Problems

(1) Implement the trapezoidal rule to solve the initial value problem

$$y' = f(t, y)$$

where

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, f(t, y) = \begin{pmatrix} f_1(t, y) \\ f_2(t, y) \end{pmatrix}, y(0) = y_0.$$

Use repeated Richardson extrapolation to improve the results. Then, use your code to solve

$$\begin{cases} t^2 y'' + t y' + (t^2 - 1) y = 0, t \in [0, 3\pi] \\ y(0) = 0, \quad y'(0) = \frac{1}{2} \end{cases}$$

Use repeated Richardson extrapolation to compute $y(3\pi)$ with 10 digits of accuracy.

First, let's reduce our ODE to a system of first order ODEs as

$$\mathbf{y}' = \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ \frac{1}{t^2}((1 - t^2) y_1 - t y_2) \end{pmatrix}$$

where $y_1 = y$ and $y_2 = y'$. Now, notice our ODE has a singularity at $t = 0$. To get around this, we can simply expand our solution about $t = 0$ to get the approximate system

$$\mathbf{y}' = \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ -\frac{3}{8} t + \mathcal{O}(t^2) \end{pmatrix}.$$

So, to use these systems in my code, I will use my code to solve the second system when we are near $t = 0$ to avoid the singularity and then use the first system when we are away form the singularity. Doing so, my code gives me

$$y(3\pi) = 0.17672519830082117$$

which is accurate to at least 10 digits of the Bessel function of the first kind of order one at $t = 3\pi$!

My code is given at the end of the document.

(2) Show that the two step method

$$y_{n+1} = \frac{1}{2}(y_n + y_{n-1}) + \frac{h}{4}\big(4f(t_{n+1}, y_{n+1}) - f(t_n, y_n) + 3f(t_{n-1}, y_{n-1})\big)$$

is second order.

$$\begin{aligned}
y(t_{n+1}) - \frac{1}{2}(y(t_n) + y(t_{n-1})) - \frac{h}{4}\big(4f(t_{n+1}, y(t_{n+1})) - f(t_n, y(t_n)) + 3f(t_{n-1}, y(t_{n-1}))\big) &= \\
y(t_n) + hy'(t_n) + \frac{1}{2}h^2 y''(t_n) + \mathcal{O}(h^3) - \\
\frac{1}{2}(y(t_n) + y(t_n) - hy'(t_n) + \frac{1}{2}h^2 y''(t_n) + \mathcal{O}(h^3)) - \\
\frac{h}{4}(4(y'(t_n) + hy''(t_n) + \mathcal{O}(h^2)) - y'(t_n) + 3(y'(t_n) - hy''(t_n) + \mathcal{O}(h^2))) &= \\
y(t_n) - y(t_n) + \mathcal{O}(h^3) + \\
h\left(y'(t_n) + \frac{1}{2}y'(t_n) - y'(t_n) + y'(t_n) + \frac{3}{4}y'(t_n)\right) + \mathcal{O}(h^3) + \\
h^2\left(\frac{1}{2}y''(t_n) - \frac{1}{4}y''(t_n) - y''(t_n) + \frac{3}{4}y''(t_n)\right) + \mathcal{O}(h^3) &= \mathcal{O}(h^3).
\end{aligned}$$

So, the method is order two.

(3) Determine order of the multistep method

$$y_{n+1} = 4y_n - 3y_{n-1} - 2hf(t_{n-1}, y_{n-1})$$

and illustrate with an example that the method is unstable.

$$\begin{aligned}
y(t_{n+1}) - 4y(t_n) + 3y(t_{n-1}) + 2hf(t_{n-1}, y(t_{n-1}) &= \\
y(t_n) + hy'(t_n) + \frac{1}{2}h^2y''(t_n) + \frac{1}{6}h^3y'''(t_n) + \mathcal{O}(h^4) &- \\
4y(t_n) + 3(y(t_n) - hy'(t_n) + \frac{1}{2}h^2y''(t_n) - \frac{1}{6}h^3y'''(t_n)) + \mathcal{O}(h^4) &+ \\
2h(y'(t_n) - hy''(t_n) + \frac{1}{2}h^2y'''(t_n) + \mathcal{O}(h^3)) &= \\
\frac{1}{6}h^3y'''(t_n) + \mathcal{O}(h^4) &= \mathcal{O}(h^3).
\end{aligned}$$

So, the method is order two.

