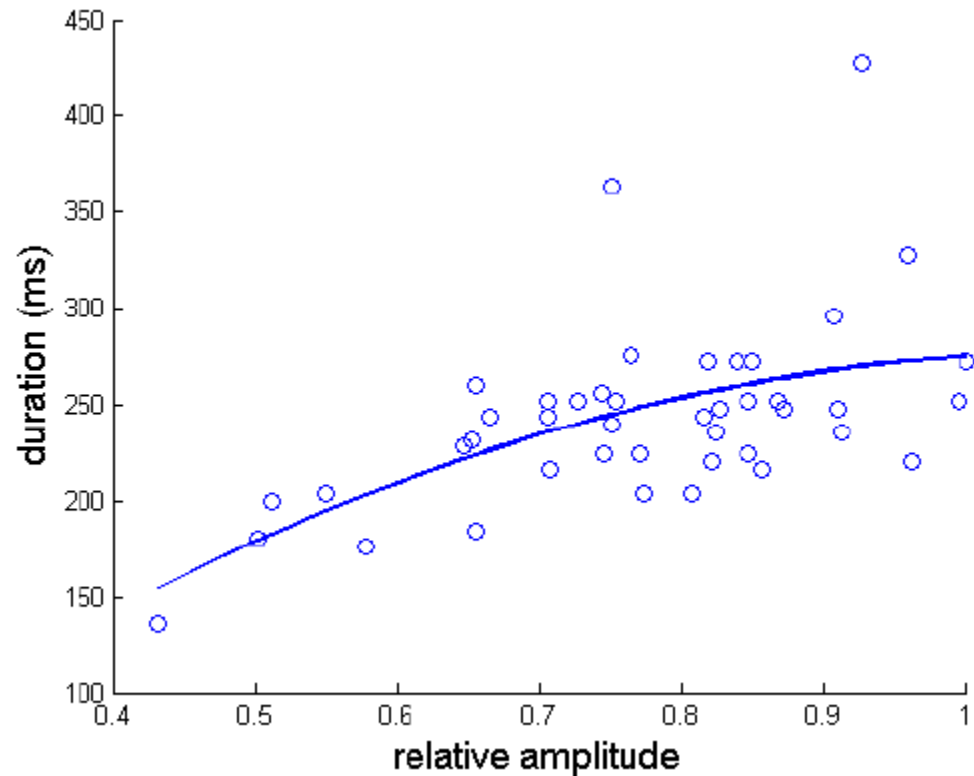


Chapter 3

Overdetermined linear systems



Overdetermined linear systems

- In many situations, we want to put a simple curve through data
- We discussed interpolating data previously, but there would be relatively little data for this.
- For interpolating a quadratic curve, with three constants to find, we would need (x_j, y_j) for $j = 1, 2, 3$
- We would then have the right number of equations (3) to find the coefficients for $f(x) = a_1x^2 + a_2x + a_3$ to pass through the data
- In that case, we would have to solve $y_j = f(x_j)$ for $j = 1, 2, 3$ for the a_j

Overdetermined linear systems

- The system is of the following form, with $n = m = 3$:

$$\begin{bmatrix} t_1^{n-1} & t_1^{n-2} & \dots & t_1 & 1 \\ t_2^{n-1} & t_2^{n-2} & \dots & t_2 & 1 \\ t_3^{n-1} & t_3^{n-2} & \dots & t_3 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ t_n^{n-1} & t_n^{n-2} & \dots & t_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

- But, there are many cases where there is too much data to interpolate because the oscillation of polynomials would be a poor model

Overdetermined linear systems

- When there is more data, we have $n < m$, and sometimes $n \ll m$
- The system is still of the form:

