Another example

assume 3-digit decimal arithmetic.

$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .217 \\ .254 \end{bmatrix}$$

If we compute the solution with pivoting, we obtain

$$x = \begin{bmatrix} -.443 \\ 1.000 \end{bmatrix}, \quad r = \begin{bmatrix} -.000460 \\ -.000541 \end{bmatrix}$$

$$x_{true} = \begin{bmatrix} 1.000 \\ -1.000 \end{bmatrix}$$

- Solution has small residual but very large error
- In fact signs of the solution are opposite!

Condition Number of a Matrix

The first problem is **well-conditioned**; small changes in the data produce small changes in the answer.

The second problem is **ill-conditioned**; small changes in the data can produce large changes in the answer.

A measure of how close a matrix is to singular

$$\text{cond}(A) = \kappa(A) = \|A\| \cdot \|A^{-1}\|$$

$$= \frac{\text{maximum stretch}}{\text{maximum shrink}} = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$$

- $\text{cond}(I) = 1$
- $\text{cond}(\text{singular matrix}) = \infty$

Properties of the condition number

Some properties:

- $\kappa(A) \geq 1$ for all matrices.
- $\kappa(A) = \infty$ for singular matrices.
- $\kappa(cA) = \kappa(A)$ for any nonzero scalar c.
- $\kappa(D) = \max |d_{ii}| / \min |d_{ii}|$ if D is diagonal.
- κ measures closeness to singularity better than the determinant.

Relation between condition number and error

$$Ax_{true} = b \quad \rightarrow \quad ||b|| = ||Ax_{true}|| \leq ||A||\,||x_{true}||$$

$$||x_{true}|| \geq \frac{||b||}{||A||} \quad \rightarrow \quad \frac{\mathbf{1}}{||\mathbf{x_{true}}||} \leq \frac{||\mathbf{A}||}{||\mathbf{b}||}$$

$$Ax = b - r \quad \rightarrow \quad A(x_{true} - x) = r$$

$$(x_{true} - x) = A^{-1}r \quad \rightarrow \quad ||\mathbf{x_{true}} - \mathbf{x}|| \leq ||\mathbf{A}^{-1}||\,||\mathbf{r}||$$

$$\begin{aligned} \frac{||\mathbf{x_{true}} - \mathbf{x}||}{||\mathbf{x_{true}}||} &\leq \frac{||\mathbf{r}||}{||\mathbf{b}||}||\mathbf{A}||||\mathbf{A}^{-1}|| \\ &= \frac{||r||}{||b||}\kappa(A)\,. \end{aligned}$$

- In words: relative error is smaller than norm of residual divided by norm of rhs times condition number
- So larger condition number means larger error

Closing remarks

- Never compute matrix inverse
- Use a stable algorithm
- Check residual and condition number of problem
- If condition number is large, do not trust solution
  - Can problem be reformulated somehow?

LU code

- Initialize
  - Matrix size
  - Permutation vector
- Second output argument to `max` is index of max element
- If max element is zero then we need not eliminate
- Exchange rows
- update permutation vector

```
%LU Triangular factorization
%   [L,U,p] = lutx(A) produces a unit lower triangular
%   matrix L, an upper triangular matrix U, and a
%   permutation vector p, so that L*U = A(p,:).

[n,n] = size(A);
p = (1:n)'

for k = 1:n-1

   % Find largest element below diagonal in k-th column
   [r,m] = max(abs(A(k:n,k)));
   m = m+k-1;

   % Skip elimination if column is zero
   if (A(m,k) ~= 0)

      % Swap pivot row
      if (m ~= k)
         A([k m],:) = A([m k],:);
         p([k m]) = p([m k]);
      end
```

