MA 580; Linear Least Squares

C. T. Kelley
NC State University
tim_kelley@ncsu.edu

Version of September 20, 2016

Time for sections 3.7 and 3.8 and chapters 4 and 5 of Pink book.

NCSU, Fall 2016

Part Va: Linear Least Squares, Normal Equations, QR Factorization

Data fitting or let's get rich

I will predict the stock market on the assumption that the Dow Jones Industrial Average depends on the prior day's

- phase of the moon, $p(t)$,
- mean temperature in Central Park, $c(t)$,
- solar radiation in Grand Canyon, $r(t)$

linearly

$$Dow(t) = x_1 p(t-1) + x_2 c(t-1) + x_3 r(t-1)$$

Easy path to great wealth

All I need to do is figure out what

$$\mathbf{x} = (x_1, x_2, x_3)^T$$

is. Let's collect historical data for a 300 business days

$$\begin{aligned}
\mathbf{a}_1 &= (p(t_1), p(t_2), \ldots p(t_{300}))^T \\
\mathbf{a}_2 &= (c(t_1), c(t_2), \ldots c(t_{300}))^T \\
\mathbf{a}_3 &= (r(t_1), r(t_2), \ldots r(t_{300}))^T \\
\mathbf{b} &= (Dow(t_1 + 1), Dow(t_2 + 1), \ldots Dow(t_{300} + 1))^T.
\end{aligned}$$

and tune $\mathbf{x}$ to make the model look the best.

How might we do that?

A least squares fit would find $\mathbf{x}$ by minimizing

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$$

where $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$.
We could the use the solution to . . .

- use the coefficients to predict the stock market on Monday,
- sink life savings into that prediction, and
- drive away in Lamborghini on Tuesday.

Isn't math great?

Linear Least Squares

Problem, find $\mathbf{x} \in R^N$ so that

$$\|\mathbf{Ax} - \mathbf{b}\|_2$$

is minimized.
Here $\mathbf{A}$ is $M \times N$.

- $M \geq N$ is **overdetermined** show them the geometry
- $M = N$ is square (it's an equation)
- $M < N$ is **underdetermined**

We'll begin with the overdetermined case.
In our stock market example $M = 300$ and $N = 3$.

Overdetermined Least Squares Problems

- The **Range** of **A**, $Ran(\mathbf{A})$ is the span of the columns

$$Ran(A) = \{\mathbf{Ax} \,|\, \mathbf{x} \in R^N\} \subset R^M.$$

- The solution of the least squares problem is **x**, where **Ax** is the projection of **b** onto $Ran(\mathbf{A})$.

Solution with calculus

- Assumption: The columns of **A** are linearly independent. In this case we say **A has full column rank**.
- This implies (check it) that $\mathbf{A}^T\mathbf{A}$ is spd.

So, our job is to minimize

$$\phi(\mathbf{x}) = \sum_{i=1}^{N} \left(\sum_{j=1}^{N} a_{ij} x_j - b_i \right)^2$$

Calculus says we should look at points where $\nabla\phi = 0$, ie

$$\partial\phi(\mathbf{x})/\partial x_k = 0 \text{ for all } 1 \le k \le N$$

Gradient of ϕ

Check this out if it's not clear ...

$$\frac{\partial \phi(\mathbf{x})}{\partial x_k} = 2\sum_{i=1}^{N}\left(\sum_{j=1}^{N} a_{ij}x_j - b_i\right) a_{ik} = 2(\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b})_k$$

So $\nabla\phi(\mathbf{x}) = 0$ if and only if

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

These are the **normal equations**

Is this a max or a min?

It's a minimum for many reasons.

- $\phi(\mathbf{x}) \to \infty$ as $\|\mathbf{x}\| \to \infty$ because $\mathbf{a}$ has full column rank.
- The Hessian matrix

$$\mathbf{H}_{ij} = \frac{\partial^2 \phi(\mathbf{x})}{\partial x_i \partial x_j}$$

is $2\mathbf{A}^T\mathbf{A}$ and is spd (because $\mathbf{A}$ has full column rank).

The Normal Equations Method

Solve

$$\min \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$$

by using a Cholesky factorization to solve the normal equations

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

Cost:

- $N^2 M$ for the matrix product
- $N^3/6$ for the Cholesky factorization

A few facts

- The normal equations method costs too much.
- It squares the condition number
  which we will define later for $M \neq N$.
- It's very useful for small-scale hand calculations.

Examples: I

$N = 2,\ M = 3$

$$\mathbf{A} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}, \mathbf{A}^T\mathbf{A} = \begin{pmatrix} 14 & 32 \\ 32 & 77 \end{pmatrix}, \mathbf{A}^T\mathbf{b} = \begin{pmatrix} 46 \\ 109 \end{pmatrix}$$

Normal Equations

$$\begin{pmatrix} 14 & 32 \\ 32 & 77 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 46 \\ 109 \end{pmatrix}$$

Solution (How did I get this with no work?)

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$M = 3,\ N = 1$

Find the scalar x that minimizes

$$\|\mathbf{a}x - \mathbf{b}\|$$

Normal equations:

$$\|\mathbf{a}\|_2^2 x = \mathbf{a}^T\mathbf{b} \text{ and so } x = \mathbf{a}^T\mathbf{b}/\|\mathbf{a}\|_2^2$$

which says that $x\mathbf{a}$ is the projection of $\mathbf{b}$ onto $\mathbf{a}$.

QR decomposition: $M \geq N$

The QR decomposition of an $M \times N$ matrix $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{QR}$$

where

- $\mathbf{Q}$ is $M \times N$ with orthonormal columns and
- $\mathbf{R}$ is $N \times N$ upper triangular with positive diagonal.

