\mathbf{b}-\mathbf{A}\mathbf{x}) \\ &= \mathbf{b}^T\mathbf{b} - \mathbf{x}^T\mathbf{A}\mathbf{b} - \underbrace{\mathbf{b}^T\mathbf{A}\mathbf{x}}_{\mathbf{x}^T\mathbf{A}\mathbf{b}} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x}\end{aligned}$$

- Minimum where gradient $\phi = 0$:

$$[\nabla\phi]_k := \frac{\partial\phi}{\partial x_k} = 0, \quad k = 1, \ldots, n$$

- Differentiate term-by-term

$$\frac{\partial}{\partial x_k}\mathbf{b}^T\mathbf{b} = 0$$

$$\frac{\partial}{\partial x_k}\mathbf{x}^T\mathbf{A}^T\mathbf{b} = \frac{\partial}{\partial x_k}\mathbf{x}^T\mathbf{c} = \frac{\partial}{\partial x_k}\sum_{j=1}^{n} x_j c_j = c_k, \qquad \mathbf{c} := A^T\mathbf{b}$$

$$\frac{\partial}{\partial x_k}\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} = \frac{\partial}{\partial x_k}\mathbf{x}^T\mathbf{H}\mathbf{x} = \frac{\partial}{\partial x_k}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i H_{ij} x_j \qquad \mathbf{H} := A^T\mathbf{A}$$

$$= \sum_{j=1}^{n} H_{kj} x_j + \underbrace{\sum_{i=1}^{n} x_i H_{ik}}_{\sum_{j=1}^{n} H_{kj} x_j} \qquad H_{ij} = H_{ji}$$

$$= 2\sum_{j=1}^{n} H_{kj} x_j$$

Normal Equations

- Combining the results,

$$\begin{aligned} 0 &= 2\left[\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}\right]_k, \qquad k = 1, \ldots, n \\ 0 &= \mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b} \end{aligned}$$

- Thus, $\nabla\phi(\mathbf{x}) = 0$ yields the ***Normal Equation***

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

- Solution is

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

  What is the shape of $\mathbf{A}^T\mathbf{A}$?

- Does it always have an inverse?

- **Yes.**, if $\mathbf{A}$ is full rank. In this case, $\mathbf{A}^T\mathbf{A}$ is SPD *but not necessarily well-conditioned.*

Pseudoinverse and Condition Number

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

$$\mathbf{A}^{+} = \left(\mathbf{A}^{T}\mathbf{A}\right)^{-1}\mathbf{A}^{T}$$

  and condition number by

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}^{T}\|_2 \cdot \|\mathbf{A}^{+}\|_2$$

- By convention, $\text{cond}(\mathbf{A}) = \infty$ if $\text{rank}(\mathbf{A}) < n$
- Just as condition number of a square matrix measures closeness to singularity, condition number of a rectangular matrix measures closeness to rank deficiency
- Least squares solution of $\mathbf{A}\mathbf{x} \approx \mathbf{b}$ is given by $\mathbf{x} = \mathbf{A}^{+}\mathbf{b}$

Sensitivity and Conditioning

- Sensitivity of LLSQ $\mathbf{Ax} \approx \mathbf{b}$ depends on $\mathbf{b}$ as well as $\mathbf{A}$.

- Define θ as angle between $\mathbf{b}$ and $\mathbf{y} = \mathbf{Ax}$ by

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

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

$$\frac{\|\Delta\mathbf{x}\|_2}{\|\mathbf{x}\|_2|} \leq \operatorname{cond}(\mathbf{A}) \frac{1}{\cos(\theta)} \frac{\|\Delta\mathbf{b}\|_2}{\|\mathbf{b}\|_2|}$$

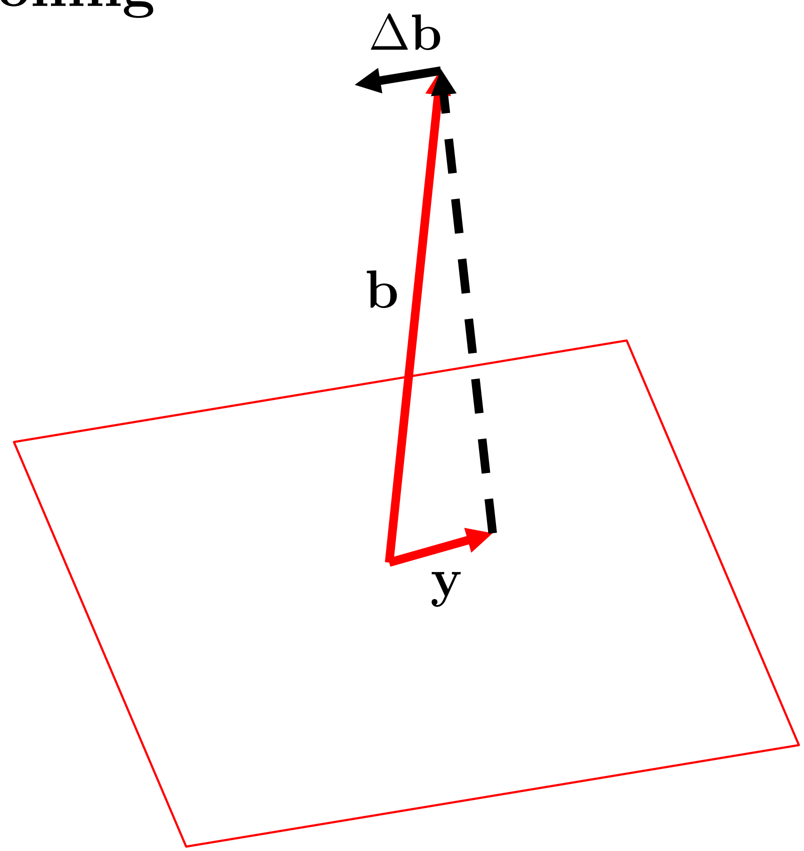
Mathematics & Geometry of LSQ Conditioning

$$\Delta\mathbf{y} \;=\; A\Delta\mathbf{x} \;\approx\; \Delta\mathbf{b}, \text{ if } \Delta\mathbf{b} \in \mathcal{R}(A)$$

$$\|\Delta\mathbf{x}\| \;\leq\; \|A^{\dagger}\|\,\|\Delta\mathbf{b}\|$$

$$\|\mathbf{y}\| \;=\; \|A\mathbf{x}\| \;=\; \cos\theta\,\|\mathbf{b}\|$$

$$\Longrightarrow 1 \;=\; \frac{\|A\mathbf{x}\|}{\cos\theta\,\|\mathbf{b}\|}$$



$$\|\Delta\mathbf{x}\| \;\leq\; \|A^{\dagger}\|\,\|\Delta\mathbf{b}\|\,\frac{\|A\mathbf{x}\|}{\cos\theta\,\|\mathbf{b}\|}$$

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \;\leq\; \|A^{\dagger}\|\|A\|\,\frac{\|\Delta\mathbf{b}\|}{\cos\theta\,\|\mathbf{b}\|}$$

$$\;=\; \mathrm{cond}(A)\,\frac{1}{\cos\theta}\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

- $\Delta\mathbf{b}$ is small with respect to $\mathbf{b}$, but not relative to $\mathbf{y}$.
- The ill-conditioning arises when a large part of $\mathbf{b}$ has no influence on $\mathbf{y}$.
- That is, when $\mathbf{b}$ is nearly orthogonal to $\mathcal{R}(A)$.

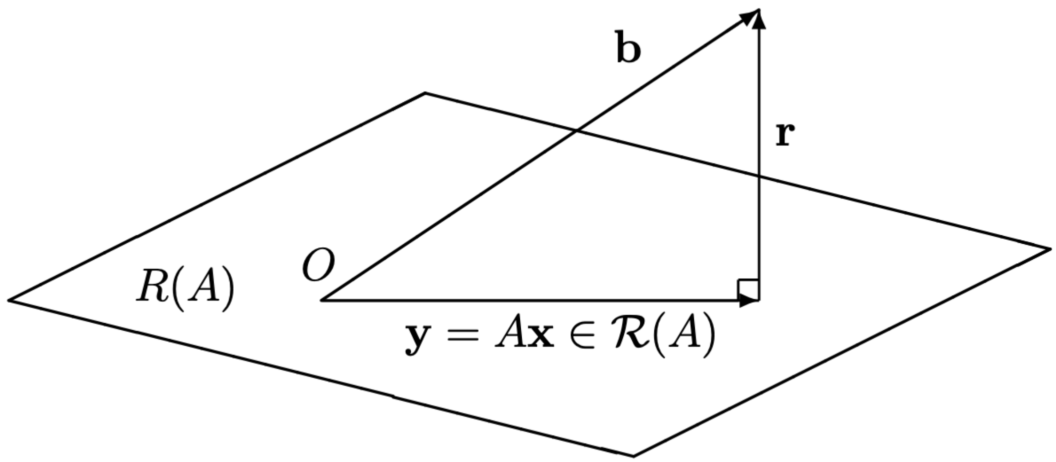
Similarly, for perturbation $\boldsymbol{E}$ in matrix $\boldsymbol{A}$,

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

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

Least Squares Viewed as Orthogonal Projection

- Recall our earlier picture.



- Geometrically, we have $(\mathbf{b}-\mathbf{y}) \perp \mathbf{A} \iff (\mathbf{b}-\mathbf{y}) \perp \mathbf{a}_j \;\; j=1,\ldots,n.$

- In matrix form,

$$\mathbf{A}^T(\mathbf{b}-\mathbf{y}) \;=\; 0$$

$$\mathbf{A}^T\mathbf{y} \;=\; \mathbf{A}^T\mathbf{b}$$

$$\mathbf{A}^T\mathbf{A}\mathbf{x} \;=\; \mathbf{A}^T\mathbf{b}$$

- Normal equations!

Least Squares Viewed as Orthogonal Projection

- Note that

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

$$\mathbf{u} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \mathbf{P}\mathbf{b}$$

- Here, we define $\mathbf{P} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ as the *orthogonal projector* onto the column space of $\mathbf{A}$ (i.e., $\mathcal{R}(\mathbf{A})$)

- Note that $\mathbf{P} = \mathbf{P}^2$, which is an intrinsic property of square projection matrices.

