Homework 2

AM213B

Kevin Silberberg

2025-04-18

Problem 1

Suppose k satisfies the equation

$$k = he^{1+k} \quad \text{where } k \text{ is small} \tag{1}$$

Recall the approach of iterative expansion we used in lecture.
Start with $k = O(h)$. Expand k iteratively into $k = a_1 h + a_2 h^2 + \cdots$.
Find the coefficients a_1 and a_2.

Solution

Let us plug in $k = a_1 h + a_2 h^2$ into $k = he^{1+k}$.

$$\begin{aligned} k &= he^{1+k} & (2) \\ a_1 h + a_2 h^2 &= he^{1+a_1 h + a_2 h^2} & (3) \\ &= he^1 \cdot e^{a_1 h} \cdot e^{a_2 h^2} & (4) \end{aligned}$$

Recall that the function e^x can be expressed as the infinite sum $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$
thus we have

$$\begin{aligned} a_1 h + a_2 h^2 &= he \cdot (1 + a_1 h + \cdots) \cdot (1 + a_2 h^2 + \cdots) & (5) \\ &= e\left(h + h^2 a_1 + h^3 a_2 + h^4 a_1 a_2 + \cdots\right) & (6) \end{aligned}$$

Collecting terms we have,

$$\begin{aligned} O(h) &: a_1 = e & (7) \\ O(h^2) &: a_2 = a_1 e = e \cdot e = e^2 & (8) \end{aligned}$$

Problem 2

Consider the Runge-Kutta (RK) method specified by the Butcher array,

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{9}$$

Write out the method in the form of $k_1 = \cdots$, $k_2 = \cdots$, and

$$u_{n+1} = u_n + \cdots \tag{10}$$

Solution

Let us write out the method corresponding to the Butcher array in (9), notice that the method is implicit since the $\mathbf{A}$ matrix is not strictly lower triangular.

The general form of implicit Runge-Kutta methods is as follows,

$$k_i = hf\left(u_n + \sum_{j=1}^{p} a_{ij}k_j, t_n + c_i h\right), \quad i = 1, \cdots, p \tag{11}$$

$$u_{n+1} = u_n + \sum_{i=1}^{p} b_i k_i \tag{12}$$

here,

$$p = 2, \quad \mathbf{A} = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{13}$$

It is a two-stage, second order, implicit method.

$$\begin{cases} u_{n+1} & = u_n + \frac{1}{2}(k_1 + k_2) \\ k_1 & = hf(u_n, t_n) \\ k_2 & = hf\left(u_n + \frac{1}{2}(k_1 + k_2), t_n + h\right) \end{cases} \tag{14}$$

Part 1

By expanding k_1 and k_2 and $u(t_k + h)$, show that the local truncation error is of order $O(h^3)$

Solution

The method (14) is an approximate solution to the system $\dot{\mathbf{u}} = f(\mathbf{u}, t)$ or

$$u_{n+1} \approx u(t_n + h) \tag{15}$$

Let us taylor expand the true solution $u(t_n + h)$ in order to characterize the local truncation error, or the error accrued by making a single prediction one time step h forward in time.

$$u(t_n+h) \approx u(t_n) + h\dot{u}(t_n) + \frac{h^2}{2}\ddot{u}(t_n) + \frac{h^3}{6}\dddot{u}(t_n) + \cdots \tag{16}$$

Let us write the terms $\dot{u}(t_n)$ and $\ddot{u}(t_n)$

$$\dot{u}(t_n) = f(u_n, t_n) = f_n \tag{17}$$

$$\ddot{u}(t_n) = \frac{d}{dt}f(u_n,t_n) = \left(\frac{\partial f(u_n,t_n)}{\partial u_n}\dot{u}(t_n) + \frac{\partial f(u_n,t_n)}{\partial t_n}\right) = \left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) \tag{18}$$

(16) becomes

$$u(t_n+h) \approx u(t_n) + hf_n + \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) + O(h^3) \tag{19}$$

The local truncation error of a Runge-Kutta method is defined as

$$e_n(h) = u(t_n+h) - u(t_n) - \sum_{i=1}^{p} b_i k_i \tag{20}$$

for our problem this is

$$e_n(h) = u(t_n+h) - u(t_n) - \frac{1}{2}k_1 - \frac{1}{2}k_2 \tag{21}$$

$$= u(t_n) + hf_n + \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) + O(h^3) - u(t_n) - \frac{1}{2}(k_1+k_2) \tag{22}$$

$$= hf_n + \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) + O(h^3) - \frac{1}{2}(k_1+k_2) \tag{23}$$

So in order to show that the LTE for the method is $O(h^3)$ we just need to show that

$$\frac{1}{2}(k_1+k_2) = hf_n + \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) \tag{24}$$

Let us expand k_1 and k_2

for k_1 we have

$$k_1 = hf(u_n, t_n) = hf_n \tag{25}$$

for k_2 we have

$$k_2 = hf(u_n + \frac{1}{2}(k_1 + k_2), t_n + h) \tag{26}$$

Let $r = \frac{1}{2}(k_1 + k_2)$

We know that $k_1 = hf_n$ so we can express r as

$$r = \frac{1}{2}\left(hf_n + hf(u_n + \frac{1}{2}(k_1 + k_2), t_n + h)\right) \tag{27}$$

Let us taylor expand $f(u_n + r, t_n + h)$ around the point (u_n, t_n)

$$f(u_n + r, t_n + h) = f_n + r\frac{\partial f_n}{\partial u_n} + h\frac{\partial f_n}{\partial t_n} + O(r^2, h^2) \tag{28}$$

Substituting this into the expression (27) and solving for r we have

$$\begin{aligned} r &= \frac{1}{2}\left(hf_n + h\left(f_n + r\frac{\partial f_n}{\partial u_n} + h\frac{\partial f_n}{\partial t_n} + O(r^2, h^2)\right)\right) && (29)\\ &= hf_n + \frac{hr}{2}\frac{\partial f_n}{\partial u_n} + \frac{h^2}{2}\frac{\partial f_n}{\partial t_n} + O(r^2h, h^3) && (30)\\ r\left(1 - \frac{h}{2}\frac{\partial f_n}{\partial u_n}\right) &= hf_n + \frac{h^2}{2}\frac{\partial f_n}{\partial t_n} + O(r^2h, h^3) && (31)\end{aligned}$$

Recall that $\frac{1}{1-a} \to 1 + a$ as $a \to 0$. So when h is small we can approximate $\left(1 - \frac{h}{2}\frac{\partial f_n}{\partial u_n}\right)^{-1} \approx \left(1 + \frac{h}{2}\frac{\partial f_n}{\partial u_n}\right)$.