Look at LU code

- Multipliers for each row below diagonal
  - Note multipliers are stored in the lower triangular part of A
- Vectorized update
  - A(i,k)*A(k,j) multiplies column vector by row vector to produce a square, rank 1 matrix of order n-k.
  - matrix is then subtracted from the submatrix of the same size in the bottom right corner of A.
  - In a programming language without vector and matrix operations, this update of a portion of A would be done with doubly nested loops on i and j.
  - Cost is n^2 and done n times for a total cost of n^3
- Computes decomposition in the matrix A itself
- Here they are separated, but when memory is important it can be left there

```
        % Compute multipliers
        i = k+1:n;
        A(i,k) = A(i,k)/A(k,k);

        % Update the remainder of the matr
        j = k+1:n;
        A(i,j) = A(i,j) - A(i,k)*A(k,j);
    end
end

% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
```

Code to solve linear system using LU

- In Matlab the backslash operator can be used to solve linear systems.
- For square matrices it employs LU or special variants
  - Lower triangular
  - Upper triangular
  - symmetric
- Symmetric LU is called Cholesky decomposition
  - $A=LL^T$
  - Upper and lower triangular are equal (transposes)
  - If matrix not positive-definite go to regular solution

```
function x = bslashtx(A,b)
% BSLASHTX  Solve linear system (backslash)
% x = bslashtx(A,b) solves A*x = b

[n,n] = size(A);
if isequal(triu(A,1),zeros(n,n))
   % Lower triangular
   x = forward(A,b);
   return
elseif isequal(tril(A,-1),zeros(n,n))
   % Upper triangular
   x = backsubs(A,b);
   return
elseif isequal(A,A')
   [R,fail] = chol(A);
   if ~fail
      % Positive definite
      y = forward(R',b);
      x = backsubs(R,y);
      return
   end
end
```

Code continues

- Call LU
  - Solve y=Lb
  - Solve x=Uy

```
% Triangular factorization
[L,U,p] = lutx(A);

% Permutation and forward elimination
y = forward(L,b(p));
x = backsubs(U,y);
```

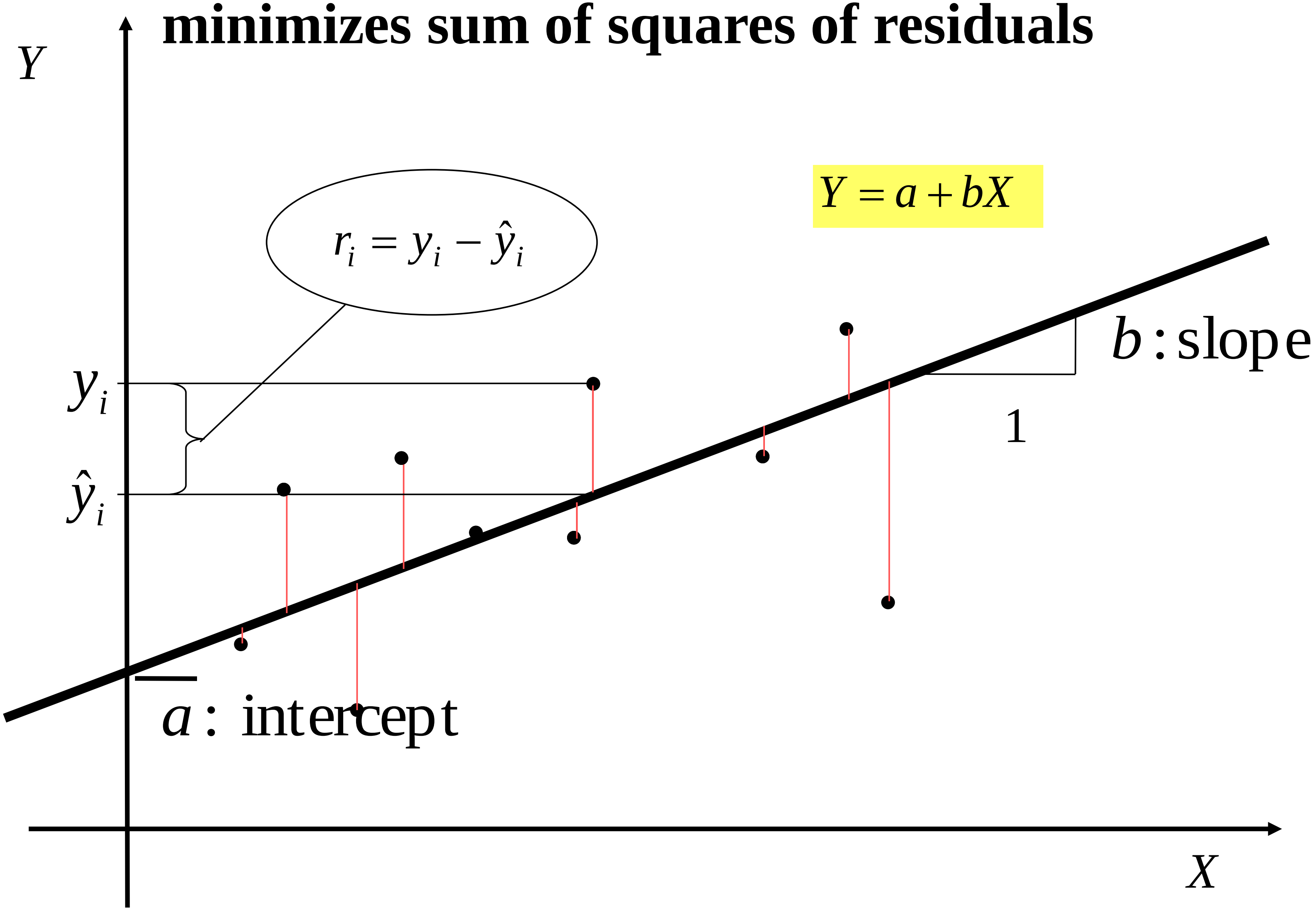
```
function x = forward(L,x)
% FORWARD. Forward elimination.
% For lower triangular L, x = forward(L,b) solves L*x = b.
[n,n] = size(L);
for k = 1:n
   j = 1:k-1;
   x(k) = (x(k) - L(k,j)*x(j))/L(k,k);
end
```

```
function x = backsubs(U,x)
% BACKSUBS.  Back substitution.
% For upper triangular U, x = backsubs(U,b) solves U*x = b.
[n,n] = size(U);
for k = n:-1:1
   j = k+1:n;
   x(k) = (x(k) - U(k,j)*x(j))/U(k,k);
end
```

