

Problems

- (1) A popular explicit Runge-Kutta method is defined by the following formulas:

$$\begin{aligned}k_1 &= hf(x_n, y_n) \\k_2 &= hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\k_3 &= hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right) \\k_4 &= hf(x_n + h, y_n + k_3) \\y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\end{aligned}$$

With this Runge-Kutta method in mind, we want to approximate the region of absolute stability acting on the standard test problem $y' = \lambda y$. So, let's just plug this test problem into our scheme as

$$\begin{aligned}k_1 &= h\lambda y_n \\k_2 &= h\left(\lambda y_n + \frac{1}{2}k_1\right) \\k_3 &= h\left(\lambda y_n + \frac{1}{2}k_2\right) \\k_4 &= h(\lambda y_n + k_3) \\y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\&= y_n + h\lambda y_n + \frac{1}{2}(h\lambda)^2 y_n + \frac{1}{6}(h\lambda)^3 y_n + \frac{1}{24}(h\lambda)^4 y_n \\&= \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{24}(h\lambda)^4\right) y_n.\end{aligned}$$

If we let, $z = h\lambda$, then our final expression can be written as

$$y_{n+1} = \left(1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4\right) y_n$$

which shows us our scheme will converge if

$$p(z) = \left|1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4\right| < 1.$$

This equation is quite tricky to work with so we will find try to approximate the region of absolute stability. First, we will find the solutions of the equation $|p(z)| = 1$ for purely real z . Doing so, we find the two real roots to be at

$$z = 0$$

and

$$z = \frac{1}{3} \left(-10\sqrt[3]{\frac{2}{9\sqrt{29}-43}} + 2^{2/3}\sqrt[3]{9\sqrt{29}-43} - 4 \right).$$

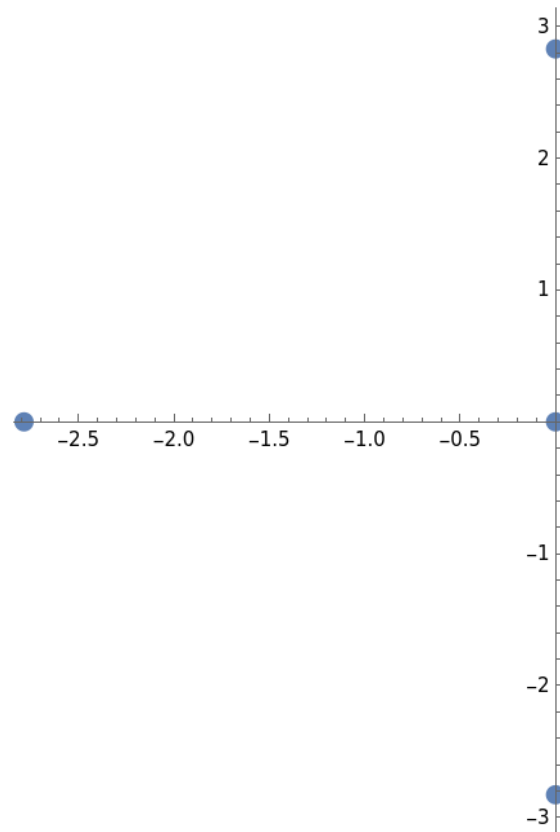
Next, we seek the purely imaginary solutions of $p(z) = 1$ which yields

$$z = -i2\sqrt{2}$$

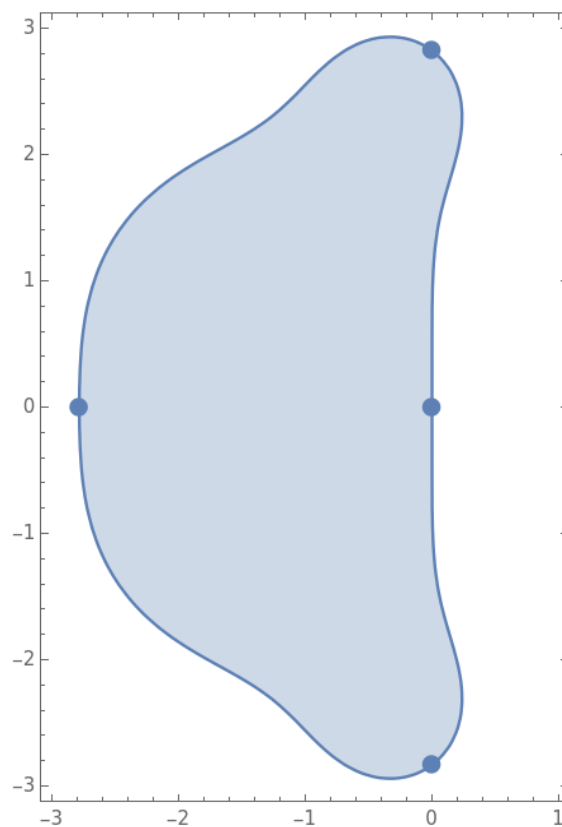
and

$$z = i2\sqrt{2}.$$

Plotting all of these points marks out the rough region below



We can compare this rough region with Mathematica's region plot as



(2) One seeks the solution of the eigenvalue problem

$$\frac{d}{dx} \left(\frac{1}{1+x} \frac{dy}{dx} \right) + \lambda y = 0$$

with boundary conditions $y(0) = y(1) = 0$. We want to find the λ such that the initial conditions $y(0) = 0$ and $y'(0) = 1$ leads to the boundary condition $y(1) = 0$ being met. Using my Richardson's extrapolation and trapezoidal method code from the last assignment, we can introduce the bisection method to find the desired λ . The bisection method is super easy to implement (*my code is attached at the end*), we just need to choose a starting search range. In this case, we will start with the range $[6.7, 6.8]$. Then, because our ODE is second order, we will need to reduce the ODE into a system of first order ODEs. Reducing this ODE yields the system