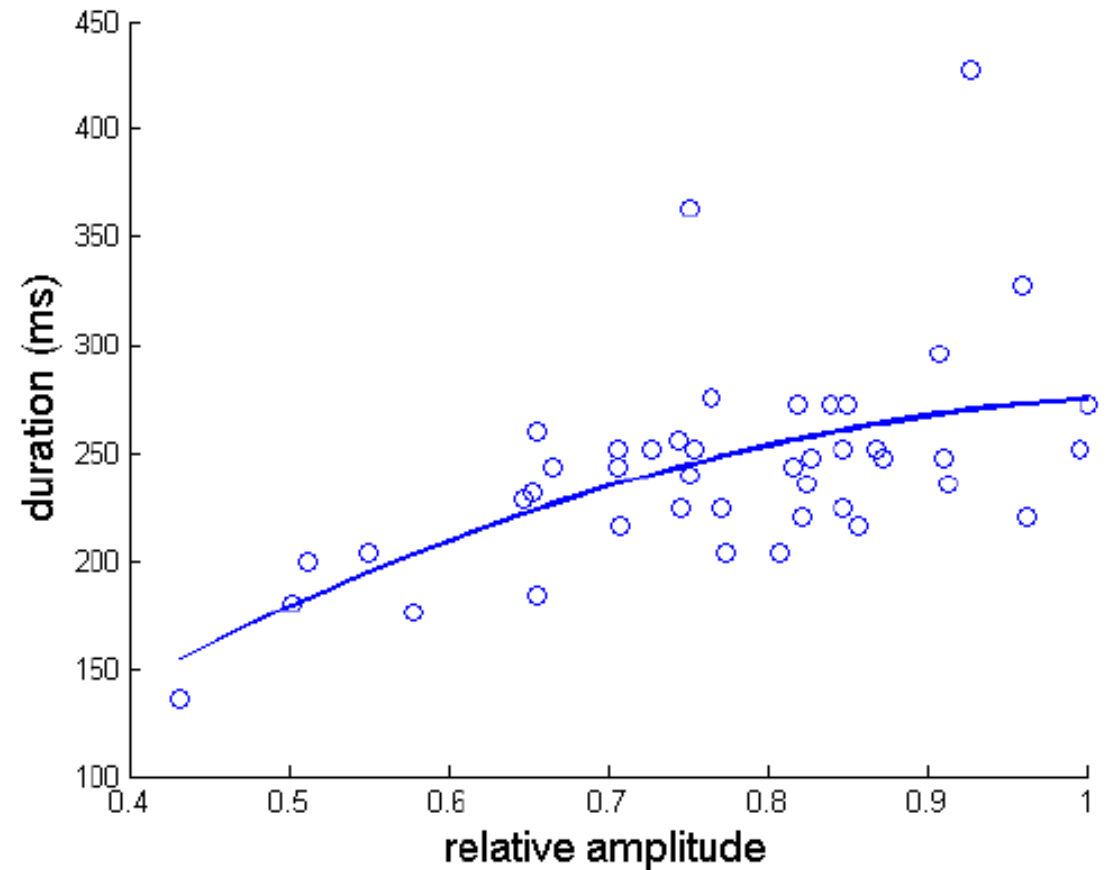
$$\begin{bmatrix} t_1^{n-1} & t_1^{n-2} & \dots & t_1 & 1 \\ t_2^{n-1} & t_2^{n-2} & \dots & t_2 & 1 \\ t_3^{n-1} & t_3^{n-2} & \dots & t_3 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ t_n^{n-1} & t_n^{n-2} & \dots & t_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

- How to solve? Find the a_j to minimize the distance from the data to the curve:

$$\min_a \sum_{i=1}^n [y_i - (a_1 x_i^2 + a_2 x_i + a_3)]^2$$

Overdetermined linear systems

- Example: eye blink durations (y_j) and amplitudes (x_j) with $m=43$ blinks
- We want to put a quadratic through the data
- Solving the equation and plotting the quadratic and data gives plot at right
- One could rightly wonder if a different function could fit the data better here



Overdetermined linear systems

- How to solve these problems?

$$\min_a \sum_{i=1}^n [y_i - (a_1 x_i^2 + a_2 x_i + a_3)]^2$$

- For this minimization problem, we could compute the partial derivative with respect to each a_j and set the derivative equal to zero.
- Solving those equations would produce the coefficients and thus the function we seek.
- Linear least squares are often approached this way.
- We want to focus on a more linear algebra oriented approach

Overdetermined systems

- More generally, we can seek linear combos of functions for fitting

$$f(t) = c_1 f_1(t) + \cdots + c_n f_n(t).$$

- For this minimization problem, we consider the residual

$$R(c_1, \dots, c_n) = \sum_{i=1}^m [y_i - f(t_i)]^2.$$

- From linear algebra, $R = \mathbf{r}^T \mathbf{r}$, and we can write the following...

Overdetermined systems

- More generally, we can seek linear combos of functions for fitting

$$\mathbf{r} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix} - \begin{bmatrix} f_1(t_1) & f_2(t_1) & \cdots & f_n(t_1) \\ f_1(t_2) & f_2(t_2) & \cdots & f_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(t_{m-1}) & f_2(t_{m-1}) & \cdots & f_n(t_{m-1}) \\ f_1(t_m) & f_2(t_m) & \cdots & f_n(t_m) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

- Now $\mathbf{r}^T \mathbf{r} = \|\mathbf{r}\|_2^2$ so that, for $A = \mathbb{R}^{m \times n}$, we solve

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{b} - A\mathbf{x}\|_2^2$$

- The unknowns \mathbf{x} are the coefficients we seek
- Let's see how to do this...