Fitting data to a model

- Practical science involves lots of fitting of data to models
- Tasks arise commonly in science
  - Fit straight lines and curves to data
  - More generally fit a parametric model to data
- Parametric: Model contains parameters
  - Job of fitting is to estimate the parameters that "best" make the model fit the data
  - "best" → define best
  - For this section, best is least square error
- Simplest example of model fitting problem
  - Linear regression

Want to estimate Regression Line that minimizes sum of squares of residuals



How do we find a and b?

Find *a* and *b* by minimizing sum of squares of individual point residuals, with respect to *a* and *b*

$$E(a,b)=\sum_{i=1}^{N}(r_i)^2 = \sum_{i=1}^{N}(y_i - [bx_i + a])^2$$

- Differentiate cost function with respect to *b* and *a* and get two equations in two unknowns

$$\frac{\partial E}{\partial a} = -\sum_{i=1}^{n} 2(y_i - (a + bx_i))\frac{\partial(a+bx_i)}{\partial a} = 0 \qquad n\,a + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$\frac{\partial E}{\partial b} = -\sum_{i=1}^{n} 2(y_i - (a + bx_i))\frac{\partial(a+bx_i)}{\partial b} = 0 \qquad a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i\, y_i\,.$$

Least Squares for more unknowns

- Suppose we wish to solve

$$\mathbf{Ac}=\mathbf{y}$$

A is a $m \times n$ matrix, **c** is a n vector, and **y** is a m vector

- Look for solution **c** that minimizes same cost function, the sum of squares of residuals r_i at each data point x_i

$$F(\mathbf{c})=\|\mathbf{Ac}-\mathbf{y}\|_2^2$$

$$F(\mathbf{c}) = \sum_{i=1}^{m} r_i^2 = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} A_{ij}c_j - y_i\right)^2 = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} A_{ij}c_j - y_i\right)\left(\sum_{k=1}^{n} A_{ik}c_k - y_i\right)$$