$$\begin{aligned} y' &= u \\ u' &= \frac{1}{1+x}u - (1+x)y \end{aligned}$$

with initial conditions $y(0) = 0$ and $u(0) = 1$. Then, plugging this system and search range into my code finds λ to be

$$\lambda = 6.773873469310571$$

which, when compared to Mathematica's solution, has a relative error of

$$1.4423 \cdot 10^{-15}.$$

So, it appears my code is working well and is also quite fast!

Code Used

Note: some of the symbols are missing in my code snippet because \LaTeX does not support all unicode characters.

```

1  #=
2  # 2x2 First Order ODE Solver Using Trapezoidal rule
3  #
4  # Author: Caleb Jacobs
5  # Date last modified: 03-04-2022
6  =#
7
8  using Plots
9  using SpecialFunctions
10 using ForwardDiff
11 using LinearAlgebra
12
13 # First order Bessel Equation system
14 function f(t, y)
15     # Use approximation if we are near the singularity t = 0
16     if t < 1e-10
17         return [y[2], -3*t/8] # Higher order terms + 5*(t^3)/96 - 7*(t^5)/3072]
18     else
19         return [y[2], ((1 - t^2)*y[1] - t*y[2]) / (t^2)]
20     end

```

```
21 end
22
23 # Tricky eigenvalue system
24 h(x, y, ) = [y[2], y[2] / (1 + x) - (1 + x) * * y[1]]
25
26 # Evaluate system at specified eigenvalue
27 function H()
28     htmp(x, y) = h(x, y, )
29     richTrap(htmp, 0, 1, [0, 1], n = 1, rn = 9)
30 end
31
32 # Newton method system solver
33 function newton(f; maxIts = 100, = 1e-8, y0 = [0, 0])
34     y = y0 # Initial guess
35
36     for i = 1 : maxIts
37         J = ForwardDiff.jacobian(f, y) # Get jacobian
38         ynew = y - (J \ f(y)) # Find next iterate
39
40         # Check for convergence
41         if norm(y - ynew) <=
42             y = ynew
43
44         return y
45     end
46
47     y = ynew # Pass to next iteration
48 end
49
50 return y
51 end
52
53 # Trapezoidal rule
54 function trapz(f, a, y0, h, n)
55     yi = y0 # Initial conditions
56     ti = a # Initial time
57     tf = a + h # First time step
58
59     # Run trapezoidal until desired time
60     for i = 1 : n
61         # Current trapezoidal equation
62         g(y) = y - (yi + h * (f(ti, yi) + f(tf, y)) / 2)
63
64         yi = newton(g) # Solve trapezoidal equation
65
66         ti = tf # Store new time
67         tf = ti + h # Compute next time
68     end
69
70     return yi
71 end
```

```
72
73 # Trapezoidal rule with Richardson Extrapolation
74 function richTrap(f, a, b, y0; n = 1, rn = 1)
75     h = (b - a) / n                # Compute time step
76
77     r = zeros(Float64, rn, rn)      # Initialize richardson matrix
78     sol = trapz(f, a, y0, h, n)     # Get initial solution
79     r[1, 1] = sol[1]                # Store initial solution
80
81     # Begin Richardson exptrapolation
82     for i = 1 : rn - 1
83         h /= 2                      # Half time step size
84         n *= 2                      # Double number of step to take
85
86         sol = trapz(f, a, y0, h, n) # Get solution with current step size
87         r[i + 1, 1] = sol[1]        # Store solution
88
89         # Compute richardson exptrapolation with current data
90         for j = 1 : i
91             r[i + 1, j + 1] = ((4^j) * r[i + 1, j] - r[i, j]) / (4^j - 1)
92         end
93     end
94
95     return r[rn, rn]
96 end
97
98 # Bisection method for root finding
99 function bisect(f, a, b; maxIts = 100, tol = 1e-8)
100     # Check required conditions
101     if a > b
102         return NaN
103     end
104
105     fa = f(a)                # Left function value
106     fb = f(b)                # Right function value
107
108     if (sign(fa) == sign(fb))
109         return NaN
110     end
111
112     c = 0.0                  # Initialize solution
113
114     # Begin bisecting
115     for n = 1 : maxIts
116         c = (a + b) / 2
117
118         fc = f(c)
119
120         # Check for convergence
121         if abs(fc) < tol || (b - a) / 2 < tol
122             return c
```

```
123         end
124
125         # Check for which side to cut interval
126         if sign(fc) == sign(fa)
127             a = c
128             fa = fc
129         else
130             b = c
131             fb = fc
132         end
133     end
134
135     display("Convergence never met, found:")
136     c
137 end
138
139 # besselJ = richTrap(f, 0, 3*, [0,1/2], n = 40, rn = 10)
140 # display(besselJ)
141 # display(besselJ - besselj(1, 3*))
142
143 sol = bisect(H, 6.7, 6.8, = 1e-15)
144 tru = 6.773873469310561
145 display(sol)
146 display(abs(tru - sol) / tru)
```