- To illustrate,

$$\begin{aligned}\mathbf{P}^2 &= \left(\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\right) \cdot \left(\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\right) \\ &= \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{A})(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \\ &= \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \mathbf{P}\end{aligned}$$

- Geometrically, $\mathbf{P}^2 = \mathbf{P}$ simply says that *once* $\mathbf{b}$ *has been projected onto* $\mathcal{R}(\mathbf{A})$, *the projection of the result* ($\mathbf{y}$) *is unchanged.*

- If $\mathbf{P} = \mathbf{P}^T$ we refer to $\mathbf{P}$ as an **orthogonal projector**

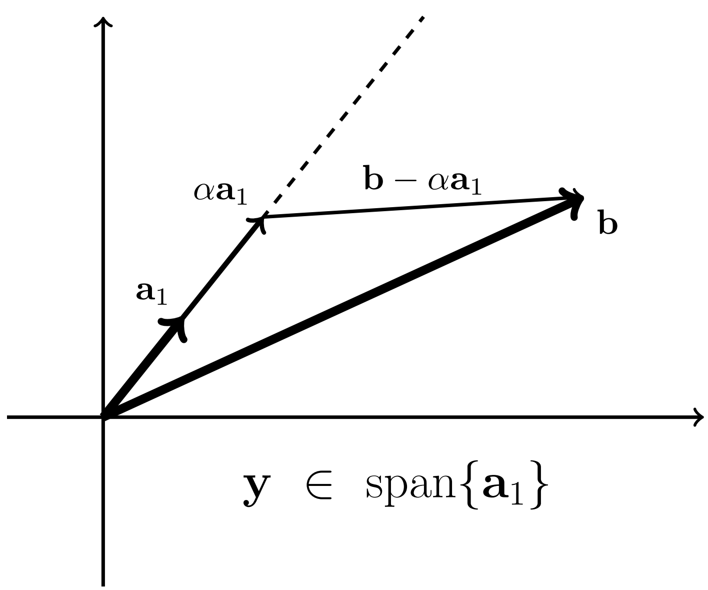
1D Projection

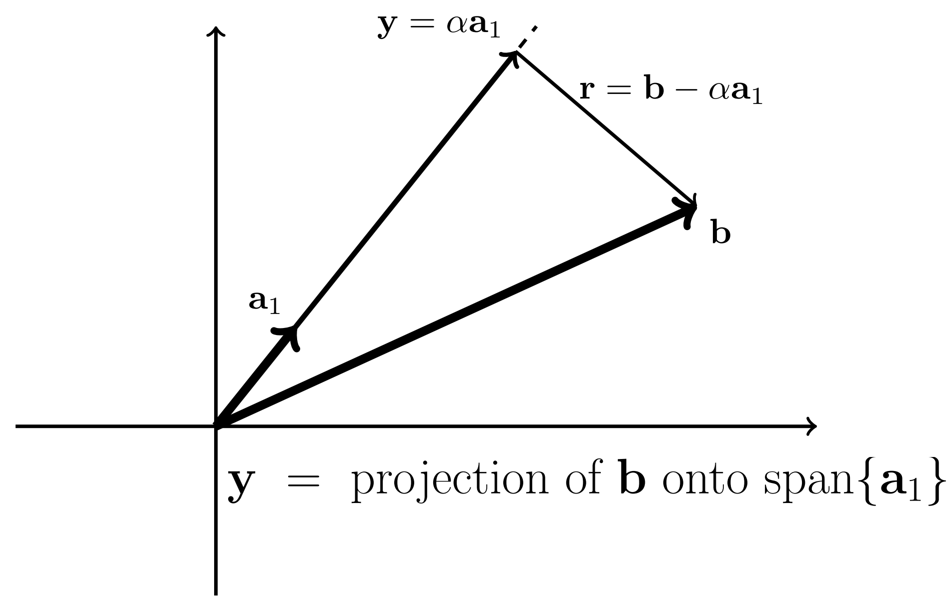
- Consider the 1D subspace of $\mathbb{R}^2$ spanned by $\mathbf{a}_1$:

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

- 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

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

- 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 also that the denominator $\mathbf{a}_1^T\mathbf{a}_1 > 0$ unless $\mathbf{a}_1 = 0$.

Examples

- Find the projection of $\mathbf{b}$ onto $\mathcal{R}(A)$ for the following cases.

$$A = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

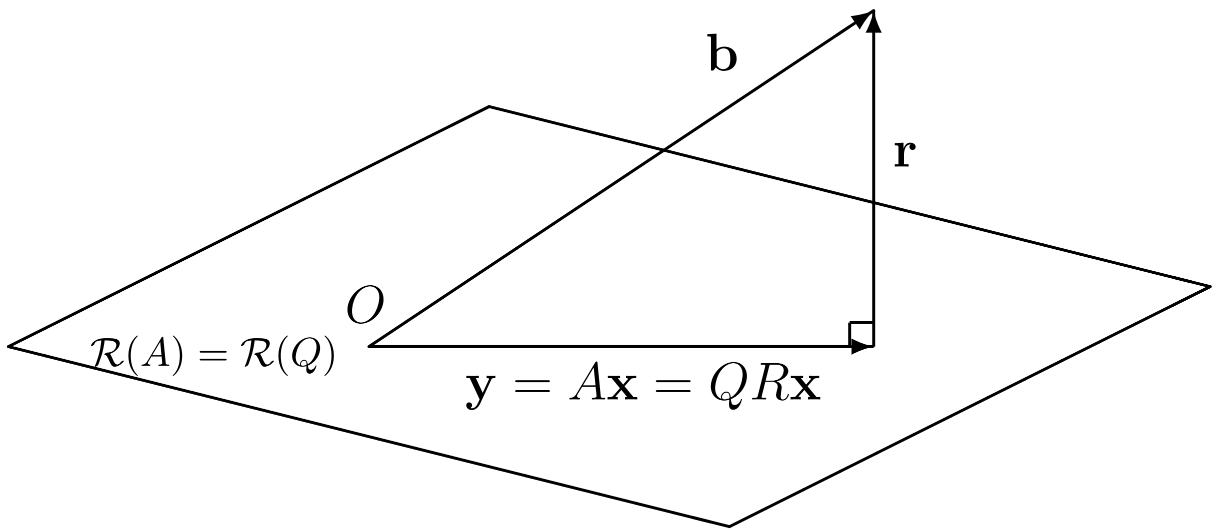
$$A = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

$$A = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 30 \\ 40 \end{pmatrix}$$

$$A = \begin{bmatrix} 3 \\ 4 \end{bmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 30 \\ 40 \end{pmatrix}$$

$$A = \begin{bmatrix} 3 \\ 4 \end{bmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 10 \\ 0 \end{pmatrix}$$

Projection via QR factorization



- Find matrix Q whose columns span $\mathcal{R}(A)$ such that $\mathbf{q}_i \perp \mathbf{q}_j$ (columns are orthogonal).
- Normalize each $\mathbf{q}_j$ to have unit length such that $\mathbf{q}_i^T \mathbf{q}_j = \delta_{ij}$.
- Q is referred to as an *orthogonal* matrix: $Q^T Q = I$.
- R will be square upper triangular and invertible if columns of A are linearly independent.

Example

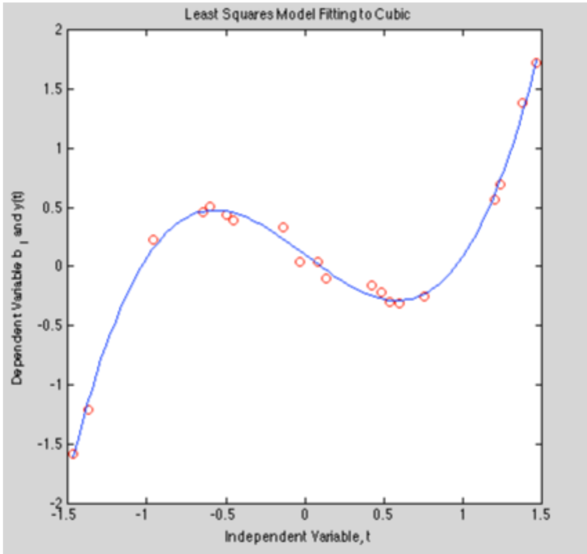
- Suppose we have observational data, { b_i } at some independent times { t_i } (red circles).
- The t_i s do not need to be sorted and can in fact be repeated.

- 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...)

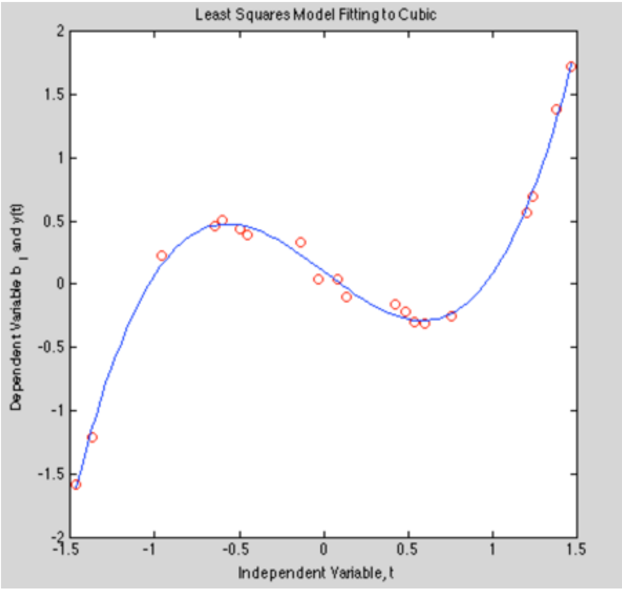
- We then set $\{\underline{a}_j\}_i = \phi_j(t_i)$ for each column $j = 1, \ldots, n$.

- 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 – Normal Eqn (bad) Approach

```
% 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)')
```

Normal Equations Method

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

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

  to obtain solution $\mathbf{x}$ to system of normal equations,

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

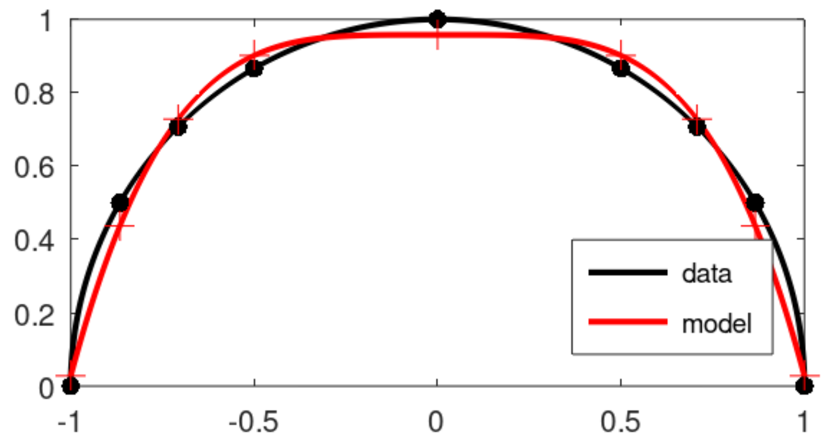
  which gives the solution to the LLSQ problem $\mathbf{A}\mathbf{x} \approx \mathbf{b}$

- Normal equations approach involves transformations

$$\text{rectangular} \longrightarrow \text{square} \longrightarrow \text{triangular}$$

Normal Equations Example

- Consider trying to fit a 4th-order polynomial of the form $y(t) = a + bt^2 + ct^4$ to a semi-circle on [-1,1].



