Project 2

Name: Wang Fangyu Student ID: MAT2309449 Marks:____

Computer Project (due at 11:55 pm on May 22).

1. (Page 336, 6.6 Computer Problems, 1(a)) Apply Backward Euler, using Newton's Method as a solver, for the initial value problem

$$\begin{cases} y' = y^2 - y^3 \\ y(0) = 1/2 \end{cases}$$

with $t \in [0, 20]$. Which of the equilibrium solutions are approached by the approximate solution? Apply forward Euler's Method. From your numerical test, for what approximate range of Δt can forward Euler solution converge to the equilibrium? (You can find this approximate range by performing some numerical tests.) Plot approximate solutions given by Backward Euler, and by forward Euler with an excessive step size. [5 marks]

Solution

By Backward Euler method, we have

$$y_{n+1} = y_n + \Delta t \cdot y'_{n+1}.$$

Subsitute $y' = y^2 - y^3$ we have

$$y_{n+1} = y_n + \Delta t \cdot (y_{n+1}^2 - y_{n+1}^3).$$

To solve for y_{n+1} at each time step, define $F(y_{n+1})$ as

$$F(y_{n+1}) = y_{n+1} - y_n - \Delta t(y_{n+1}^2 - y_{n+1}^3) = 0.$$

Its derivative with respect to y_{n+1} is $F'(y_{n+1}) = 1 - \Delta t(2y_{n+1} - 3y_{n+1}^2)$.

Using Newton's Method as the solver with the following iteration we have:

$$y_{n+1}^{(k+1)} = y_{n+1}^{(k)} - \frac{F(y_{n+1}^{(k)})}{F'(y_{n+1}^{(k)})},$$

where $y_{n+1}^{(k)}$ is the k-th iteration of y_{n+1} and let $y_{n+1}^{(0)} = y_n$.

To find the equilibrium solutions are approached by the approximate solution, set $y' = 0$, we get $y^2 - y^3 = 0$, which yields equilibrium solutions $y = 0$ and $y = 1$. With the initial condition $y(0) = 1/2$, the initial derivative is $y'(0) = 1/8 > 0$. Since the function is strictly

increasing from $y = 0.5$, the approximate solution approaches the equilibrium solution $y = 1$.

By Forward Euler method we have $y_{n+1} = y_n + \Delta t(y_n^2 - y_n^3)$. The following is the Julia output.

Numerical Test Output:

```
dt = 0.5        -> Status: Converged to 1
dt = 0.6        -> Status: Converged to 1
dt = 0.7        -> Status: Converged to 1
dt = 0.8        -> Status: Converged to 1
dt = 0.9        -> Status: Converged to 1
dt = 1.0        -> Status: Converged to 1
dt = 1.1        -> Status: Converged to 1
dt = 1.2        -> Status: Converged to 1
dt = 1.3        -> Status: Converged to 1
dt = 1.4        -> Status: Converged to 1
dt = 1.5        -> Status: Converged to 1
dt = 1.6        -> Status: Converged to 1
dt = 1.7        -> Status: Converged to 1
dt = 1.8        -> Status: Converged to 1
dt = 1.9        -> Status: Converged to 1
dt = 2.0        -> Status: Oscillating
dt = 2.1        -> Status: Oscillating
dt = 2.2        -> Status: Oscillating
dt = 2.3        -> Status: Oscillating
dt = 2.4        -> Status: Oscillating
dt = 2.5        -> Status: Oscillating
```

Through numerical testing, the Forward Euler solution can converge to the equilibrium $y = 1$ only when the step size Δt is within the approximate range of $0 < \Delta t < 2$.

Set the step size ($\Delta t = 2.1$), the following figure shows the approximate solutions given by Backward Euler method, and by Forward Euler method.