Differentiate with respect to the unknowns c_l and set to zero

$$\frac{\partial F}{\partial c_l} = \sum_{i=1}^{m}\left[\left(\sum_{j=1}^{n} A_{ij}\frac{\partial c_j}{\partial c_l}\right)\left(\sum_{k=1}^{n} A_{ik}c_k - y_i\right) + \left(\sum_{j=1}^{n} A_{ij}c_j - y_i\right)\left(\sum_{k=1}^{n} A_{ik}\frac{\partial c_k}{\partial c_l}\right)\right] = 0$$

- Derivative is 1 for $j=l$ (or $k=l$), otherwise zero. Define

$$\frac{\partial c_k}{\partial c_l} = \delta_{kl} = \begin{cases} 1 & \text{if } k = l \\ 0 & \text{otherwise} \end{cases}$$

  – Also called Kronecker Delta
  – This yields the following equation for each l

$$\sum_{i=1}^{m}\left[\left(\sum_{k=1}^{n} A_{il}A_{ik}c_k - A_{il}y_i\right) + \left(\sum_{j=1}^{n} A_{ij}A_{il}c_j - A_{il}y_i\right)\right] = 0$$

$$2\left(\sum_{i=1}^{m}\sum_{k=1}^{n} A_{il}A_{ik}c_k - \sum_{i=1}^{m} A_{il}y_i\right) = 0$$

- Can recognize these as the following equation

$$2\left(\mathbf{A}^t\mathbf{A}\mathbf{c} - \mathbf{A}^t\mathbf{y}\right) = 0 \quad \text{or} \quad \mathbf{A}^t\mathbf{A}\mathbf{c} = \mathbf{A}^t\mathbf{y}$$

Normal equations $\mathbf{A}^t \mathbf{A} \mathbf{c} = \mathbf{A}^t \mathbf{y}$

- For **A** size $m \times n$ and **c** of size n and **y** of size m what are the dimensions of the normal equations?
  - $n \times n$
- Have converted it to a regular system that we know how to solve
- Solve via LU decomposition
- Solution accurate if the matrix $\mathbf{A}^t\mathbf{A}$ is well conditioned
- Cost of solving normal equations
  - Matrix product n^2m operations.
  - Matrix vector product nm operations
  - LU decomposition $n^3/3$ operations

More on Normal Equations

- Normal equations are only important theoretically
- In practice least squares solved via a different process
  - QR decomposition
- Why?
  - Somewhat expensive as we have to form $A^t A$
  - involves matrix multiplication and then solution
  - More importantly it is poorly conditioned
  - $\text{cond}(A^tA) = (\text{cond}(A))^2$
- Would like a method whose errors are closer to the condition number of A

Look at the fitting matrix in more detail

- Instead
  - Look for methods that can directly operate on **A** to get the solution
  - Recall in LU we did a set of transformations to **A** and the r.h.s. to find **c**
  - Today we will look at the QR algorithm
- Goal in least squares is to find the coordinates of the vector in the ***column space of matrix*** that ***best approximates*** the right hand side in a ***least squares sense***
- ***Matrix vector product***
  - ***Produces a vector that is a combination of column vectors of matrix***

Key Ideas

- Column space of a matrix: the vector space formed by the collection of column vectors in a matrix
- Every matrix vector product results in a vector formed by linear combination of vectors in the column space

$$\begin{bmatrix} 1 & 8 & 13 & 12 \\ 14 & 11 & 2 & 7 \\ 4 & 5 & 16 & 9 \\ 15 & 10 & 3 & 6 \end{bmatrix}$$

- A $m \times n$ rectangular matrix **A** takes n vectors into m vectors
- Let the least squares problem be **Ac**=**f**
- Let the solution which minimizes the residual be $\mathbf{c}_*$
- Then $\mathbf{c}_*$ creates on matrix vector product a rhs $\mathbf{f}_*$ that is in the column space of **A**
- We want that $\mathbf{c}_*$ minimizes $\mathbf{r}=||\mathbf{f}-\mathbf{f}_*||$

