CAD Laboratory (CE4P001) — Assignment 1

Session: Autumn 2025

by

Himanshu Bairagi

(22CE02010)

School of Infrastructure

Indian Institute of Technology, Bhubaneswar

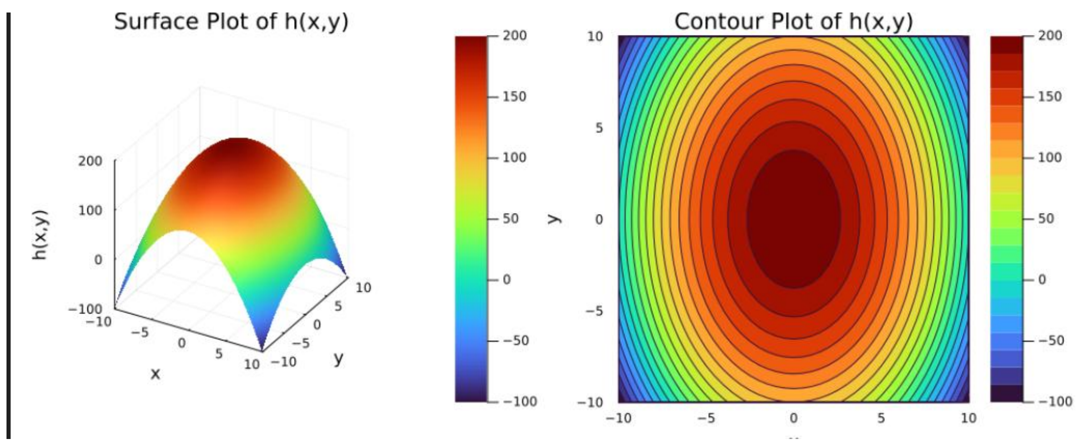
Odisha

October 2025

Question 1: Scalar Field of Hill Height

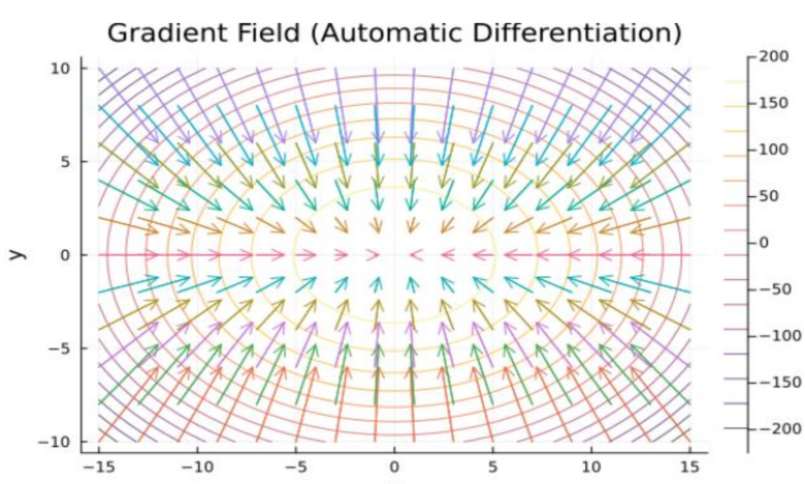
(a) Plotting the Scalar Field

- The function `h(x, y) = 200 - x^2 - 2y^2` is defined
- Grids for `x` and `y` are generated using `step range` or broadcasting
- A z range is defined to obtain values of height function for defined x and y
- `Plots.jl` is used to generate both `surface()` (3D plot) and `contour()` (2D plot)



(b) Automatic Gradient Calculation

- The package `CalculusWithJulia.jl` is imported
- The function `G(v) = 200 – v[1]^2 – 2v[1]^2` is defined, where v is an array
- The gradient is calculated automatically using the operator `gradient(G, [x, y])`
- X and Y are defined as grid of base positions of vectors
- U and V are defined as scaled horizontal and vertical components of vectors
- The grid points are passed to `quiver()` to visualize gradient directions