Overdetermined systems: The Normal Equations

- Let's take a linear algebraic view of the minimization problem
- To see how to solve the minimization problem consider Theorem 3.1:
If x satisfies $A^T(Ax - b) = 0$, then x solves the least squares problem $\min_x \|b - Ax\|_2$

$$\begin{aligned}\|A(x + y) - b\|_2^2 &= [(Ax - b) + (Ay)]^T [(Ax - b) + (Ay)] \\ &= (Ax - b)^T (Ax - b) + 2(Ay)^T (Ax - b) + (Ay)^T (Ay) \\ &= \|Ax - b\|_2^2 + 2y^T A^T (Ax - b) + \|Ay\|_2^2 \\ &= \|Ax - b\|_2^2 + \|Ay\|_2^2 \geq \|Ax - b\|_2^2.\end{aligned}$$

- To make this work, we needed $A^T(Ax - b) = 0$, or $A^T Ax = A^T b$
- These are the “normal equations”: solve them for x

The Normal Equations

- The normal equations $\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}$ are a square linear system for \mathbf{x}
- The theoretical solution is $\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$
- We could write $\mathbf{x} = \mathbf{A}^+ \mathbf{b}$, where $\mathbf{A}^+ = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$
- \mathbf{A}^+ is the “pseudoinverse” of \mathbf{A}
- In MATLAB, it may be obtained from `pinv(A)`
- Some properties of $\mathbf{A}^T \mathbf{A}$ are important...

The Normal Equations

- Some properties of $\mathbf{A}^T \mathbf{A}$ are important...

Theorem 3.2. *For any real $m \times n$ matrix \mathbf{A} with $m \geq n$, the following are true:*

1. $\mathbf{A}^T \mathbf{A}$ is symmetric.
 2. $\mathbf{A}^T \mathbf{A}$ is singular if and only if the columns of \mathbf{A} are linearly dependent. (Equivalently, we say that \mathbf{A} is rank deficient, which means that the rank of \mathbf{A} is less than n .)
 3. If $\mathbf{A}^T \mathbf{A}$ is nonsingular, then it is positive definite.
- For 2, if $\mathbf{A}^T \mathbf{A}$ is singular, then $\mathbf{A}^T \mathbf{A} \mathbf{z} = \mathbf{0}$ for nonzero \mathbf{z} ; we need to show that this happens only if \mathbf{A} is singular. If we premultiply by \mathbf{z}^T , then $\mathbf{0} = \mathbf{z}^T \mathbf{A}^T \mathbf{A} \mathbf{z} = \|\mathbf{A} \mathbf{z}\|_2^2$. This can only happen if $\mathbf{A} \mathbf{z} = \mathbf{0}$; if that happens, \mathbf{A} is singular.

The Normal Equations

- To use the normal equations to solve the least squares problem, do the following:
 1. Compute $N = A^T A$
 2. Compute $z = A^T b$
 3. Solve the $n \times n$ linear system $Nx = z$ for x
- The computation is easy, but the conditioning is poor.
- In the homework, you are asked to prove that $\kappa(A^T A) = \kappa(A)^2$, so that the magnification of the residual may be large.

The Normal Equations

- Example: do a cubic fit to some data as follows

```
>> t = [1; 2; 3; 4; 5; 6];  
>> y = [1.5; 3.9; 6; 13; 27; 30];  
>> A = [t.^3 t.^2 t ones(size(t))];  
>> N = A'*A
```

N =

67171	12201	2275	441
12201	2275	441	91
2275	441	91	21
441	91	21	6

- The columns of A are powers of t (a Vandermonde matrix).

The Normal Equations

- Example: do a cubic fit to some data
- The condition number for N is fairly large for only a 4×4 matrix
- Solving manually for the coefficients \mathbf{a} is shown
- The residual for this approach is about $1e-11$
- Using $\mathbf{a} = \text{pinv}(\mathbf{A}) * \mathbf{y}$, one gets a residual of about $3e-11$, very similar
- Our error could be as bad as $1e6$ larger

```
>> cond(N)
ans =
    2.1515e+06
>> cond(A)
ans =
    1.4668e+03
>> a = N \ (A' * y)
a =
   -0.4370
    5.4925
  -13.9276
   11.1333
>> norm(A' * y - N * a)
ans =
    9.1803e-12
```

