Lecture 24: Schur decomposition, The QR Algorithm and Companion Matrices

In this lecture we will see that eigenvalues are calculatable. We will also establish that symmetric matrices have eigenvalue decompositions with orthogonal eigenvectors.

Jordan canonical form

A fact one should remember from linear algebra is that every matrix can be reduced to Jordan canonical form: given a square matrix $A \in \mathbb{R}^{n\times n}$, here exists a decomposition

$$A = VJV^{-1}$$

where $V \in \mathbb{C}^{n\times n}$ and $J \in \mathbb{C}^{n\times n}$ is block diagonal

$$J = \begin{pmatrix} J_1 & & \\ & \ddots & \\ & & J_p \end{pmatrix}$$

and the blocks J_k are upper triangular containing an eigenvalue repeated on the diagonal with ones on the super diagonal:

$$J_k = \begin{pmatrix} \tilde{\lambda_k} & 1 & & \\ & \tilde{\lambda_k} & \ddots & \\ & & \ddots & 1 \\ & & & \tilde{\lambda_k} \end{pmatrix}$$

(We use $\tilde{\lambda_k}$ since λ_k will have eigenvalues repeated multiple times.

The key point for us is that J is upper triangle.

As mentioned before: we will never use Jordan canonical form in programming, for two reasons:

1. The condition number of `V` can be large, meaning it can introduce very large errors
2. The decomposition is unstable to perturbation: a small perturbation renders a matrix diagonalize, which is jump.

Here is a simple example of the second issue. The following matrix is already in Jordan canonical form, and therefore the `eig` command fails (`V` is not invertible):

In [5]:

```
A=[1 1.0;
   0 1.0]

λ,V=eig(A)

V
```

Out[5]:

```
2x2 Array{Float64,2}:
 1.0  -1.0
 0.0   2.22045e-16
```

The the related property $AV = V\Lambda$ still works:

In [7]:

```
norm(A*V - V*diagm(λ))
```

Out[7]:

```
2.220446049250313e-16
```

A small perturbation, however, renders the matrix diagonalizable:

In [44]:

```
A=[1 1.0;
   0 1.0]

ΔA=0.00000001rand(2,2)

λ,V=eig(A+ΔA)

rank(V)
```

Out[44]:

```
2
```

Unfortunately, because it is near a degenerate case, the diagonalization is prone to large errors:

In [45]:

```
cond(V)
```

Out[45]:

```
13396.757611710864
```

Conjugate transpose

Eigenvalue decompositions naturally lead to complex matrices, even when the input is real. We therefore need to introduce some basic complex number tools, beginning with the conjugate transpose of a matrix.

When a matrix $A \in \mathbb{C}^{n\times m}$ is complex, we define its complex conjugate entry-wise:

$$\bar{A} \triangleq \begin{pmatrix} \overline{a_{11}} & \cdots & \overline{a_{1m}} \\ \vdots & \ddots & \vdots \\ \overline{a_{n1}} & \cdots & \overline{a_{nm}} \end{pmatrix}$$

where the complex conjugate is defined for complex numbers via $\overline{p+iq} = p - iq$. Recall that complex conjugation satisfies the following properties that allow us to interchange conjugation with addition and multiplication:

Complex conjugation for matrices therefore satisfies:

1. $\overline{A+B} = \bar{A} + \bar{B}$
2. $\overline{AB} = \bar{A}\bar{B}$
3. $A^\star \triangleq \overline{A^\top} = \bar{A}^\top$

The last property defines the *conjugate transpose*, which in Julia is the default behaviour of `'`, with `.'` being the standard transpose:

In [16]:

```
A=[1.+im 1.+im;
    0     2.+im  ]
A', A.'
```

Out[16]:

```
(
2x2 Array{Complex{Int64},2}:
 1-1im  0+0im
 1-1im  2-1im,

2x2 Array{Complex{Int64},2}:
 1+1im  0+0im
 1+1im  2+1im)
```

Unitary matrices

Before we proceed, we need to introduce the complex-valued version of an orthogonal matrix: a *unitary matrix*. A complex matrix $Q \in \mathbf{C}^{n\times m}$ is unitary if

$$Q^\star Q = I$$

For square matrices $Q \in \mathbf{C}^{n\times n}$, this implies that

$$Q^{-1} = Q^\star.$$

Note that every (real valued) orthogonal matrix is also unitary.