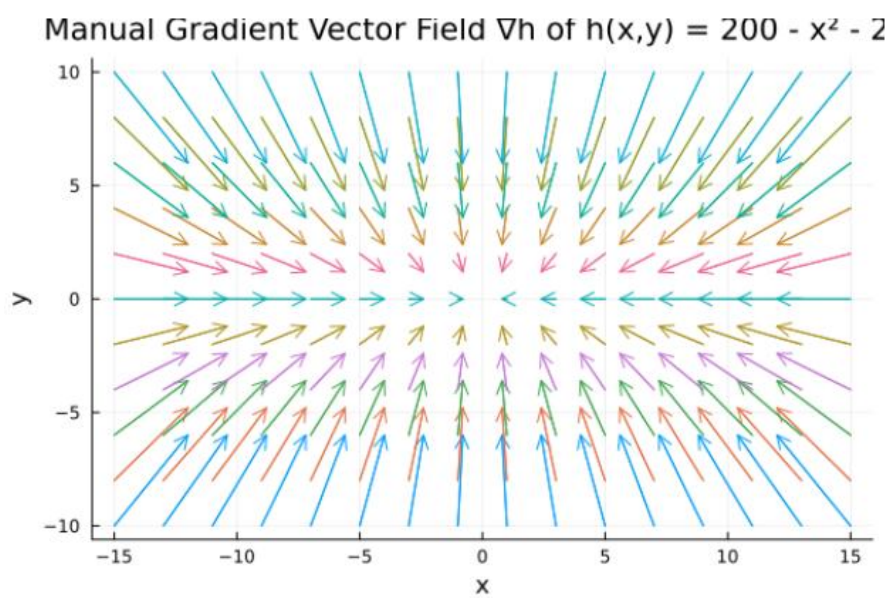


- Gradient field showing that The gradient of a scalar field is a vector that points in the direction of the greatest rate of increase of the field.

Manual Gradient Calculation

- Partial derivatives are defined manually: grad_x(x, y) = -2*x,

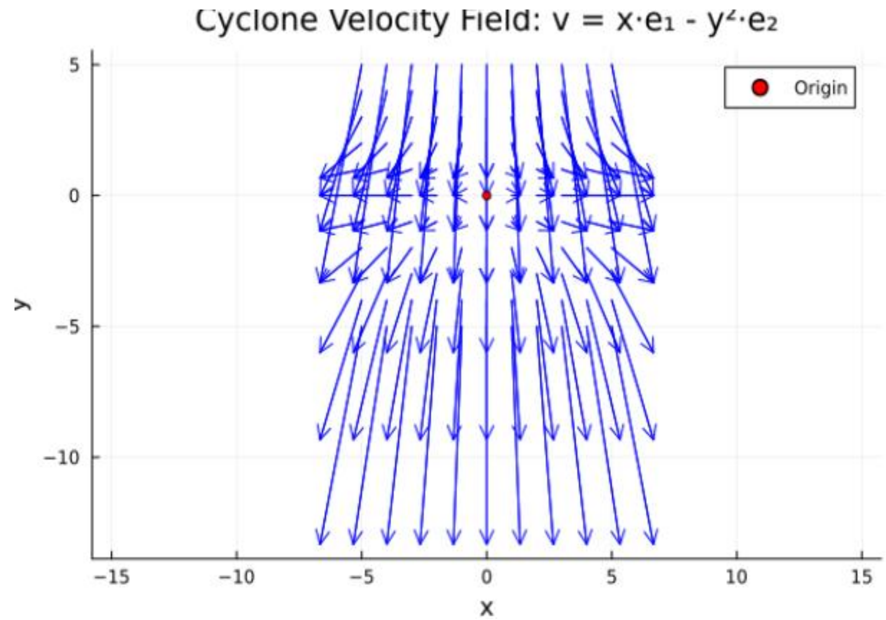
  Grad_y(x, y) = -4*y.
- X and Y are defined as grid of base positions of vectors.
- U = grad_x.(X, Y) , V = grad_y.(X, Y) are defined for grid.
- The field is evaluated across the grid and plotted using `quiver()`
- Both automatic and manual gradient results are identical.