In matlab

```
[Q,R]=qr(A,0);
```

The second argument 0 matters.

Solving Linear Least Squares Problems with QR

If $\mathbf{Q}$ is $M \times N$ with orthonormal columns, then

- $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, the $N \times N$ identity.
- $\mathbf{Q}\mathbf{Q}^T = \mathbf{P}$, the projection onto the column space of $\mathbf{Q}$ which is $Ran(\mathbf{A})$ if $\mathbf{A} = \mathbf{QR}$ is the QR decomposition of $\mathbf{A}$.

Claim: $\mathbf{x} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b}$ is the solution of the problem.
In matlab, the backslash does the right thing.

```
x = A\b;
```

even if $\mathbf{A}$ is not square.

QR solves the problem

We will work it out in terms of the normal equations.
$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ and $\mathbf{A} = \mathbf{Q}\mathbf{R}$ imply that

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

Since $\mathbf{R}$ is nonsingular and $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, we get

$$\begin{aligned} \mathbf{R}\mathbf{x} &= \mathbf{Q}^T\mathbf{b}, \text{ that's it} \\ \mathbf{x} &= \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b} \end{aligned}$$

Solving least squares problem with QR

Assumptions: $M \geq N$ and $\mathbf{A}$ has full column rank.

- Factor $\mathbf{A} = \mathbf{QR}$
- $\mathbf{c} \leftarrow \mathbf{Q}^T\mathbf{b}$
- Solve $\mathbf{Rx} = \mathbf{c}$

Some Matlab: What you get from qr

```
>> A=[1 2; 3 4; 5 6; 7 8]

A =

     1     2
     3     4
     5     6
     7     8

>> [q,r]=qr(A,0);
>> q

q =

  -1.0911e-01  -8.2952e-01
  -3.2733e-01  -4.3916e-01
  -5.4554e-01  -4.8795e-02
  -7.6376e-01   3.4157e-01

>> r

r =

  -9.1652e+00  -1.0911e+01
            0  -9.7590e-01
```

Q^TQ and QQ^T

```
>> q'*q

ans =

   1.0000e+00   2.4980e-16
   2.4980e-16   1.0000e+00

>> q*q'

ans =

   7.0000e-01   4.0000e-01   1.0000e-01  -2.0000e-01
   4.0000e-01   3.0000e-01   2.0000e-01   1.0000e-01
   1.0000e-01   2.0000e-01   3.0000e-01   4.0000e-01
  -2.0000e-01   1.0000e-01   4.0000e-01   7.0000e-01
```

Is $\mathbf{QQ}^T$ an orthogonal projector onto $Ran(\mathbf{A})$

We need $\mathbf{P} = \mathbf{QQ}^T = \mathbf{P}^2 = \mathbf{P}^T$ and $\mathbf{PA} = \mathbf{A}$.

```
>> p=q*q'; [norm(p*A-A), norm(p*p - p), norm(p'-p)]

ans =

   7.5261e-15   5.0915e-16             0
```

Proof of Projection Properties

We'll use $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_{N\times N}$ all over.

- $\mathbf{P}^2 = \mathbf{Q}(\mathbf{Q}^T\mathbf{Q})\mathbf{Q}^T = \mathbf{Q}\mathbf{Q}^T = \mathbf{P}$
- $\mathbf{P}^T = (\mathbf{Q}\mathbf{Q}^T)^T = \mathbf{Q}\mathbf{Q}^T = \mathbf{P}$
- $\mathbf{P}\mathbf{A} = \mathbf{P}\mathbf{Q}\mathbf{R} = \mathbf{Q}(\mathbf{Q}^T\mathbf{Q})\mathbf{R} = \mathbf{Q}\mathbf{R} = \mathbf{A}$

More on $\mathbf{QQ}^T$

If $\mathbf{x}^*$ is the solution of the least squares problem, then

$$\mathbf{Ax}^* = \mathbf{QQ}^T\mathbf{b}$$

because both $\mathbf{Ax}$ and $\mathbf{QQ}^T\mathbf{b}$ are the nearest points in $Ran(\mathbf{A})$ to $\mathbf{b}$. Algebraic argument:

$$\mathbf{Ax}^* = \mathbf{QRR}^{-1}\mathbf{Q}^T\mathbf{b} = \mathbf{QQ}^T\mathbf{b}.$$

A QR trick

- Type `help qr` at the matlab prompt. You'll be glad you did.
- We'll see that **Q** is tricky to store and can get large.
- What would you do if you only had **R**?
  `R = qr(A,0)` (sparse) or `R = triu(qr(A,0));`
  `R=R(1:N,:);`

Hint:

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}^T\mathbf{Q}^T\mathbf{Q}\mathbf{R} = \mathbf{A}^T\mathbf{A}.$$

So $\mathbf{R}^T$ is the Cholesky factor for the normal equations.

Combine QR with Normal equations

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

$$\mathbf{x} = \mathbf{R}^{-1}\mathbf{R}^{-T}\mathbf{A}^T\mathbf{b}.$$

- Good news:
  - No matrix-matrix multiply
  - Very effective if $\mathbf{A}$ is sparse because $\mathbf{Q}$ could be dense.
- Bad news: Conditioning of $\mathbf{A}^T\mathbf{A}$ did not get any better.

