Finding eigenvalues and eigenvectors using power iteration:

Assumption the eigenvalues of largest magnitude are the exactly the same...

Order the eigenvalues from big to small...,

$$|\lambda_1| = |\lambda_2| \simeq \cdots \simeq |\lambda_m| > |\lambda_{m+1}| \geq \cdots \geq |\lambda_n|$$

all the eigenvalues that happen to have the same largest magnitude are the same

$$\lambda_1 = \lambda_2 \simeq \cdots \simeq \lambda_m$$

Assumption: $A \in \mathbb{R}^{n \times n}$ and there are n eigenvalues with n linearly independent eigenvectors.

(Statistically both assumptions hold for random matrices)

Let the corresponding eigenvectors be ξ_i.

Thus

$$A\xi_i = \lambda_i \xi_i \quad \text{for} \quad i = 1, \ldots, n$$

The power method (from Oct 7).

$$y_1 = Ax_0$$
$$x_1 = y_1/\|y_1\|$$
$$\vdots$$
$$y_{n+1} = Ax_n$$
$$x_n = y_n/\|y_n\|$$

Idea write $x_0 \in \mathbb{R}^n$ as a combination of the n linearly independent eigenvectors.

$$x_0 = c_1\xi_1 + c_2\xi_2 + \cdots + c_n\xi_n$$

Assumption that $c_1 \neq 0$.

Note if x_0 is chosen randomly, the $c_1 \neq 0$ is statistically almost certain.

$$y_1 = Ax_0 = A\left(c_1\xi_1 + c_2\xi_2 + \cdots + c_n\xi_n\right)$$

$$= c_1A\xi_1 + c_2A\xi_2 + \cdots + c_nA\xi_n$$

$$= c_1\lambda_1\xi_1 + c_2\lambda_2\xi_2 + \cdots + c_n\lambda_n\xi_n$$

$$x_1 = \frac{y_1}{\|y_1\|} = \frac{c_1\lambda_1\xi_1 + c_2\lambda_2\xi_2 + \cdots + c_n\lambda_n\xi_n}{\|c_1\lambda_1\xi_1 + c_2\lambda_2\xi_2 + \cdots + c_n\lambda_n\xi_n\|}$$

$$y_2 = Ax_1 = \frac{c_1\lambda_1^2\xi_1 + c_2\lambda_2^2\xi_2 + \cdots + c_n\lambda_n^2\xi_n}{\| c_1\lambda_1\xi_1 + c_2\lambda_2\xi_2 + \cdots + c_n\lambda_n\xi_n \|}$$

$$x_2 = \frac{y_2}{\|y_2\|} = \frac{c_1\lambda_1^2\xi_1 + c_2\lambda_2^2\xi_2 + \cdots + c_n\lambda_n^2\xi_n}{\| c_1\lambda_1^2\xi_1 + c_2\lambda_2^2\xi_2 + \cdots + c_n\lambda_n^2\xi_n \|}$$

$$\vdots$$

$$x_k = \frac{c_1\lambda_1^k\xi_1 + c_2\lambda_2^k\xi_2 + \cdots + c_n\lambda_n^k\xi_n}{\| c_1\lambda_1^k\xi_1 + c_2\lambda_2^k\xi_2 + \cdots + c_n\lambda_n^k\xi_n \|}$$

(factor out the λ_1^n)

usually what's here is just one term

$$= \frac{\lambda_1^k\left[\left(c_1\xi_1 + \cdots + c_m\xi_m\right) + c_{m+1}\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k\xi_{m+1} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\xi_n\right]}{\left\| \lambda_1^k\left[\left(c_1\xi_1 + \cdots + c_m\xi_m\right) + c_{m+1}\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k\xi_{m+1} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\xi_n\right] \right\|}$$

$$= \left(\frac{\lambda_1}{|\lambda_1|}\right)^k \cdot \frac{\left(c_1\xi_1 + \cdots + c_m\xi_m\right) + c_{m+1}\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k\xi_{m+1} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\xi_n}{\left\| \left(c_1\xi_1 + \cdots + c_m\xi_m\right) + c_{m+1}\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k\xi_{m+1} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\xi_n \right\|}$$

ignore this for now

look at this term first

This term converges as $k \to \infty$

complex number of norm one

$\frac{\lambda_1}{|\lambda_1|} = e^{i\theta}$ for some θ, ← This term spins around in the complex circle of unit length and does not converge...