- Here, we leverage the fact that the semi-circle has even symmetry so we do not need the linear or cubic terms in our polynomial expansion.
- The columns of $\mathbf{A}$ are therefore $\mathbf{a}_1 = 1$, $\mathbf{a}_2 = [t_i^2]$, and $\mathbf{a}_3 = [t_i^4]$, evaluated at $t_i = [-1 \;\; -\sqrt{3}/2 \;\; -\sqrt{2}/2 \; 0 \; \sqrt{2}/2 \; \sqrt{3}/2 \; 1]$

$$\mathbf{A} = \begin{bmatrix} 1.0000 & 1.0000 & 1.0000 \\ 1.0000 & 0.7500 & 0.5625 \\ 1.0000 & 0.5000 & 0.2500 \\ 1.0000 & 0.2500 & 0.0625 \\ 1.0000 & 0 & 0 \\ 1.0000 & 0.2500 & 0.0625 \\ 1.0000 & 0.5000 & 0.2500 \\ 1.0000 & 0.7500 & 0.5625 \\ 1.0000 & 1.0000 & 1.0000 \end{bmatrix}$$

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 9.000 & 5.000 & 3.750 \\ 5.000 & 3.750 & 3.125 \\ 3.750 & 3.125 & 2.766 \end{bmatrix} = \mathbf{L}\mathbf{L}^T$$

$$\mathbf{L} = \begin{bmatrix} 3.000 & 0 & 0 \\ 1.667 & 0.987 & 0 \\ 1.250 & 1.056 & 0.295 \end{bmatrix}$$

Normal Equations Example

- Solving lower triangular system $\mathbf{Lz} = \mathbf{A}^T\mathbf{b}$ with forward substitution yields $\mathbf{z} = [1.7154 \text{ -}0.9827 \text{ -}0.2774]^T$

- Solving upper triangular system $\mathbf{L}^T\mathbf{x} = \mathbf{z}$ with backward substitution yields $\mathbf{x} = [\ 0.957585\ 0.010732\ \text{-}0.940176\ ]^T$

- We can the plot the model $\mathbf{y}$ which has the same number of entries as $\mathbf{b}$. However, we can also plot a *finely sampled* model, $y(\mathbf{t}_f)$, because $y(t)$ is continuous in t.

```
hdr

s2=sqrt(2)/2; s3=sqrt(3)/2;
t=[-1 -s3 -s2 -.5 0 .5 s2 s3 1]'; m=length(t);
theta = acos(t); b=sin(theta);

A=ones(m,3); A(:,2) = t.^2; A(:,3) = t.^4;

AtA=A'*A
L=chol(AtA)'
z=L\(A'*b); x=(L')\z
y = A*x;

% Sample model and function on fine mesh for plotting
th = pi*[0:200]'/200;   tf=cos(th);
Af = ones(201,1); Af(:,2)=tf.^2; Af(:,3)=tf.^4;
mf = Af*x; bf = sqrt(1-tf.*tf);
plot(tf,bf,'k-',lw,2,tf,mf,'r-',lw,2,t,b,'k.',ms,19,t,y,'r+',ms,10);
axis equal; axis([-1 1 0 1]); legend('data','model','location','southeast')
```

Shortcomings of the Normal Equations

- Information can be lost in forming $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$
- For example, consider, for $0 < \epsilon < \sqrt{\epsilon_M}$,

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

- In floating point arithmetic the SPD matrix $\mathbf{A}^T\mathbf{A}$ evaluates to a singular system

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

- Sensitivity is also worsened since, in general,

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

- Avoid normal equations:
  - $A^T A x = A^T b$
- *Instead, orthogonalize columns of* $A = QR$
  - *Columns of* Q *are orthonormal* → $Q^T Q = I$
  - R *is upper triangular*
  - $Rx = Q^T Q R x = Q^T b$
  - $Q^T (Ax - b) = 0$

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

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

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

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,

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

  and the system is easily solved,

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

- 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 $[0, \pi]$:

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

In this case,

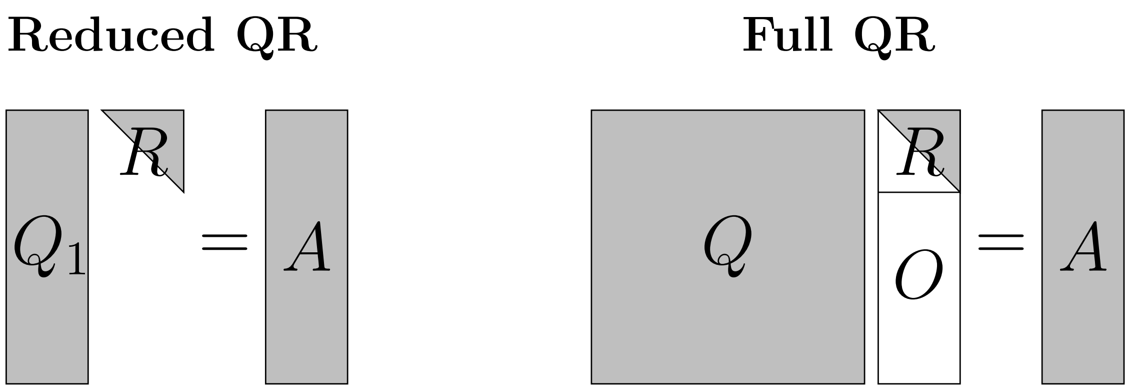
$$A \;=\; (\; \phi_1(t_i) \;\; \phi_2(t_i) \;\; \cdots \;\; \phi_n(t_i) \;)\,,$$

$$A^T A \;=\; \frac{n}{2} I.$$

Stop Here

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:



- 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.

…but maybe not the most stable

- We are interested in generating orthogonal subspaces that match the nested column spaces of A,

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

QR Factorization: Gram-Schmidt

- It's clear that the conditions

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

  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.

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

Orthogonal Projector onto $R(\mathbf{q}_1 \ldots \mathbf{q}_{k-1}) = R(\mathbf{a}_1 \ldots \mathbf{a}_{k-1})$

$$\text{for } k = 2, \ldots, n-1$$

$$\mathbf{v}_k = \mathbf{a}_k - P_{k-1}\,\mathbf{a}_k = (I - P_{k-1})\,\mathbf{a}_k = P_{\perp,k-1}\,\mathbf{a}_k$$

$$\mathbf{q}_k = \frac{\mathbf{v}_k}{||\mathbf{v}_k||} = \frac{P_{\perp,k-1}\mathbf{a}_k}{||P_{\perp,k-1}\mathbf{a}_k||}$$

$$\text{end}$$

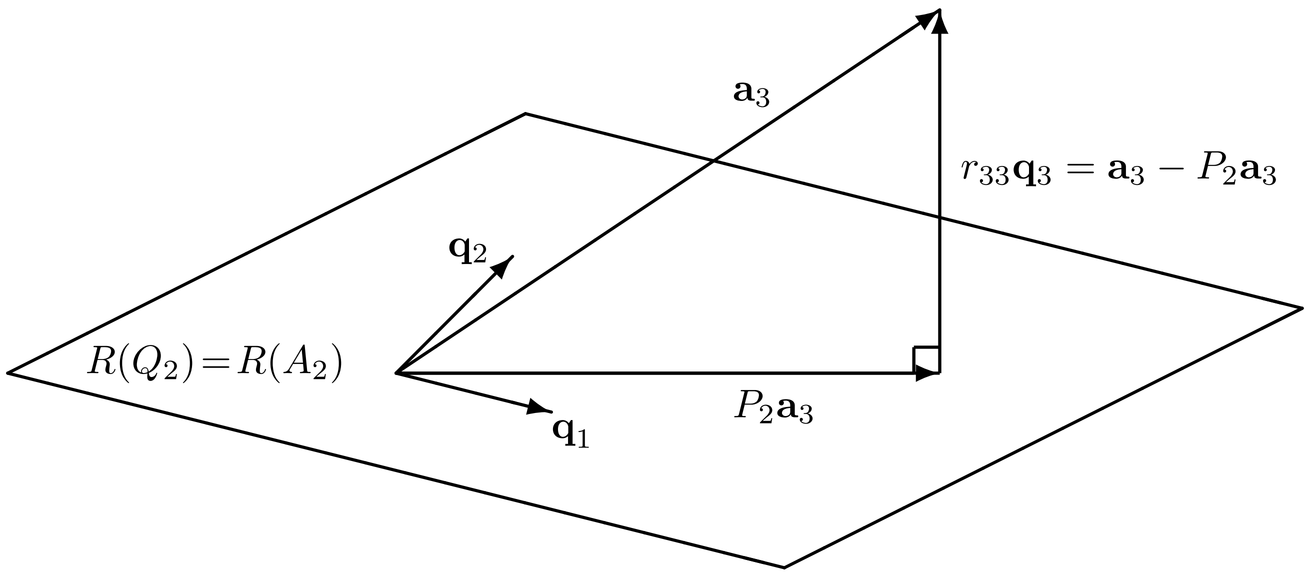
- This is *Gram-Schmidt orthogonalization.*

- Each new vector $\mathbf{q}_k$ starts with $\mathbf{a}_k$ and subtracts off the projection onto $\mathcal{R}(Q_{k-1})$, followed by normalization.