A nice and simple example where the multistep is method is unstable is in the autonomous ODE

$$\begin{cases} y' = y(2-y) \\ y(0) = 1 \end{cases}$$

which should have a solution that just tends to $y = 2$ as t increases. Furthermore, even if the solution over shoots $y = 2$, we should expect the ODE to pull the equation back down to 2. If we start the multistep method with $y_0 = 0$, $y_1 = 1$, and $h = 0.1$ we get

$$\begin{aligned}
y_2 &= 4 \\
y_3 &= 12.8 \\
y_4 &= 40.8 \\
y_5 &= 152.448
\end{aligned}$$

and so on with a number that just keeps growing even though our ODE has a stable equilibrium at $y = 2$. Even with very tiny step sizes, our solution still blows up because the $4y_n$ term dominates the growth of the solution. One last thing, even if we start on the stable equilibrium, this multistep method will pull away from the stable equilibrium to infinity.

(4) Show that the multistep method

$$y_{n+3} + a_2 y_{n+2} + a_1 y_{n+1} + a_0 y_n = h(b_2 f(t_{n+2}, y_{n+2}) + b_1 f(t_{n+1}, y_{n+1}) + b_0 f(t_n, y_n))$$

is fourth order only if $a_0 + a_2 = 8$ and $a_1 = -9$. Deduce that this method cannot be both fourth order and convergent.

$$\begin{aligned}
y_{n+3} + a_2 y_{n+2} + a_1 y_{n+1} + a_0 y_n - h(b_2 f(t_{n+2}, y_{n+2}) - b_1 f(t_{n+1}, y_{n+1}) - b_0 f(t_n, y_n)) = \\
y(t_n) + 3hy'(t_n) + \frac{1}{2}3^2 h^2 y''(t_n) + \frac{1}{6}3^3 h^3 y'''(t_n) + \frac{1}{24}3^4 h^4 y^{(4)}(t_n) + \mathcal{O}(h^5) + \\
a_2 y(t_n) + 2a_2 hy'(t_n) + \frac{1}{2}2^2 a_2 h^2 y''(t_n) + \frac{1}{6}2^3 a_2 h^3 y'''(t_n) + \frac{1}{24}2^4 a_2 h^4 y^{(4)}(t_n) + \mathcal{O}(h^5) + \\
a_1 y(t_n) + a_1 hy'(t_n) + \frac{1}{2}a_1 h^2 y''(t_n) + \frac{1}{6}a_1 h^3 y'''(t_n) + \frac{1}{24}a_1 h^4 y^{(4)}(t_n) + \mathcal{O}(h^5) + \\
a_0 y(t_n) + \\
-b_2 hy'(t_n) - 2b_2 h^2 y''(t_n) - \frac{1}{2}2^2 b_2 h^3 y'''(t_n) - \frac{1}{6}2^3 b_2 h^4 y^{(4)}(t_n) + \mathcal{O}(h^5) + \\
-b_1 hy'(t_n) - b_1 h^2 y''(t_n) - \frac{1}{2}b_1 h^3 y'''(t_n) - \frac{1}{6}b_1 h^4 y^{(4)}(t_n) + \mathcal{O}(h^5) + \\
-b_0 hy'(t_n) = \\
(1 + a_2 + a_1 + a_0) y(t_n) + \mathcal{O}(h^5) + \\
(3 + 2a_2 + a_1 - b_2 - b1 - b_0) hy'(t_n) + \mathcal{O}(h^5) + \\
\left(\frac{1}{2}3^2 + \frac{1}{2}2^2 a_2 + \frac{1}{2}a_1 - 2b_2 - b_1\right) h^2 y''(t_n) + \mathcal{O}(h^5) + \\
\left(\frac{1}{6}3^3 + \frac{1}{6}2^3 a_2 + \frac{1}{6}a_1 - \frac{1}{2}2^2 b_2 - \frac{1}{2}b_1\right) h^3 y'''(t_n) + \mathcal{O}(h^5) + \\
\left(\frac{1}{24}3^4 + \frac{1}{24}2^4 a_2 + \frac{1}{24}a_1 - \frac{1}{6}2^3 b_2 - \frac{1}{6}b_1\right) h^4 y^{(4)}(t_n) + \mathcal{O}(h^5)
\end{aligned}$$

which implies our multistep method is only fourth order if our coefficients solve the system

$$\begin{aligned}
1 + a_2 + a_1 + a_0 &= 0 \\
3 + 2a_2 + a_1 - b_2 - b1 - b_0 &= 0 \\
\frac{1}{2}3^2 + \frac{1}{2}2^2 a_2 + \frac{1}{2}a_1 - 2b_2 - b_1 &= 0 \\
\frac{1}{6}3^3 + \frac{1}{6}2^3 a_2 + \frac{1}{6}a_1 - \frac{1}{2}2^2 b_2 - \frac{1}{2}b_1 &= 0 \\
\frac{1}{24}3^4 + \frac{1}{24}2^4 a_2 + \frac{1}{24}a_1 - \frac{1}{6}2^3 b_2 - \frac{1}{6}b_1 &= 0.
\end{aligned}$$