$$\lim_{k\to\infty} \frac{\left(c_1\xi_1 + \cdots + c_m\xi_m\right) + c_{m+1}\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k \xi_{m+1} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k \xi_n}{\left\| \left(c_1\xi_1 + \cdots + c_m\xi_m\right) + c_{m+1}\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k \xi_{m+1} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k \xi_n \right\|}$$

as $k \to \infty$, $\left(\frac{\lambda_{m+1}}{\lambda_1}\right)^k \to 0$ and $\left(\frac{\lambda_n}{\lambda_1}\right)^k \to 0$ as $k \to \infty$

$$= \frac{c_1\xi_1 + \cdots + c_m\xi_m}{\left\| c_1\xi_1 + \cdots + c_m\xi_m \right\|}$$

what's the rate of convergence of this limit? The rate of convergence is determined by the ratio $|\lambda_{m+1}/\lambda_1|$.

Claim: This is an eigenvector... Since $\lambda_1 = \lambda_2 = \cdots = \lambda_m$ by hypothesis

$$A\left(c_1\xi_1 + \cdots + c_m\xi_m\right)$$
$$= c_1\lambda_1\xi_1 + \cdots + c_m\lambda_m\xi_m$$
$$= \lambda_1\left(c_1\xi_1 + \cdots + c_m\xi_m\right)$$

In regular analysis like calculus, one is happy to take the limit as $k \to \infty$ and so the focus is on what that limit is. In numerical analysis we also need to approximate the limit by taking k large. For this reason we are also interested in how fast does the limit converge.

Then the x_k's "converge" to an eigenvector with eigenvalue λ_1 as $k \to \infty$.. upto that phase $\left(\frac{\lambda_1}{|\lambda_1|}\right)^k$

example: The largest eigenvalue is not real...

$$A = S\begin{bmatrix} 2+3i & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} S^{-1}$$

where $S \in \mathbb{R}^4$ is any invertible matrix...

load the linear algebra library

```
julia> using LinearAlgebra
```

practice how complex numbers work

```
julia> 1im*1im
-1 + 0im
```

make the diagonal matrix. Note that $\lambda_1 = 3$ is the eigenvalue of largest magnitude and it's complex.

```
julia> D=diagm([2+3im,2,-3,1])
4×4 Matrix{Complex{Int64}}:
 2+3im  0+0im   0+0im  0+0im
 0+0im  2+0im   0+0im  0+0im
 0+0im  0+0im  -3+0im  0+0im
 0+0im  0+0im   0+0im  1+0im
```

create a random matrix with some entries negative and some positive

```
julia> S=rand(4,4).-0.5
4×4 Matrix{Float64}:
  0.00262579  -0.13372   -0.335557  -0.0120293
 -0.152103    -0.262071  -0.351501  -0.321613
  0.12013     -0.130513   0.368826  -0.171375
 -0.470957     0.485539   0.117832  -0.445887
```

These columns are the eigenvectors

practice making inverse matrices using Julia.

```
julia> inv(S)
4×4 Matrix{Float64}:
 17.4768   -8.04579   7.44301    2.47116
  9.9785   -5.55521   2.95256    2.60291
 -6.48615   2.04366  -0.942862  -0.936697
 -9.30762   2.98899  -4.89552   -2.26597

julia> S*inv(S)
4×4 Matrix{Float64}:
  1.0           -5.4148e-17   -3.26485e-17  -1.55558e-17
 -2.95067e-16    1.0           2.17969e-17   2.0188e-17
  6.55715e-17   -1.93271e-16   1.0           5.12217e-17
 -1.72653e-15    1.17576e-15  -1.19697e-15   1.0
```

Up to rounding error this is the identity matrix

This is our test problem. A matrix for which we know what the eigenvalues are.

```
julia> A=S*D*inv(S)
4×4 Matrix{ComplexF64}:
  -8.99433+0.137671im   3.46477-0.0633797im  -1.64081+0.0586313im  -1.59884+0.0194663im
  -14.3929-7.97483im    6.55304+3.67137im    -3.23155-3.39631im    -2.37502-1.12761im
   10.3662+6.29849im   -3.25654-2.89963im     2.89979+2.68239im     1.33906+0.890585im
 -0.328753-24.6925im   0.128708+11.3677im    -1.62735-10.516im      1.5415-3.49143im
```

pick random initial vector to start with. We take $x_0 \in \mathbb{C}^4$ here, but it could have been real and work just as well.

```
julia> x0=rand(4)+rand(4)*1im
4-element Vector{ComplexF64}:
 0.3646362473733564 + 0.2859359924723315im
 0.7144674878227177 + 0.5273087180605243im
 0.4863921449522768 + 0.9256639388101893im
 0.1298430868896605 + 0.8692799405671106im
```