Question 2: Cyclone Velocity Field

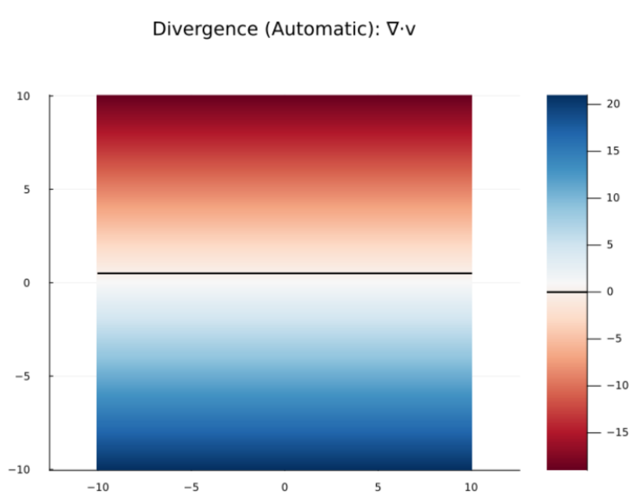
(a) Plotting the Vector Field

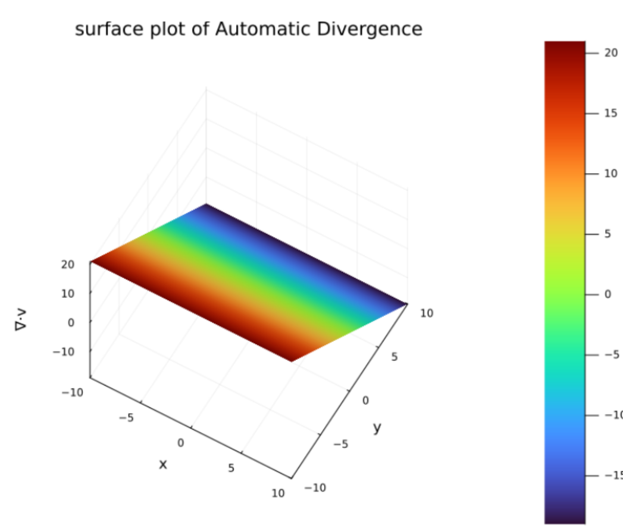
- The velocity field components are defined as v_x(x, y) = x , v_y(x, y) = -y^2
- X and Y are defined as grid of base positions of vectors.
- Value of vector field calculated by: U = v_x.(X, Y), V = v_y.(X, Y)
- The function is evaluated over a grid
- `quiver()` is used to plot arrows showing direction and magnitude of wind velocity

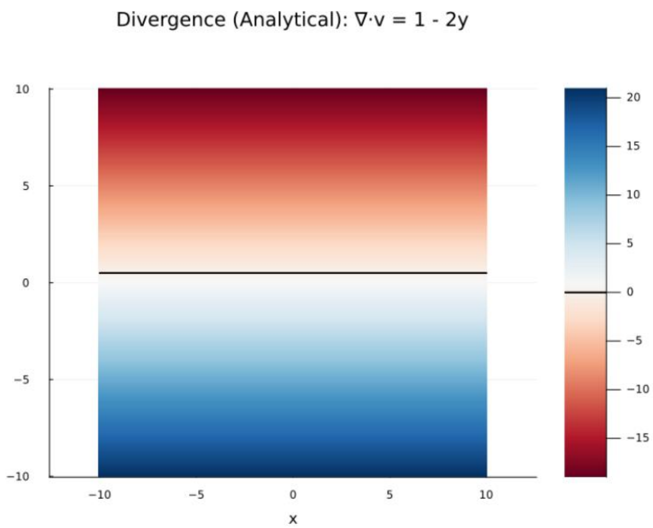


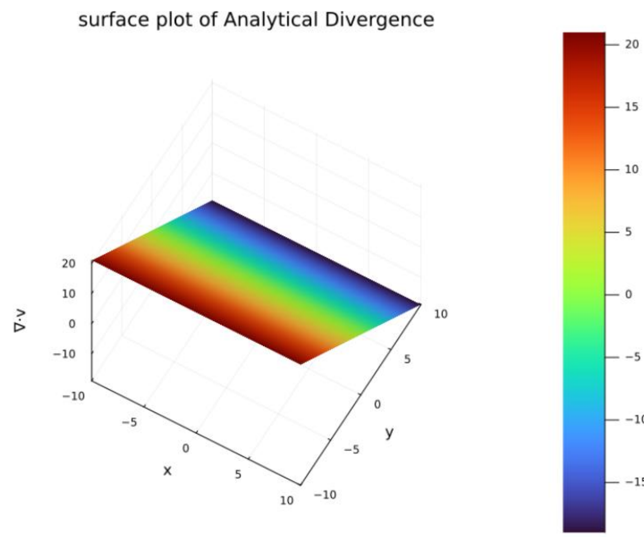
(b) Divergence Calculation and Comparison

