Homework 4

AM213B

Kevin Silberberg

2025-05-10

1 Problem 1

Consider the one-stage implicit RK method described by the Butcher Array

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

1.1 Part 1

Show that it has second order accuracy.

1.1.1 Solution

We can show that the 1-stage method has second order accuracy by showing that the RK method satisfies the order-conditions up to order 2.

The first order condition is

$$\sum_{i=1}^{1} b_i = 1 \tag{2}$$

this condition is easily satisfied since $b_1 = 1$

the second order condition is

$$\sum_{i=1}^{1} b_i c_i = \frac{1}{2} \tag{3}$$

this condition is also easily satisfied since $c_1 = \frac{1}{2}$ and

$$b_1 c_1 = 1 \cdot \frac{1}{2} = \frac{1}{2} \tag{4}$$

The first third order condition is

$$\sum_{i,j=1}^{1} b_i a_{ij} c_j = \frac{1}{6} \tag{5}$$

$$b_1 a_{11} c_1 = 1 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \neq \frac{1}{6} \tag{6}$$

the second third order condition is

$$\sum_{i=1}^{1} b_i c_i^2 = \frac{1}{3} \tag{7}$$

$$b_1 c_1^2 = 1 \cdot \frac{1}{4} \neq \frac{1}{3} \tag{8}$$

both third order conditions are not satisfied, thus the method is second order.

1.2 Part 2

Derive its stability function $\Phi(z)$

1.2.1 Solution

The stability function is

$$\begin{aligned} \phi(z) &= 1 + zb(1 - zA)^{-1} \qquad (9)\\ &= 1 + \frac{z}{1 - \frac{z}{2}} \qquad (10)\\ &= \frac{2+z}{2-z} \qquad (11)\end{aligned}$$

1.3 Part 3

Show that it is A-stable, but not L-stable.

1.3.1 Solution

An RK method is A-stable if it's stability function satisfies

$$|\phi(z)| \leq 1, \quad \text{Re}(z) \leq 0 \tag{12}$$

where $\text{Re}(z)$ denotes the real part of a complex number $z \in \mathbb{C}$.

L-stability is a special case of A-stability. A method is L-stable if it is both A-stable and the stability function $\phi(z) \to 0$ as $z \to \infty$.

Let us show that the method is A-stable.

$$|\phi(z)| = \frac{|2-z|}{|2+z|} \leq 1 \text{ when } \text{Re}(z) \leq 0 \tag{13}$$

this is because the numerator is less than the denominator in absolute value for all $z \in \mathbb{C}$ when $\text{Re}(z) \leq 0$.

$$|2+z| < |2-z| \quad \text{when } \text{Re}(z) \neq 0 \tag{14}$$

Thus the condition for A-stability is satisisfied.

Let us compute the limit of the stability function as $z \to \infty$ to determine if the method is L-stable.

$$\lim_{z\to\infty} \frac{2+z}{2-z} = \lim_{z\to\infty} \frac{1}{-1} = -1, \quad \text{L'Hopital's rule} \tag{15}$$

Thus the method is not L-stable.

2 Problem 2

Plot the region of stability for each of the methods below:

- 2-step BDF
- 3-step BDF
- 4-step BDF
- 5-step BDF

2.0.1 Solution

See Section 6.1 for the implementation.

2-step BDF

$$\frac{3}{2}u_{n+2} - 2u_{n+1} + \frac{1}{2}u_n = hf(u_{n+2}, t_{n+2}) \tag{16}$$

Characteristic Polynomials:

$$\alpha_0 = \frac{1}{2} \qquad \beta_0 = 0 \tag{17}$$
$$\alpha_1 = -2 \qquad \beta_1 = 0 \tag{18}$$
$$\alpha_2 = \frac{3}{2} \qquad \beta_2 = 1 \tag{19}$$

$$\rho(\xi) = \frac{3}{2}\xi^2 - 2\xi + \frac{1}{2} \tag{20}$$
$$\sigma(\xi) = \xi^2 \tag{21}$$

Stability Polynomial:

$$\Pi(\xi; z) = \rho(\xi) - z\sigma(\xi) \tag{22}$$
$$= (\frac{3}{2} - z)\xi^2 - 2\xi + \frac{1}{2} \tag{23}$$

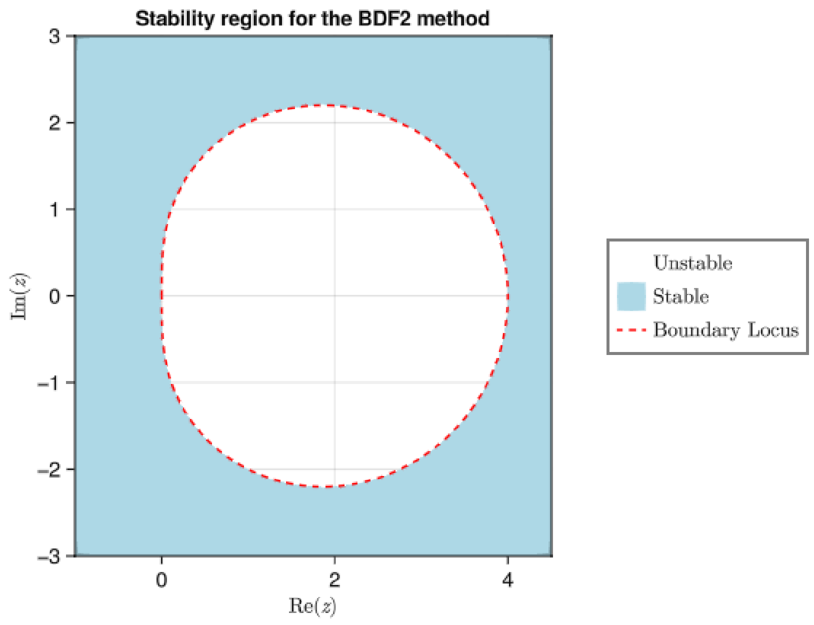


Figure 1: Stability Region for the BDF2 method.