Experimenting

Lets solve $\min \|\mathbf{AX} - \mathbf{b}\|_2$ where

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Analytic solution via normal equations

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 84 & 100 \\ 100 & 120 \end{pmatrix} \text{ and } \mathbf{A}^T\mathbf{b} = \begin{pmatrix} 16 \\ 20 \end{pmatrix}$$

I can eyeball it and get $\mathbf{x} = (-1, 1)^T$.

Solution with QR

```
>> b=ones(4,1);
>> x1=A\b

x1 =

  -1.0000e+00
   1.0000e+00
```

Solution with R alone

```
>> R=triu(qr(A)); R=R(1:N,:)

R =

  -9.1652e+00  -1.0911e+01
            0  -9.7590e-01

>> R\(R'\(A'*b))

ans =

  -1.0000e+00
   1.0000e+00
```

There's a bit more to this coming later.

Cost of solve with QR: leading order

- Compute QR factorization (later): $N^2M - N^3/3$
- Multiply $\mathbf{b}$ with $\mathbf{Q}^T$: $O(MN)$
  The prefactor depends on how one stores $\mathbf{Q}$
- Solve $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$: $N^2/2$

One can do the factorization without storing $\mathbf{Q}$ as a full matrix.

The QR factorization is unique

Lemma: If $\mathbf{R}$ is upper triangular with positive diagonal, so is $\mathbf{R}^{-1}$.
proof: As you do the backsolve for $\mathbf{Ax} = \mathbf{b}$ you'll see that

$$x_i = b_i / r_{ii} - \left(\sum_{j=i+1}^{N} r_{ij} b_j \right) / r_{ii}$$

So $(\mathbf{A}^{-1}\mathbf{x})_i$ only depends on b_j for $j \geq i$, hence the inverse is upper triangular. The ith diagonal element of $\mathbf{R}^{-1}$ is $1/r_{ii} > 0$.

The QR factorization is unique: II

Suppose $\mathbf{A}$ has full column rank and

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

are two QR factorizations of $\mathbf{A}$. Then

$$(\mathbf{Q}')^T\mathbf{Q} = \mathbf{R}'\mathbf{R}^{-1}$$

So the orthogonal matrix $(\mathbf{Q}')^T\mathbf{Q}$ (products of orthogonal matrices are orthogonal) is upper triangular with positive diagonal. This means all the eigenvalues of $(\mathbf{Q}')^T\mathbf{Q}$ are 1 and $(\mathbf{Q}')^T\mathbf{Q}$ is diagonal. So $\mathbf{Q}' = \mathbf{Q}$ and $\mathbf{R}' = \mathbf{R}$.

QR by a sequence of orthogonal transformations

Suppose you have a sequence of $M \times M$ orthogonal matrices $\{\mathbf{Q}_i\}_{i=1}^{K}$ and

$$\mathbf{Q}_K \ldots \mathbf{Q}_1 \mathbf{A} = \mathbf{R}'$$

where $\mathbf{R}'$ is $M \times N$ upper triangular. Then

- the upper $N \times N$ block of $\mathbf{R}'$ is $\mathbf{R}$
- the left $M \times N$ block of

$$(\mathbf{Q}_K \ldots \mathbf{Q}_1)^T \text{ is } \mathbf{Q}.$$

Stability

Multiplication by orthogonal matrices is stable. The ℓ^2 norm does not grow. So

$$\|\mathbf{R}\|_2 = \|\mathbf{R}'\|_2 = \|\mathbf{Q}_K \ldots \mathbf{Q}_1 \mathbf{A}\|_2. = \|\mathbf{Q}^T \mathbf{A}\|_2 = \|\mathbf{A}\|_2.$$

For non-square $M \times N$ matrices

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|.$$

As we'll see, you'll rarely store the $\mathbf{Q}$s as full matrices.