Reducing this system yields

$$\begin{aligned}
a_0 + a_2 &= 8 \\
a_1 &= -9 \\
-\frac{1}{3}a_2 + b_2 &= 3 \\
-\frac{4}{3}a_2 + b_1 &= -6 \\
-\frac{1}{3}a_2 + b_0 &= -3.
\end{aligned}$$

So, for our method to have any hope of being fourth order, we must have $a_0 + a_2 = 8$, $a_1 = -9$, and a few other conditions. Now, the polynomials associated with our multistep method are given by

$$\rho(\omega) = a_0 + a_1\omega + a_2\omega^2 + \omega^3$$

and

$$\sigma(\omega) = b_0 + b_1\omega + b_2\omega^2.$$

Then, if we want our method to be fourth order, our $\rho(\omega)$ becomes

$$\rho(\omega) = 8 - a_2 - 9\omega + a_2\omega^2 + \omega^3.$$

So, finding the roots of $\rho(\omega)$ yields the roots

$$\begin{aligned}
\omega_1 &= 1 \\
\omega_2 &= \frac{1}{2}\left(-a_2 - 1 - \sqrt{a_2^2 - 2a_2 + 33}\right) \\
\omega_3 &= \frac{1}{2}\left(-a_2 - 1 + \sqrt{a_2^2 + 2a_2 + 33}\right).
\end{aligned}$$

From these, we can clearly see that ω_1 is on the unit disk and simple. However, there is no a_2 such that both ω_2 and ω_3 are within the unit disk and so there no a_2 such that our polynomial satisfies the root condition. Therefore, by the Dahlquist equivalence theorem, our fourth order multistep method cannot converge.

Code Used

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

```
#=
# 2×2 First Order ODE Solver Using Trapezoidal rule
#
# Author: Caleb Jacobs
# Date last modified: 14-Mar-2022
=#

using Plots
using SpecialFunctions
using ForwardDiff
using LinearAlgebra

# First order Bessel Equation system
function f(t, y)
    # Use approximation if we are near the singularity t = 0
    if t < 1e-10
        return [y[2], -3*t/8] # Higher order terms + 5*(t^3)/96 - 7*(t^5)/3072]
    else
        return [y[2], ((1 - t^2)*y[1] - t*y[2]) / (t^2)]
    end
end

# Newton method system solver
function newton(f; maxIts = 100,   = 1e-8, y0 = [0, 0])
    y = y0                      # Initial guess

    for i   1 : maxIts
        J = ForwardDiff.jacobian(f, y)  # Get jacobian
        ynew = y - (J \ f(y))           # Find next iterate

        # Check for convergence
        if norm(y - ynew) <=
            y = ynew

            return y
        end

        y = ynew                        # Pass to next iteration
    end

    return y
end

# Trapezoidal rule
function trapz(f, a, y0, h, n)
    yi = y0             # Initial conditions
    ti = a              # Initial time
```

```
    tf = a + h          # First time step

    # Run trapezoidal until desired time
    for i = 1 : n
        # Current trapezoidal equation
        g(y) = y - (yi + h * (f(ti, yi) + f(tf, y)) / 2)

        yi = newton(g)  # Solve trapezoidal equation

        ti = tf         # Store new time
        tf = ti + h     # Compute next time
    end

    return yi
end

# Trapezoidal rule with Richardson Extrapolation
function richTrap(f, a, b, y0; n = 1, rn = 1)
    h = (b - a) / n                    # Compute time step

    r = zeros(Float64, rn, rn)         # Initialize richardson matrix
    sol = trapz(f, a, y0, h, n)        # Get initial solution
    r[1, 1] = sol[1]                   # Store initial solution

    # Begin Richardson exptrapolation
    for i = 1 : rn - 1
        h /= 2                         # Half time step size
        n *= 2                         # Double number of step to take

        sol = trapz(f, a, y0, h, n) # Get solution with current step size
        r[i + 1, 1] = sol[1]           # Store solution

        # Compute richardson exptrapolation with current data
        for j = 1 : i
            r[i + 1, j + 1] = ((4^j) * r[i + 1, j] - r[i, j]) / (4^j - 1)
        end
    end

    return r[rn, rn]
end

besselJ = richTrap(f, 0, 3*, [0,1/2], n = 40, rn = 10)
display(besselJ)
display(besselJ - besselj(1, 3*))
```