Normal Equations: better

- We can do a little bit better by using Cholesky factorization because $A^T A$ is SPD
- We do less work finding only the single upper triangular matrix R

```
1  function x = lsnormal(A,b)
2  % LSNORMAL    Solve linear least squares by normal equations.
3  % Input:
4  %   A        coefficient matrix (m by n, m>n)
5  %   b        right-hand side (m by 1)
6  % Output:
7  %   x        minimizer of || b-Ax ||
8
9  N = A'*A;  z = A'*b;
10 R = chol(N);
11 w = forwardsub(R',z);           % solve R'z=c
12 x = backsub(R,w);               % solve Rx=z
```

The Normal Equations

- For polynomial fits, we need a Vandermonde matrix
- The columns are powers of t
- It's poorly conditioned whether using Cholesky factorization or not
- The underlying problem is that the columns are less different as n or the degree increases: try it.

```
n = [10:10:100]';  
condA_2 = zeros(size(n));  
for k=1:length(n)  
    t=[1:n(k)]';  
    A=[t.^2 t ones(size(t))];  
    condA_2(k) = cond(A,2);  
end  
format short  
table(n,condA_2)
```


Overdetermined systems: better

- We need a better way to compute a least squares fit for many problems.
- We can make use of a factorization that creates a matrix with orthonormal columns (an ONC matrix).
- First, consider why an ONC matrix is good.
- If we make orthonormal columns, the conditioning is the best we can do
- In comparison, the Vandermonde matrix is poorly conditioned, but we convert it into something much better

Orthonormal column (ONC) matrices

- Consider a set of vectors $\mathbf{q}_1, \dots, \mathbf{q}_k$.
- Orthogonal if $\mathbf{q}_i^T \mathbf{q}_j = 0$, if $i \neq j$, and nonzero if $i = j$
- Orthonormal if $\mathbf{q}_i^T \mathbf{q}_i = 1$, for all $i = 1, \dots, k$
- Consider square of difference of two vectors:

$$\begin{aligned}\|\mathbf{q}_1 - \mathbf{q}_2\|^2 &= (\mathbf{q}_1 - \mathbf{q}_2)^T (\mathbf{q}_1 - \mathbf{q}_2) \\ &= \mathbf{q}_1^T \mathbf{q}_1 - 2\mathbf{q}_1^T \mathbf{q}_2 + \mathbf{q}_2^T \mathbf{q}_2 = \|\mathbf{q}_1\|^2 + \|\mathbf{q}_2\|^2.\end{aligned}$$

- The difference term drops out: this avoids subtractive cancellation where the difference term becomes negligible

Orthonormal column (ONC) matrices

- Now make a $k \times k$ matrix \mathbf{Q} with ONCs: $\mathbf{q}_1, \dots, \mathbf{q}_k$.

$$\mathbf{Q}^T \mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_k \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1^T \mathbf{q}_1 & \mathbf{q}_1^T \mathbf{q}_2 & \cdots & \mathbf{q}_1^T \mathbf{q}_k \\ \mathbf{q}_2^T \mathbf{q}_1 & \mathbf{q}_2^T \mathbf{q}_2 & \cdots & \mathbf{q}_2^T \mathbf{q}_k \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{q}_k^T \mathbf{q}_1 & \mathbf{q}_k^T \mathbf{q}_2 & \cdots & \mathbf{q}_k^T \mathbf{q}_k \end{bmatrix}$$

- The resulting matrix is a $k \times k$ identity matrix because only diagonal terms are non zero
- Inverse of ONC matrix is easy: $\mathbf{Q}^{-1} = \mathbf{Q}^T$!!

Orthonormal column (ONC) matrices

- For real-valued $n \times k$ matrix \mathbf{Q} with ONCs, Theorem 3.3:

1. $\mathbf{Q}^T \mathbf{Q} = \mathbf{I}$ ($k \times k$ identity)

2. $\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ for all k -vectors \mathbf{x} .

3. $\|\mathbf{Q}\|_2 = 1$.