- CalculusWithJulia is imported.
- Velocity field defined as: velocity_field(v)=[v[1], -v[2]^2] .
- Automatic divergence is computed using `divergence(velocity_field,[x,y])`
- Manual calculation uses partial derivatives: $\partial v_1/\partial x = 1$, $\partial v_2/\partial y = -2y$, resulting in divergence = 1 – 2y
- Both divergence results are evaluated and plotted using heatmap() and `surface()` for comparison
- Both results are identical



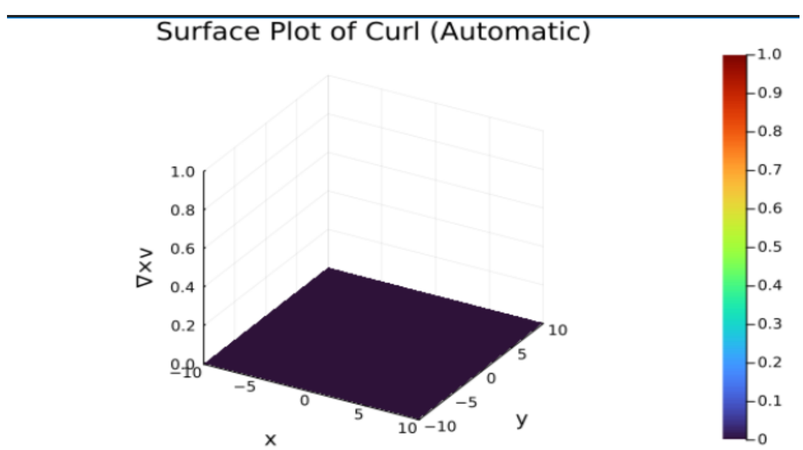






(a) Curl Calculation and Comparison

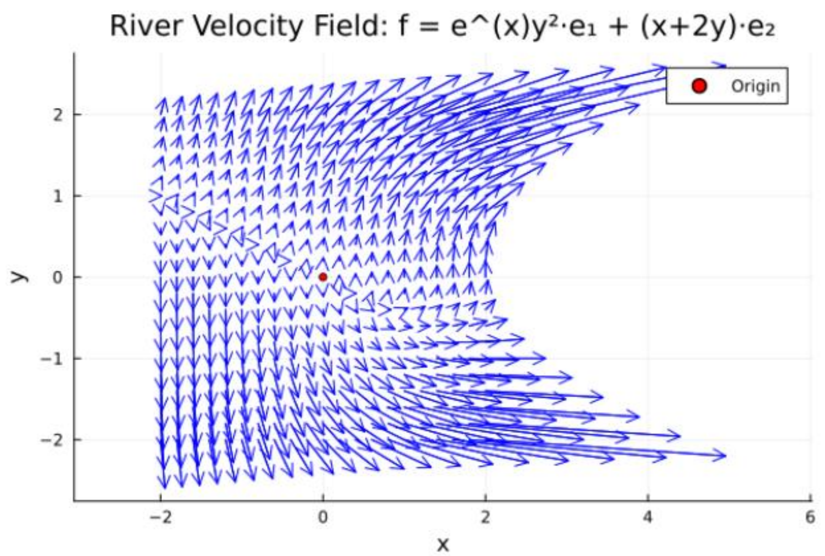
- Automatic curl is obtained using `curl_auto(x, y) = curl(velocity_field, [x, y])`
- Manual curl is computed as curl = $\partial v_2/\partial x - \partial v_1/\partial y$
- Both curl results are evaluated and plotted using `surface()` for comparison
- The resulting z-component is zero, confirming the flow is irrotational
- ∇×v = 0 means fluid particles do not rotate.
- The flow may converge/diverge but doesn't swirl