Null Space of A

- Not all *m* vectors will be reachable even if we supply arbitrary *n* vectors
- *Range* of A: the part of the space of *m* vectors that are reachable

$$Range(\text{A}) = \{\mathbf{y} \in R^m : \mathbf{y}=\mathbf{Ax} \text{ for some } \mathbf{x} \in R^n\}$$

  - ***The range of* A *contains all those vectors that can be made up using the columns of* A**
  - *Rank*(**A**) is the dimension of the range of **A**
  - Null space of **A**: those vectors x, for which Ax is zero

$$\text{Null}(\mathbf{A}) = \{\mathbf{x} \in R^n : \mathbf{Ax}=0\}$$

$$\text{Dim}(\text{Null}(\mathbf{A}))+\text{Rank}(\mathbf{A})=n$$

- Key idea: Minimize the error in the part that can be reached

QR decomposition

- Suppose we can write

$$\mathbf{A}=\mathbf{Q}'\mathbf{R}'$$

  – **Q**' is an orthonormal matrix of dimension $m \times m$
  – **R**' is a $m \times n$ matrix that can be written as $\begin{bmatrix}\mathbf{R} \\ \mathbf{0}\end{bmatrix}$

  R is a triangular $n \times n$ matrix and **0** is a matrix of zeroes of size $m\text{-}n \times n$

  Q' can also be partitioned as [**Q** Q~] with **Q** containing n orthonormal columns of size m and **Q**~ $m\text{-}n$ orthonormal columns

- If **Ax**=**b** then (**Q**' **R**')**x**=**b** or **Q**'(**R**'**x**)=**b** or **Q**'**y**=**b**
  – So if **b** is in range(**A**), it is also in range(**Q**')
  – Similarly if **Q**'**y**=**b**; then **b**=**Ax** with $\mathbf{x}=\mathbf{R}^{-1}\mathbf{y}$
  – Columns of **Q** form an orthonormal basis for range(**A**)

Orthogonal Matrices

- Orthogonal matrices are square matrices that have their columns orthonormal to each other
  - dot product of different column vectors is zero, while of the same column is one
  - Denoted **Q**
  - Most trivial orthogonal matrix is the identity matrix
  - $\mathbf{Q}^t\mathbf{Q}=\boldsymbol{\Lambda}$
  - For an orthonormal matrix
  - $\mathbf{Q}^t\mathbf{Q}=\mathbf{I}$
  - So $\mathbf{Q}^{-1}=\mathbf{Q}^T$

generalization: a complex matrix is *Hermitian* iff $\mathbf{Q}^{-1}=\mathbf{Q}^H$ where superscript ^{H} denotes complex conjugate transpose

Orthogonal matrix facts

- Suppose **Q** is an orthonormal matrix
- Then for any vector **r** the Euclidean norm is preserved in an orthonormal transformation
- Proof

$$||\mathbf{Q}r||^2=(Qr)^t\,(Qr)= r^t\,Q^t\,Q\,r = r^t\,(Q^t\,Q)\,r =r^t\,r =||r||^2$$

- If Q is an orthonormal matrix so is the extended matrix Q_e

$$Q_e = \begin{bmatrix} I & 0 \\ 0 & Q \end{bmatrix}$$

- Easy to show from definition that

$$Q_e^t\,Q_e = I$$

Solving least squares with QR

- $A=Q'R'$
- Let $r= y-Ac$ $b=Q'^t y$
- Goal of least squares find the c that minimizes squared error (residue)
- Partition b in to two pieces
  - b_1 of dimension *n*
  - b_2 of dimension *m-n*
  - $||r||^2 = || y- Ac||^2 = ||y - Q' R' c||^2$
  - Length is not changed by multiplication with orthogonal matrix
  - So $||r||^2 =||Q'^t r||^2 =||Q'^t [y - Q' R' c]||^2$

$$=||b_1 - R c ||^2 + ||b_2 - 0c||^2$$

So no matter what c is the second term remains unchanged

If we minimize $||r||^2$ the best we can do is minimize first term

Solving LS via QR

- How do we minimize $||c_1 - R\,x\,||^2$
  - If R is full rank set solve Rx=c then we have done the best we can
  - (if R is rank deficient solve in least squares sense)
  - Recall R is triangular so this equation can be easily solved
- Algorithm
  - Compute QR factorization of A=Q'R'
  - Form $c_1=Q^t\,b$
  - Solve $Rx=c_1$
  - If R is full rank and $Q^{\sim}$ is available then the norm of the residual is $||Q^{\sim t}\,b||$. Else $r = b - A\,x$.

