Deflation...

Eigenvector - Eigenvalue problem...

Assumption. We've already found one eigenvector and the corresponding eigenvalue and want to find another pair.

eigenvector

eigenvalue

$$Ax = \lambda x \quad \text{for some } x \text{ and } \lambda$$

assume $\|x\| = 1$.

so $c = 1$.

Want to use a Householder reflector to map x into e_1

$$v = \frac{x - e_1}{\|x - e_1\|} \qquad H = I - 2vv^T$$

Claim $Hx = e_1$ (obvious because that's what we derived a month ago)

Note that since H is a reflection then $H^2 = I$

$$H^2 x = He_1 \quad \text{so} \quad \boxed{He_1 = x}$$

Check:

$$He_1 = (I - 2vv^T)e_1 = e_1 - 2\underbrace{\frac{x-e_1}{\|x-e_1\|}\left(\frac{x-e_1}{\|x-e_1\|}\right)^T}e_1$$

denominator is

$$\|x - e_1\|^2 = (x-e_1)\cdot(x-e_1) = x\cdot x - \underbrace{x\cdot e_1 - e_1\cdot x} + e_1\cdot e_1$$

$$\|x-e_1\|^2 = 1 - 2x\cdot e_1 + 1 = \cancel{2}(1 - x\cdot e_1)$$

$$He_1 = e_1 - \frac{(x-e_1)(x-e_1)^T e_1}{1 - x\cdot e_1}$$

$$= e_1 - \frac{(x-e_1)\cancel{(x\cdot e_1 - 1)}^{-1}}{\cancel{1-x\cdot e_1}} = \cancel{e_1} + x - \cancel{e_1} = x$$

What to do with H?

$$\boxed{\begin{array}{ll} He_1 = x & \\ Hx = e_1 & Ax = \lambda x \end{array}}$$

HAH^T ← has the same eigenvalues as A. but different eigenvectors

Note the spectral mapping theorem used last week preserved the eigen vectors but changed the eigenvalues.

$$HAH^{-1}e_1 = HAHe_1 = HAx = H\lambda x = \lambda Hx = \lambda e_1$$

note this string of equalities is also the explanation why the eigenvalues stay the same for similarity transform

Thus $HAH^{-1}e_1 = \lambda e_1$ and e_1 is an eigenvector of the matrix HAH^{-1}.

$$HAH^{-1} = HAH^{-1}I = HAH^{-1}\left[e_1 \middle| e_2 \middle| \cdots \middle| e_n\right]$$

$$= \left[\, HAH^{-1}e_1 \;\middle|\; HAH^{-1}e_2 \;\middle|\; \cdots \;\middle|\; HAH^{-1}e_n \,\right]$$

$$= \left[\, \lambda e_1 \;\middle|\; \text{stuff} \,\right] = \left[\begin{array}{c|c} \lambda & b^T \\ \hline 0 & B \end{array}\right]$$

- Since this is block diagonal, the eigenvalues of HAH^{-1} are given by λ plus the eigenvalues of B.
- Since HAH^{-1} is a similarity, then the eigenvalues of HAH^{-1} are the same as A.

Try the deflation idea out numerically...

```
julia> using LinearAlgebra
```

← we'll use this library later

```
julia> A=rand(5,5) .- 0.5
```

← Create a test matrix

```
5×5 Matrix{Float64}:
 -0.341857   0.041353    0.442328  -0.428871   0.253869
  0.469352   0.281581    0.461852   0.178746   0.0226348
  0.245477   0.0779004   0.102411   0.191063  -0.168007
 -0.293541  -0.287088   -0.256939   0.198179  -0.482268
  0.459791   0.41545     0.285071  -0.337974  -0.333454

julia> x0=rand(5)
```

← random starting vector for the power method

```
5-element Vector{Float64}:
 0.05089159757780837
 0.9276924442873629
 0.3217987911832698
 0.27976382464703464
 0.18882214600236114
```

```
julia> x=x0
       for n=1:100
           y=A*x
           x=y/norm(y)
       end
```

Power method to find an eigenvector

```
julia> x
5-element Vector{Float64}:
  0.45490077231561543
 -0.028869485787564746
 -0.2254325492964925
 -0.3610300097680237
 -0.7817092412794024
```

the approximate eigenvector

Now we create the Householder transform H which deflates A using this eigenvector...

$$v = \frac{x - e_1}{\|x - e_1\|} \qquad H = I - 2vv^T$$

recall this from above

```
julia> e1=[1,0,0,0,0]
5-element Vector{Int64}:
 1
 0
 0
 0
 0
```

There must be better ways to make the standard basis vectors.