Question 3: Water Flow Velocity Field

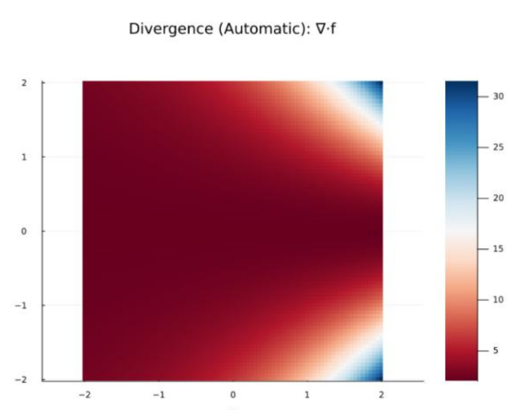
(a) Plotting the Vector Field

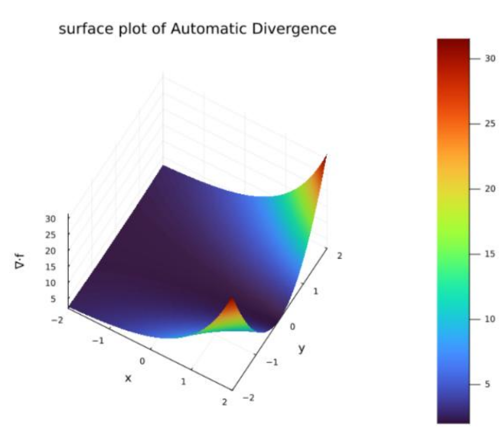
- The components of field are defined as `f_x(x, y)=exp(x)*y^2, f_y(x, y)=x + 2*y`
- X and Y are defined as grid of base positions of vectors.
- Field values calculated using U = f_x.(X, Y), V = f_y.(X, Y)
- The field is visualized with `quiver()` using computed component matrices

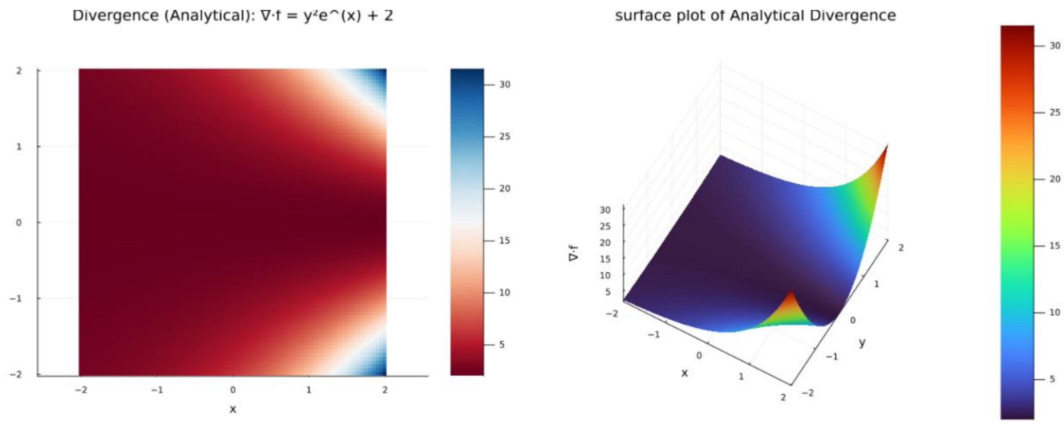


(b) Divergence Calculation and Comparison

- Automatic divergence is computed with `V_field_div(x,y)=divergence(V_field, [x,y])`
- Manual computation uses symbolic differentiation: $\partial(e^x y^2)/\partial x$ = exp(x)*y^2, $\partial(x + 2y)/\partial y = 2$
- The total divergence $e^x y^2 + 2$ is verified against the automatic result
- Both divergence results are evaluated and plotted using heatmap() and surface() for comparison
- Both plots are identical.