3-step BDF

$$\frac{11}{6}u_{n+3} - 3u_{n+2} + \frac{3}{2}u_{n+1} - \frac{1}{3}u_n = hf(u_{n+3}, t_{n+3}) \tag{24}$$

Characteristic Polynomials:

$$\alpha_0 = -\frac{1}{3} \qquad \beta_0 = 0 \tag{25}$$
$$\alpha_1 = \frac{3}{2} \qquad \beta_1 = 0 \tag{26}$$
$$\alpha_2 = -3 \qquad \beta_2 = 0 \tag{27}$$
$$\alpha_3 = \frac{11}{6} \qquad \beta_3 = 1 \tag{28}$$

$$\rho(\xi) = \frac{11}{6}\xi^3 - 3\xi^2 + \frac{3}{2}\xi - \frac{1}{3} \tag{29}$$
$$\sigma(\xi) = \xi^3 \tag{30}$$

Stability Polynomial:

$$\Pi(\xi; z) = (\frac{11}{6} - z)\xi^3 - 3\xi^2 + \frac{3}{2}\xi - \frac{1}{3} \tag{31}$$

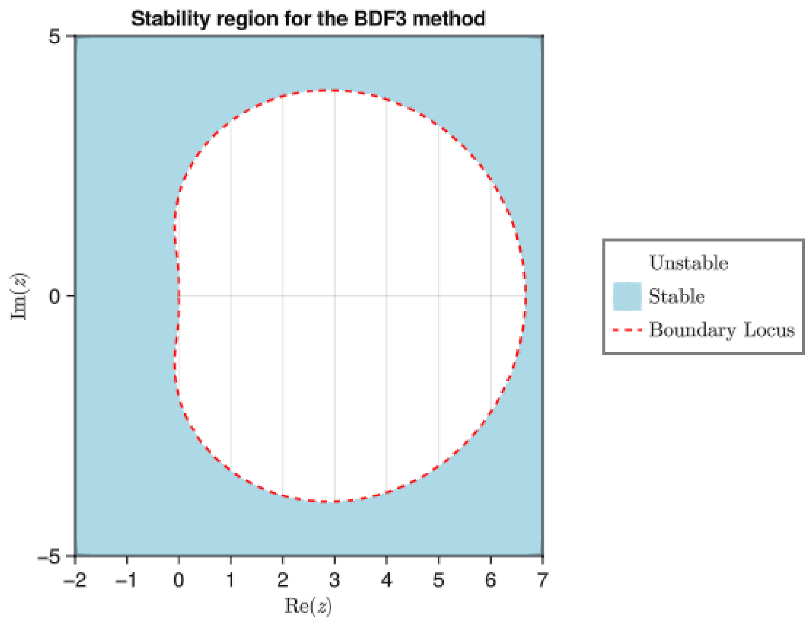


Figure 2: Stability region for the BDF3 method.

4-step BDF

$$\frac{25}{12}u_{n+4} - 4u_{n+3} + 3u_{n+2} - \frac{4}{3}u_{n+1} + \frac{1}{4}u_n = hf(u_{n+4}, t_{n+4}) \tag{32}$$

Characteristic Polynomials:

$$\alpha_0 = \frac{1}{4} \qquad \beta_0 = 0 \tag{33}$$
$$\alpha_1 = -\frac{4}{3} \qquad \beta_1 = 0 \tag{34}$$
$$\alpha_2 = 3 \qquad \beta_2 = 0 \tag{35}$$
$$\alpha_3 = -4 \qquad \beta_3 = 0 \tag{36}$$
$$\alpha_4 = \frac{25}{12} \qquad \beta_4 = 1 \tag{37}$$

$$\rho(\xi) = \frac{25}{12}\xi^4 - 4\xi^3 + 3\xi^2 - \frac{4}{3}\xi + \frac{1}{4} \tag{38}$$
$$\sigma(\xi) = \xi^4 \tag{39}$$

Stability Polynomial:

$$\Pi(\xi; z) = (\frac{25}{12} - z)\xi^4 - 4\xi^3 + 3\xi^2 - \frac{4}{3}\xi + \frac{1}{4} \tag{40}$$

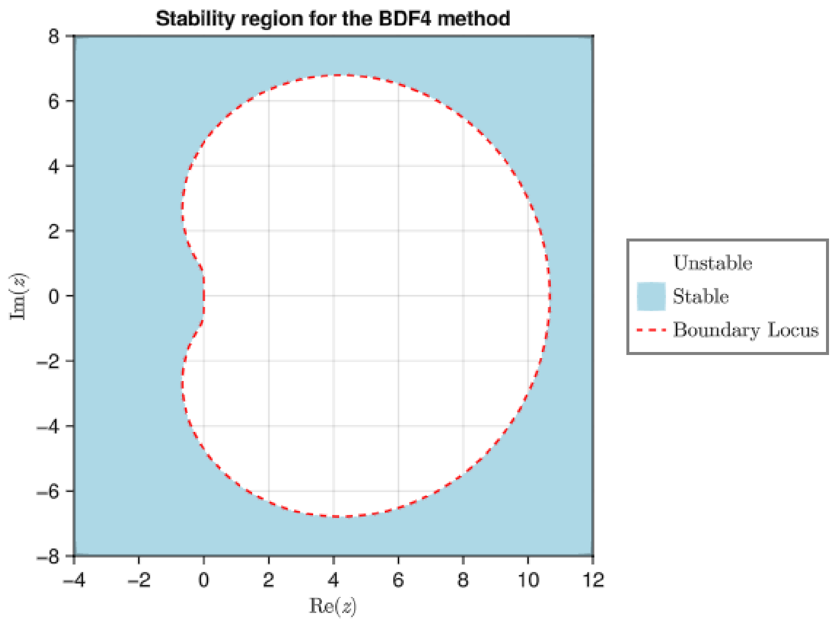


Figure 3: Stability region for the BDF4 method.