Classical Gram-Schmidt Orthogonalization

$A_k := [\mathbf{a}_1 \cdots \mathbf{a}_k]$

$Q_k := [\mathbf{q}_1 \cdots \mathbf{q}_k]$



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

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, $\mathbf{v}_k = \mathbf{a}_k \; - \; P_{k-1}\,\mathbf{a}_k$.
- The classical (unstable) GS projection is executed as

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

- The modified GS projection is executed as

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

Mathematical Difference Between CGS and MGS

- Let $\tilde{P}_{\perp,k}, := I - \mathbf{q}_k \mathbf{q}_k^T$
- The CGS projection step amounts to

$$\begin{aligned}
\mathbf{v}_k &= \left(\tilde{P}_{\perp,k-1}\, \tilde{P}_{\perp,k-2} \cdots \tilde{P}_{\perp,1} \right) \mathbf{a}_k \\
&= \left(I - \tilde{P}_1 - \tilde{P}_2 - \cdots - \tilde{P}_{k-1} \right) \mathbf{a}_k \\
&= \mathbf{a}_k - \tilde{P}_1 \mathbf{a}_k - \tilde{P}_2 \mathbf{a}_k - \cdots - \tilde{P}_{k-1} \mathbf{a}_k \\
&= \mathbf{a}_k - \sum_{j=1}^{k-1} \tilde{P}_j\, \mathbf{a}_k .
\end{aligned}$$

- The MGS projection step is equivalent to

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

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, $\mathbf{r}_k := \mathbf{a}_k - P_{k-1}\mathbf{a}_k$.
- 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.)

CGS: Classical Gram-Schmidt Orthogonalization

- The CGS algorithm proceeds as follows

$$
\begin{array}{l}
\text{for } k = 1 \text{ to } n \\
\quad \mathbf{q}_k = \mathbf{a}_k \\
\quad \text{for } j = 1 \text{ to } k-1 \\
\quad\quad r_{jk} = \mathbf{q}_j^T \mathbf{a}_k \\
\quad\quad \mathbf{q}_k = \mathbf{q}_k - \mathbf{q}_j r_{jk} \qquad\qquad (\textit{project } \mathbf{q}_k \textit{ onto } Q_{k-1}^{\perp}) \\
\quad \text{end} \\
\quad r_{kk} = \|\mathbf{q}_k\|_2 \\
\quad \mathbf{q}_k = \mathbf{q}_k / r_{kk} \\
\text{end}
\end{array}
$$

- Resulting $\mathbf{q}_k$ and r_{jk} yield *reduced* QR factorization of $\mathbf{A}$

Pros/Cons of Classical Gram-Schmidt

- The CGS algorithm can suffer loss of orthogonality in finite precision.
- Nominally requires separate storage for $\mathbf{A}$ and $\mathbf{Q}$ (but this likely can be avoided)
- These deficiencies can be addressed with *modified Gram-Schmidt*, which also allows column pivoting (Bjork)
- We will see, however, that other factors can come into play in the CGS/MGS evaluation

MGS: Modified Gram-Schmidt Orthogonalization

- The MGS algorithm proceeds as follows

$$
\begin{aligned}
&\text{for } k = 1 \text{ to } n \\
&\quad \mathbf{q}_k = \mathbf{a}_k \\
&\quad \text{for } j = 1 \text{ to } k-1 \\
&\quad\quad r_{jk} = \mathbf{q}_j^T \mathbf{q}_k \\
&\quad\quad \mathbf{q}_k = \mathbf{q}_k - \mathbf{q}_j r_{jk} \qquad\qquad (\textit{project } \mathbf{q}_k \textit{ onto } Q_{k-1}^{\perp}) \\
&\quad \text{end} \\
&\quad r_{kk} = \|\mathbf{q}_k\|_2 \\
&\quad \mathbf{q}_k = \mathbf{q}_k / r_{kk} \\
&\text{end}
\end{aligned}
$$

- Resulting $\mathbf{q}_k$ and r_{jk} yield *reduced* QR factorization of $\mathbf{A}$

CGS/MGS Code Comparison

CGS:

for $k = 1$ to n
$\quad \mathbf{q}_k = \mathbf{a}_k$
$\quad$ for $j = 1$ to $k-1$
$\quad\quad r_{jk} = \mathbf{q}_j^T \mathbf{a}_k$
$\quad\quad \mathbf{q}_k = \mathbf{q}_k - \mathbf{q}_j r_{jk}$
$\quad$ end
$\quad r_{kk} = \|\mathbf{q}_k\|_2$
$\quad \mathbf{q}_k = \mathbf{q}_k / r_{kk}$
end

MGS:

for $k = 1$ to n
$\quad \mathbf{q}_k = \mathbf{a}_k$
$\quad$ for $j = 1$ to $k-1$
$\quad\quad r_{jk} = \mathbf{q}_j^T \mathbf{q}_k$
$\quad\quad \mathbf{q}_k = \mathbf{q}_k - \mathbf{q}_j r_{jk}$
$\quad$ end
$\quad r_{kk} = \|\mathbf{q}_k\|_2$
$\quad \mathbf{q}_k = \mathbf{q}_k / r_{kk}$
end

- CGS uses a static projection, with coefficents based only on $\mathbf{q}_j^T \mathbf{a}_k$
- MGS computes the projection $(\mathbf{I} - \mathbf{Q}_{k-1}\mathbf{Q}_{k-1}^T)\mathbf{a}_k$ based on successive removal of components in directions $\mathbf{q}_j$, $j = 1, \ldots, k-1$.

CGS/MGS Comparison

- Here we consider an example with the Vandermonde matrix for $t \in [0:29]$ up to polynomial order 9

- As a consequence, $\text{cond}(\mathbf{A}) \approx 10^{13}$, which stresses the orthogonalization process in QR factorization

demo5/

- *cgs_mgs.m*
- *cgs2 - two-pass*
- *cg2*

```
hdr;

m=30; n=10; t=[0:m-1];
A=ones(m,n);   % Form Vandermonde matrix
for j=2:n;
  A(:,j) = t.^(j-1);
end;
cond(A)

Q=A; R=zeros(n,n);
for k=1:n                    %% MGS
    qk=A(:,k);
    for j=1:k-1;
        R(j,k) = Q(:,j)'*qk;
        qk     = qk - Q(:,j)*R(j,k);
    end;
    R(k,k)=norm(qk,2); Q(:,k)=qk/R(k,k);
end;
Qmgs = Q; Rmgs = R;

for k=1:n                    %% CGS
    ak=A(:,k); qk=ak;
    for j=1:k-1;
        R(j,k) = Q(:,j)'*ak;
        qk     = qk - Q(:,j)*R(j,k);
    end;
    R(k,k)=norm(qk,2); Q(:,k)=qk/R(k,k);
end;
Qcgs = Q; Rcgs = R;


%% Test orthogonality
I=eye(n);
QQc = Qcgs'*Qcgs; Tc=QQc-I; tcn = norm(Tc)
QQm = Qmgs'*Qmgs; Tm=QQm-I; tmn = norm(Tm)

%% Test LLSQ solution
b=rand(m,1); x = A \ b;
xc=Rcgs\(Qcgs'*b); xm=Rmgs\(Qmgs'*b);
ec=norm(x-xc)/norm(x), em=norm(x-xm)/norm(x)
```

Two-Pass Classical Gram-Schmidt

- A significant advantage of CGS is that the coefficients $r_{jk} = \mathbf{q}_j^T \mathbf{a}_k$, $j = 1, \ldots, k-1$, can be computed all at once, which is important in parallel computing because dot-products (or any *vector reduction,* $\mathbb{R}^n \longrightarrow \mathbb{R}$) require global communication which typically has a communication cost

$$t_{\text{comm}} \approx 2\alpha \log_2 P$$

$$\alpha = \textit{communication latency} \approx 4\mu s$$

$$P = \textit{number of processes} \approx 10^3\text{–}10^7$$

- With MGS, need $k-1$ communications for $k = 2:n$

- With CGS, need one communication of length $k-1$ for $k = 2:n$

$$\text{MGS comm cost} \approx n^2\alpha \log_2 P$$

$$\text{CGS comm cost} \approx n\alpha \log_2 P$$

Two-Pass Classical Gram-Schmidt

- *Two-pass CGS* is accurate and potentially less expensive than MGS
- The idea is to *re-orthogonalize* the columns of $\mathbf{Q}$ with a second pass of the algorithm.
- If $\mathbf{Q}_1\mathbf{R}_1 = \mathbf{A}$ is the CGS-based QR factorization of the first pass, we generate a second factorization $\mathbf{Q}_2\mathbf{R}_2 = \mathbf{Q}_1$
- In this case, the starting point is the well-conditioned matrix of column vectors $\mathbf{Q}_1$ which span the column space of $\mathbf{A}$, as is also true for $\mathbf{Q}_2$.
- The full QR factorization of $\mathbf{A}$ is then

$$\mathbf{A} \;=\; \mathbf{Q}_1\mathbf{R}_1 \;=\; \mathbf{Q}_2 \underbrace{\mathbf{R}_2\mathbf{R}_1}_{\mathbf{R}}$$

- Note that $\text{cond}(\mathbf{R}_2) \approx 1$, so computation of $\mathbf{R} = \mathbf{R}_2\mathbf{R}_1$ does not introduce additional error
- It turns out that *only one* additional pass is needed.

Two-Pass Classical Gram-Schmidt

```
for k=1:n                %% MGS
    qk=A(:,k);
    for j=1:k-1;
        R(j,k) = Q(:,j)'*qk;
        qk     = qk - Q(:,j)*R(j,k);
    end;
    R(k,k)=norm(qk,2); Q(:,k)=qk/R(k,k);
end;
Qmgs = Q; Rmgs = R;

for ipass=1:2;
 for k=1:n               %% CGS
    ak=A(:,k); qk=ak;
    for j=1:k-1;
        R(j,k) = Q(:,j)'*ak;
        qk     = qk - Q(:,j)*R(j,k);
    end;
    R(k,k)=norm(qk,2); Q(:,k)=qk/R(k,k);
 end;
 if ipass==1; Q1=Q; R1=R; A=Q; end;
 if ipass==2; R2=R*R1; end;
end;
Qcgs = Q; Rcgs = R;

I=eye(n);

QQc = Qcgs'*Qcgs; Tc=QQc-I; tcn = norm(Tc)
QQm = Qmgs'*Qmgs; Tm=QQm-I; tmn = norm(Tm)

b=rand(m,1);
xc=Rcgs\(Qcgs'*b);
xm=Rmgs\(Qmgs'*b);
x = A \ b;
ec=norm(x-xc)/norm(x)
em=norm(x-xm)/norm(x)
```

```
Cond(A):            ans = 6.2467e+13

Ortho-test CGS      tcn = 1.6324e-03
CGS LLSQ vs. A\     ec  = 6.0648e-04

Ortho-test MGS      tmn = 8.0106e-11
MGS LLSQ vs. A\     em  = 3.4813e-05
```

```
2-Pass CGS
4.8899e-16
5.3060e-16
```

