In science and engineering an important goal is to become a skilled practitioner by doing it yourself. The computer labs provide computational experience related to the analytic theory presented in the lectures.

The Companion Matrix

This lab is about using the eigenvalue-eigenvector problem to find the roots of a polynomial via what is called the companion matrix.

In an introductory linear algebra course one uses the roots of the characteristic polynomial

$$\chi(\lambda) = \det(A - \lambda I)$$

to solve for the eigenvalues of a matrix. From a numerical point of view it's actually better to reverse this idea and use the eigenvalues of a matrix to solve for the roots of a polynomial. The main reason this is practical is the existence of iterative methods for finding the eigenvalues of a matrix that rely on the QR and Schur factorizations rather than root finding.

The companion matrix of the polynomial

$$p(\lambda) = \lambda^n + a_{n-1}\lambda^{n-1} + \ldots + a_1\lambda + a_0 \tag{5.1}$$

is the $n \times n$ matrix given by

$$C_p = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix}.$$

We claim that $p(\lambda) = (-1)^n \det(C_p - \lambda I)$.

Proof. When $n = 1$ then $p(\lambda) = \lambda + a_0$. The corresponding companion matrix is $C_p = [\, -a_0 \,]$. Thus $(-1)^n \det(C_p - \lambda I) = -\det[-a_0 - \lambda] = p(\lambda)$.

For induction suppose $q(\lambda) = (-1)^k \det(C_q - \lambda I)$ for all polynomials q of degree $k < n$. Let p be a polynomial of degree n given by (5.1). By

expanding along the first row we obtain that

$$\begin{aligned}(-1)^n \det(C_p - \lambda I) &= (-1)^n \det \begin{bmatrix} -\lambda & 0 & \cdots & 0 & -a_0 \\ 1 & -\lambda & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} - \lambda \end{bmatrix} \\ &= (-1)^n \big\{ -\lambda \det(C_q - \lambda I) + (-1)^n a_0 \big\} \\ &= \lambda(-1)^{n-1} \det(C_q - \lambda I) + a_0\end{aligned}$$

where $q(\lambda) = \lambda^{n-1} + a_{n-1}\lambda^{n-2} + \cdots + a_2\lambda + a_1$. Since the degree of q is $n-1$, the induction hypothesis implies

$$(-1)^{n-1} \det(C_q - \lambda I) = q(t).$$

Consequently,

$$(-1)^n \det(C_p - \lambda I) = \lambda q(t) + a_0 = p(t)$$

completes the induction. ////

To illustrate how the eigenvalue-eigenvector problem can be used to find the roots of a polynomial, consider the polynomial

$$p(\lambda) = (\lambda - 1)(\lambda - 2)(\lambda^2 + 4)$$

with roots $\lambda = 1,\ 2,\ 2i$ and $-2i$. Multiplying this polynomial out obtains

$$p(\lambda) = \lambda^4 - 3\lambda^3 + 6\lambda^2 - 12\lambda + 8.$$

Therefore $a_0 = 8$, $a_1 = -12$, $a_2 = 6$, $a_3 = -3$ and the companion matrix for p is

$$C_p = \begin{bmatrix} 0 & 0 & 0 & -8 \\ 1 & 0 & 0 & 12 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

To find the eigenvalues of this matrix in Julia type

```
julia> using LinearAlgebra

julia> Cp=[0 0 0 -8; 1 0 0 12; 0 1 0 -6; 0 0 1 3]
4×4 Matrix{Int64}:
 0  0  0  -8
 1  0  0  12
 0  1  0  -6
 0  0  1   3

julia> eigvals(Cp)
4-element Vector{ComplexF64}:
 6.938893903907228e-16 - 2.0000000000000004im
 6.938893903907228e-16 + 2.0000000000000004im
    1.0000000000000004 + 0.0im
                   2.0 + 0.0im
```

Up to rounding errors note that the eigenvalues of C_p are equal to the already known roots of the polynomial p.

Finding the Roots

For this lab you will use the companion matrix to find the roots of an individualized polynomial and then plug those roots in to compute the residual errors. Each person will have a different polynomial. Click on the following link to retrieve your polynomial:

`https://fractal.math.unr.edu/~ejolson/466-22/poly/mkpoly.cgi`

Please do not use anyone else's polynomial for this lab.

The rest of this discussion presents a step-by-step walk through for the polynomial which appears when I click the above link. That polynomial is a different than the one which will appear when you click the same link. To finish this lab you will need to repeat these same steps but for your own individualized polynomial.

Upon clicking on the link, I obtained the example

> Your polynomial is $p(\lambda) = \lambda^5 - 9\lambda^4 - 4\lambda^3 - 6\lambda^2 + 4$.
>
> In Julia-compatible code that is
>
> ```
> p(x) = x^5 - 9*x^4 - 4*x^3 - 6*x^2 + 4
> ```

Although it would not be difficult to construct the companion matrix C_p by hand from the polynomial p as in the previous example, it would be nice to extract the polynomial coefficients a_k into an array and then create the companion matrix C_p automatically.

