Full QR factorization: $A = Q\ R$ ($m\times n$ = $m\times m$ $m\times n$)

From last time:

$$H_2 H_1 A = R$$

singular...

$$A = H_1^{-1} H_2^{-1} R = H_1^T H_2^T R = H_1 H_2 R = QR$$

where $Q = \underbrace{H_1 H_2}$. This is 2 H's so $2m$ parameters...

$H_1 = I - 2v_1 v_1^T$

$H_2 = I - 2v_2 v_2^T$

In general $H \in \mathbb{R}^{m\times m}$ but actually since $v \in \mathbb{R}^m$ there are only m parameters...
actually v has $m-1$ free parameters since $\|v\| = 1$.

Solve equations... $Ax = b$

since Q has orthonormal columns then $Q^TQ = I$.

$$QRx = b$$
$$Q^TQRx = Q^Tb$$
$$Rx = Q^Tb$$

- Recall each reflection H_j cleared out a column of A so in general we need n reflections when $A \in \mathbb{R}^{m\times n}$.

$Q = H_1 H_2 \cdots H_n$ $\quad H_n \cdots H_2 H_1$

$$Q^T = (H_1 H_2 \cdots H_n)^T = H_n^T \cdots H_2^T H_1^T = H_n \cdots H_2 H_1$$

$$Q^Tb = H_n \cdots H_2 H_1 b = \underbrace{(I - 2v_n v_n^T) \cdots (I - 2v_2 v_2^T)(I - 2v_1 v_1^T) b}$$

Weirdly ... expanding this is more efficient

$$\underbrace{(I - 2v_1 v_1^T) b} = b - 2v_1 v_1^T b = b - 2v_1 (v_1 \cdot b)$$

this look like matrix mult by an $m\times m$ matrix

m dot product to make $m\times m$ matrix-vector mult.

$m \cdot m$ operations

vector difference (m)

dot product (m)

total $2m$ operation

In the end have to do H_2, H_3 and the others this way too!

Real total $2mn$

The difference between mn^2 and $2mn$ becomes important in applications where m is in the millions (billions) and n is 3 or 4.

Example the lab on Friday... $m=100$ and $n=4$.

The eigenvalue eigenvector problem... Assume $A^T=A$ ...

guarantees real eigenvalues..

Algorithm 27.1. Power Iteration

$v^{(0)}$ = some vector with $\|v^{(0)}\| = 1$

for $k = 1, 2, \ldots$

$w = Av^{(k-1)}$	apply A
$v^{(k)} = w/\|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^T Av^{(k)}$	Rayleigh quotient

```
julia> A=rand(6,6)
6×6 Matrix{Float64}:
 0.338898  0.65032   0.00550222  0.353504  0.477489  0.514907
 0.852617  0.434508  0.0465316   0.15887   0.872625  0.885462
 0.838902  0.691429  0.0877167   0.154086  0.207913  0.200002
 0.585725  0.783279  0.776208    0.424348  0.841904  0.239073
 0.65027   0.296483  0.0680904   0.689072  0.737583  0.0584512
 0.55774   0.111011  0.879008    0.100965  0.411938  0.225105

julia> A=A'*A
6×6 Matrix{Float64}:
 2.62256  1.8844    1.1043    1.13747   2.28276  1.40083
 1.8844   1.80354   0.810201  0.953349  1.7573   1.08746
 1.1043   0.810201  1.38968   0.487904  1.12728  0.448998
 1.13747  0.953349  0.487904  0.839033  1.24656  0.517967
 2.28276  1.7573    1.12728   1.24656   2.45522  1.39724
 1.40083  1.08746   0.448998  0.517967  1.39724  1.20042
```

← actually positive definite as well as symmetric

```
julia> v=rand(6)
6-element Vector{Float64}:
 0.13935549535585556
 0.8976829826737764
 0.6946155192780799
 0.9684097413726618
 0.8212233186518634
 0.1479496110717311
```

← start with a random vector

```
julia> vn=v
       for n=1:5
           wn=A*vn
           vn=wn/norm(wn)
           println("lambda = ",vn'*A*vn)
       end
lambda = 8.233308471278532
lambda = 8.248336628685696
lambda = 8.248456717034882
lambda = 8.248458067529066
lambda = 8.248458084428368
```

Iterate taking powers of A.

```
julia> eigvals(A)
6-element Vector{Float64}:
 0.022633465641616772
 0.2523364390461248
 0.36362830365930937
 0.48222254064090575
 0.9411747804970814
 8.248458084648965
```

converged to the largest eigenvalue of A.