Classical & Modified GS: Notes

```
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;
```

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]
```

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

Using Orthogonal Transformations

- We've seen how we can use CGS/MGS QR factorizations to transform the LLSQ problem into triangular form.

- Here we take a different approach for Householder reflections and Givens rotations that offer alternative cost/benefits

- We seek numerically robust *transformations* that produce an easier problem without changing the solution.

- As with LU factorization, we'll look for a sequence of elementary transformation that yield an upper triangular form ($\mathbf{R}$), but instead of a companion lower triangular matrix, we have an orthogonal matrix ($\mathbf{Q}$) and the sequence of transformations will be *norm preserving*

- We use square *orthogonal* matrices $\mathbf{Q}$ satisfying $\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.

- These preserve the Euclidean norm

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

Orthogonal Transformations

- Note that if $\mathbf{Q}$ is a square orthogonal matrix, then $\mathbf{Q}^T$ is also an orthogonal matrix

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

- $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$

- $\|\mathbf{r}\|_2 = \|\mathbf{Q}\mathbf{r}\|_2 = \|\mathbf{Q}^T\mathbf{r}\|_2$

Using Orthogonal Transformations

- We've seen how we can use CGS/MGS QR factorizations to transform the LLSQ problem into triangular form.

- Here we take a different approach for Householder reflections and Givens rotations that offer alternative cost/benefits

- We seek numerically robust *transformations* that produce an easier problem without changing the solution.

- As with LU factorization, we'll look for a sequence of elementary transformation that yield an upper triangular form ($\mathbf{R}$), but instead of a companion lower triangular matrix, we have an orthogonal matrix ($\mathbf{Q}$) and the sequence of transformations will be *norm preserving*

- We use square *orthogonal* matrices $\mathbf{Q}$ satisfying $\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.

- These preserve the Euclidean norm

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

- Multiplying both sides of LLSQ by $\mathbf{Q}$ does not change its solution

Orthogonal Transformations

- Note that if $\mathbf{Q}$ is a square orthogonal matrix, then $\mathbf{Q}^T$ is also an orthogonal matrix

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

- $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$

- $\|\mathbf{r}\|_2 = \|\mathbf{Q}\mathbf{r}\|_2 = \|\mathbf{Q}^T\mathbf{r}\|_2$

Orthogonal Transformations

- For our LLSQ problems, we have been working with $m \times n$ matrix $\mathbf{A}$ and the corresponding (nonsquare) matrix "$\mathbf{Q}$" which we will (for now) denote as $\mathbf{Q}_1$.

- With $m > n$, we consider the partition of the orthogonal $m \times m$ matrix $\mathbf{Q}$,

$$\mathbf{Q} \;=\; [\mathbf{Q}_1 \, \mathbf{Q}_2]$$

  where $\mathbf{Q}_1$ is $m \times n$ and $\mathcal{R}(\mathbf{Q}_1) = \mathcal{R}(\mathbf{A})$ and $\mathcal{R}(\mathbf{Q}_2) \perp \mathcal{R}(\mathbf{Q}_1)$

- Consider application of $\mathbf{Q}^T$ to the residual, $\mathbf{r}$:

$$\mathbf{Q}^T\mathbf{r} \;=\; \mathbf{Q}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) \;=\; [\mathbf{Q}_1\mathbf{Q}_2]^T(\mathbf{b} - \mathbf{A}\mathbf{x}) \;=\; \underbrace{\begin{bmatrix} \mathbf{Q}_1^T(\mathbf{b} - \mathbf{A}\mathbf{x}) \\ \mathbf{Q}_2^T(\mathbf{b} - \mathbf{A}\mathbf{x}) \end{bmatrix}}_{\text{a } \textit{vector}}$$

$$= \begin{bmatrix} \mathbf{Q}_1^T\mathbf{b} - \mathbf{R}\mathbf{x} \\ \mathbf{Q}_2^T\mathbf{b} - \mathbf{0} \end{bmatrix} \;=\; \begin{bmatrix} \mathbf{0} \\ \mathbf{Q}_2^T\mathbf{b} \end{bmatrix}$$

Triangular LLSQ

- Consider a LLSQ problem with $\mathbf{R}$ being $n \times n$ upper triangular,

$$\begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix} \approx \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$$

- Residual is

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

- No control over $\|\mathbf{b}_2\|_2^2$ term, bust first term becomes zero if $\mathbf{x}$ satisfies $n \times n$ triangular system

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

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

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

Orthogonal Bases

- Consider *full* QR,

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

- $\mathbf{Q}_1\mathbf{R}$ is the *reduced* (or "economy") QR factorization of $\mathbf{A}$

- Columns of $\mathbf{Q}_1$ are orthonoromal basis for $\mathcal{R}(\mathbf{A})$, and columns of $\mathbf{Q}_2$ are orthonoromal basis for $\mathcal{R}^{\perp}(\mathbf{A})$

- $\mathbf{Q}_1\mathbf{Q}_1^T$ is orthogonal projector onto $\mathcal{R}(\mathbf{A})$

- Solution to LLSQ $\mathbf{A}\mathbf{x} \approx \mathbf{b}$ is sollution to square system

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

  as we've seen before.

- Generally, we will use the *reduced* QR as it is significantly less expensive than full QR

QR for Solving Least Squares

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

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

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

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

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

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

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

Computing QR via Householder or Givens

- In Gram-Schmidt, we successively transformed the columns of $\mathbf{A}$ to columns of $\mathbf{Q}_1$.

- Here, we consider methods that transform $\mathbf{A}$ into

$$\begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix}$$

- Similar to LU factorization, but using orthogonal (norm preserving) transformations instead of elementary elimination matrices

Method 2: Householder Transformations

Successive Householder Transformations

- Gram-Schmidt transforms A into Q.
- Householder QR transforms A into $\begin{bmatrix} R \\ O \end{bmatrix}$.
- To do so, it applies a sequence of orthogonal transformations (H_k) known as *Householder transformations* (or *reflections*).

$$\underset{A}{\begin{bmatrix} \tilde{x} & \tilde{x} & \tilde{x} \\ \tilde{x} & \tilde{x} & \tilde{x} \\ \tilde{x} & \tilde{x} & \tilde{x} \\ \tilde{x} & \tilde{x} & \tilde{x} \\ \tilde{x} & \tilde{x} & \tilde{x} \end{bmatrix}} \xrightarrow{H_1} \underset{H_1 A}{\begin{bmatrix} \tilde{x} & \tilde{x} & \tilde{x} \\ 0 & \tilde{x} & \tilde{x} \\ 0 & \tilde{x} & \tilde{x} \\ 0 & \tilde{x} & \tilde{x} \\ 0 & \tilde{x} & \tilde{x} \end{bmatrix}} \xrightarrow{H_2} \underset{H_2 H_1 A}{\begin{bmatrix} \tilde{x} & \tilde{x} & \tilde{x} \\ & \tilde{x} & \tilde{x} \\ & 0 & \tilde{x} \\ & 0 & \tilde{x} \\ & 0 & \tilde{x} \end{bmatrix}} \xrightarrow{H_3} \underset{H_3 H_2 H_1 A}{\begin{bmatrix} \tilde{x} & \tilde{x} & \tilde{x} \\ & \tilde{x} & \tilde{x} \\ & & \tilde{x} \\ & & 0 \\ & & 0 \end{bmatrix}}$$

Householder Transformations/Orthogonal Projectors

- *Householder transformation* is the $m \times m$ orthogonal matrix

$$\mathbf{H} \;=\; \mathbf{I} \;-\; 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}$$

  for nonzero vector $\mathbf{v}$

- Recall that, if $\mathbf{q}$ is an m-vector with unit 2-norm, then we have two projectors

$$\mathbf{v} \;=\; P_{\mathbf{q}}\mathbf{u} \;=\; (\mathbf{q}\mathbf{q}^T)\mathbf{u}$$

$$\mathbf{w} \;=\; P_{\mathbf{q}}^{\perp}\,\mathbf{u} \;=\; [\mathbf{I} - (\mathbf{q}\mathbf{q}^T)]\mathbf{u} \;=\; \mathbf{u} - \mathbf{q}(\mathbf{q}^T\mathbf{u})$$

- The Householder transformation is *almost* a projection onto $\mathcal{R}^{\perp}(\mathbf{v})$

- However, because of the "2" it projects *past* $\mathcal{R}^{\perp}(\mathbf{v})$, or *reflects* about $\mathcal{R}^{\perp}(\mathbf{v})$

Householder Transformations

- We construct Householder *reflector* $\mathbf{H}$ through careful selection of $\mathbf{v}$

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

- $\mathbf{H}$ is orthogonal and symmetric, $\mathbf{H} = \mathbf{H}^T = \mathbf{H}^{-1}$

- Given a vector $\mathbf{a}$, want to chose $\mathbf{v}$ such that

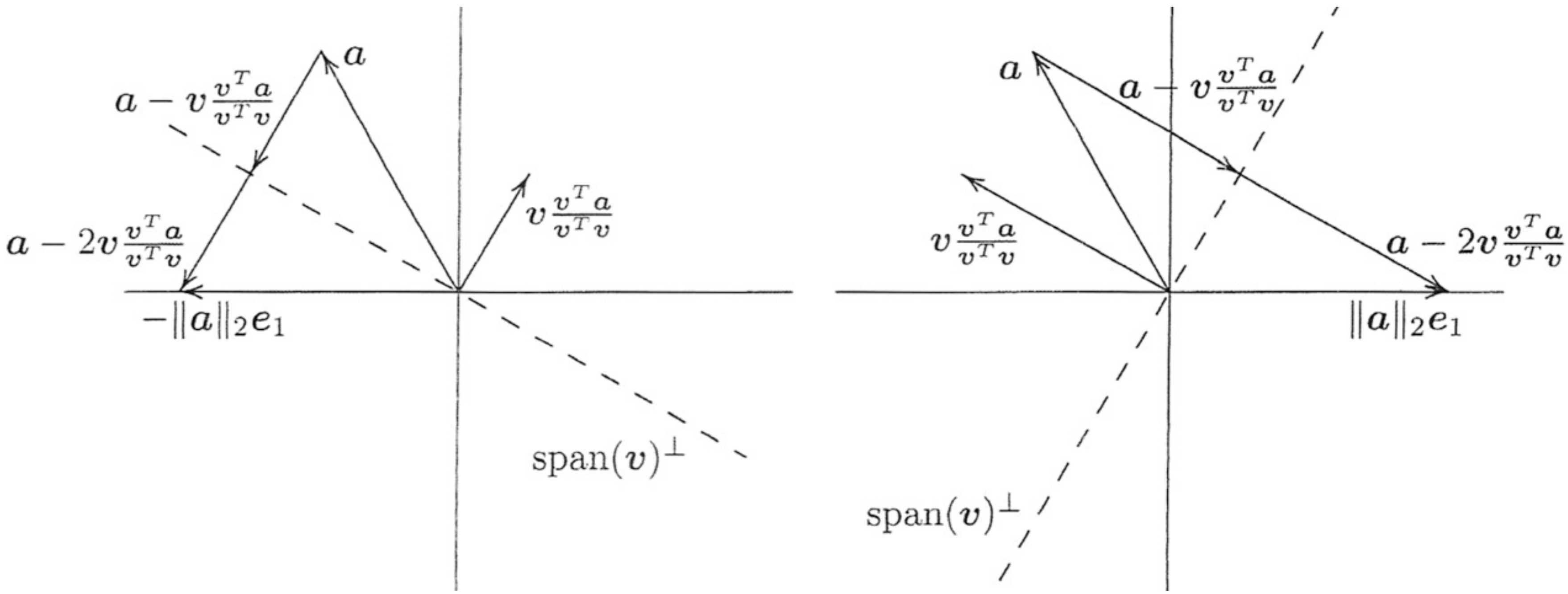
$$\mathbf{H}\mathbf{a} = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha\mathbf{e}_1$$

- Substituing in formula for $\mathbf{H}$ we can take

$$\mathbf{v} = \mathbf{a} - \alpha\mathbf{e}_1$$

and $\alpha = \pm\|\mathbf{a}\|_2$, with ***sign chosen to avoid cancelation***

Householder Reflection



- Recall, $I - \mathbf{v}(\mathbf{v}^T\mathbf{v})^{-1}\mathbf{v}^T$ is a projector onto $R^\perp(\mathbf{v})$.
- Therefore, $I - 2\mathbf{v}(\mathbf{v}^T\mathbf{v})^{-1}\mathbf{v}^T$ will reflect the transformed vector past $R^\perp(\mathbf{v})$.
- Notice Householder transformation subtracts a *multiple of* $\mathbf{v}$ *from* $\mathbf{a}$.
- With Householder, choose $\mathbf{v}$ such that the reflected vector has all entries below the kth one set to zero.
- Also, choose $\mathbf{v}$ to avoid cancellation in kth component.

Householder Derivation

$$H\mathbf{a} \;=\; \mathbf{a} - 2\frac{\mathbf{v}^T\mathbf{a}}{\mathbf{v}^T\mathbf{v}}\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\mathbf{v} \;=\; \mathbf{a} - \alpha\mathbf{e}_1 \;\longleftarrow\; \text{Choose } \alpha \text{ to get desired cancellation.}$$

$$\mathbf{v}^T\mathbf{a} \;=\; \mathbf{a}^T\mathbf{a} - \alpha a_1, \qquad \mathbf{v}^T\mathbf{v} = \mathbf{a}^T\mathbf{a} - 2\alpha a_1 + \alpha^2$$

$$\begin{aligned} H\mathbf{a} \;&=\; \mathbf{a} - 2\frac{\left(\mathbf{a}^T\mathbf{a} - \alpha a_1\right)}{\mathbf{a}^T\mathbf{a} - 2\alpha a_1 + \alpha^2}\left(\mathbf{a} - \alpha\mathbf{e}_1\right) \\ &=\; \mathbf{a} - 2\frac{||\mathbf{a}||^2 \pm ||\mathbf{a}||a_1}{2||\mathbf{a}||^2 \pm 2||\mathbf{a}||a_1}\left(\mathbf{a} - \alpha\mathbf{e}_1\right) \\ &=\; \mathbf{a} - \left(\mathbf{a} - \alpha\mathbf{e}_1\right) \;=\; \alpha\mathbf{e}_1. \end{aligned}$$

Choose $\quad \alpha = -\text{sign}(a_1)||\mathbf{a}|| = -\left(\dfrac{a_1}{|a_1|}\right)||\mathbf{a}||.$

Example: Householder Reflection

- Consider $\mathbf{a} = [2 \;\; 1 \;\; 2]^T$.

- 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, take $\alpha = -\|\mathbf{a}\|_2$ to avoid cancellation

$$\mathbf{v} \;=\; \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} \;=\; \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$$

- Confirm that transformation works:

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

Householder QR Factorization

- To compute QR factorization from $\mathbf{A}$, use Householder reflectors to annihilate subdiagonal entries of each successive column

- Each Householder refector ($\mathbf{H}_k$) is applied to entire matrix, but does not affect prior columns, so zeros are preserved

- Applying $\mathbf{H}$ to arbitrary vector $\mathbf{u}$,

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

  which is $O(m)$ work; much cheaper than general matrix-vector product.

- Requires only vector $\mathbf{v}$, not full matrix $\mathbf{H}$

Householder QR Factorization, continued

- Process produces factorization

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

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

- If $\mathbf{Q} = \mathbf{H}_1 \cdots \mathbf{H}_n$ then $\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix} \iff \begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix} = \mathbf{Q}^T \mathbf{A}$

- To preserve solution of LLSQ, right hand side $\mathbf{b}$ is transformed by same sequence

- Then solve triangular LLSQ problem $\begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix} \mathbf{x} \approx \mathbf{c} := \mathbf{Q}^T \mathbf{b}$

Householder QR Factorization, continued

- For solving LLSQ, product $\mathbf{Q}$ of $\mathbf{H}_k$ is not needed

- $\mathbf{R}$ can be stored in upper-triangular part of $\mathbf{A}$

- Householder vectors $\mathbf{v}$ can be stored in now-zero lower portion of $\mathbf{A}$ (almost)

- Householder transformations most easily applied in that form anyway

kth Householder Transformation (Reflection)

kth column

$\downarrow$

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

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

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?

Note: Householder Procedure

$$H_3\,H_2\,H_1 A \;=\; \left(\begin{array}{c} R \\ O \end{array}\right), \qquad A \;=\; Q \left(\begin{array}{c} R \\ O \end{array}\right).$$

$$H_3\,H_2\,H_1 A \;=\; Q^{-1}\,Q \left(\begin{array}{c} R \\ O \end{array}\right) \;=\; Q^T\,Q \left(\begin{array}{c} R \\ O \end{array}\right) \;=\; Q^T\,A.$$

$$Q^T \;=\; H_3\,H_2\,H_1$$

$$Q \;=\; H_1^T\,H_2^T\,H_3^T \;=\; H_1\,H_2\,H_3.$$

- 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:

$$H_k\,\mathbf{u} = \mathbf{u} - 2\left(\frac{\mathbf{v}_k^T\mathbf{u}}{\mathbf{v}_k^T\mathbf{v}_k}\right)\mathbf{v}_k.$$

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

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

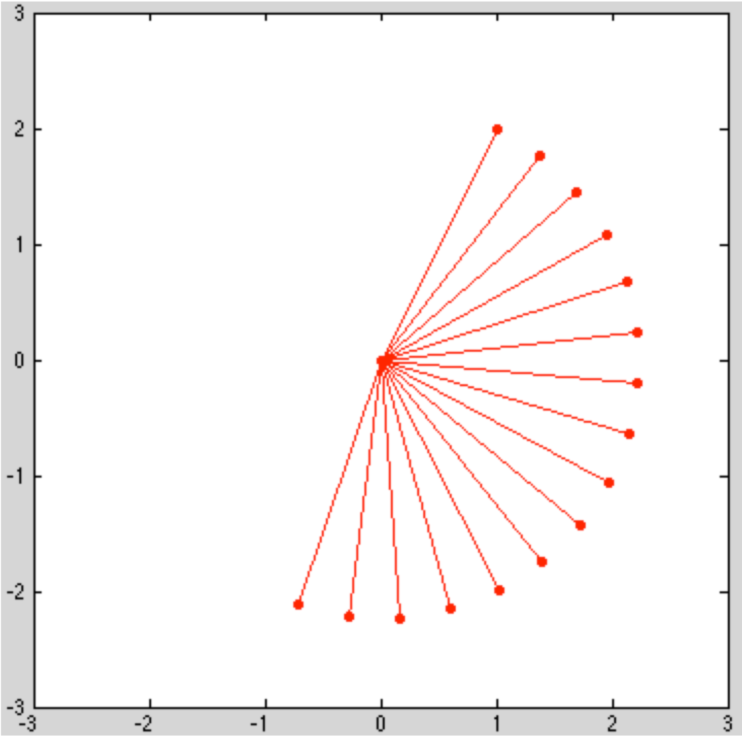
- Now solve upper triangular system $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{c}_1$ by back-substitution to obtain $\boldsymbol{x} = \begin{bmatrix} 0.086 & 0.400 & 1.429 \end{bmatrix}^T$

Method 3: Givens Rotations

Stopped Here

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;
```