5-step BDF

$$\frac{137}{60}u_{n+5} - 5u_{n+4} + 5u_{n+3} - \frac{10}{3}u_{n+2} + \frac{5}{4}u_{n+1} - \frac{1}{5}u_n = hf(u_{n+5}, t_{n+5}) \tag{41}$$

Characteristic Polynomials:

$$\alpha_0 = -\frac{1}{5} \qquad \beta_0 = 0 \tag{42}$$
$$\alpha_1 = \frac{5}{4} \qquad \beta_1 = 0 \tag{43}$$
$$\alpha_2 = -\frac{10}{3} \qquad \beta_2 = 0 \tag{44}$$
$$\alpha_3 = 5 \qquad \beta_3 = 0 \tag{45}$$
$$\alpha_4 = -5 \qquad \beta_5 = 0 \tag{46}$$
$$\alpha_5 = \frac{137}{60} \qquad \beta_5 = 1 \tag{47}$$

$$\rho(\xi) = \frac{137}{60}\xi^5 - 5\xi^4 + 5\xi^3 - \frac{10}{3}\xi^2 + \frac{5}{4}\xi - \frac{1}{5} \tag{48}$$
$$\sigma(\xi) = \xi^5 \tag{49}$$

Stability Polynomials:

$$\Pi(\xi; z) = (\frac{137}{60} - z)\xi^5 - 5\xi^4 + 5\xi^3 - \frac{10}{3}\xi^2 + \frac{5}{4}\xi - \frac{1}{5} \tag{50}$$

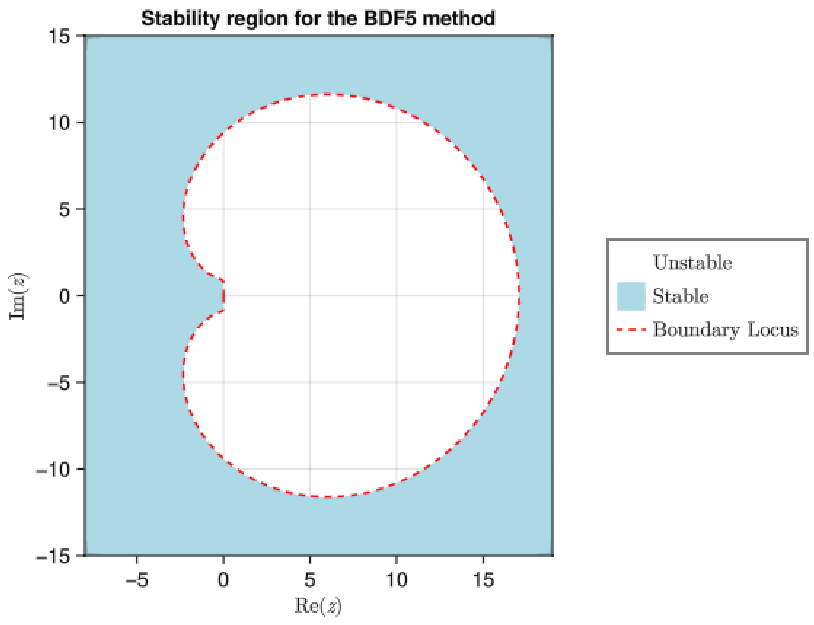


Figure 4: Stability region for the BDF5 method.

3 Problem 3

Implement the shooting method to solve the two-point BVP

$$\begin{cases} u'' - \left(1 + 0.5u'^2\right) u = \sin(x) \\ u(0) = 1, \quad u(2) = 1.5 \end{cases} \tag{51}$$

Convert the 2nd-order ODE to a first order system of ODE.

Let

$$z_1(x) = u(x) \qquad\qquad z_2(x) = u'(x) \tag{52}$$

such that

$$\frac{dz_1}{dx} = z_2 \tag{53}$$

$$\frac{dz_2}{dx} = \sin(x) + z_1(1 + 0.5z_2^2) \tag{54}$$

$$z_1(0) = 1 \quad z_1(2) = 1.5 \tag{55}$$

for $t \in [0, 2]$

Use RK4 with $h = 0.002$ as the ODE solver to evaluate $G(v)$. Use Newton's method with $v_0 = -1$ to solve $G(v) = 0$.

3.1 Part 1

Report the value of v found in the shooting method.

3.1.1 Solution

We need to cast the BVP above into an IVP. We use the value for the right boundary as the velocity in the initial condition.

See Section 6.2 for the implementation in the appendix.

The value of v = -1.1651816312367032

3.2 Part 2

Plot $u(x)$ vs x

3.2.1 Solution

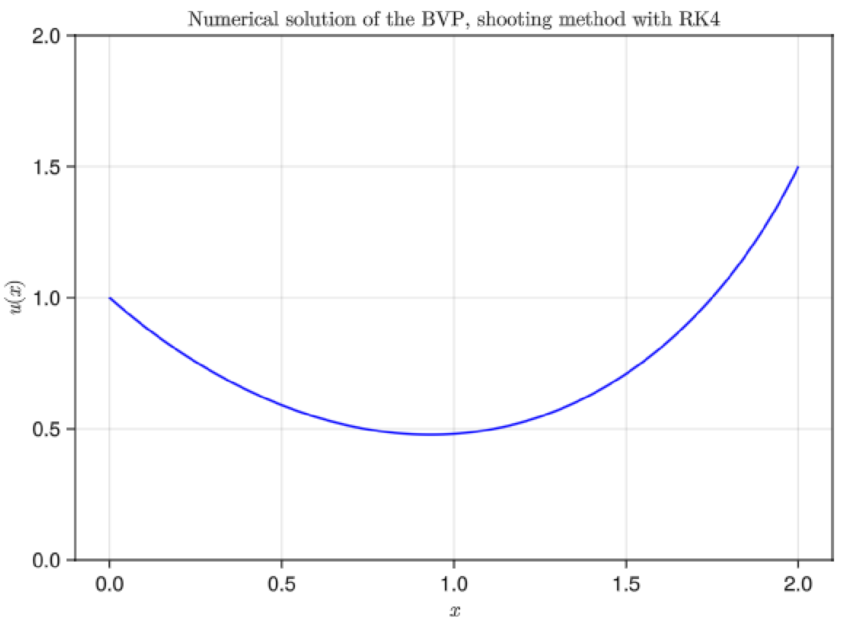


Figure 5: The solution to the boundary value problem (51) using the shooting method with RK4.