Thus solving for r we have,

$$\begin{aligned} r &= \left(1 + \frac{h}{2}\frac{\partial f_n}{\partial u_n}\right)\left(hf_n + \frac{h^2}{2}\frac{\partial f_n}{\partial t_n} + O(r^2h, h^3)\right) && (32)\\ &= hf_n + \frac{h^2}{2}\frac{\partial f_n}{\partial t_n} + \frac{h^2}{2}\frac{\partial f_n}{\partial u_n}f_n \underbrace{+ \frac{h^3}{4}\frac{\partial^2 f_n}{\partial t_n \partial u_n} + O(r^2h, h^3) + O(r^2h^2, h^4)}_{O(h^3)} && (33)\\ &= hf_n + \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) + O(h^3) && (34)\end{aligned}$$

Plugging this result back into (23) we have

$$\begin{aligned} e_n(h) &= hf_n + \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) + O(h^3) - hf_n - \frac{h^2}{2}\left(\frac{\partial f_n}{\partial u_n}f_n + \frac{\partial f_n}{\partial t_n}\right) - O(h^3) && (35)\\ &= O(h^3) - O(h^3) = O(h^3) && (36)\end{aligned}$$

Part 2

What is the order of the global error?

Solution

Given that the method is stable, the global error is one order lower than the local truncation error.

Thus the global error is

$$E_N(h) = O(h^2) \tag{37}$$

Problem 3

Use the 3-step Adams-Bashforth method to solve the following IVP,

$$\begin{cases} \ddot{y} + y = 0 \\ y(0) = 1, \quad \dot{y}(0) = 0 \end{cases} \tag{38}$$

Let us convert the 2nd order ODE into a system of first order DE's

Let $w_1 = y$ and $w_2 = \dot{y}$ such that (38) becomes

$$\begin{cases} \dot{w_1} & = w_2 \quad w_1(0) = 1 \\ \dot{w_2} & = -w_1 \quad w_2(0) = 0 \end{cases} \tag{39}$$

or in vector form

$$\dot{\mathbf{w}} = \begin{bmatrix} w_2 \\ -w_2 \end{bmatrix}, \quad \mathbf{w}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{40}$$

where the exact solution of the IVP is

$$\mathbf{w}(t) = \begin{bmatrix} \cos t \\ -\sin t \end{bmatrix} \tag{41}$$

Part 1

HINT: Use the exact solution for $\mathbf{w}(h)$ and $\mathbf{w}(2h)$ to get it started.

Use $h = 0.02$ and $h = 0.01$, respectively, to solve the IVP to $T = 10$.

Consider the true error $E_n(h) = \mathbf{w}_n(h) - \mathbf{w}(t_n), \quad t_n = nh$

Plot $\|E_n(h)\|$ vs t_n respectively for $h = 0.02$ and $h = 0.01$. Use the log-scale for $\|E_n(h)\|$.

Plot the two curves in ONE figure for comparison.

Solution

The AB3 method is as follows,

$$u_{k+1} = u_k + \frac{h}{12}\left(23f(u_k,t_k) - 16f(u_{k-1},t_{k-1}) + 5f(u_{k-2},t_{k-2})\right) \tag{42}$$

Let us write a code that implements this method

```
using LinearAlgebra
using StaticArrays
using GLMakie
function sol_prob3(t::Float64)
    return SVector{2, Float64}(cos(t), -sin(t))
end

function AB3_part1(f::Function, u0::SVector{2, Float64}, tspan::SVector{2, Float64}, h::Float64)
    a::Float64 = 0.08333333333333333
    t0, tf = tspan
    N = Int(floor((tf-t0)/h))
    t = Vector{Float64}(undef, N+1)
    u = Vector{SVector{2, Float64}}(undef, N+1)
    t[1] = t0; u[1] = u0
    t[2] = t[1] + h; u[2] = sol_prob3(h)
    t[3] = t[2] + h; u[3] = sol_prob3(2*h)
    for i = 3:N
        t[i+1] = t[i] + h
        u[i+1] = u[i] + h*a*(23.0*f(u[i], t[i]) - 16*f(u[i-1], t[i-1]) + 5*f(u[i-2], t[i-2]))
    end
    return t, u
end
```

```
AB3_part1 (generic function with 1 method)
```

Let us solve for the trajectories of the system and compute the global error over the time domain.

```
function prob3ODE(u::SVector{2, Float64}, t::Float64)
    return SVector{2, Float64}(u[2], -u[1])
end

function  prob3driver(AB3::Function, limits::Tuple)
    hs = SA[0.02, 0.01]; ts = SA[0.0, 10.0]; u0 = SA[1.0, 0.0]
    lbs = SA[L"$h = 0.02$", L"$h = 0.01$"]
    fig = Figure()
    ax = Axis(fig[1, 1],
              title = "Absolute value of true error for every time step ",
              xlabel = "time", ylabel = L"$log(||E_n(h)||)$",
              yscale=log10, limits = limits)
    for i in eachindex(hs)
        t, u = AB3(prob3ODE, u0, ts, hs[i])
        u_true = sol_prob3.(t)
        E_n = u_true .- u
        lines!(ax, t, norm.(E_n), label=lbs[i])
    end
    Legend(fig[1, 2], ax)
    fig
```

```
end
prob3driver(AB3_part1, (nothing, (1e-9, 1e-4)))
```