The coefficients a_k can be found via Taylor's theorem. Recall Taylor's theorem states that

$$p(\lambda) = \sum_{k=0^n} \frac{\lambda^k}{k!} p^{(k)}(0) + R_n \quad \text{where} \quad R_n = \frac{\lambda^{n+1}}{(n+1)!} p^{(n+1)}(\xi)$$

for some ξ between 0 and λ. Now, since p is a polynomial of degree n then $p^{(n+1)} = 0$ and so $R_n = 0$. This means the polynomial is given exactly by the sum and in particular that

$$a_k = \frac{1}{k!} p^{(k)}(0).$$

Here is a function in Julia that uses the `Symbolics` package to find the coefficients for the Taylor polynomial of degree n. The coefficients are then returned as a vector of $n+1$ floating point values.

```
# taylor.jl -- Find the Taylor polynomial of degree n

using Symbolics

function taylor(p,n)
    @variables t
    D(g)=expand_derivatives(Differential(t)(g))
    dp=p(t)
    a=zeros(n+1)
    for k=1:n+1
        a[k]=Symbolics.value(substitute(dp,t=>0))
        dp=D(dp/k)
    end
    return a
end
```

Note that line 12 repeatedly takes the derivative while dividing by k to obtain the factorial. As written `taylor` forms an expansion about 0. That's exactly what's needed to find the coefficients of a polynomial; however, one could change `=>0` on line 11 to expand around a different value.

Assuming `taylor.jl` is in the current directory, find the coefficients of the polynomial with

```
julia> include("taylor.jl")
taylor (generic function with 1 method)

julia> p(x) = x^5 - 9*x^4 - 4*x^3 - 6*x^2 + 4
p (generic function with 1 method)

julia> a=taylor(p,5)
6-element Vector{Float64}:
  4.0
  0.0
 -6.0
 -4.0
 -9.0
  1.0
```

Now that the coefficients of p are stored as the 6-element vector `a`, construct the companion matrix using the `SpecialMatrices` library.

```
julia> using SpecialMatrices,Polynomials

julia> Cp=Companion(Polynomial(a))
5×5 Companion{Float64}:
 0.0  0.0  0.0  0.0  -4.0
 1.0  0.0  0.0  0.0  -0.0
 0.0  1.0  0.0  0.0   6.0
 0.0  0.0  1.0  0.0   4.0
 0.0  0.0  0.0  1.0   9.0
```

The only thing left is to the find the eigenvalues λ of C_p and compute the residual errors by plugging those values into the polynomial p.

```
julia> using LinearAlgebra

julia> lambda=eigvals(Cp)
5-element Vector{ComplexF64}:
  -0.7063426833492995 + 0.0im
 -0.18765624022179056 - 0.9847793938671874im
 -0.18765624022179056 + 0.9847793938671874im
   0.5938991041128365 + 0.0im
    9.487756059680041 + 0.0im

julia> for x in lambda
           println("|p($x)| = ",abs(p(x)))
       end
|p(-0.7063426833492995 + 0.0im)| = 2.6645352591003757e-15
|p(-0.18765624022179056 - 0.9847793938671874im)| = 1.1990408665
95169e-14
|p(-0.18765624022179056 + 0.9847793938671874im)| = 1.1990408665
95169e-14
|p(0.5938991041128365 + 0.0im)| = 2.6645352591003757e-15
|p(9.487756059680041 + 0.0im)| = 2.4215296434704214e-11
```

While all the residual errors are small, the residual for

$$\hat{x} = 9.487756059680041$$

is the largest. This, however, does not mean that root was computed less accurately. To determine the error in the approximation $\hat{x}$ of the exact root x it's possible to employ backwards error analysis. In this case

$$p'(\hat{x}) = 8575.389764325777$$

implies

$$|\hat{x} - x| \approx \left| \frac{p(\hat{x})}{p'(\hat{x})} \right| = 2.823812922817986 \times 10^{-15}.$$

Thus, $\hat{x}$ is accurate to 15 significant digits, which is as much as could be expected for a double-precision floating point number.

Note backwards error analysis is not the focus of this lab and need not be included for full credit. If you have time and would like extra credit, you may further estimate $|\hat{x} - x|$ for each approximation.

Submitting Your Work

A single PDF file should be submitted for grading that contains

- A program that uses the companion matrix to approximate the roots of your individualized polynomial and then plugs those approximations back into the polynomial to find the residual errors.
- The output from running that program.

If you want to avoid the `Symbolics` packages and construct the companion matrix by hand that is okay. For extra credit your program may further estimate errors $|\hat{x} - x|$ using backward error analysis.

After debugging and making sure your program runs correctly, please prepare your submission by typing

```
$ julia lab05.jl >lab05.out
$ j2pdf -o lab05.pdf lab05.jl lab05.out
```

Before uploading, check `lab05.pdf` using the PDF previewer with

```
$ evince lab05.pdf &
```

to make sure the output looks correct. Please reboot into Microsoft Windows before leaving the lab.