- For part 2: using 2 norm,

$$\|\mathbf{Q}\mathbf{x}\|_2^2 = (\mathbf{Q}\mathbf{x})^T (\mathbf{Q}\mathbf{x}) = \mathbf{x}^T \mathbf{Q}^T \mathbf{Q} \mathbf{x} = \mathbf{x}^T \mathbf{I} \mathbf{x} = \|\mathbf{x}\|_2^2.$$

Orthonormal column (ONC) matrices

- For real-valued $n \times n$ matrix \mathbf{Q} with ONCs, Theorem 3.4:
 1. \mathbf{Q}^T is also an orthogonal matrix.
 2. $\kappa(\mathbf{Q}) = 1$ in the 2-norm.
 3. For any other $n \times n$ matrix \mathbf{A} , $\|\mathbf{A}\mathbf{Q}\|_2 = \|\mathbf{A}\|_2$.
 4. If \mathbf{U} is another $n \times n$ orthogonal matrix, then $\mathbf{Q}\mathbf{U}$ is also orthogonal.
- Doesn't change the norm of a matrix either, and keeps a matrix orthogonal if it started that way

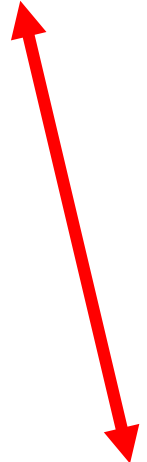
Factoring A into QR

- Theorem 3.5: Every real-valued $m \times n$ matrix A , with $m > n$, can be written as $A=QR$, where:
 - Q is an $m \times m$ orthogonal matrix and
 - R is an $m \times n$ upper triangular matrix
- The result has m orthonormal columns in Q and n nonzero rows for $m > n$

$$A = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_m \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Factoring A into QR

- When $m \gg n$, those zero rows in R , and the last $n + 1:m$ columns in Q are a waste
- A compressed version drops those rows of R and columns of Q
- The compressed form is used for solving the overdetermined system

$$A = \begin{bmatrix} q_1 & q_2 & \cdots & q_m \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$


$$A = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix} = \hat{Q}\hat{R}$$

QR for overdetermined system

- Apply QR factorization to A in rectangular system
- Use compressed form

```
1  function x = lsqrfact(A,b)
2  % LSQRFACT    Solve linear least squares by QR factorization.
3  % Input:
4  %   A        coefficient matrix (m by n, m>n)
5  %   b        right-hand side (m by 1)
6  % Output:
7  %   x        minimizer of || b-Ax ||
8
9  [Q,R] = qr(A,0);                % compressed factorization
10 c = Q'*b;
11 x = backsub(R,c);
```


Factoring A into QR

- Try what happens with different methods
- Start with `vdmonde_cond.m` and `vdmonde_solns.m`

```
% solution methods
%
% N = A'*A; z = A'*b; x_comp = N\z;
% res_sol(k) = norm(z-N*x_comp,2);
%
% x_comp = lsnormal(A,b);
% R = chol(A'*A);
% res_sol(k) = norm(A'*b-(R'*R)*x_comp,2);
%
% x_comp = A\b;
% res_sol(k) = norm(b-A*x_comp,2);
%
% x_comp=lsqrfact(A,b);
% res_sol(k) = norm(b-A*x_comp,2);
```

How to compute the QR factorization?

- The underlying idea is simple. We want to build R by zeroing out each column below the diagonal.
- That part is like constructing U in the LU factorization.
- However, instead of L , we will build an orthogonal matrix Q instead.
- And, we have to be able to do this with rectangular matrices.
- Start with this idea: Can we create an orthogonal matrix that zeros the first column of the matrix A below A_{11} ?
- We want the norm of the original column to replace A_{11} too

Computing the QR factorization

- If we can do this with a matrix, say Q_1 , then we have changed

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \mathbf{a}_3]$$

into (red is changed):

$$Q_1 A = \begin{bmatrix} \|\mathbf{a}_1\| & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{bmatrix}$$

- For this discussion, $\|\mathbf{a}\| = \|\mathbf{a}\|_2$ (norms are 2-norms)

Computing the QR factorization

- If we can do this for one column, then we could operate on the second column with, say Q_2 , such that

