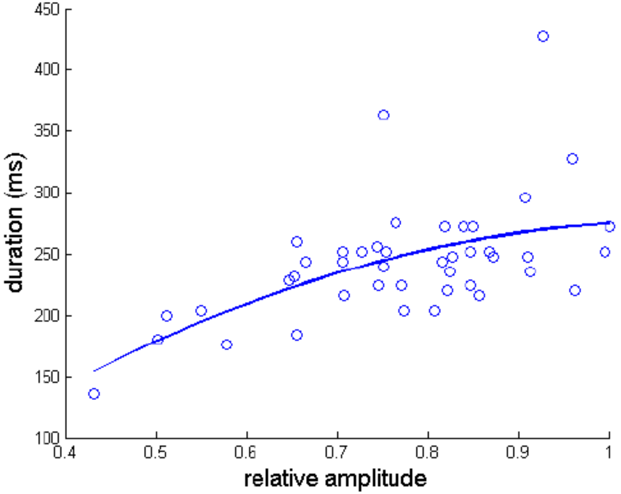
$$\begin{bmatrix} A \end{bmatrix}\begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

# Chapter 3

## Overdetermined linear systems

$$\begin{bmatrix} \\ \\ A \\ \\ \\ \end{bmatrix}\begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} \\ \\ b \\ \\ \\ \end{bmatrix}$$



3.1| Fitting Functions to Data

# 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 $f(x) = a_1 x^2 + a_2 x + a_3$ 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 $n = m = 3$:

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

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

[Example 3.1.1]

# Overdetermined linear systems

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

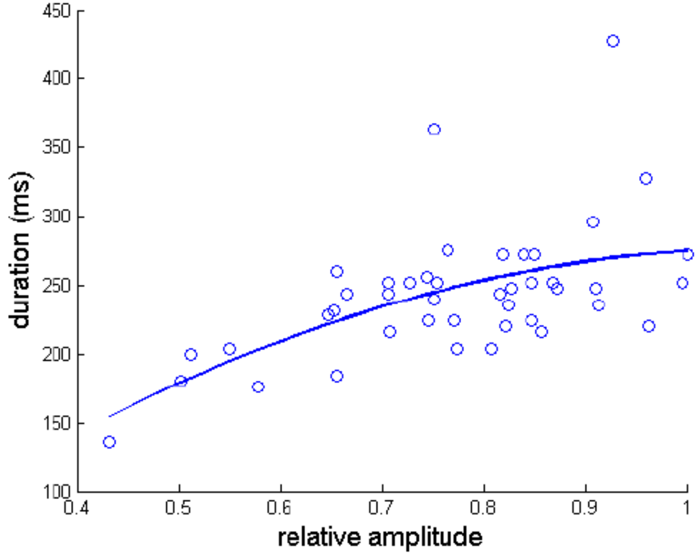
$$\begin{bmatrix} t_1^{n-1} & t_1^{n-2} & \cdots & t_1 & 1 \\ t_2^{n-1} & t_2^{n-2} & \cdots & t_2 & 1 \\ t_3^{n-1} & t_3^{n-2} & \cdots & t_3 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ t_n^{n-1} & t_n^{n-2} & \cdots & t_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}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

- 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




[Example 3.1.2]

# Overdetermined linear systems

$f(x) = \frac{1}{2}\|Ax-b\|_2^2$

- How to solve these problems?

$\nabla f(x) = A^T(Ax-b) = 0$

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

- 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

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

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

$$r = \begin{bmatrix} y_1 - f(t_1) \\ \vdots \\ y_m - f(t_m) \end{bmatrix}$$

# Overdetermined systems

$f_1(t) = 1$

$f_2(t) = t$

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

$f_3(t) = t^2$

$f_4(t) = t^3$

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

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

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

$R(x) = \|b - Ax\|_2^2$ $\quad \nabla R(x) = 2A^T(Ax-b)$

- The unknows ***x*** are the coefficients we seek
- Let's see how to do this…

[Example 3.1.3]