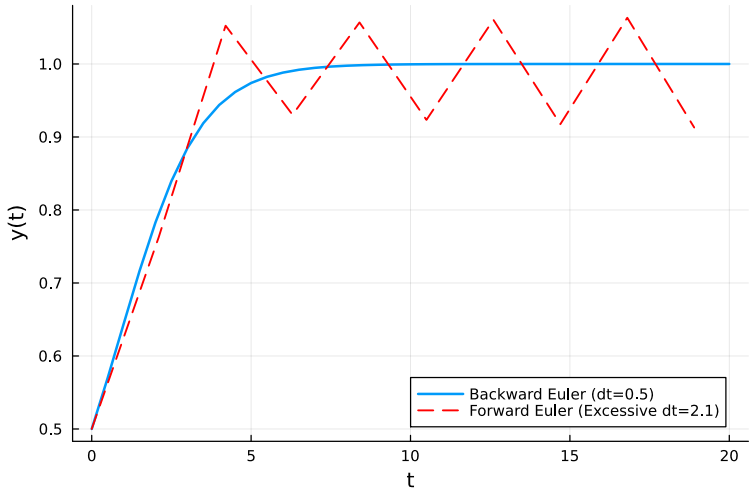

FIGURE 1

Julia Code

```
using Plots

f(y) = y^2 - y^3
df(y) = 2y - 3y^2

function backward_euler(dt, t_end; y0=0.5, max_iter=20, tol=1e-8)
    t = 0:dt:t_end
    N = length(t)
    y = zeros(N)
    y[1] = y0

    for n in 1:(N-1)
        y_new = y[n]
        for _ in 1:max_iter
            F = y_new - y[n] - dt * f(y_new)
            dF = 1.0 - dt * df(y_new)
            step = F / dF
            y_new -= step
            if abs(step) < tol
                break
            end
        end
        y[n+1] = y_new
    end
```

```
    return t, y
end

function forward_euler(dt, t_end; y0=0.5)
    t = 0:dt:t_end
    N = length(t)
    y = zeros(N)
    y[1] = y0
    for n in 1:(N-1)
        y[n+1] = y[n] + dt * f(y[n])
    end
    return t, y
end

t_end = 20.0

t_be, y_be = backward_euler(0.5, t_end)
p1 = plot(t_be, y_be, label="Backward Euler (dt=0.5)", lw=2, xlabel="t", ylabel="y(t)",
          title="", legend=:bottomright)

t_fe_excessive, y_fe_excessive = forward_euler(2.1, t_end)
plot!(p1, t_fe_excessive, y_fe_excessive, label="Forward Euler (Excessive dt=2.1)",
      lw=1.5, linestyle=:dash, color=:red)

println("")
for test_dt in [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8,
     1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5]
    _, y_fe_test = forward_euler(test_dt, 50.0)
    final_y = y_fe_test[end]
    status = abs(final_y - 1.0) < 1e-2 ? "Converged to 1" :
             (isnan(final_y) || isinf(final_y) || abs(final_y) > 10) ? "Diverged" : "Osc
    println("dt = $test_dt \t-> Status: $status")
end

display(p1)
savefig(p1, "1.pdf")
```

2. (Page 346, 6.7 Computer Problems, 5.) Plot the Adams-Bashforth Three-Step Method approximate solution on $[0, 1]$ for the differential equation

$$y' = 1 + y^2$$

and initial condition (a) $y(0) = 0$ (b) $y(0) = 1$, along with the exact solution is $y = \tan(t)$ for $y_0 = 0$ and $y = \tan\left(t + \frac{\pi}{4}\right)$ for $y_0 = 1$. Use step sizes $\Delta t = 0.1$ and 0.05. (You can use a second order Runge-Kutta method to prepare the data to start your iteration.)

Please note that for (a), the error at $\Delta t = 0.05$ may not be 1/8 of the error at $\Delta t = 0.1$ because error $= \mathcal{O}(\Delta t^3)$ could be given by, say, error $= 0.1 \times \Delta t^3 + 100\Delta t^4$. In that case, you can only see 3rd order convergence when Δt is small enough so that $100\Delta t^4 \ll 0.1 \times \Delta t^3$. To check whether you code is correct or not, you should check if you can get 3rd order convergence when Δt is small enough. I will demonstrate that in my solution, even though you do not need to do that in your solution since it is not asked in the textbook.