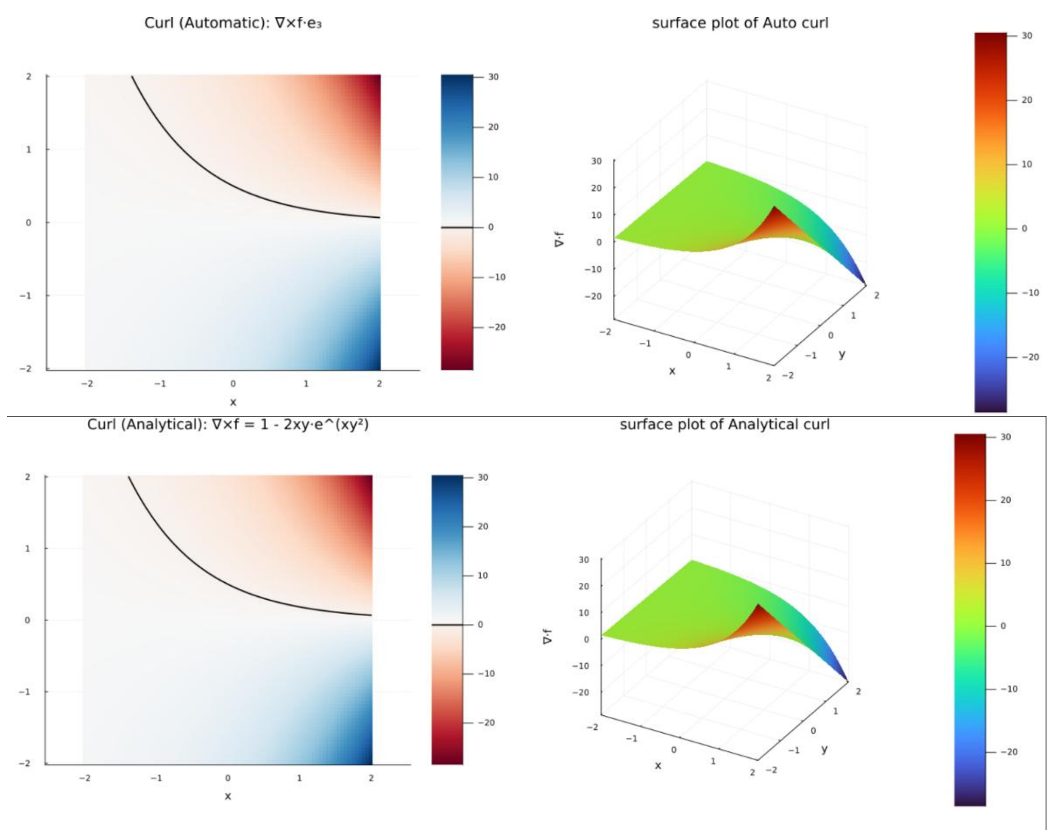






Curl Calculation and Comparison

- Automatic curl computed using `curl(V_field, v)`
- Manual computation gives curl = $\partial f_2/\partial x - \partial f_1/\partial y = 1 - 2ye^x$
- Both divergence results are evaluated and plotted using heatmap() and surface() for comparison
- Both automatic and manual curls are numerically equal.



Question 4: Beam Problem 1

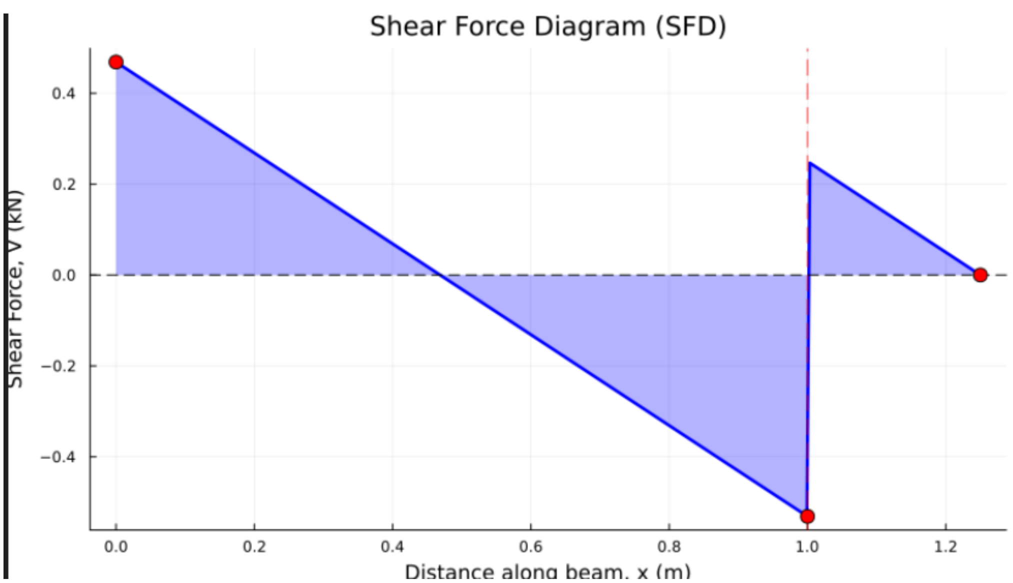
- Inputs L (length in m) distance bw supports A and B, q (UDL in kN/m) is UDL over entire length of beam.
- $L_{overhang}$=0.25L. Total length= L+ $L_{overhang}$.
- Equilibrium equations are used to calculate reactions R_a and R_b.
- To calculate shear force function Shear force() is used.