Householder Reflections

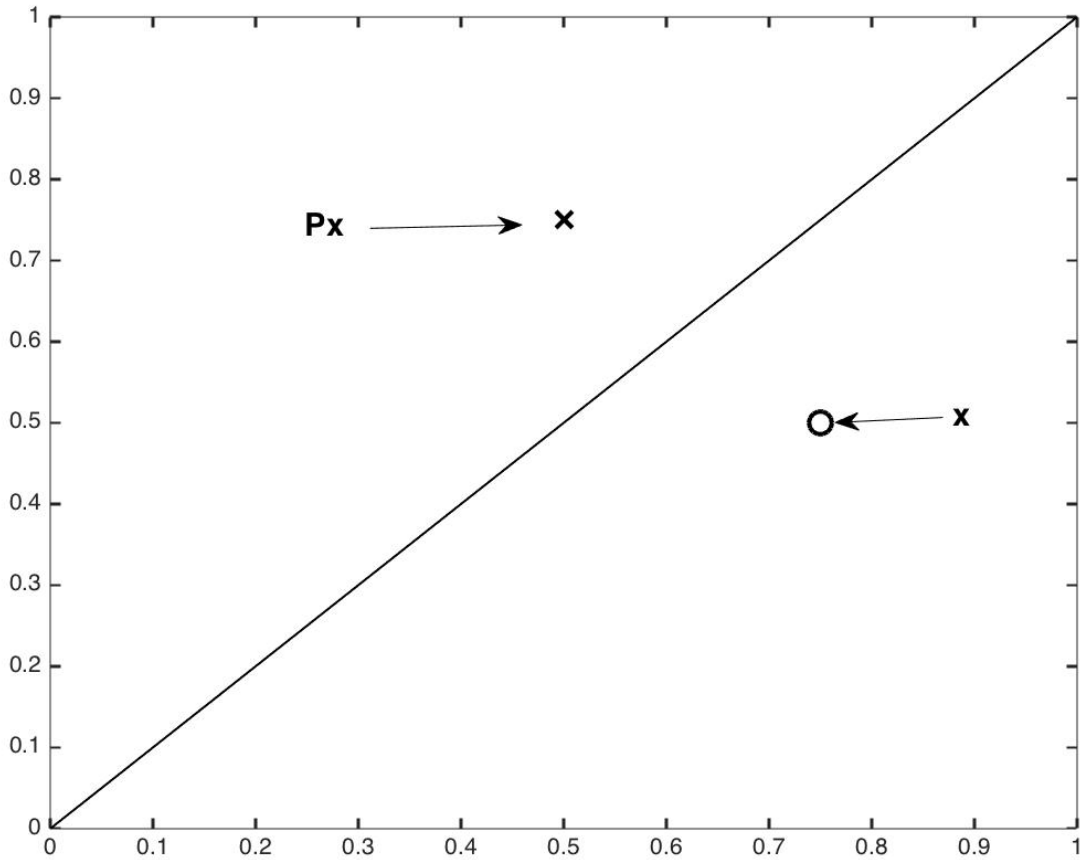
A **Householder Reflection** is

$$\mathbf{P_u} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T$$

where $\mathbf{u}$ is a unit vector.
Reflection means that $\mathbf{Px}$ is a reflection of $\mathbf{x}$ through the hyperplane

$$\mathbf{x}^\perp = \{\mathbf{z} \,|\, \mathbf{z}^T\mathbf{u} = 0\}$$

$u = (-1, 1)^T/\sqrt{2}$, $x = (1/2, 1)^t$



Using Householder Reflections in QR

Simple idea: Given $\mathbf{x} \in R^M$ find $\mathbf{u}$ so that

$$\mathbf{Px} = \begin{pmatrix} * \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

If we can do this, we can apply a sequence of Householders to transform $\mathbf{A}$ to upper triangular.

So what's $\mathbf{u}$?

What we need is

$$\mathbf{x} - 2\mathbf{u}(\mathbf{u}^T\mathbf{x}) = \mp\|\mathbf{x}\|_2\mathbf{e}_1$$

How do I know what the right side has to be?
So $\mathbf{u}$ is a linear combination of $\mathbf{x}$ and $\mathbf{e}_1$ and is parallel to

$$\mathbf{w} = \mathbf{x} \pm \|\mathbf{x}\|_2\mathbf{e}_1 \text{ which means } \mathbf{u} = \frac{\mathbf{x} \pm \|\mathbf{x}\|_2\mathbf{e}_1}{\|\mathbf{x} \pm \|\mathbf{x}\|_2\mathbf{e}_1\|}$$

How do I pick the $\pm$ sgn? Easy, don't subtract.

Formula for **u**

$$\mathbf{u} = \frac{\mathbf{x} + sign(x_1)\|\mathbf{x}\|_2 \mathbf{e}_1}{\|\mathbf{x} + sign(x_1)\|\mathbf{x}\|_2 \mathbf{e}_1\|}$$

where

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Cost: $3M$ to leading order.

Don't take my word for it.

Let $\mathbf{x} = \begin{pmatrix} 3 & 0 & 4 \end{pmatrix}^T$. The formula says

$$\mathbf{w} = \mathbf{x} + sign(x_1)\|\mathbf{x}\|_2\mathbf{e}_1 = \mathbf{x} + 5\mathbf{e}_1 = \begin{pmatrix} 8 \\ 0 \\ 4 \end{pmatrix}$$

and $\mathbf{u} = \mathbf{w}/\|\mathbf{w}\| = \mathbf{w}/\sqrt{80}$.

Keep cranking . . .

$$\mathbf{P_u} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T = \frac{1}{10}\begin{pmatrix} -6 & 0 & -8 \\ 0 & 1 & 0 \\ -8 & 0 & 6 \end{pmatrix}$$

and you can check that

$$\mathbf{P_u x} = \begin{pmatrix} -5 \\ 0 \\ 0 \end{pmatrix}.$$

Applying $\mathbf{P_u}$ to a vector

DO NOT FORM THE MATRIX FOR $\mathbf{P_u}$!!!

Rather: to compute $\mathbf{z} = \mathbf{P_u x}$

- Compute $\mathbf{u}^T\mathbf{x}$
- Then $\mathbf{z} = \mathbf{x} - 2(\mathbf{u}^T\mathbf{x})\mathbf{u}$

Total cost: $2M$ to leading order.

Zeroing out part of a vector

Suppose you want $\mathbf{u}$ so that

$$(\mathbf{z})_i = (\mathbf{P_u x})_i = 0 \text{ for } M \leq i \leq k+1$$

and you don't care about the upper k entries of $\mathbf{z}$.
Then

- you only need to compute and store a vector of size $M-k+1$
- and apply the reflection to the lower $M-k+1$ entries of $\mathbf{x}$.
- Cost of $\mathbf{P_u}$ matrix multiply $= 2(M-k)$ to leading order.

Example

Objective: zero out the last entry in $\mathbf{x} = \begin{pmatrix} 1 & 3 & 4 \end{pmatrix}^T$.
Process:

- Find a vector $\mathbf{z} \in R^2$ so that

$$\mathbf{P_z} \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} \pm - 5 \\ 0 \end{pmatrix}$$

- Then $\mathbf{u} = \begin{pmatrix} 0 & 0 & \dots & \mathbf{z}^T \end{pmatrix}^T$ (if you need $\mathbf{u}$)
- To do a $\mathbf{P_u}$ matrix multiply you'll only need $\mathbf{z}$.