Press [alt]+[enter] here to put everything in the same block.

```
julia> x=x0
       for j=1:10
           y=A*x
           x=y/norm(y)
       end

julia> x
4-element Vector{ComplexF64}:
 0.014021934180851738 + 0.018264906078432 21im
 0.024578759305304863 + 0.3202638980053197im
 -0.02353715395434897 - 0.26021021001426836im
  0.0256770040988171 5 + 0.9096010330039171im
```

same code as from Oct 7

an approximation of the eigenvector...

Does it converge? What did "converge" in quotes mean when we discussed the theory?

In this case $\frac{\lambda_1}{|\lambda_1|} = \frac{2+3i}{|2+3i|}$ is complex, so the phase of x_k keeps changing as $k \to \infty$.

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

julia> x
4-element Vector{ComplexF64}:
  0.002260004914597034 - 0.004634087598469032im
 -0.1309142135485839 + 0.26843676004016936im
  0.1033955890071581 - 0.2120104172567377im
 -0.4053499186909257 + 0.831161218251763im
```

← change this

press [↑] and then [←] to edit the previous code block and change the iterations to 100.

← another eigenvector approximation

```
julia> x=x0
       for j=1:200
           y=A*x
           x=y/norm(y)
       end

julia> x
4-element Vector{ComplexF64}:
 -0.0050234935150534525 + 0.0011605658871625363im
   0.29099360636897936 - 0.06722756820969054im
  -0.22982573539137985 + 0.05309609889770196im
    0.9010040277781879 - 0.20815684059346198im
```

← do it again for 200 iterations

← seems like a completely different vector. Both, however are good approximations of an eigenvector.

They simply differ from each other by a multiple of $e^{i\theta}$.

Last week we checked the eigenvector by dividing as

```
julia> A*x./x
4-element Vector{ComplexF64}:
  1.999999999999885 + 3.0000000000000973im
  2.00000000000001 + 2.9999999999999973im
  2.00000000000001 + 2.9999999999999982im
 1.9999999999999993 + 3.0im
```

The problem is that this gives 4 different values for λ and if any of the elements of x were zero the element by element division would not work.

Given something x_k that supposed approximate an eigenvector how to find the corresponding eigenvalue?

least squares approximation...

$$Ax_k = \boxed{?}\, x_k$$ ← n equations, 1 unknown

? = eigenvalue...

Minimize $\|Ax_k - \lambda x_k\|_2^2$ by choosing λ

Usual least squares

eqs ↓ $m > n$ ↑ unknowns

$Ax = b$

$m \times n$

x unknown

Solve

$A^TAx = A^Tb$ n eqs.

$n \times m$ $m \times n$

$A^TA \in \mathbb{R}^{n \times n}$

$x_k \lambda = Ax_k$

$x_k^T x_k \lambda = x_k^T A x_k$ one eq.

$\lambda \approx \dfrac{x_k \cdot Ax_k}{x_k \cdot x_k}$ least sq. approx...

called the Rayleigh-Quotient

```
julia> x'*A*x/(x'*x)
1.9999999999999998 + 3.0000000000000001im
```

compute the Rayleigh-Quotient for the approximation x found before.

This is $2+3i$ within rounding error, so the method worked for our test problem

Note that constructing a good test problem is an important part of numerical computing.

Usually computers are used to solve problems that people can't find the answer to. In that case, however, it is difficult to check whether the answer produced by the computer is correct.

Again, what separates science from just simply making up answers is having an analysis and confirmation of how correct the answers are.

That's why it's important to construct problems to which one knows the answer when testing and debugging computer programs. Otherwise, the computer provides an answer but there is no way to verify the answer is correct.

That's what we did here. By starting with the eigenvalues and then constructing a complicated matrix using a similarity transform that was guaranteed to have those same eigenvalues.

Next class focuses on how to speed up the rate of convergence and how to find other eigenvalues...

Don't forget there is an exam next Tuesday Oct 26 in class.

Please look at the study topics on the website and let me know if you have any questions next time.