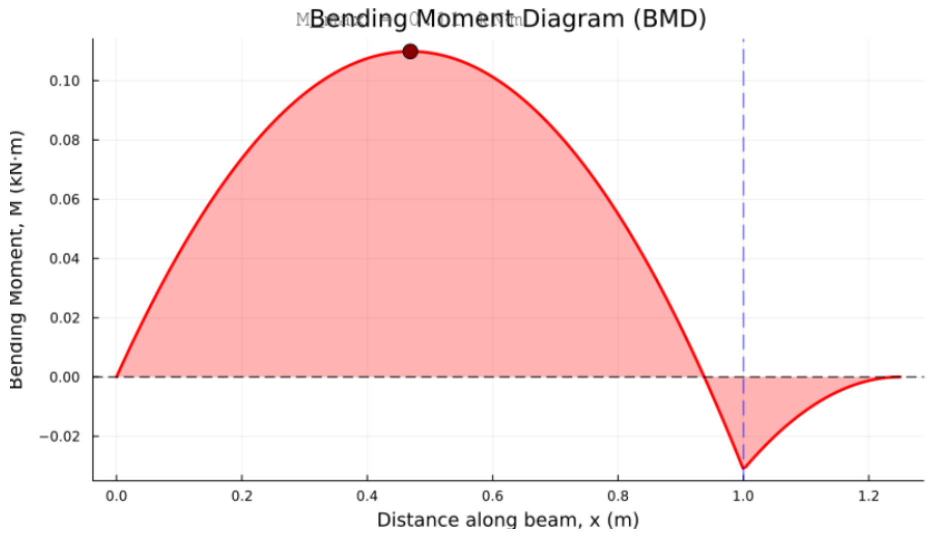
```
function shear_force(x, q, l, R_A, R_B)
    if x <= l
        return R_A - q * x
    else
        return R_A + R_B - q * (x)
    end
end
```

- To calculate Bending Moment function bending_moment() is used:

```
function bending_moment(x, q, l, R_A, R_B)
    if x <= l
        return R_A * x - (q * x^2) / 2
    else
        return R_A * x - (q * x^2) / 2 + R_B * (x - l)
    end
end
```

- Some key values like max BM and position of Zero shear are calculated.
- `plot()` is used to display both SFD and BMD .





Question 5: Beam Problem 2

- Inputs l (length in m) and q (UDL in kN/m) are taken.

```
# Beam configuration
P = 0.8 * q * l         # Point load at E
x_A = 0.0               # Position of support A (Roller)
x_E = 0.4 * l           # Position of point load E
x_D = 0.8 * l           # Position of HINGE D
x_B = 1.0 * l           # Position of support B (Roller)
x_C = 2.0 * l           # Position of support C (Pinned)
L_total = x_C           # Total beam length

# UDL from B to C

udl_start = x_B
udl_end = x_C
udl_length = udl_end - udl_start
```

- Equilibrium equations and equation of condition of internal hinge are used to calculate reactions at roller supports A and B as R_a and R_b and Pinned support R_c.