If we were writing a loop to do deflation, this could be coded like we did in our QR factorization routine

```
julia> v=(x-e1)/norm(x-e1)
5-element Vector{Float64}:
 -0.5220628447248398
 -0.027649435388167477
 -0.21590556728405913
 -0.3457725572850218
 -0.748673506626786
```

The unit vector v

The Householder reflector.
Note we wrote this as a matrix rather than keeping it in terms of v

```
julia> H=I - 2*v*v'
5×5 Matrix{Float64}:
  0.454901   -0.0288695  -0.225433   -0.36103    -0.781709
 -0.0288695   0.998471   -0.0119393  -0.0191208  -0.0414008
 -0.225433   -0.0119393   0.90677    -0.149308   -0.323286
 -0.36103    -0.0191208  -0.149308    0.760883   -0.517742
 -0.781709   -0.0414008  -0.323286   -0.517742   -0.121024
```

```
julia> HAH=H*A*H
5×5 Matrix{Float64}:
 -0.659561      -0.237617    -0.0404357   -0.171031     0.32689
 -4.64348e-6     0.24284      0.249135    -0.107318    -0.622457
 -1.24532e-6    -0.0435133   -0.144424     0.222133    -0.301417
  2.04495e-6    -0.445165    -0.368262     0.702738     0.288816
  1.19253e-6     0.0151674   -0.408974     0.0290171   -0.234732
```

← The block diagonal matrix

Supposed to be zero

The deflated matrix B...

Yikes! The stuff that was supposed to be zero is not very close to zero. Sometimes the power method doesn't converge very fast. Remember the rate of convergence is governed by the ratio

$$\frac{|\text{second largest eigenvalue}|}{|\text{largest eigenvalue}|}.$$

If this ratio is close to 1, then the power method will converge more slowly. The rate of convergence can (and should) be increased using shifted inverse iteration. Instead of that, let's just use more iterations so these notes don't differ very much from what we did in class.

```
julia> x=x0
       for n=1:10000
           y=A*x
           x=y/norm(y)
       end
```

← make super sure x is a good approximation

Now repeat the same steps to try deflating A using this better eigenvector...

```
julia> v=(x-e1)/norm(x-e1)
5-element Vector{Float64}:
 -0.5220627079879024
 -0.027645745563189186
 -0.2159037973128917
 -0.3457719826733547
 -0.7486745140477299

julia> H=I - 2*v*v'
5×5 Matrix{Float64}:
  0.454901    -0.0288656  -0.225431   -0.361029   -0.78171
 -0.0288656    0.998471   -0.0119376  -0.0191182  -0.0413953
 -0.225431    -0.0119376   0.906771   -0.149307   -0.323283
 -0.361029    -0.0191182  -0.149307    0.760883   -0.517741
 -0.78171     -0.0413953  -0.323283   -0.517741   -0.121027

julia> HAH=H*A*H
5×5 Matrix{Float64}:
 -0.659563      -0.237615   -0.0404346  -0.171032   0.326885
  3.2447e-18     0.242845    0.249141   -0.107318  -0.622455
  3.03774e-17   -0.0435107  -0.144421    0.222133  -0.301418
 -8.21321e-18   -0.445167   -0.368263    0.702736   0.288812
  1.19525e-16    0.0151695  -0.408973    0.029018  -0.234736
```

eigenvalue λ

much closer to zero

The matrix B

$$HAH^{-1} = \left[\begin{array}{c|c} \lambda & b^T \\ \hline 0 & B \end{array}\right]$$

```
julia> B=HAH[2:5,2:5]
4×4 Matrix{Float64}:
  0.242845    0.249141  -0.107318  -0.622455
 -0.0435107  -0.144421   0.222133  -0.301418
 -0.445167   -0.368263   0.702736   0.288812
  0.0151695  -0.408973   0.029018  -0.234736
```

← extract the matrix B

Now we check that the eigenvalues of B plus -0.659563 are the same as the eigenvalues of A using the built-in library function in julia...