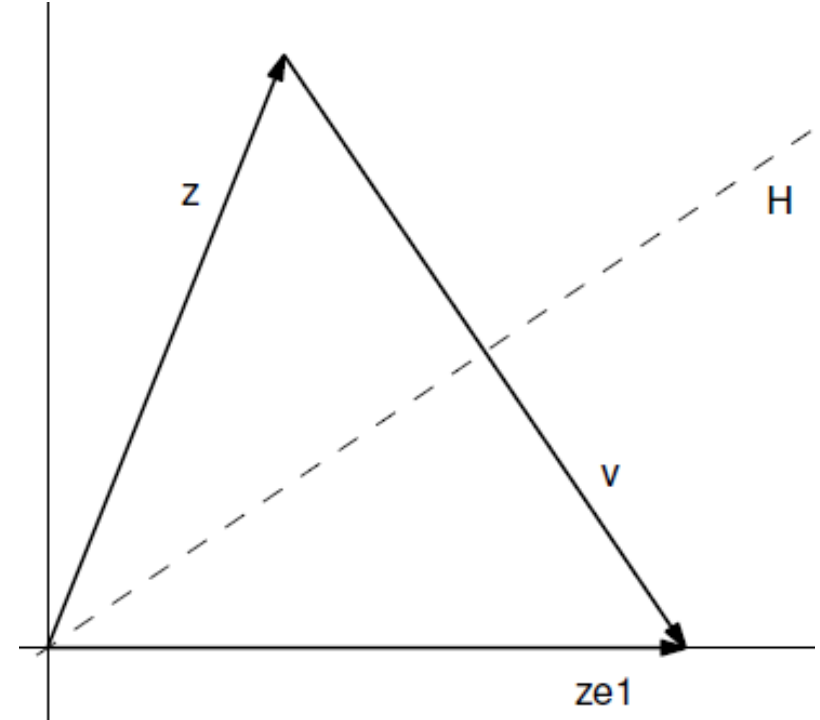
$$Q_2(Q_1 A) = \begin{bmatrix} ||\mathbf{a}_1|| & \textcolor{red}{A}_{12} & \textcolor{red}{A}_{13} \\ 0 & ||\hat{\mathbf{a}}_2|| & \textcolor{blue}{A}_{23} \\ 0 & 0 & \textcolor{blue}{A}_{33} \end{bmatrix}$$

- For this small matrix, this would be R .
- In this case, we only operate on the red 2x2 submatrix, so that we don't mess up what we did with the first column.
- For larger matrices, we can do the same thing, and string more of the Q_j together; these are like the L_j in LU factorization

Computing the QR factorization

- Let's figure out how to do this for a single vector first.
- We want to find \mathbf{P} to get the +/- the norm of the vector times \mathbf{e}_1
- The vector \mathbf{z} is given; define
$$\mathbf{v} = \|\mathbf{z}\|\mathbf{e}_1 - \mathbf{z}$$
- In terms of vectors, \mathbf{v} connects \mathbf{z} , the given vector, to what we want, which is $\|\mathbf{z}\|\mathbf{e}_1$

$$\mathbf{Pz} = \begin{bmatrix} \pm \|\mathbf{z}\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \pm \|\mathbf{z}\| \mathbf{e}_1$$



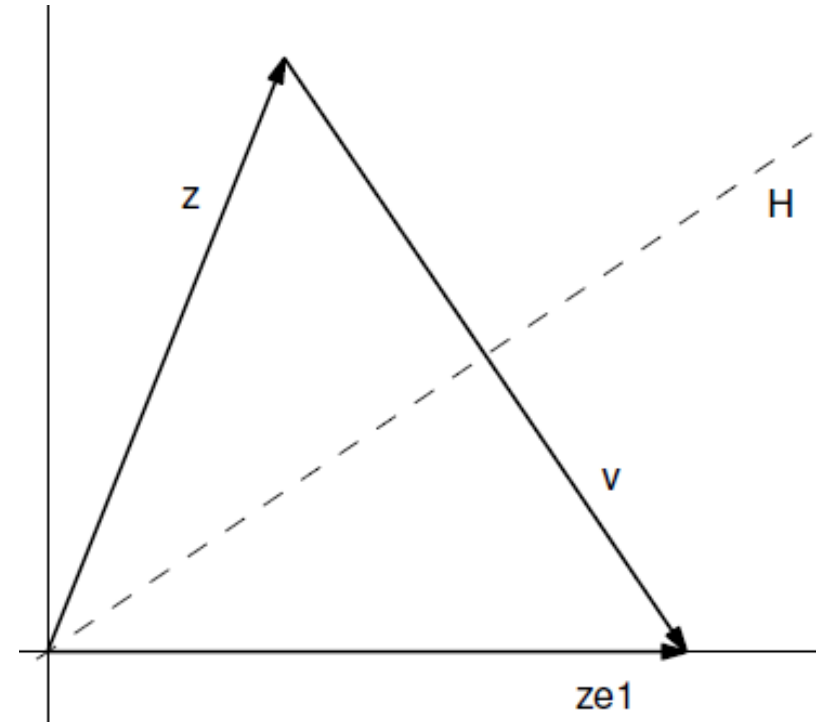
Computing the QR factorization

- H is a sketch of the orthogonal subspace to \mathbf{v}
- To get from \mathbf{z} to $\|\mathbf{z}\|\mathbf{e}_1$, we are reflecting about H

- The matrix that does this is

$$\mathbf{P} = \mathbf{I} - 2 \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}, \mathbf{v} \neq \mathbf{0}$$