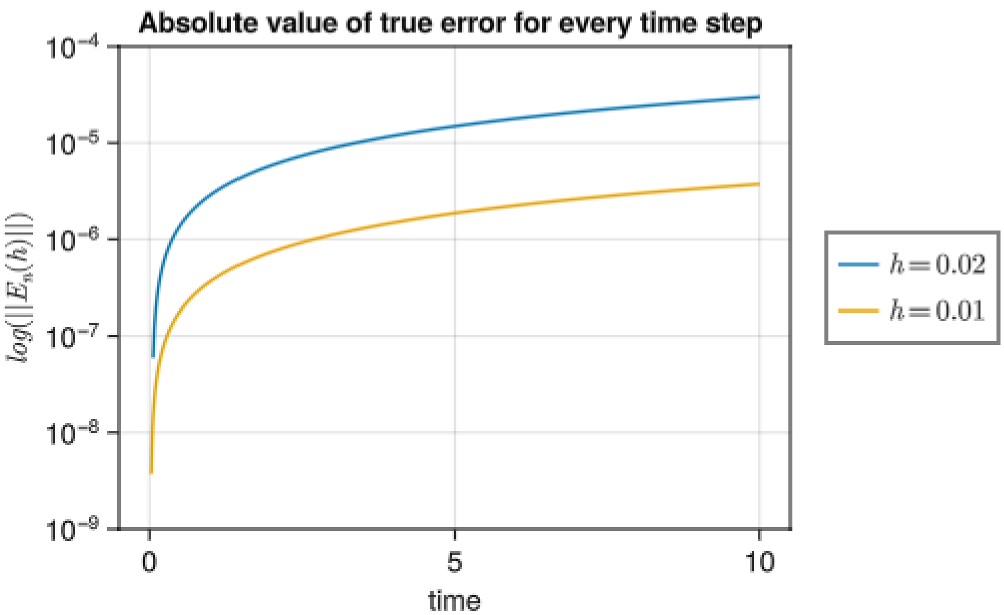


Figure 1: Plot of the true error on a log scale for every time step in the domain $T = [0, 10]$, for the temporal resolutions $h = 0.02$ in blue and $h = 0.01$ in yellow.

Part 2

HINT: Use the Euler method with the given h to calculate $\mathbf{w}(h)$ and $\mathbf{w}(2h)$

Repeat Part 1 with the new starting steps. Compare two figures.

Solution

Euler method is as follows,

$$u_{k+1} = u_k + hf(u_k, t_k) \tag{43}$$

let us define the augmented AB3 method

```
function AB3_part2(f::Function, u0::SVector{2, Float64}, tspan::SVector{2, Float64}, h::Float64)
    a::Float64 = 0.08333333333333333
    t0, tf = tspan
    N = Int(floor((tf-t0)/h))
    t = Vector{Float64}(undef, N+1)
    u = Vector{SVector{2, Float64}}(undef, N+1)
    t[1] = t0
    u[1] = u0
    for i = 1:2
        t[i + 1] = t[i] + h
        u[i + 1] = u[i] + h*f(u[i], t[i])
```

```
    end
    for i = 3:N
        t[i+1] = t[i] + h
        u[i+1] = u[i] + h*a*(23.0*f(u[i], t[i]) - 16*f(u[i-1], t[i-1]) + 5*f(u[i-2], t[i-2]))
    end
    return t, u
end
prob3driver(AB3_part2, (nothing, (1e-5, 1e-3)))
```

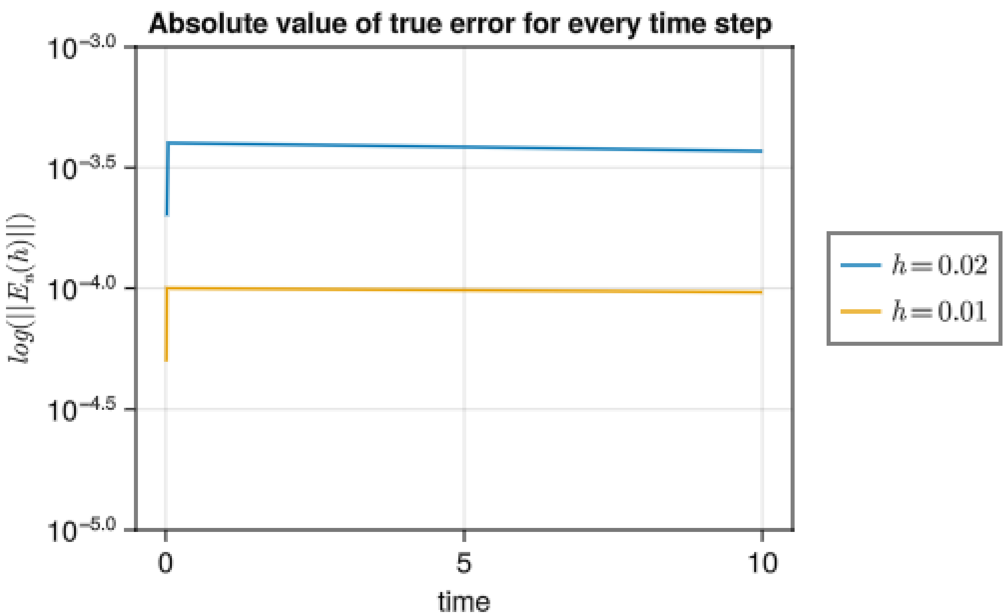


Figure 2: Same as the plot above, but replacing the exact solution for $w(h)$ and $w(2h)$ with the euler method.

Starting the solution to the AB3 method with the exact solution in Figure 1 results in the true error being improved by doubling the number of interpolating points, while starting the AB3 method with the euler method as in Figure 2 results in a negligible improvmentin the true error even with doubling the number of interpolating points.

Problem 4

Read the sample code on implementing RK4. Write your own code to implement RK4.