3.2) The Normal Equations

# 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 $\boldsymbol{A}^T(\boldsymbol{Ax}-\boldsymbol{b}) = \boldsymbol{0}$, then x solves the least squares problem $\min_{\boldsymbol{x}} \left|\left|\boldsymbol{b}-\boldsymbol{Ax}\right|\right|_2$

$u^T v = v^T u$

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

$\forall y \in \mathbb{R}^n$

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

# The Normal Equations

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

# The Normal Equations

- Some properties of $\boldsymbol{A}^T\boldsymbol{A}$ 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.* ie use chol, not lu

- For 2, if $\boldsymbol{A}^T\boldsymbol{A}$ is singular, then $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{z} = \boldsymbol{0}$ for nonzero $\boldsymbol{z}$; we need to show that this happens only if $\boldsymbol{A}$ is singular. If we premultiply by $\boldsymbol{z}^T$, then $\boldsymbol{0} = \boldsymbol{z}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{z} = ||\boldsymbol{A}\boldsymbol{z}||_2^2$. This can only happen if $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{0}$; if that happens, $\boldsymbol{A}$ is singular.

# The Normal Equations

- To use the normal equations to solve the least squares problem, do the following:

  $A^TA\,x = A^Tb$

  1. Compute $\boldsymbol{N} = \boldsymbol{A}^T\boldsymbol{A}$
  2. Compute $\boldsymbol{z} = \boldsymbol{A}^T\boldsymbol{b}$
  3. Solve the $n \times n$ linear system $\boldsymbol{N}\boldsymbol{x} = \boldsymbol{z}$ for $\boldsymbol{x}$
- The computation is easy, but the conditioning is poor.
- In the homework, you are asked to prove that $\kappa(A^TA) = \kappa(A)^2$, so that the magnification of the residual may be large.

eg $\kappa(A) = 1000 \implies \kappa(A^TA) = 1\,000\,000$ !

# 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 $4 \times 4$ matrix
- Solving manually for the coefficients $\boldsymbol{a}$ is shown
- The residual for this approach is about 1e-11
- Using a=pinv(A)*y, one gets a residual of about 3e-11, very similar
- Our error could be as bad as 1e6 larger

[Example 3.2.1]

```
>> 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 $\boldsymbol{A}^T\boldsymbol{A}$ is SPD
- We do less work finding only the single upper triangular matrix ***R***

```
function x = lsnormal(A,b)
% LSNORMAL   Solve linear least squares by normal equations.
% Input:
%   A      coefficient matrix (m by n, m>n)
%   b      right-hand side (m by 1)
% Output:
%   x      minimizer of || b-Ax ||