- Or $\mathbf{P} = \mathbf{I}, \mathbf{v} = \mathbf{0}$
- Note: $\mathbf{v}\mathbf{v}^T$ is a matrix, $\mathbf{v}^T\mathbf{v}$ scalar

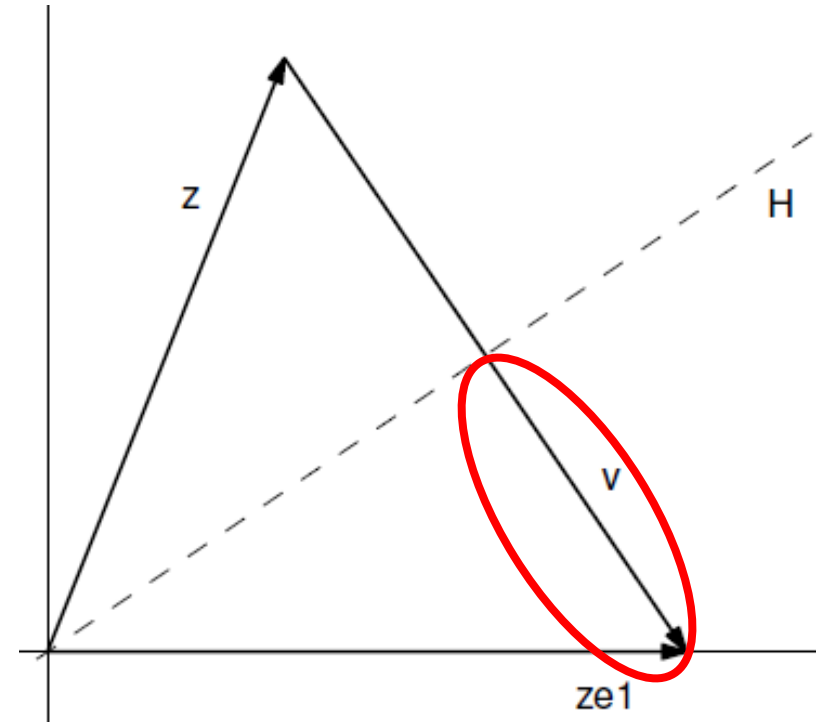


Computing the QR factorization

- How does P work? (i.e., proof)
- First compute Pz
- The matrix that does this is

$$Pz = z - 2 \frac{vv^T}{v^T v} z = z - 2 \frac{z^T v}{v^T v} v$$

- But, the last term is twice the vector component of z in the v direction!
- So, by eye, this works great...



Computing the QR factorization

- This process is called a Householder reflection
- The reflection is about the subspace H that is orthogonal to \mathbf{v}
- The reflection matrix \mathbf{P} is both symmetric ($\mathbf{P} = \mathbf{P}^T$) and orthogonal ($\mathbf{P}^T = \mathbf{P}^{-1}$)
- The size of \mathbf{P} depends on the size of \mathbf{z} : if $\mathbf{z} \in \mathbb{R}^{m \times 1}$ then $\mathbf{P} \in \mathbb{R}^{m \times m}$

Computing the QR factorization

- Proving it now, we need

$$\mathbf{v}^T \mathbf{v} = \|\mathbf{z}\|^2 - 2\|\mathbf{z}\|z_1 + \mathbf{z}^T \mathbf{z} = 2\|\mathbf{z}\|(\|\mathbf{z}\| - z_1),$$

$$\mathbf{v}^T \mathbf{z} = \|\mathbf{z}\|z_1 - \mathbf{z}^T \mathbf{z} = -\|\mathbf{z}\|(\|\mathbf{z}\| - z_1),$$

- Then we find

$$\mathbf{Pz} = \mathbf{z} - 2 \cdot \frac{-\|\mathbf{z}\|(\|\mathbf{z}\| - z_1)}{2\|\mathbf{z}\|(\|\mathbf{z}\| - z_1)} \mathbf{v} = \mathbf{z} + \mathbf{v} = \|\mathbf{z}\| \mathbf{e}_1.$$

- Just what we needed!

Computing the QR factorization

- Now, how to work with one column at a time in

$$A \in \mathbb{R}^{m \times n}, m > n?$$

- There, the first column was \mathbf{a}_1 ; that replaces \mathbf{z} , so that

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

- And then, as before,

$$\mathbf{P}_1 = \mathbf{I} - 2 \frac{\mathbf{v} \mathbf{v}^T}{\mathbf{v}^T \mathbf{v}}$$

- We define $\mathbf{Q}_1 = \mathbf{P}_1$ now, so that the first column is $\|\mathbf{a}_1\| \mathbf{e}_1$, and in general all the other columns are modified.