4 Problem 4

Use the finite difference method to solve the two-point BVP.

$$\begin{cases} u'' - 625u = -625x \\ u(0) = 1, \quad u(2) = 1 \end{cases} \tag{56}$$

The exact solution of the BVP is

$$u(x) = x + \frac{1 + e^{-50}}{1 - e^{-100}} \left(e^{-25x} - e^{25(x-2)} \right) \tag{57}$$

Solve the BVP numerically using the FDM with $N = 1000$ ($h = 0.002$)

4.1 Part 1

Plot the numerical $u(x)$ vs x and the exact $u(x)$ vs x in one figure.

4.1.1 Solution

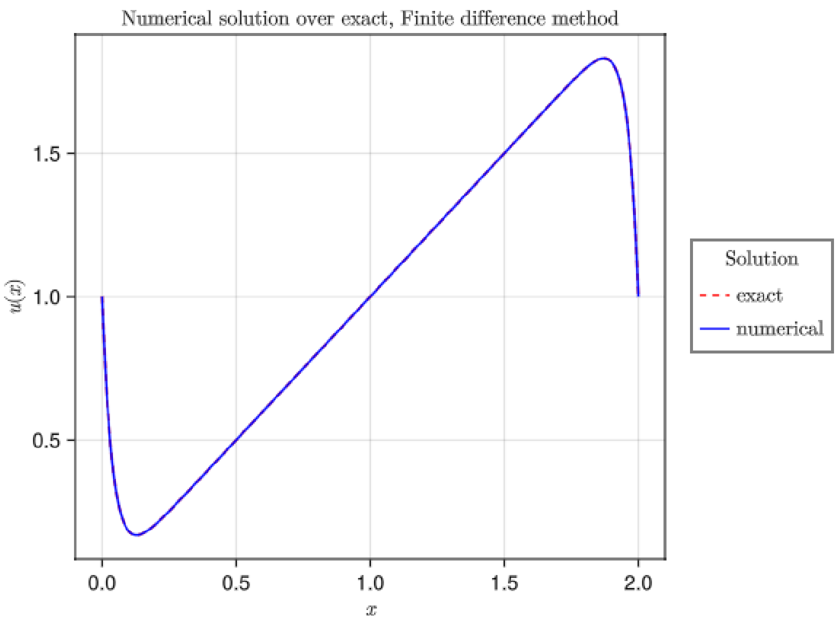


Figure 6: Numerical solution of (56) using the FDM method over the exact solution over the domain $x \in [0, 2]$

4.2 Part 2

Estimate the error in the numerical $u(x)$ using the results of $N = 1000$ and $N = 2000$. Calculate the exact error in numerical $u(x)$ using the exact $u(x)$ given above.

Plot the estimated error vs x and the exact error vs x in on figure. Use linear scales for both x and the errors.

4.2.1 Solution

We estimate the error using the following formula:

$$E_n(h) = \frac{1}{1-(0.5)^2}\left(u_n(h) - u_{2n}(h/2)\right) \tag{58}$$

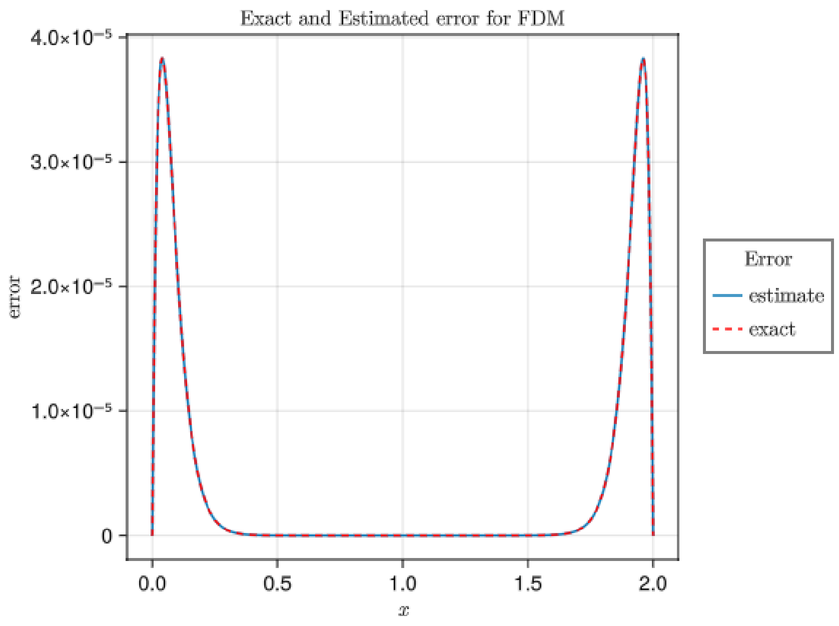


Figure 7: The estimated error in blue, plotted over the exact error in dash red. The exact error is computed by considering the absolute element wise difference between the exact solution and the approximated solution.

4.3 Part 3

The exact value of v is $v_{\text{exact}} = u'(0) = 1 - 25\frac{(1+e^{-50})^2}{1-e^{-100}}$

Use RK4 with $h = 0.002$ to solve the IVP:

$$\begin{cases} u'' - 625u = -625x \\ u(0) = 1, \quad u'(0) = v_{\text{exact}} \end{cases} \tag{59}$$

Report the value of $G(v_{\text{exact}}) = u(2)\Big|_{u'(0)=v_{\text{exact}}} - 1$

Hint: you will get a large negative value for $G(v_{\text{exact}})$

Remark: This problem demonstrates the advantage of FDM. The shooting method does not work when there is an exponentially "growing" mode.

4.3.1 Solution

Let us use RK4 with $h = 0.002$ to solve the IVP (59).