For (b), please pay attention to what happens to the exact solution when $t \to \pi/4$ which happens before $t = 1$. So, don't be too surprised when you see something strange.

[5 marks]

Solution

We first use the 2nd-order Runge-Kutta method to compute y_1 and y_2 to start the Adams-Bashforth Three-Step Method. By 2nd-order Runge-Kutta method we have:

$$\begin{aligned} k_1 &= f(y_0) = 1 + y_0^2 \\ k_2 &= f(y_0 + \Delta t \cdot k_1) = 1 + (y_0 + \Delta t \cdot k_1)^2 \\ y_1 &= y_0 + \frac{\Delta t}{2}(k_1 + k_2) \end{aligned}$$

Follow by the same method we can also get y_2.

Then we can start our Adams-Bashforth Three-Step Method as:

$$y_{n+1} = y_n + \frac{\Delta t}{12}\left[23f(y_n) - 16f(y_{n-1}) + 5f(y_{n-2})\right]$$

Condition (a) The exact solution for this initial value problem is $y = \tan(t)$. Set $t = 1.0$ and calculate the error of AB3 with respect to $y = \tan(t)$ at t=1 for steps of 0.1 and 0.05 respectively:

- Error at $\Delta t = 0.1$: ≈ 0.01403
- Error at $\Delta t = 0.05$: ≈ 0.00256

The actual ratio of the errors is $\frac{0.01403}{0.00256} \approx 5.47$. This ratio is less than 8 ($2^3 = 8$). This is because the chosen step sizes ($\Delta t = 0.1$ and 0.05) are not sufficiently small, thus the higher-order error terms (e.g., $\mathcal{O}(\Delta t^4)$) still have a sinificant influence to the total error.

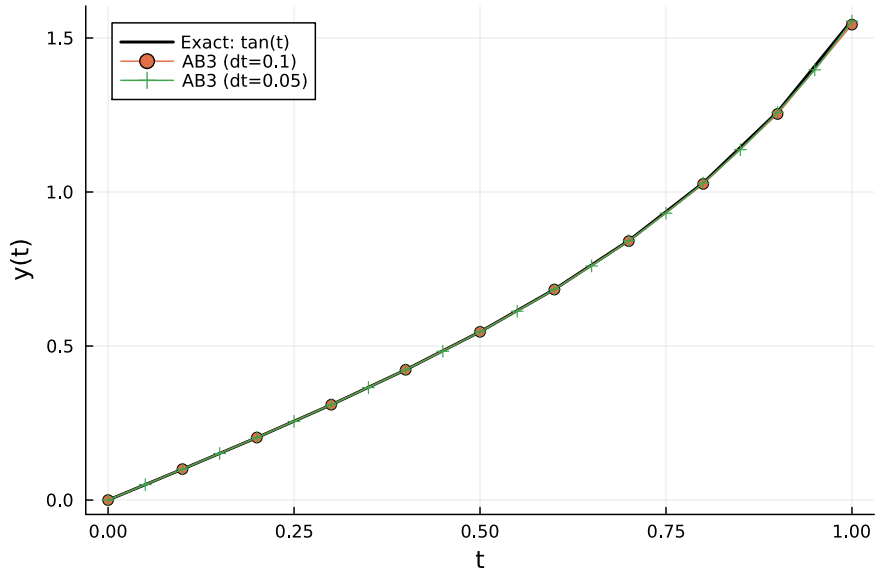

FIGURE 2. AB3 approximate solution and the exact solution $y = \tan(t)$ for $y(0) = 0$.

Condition (b) The exact solution here is $y = \tan(t + \frac{\pi}{4})$. As $t \to \pi/4 \approx 0.785$, $y = \tan(t + \frac{\pi}{4}) \to \infty$. As we can see in Figure 3, the numerical solution follows the exact curve closely until it reaching $t = 1$.