N = A'*A;  z = A'*b;
R = chol(N);
w = forwardsub(R',z);                  % solve R'z=c
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)
```

3.3) The QR Factorization

# Overdetermined systems: better

$A = QR$

- 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

$Q = [q_1 \cdots q_k]$

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

$\|x\|^2 = x^T x$

$$\begin{aligned} \|q_1 - q_2\|^2 &= (q_1 - q_2)^T (q_1 - q_2) \\ &= q_1^T q_1 - 2 q_1^T q_2 + q_2^T q_2 = \|q_1\|^2 + \|q_2\|^2. \end{aligned}$$

- 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 $\boldsymbol{Q}$ with ONCs: $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k$.

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

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

$Qx = b$

$x = Q^T b$

Q is orthogonal

# Orthonormal column (ONC) matrices

- For real-valued $n \times k$ matrix $\boldsymbol{Q}$ 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. $\|\mathbf{Q}\|_2 = 1$. $\Rightarrow$ $\kappa(Q) = \|Q\|_2 \cdot \|Q^{-1}\|_2$ ($n \times n$ $Q$)
     $= 1 \cdot 1 = 1.$

- 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 $\boldsymbol{Q}$ 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* $\mathbf{A}$, $\|\mathbf{AQ}\|_2 = \|\mathbf{A}\|_2$.
  4. *If* $\mathbf{U}$ *is another* $n \times n$ *orthogonal matrix, then* $\mathbf{QU}$ *is also orthogonal.*

$Q^TQ = I$

$(QU)^T(QU) = U^TQ^TQU = U^TIU = U^TU = I$

- Doesn't change the norm of a matrix either, and keeps a matrix orthogonal if it started that way

# Factoring *A* into *QR*

$\boxed{A} = \boxed{Q}\ \boxed{R}$

- Theorem 3.5: Every real-valued $m \times n$ matrix $\boldsymbol{A}$, 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 $m > n$

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

# Factoring $A$ into $QR$

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

$$\mathbf{A} = \overbrace{[\mathbf{q}_1 \quad \mathbf{q}_2 \quad \cdots \quad \mathbf{q}_m]}^{Q} \overbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}}^{R}$$

$$A = \hat{Q}\hat{R}$$

$$\mathbf{A} = \underbrace{[\mathbf{q}_1 \quad \mathbf{q}_2 \quad \cdots \quad \mathbf{q}_n]}_{\hat{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix}}_{\hat{R}} = \hat{\mathbf{Q}}\hat{\mathbf{R}},$$

[Example 3.3.1]

# QR for overdetermined system

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

$$A = \hat{Q}\hat{R}$$

$\hat{Q}$ $m \times n$, ONC
$\hat{R}$ $n \times n$, upper tri. nonsing.

$$A^T A x = A^T b$$

$$\hat{R}^T \hat{Q}^T \hat{Q} \hat{R} x = \hat{R}^T \hat{Q}^T b$$

$$\hat{R}^T \hat{R} x = \hat{R}^T (\hat{Q}^T b)$$

$$\boxed{\hat{R} x = \hat{Q}^T b}$$

use backsub to solve

```
function x = lsqrfact(A,b)
% LSQRFACT   Solve linear least squares by QR factorization.
% Input:
%    A       coefficient matrix (m by n, m>n)
%    b       right-hand side (m by 1)
% Output:
%    x       minimizer of || b-Ax ||

[Q,R] = qr(A,0);                                  % compressed factorization
c = Q'*b;
x = backsub(R,c);
```

# Factoring $A$ into $QR$

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

[Test lsnormal
+ lsqrfact]