demo7/rotate.m

Givens Rotations

- ***Givens rotations*** introduce zeros one at a time.

- Given vector $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$, choose scalars c and s so that

Orthogonal matrix, G →

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

$$\mathbf{G} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$$

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

- Rearranging preceding equation, can solve for c and s

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

- Gaussian elimination leads to 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

- Back-substitution yields sine and cosine

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

- Because $c^2 + s^2 = 1 \iff \alpha = \sqrt{a_1^2 + a_2^2}$, we have

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

Example: Givens Rotations

- Let $\mathbf{a} = [4 \;\; 3]^T$

- To annihilate the second entry, 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 produced by orthogonal matrix

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

- Check by applying $\mathbf{G}$ to $\mathbf{a}$

$$\mathbf{Ga} = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

Givens QR Factorization

- To annihilate selected component of $\mathbf{a}$, rotate target component with another

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

- Using a sequence of Givens rotations, systematically annihilate successive entries to reduce matrix to upper triangular form

- Each rotation is orthogonal, so their product is orthogonal, producing QR factorization

Successive Givens Rotations

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

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

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

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

- **How many Givens rotations (total) are required for the $m \times n$ case?**
- **How does $\dots G_3\,G_2\,G_1$ relate to Q or Q_1?**
- **What is Q in this case?**

Givens QR Factorization

- Straightforward implementation of Givens QR requires about 50% more work than Householder and also requires more storage because each rotation requires two numbers, c and s, to define it.