  - Function for SFD is shear_force()

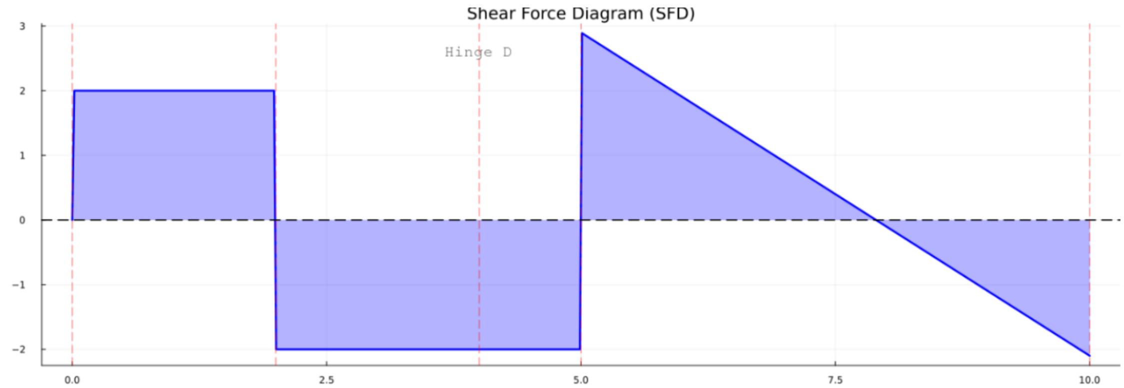
```
function shear_force(x, q, l, P, R_A, R_B, R_C)
    V = 0.0
    if x > x_A
        V += R_A
    end

    if x > x_E
        V -= P
    end

    if x > x_B
        V += R_B
    end
```

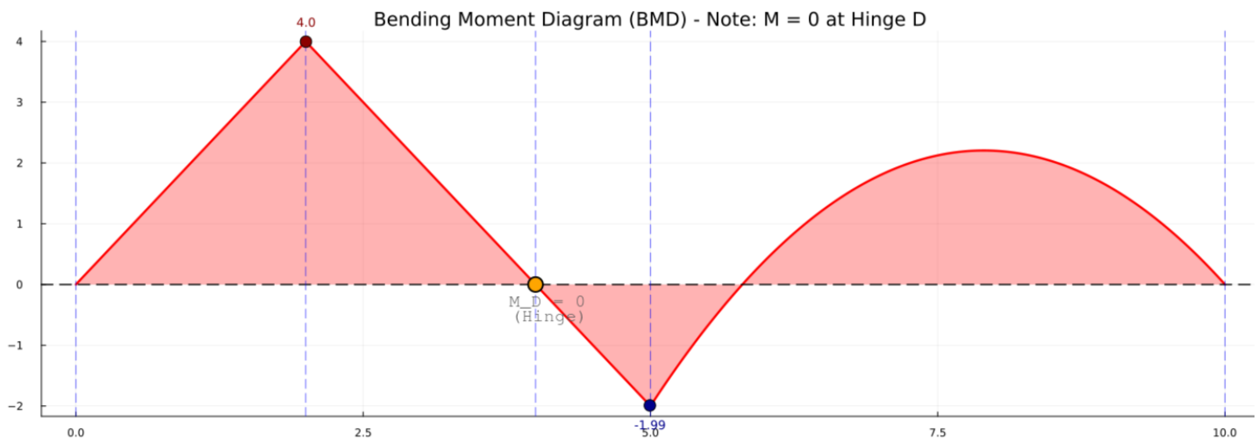
```
    if x > x_B && x <= x_C
        V -= q * (x - x_B)
    elseif x > x_C
        V -= q * (x_C - x_B)
    end

    # Adding reaction at C if we've passed it
    if x > x_C
        V += R_C
    end

    return V
end
```

- Function for Bending moment is :

```
function bending_moment(x, q, l, P, R_A, R_B, R_C)
    M = 0.0

    if x > x_A
        M += R_A * (x - x_A)
    end

    if x > x_E
        M -= P * (x - x_E)
    end

    if x > x_B
        M += R_B * (x - x_B)
    end

    if x > x_B && x <= x_C
        udl_dist = x - x_B
        M -= q * udl_dist * (udl_dist / 2)
    elseif x > x_C
        udl_dist = x_C - x_B
        M -= q * udl_dist * (x - x_B - udl_dist / 2)
    end

    # Contribution from reaction at C
    if x > x_C
        M += R_C * (x - x_C)
    end

    return M
end
```

- .Some of the key values of the BM are calculated.
- `plot()` is used to display both SFD and BMD


Shear Force Diagram (SFD)
Hinge D


Bending Moment Diagram (BMD) - Note: M = 0 at Hinge D
4.0
M_D = 0
(Hinge)
-1.99