Algorithm 1 RK4 Algorithm

1: **for** $n = 0 : N - 1$ **do**
2: **for** $I = 1 : P$ **do**
3: calculate k_i
4: **end for**
5: $u_{n+1} = u_n + sum(b_i k_i)$
6: **end for**

Use RK4 to solve the following IVP

$$\begin{cases} \ddot{y} - \mu\left(2 - e^{\dot{y}^2}\right)\dot{y} + y = 0 \\ y(0) = y_0, \quad \dot{y} = v_0 \end{cases} \tag{44}$$

Before applying the numerical method, we need to convert this second order ODE to a system of first order DE's.

Let $w_1 = y$ and $w_2 = \dot{y}$ such that (44) becomes

$$\begin{cases} \dot{w_1} = w_2 \\ \dot{w_2} = \mu\left(2 - e^{w_2^2}\right)w_2 - w_1 \\ w_1(0) = y_0, \quad w_2(0) = v_0 \end{cases} \tag{45}$$

Use $y_0 = 3$ $v_0 = 0.5$, and $h = 0.025$. Solve the IVP to $T = 30$ respectively for $\mu = 0.5, \quad \mu = 2, \quad$ and $\quad \mu = 4$

Part 1

For each μ, plot $y(t)$ vs t and $\dot{y}(t)$ vs t in one figure.

Solution

The RK4 method is a 4-stage, 4th-order, explicit method defined as

$$u_{n+1} = u_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{46}$$

$$k_1 = hf(u_n, t_n) \tag{47}$$

$$k_2 = hf(u_n + \frac{1}{2}k_1, t_n + \frac{h}{2}) \tag{48}$$

$$k_3 = hf(u_n + \frac{1}{2}k_2, t_n + \frac{h}{2}) \tag{49}$$

$$k_4 = hf(u_n + k_3, t_n + h) \tag{50}$$

Let write a code that impliments the RK4 method as defined above

```
function RK4(f::Function, u0::SVector{2, Float64}, tspan::SVector{2, Float64}, h::Float64, p::Float64)
    a::Float64 = 0.16666666666666666
    b::Float64 = 0.3333333333333333
    t0, tf = tspan
    N = Int(floor((tf-t0)/h))
    t = Vector{Float64}(undef, N+1)
    u = Vector{SVector{2, Float64}}(undef, N+1)
    t[1] = t0
    u[1] = u0
    for i = 1:N
        k1 = h*f(u[i], p, t[i])
        k2 = h*f(u[i] + 0.5*k1, p, t[i] + 0.5*h)
        k3 = h*f(u[i] + 0.5*k2, p, t[i] + 0.5*h)
        k4 = h*f(u[i] + k3, p, t[i] + h)
        u[i+1] = u[i] + a*k1 + b*k2 + b*k3 + a*k4
        t[i+1] = t[i] + h
```

```
    end
    return t, u
end
```

```
RK4 (generic function with 1 method)
```

Let us write a driver code that solves the IVP

```
function prob4ODE(u::SVector{2, Float64}, p::Float64, t::Float64)
    return SVector{2, Float64}(u[2], p*(2.0 - exp(u[2]*u[2]))*u[2] - u[1])
end
function prob4part1()
    h = 0.025; u0 = SA[3.0, 0.5]; ts = SA[0.0, 30.0]; mus = SA[0.5, 2.0, 4.0]
    lbs = SA[L"$\mu = 0.5$", L"$\mu = 2.0$", L"$\mu = 4.0$"]
    fig = Figure(resolution=(650, 300))
    for i in eachindex(mus)
        ax = Axis(fig[1, i], title = lbs[i],
                  xlabel = L"\text{time}", ylabel = L"$y(t)$", limits=(nothing, (-2.1, 3.2)))
        t, u = RK4(prob4ODE, u0, ts, h, mus[i])
        lines!(ax, t, [u[k][1] for k in eachindex(t)], label = L"$y$")
        lines!(ax, t, [u[k][2] for k in eachindex(t)], label = L"$\dot{y}$")
        i == lastindex(mus) ? Legend(fig[1, i+1], ax) : continue
    end
    Label(fig[0, :], L"RK4 solution $y(t)$, $\dot{y}(t)$ vs $t$ for each $\mu$")
    fig
end
prob4part1()
```

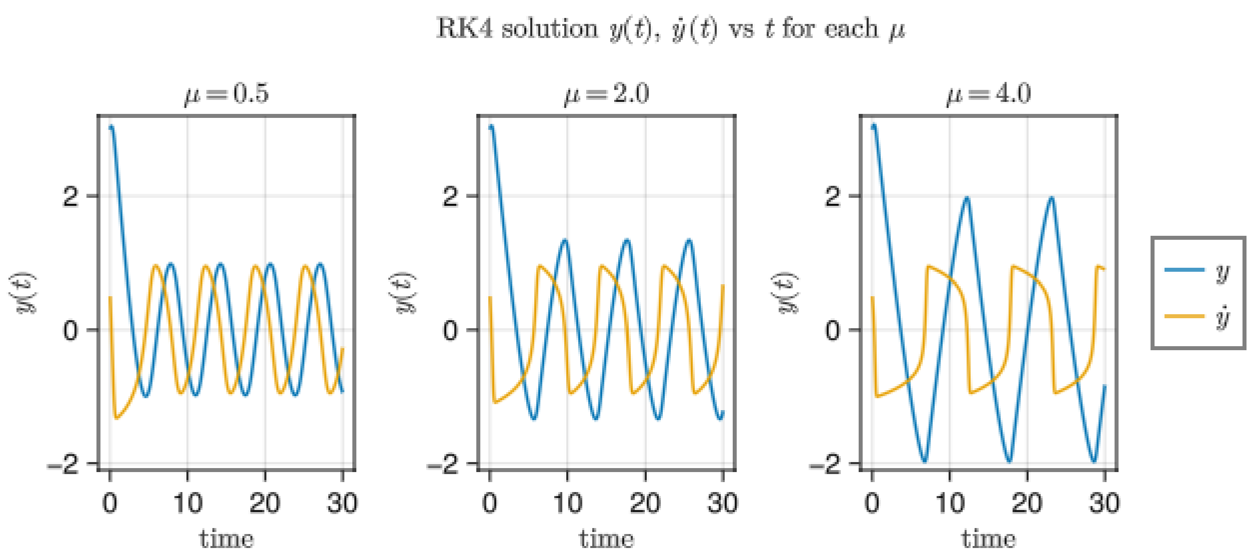


Figure 3: Trajectories $y(t)$ in blue and $\dot{y}(t)$ in yellow for the solution of the IVP in the time domain $t \in [0, 30]$, with spatial resolution $h = 0.025$, with initial conditions $y_0 = 3$ and $v_0 = 0.5$ for parameter $\mu = 0.5$ (left), $\mu = 2.0$ (middle), and $\mu = 4.0$ (right).