Computing the QR factorization

- In LU: Converted matrix A to triangular matrix U by adding multiples of other rows
  - Elements below a given column were zeroed out
  - The multipliers were stored in L which gave us A=LU
- Here want to zero out entries below the diagonal and convert to triangular matrix R but do it with orthogonal matrices
- Two strategies
- Zero out a column at a time using a matrix Q_1 so that Q^t_1 A gives us all entries below a certain one in a column as zero
  - Householder transformations
  - Result $Q^t_n \ldots Q^t_2 Q^t_1 A = R$ or $A = Q_1 \ldots Q_{n-1} Q_n R = Q R$
- Zero out one specific entry of a column at a time
  - Givens rotations
- Product of orthogonal matrices is orthogonal

To compute QR

- Perform a sequence of orthogonal transformations that zero out elements
- Orthogonal transformations can be rotations or reflections or combinations
- Givens Rotation:
- Givens matrix has elements

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

- $c^2+s^2=1$
- How do we use a rotation to zero out an element?
- Let z= $[z_1\ z_2]^t$
- We want

$$Gz = \begin{bmatrix} cz_1 + sz_2 \\ sz_1 - cz_2 \end{bmatrix} = xe_1$$

- Eliminate z_2

$$(c^2 + s^2)z_1 = cx\,, \qquad c = z_1/x\,.$$

- Similarly we get $s=z_2/x$, and $z_1^2+z_2^2 = x^2$

Givens QR

- To apply idea to larger matrix, embed the Givens matrix in identity matrix. We will use the notation G_{ij} to denote an $n \times n$ identity matrix with its ith and jth rows modified to include the Givens rotation: for example, if $n = 6$, then

$$G_{25} = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{c} & 0 & 0 & \mathbf{s} & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{s} & 0 & 0 & -\mathbf{c} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix},$$