Complex matrices have QR Decompositions where `Q` is untary:

In [20]:

```
A=rand(10,5)+im*rand(10,5)
Q,R=qr(A)

norm(Q'*Q-I)
```

Out[20]:

```
5.331636998505262e-16
```

The Schur decomposition

Instead of Jordan canonical form/eigenvalue decompositions, we will use *Schur decomposition*

$$A = QUQ^\star$$

where Q is unitary and U is *upper triangular*. A Schur factorization arises from a QR Decomposition of the eigenvector matrices $QR = V$:

$$A = VJV^{-1} = QRJ(QR)^{-1} = QRJR^{-1}Q^\star = QUQ^\star$$

This can be calculated via `schur`:

In [32]:

```
A=rand(5,5) +rand(5,5)*im

U,Q,λ=schur(A)
norm(A-Q*U*Q')
```

Out[32]:

```
5.7401229275105705e-15
```

Because they are built on top of orthogonal operations, they are not as sensitive to perturbations as eigenvalue decompositions:

In [47]:

```
A=[1 1.0;
   0 1.0]

ΔA=0.000001rand(2,2)

U,Q,λ=schur(A+ΔA)

norm(A+ΔA-Q*U*Q')
```

Out[47]:

```
3.592760104303804e-16
```

Eigenvalue decompositions of real symmetric matrices

Computationally, for general matrices one should only use the Schur factorization: eigenvalue decompositions may introduce substantial errors. Fortunately, special matrices have *symmetries* which give extra structure in the eigenvalue decomposition.

We will show that a real symmetric matrix (satisfying $A = A^\top$) has an eigenvalue decomposition with orthogonal eigenvectors: We have

$$A = QUQ^\star = QU^\star Q^\star = A^\star = A^\top$$

But this implies that

$$Q^\star A Q = U = U^\star = Q^\star A^\top Q$$

All upper triangular matrices U satisfying $U = U^\star$ are

1. diagonal
2. real

Thus we have

$$A = Q\Lambda Q^\star.$$

Exercise (Advanced) Prove that Q is also real valued, thus we have

$$A = Q\Lambda Q^\top.$$

QR Algorithm

Last point: Schur decompositions are computable using the *QR Algorithm* (**Note!!** QR Algorithm ≠ QR Decomposition, but the QR Algorithm is built using the QR Decomposition.) We begin with the symmetric case, where it is the same as the eigenvalue decomposition:

In [48]:

```
n=10
A=rand(10,10)
A=A+A'      # a real, symmetric matrix
```

Out[48]:

```
10x10 Array{Float64,2}:
 0.629579  1.17426   1.30773   1.10905   …  0.633427  0.893826  0
.895246
 1.17426   1.57507   0.955505  0.752102     1.00802   0.442039  0
.618094
 1.30773   0.955505  0.906221  1.33977      1.89337   0.850592  1
.5802
 1.10905   0.752102  1.33977   1.65559      0.865779  1.81067   1
.32032
 0.616983  1.29894   1.76134   0.650936     0.40762   1.55697   0
.408059
 0.697097  1.16665   0.950683  1.13164   …  0.490958  1.05758   1
.49264
 1.07365   0.821973  1.04927   1.58266      0.639588  1.11594   1
.74319
 0.633427  1.00802   1.89337   0.865779     1.93083   0.761516  0
.198298
 0.893826  0.442039  0.850592  1.81067      0.761516  0.298269  0
.554476
 0.895246  0.618094  1.5802    1.32032      0.198298  0.554476  1
.28652
```

Conjugating a matrix A by another matrix and its inverse leaves eigenvalues alone: the eigenvalues of A equal the eigenvalues of $W^\top A W$. Choosing $W = Q$ to be orthogonal means that the eigenvalues of A equal the eigenvalues of $Q^\top A Q$:

In [51]:

```
Q,R=qr(A)

sort(eigvals(A))-sort(eigvals(Q'*A*Q))
```

Out[51]:

```
10-element Array{Float64,1}:
  1.77636e-15
 -1.55431e-15
  3.33067e-16
  2.22045e-16
  1.94289e-16
  5.55112e-16
  1.55431e-15
  3.33067e-15
 -8.88178e-16
  0.0
```

We want to conjugate by a sequence of orthogonal matrices so that it eventually is diagonal. The QR Decomposition provides one such sequence:

In [52]:

```
B=A

for k=1:200
    Q,R=qr(B)
    B=Q'*B*Q
end



sort(eigvals(B))-eigvals(A)
```

Out[52]:

```
10-element Array{Float64,1}:
 -1.42109e-14
 -8.88178e-15
 -7.66054e-15
  2.77556e-16
 -5.55112e-16
  3.44169e-15
  5.10703e-15
  1.06581e-14
  4.44089e-16
  1.42109e-14
```

`B` has become almost diagonal:

In [54]:

```
norm(B-diagm(diag(B)))
```

Out[54]:

```
0.00011316230506445243
```

Continuing the procedure gets closer and closer to a diagonal matrix:

In [56]:

```
B=A


for k=1:1000
    Q,R=qr(B)
    B=Q'*B*Q
end



norm(B-diagm(diag(B)))
```

Out[56]:

```
1.8734116074365207e-15
```

Thus the diagonal approximates the eigenvalues:

In [59]:

```
norm(sort(diag(B))-sort(eigvals(A)))
```

Out[59]:

```
5.1391365172691475e-14
```

To get the exact eigenvalues requires an infinite number of iterations, but to get eigenvalues within a tolerance can be done with a finite number of iterations.

The QR Algorithm can be modified so that the number of iterations is much smaller than this example, but this is beyond the scope of the course.