Example Continued

For this example, let $\mathbf{x} = \begin{pmatrix} 3 & 4 \end{pmatrix}^T$,

$$\mathbf{z} = \frac{\mathbf{x} + 5\mathbf{e}_1}{\sqrt{80}} = \frac{1}{\sqrt{80}} \begin{pmatrix} 8 \\ 4 \end{pmatrix}$$

and

$$\mathbf{P}_z = \mathbf{I} - \begin{pmatrix} -.6 & -.8 \\ -.8 & .6 \end{pmatrix}$$

Padding with zeros

So

$$\mathbf{u} = \begin{pmatrix} 0 \\ z_1 \\ z_2 \end{pmatrix} \text{ and } \mathbf{P_u} = \mathbf{I} - \begin{pmatrix} 0 & 0 & 0 \\ 0 & -.6 & -.8 \\ 0 & -.8 & .6 \end{pmatrix}$$

QR with Householders: **A** $M \times N$, full rank

for $j = 1 : N$ **do**
  Compute $\mathbf{u} \in R^{M-j+1}$ so that column $\mathbf{P_u a}_j$ has zeros in rows $j+1 \ldots N$.
  Apply the Householder to **A**, keeping in mind that you don't have to touch rows $1, \ldots, j-1$.
  Store **u**
end for

At the end, you've overwritten the upper triangle of **A** with **R**.

What about the positive diagonal?

You may not get a positive diagonal at the end, but . . .

- That's does not matter for the solvers.
- It's easy to fix by replacing

$$\mathbf{QR} \text{ with } (\mathbf{QD})(\mathbf{DR})$$

where $\mathbf{D}$ is $N \times N$ diagonal with ± 1 on the diagonal.

Cost for Factorization

For each of the N columns you'll need to

- Compute $\mathbf{u} \in R^{M-j+1}$ for $3(M-j+1)$ multiplies.
- Inner loop:
  - Apply the Householder to each column of a matrix of size $(N-j)(M-j+1)$. Cost of $2(M-j+1)$ for each of the $N-j$ columns.

To leading order

$$\int_1^N 2(N-j)(M-j+1)\, dj = N^2M - N^3/3 + \text{lower order}$$

So if $N = M$, it's twice an LU.

Multiplying $\mathbf{b}$ by $\mathbf{Q}^T$

If

$$\mathbf{Q}^T = \mathbf{P}_1 \dots \mathbf{P}_N$$

where $\mathbf{P}_i = \mathbf{I} - 2\mathbf{u}_i\mathbf{u}_i^T$ and $\mathbf{u}_i$ has i nonzeros. Then

$$\mathbf{P}^T\mathbf{b} = \mathbf{P}_N \dots \mathbf{P}_1\mathbf{b}$$

Cost

$$\int_1^N 2(M - j + 1)\, dj = 2MN - N^2.$$

Cost of Solve

- Multiplication with $\mathbf{Q}^T$.
  - Multiply $\mathbf{b}$ with the $\mathbf{P_u}$'s in the reverse order
- Trigangular solve with $\mathbf{R}$: $N^2/2$

QR as a linear solver. $M = N$

- Cost $= 2N^3/3$
- Double GEPP
- Not worth the expense.

Bottom Line on QR with Householders

- Going through this is one way to prove QR factorization exists.
- Storage: Same as **A** plus one vector of size N.
- Householders process one column at a time.
- Cost: $MN^2 - N^3/3$ (factorization); $2MN - N^2/2$ (solve)
- Stability: no prolblem
- Matlab: It's in the backslash command.

Givens Rotations

A 2×2 **Givens rotation** is a matrix of the form

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

where $c = \cos(\theta)$, $s = \sin(\theta)$ for $\theta \in [-\pi, \pi]$.

- **G** is orthogonal
- **G rotates** the unit vector $\begin{pmatrix} c & -s \end{pmatrix}^T$ which makes an angle $-\theta$ through an angle θ so that the y-component is 0

Why would we ever want such a thing?

$$\mathbf{G}\begin{pmatrix} c \\ -s \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

An $M \times M$ Givens rotation replaces row and column i and j in the $M \times M$ identity with the 2×2 Givens rotation in row i and column j.

$M \times M$ Givens rotation example

Here $j = i + 1$

$$\mathbf{G}(i, i+1, \theta) = \begin{pmatrix} 1 & 0 & & \dots & & & 0 \\ 0 & \ddots & \ddots & & & & \\ & \ddots & c & -s & & & \\ \vdots & & s & c & 0 & & \vdots \\ & & & 0 & 1 & \ddots & \\ & & & & \ddots & \ddots & 0 \\ 0 & & & \dots & & 0 & 1 \end{pmatrix}.$$

4×4 example: $i = 2$, $j = 4$

$$\mathbf{G}(2,4,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & 0 & -s \\ 0 & 0 & 1 & 0 \\ 0 & s & 0 & c \end{pmatrix}$$

Using Givens in QR

To rotate

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ into } \begin{pmatrix} \sqrt{x_1^2 + x_2^2} \\ 0 \end{pmatrix}$$

we will need

$$c = x_1/\sqrt{x_1^2 + x_2^2} \text{ and } s = -x_2/\sqrt{x_1^2 + x_2^2}$$

Never take my word for anything.

$$\mathbf{Gx} = \begin{pmatrix} cx_1 - sx_2 \\ sx_1 + cx_2 \end{pmatrix} = \begin{pmatrix} \frac{x_1^2+x_2^2}{\sqrt{x_1^2+x_2^2}} \\ \frac{-x_1x_2+x_1x_2}{\sqrt{x_1^2+x_2^2}} \end{pmatrix} = \begin{pmatrix} \sqrt{x_1^2+x_2^2} \\ 0 \end{pmatrix}$$

as asserted.