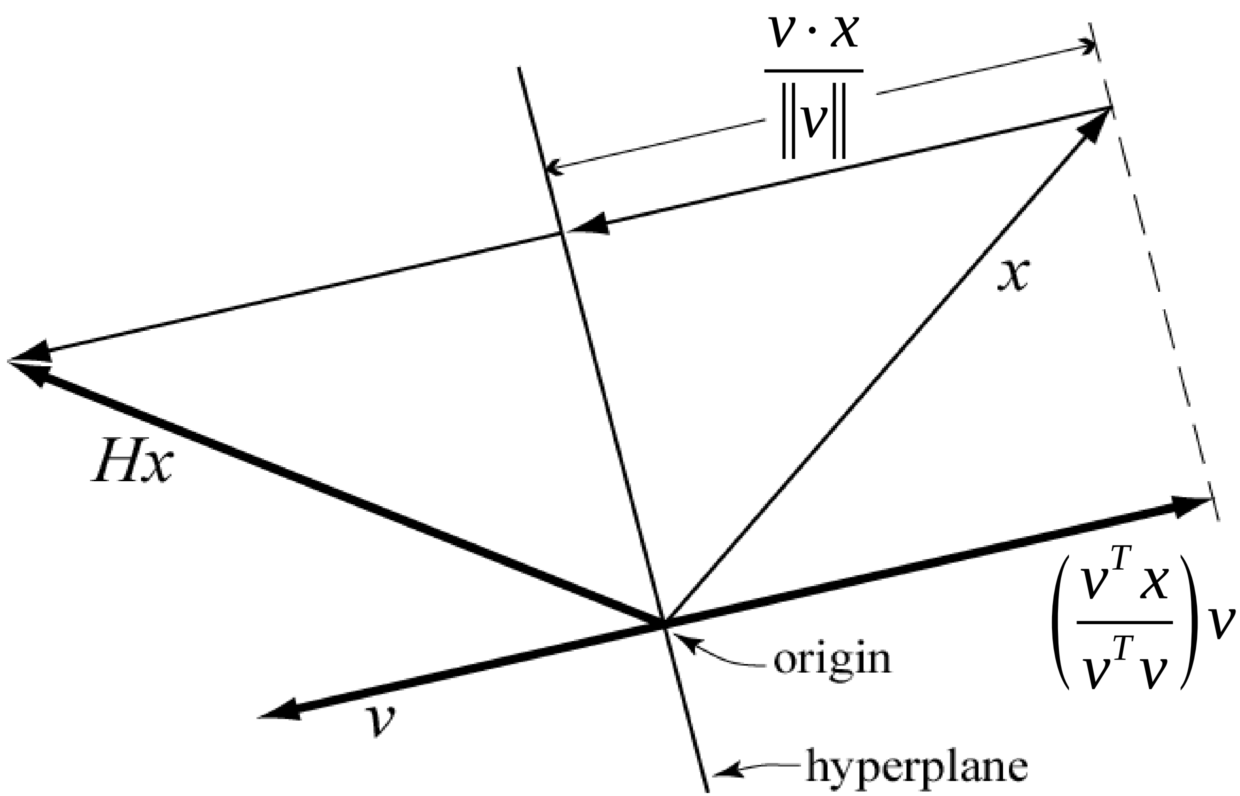
and multiplication of a vector by this matrix leaves all but rows 2 and 5 of the vector unchanged.

- Algorithm

for $i=1, \ldots, n$
    for $j=i+1, \ldots, m$
        Find Givens matrix G_{ij} to zero out j,i element of A using the the value at position (i,i)
        $A=G_{ij}A$
    end
end

Householder Geometry

- ***Hx* is *x* reflected through the hyperplane perpendicular to *v*** ($p : p^{\mathrm{T}}v=0$)



Householder Transformations

The *Householder transformation* determined by vector v is:

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

outer product, n×n matrix

inner product, scalar

To apply it to a vector *x*, compute:

$$Hx = \left(I - 2\frac{vv^T}{v^T v} \right) x = x - 2\frac{v(v^T x)}{v^T v}$$

$$\boxed{Hx = x - \left(2\frac{v^T x}{v^T v} \right) v}$$

scalar

Householder Properties

- H is symmetric, since

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

- H is orthogonal, since

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

$$= I - 4\frac{vv^T}{v^T v} + 4\frac{v(v^T v)v^T}{(v^T v)^2} = I - 4\frac{vv^T}{v^T v} + 4\frac{vv^T}{v^T v} = I$$

and $H^T H = I$ implies $H^T = H^{-1}$

Householder to Zero Matrix Elements

We'll use Householder transformations to zero subdiagonal elements of a matrix.

Given any vector *a*, find the *v* that determines an *H* such that,

$$Ha = \alpha e_1 = \alpha[1, 0, 0, \ldots, 0]^T$$

Now solve for *v*:

$$Ha = a - \left(2\frac{v^T a}{v^T v}\right)v = a - \mu v = \alpha e_1$$

where μ is parenthesized scalar, related to length of v

$$\Rightarrow \; v = (a - \alpha e_1) / \mu$$

We're free to choose $\mu = 1$, since $\|v\|$ does not affect H

Choosing the Vector *v*

So $v = a - \alpha e_1$ for some scalar α.

But $\|Ha\|_2 = \|a\|_2$

(prove this by expanding $\|Ha\|_2^2 = (Ha)^T Ha$)

and $\|Ha\|_2 = |\alpha|$ by design, so $\alpha = \pm\|a\|_2$

(either sign will work).

To avoid $v \approx 0$ we choose $\alpha = -\text{sign}(a_1)\|a\|_2$,

so $v = a + \text{sign}(a_1)\|a\|_2 e_1$ is our answer.