We need to first convert the 2nd order system to a system of first order

Let

$$u_1 = u \qquad\qquad u_2 = u' \tag{60}$$

such that

$$\frac{du_1}{dx} = u_2 \tag{61}$$

$$\frac{du_2}{dx} = 625u_1 - 625x \tag{62}$$

with initial condition

$$\mathbf{u}_0 = \begin{bmatrix} 1.0 \\ v_{\text{exact}} \end{bmatrix} \tag{63}$$

See Section 6.3 for the implementation

G(v_exact) = 70726.41559615127

5 Problem 5

We study the FDM on the general two-point BVP of Type-3

$$\begin{cases} u'' + p(x)u' + q(x)u = g(x) \\ u'(a) = \alpha, \quad u(b) = \beta \end{cases} \tag{64}$$

To accommodate $u'(a) = \alpha$, we need to put a at the middle of two grid points.

The numerical grid is such that

$$h = \frac{b-a}{N - \frac{1}{2}} \tag{65}$$

$$x_i = a + \left(i - \frac{1}{2}\right) h \quad \text{for} \quad i = 0, 1, 2, \cdots, N \tag{66}$$

$$x_0 = a - \frac{h}{2} \qquad\qquad x_{\frac{1}{2}} = a \qquad\qquad x_N = a + \left(N - \frac{1}{2}\right) h = b \tag{67}$$

the internal points x_i have indices $i = 1, 2, \cdots, N-1$

Combining the finite difference discretization and the boundary conditions, we write out a linear system of $\{u_1, u_2, \cdots, u_{N-1}\}$

$$\begin{cases} \left(\frac{1}{h^2} - \frac{p_i}{2h}\right) u_{i-1} + \left(-\frac{2}{h^2} + q_i\right) u_i + \left(\frac{1}{h^2} + \frac{p_i}{2h}\right) u_{i+1} = g_i & 1 \le i \le N-1 \\ \frac{u_1 - u_0}{h} = \alpha \Longrightarrow u_0 = u_1 - h\alpha & u_N = \beta \end{cases} \tag{68}$$

We write linear system in the matrix-vector form $\mathbf{AU} = \mathbf{F}$

5.1 Part 1

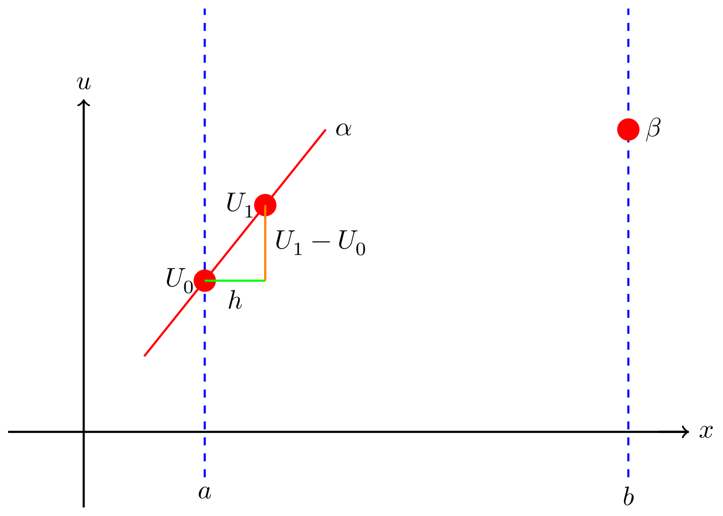
Write out the first row of $\mathbf{AU} = \mathbf{F}$.

5.1.1 Solution

If we consider the entire domain of the problem in the form $\mathbf{AU} = \mathbf{F}$ for the Dirichlet boundary conditions $u(a) = \alpha$ and $u(b) = \beta$ in the domain $[a, b]$ the matrix takes the form:

$$\underbrace{\frac{1}{h^2}\begin{bmatrix} h^2 & 0 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 0 & h^2 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_{m-1} \\ U_m \\ U_{m+1} \end{bmatrix}}_{\mathbf{U}} = \underbrace{\begin{bmatrix} \alpha \\ f(x_1) \\ \vdots \\ \vdots \\ f(x_m) \\ \beta \end{bmatrix}}_{\mathbf{F}} \tag{69}$$

Now for the Neumann boundary condition at the left boundary we have $u'(a) = \alpha$. Recall that the derivative at the point $U_0 = \alpha$ is nothing but a slope which we can equate by considering the rise/run over the first two points U_0 (the point at the left most boundary) and U_1 (the point h away from U_0) but still in the domain $[a, b]$.



solving for U_0 we have

$$\frac{U_1 - U_0}{h} = \alpha \tag{70}$$

$$U_0 = U_1 - h\alpha \tag{71}$$

if we consider how this effects (69), we essentially need to update the matrix $\mathbf{A}$ such that when we carry out the matrix multiplication between $\mathbf{A}$ and $\mathbf{U}$ for the first two elements U_0 and U_1 we get the result in (70).

We can do this by setting the first row of $\mathbf{AU} = \mathbf{F}$ as follows:

$$\frac{1}{h^2}\begin{bmatrix} -h & h \end{bmatrix}\begin{bmatrix} U_0 \\ U_1 \end{bmatrix} = \frac{1}{h^2}(-hU_0 + hU_1) = \frac{U_1 - U_0}{h} = \alpha \tag{72}$$

For the purpose of re-writing the numerical solver, we need to consider the iterative solution and plug in the value for U_0 into equation. For the general linear equation

$$\begin{cases} u''(x) + p(x)u'(x) + q(x)u(x) = f(x) \\ u'(a) = \beta, \quad u(b) = \beta \end{cases} \tag{73}$$

we can discretize to second order by

$$\left(\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2}\right) + p_i\left(\frac{U_{i+1} - U_{i-1}}{2h}\right) + q_iU_i = f_i \tag{74}$$

at the left boundary when we plug in $i = 1$ we get

$$\frac{1}{h^2}(U_0 - 2U_1 + U_2) + \frac{p_1}{2h}(U_2 - U_0) + q_1U_1 = f_1 \tag{75}$$

$$U_0\left(\frac{1}{h^2} - \frac{p_1}{2h}\right) + U_1\left(q_1 - \frac{2}{h^2}\right) + U_2\left(\frac{p_1}{2h} + \frac{1}{h^2}\right) = f_1 \tag{76}$$