Part 2

For each μ, plot $\dot{y}(t)$ vs $y(t)$

Solution

```
function prob4part2()
    h = 0.025; u0 = SA[3.0, 0.5]; ts = SA[0.0, 30.0]; mus = SA[0.5, 2.0, 4.0]
    lbs = SA[L"$\mu = 0.5$", L"$\mu = 2.0$", L"$\mu = 4.0$"]
    fig = Figure(resolution=(650, 300))
    for i in eachindex(mus)
        ax = Axis(fig[1, i], title = lbs[i],
                  xlabel = L"$y(t)$", ylabel = L"$\dot{y}(t)$", limits = ((-2.1, 3.5), (-1.5, 1.1)))
        t, u = RK4(prob4ODE, u0, ts, h, mus[i])
        lines!(ax, [u[k][1] for k in eachindex(t)], [u[k][2] for k in eachindex(t)])
    end
    Label(fig[0, :], L"RK4 solution $\dot{y}(t)$ vs $y(t)$ for each $\mu$")
    fig
end
prob4part2()
```

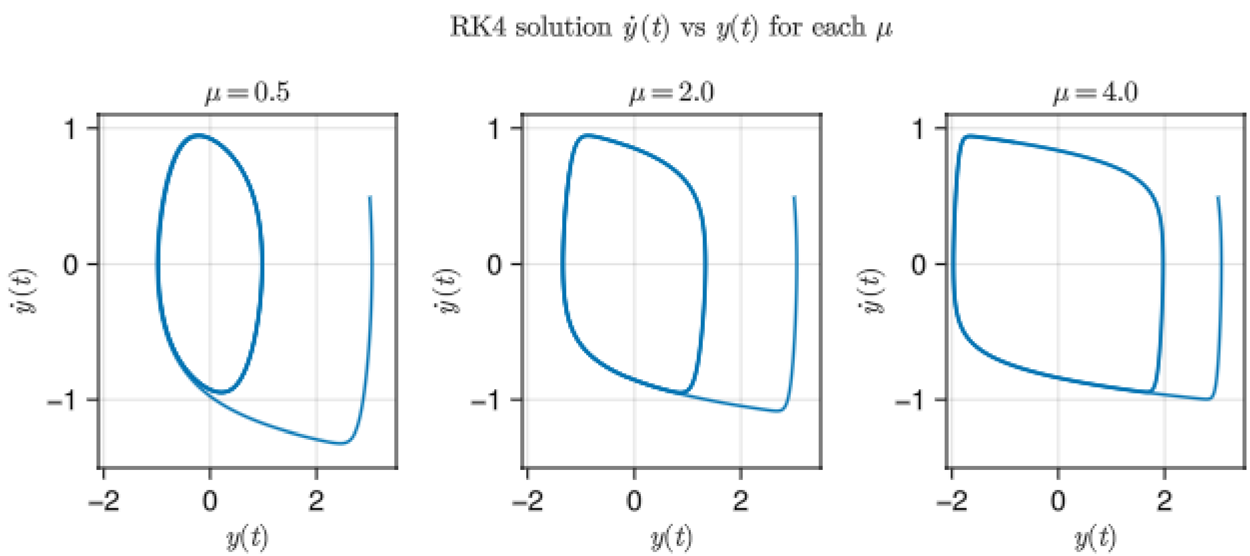


Figure 4: Phase diagram $\dot{y}(t)$ by $y(t)$ in blue. For the same solutions above for each μ.

Problem 5

For this problem use the IVP from problem 4 (44).

Use $y_0 = 3, \quad v_0 = 0.5, \quad \text{and} \quad \mu = 4$. Use RK4 to solve the IVP to $T = 30$.

Run simulations, respectively, with time steps $h = 2^i \quad \forall i \in \{-3, -4, \cdots, -11\}$

Use these simulations to estimate errors in numerical solutions.

$$E_n(h) = \frac{1}{1-(0.5)^4}\left(w_n(h) - w_{2n}(\frac{h}{2})\right) \tag{51}$$

```julia
function error(w::Vector{SVector{2, Float64}}, w2::Vector{SVector{2, Float64}})
    a::Float64 = 1.0666666666666667
    E = Vector{Float64}(undef, length(w))
    for n in eachindex(E)
        E[n] = norm(a*(w[n] - w2[2*n-1]))
    end
    E
end
```

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

Part 1

For each time step h, consider the maximum error over the interval $t \in [0, 30]$.

$$E_{\max}(h) = \max_{nh \in [0,30]} \|E_n(h)\| \tag{52}$$

Plot $E_{\max}(h)$ vs h in a log-log plot.