Sparse QR

Suppose **A** has only a few zeros in column 1, say in row 1 (which is ok) and rows $1 < k < l < m$.

- Apply $\mathbf{G}(1, m, \theta)$ to zero out $a_{m,1}$.
- Apply $\mathbf{G}(1, l, \theta)$ to zero out $a_{l,1}$.
- Apply $\mathbf{G}(1, k, \theta)$ to zero out $a_{k,1}$.

So you have a sparse QR using orthogonal transformations.
Note: cost is application of **G** to every column and the appropriate two rows.

Cost of Givens

Move along the columns in QR

- Elimination of a row in column j requires application of one rotation for each non-zero at a cost per column of
  - 2 to compute c and s and populate the rotation (outer loop) + and * do not balance, but a count of 2 is close enough
  - 4 to do the matrix vector product (inner loop)
- $4(M-j+1)$ for each nonzero below row j in column j.

If the **A** is dense, then the total is, to leading order,

$$\int_1^N 4(N-j)(M-j+1)\, dj = 2(N^2M - N^3/3)$$

which is twice Householder.

When is Givens happy?

Answer, where there are very few zeros below the diagonal.
Simple example:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 0 & 3 \\ 1 & 0 \end{pmatrix}$$

So set $c = -s = 1\sqrt{2}$ and then

$$\mathbf{A}_1 = \mathbf{GA} = \begin{pmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{pmatrix} \mathbf{A} = \begin{pmatrix} \sqrt{2} & \sqrt{2} \\ 0 & 3 \\ 0 & -\sqrt{2} \end{pmatrix}$$

So we've added a zero in the next column. Gotta fix that.

Killing the last non-zero

So now we will use $i = 2$ and $j = 3$, to avoid messing with row 1.

$$c_2 = 3/\sqrt{11}; s_2 = \sqrt{2}/\sqrt{11}$$

so

$$\mathbf{A}_2 = \mathbf{G}_2\mathbf{A}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_2 & -s_2 \\ 0 & s_2 & c_2 \end{pmatrix} \mathbf{A} = \begin{pmatrix} \sqrt{2} & \sqrt{2} \\ 0 & 3.3166 \\ 0 & 0 \end{pmatrix}$$

and we did our job.

A sparse QR code is hard to write

You'll have to

- Keep track of fill-in in the upcoming columns
- Reorder rows/columns to minimize that fill-in
- Carefully store the c's, s's, and the history of the row/column interchanges

Bottom line: use tools! Do not try this at home.

But you could do this at home.

Suppose **A** is $N+1 \times N$ **upper Hessenberg**. That means

$$a_{ij} = 0 \text{ if } i > j+1.$$

Only the subdiagonal has non-zeros. Like this matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 0 & 5 \end{pmatrix}$$

Upper Hessenberg QR with Givens

for j=1:N **do**
  Apply $G(j-1, j, \theta)$ to zero out $a_{j,j+1}$
end for

The application of $G(j-1, j)$ to **A** does not affect the previous columns because you'll only be multiplying by zero.

Cost of Upper Hessenberg QR with Givens

At column j you

- Build $\mathbf{G}$ for cost of 2 (outer loop)
- Apply $\mathbf{G}$ to rows j and $j+1$ for all columns $j \ldots N$. Cost $= 4$ per column

To leading order

$$\int_1^N 4(N-j)\, dj = 2N^2$$

Example

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 0 & 5 \end{pmatrix}$$

We'll zero out $a_{1,2} = 3$ first.

$$c_1 = 1/\sqrt{10}; s_1 = -3/sqrt10;$$

$$\mathbf{A}^1 = \mathbf{G}_1\mathbf{A} = \begin{pmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{A} = \begin{pmatrix} \sqrt{10} & 1.4/\sqrt{10} \\ 0 & -\sqrt{.4} \\ 0 & 5 \end{pmatrix}$$

Now it's time to zero out $a_{2,3} = 5$

$$c_2 = -.12549; s_2 = -.99209$$

and we have it

$$\mathbf{G}_2\mathbf{A}_1 = \begin{pmatrix} 3.1623 & 4.4272 \\ 0 & 5.0398 \\ 0 & 0 \end{pmatrix}$$

Bottom Line on Givens

- Designed for sparse problems
- $2\times$ cost of Householder for dense problems
- The method of choice for upper Hessenberg

Gram-Schmidt Orthogonalization

We will do a QR factorization by brute force.

for $j = 1 : N$ **do**
  $\mathbf{w} = \mathbf{a}_j$
  for $i = 1 : j-1$ **do**
    $r_{ij} = \mathbf{q}_i^T \mathbf{a}_j$
  end for
  for $i = 1 : j-1$ **do**
    $\mathbf{w} = \mathbf{w} - r_{ij}\mathbf{q}_i$
  end for
  $r_{jj} = \|\mathbf{w}\|_2; \mathbf{q}_j = \mathbf{w}/\|\mathbf{w}\|_2$
end for

I know you don't believe me. Here's an example.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$$

So

$$\mathbf{q}_1 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \text{ and } r_{11} = \sqrt{3}$$

There's more

So

$$r_{12} = \mathbf{a}_2^T \mathbf{q}_1 = \frac{9}{\sqrt{3}}$$

and

$$\mathbf{w} = \mathbf{a}_2 - r_{12}\mathbf{q}_1 = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} - \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

So $r_{22} = \sqrt{2}$ and $\mathbf{q}_2 = \mathbf{w}/\sqrt{2}$

Bottom Line

$$\mathbf{Q} = \begin{pmatrix} \frac{1}{\sqrt{3}} & -1/\sqrt{2} \\ \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{3}} & 1/\sqrt{2} \end{pmatrix} \text{ and } \mathbf{R} = \begin{pmatrix} \sqrt{3} & 3\sqrt{3} \\ 0 & \sqrt{2} \end{pmatrix}$$

So (check it out) $\mathbf{QR} = \mathbf{A}$.