- These disadvantages can be overcome with more sophisticated implementation

- Givens offers an advantage, however, when many of the matrix entries are already zero because those annihilations can then be skipped

Givens QR Factorization

- A particularly attractive use of Givens QR is when $\mathbf{A}$ is upper Hessenberg $\iff$ $\mathbf{A}$ is upper triangular with one additional nonzero diagonal below the main one, i.e., $a_{ij} = 0$ 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 $O(n^2)$ vs. $O(n^3)$ for Householder

- Upper Hessenberg matrices when computing eigenvalues and in Krylov subspace methods such as GMRES for solving sparse linear systems

Rank Deficiency

- If $\text{rank}(\mathbf{A}) < n$, then QR factorization still exists, but yields singular upper triangular factor, $\mathbf{R}$, and multiple vectors $\mathbf{x}$ give minimum residual norm

- Common practice selects minimum residual solution $\mathbf{x}$ having smallest norm

- Can be computed by QR factorization with column pivoting or by SVD

- Matrix rank is not clear cut in practice so relative tolerance is used to determine rank

Example: Near Rank Deficiency

- Consider 3×2 matrix

$$\mathbf{A} = \begin{bmatrix} .300 & .100 \\ .100 & .033 \\ .200 & .066 \end{bmatrix}$$

- QR factorization gives

$$\mathbf{R} = \begin{bmatrix} -.3742 & -.1243 \\ 0 & .0006 \end{bmatrix},$$

  which is close to singular

- If $\mathbf{R}$ is used to solve LLSQ problem result will be sensitive to perturbations in right-hand side

- For practical purposes, rank($\mathbf{A}$)=1 rather than 2 because columns of $\mathbf{A}$ are nearly parallel

QR with Column Pivoting

- At each stage k, choose to reduce column having maximum 2-norm for (reduced) submatrix $\mathbf{A}(k:m, k:n)$

- If $\text{rank}(\mathbf{A}) = k < n$, then after k steps norms of remaining unreduced columns will be negligible below row k

- Rank is determined when maximum norm of remaining unreduced columns falls below chosen tolerance

- Orthogonal factorization will be of form

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

  where nonsingular $\mathbf{R}$ is $k \times k$ upper triangular and $\mathbf{P}$ performs column interchanges

Singular Value Decomposition

- Singular value decomposition (SVD) of $m \times n$ matrix $\mathbf{A}$ is

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

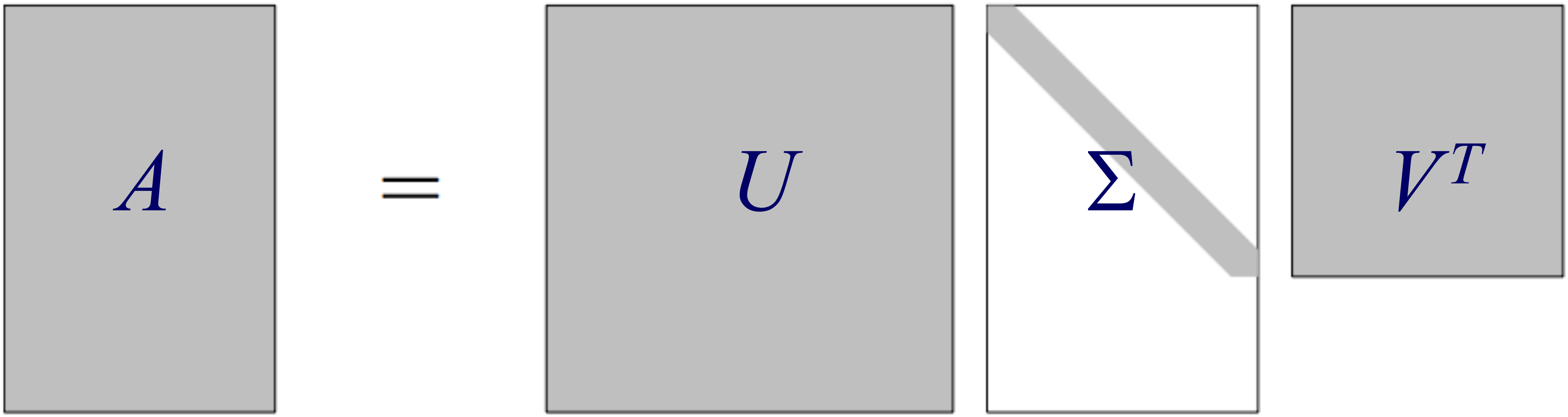
  where $\mathbf{U}$ is $m \times m$ orthogonal matrix, $\mathbf{V}$ is $n \times n$ orthogonal matrix, $\boldsymbol{\Sigma}$ is $m \times n$ diagonal matrix with

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

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

- Columns $\mathbf{u}_j$ of $\mathbf{U}$ and $\mathbf{v}_j$ of $\mathbf{V}$ are respective 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, $\sigma_i \geq 0$.
- V is $n \times n$, orthogonal.

Example: SVD

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

$$\mathbf{U} = \begin{bmatrix} -0.4036 & 0.7329 & 0.5110 & 0.1972 \\ -0.4647 & 0.2898 & -0.8283 & 0.1180 \\ -0.5259 & -0.1532 & 0.1236 & -0.8275 \\ -0.5870 & -0.5962 & 0.1937 & 0.5123 \end{bmatrix}$$

$$\mathbf{\Sigma} = \begin{bmatrix} 25.437 & 0 & 0 \\ 0 & 1.7226 & 0 \\ 0 & 0 & 1.42e-15 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{V}^T = \begin{bmatrix} -0.2067 & -0.5183 & -0.8298 \\ -0.8892 & -0.2544 & 0.3804 \\ 0.4082 & -0.8165 & 0.4082 \end{bmatrix}$$

Applications of SVD

- *Minimum norm solultion* to $\mathbf{A}\mathbf{x} \approx \mathbf{b}$ is

$$\mathbf{x} \;=\; \sum_{\sigma_j \neq 0} \frac{1}{\sigma_j}(\mathbf{u}_j^T \mathbf{b})\mathbf{v}_j$$

  For ill-conditioned or rank-deficient cases, replace $1/\sigma_j$ with 0 to stabilize solution. (Keeps $\mathbf{y} = \mathbf{A}\mathbf{x}$ in the "true" space spanned by $[\mathbf{a}_1\ \mathbf{a}_2\ \ldots\ \mathbf{a}_n]$)

- *Euclidian matrix norm*: $\|\mathbf{A}\|_2 = \sigma_{\max}$

- *Euclidian condition number*: $\mathrm{cond}(\mathbf{A}) = \frac{\sigma_{\max}}{\sigma_{\min}}$

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

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

$$
\begin{aligned}
A\underline{x} &\approx \underline{b} \\
U\Sigma V^T &\approx \underline{b} \\
U^T U\Sigma V^T &\approx U^T\underline{b} \\
\Sigma V^T &\approx U^T\underline{b} \\
\left[\begin{array}{c} \tilde{R} \\ O \end{array}\right]\underline{x} &\approx \left(\begin{array}{c} \underline{c}_1 \\ \underline{c}_2 \end{array}\right) \\
\tilde{R}\underline{x} &= \underline{c}_1
\end{aligned}
$$

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

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 $\mathbf{A}$ is

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

- Pseudoinverse always exists whether or not matrix is square or has full rank

- If $\mathbf{A}$ is square and nonsingular then $\mathbf{A}^+ = \mathbf{A}^{-1}$

- In all cases, minimum-norm solution to $\mathbf{A}\mathbf{x} \approx \mathbf{b}$ is $\mathbf{x} = \mathbf{A}^+\mathbf{b}$

Orthogonal Bases

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

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

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

- Columns of $\mathbf{V}$ corresponding to zero singular values form orthonormal basis for nullspace of $\mathbf{A}$,

$$\mathcal{R}(\mathbf{V}) = \mathcal{N}(\mathbf{A}) = \{\mathbf{v} \neq \mathbf{0} \,:\, \mathbf{v} \in \mathcal{N}(\mathbf{A}) \iff \mathbf{A}\mathbf{v} = \mathbf{0}\}$$

- Remaining columns of $\mathbf{V}$ form orthonormal basis for orthogonal complement $\mathcal{N}^\perp(\mathbf{A})$

Low-Rank Approximations to Matrices or Data

- With $\mathbf{E}_i := \mathbf{u}_i \mathbf{v}_i^T$, the SVD of $\mathbf{A}$ can be expanded term by term as

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

- Each $m \times n$ matrix $\mathbf{E}_i$ is rank 1 and can be stored using only $m+n$ storage

- Product $\mathbf{E}_i \mathbf{x}$ can be evaluated using only $m+n$ multiplications and $m+n$ additions

- Condensed approximation to $\mathbf{A}$ is obtained by omitting from summation terms corresponding to small singular values

- If singular values are ordered

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$$

  then using the first k terms will give *best rank k approximation* to $\mathbf{A}$ (Eckart-Young-Mirsky Theorem)

- Storage and work costs are $O(k(m+n)) \ll O(mn)$ if k is relatively small

- Approximation is useful in data compression, image processing, information retrieval, cryptography, etc.

Low-Rank Approximations of A

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

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

- A *rank k* approximation to $\mathbf{A}$ is given by

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

- $\mathbf{A}_k$ is the best approximation to $\mathbf{A}$ in the Frobenius norm,

$$||\mathbf{A}||_F := \sqrt{\sum_{ij} a_{ij}^2}$$

- Note that in this context, its more common to think of $\mathbf{A}$ as the *data* which is to be approximated and $\mathbf{A}_k$ as the model.

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.

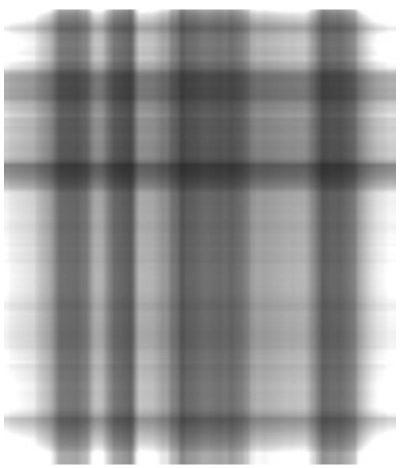
$$A \approx A_k := \sum_{j=1}^{k} \underline{u}_j \sigma_j \underline{v}_j^T$$

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.