Solution

The following code approximates the solution $\mathbf{w} = \begin{bmatrix} w_1(t) \\ w_2(t) \end{bmatrix}$ in the time domain $t \in [0, 30]$ and computes the error for each h by computing the solutions $\mathbf{w}(h)$ and $\mathbf{w}(\frac{h}{2})$ and then populates a vector of size h with the maximum error $E_{\max}(h)$

```julia
function prob5part1()
    u0 = SA[3, 0.5]; μ = 4.0; ts = SA[0.0, 30.0]
    hs = [2.0^i for i in -3:-1:-11]
    fig = Figure()
    ax = Axis(fig[1, 1], title = L"Plot of the maximum error for varying $h$",
              xlabel = L"h", ylabel = L"$E_{\text{max}}(h)$",
              xscale = log10, yscale = log10)
    maxEns = Vector{Float64}(undef, length(hs))
    for i in eachindex(hs)
        h = hs[i]
        t1, u1 = RK4(prob4ODE, u0, ts, h, μ)
        t2, u2 = RK4(prob4ODE, u0, ts, h*0.5, μ)
        En = error(u1, u2)
        maxEns[i] = maximum(En)
    end
    scatterlines!(ax, hs, maxEns, color = :red)
    hlines!(ax, 1e-8, color = :green, linestyle=:dash, label=L"5\times10^{-8}")
    Legend(fig[1, 2], ax)
    fig
end
prob5part1()
```

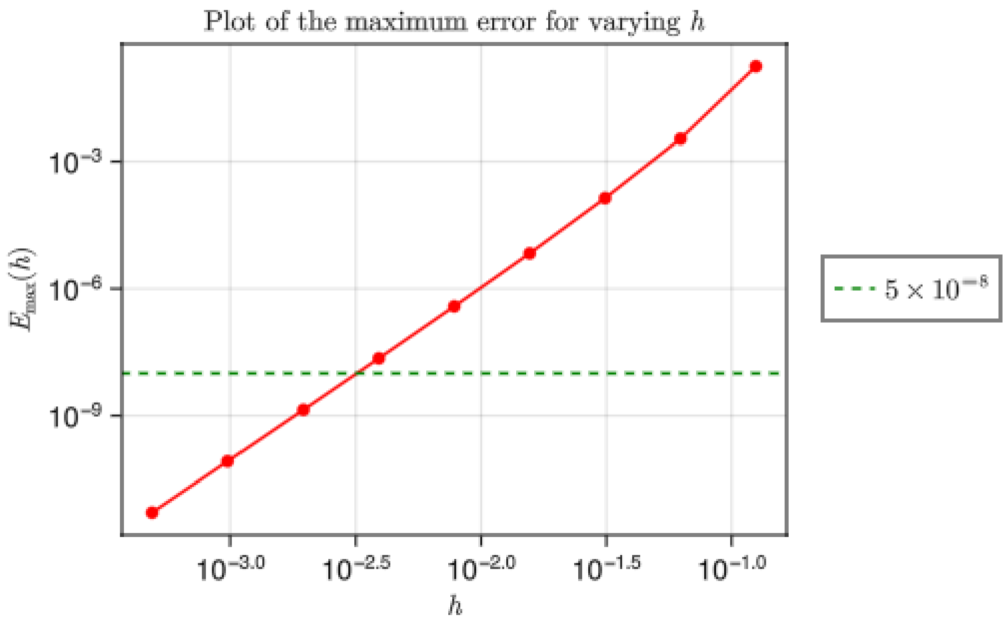


Figure 5: log-log plot of the maximum error for the solution of the IVP with $h = 2^i \quad \forall i \in \{-3, -4, \cdots, -11\}$. We can clearly see that when $h = 2^{-9}$ the maximum error is below $5imes10^{-8}$ (horizontal dashed line in green).

Part 2

From the sequence $h = 2^i \quad \forall i \in \{-3, -4, \cdots, -10\}$, find the time step h_c such that

$$E_{\max}(h_c) < 5 \times 10^{-8} \tag{53}$$

Report the value of h_c.

Solution

As you can see from Figure 5 the value of h that such that the maximum error is less than 5×10^{-8} corresponds to

$$h_c = 2^{-9} \approx 10^{-2.7} \tag{54}$$

Part 3

Plot $\|E_n(h)\|$ vs t_n respectively for time step sizes h_c and $\frac{h_c}{2}$. Use the log-scale for $\|E_n(h)\|$. Plot the two curves in ONE figure for comparison.

HINT: In the error estimation, $w_n(h)$ is compared with $w_{2n}(\frac{h}{2})$, NOT $w_n(\frac{h}{2})$. Look at the sample code on how to estimate error in numerical solution of ODE systems.

Solution

```
function prob5part3()
    u0 = SA[3.0, 0.5]; μ::Float64 = 4.0; ts = SA[0.0, 30.0]; hs = SA[2.0^(-9), 2.0^(-10)]
    lbs = SA[L"$h = 2^{-9}$", L"$h = 2^{-10}$"]
    fig = Figure()
    ax = Axis(fig[1, 1], title = L"True error for the solution over $t \in [0, 30]$",
              xlabel = L"$t$", ylabel = L"$log(||E_n(h)||)$")
    for i in eachindex(hs)
        t1, u1 = RK4(prob4ODE, u0, ts, hs[i], μ)
        t2, u2 = RK4(prob4ODE, u0, ts, hs[i]*0.5, μ)
        En = error(u1, u2)
        lines!(ax, t1, En, label = lbs[i])
    end
    Legend(fig[1, 2], ax)
    fig
end
prob5part3()
```