Plugging in our result from (70) we have

$$(U_1 - h\alpha)\left(\frac{1}{h^2} - \frac{p_1}{2h}\right) + U_1\left(q_1 - \frac{2}{h^2}\right) + U_2\left(\frac{p_1}{2h} - \frac{1}{h^2}\right) = f_1 \tag{77}$$

$$U_1\left(q_1 - \frac{1}{h^2} - \frac{p_1}{2h}\right) + U_2\left(\frac{p_1}{2h} - \frac{1}{h^2}\right) = f_1 + \frac{\alpha}{h} + \frac{p_1}{2}\alpha \tag{78}$$

So the first row of the matrix $\mathbf{AU} = \mathbf{F}$ is thus:

$$\begin{bmatrix} (q_1 - \frac{1}{h^2} - \frac{p_1}{2h}) & (\frac{p_1}{2h} - \frac{1}{h^2}) & 0 & 0 \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \\ \end{bmatrix} = \begin{bmatrix} f_1 + \alpha\left(\frac{1}{h} + \frac{p_1}{2}\right) \\ \\ \\ \end{bmatrix} \tag{79}$$

5.2 Part 2

Implement the FDM to solve the following BVP

$$\begin{cases} u'' + e^{-\sin(x)}u' - u = -5 - \sin^2(x) \\ u'(0) = 2.5 \qquad\qquad u(2) = 0.5 \end{cases} \tag{80}$$

Plot the numerical solution $u(x)$ vs x.

5.2.1 Solution

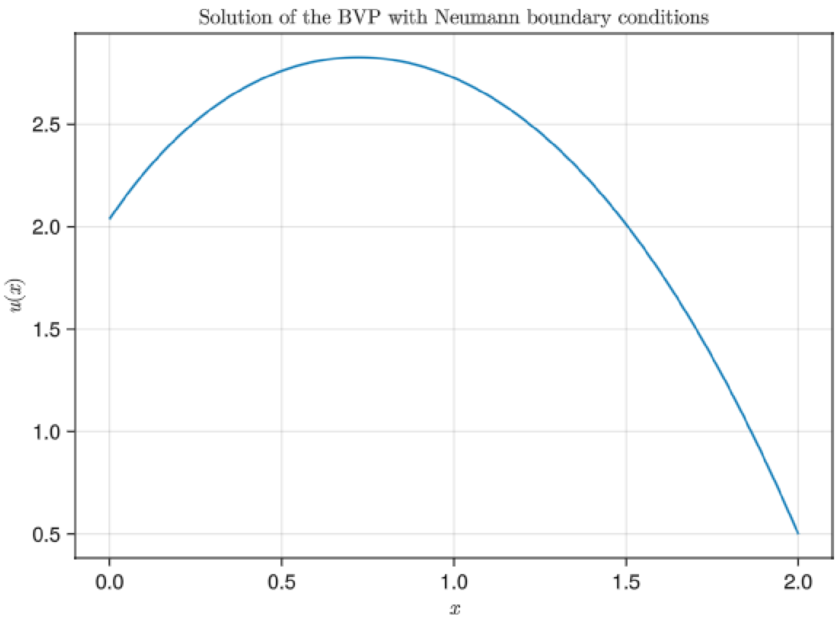


Figure 8: Solution of the boundary value problem with Neumann boundary conditions using Finite difference solver. See Section 6.4 for implementation.

5.3 Part 3

Estimate the error in numerical $u(x)$ using the results of $N = 1000$ and $N = 2000$.

Plot the estimated error vs x. Use linear scales for both x and the error.

5.3.1 Solution

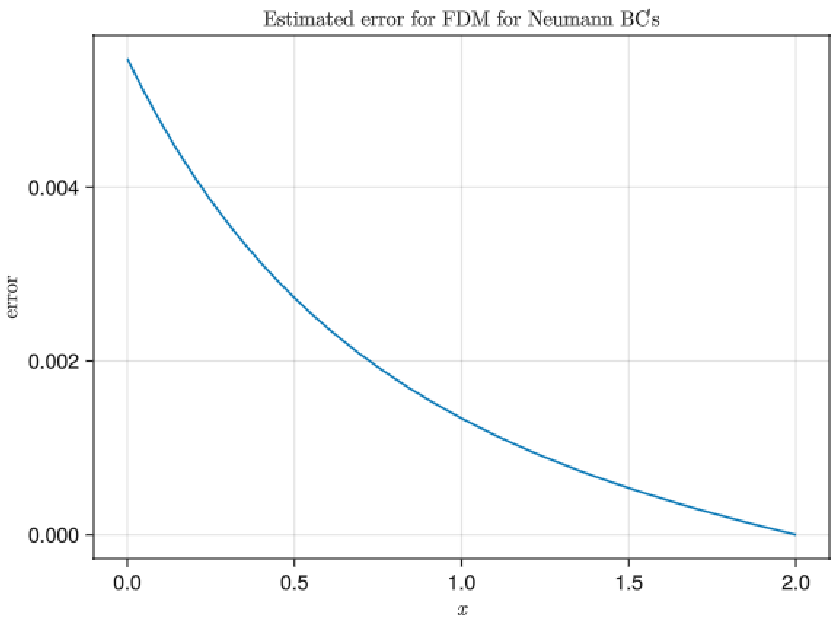


Figure 9: Plot of the estimated error using the formula $\frac{1}{1-0.5}\left(u_n(h) - u_{2n}(h/2)\right)$ since the method of FDM used above is only first order accurate.