```
% 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);
```

3.4) Computing QR Factorizations

# 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 $\boldsymbol{A}$ 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 $\boldsymbol{Q}_1$ , then we have changed

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

into (red is changed):

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

$$Q_1 a_1 = \begin{bmatrix} \|a_1\| \\ 0 \\ 0 \end{bmatrix}$$

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

$$\|Q_1 a_1\| = \|a_1\|$$

# Computing the QR factorization

- If we can do this for one column, then we could operate on the second column with, say $\boldsymbol{Q}_2$ , such that

$$\boldsymbol{Q}_2(\boldsymbol{Q}_1\boldsymbol{A}) = \begin{bmatrix} ||\boldsymbol{a}_1|| & A_{12} & A_{13} \\ 0 & ||\widehat{\boldsymbol{a}}_2|| & A_{23} \\ 0 & 0 & A_{33} \end{bmatrix}$$

- For this small matrix, this would be $\boldsymbol{R}$.
- 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 $\boldsymbol{Q}_j$ together; these are like the $\boldsymbol{L}_j$ in $\boldsymbol{LU}$ factorization

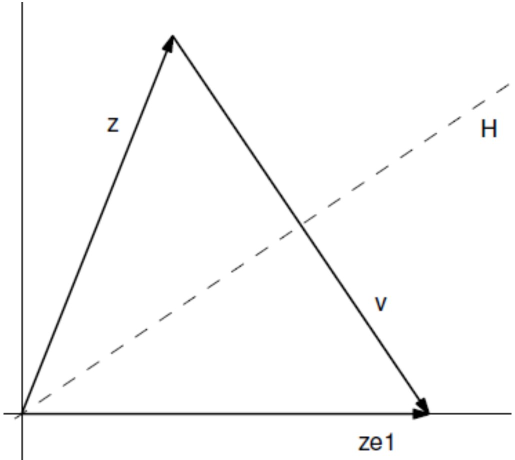
# Computing the QR factorization

- Let's figure out how to do this for a single vector first.
- We want to find $\boldsymbol{P}$ to get the +/- the norm of the vector times $\boldsymbol{e}_1$
- The vector $\boldsymbol{z}$ is given; define

$$\boldsymbol{v} = \|\boldsymbol{z}\|\boldsymbol{e}_1 - \boldsymbol{z}$$

- In terms of vectors, $\boldsymbol{v}$ connects $\boldsymbol{z}$, the given vector, to what we want, which is $\|\boldsymbol{z}\|\boldsymbol{e}_1$

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

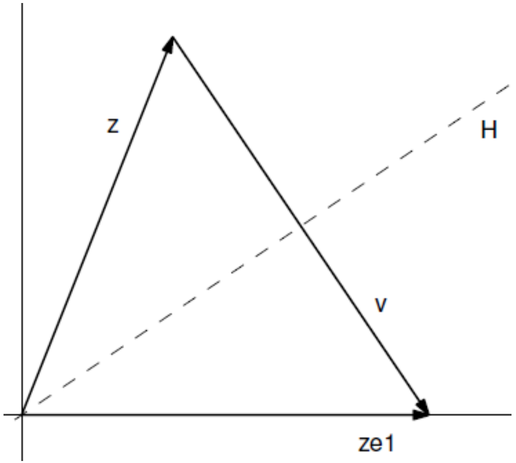


# Computing the QR factorization

- H is a sketch of the orthogonal subspace to $\boldsymbol{v}$
- To get from $\boldsymbol{z}$ to $||\boldsymbol{z}||\boldsymbol{e}_1$, we are reflecting about H
- The matrix that does this is

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

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



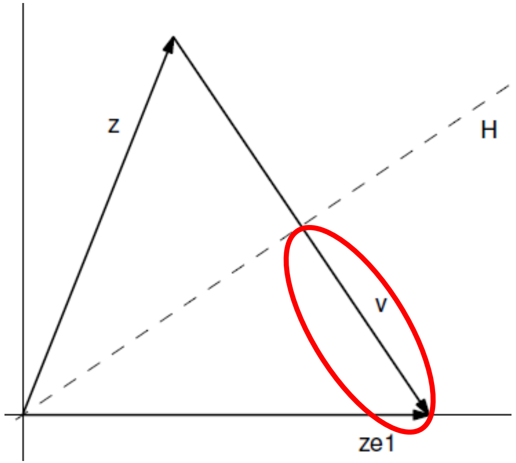

$Pz = ||z|| e_1$

# Computing the QR factorization

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

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

- But, the last term is twice the vector component of $\boldsymbol{z}$ in the $\boldsymbol{v}$ 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 $\boldsymbol{v}$
- The reflection matrix $\boldsymbol{P}$ is both symmetric ($\boldsymbol{P} = \boldsymbol{P}^T$) and orthogonal ($\boldsymbol{P}^T = \boldsymbol{P}^{-1}$)

$P = P^{-1}$ $P^2 = I$

- The size of $\boldsymbol{P}$ depends on the size of $\boldsymbol{z}$: if $\boldsymbol{z} \in \mathbb{R}^{mx1}$ then $\boldsymbol{P} \in \mathbb{R}^{mxm}$

# 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

$$P\mathbf{z} = \mathbf{z} - 2 \cdot \frac{-\|\mathbf{z}\|(\|\mathbf{z}\| - z_1)}{2\|\mathbf{z}\|(\|\mathbf{z}\| - z_1)}\mathbf{v} = \mathbf{z} + \mathbf{v} = \|\mathbf{z}\|\mathbf{e}_1. = \begin{bmatrix} \|z\| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- 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 $\boldsymbol{a}_1$; that replaces $\boldsymbol{z}$, so that