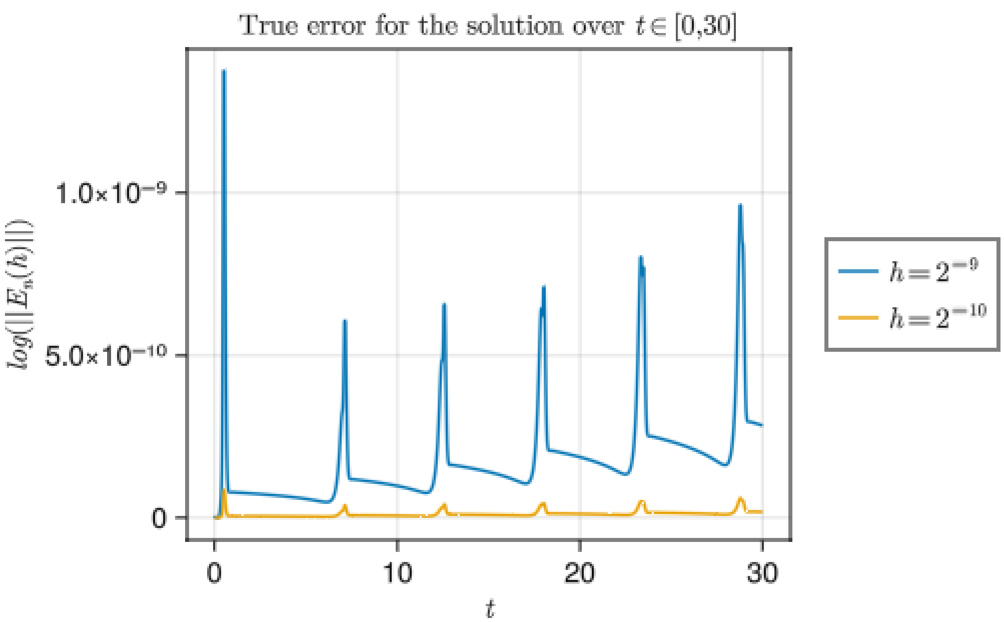


Figure 6: log-linear plot of the solution to the IVP in problem 4 using RK4 for $h = 2^{-9}$ in blue and $h = 2^{-10}$ in yellow (which is $\frac{2^{-9}}{2}$)

Problem 6

The Fehlberg method is an embedded Runge-Kutta method with orders 5 and 4. Implement the Fehlberg method to solve the IVP to $T = 30$.

Use $y_0 = 3$, $v_0 = 0.5$, and $h = 0.025$. In each time step, the Fehlberg error is estimated as:

$$E_{n+1}(h) \approx \|\mathbf{w}_{n+1} - \tilde{\mathbf{w}}_{n+1}\| \tag{55}$$

where $\mathbf{w}_{n+1}$ and $\tilde{\mathbf{w}}_{n+1}$ are respectively the results of the two methods in the Fehlberg.

Part 1

Plot the Fehlberg error (55) over time. Use log-scale for the error.

Solution

The Runge-Kutta-Fehlberg order 4/order 5 embedded pair is as follows

$$\begin{cases} s_1 & = f(u_k, t_k) \\ s_2 & = f\left(u_k + \frac{1}{4}hs_1, t_k + \frac{1}{4}h\right) \\ s_3 & = f\left(u_k + \frac{3}{32}hs_1 + \frac{9}{32}hs_2, t_k + \frac{3}{8}h\right) \\ s_4 & = f\left(u_k + \frac{1932}{2197}hs_1 - \frac{7200}{2197}hs_2 + \frac{7296}{2197}h * s_3, t_k + \frac{12}{13}h\right) \\ s_5 & = f\left(u_k + \frac{439}{216}hs_1 - 8hs_2 + \frac{3680}{513}hs_3 - \frac{845}{4104}hs_4, t_k + h\right) \\ s_6 & = f\left(u_k - \frac{8}{27}hs_1 + 2hs_2 - \frac{3544}{2565}hs_3 + \frac{1859}{4104}hs_4 - \frac{11}{40}hs_5, t_k + \frac{1}{2}h\right) \\ u_{k+1} & = u_k + h\left(\frac{25}{216}s_1 + \frac{1408}{2565}s_3 + \frac{2197}{4104}s_4 - \frac{1}{5}s_5\right) \\ z_{k+1} & = u_k + h\left(\frac{16}{135}s_1 + \frac{6656}{12825}s_3 + \frac{28561}{56430}s_4 - \frac{9}{50}s_5 + \frac{2}{55}s_6\right) \\ e_{k+1} & = \|z_{k+1} - u_{k+1}\| = h\|\frac{1}{360}s_1 - \frac{128}{4275}s_3 - \frac{2197}{75240}s_4 + \frac{1}{50}s_5 + \frac{2}{55}s_6\| \end{cases} \tag{56}$$

where z_{k+1} is an order 5 approximation, u_{k+1} is order 4, and e_{k+1} is the error estimate needed for step size control.

Set a relative error tolerance ϵ and an initial step size h. After computing $u_1, z_1,$ and e_1, the relative error test

$$\frac{e_k}{\|w_k\|} < \epsilon \tag{57}$$

is checked for $k = 1$. If successful, the new u_1 is replaced with the locally extrapolated version z_1, and the program moves on to the next step. If the test fails, the step is tried again with step size h_* given by

$$h_* = 0.8\left(\frac{\epsilon\|w_k\|}{e_k}\right)^{\frac{1}{5}} h_k \tag{58}$$

A repeated failure, which is unlikely, is treated by cutting step size in half until success is reached.