Applying Householder Transforms

- Don't compute Hx explicitly, that costs $3n^2$ flops.
- Instead use the formula given previously,

$$Hx = x - \left(2 \frac{v^T x}{v^T v} \right) v$$

which costs $4n$ flops (if you pre-compute $v^T v$ or pre-normalize $v^T v=2$).

- Typically, when using Householder transformations, you never compute the matrix H; it's only used in derivation and analysis.

Rank Revealing QR: AP=QR

- Crucial addition similar to pivoting

for k=1:N

  - Compute the norm of columns A(k:M, k:N)
  - If max norm of all columns is below threshhold stop
  - Swap column *k* of the matrix with the column with maximum norm
  - Compute Householder transform using that column
  - Apply to other columns

end

- Store column swaps in a permutation

QR Decomposition

- Householder transformations are a good way to zero out subdiagonal elements of a matrix.
- A is decomposed:

$$Q^T A = \begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{or} \quad QQ^T A = A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$$

- where $Q^{\mathrm{T}}=H_n \ldots H_2 H_1$ is the orthogonal product of Householders and R is upper triangular.
- Overdetermined system $Ax=b$ is transformed into the easy-to-solve

$$\begin{bmatrix} R \\ 0 \end{bmatrix} x = Q^T b$$

Other Norms

- Here we fit using the “least-squares” or L_2 norm
- Could minimize the residual in other norms
- For example we may have more confidence in some data, and want to be sure that their residual is lower
  - Attach a weight to each residual

$$\|r\|_w^2 = \sum_1^m w_i r_i^2$$

- Or we may like the 1-norm or infinity norm better

$$\|r\|_1 = \sum_1^m |r_i| \qquad \|r\|_\infty = \max_i |r_i|$$

SVD and Pseudo-Inverse

- $\mathbf{Ax=b}$ $\mathbf{A}$ is $m \times n$, $\mathbf{x}$ is $n \times 1$ and $\mathbf{b}$ is $m \times 1$.
- $\mathbf{A=USV}^t$ where $\mathbf{U}$ is $m \times m$, $\mathbf{S}$ is $m \times n$ and $\mathbf{V}$ is $n \times n$
- $\mathbf{USV}^t\,\mathbf{x=b}$. So $\mathbf{SV}^t\,\mathbf{x=U}^t\mathbf{b}$
- If $\mathbf{A}$ has rank r, then r singular values are significant

$\mathbf{V}^t\mathbf{x} = \mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_r^{-1}, 0, \ldots, 0)\mathbf{U}^t\mathbf{b}$

$\mathbf{x} = \mathbf{V}\mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_r^{-1}, 0, \ldots, 0)\mathbf{U}^t\mathbf{b}$

$$\mathbf{x}_r = \sum_{i=1}^{r} \frac{\mathbf{u}_i^t \mathbf{b}}{\sigma_i} \mathbf{v}_i \qquad \sigma_r > \varepsilon, \quad \sigma_{r+1} \le \varepsilon$$

- Pseudoinverse $\mathbf{A}^+ = \mathbf{V}\,\mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_r^{-1}, 0, \ldots, 0)\,\mathbf{U}^t$
  - $\mathbf{A}^+$ is a $n \times m$ matrix.
  - If rank $(\mathbf{A}) = n$ then $\mathbf{A}^+ = (\mathbf{A}^t\mathbf{A})^{-1}\mathbf{A}$
  - If $\mathbf{A}$ is square $\mathbf{A}^+ = \mathbf{A}^{-1}$

Q~ forms Nullspace of (A^t)

- Choose z in nullspace of A^t
- Let $A^t z = 0$
  - $(Q'R')^t z = R'^t Q'^t z = 0$
  - So $R^t y = 0$ for $y = Q^t z$
  - If R is full rank this means y has to be the zero vector
  - So $Q^t z = 0$
  - So z must be composed of the elements from $Q^{\sim}$
  - So the columns of $Q^{\sim}$ form an orthonormal basis for nullspace(A^t)