Computing the QR factorization

- Let $\mathbf{Q}_1 \mathbf{A} = \mathbf{A}^{(1)}$. For the second column, we choose to work with the $m - 1 \times n - 1$ submatrix $\mathbf{A}^{(1)}(2:m, 2:n)$ (in Matlab notation)

- Then, we construct a new \mathbf{P} , the column we work on is $\hat{\mathbf{a}}_2$

$$\hat{\mathbf{a}}_2 = \mathbf{a}_2^{(1)}(2:m) = \mathbf{A}^{(1)}(2:m, 2)$$

- This replaces \mathbf{z} , so that

$$\mathbf{v} = \|\hat{\mathbf{a}}_2\| \mathbf{e}_1 - \hat{\mathbf{a}}_2$$

- And then, as before,

$$\mathbf{P}_2 = \mathbf{I} - 2 \frac{\mathbf{v} \mathbf{v}^T}{\mathbf{v}^T \mathbf{v}}$$

- We define $\mathbf{Q}_2 = \mathbf{I}_{m \times m}$ then set $\mathbf{Q}_2(2:m, 2:m) = \mathbf{P}_2$ now, so that the second column below $\mathbf{A}_{12}^{(1)}$ becomes $\|\hat{\mathbf{a}}_2\| \mathbf{e}_1$, with the whole submatrix modified.

Computing the QR factorization

- Let $\mathbf{Q}_2 \mathbf{Q}_1 \mathbf{A} = \mathbf{A}^{(2)}$. For the third column ($k = 3$), we choose to work with the $m - 2 \times n - 2$ submatrix $\mathbf{A}^{(2)}(\mathbf{3:m}, \mathbf{3:n})$ (in Matlab notation)
- Then, we construct a new $\mathbf{P} = \mathbf{P}_3$, using $\mathbf{v} = \hat{\mathbf{a}}_3$, that is $m - 2 \times m - 2$
- And then, as before,

$$\mathbf{P}_3 = \mathbf{I} - 2 \frac{\mathbf{v} \mathbf{v}^T}{\mathbf{v}^T \mathbf{v}}$$

- We keep going until we are done with the n th column, with $\mathbf{Q}_k = \mathbf{I}_{m \times m}$ then set $\mathbf{Q}_k(k:m, k:m) = \mathbf{P}_k$ until $k = n$ is done

Computing the QR factorization

- Let $\mathbf{Q}_n \dots \mathbf{Q}_2 \mathbf{Q}_1 \mathbf{A} = \mathbf{R}$.
- Then, the orthogonal matrix property is again very handy...
- Premultiplying by the transpose of each of those \mathbf{Q}_j and combining the product acting on \mathbf{R} gives \mathbf{Q} :

$$\mathbf{A} = \mathbf{Q}_1^T \dots \mathbf{Q}_n^T \mathbf{R} = \mathbf{Q} \mathbf{R}, \quad \mathbf{Q} = \mathbf{Q}_1^T \dots \mathbf{Q}_n^T$$

- We can implement a simple version that constructs each of these matrices, but it is inefficient. Explore that first
- Matlab's `qr` function takes about $(2mn^2 - n^3)/3$ flops

Computing the QR factorization

Function from text: qrfact.m

```

1 function [Q,R] = qrfact(A)
2 % QRFACT QR factorization by Householder reflections.
3 % (demo only--not efficient)
4 % Input:
5 %   A      m-by-n matrix
6 % Output:
7 %   Q,R    A=QR, Q m-by-m orthogonal, R m-by-n upper triangular
8
9 [m,n] = size(A);
10 Q = eye(m);
11 for k = 1:n
12     z = A(k:m,k);
13     v = [ -sign(z(1))*norm(z) - z(1); -z(2:end) ];
14     nrmv = norm(v);
15     if nrmv < eps, continue, end % nothing is done in this iteration
16     v = v / nrmv; % removes v'*v in other formulas
17     % Apply the reflection to each relevant column of A and Q
18     for j = 1:n
19         A(k:m,j) = A(k:m,j) - v*( 2*(v'*A(k:m,j)) );
20     end
21     for j = 1:m
22         Q(k:m,j) = Q(k:m,j) - v*( 2*(v'*Q(k:m,j)) );
23     end
24 end
25
26 Q = Q';
27 R = triu(A); % enforce exact triangularity

```

Computing the QR factorization

- Download qrfact.m from Sakai (in files from book)
- Edit qrfact.m into a script or the function qrfactshow.m as in the handout you are given.
- Reproduce that example from qrfactshow.m
- If time permits, I will have additional data.