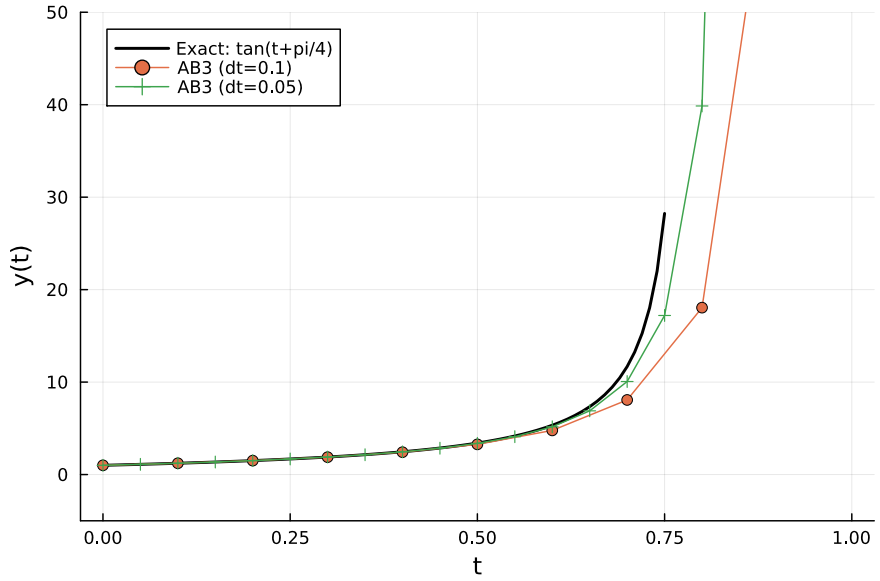

FIGURE 3. AB3 approximate solution and the exact solution $y = \tan(t + \frac{\pi}{4})$ for $y(0) = 1$.

Julia Code

```
using Plots

f(y) = 1.0 + y^2
```

```
exact_a(t) = tan(t)
exact_b(t) = tan(t + pi/4)
function rk2_step(y, dt)
    k1 = f(y)
    k2 = f(y + dt * k1)
    return y + (dt / 2.0) * (k1 + k2)
end

function ab3_solve(y0, dt, t_end)
    t = 0.0:dt:t_end
    N = length(t)
    y = zeros(N)
    y[1] = y0

    if N > 1
         y[2] = rk2_step(y[1], dt)
    end
    if N > 2
         y[3] = rk2_step(y[2], dt)
    end

    for n in 3:(N-1)
        y[n+1] = y[n] + (dt / 12.0) * (23*f(y[n]) - 16*f(y[n-1]) + 5*f(y[n-2]))
    end
    return t, y
end
t_a_01, y_a_01 = ab3_solve(0.0, 0.1, 1.0)
t_a_005, y_a_005 = ab3_solve(0.0, 0.05, 1.0)

exact_val_a = exact_a(1.0)
err_01 = abs(y_a_01[end] - exact_val_a)
err_005 = abs(y_a_005[end] - exact_val_a)
ratio = err_01 / err_005

println("Error at dt=0.1:  ", err_01)
println("Error at dt=0.05: ", err_005)
println("Ratio of errors:  ", ratio)
```

```
p_a = plot(xlabel="t", ylabel="y(t)", legend=:topleft)
plot!(p_a, t_a_01, exact_a.(t_a_01), label="Exact: tan(t)", lw=2, color=:black)
plot!(p_a, t_a_01, y_a_01, label="AB3 (dt=0.1)", marker=:circle)
plot!(p_a, t_a_005, y_a_005, label="AB3 (dt=0.05)", marker=:cross)
savefig(p_a, "2a.pdf")


t_b_01, y_b_01 = ab3_solve(1.0, 0.1, 1.0)
t_b_005, y_b_005 = ab3_solve(1.0, 0.05, 1.0)

p_b = plot(xlabel="t", ylabel="y(t)", legend=:topleft, ylims=(-5, 50))
t_exact_b = 0:0.01:0.75
plot!(p_b, t_exact_b, exact_b.(t_exact_b), label="Exact: tan(t+pi/4)", lw=2, color=:black)
plot!(p_b, t_b_01, y_b_01, label="AB3 (dt=0.1)", marker=:circle)
plot!(p_b, t_b_005, y_b_005, label="AB3 (dt=0.05)", marker=:cross)
savefig(p_b, "2b.pdf")
```