Let us implement the above algorithm

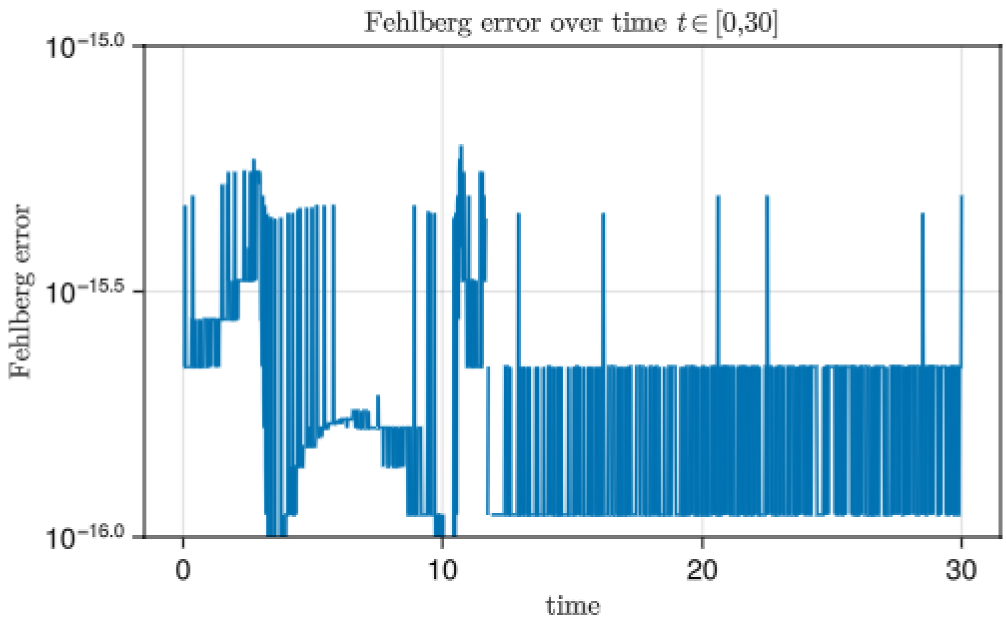
```
function updateRKF45!(f::Function, uk::SVector{2, Float64}, tk::Float64, h::Float64, p::Float64)
    c1  = 0.8793809740555303; c2 = 3.277196176604461
    c3  = 3.3208921256258535; c4 = 0.9230769230769231
    c5  = 2.0324074074074074; c6 = 7.173489278752436
    c7  = 0.20589668615984405; c8 = 0.2962962962962963
    c9  = 1.3816764132553607; c10 = 0.4529727095516569
    c11 = 0.11574074074074074; c12 = 0.5489278752436647
    c13 = 0.5353313840155945; c14 = 0.11851851851851852
    c15 = 0.5189863547758284; c16 = 0.5061314903420167
    c17 = 0.03636363636363636
    s1  = f(uk, p, tk)
    s2  = f(uk+0.25*h*s1, p, tk+0.25*h)
    s3  = f(uk+0.09375*h*s1+0.28125*h*s2, p, tk+0.375*h)
    s4  = f(uk+c1*h*s1-c2*h*s2+c3*h*s3, p, tk+c4*h)
```

```
    s5  = f(uk+c5*h*s1-8.0*h*s2+c6*h*s3-c7*h*s4, p, tk+h)
    s6  = f(uk-c8*h*s1+2.0*h*s2-c9*h*s3+c10*h*s4-0.275*h*s5, p, tk+0.5*h)
    u   = uk+h*(c11*s1+c12*s3+c13*s4-0.2*s5)
    z   = uk+h*(c14*s1+c15*s3+c16*s4-0.18*s5+c17*s6)
    err = norm(z - u)
    return u, z, err
end

rel_error_test(error::Float64, uk::SVector{2, Float64}) = (error/norm(uk)) < eps(Float64)
h_star(error::Float64, uk::SVector{2, Float64}, h::Float64) = h * 0.8 * ((eps(Float64)*norm(uk))/error)^(1/5)

function RKF45(f::Function, u0::SVector{2, Float64}, tspan::SVector{2, Float64}, h::Float64, p::Float64)
    t0, tf = tspan
    N = Int(floor((tf-t0)/h))
    t = Vector{Float64}(undef, N + 1)
    sol = Vector{SVector{2, Float64}}(undef, N + 1)
    error = Vector{Float64}(undef, N + 1)
    t[1] = t0
    sol[1] = u0
    error[1] = 0.0
    for k = 1:N
        t[k+1] = t[k] + h
        uk_new, zk_new, err_new = updateRKF45!(f, sol[k], t[k], h, p)
        if rel_error_test(err_new, uk_new)
            sol[k+1] = zk_new
            error[k+1] = err_new
        else
            h_new = h_star(err_new, uk_new, h)
            uk_new, zk_new, err_new = updateRKF45!(f, sol[k], t[k], h_new, p)
            while ~(rel_error_test(err_new, uk_new))
                h_new = h_new/2
                uk_new, zk_new, err_new = updateRKF45!(f, sol[k], t[k], h_new, p)
            end
            sol[k+1] = zk_new
            error[k+1] = err_new
        end
    end
    return t, sol, error
end

function prob6part1()
    u0 = SA[3.0, 0.5]; μ = 4.0; ts = SA[0.0, 30.0]; h = 0.025
    t, u, err = RKF45(prob4ODE, u0, ts, h, μ)
    fig = Figure()
    ax = Axis(
        fig[1, 1], title = L"Fehlberg error over time $t \in [0, 30]$",
        xlabel = L"\text{time}", ylabel = L"\text{Fehlberg error}",
        yscale = log10, limits = (nothing, (1e-16, 1e-15))
    )
    lines!(ax, t, err)
    fig
end
prob6part1()
```



Part 2

Estimate the error using

$$E_n(h) = \frac{1}{1-(0.5)^5} \|\mathbf{w}_n(h) - \mathbf{w}_{2n}(\frac{h}{2})\| \tag{59}$$

Plot the Fehlberg error (55) over time and the estimate for the true error (59) above over time in ONE figure. Use log-scale for the errors.

Solution

```
function errorprob6(w::Vector{SVector{2, Float64}}, w2::Vector{SVector{2, Float64}})
    a::Float64 = 1.032258064516129
    E = Vector{Float64}(undef, length(w))
    for n in eachindex(E)
        E[n] = norm(a * w[n] - w2[2 * n - 1])
    end
    E
end

function prob6part2()
    u0 = SA[3.0, 0.5]; μ = 4.0; ts = SA[0.0, 30.0]; h = 0.025
    t1, u1, e1 = RKF45(prob4ODE, u0, ts, h, μ)
    t2, u2, e2 = RKF45(prob4ODE, u0, ts, h*0.5, μ)
    fig = Figure()
    ax = Axis(
        fig[1, 1], title = L"Comparison of Fehlberg error and estimated error over $t \in [0, 30]$",
        xlabel = L"\text{time}", ylabel = L"$\log{\text{error}}$",
        yscale = log10, limits = (nothing, (1e-16, 1e1))
```

```
    )
    En = errorprob6(u1, u2)
    lines!(ax, t1, e1, label = L"\text{Fehlberg}")
    lines!(ax, t1, En, label = L"\text{Estimated}")
    Legend(fig[1, 2], ax, L"\text{Error}", framevisible = false)
    fig
end
prob6part2()
```