6 Appendix: Source Code

6.1 Problem 2 Code

```julia
using GLMakie
using StaticArrays
using PolynomialRoots
using LinearAlgebra

function plot_stability_region(z::Function, Π::Function, method::AbstractString, lims::Tuple)
    res = 1000
    xs = collect(LinRange(lims[1][1], lims[1][2], res))
    ys = collect(LinRange(lims[2][1], lims[2][2], res))
    S = zeros(res, res)
    for idx in CartesianIndices(S)
        i, j = idx.I
        k = complex(xs[i], ys[j])
        S[idx] = maximum(abs.(roots(Π(k))))
    end
    fig = Figure()
    ax = Axis(
        fig[1, 1],
        title = "Stability region for the $(method) method",
        xlabel = L"\text{Re}(z)",
        ylabel = L"\text{Im}(z)",
        aspect = DataAspect(),
        limits = lims
    )
    z_vals = z.(LinRange(0, 2π, res))
    z_real = real.(z_vals)
    z_imag = imag.(z_vals)
    contourf!(ax, xs, ys, S, levels = [0, 1], colormap = [:lightblue, :white])
    lines!(ax, z_real, z_imag, color = :red, linestyle = :dash)
    elements = [PolyElement(color = :white),
                PolyElement(color = :lightblue),
                LineElement(color = :red, linestyle = :dash)]
    labels = [L"\text{Unstable}", L"\text{Stable}", L"\text{Boundary Locus}"]
    Legend(fig[1, 2], elements, labels, halign = :left)
    fig
end

function BDF2()
    ρ(ξ) = (3.0/2.0)*ξ^2 - 2.0*ξ + 0.5
    σ(ξ) = ξ^2
    z(θ) = ρ(exp(im*θ))/σ(exp(im*θ))
    Π(z) = [0.5, -2.0, (3.0/2.0) - z]
    fig = plot_stability_region(z, Π, "BDF2", ((-1, 4.5), (-3, 3)))
    save("prob2BDF2.png", fig)
end

function BDF3()
    ρ(ξ) = (11.0/6.0)*ξ^3 - 3.0ξ^2 + (3.0/2.0)*ξ - (1.0/3.0)
    σ(ξ) = ξ^3
    z(θ) = ρ(exp(im*θ))/σ(exp(im*θ))
    Π(z) = [-(1.0/3.0), (3.0/2.0), -3.0, (11.0/6.0) - z]
    fig = plot_stability_region(z, Π, "BDF3", ((-2.0, 7.0), (-5.0, 5.0)))
    save("prob2BDF3.png", fig)
end

function BDF4()
    ρ(ξ) = (25.0/12.0)*ξ^4 - 4.0*ξ^3 + 3.0*ξ^2 - (4.0/3.0)*ξ + (1.0/4.0)
    σ(ξ) = ξ^4
    z(θ) = ρ(exp(im*θ))/σ(exp(im*θ))
    Π(z) = [0.25, -(4.0/3.0), 3.0, -4.0, (25.0/12.0) - z]
    fig = plot_stability_region(z, Π, "BDF4", ((-4.0, 12.0), (-8.0, 8.0)))
    save("prob2BDF4.png", fig)
end

function BDF5()
    ρ(ξ) = (137.0/60.0)*ξ^5 - 5ξ^4 + 5ξ^3 - (10.0/3.0)*ξ^2 + (5.0/4.0)*ξ - (1.0/5.0)
    σ(ξ) = ξ^5
    z(θ) = ρ(exp(im*θ))/σ(exp(im*θ))
    Π(z) = [-(1.0/5.0), (5.0/4.0), -(10.0/3.0), 5.0, -5.0, (137.0/60.0) - z]
    fig = plot_stability_region(z, Π, "BDF5", ((-8, 19), (-15, 15)))
    save("prob2BDF5.png", fig)
end

BDF2()
BDF3()
BDF4()
BDF5()
```

6.2 Problem 3 Code

```
using GLMakie
using StaticArrays
using PolynomialRoots
using LinearAlgebra

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

```julia
end

function prob3ODE(u, t, p)
    SA[u[2], sin(t) + u[1]*(1.0 + 0.5*u[2]^2)]
end

function shooting(f, method, u0, tspan, v0, h, p)
    α, β = u0
    t0, tf = tspan
    local function G(v::Float64)
        u, t = method(f, SA[α, v], tspan, h, p)
        return u[end][1] - β
    end
    Gn = Inf
    v_current = v0
    while (Gn > eps())
        hv = v_current*10^(-5)
        Gn = G(v_current)
        Gn_prime = (G(v_current + hv) - G(v_current - hv))/(2.0*hv)
        v_new = v_current - (Gn/Gn_prime)
        v_current = v_new
    end
    method(f, SA[α, v_current], tspan, h, p)
end

function part1()
    u0 = SA[1.0, 1.5]
    tspan = SA[0.0, 2.0]
    v_init = -1.0
    h = 0.002
    p = nothing
    u, t = shooting(
        prob3ODE, RK4,
        u0, tspan,
        v_init, h, p
    )
    v_final = u[1][2]
    println("The value of v = $(v_final)")
end

function part2()
    u0 = SA[1.0, 1.5]
    tspan = SA[0.0, 2.0]
    v_init = -1.0
    h = 0.002
    p = nothing
    u, t = shooting(
        prob3ODE, RK4,
        u0, tspan,
        v_init, h, p
    )
    fig = Figure()
    ax = Axis(
        fig[1, 1],
        title = L"\text{Numerical solution of the BVP, shooting method with RK4}",
        xlabel = L"x",
        ylabel = L"u(x)",
        limits = (nothing, (0.0, 2.0))
    )
    lines!(ax, t, [u[k][1] for k in eachindex(t)],
        color = :blue)
    save("../figures/prob3part2.png", fig)
end
part1()
part2()
```

6.3 Problem 4 Code

```
using GLMakie
using StaticArrays
using PolynomialRoots
using LinearAlgebra

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

# analytical solution
exact(x) = x + ((1 + exp(-50))/(1 - exp(-100)))*(exp(-25*x) - exp(25*(x - 2)))

# Finite Difference Method (Dirichlet Boundary Conditions)
function FDM(f, xspan, u0, N)
    p, q, g = f # functions in terms of p, q, g
    x0, xf = xspan # support
    α, β = u0 # left and right values at the boundaries
    h = (xf - x0)/N
    xv = [x0 + i*h for i in 1:N-1]
    pv = p.(xv); qv = q.(xv); gv = g.(xv)
    A = zeros(N-1, N-1)
```