GS is the way to go when computing by hand.

Some Matlab

We just did classical Gram-Schmidt

```
function [q,r]=gram_easy(a)
% GRAM_EASY compute QR via Gram-Schmidt orthogonalization
% Classical version with loops
%
[m,n]=size(a);
%
% Store q and r. I'm storing r as a full matrix.
%
q=zeros(m,n); r=zeros(n,n);
for j=1:n
    w=a(:,j);
    for i=1:j-1
        r(i,j)=a(:,j)'*q(:,i);
    end
    for i=1:j-1
        w=w-r(i,j)*q(:,i);
    end
    r(j,j)=norm(w);
    q(:,j)=w/norm(w);
end
```

We can vectorize this to death.

- For each j, the scalar products $\mathbf{a}_j^T \mathbf{q}_i$ can be done in parallel.
- So we can replace the first `for` loop with one line:

```
r(1:j-1,j)=q(:,1:j-1)'*a(:,j);
```

- Same story with the second loop

```
w=a(:,j) - q(:,1:j-1)*r(1:j-1,j);
```

Let's think about this.

Before and after; first loop

Before:

```
for i=1:j-1
    r(i,j)=a(:,j)'*q(:,i);
end
```

After:

```
r(1:j-1,j)=q(:,1:j-1)'*a(:,j);
```

Before and after; second loop

Before:

```
w=a(:,j);
for i=1:j-1
    w=w-r(i,j)*q(:,i);
end
```

After:

```
w = a(:,j) - q(:,1:j-1)*r(1:j-1,j);
```

Vectorized Classical Gram-Schmidt (CGS)

```
function [q,r]=class_gram(a)
% CLASS_GRAM compute QR via Gram-Schmidt
    orthogonalization
% Vectorized classical version
%
%function [q,r]=class_gram(a)
[m,n]=size(a);
% Store q and r. I'm storing r as a full matrix.
q=zeros(m,n); r=zeros(n,n);
for j=1:n
    r(1:j-1,j)=q(:,1:j-1)'*a(:,j);
    w = a(:,j) - q(:,1:j-1)*r(1:j-1,j);
    r(j,j)=norm(w);
    q(:,j)=w/norm(w);
end
```

Advantages of CGS

- Easy to vectorize/parallelize
- MATLAB automatically uses your multi-core computer correctly
- If the vectors are long and there are a lot of them, you get almost perfect parallelism.

There's only one tiny problem ...

Stability of CGS

Here's a matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 10^{-7} & 10^{-7} & 0 \\ 10^{-7} & 0 & 10^{-7} \end{pmatrix}$$

As Matlab for QR, look at $\|\mathbf{Q}^T\mathbf{Q} - \mathbf{I}\|$ and $\|\mathbf{QQ} - \mathbf{A}\|$ and get

```
>> [qm,rm]=qr(A);
>> [norm(qm*rm-A) norm(qm'*qm - eye(3))]
ans =
   3.8459e-16   4.4409e-16
```

On the other hand . . .

Let's give CGS a shot!

```
>> [qc,rc]=class_gram(A);
>> [norm(qc'*qc-eye(3)) norm(qc*rc-A)]
ans =
   7.9928e-04                0
```

That's bad. What happened?

What about Q?

```
>> qc
qc =

    1.0000e+00    9.9920e-08    9.9920e-08
    1.0000e-07    9.9262e-15   -1.0000e+00
    1.0000e-07   -1.0000e+00   -7.9928e-04
```

This is not an orthogonal matrix.

You're dead, of course

```
>> b = A*ones(3,1);
>> [rm\(qm'*b) rc\(qc'*b)]
ans =
   1.0000e+00   1.0056e+00
   1.0000e+00   9.9840e-01
   1.0000e+00   9.9600e-01
>> [norm(rm\(qm'*b) - ones(3,1)) norm(rc\(qc'*b) - ones(3,1))]
ans =
   9.2222e-16   7.0632e-03
```

You should expect this, since $\mathbf{Q}^T$ is far from $\mathbf{Q}^{-1}$.

What happened?

- CGS is not QR via a sequence of orthogonal transformations,
- unlike Householder or Givens,
- and it's unstable.
  - You see the instability by **loss of orthogonality**
  - **Tremble in fear!**
  - $\mathbf{x} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b}$ is wrong because $\mathbf{Q}^T \neq \mathbf{Q}^{-1}$.
- How can we fix this mess?