The first eigenvalue found using the power method

```
julia> eigvals(A)
5-element Vector{ComplexF64}:
  -0.6595626323511271 + 0.0im
  -0.5236845756168105 + 0.0im
 0.008759836881232323 + 0.0im
   0.5406744137537962 - 0.19432790355952867im
   0.5406744137537962 + 0.19432790355952867im

julia> eigvals(B)
4-element Vector{ComplexF64}:
  -0.5236845756168103 + 0.0im
 0.008759836881232346 + 0.0im
   0.5406744137537969 - 0.19432790355952861im
   0.5406744137537969 + 0.19432790355952861im
```

same

The eigenvalues of B are the same as the remaining eigenvalues of A.

While this doesn't guarantee anything (there could be bugs in julia) it suggests everything is at least consistent.

So, how did Julia find all the eigenvalues at once? Did it use deflation?

Probably not.

Sometimes you only want one or two largest eigenvalues, in which deflation works well, but when you want many eigenvalues, there are methods that are designed to find them all at once.

Next Idea QR method to find all eigenvalues at once...

function QR-ITERATION($A \in \mathbb{R}^{n \times n}$)
  for $k \leftarrow 1, 2, 3, \ldots$
    $Q, R \leftarrow$ QR-FACTORIZE(A)
    $A \leftarrow RQ$ weird idea to mult QR in reverse order...
  return diag(R)

Recall that Q can be made out of many Householder reflectors. Intuitively, is this like doing deflation simultaneously for all the eigenvalues at once?

Let try it out with the same matrix as before...

```
julia> AA=copy(A)
       for k=1:10000
           z=qr(AA)
           AA=z.R*z.Q
       end

julia> AA
5×5 Matrix{Float64}:
 -0.659563     0.173512     0.311192    0.257565    -0.0302395
 -3.0e-323     0.624229    -0.648052   -0.0385789   -0.420734
  1.0e-323     0.0690451    0.457119    0.419908     0.326988
 -3.0e-323    -1.0e-323    -1.0e-323   -0.523685     0.240838
  0.0          0.0          0.0         0.0          0.00875984
```

Block upper triangular matrix with "all" the eigenvalues on the diagonal

note all is in quotes because you can't immediately read off the complex eigenvalues

These are zero (approximately)

```
julia> eigvals(A)
5-element Vector{ComplexF64}:
 -0.6595626323511271 + 0.0im
 -0.5236845756168105 + 0.0im
  0.008759836881232323 + 0.0im
  0.5406744137537962 - 0.19432790355952867im
  0.5406744137537962 + 0.19432790355952867im
```

Trouble with complex conjugate pairs. Not unexpected as these eigenvalues have the same magnitude, deflation would not work either.

So the QR method found the eigenvalues with unique magnitudes, but failed to find the ones that came in complex-conjugate pairs...which instead lead to a 2x2 block on the diagonal...

We should have examined that 2x2 block more carefully in class... I'll do that here in the notes...

```
julia> R=AA[2:3,2:3]
2×2 Matrix{Float64}:
 0.624229   -0.648052
 0.0690451   0.457119

julia> eigvals(R)
2-element Vector{ComplexF64}:
 0.5406744137537873 - 0.19432790355953317im
 0.5406744137537873 + 0.19432790355953317im
```

This is the 2x2 block on the diagonal

These are the missing complex eigenvalues...

But still has difficulties when there are two eigenvalues with the same magnitude, such as complex conjugate pairs...

Another idea is the shifted QR method, but I can't find it is the book right now. In particular a better algorithm closer to what Julia uses is...

Solution, as with the power method, it to shift while doing the iteration to break the symmetry and speed the convergence...

As there were a few minutes left in class, I began an application of eigenvalues...

Application of eigenvalues...

Find $\|A\|_2 = \max\{\|Ax\|_2 : \|x\|_2 = 1\}$

$$= \sqrt{\max\{\|Ax\|_2^2 : \|x\|_2 = 1\}}$$

$$\|Ax\|_2^2 = Ax \cdot Ax = (Ax)^T Ax = x^T \underbrace{A^T A}_{B} x$$

$B = A^T A$

Note B is symmetric.

B is positive (semi) definite...

The fact that B is symmetric and positive semi-definite means that the eigenvalues are real and non-negative.

That means the eigenvalue of largest magnitude has to be unique (not counting multiplicity) and so the power method will always be able to find it.

Moreover since we'll be taking a maximum in the definition of the norm, we only need the largest eigenvalue of B.

We'll start here next time...