```julia
    for i in 1:N-1
        A[i, i] = -2/h^2 + qv[i]
    end
    for i in 2:N-1
        A[i, i-1] = 1/h^2 - pv[i]/(2*h)
    end
    for i in 1:N-1-1
        A[i, i+1] = 1/h^2 + pv[i]/(2*h)
    end
    b = gv
    b[1] = gv[1] - (1/h^2 - pv[1]/(2*h))*α
    b[end] = b[end] - (1/h^2 + pv[end]/(2*h))*β
    uv = A\b
    return [α; uv; β], [x0; xv; xf]
end

function part1()
    f = SA[
        p -> 0.0,
        q -> -625.0,
        g -> -625.0*g
    ]
    xspan = SA[0.0, 2.0]
    bcs = SA[1.0, 1.0]
    N = 1000
    u, x = FDM(f, xspan, bcs, N)
    fig = Figure()
    ax = Axis(
        fig[1, 1],
        title = L"\text{Numerical solution over exact, Finite difference method}",
        xlabel = L"x",
        ylabel = L"u(x)"
    )
    u_exact = exact.(x)
    lines!(ax, x, u_exact, color = :red, linestyle = :dash, label = L"\text{exact}")
    lines!(ax, x, u, color = :blue, label = L"\text{numerical}")
    Legend(fig[1, 2], ax, L"\text{Solution}")
    save("../figures/prob4part1.png", fig)
end
#part1()

function part2()
    f = SA[
        p -> 0.0,
        q -> -625.0,
        g -> -625.0*g
    ]
    xspan = SA[0.0, 2.0]
    bcs = SA[1.0, 1.0]
    N = 1000
    u1, x1 = FDM(f, xspan, bcs, N)
    u2, x2 = FDM(f, xspan, bcs, N*2)
    u_exact = exact.(x1)
    E = zeros(Float64, length(u1))
    for n in 2:length(u1) - 1
        E[n] = abs(1.0/(1.0-0.5^2) * (u1[n] - u2[2*n-1]))
    end
    fig = Figure()
    ax = Axis(
        fig[1, 1],
        title = L"\text{Exact and Estimated error for FDM}",
        xlabel = L"x",
        ylabel = L"\text{error}"
    )
    lines!(ax, x1, E, label = L"\text{estimate}")
    lines!(ax, x1, abs.(u1 .- u_exact), color = :red, linestyle = :dash, label = L"\text{exact}")
    Legend(fig[1, 2], ax, L"\text{Error}")
    save("../figures/prob4part2.png", fig)
end
#part2()

part3ODE(u, x, p) = SA[u[2], 625.0*u[1] - 625.0*x]
function part3()
    v_exact = 1.0 - 25 * ((1.0 + exp(-50))^2/(1 - exp(-100))) -eps()*100
    u0 = SA[1.0, v_exact]
    xspan = SA[0.0, 2.0]
    h = 0.002
    p = nothing
    u, x = RK4(part3ODE, u0, xspan, h, p)
    open("../prob4part3.txt", "w") do file
        redirect_stdout(file) do
            println("G(v_exact) = ", u[end][1] - 1.0)
        end
    end
end
```

6.4 Problem 5 Code

```
using GLMakie
using StaticArrays
using LinearAlgebra
using PolynomialRoots

# Finite Difference Method (Neumann Boundary Condition) (first order LTE)
function FDM(f, xspan, u0, N)
    p, q, g = f
    x0, xf = xspan
    α, β = u0
    h = (xf - x0)/N
    xv = [x0 + i*h for i in 1:N-1]
    pv = p.(xv); qv = q.(xv); gv = g.(xv)
    A = zeros(N-1, N-1)
    for i in 1:N-1
        A[i, i] = -2/h^2 + qv[i]
    end
    for i in 2:N-1
```

```julia
        A[i, i-1] = 1/h^2 - pv[i]/(2*h)
    end
    for i in 1:N-2
        A[i, i+1] = 1/h^2 + pv[i]/(2*h)
    end
    b = gv
    b[end] = b[end] - (1/h^2 + pv[end]/(2*h))*β
    # rewrite the first row to account for left neumann boundary
    A[1, 1] = qv[1] - 1/h^2 - pv[1]/2*h; A[1, 2] = pv[1]/2*h + 1/h^2
    b[1] = gv[1] + (1/h - pv[1]/(2))*α
    uv = A\b
    return [uv[1] - h*α; uv; β], [x0; xv; xf]
end

function part2()
    N = 2000
    F = SA[
        x -> exp(-sin(x)),
        x -> -1.0,
        x -> -5.0 - sin(x)^2
    ]
    xspan = SA[0.0, 2.0]
    u0 = SA[2.5, 0.5]
    u, x = FDM(F, xspan, u0, N)
    fig = Figure()
    ax = Axis(
        fig[1, 1],
        title = L"\text{Solution of the BVP with Neumann boundary conditions}",
        xlabel = L"x",
        ylabel = L"u(x)",
        limits = (nothing, nothing)
    )
    lines!(ax, x, u)
    fig
    save("../figures/problem5part2.png", fig)
end

function part3()
    N = 1000
    F = SA[
        x -> exp(-sin(x)),
        x -> -1.0,
        x -> -5.0 - sin(x)^2
    ]
    xspan = SA[0.0, 2.0]
    u0 = SA[2.5, 0.5]
    u1, x1 = FDM(F, xspan, u0, N)
    u2, x2 = FDM(F, xspan, u0, N*2)
    E = zeros(Float64, length(u1))
    for n in 1:length(u1)
        E[n] = abs(1.0/(1.0 - 0.5) * (u1[n] .- u2[2*n-1]))
    end
    fig = Figure()
    ax = Axis(
        fig[1, 1],
        title = L"\text{Estimated error for FDM for Neumann BC's}",
        xlabel = L"x",
        ylabel = L"\text{error}"
    )
    lines!(ax, x1, E)
    fig
    save("../figures/prob5part3.png", fig)
end
```