$$\boldsymbol{v} = \ \|\boldsymbol{a}_1\|\boldsymbol{e}_1 - \boldsymbol{a}_1$$
- And then, as before,
$$\boldsymbol{P}_1 = \boldsymbol{I} - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}$$
- We define $\boldsymbol{Q}_1 = \boldsymbol{P}_1$ now, so that the first column is $\|\boldsymbol{a}_1\|\boldsymbol{e}_1$, and in general all the other columns are modified.

[Example 3.4.1]

# Computing the QR factorization

- Let $\boldsymbol{Q}_1\ \boldsymbol{A} = \boldsymbol{A}^{(1)}$. For the second column, we choose to work with the $m-1 \times n-1$ submatrix $\boldsymbol{A}^{(1)}(2{:}\,m, 2{:}\,n)$ (in Matlab notation)
- Then, we construct a new $\boldsymbol{P}$, the column we work on is $\widehat{\boldsymbol{a}}_2$

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

- This replaces $\boldsymbol{z}$, so that

$$\boldsymbol{v} = \ \|\widehat{\boldsymbol{a}}_2\|\boldsymbol{e}_1 - \widehat{\boldsymbol{a}}_2$$

- And then, as before,

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

- We define $\boldsymbol{Q_2} = \boldsymbol{I}_{mxm}$ then set $\boldsymbol{Q}_2(2{:}\,m, 2{:}\,m) = \boldsymbol{P}_2$ now, so that the second column below $A_{12}^{(1)}$ becomes $\|\widehat{\boldsymbol{a}}_2\|\boldsymbol{e}_1$, with the whole submatrix modified.

# Computing the QR factorization

- Let $\boldsymbol{Q_2 Q}_1\, \boldsymbol{A} = \boldsymbol{A}^{(2)}$. For the third column ($k = 3$), we choose to work with the $m - 2 \times n - 2$ submatrix $\boldsymbol{A}^{(2)}(\boldsymbol{3}{:}\,\boldsymbol{m}, \boldsymbol{3}{:}\,\boldsymbol{n})$ (in Matlab notation)
- Then, we construct a new $\boldsymbol{P} = \boldsymbol{P}_3$, using $\boldsymbol{v} = \widehat{\boldsymbol{a}}_3$, that is $m - 2 \times m - 2$
- And then, as before,

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

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

# Computing the QR factorization

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

$$\boldsymbol{A} = \boldsymbol{Q}_1^T \ldots \boldsymbol{Q}_n^T \boldsymbol{R} = \boldsymbol{Q}\boldsymbol{R}, \qquad \boldsymbol{Q} = \boldsymbol{Q}_1^T \ldots \boldsymbol{Q}_n^T$$

- 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

LU $\frac{2}{3}n^3 + O(n^2)$ QR $\frac{5}{6}n^3 + O(n^2)$

# Computing the QR factorization

Function from text: qrfact.m

```
function [Q,R] = qrfact(A)
% QRFACT  QR factorization by Householder reflections.
% (demo only--not efficient)
% Input:
%   A      m-by-n matrix
% Output:
%   Q,R    A=QR, Q m-by-m orthogonal, R m-by-n upper triangular

[m,n] = size(A);
Q = eye(m);
for k = 1:n
  z = A(k:m,k);
  v = [ -sign(z(1))*norm(z) - z(1); -z(2:end) ];
  nrmv = norm(v);
  if nrmv < eps, continue, end       % nothing is done in this iteration
  v = v / nrmv;                      % removes v'*v in other formulas
  % Apply the reflection to each relevant column of A and Q
  for j = 1:n
    A(k:m,j) = A(k:m,j) - v*( 2*(v'*A(k:m,j)) );
  end
  for j = 1:m
    Q(k:m,j) = Q(k:m,j) - v*( 2*(v'*Q(k:m,j)) );
  end
end

Q = Q';
R = triu(A);                         % enforce exact triangularity
```