Gram-Schmidt Orthogonalization Twice

for $j = 1 : N$ **do**
  $\mathbf{w} = \mathbf{a}_j$
  for $i = 1 : j-1$ **do**
    $r_{ij} = \mathbf{q}_i^T \mathbf{w}$
  end for
  for $i = 1 : j-1$ **do**
    $\mathbf{w} = \mathbf{w} - r_{ij}\mathbf{q}_i$
  end for
  for $i = 1 : j-1$ **do**
    $r_{ij}^1 = \mathbf{q}_i^T \mathbf{w}$
  end for
  for $i = 1 : j-1$ **do**
    $\mathbf{w} = \mathbf{w} - r_{ij}^1\mathbf{q}_i$
  end for
  for $i = 1 : j-1$ **do**
    $r_{ij} = r_{ij} + r_{ij}^1$
  end for
  $r_{jj} = \|\mathbf{w}\|_2; \mathbf{q}_j = \mathbf{w}/\|\mathbf{w}\|_2$
end for

CGS2 in Matlab

```
function [q,r]=cgs2(a)
% CGS2 compute QR via Gram-Schmidt orthogonalization twice
% Vectorized classical version; reorthogonalize!
%
%function [q,r]=cgs2(a)
%
[m,n]=size(a);
%
% Store q and r. I'm storing r as a full matrix.
%
q=zeros(m,n); r=zeros(n,n); r1=zeros(n,n);
for j=1:n
    r(1:j-1,j)=q(:,1:j-1)'*a(:,j);
    w = a(:,j) - q(:,1:j-1)*r(1:j-1,j);
%   one more time
    r1(1:j-1,j)=q(:,1:j-1)'*w;
    w = w - q(:,1:j-1)*r1(1:j-1,j);
    r(1:j-1,:)= r(1:j-1,:) + r1(1:j-1,:);
    r(j,j)=norm(w);
    q(:,j)=w/norm(w);
end
```

Testing . . .

```
>> [q2,r2]=cgs2(A);
>> [norm(q2'*q2-eye(3)) norm(q2*r2-A) norm(r2\(q2'*b) - ones(3,1))]

ans =

   2.2204e-16            0   1.0175e-15
```

Why I love CGS2

- Cures the stability problem.
- Twice the floating point work.
  But that's not an issue if you have a two-core computer.
- Easy to code. Easy to understand.
- Great if M and N are large.

But Tim, what if I have a single-core computer?

Modified Gram-Schmidt (MGS)

Here's a somewhat more stable version of GS.

for $j = 1 : N$ **do**
  $\mathbf{w} = \mathbf{a}_j$
  for $i = 1 : j-1$ **do**
    $r_{ij} = \mathbf{q}_i^T \mathbf{w}$
    $\mathbf{w} = \mathbf{w} - r_{ij}\mathbf{q}_i$
  end for
  $r_{jj} = \|\mathbf{w}\|_2; \mathbf{q}_j = \mathbf{w}/\|\mathbf{w}\|_2$
end for

Can you see the difference? You should verify that MGS is equivalent to CGS in exact arithmetic.

More on MGS

- Equivalent to CGS in exact arithmetic.
- I defy you to vectorize this algorithm.
- More stable than CGS, but it does not solve all the problems.

Matlab for MGS

```
function [q,r]=mgs(a)
% MGS compute QR via modified Gram-Schmidt
%function [q,r]=mgs(a)
%
[m,n]=size(a);
% Store q and r. I'm storing r as a full matrix.
q=zeros(m,n); r=zeros(n,n);
for j=1:n
    w=a(:,j);
    for i=1:j-1
        r(i,j)=q(:,i)'*w;
        w = w - r(i,j)*q(:,i);
    end
    r(j,j)=norm(w); q(:,j)=w/norm(w);
end
```

Let's test with the problematic matrix.

```
>> [qmg,rmg]=mgs(A);
>> [norm(qmg'*qmg-eye(3)) norm(qmg*rmg-A) norm(rmg\(qmg'*b) - ones(3,1)
>> ans =
   1.1304e-10              0    5.8735e-03
```

Wow! $\mathbf{Q}' * \mathbf{Q}$ is much better, but the answer still stinks!

One last trick.

We can test for loss of orthogonality. If, at column j,

$$\|\mathbf{a}_j\|_2 + \delta\|\mathbf{w}\|_2 = \|\mathbf{a}_j\|_2$$

to working precision, then we reorthogonalize.

- We use $\delta = 10^{-3}$ later in the course. Does no good here.
- Use $\delta = 10^{-9}$ and reorthogonalize 2 of 3 times. Works.
- No guarantees!!

Bottom line on Reorthogonalization

Method	$\|Q^TQ - I\|$	$\|QR - A\|$	$\|R^{-1}Q^Tb - x^*\|$
Matlab	4.44e-16	3.85e-16	9.22e-16
CGS	7.99e-04	0.00e+00	7.06e-03
CGS2	2.22e-16	0.00e+00	1.02e-15
MGS	1.13e-10	0.00e+00	5.87e-03
MGS2	2.22e-16	0.00e+00	1.02e-15

Tim likes CGS2.

Geometry

If $\mathbf{A}$ is $M \times N$, $M \geq N$, full column rank.

- $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b}$ is the unique solution of

$$\min \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$$

- It's the matlab backslash

```
x = A\b;
```

- $\mathbf{Q}\mathbf{Q}^T$ is the orthogonal projection onto $Ran(\mathbf{A})$
- Distance of $\mathbf{b}$ from $Ran(\mathbf{A})$ is $\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{b}\|$

Bottom line on QR

- Don't write a QR code at home.
- Householder for dense, Givens for sparse.
- Gram-Schmidt for hand computations.
  - Also good when you get one column of **A** at a time.
  - Watch out for loss of orthogonality.

What you should remember how to do.

- By hand:
  - Build small examples of Householder and Givens
  - Solve small problems
  - Do proofs of fewer than ten lines.
- On the job
  - Know how to communicate with the tools.
  - Know what you're getting at the end.

Questions?

What is QR for

$$\mathbf{A} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \mathbf{A} = 3, \mathbf{A} = -3, \mathbf{A} = 1, \mathbf{A} = \mathbf{I}$$