Application: calculate roots of a polynomial

It is well-known that there is no closed form formula for (general) degree 5 or greater polynomials. We will see that it is possible to approximate the roots by calculating eigenvalues of a *companion matrix*. Give a polynomial

$$p(z) = \sum_{k=1}^{n} c_k z^{k-1} + z^n,$$

the companion matrix is defined as the $n \times n$ matrix

$$C = \begin{pmatrix} 0 & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ -c_1 & -c_2 & -c_3 & \cdots & -c_n \end{pmatrix}$$

The eigenvalues of C are the roots λ_k of $p(\lambda_k) = 0$. To see this, we will construct an eigenvector $\mathbf{v}_k$ so that $C\mathbf{v}_k = \lambda_k \mathbf{v}_k$. Indeed, define

$$\mathbf{v}_k \triangleq \begin{pmatrix} 1 \\ \lambda_k \\ \lambda_k^2 \\ \vdots \\ \lambda_k^{n-1} \end{pmatrix}$$

Then we have, using $\lambda_k^n = -\sum_{k=1}^{n} c_k z^{k-1}$

$$C\mathbf{v}_k = \begin{pmatrix} \lambda_k \\ \lambda_k^2 \\ \vdots \\ \lambda_k^{n-1} \\ -c_1 - c_2\lambda_k - \cdots - c_n\lambda_k^{n-1} \end{pmatrix} = \begin{pmatrix} \lambda_k \\ \lambda_k^2 \\ \vdots \\ \lambda_k^{n-1} \\ \lambda_k^n \end{pmatrix} = \lambda_k \mathbf{v}_k$$

As an example, Consider the following quintic polynomial:

$$z^5 + 2z^4 - 3z^3 + 1/2z^2 + 6 = 0$$

This leads to the companion matrix:

In [64]:

```
c=[6,0,1/2,-3,2]
n=length(c)
C=zeros(n,n)
for k=1:n-1

    C[k,k+1]=1
end
C[end,:]=-c
C
```

Out[64]:

```
5x5 Array{Float64,2}:
  0.0   1.0   0.0  0.0   0.0
  0.0   0.0   1.0  0.0   0.0
  0.0   0.0   0.0  1.0   0.0

  0.0   0.0   0.0  0.0   1.0
 -6.0  -0.0  -0.5  3.0  -2.0
```

The eigenvalues of C are the solution to the equation:

In [67]:

```
λ=eigvals(C)
```

Out[67]:

```
5-element Array{Complex{Float64},1}:
  -3.08934+0.0im
    1.1049+0.68978im
    1.1049-0.68978im
 -0.560231+0.911526im
 -0.560231-0.911526im
```

We can check this:

In [68]:

```
# evaluate a polynomial at z
function p(c,z)
    n=length(c)
    ret=z^n
    for k=1:n
        ret+=c[k]*z^(k-1)
    end
    ret
end

p(c,λ[4])
```

Out[68]:

```
-2.886579864025407e-15 + 2.6645352591003757e-15im
```

C is not symmetric, but we can still employ the QR algorithm. In this case, because there are imaginary roots, it reduces the matrix to block upper triangular:

In [69]:

```
B=C
for k=1:20000
    Q,R=qr(B)
    B=Q'*B*Q
end

B
```

Out[69]:

```
5x5 Array{Float64,2}:
 -3.08934       -1.93154       -3.94329        3.77616   -1.6922

 -1.4822e-323    0.728675      -0.715838       0.26945   -0.13704
8
  9.88131e-324   0.862404       1.48113       -0.88335    0.36733
9
 -9.88131e-324  -1.4822e-323    2.96439e-323  -0.559183  -0.83488

  4.94066e-324  -4.94066e-324   9.88131e-324   0.995209  -0.561278
```

The eigenvalues of the diagonal blocks which are at most 2x2 (a solveable problem!) are very close to the roots:

In [76]:

```
λ̃=[B[1,1];eigvals(B[2:3,2:3]);eigvals(B[4:5,4:5])]

norm(λ-λ̃)
```

Out[76]:

```
4.60171037827172e-14
```

We can also see this directly:

In [77]:

```
z=λ̃[3];
z^5 + 2z^4-3z^3+0.5z^2+6
```

Out[77]:

```
-5.950795411990839e-13 - 3.3495428652940973e-13im
```

Anonymous functions (Unnamed functions)

So far, we're comfortible with named functions:

In [78]:

```
function my_f(x)
    x^2
end
```

Out[78]:

```
my_f (generic function with 1 method)
```

We can also create functions without naming them, via

In [79]:

```
g=x->x^2
```

Out[79]:

```
(anonymous function)
```

In [80]:

```
g(2.),my_f(2.)
```

Out[80]:

```
(4.0,4.0)
```

Sometimes its useful to return a function from a function: for example, when the function depends on parameters:

$$ax^2$$

In [69]:

```
function create_fun(a)
    x->a*x^2
end
```

Out[69]:

```
create_fun (generic function with 1 method)
```

In [71]:

```
f1=create_fun(2.0)
f2=create_fun(3.0)
```

Out[71]:

```
(anonymous function)
```

In [73]:

```
f1(2.0)
```

Out[73]:

```
8.0
```

In [74]:

```
f2(2.0)
```

Out[74]:

```
12.0
```