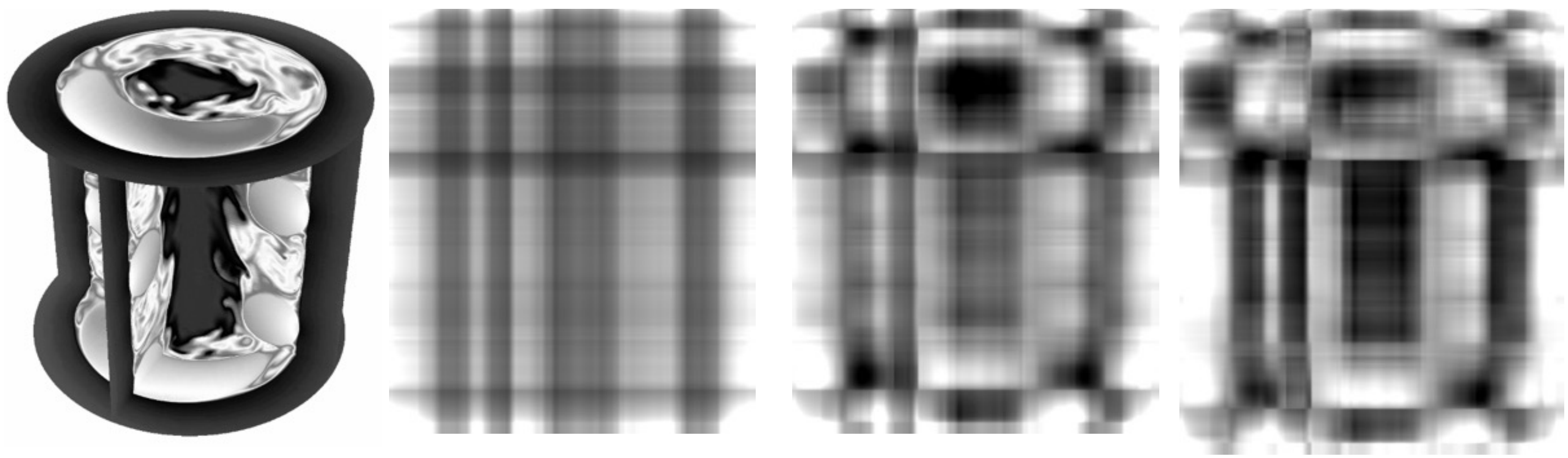


k=1

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

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)

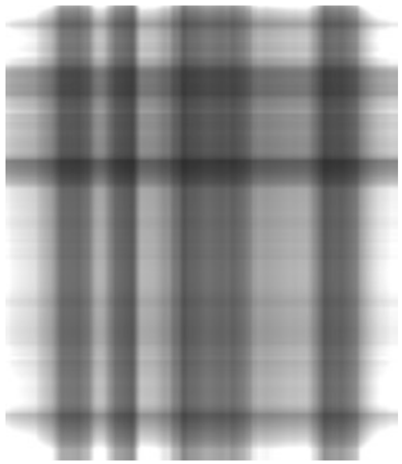
Matlab code

```
[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;
```

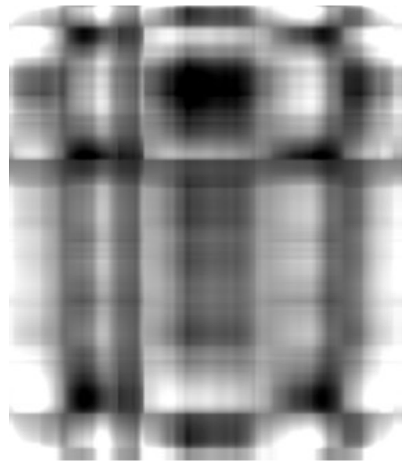
Image Compression

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

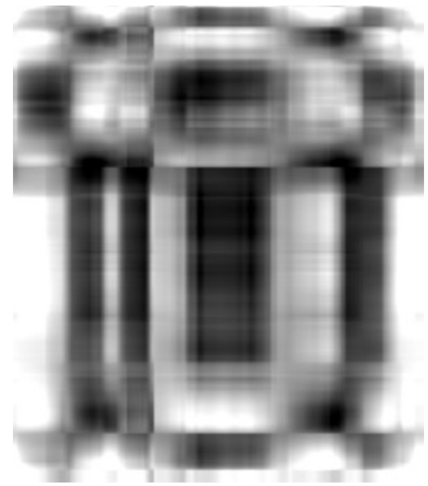
k=1



k=2



k=3



k=10

k=20

k=50

(m=536, n=462)

Low-Rank Approximations to Solutions of A$\underline{x}$ = $\underline{b}$

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

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

- 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.

Example: Total Least Squares

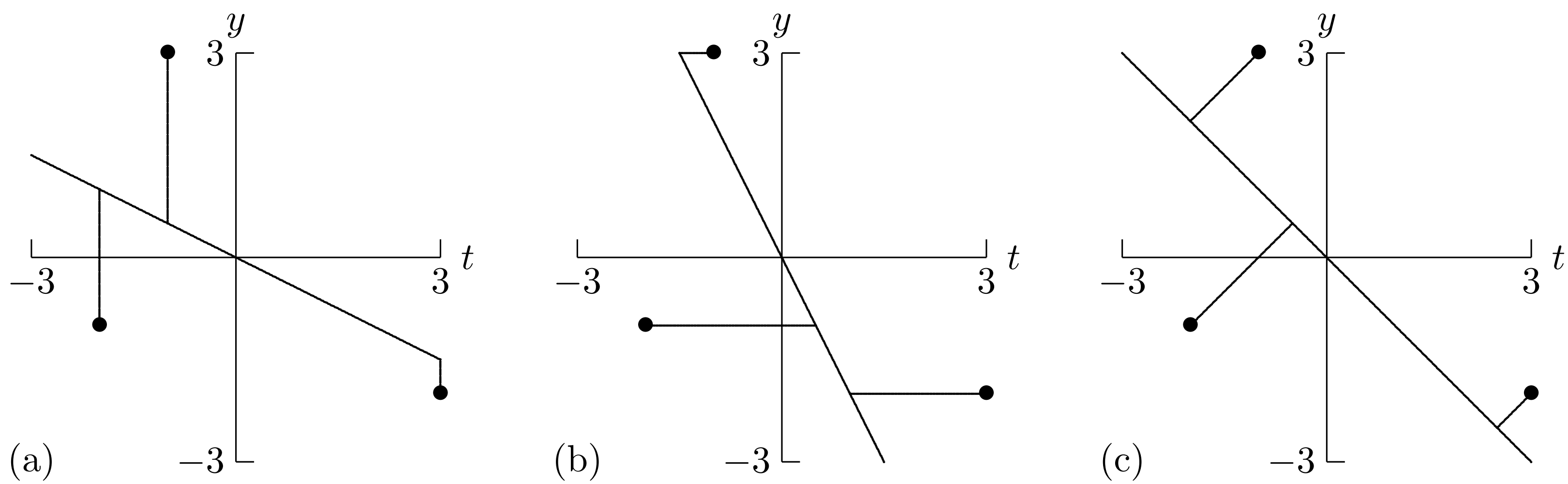


Figure 3.5: Ordinary and total least squares fits of straight line to given data.

Projecting Noisy Data in $\mathbb{R}^3$ onto 2D Plane

- Given rank-3 matrix

$$\mathbf{X} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_m & y_m & z_m \end{bmatrix}$$

- Find rank-2 matrix $\mathbf{X}_2 \approx \mathbf{X}$ that minimizes difference in Frobenius norm

- Compute $\mathbf{U\Sigma V}^T = \mathbf{X}$ and set $\mathbf{\Sigma}_2 = \mathbf{\Sigma}$, with, however, $\sigma_3 = 0$.

- Set $\mathbf{X}_2 = \mathbf{U\Sigma}_2\mathbf{V}^T$

demo7/svd5.m

```
hdr; hold off;

X=[5.0000e-01  -6.6164e-02   4.2484e-02 ;
   5.5548e-01   3.2280e-01   6.2050e-01 ;
  -2.1973e-01   4.9042e-01   8.2918e-01 ;
  -6.4594e-01   1.7391e-01   2.5801e-01 ;
  -2.5503e-01  -1.3753e-01  -5.7648e-02 ;
  -2.7895e-03  -2.5073e-02   1.5746e-03 ;
   6.8013e-02  -7.3863e-02  -5.3926e-02];
```

Original data

```
[U,S,V]=svd(X,0);

S2 = S; S2(3,3)=0;

X2 = U*S2*V';
```

Projected data

```
hold off;
xp=X(:,1); yp=X(:,2); zp=X(:,3);
xp=[xp; xp(1)]; yp=[yp; yp(1)]; zp=[zp; zp(1)];
plot3(xp,yp,zp,'bo-',lw,2)
title('Original Data',fs,24);
xlabel('X',fs,24); ylabel('Y',fs,24); zlabel('Z',fs,24);
axis equal;
pause(1); pause;

hold on;
xp=X2(:,1); yp=X2(:,2); zp=X2(:,3);
xp=[xp; xp(1)]; yp=[yp; yp(1)]; zp=[zp; zp(1)];
plot3(xp,yp,zp,'ro-',lw,2)
title('Original and Projected Data',fs,24);
xlabel('X',fs,24); ylabel('Y',fs,24); zlabel('Z',fs,24);
axis equal;
```

Comparison of Methods for LLSQ

- Forming normal equations matrix $\mathbf{A}^T\mathbf{A}$ requires $\approx n^2 m$ ops and solving resulting linear system $\approx n^3/3$ ops.

- Solving LLSQ using Householder QR requires $\approx 2n^2(m - n/3)$ ops

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

- If $m \gg n$, Householder QR requires about $2\times$ the number of ops as normal equations (but is *more robust*)

- Cost of SVD is $\approx C(mn^2 + n^3)$, with $C = 4$ to 10, depending on algorithm used

Comparison of Methods for LLSQ

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

- Required Cholesky factorization expected to break down if $\text{cond}(\mathbf{A}) \gtrsim 1/\sqrt{\epsilon_M}$

- Householder method produces solution with relative error proportional to

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

  which is best possible because of inherent sensitivity of LLSQ problem

- Householder method expected to break down (in back-substitution phase, $\mathbf{Rx} = \mathbf{c}_1$) only if $\text{cond}(\mathbf{A}) \gtrsim 1/\epsilon_M$

Comparison of Methods for LLSQ

- Householder is more accurate and broadly applicable than normal equations

- These advantages may not be worth the additional cost when problem is sufficiently well conditioned that normal equations are OK

- For rank-deficient problems, Householder with column pivoting can produce useful solution

- SVD is even more robust, but more expensive.