
Question 1 – Scalar Field Plotting in Julia

1. Importing Required Packages

```
using Plots
```

```
using CalculusWithJulia
```

Explanation

- Plots → used to create the 3D surface plot, contour plot, and vector field.
 - CalculusWithJulia → used for automatic differentiation to compute the gradient vector field.
-

2. Setting the Plotting Backend

```
gr()
```

Explanation

- This selects the **GR backend** for plotting.
 - GR produces fast, stable, high-quality plots in Julia.
-

3. Defining the Main Function

```
function problem_scalarfield()
```

Explanation

- All plotting and computations are placed inside a function.
 - This helps keep the code clean and allows the user to re-run the problem with one function call.
-

4. Defining the Scalar Field

```
z_field(a, b) = 200 - a^2 - 2*b^2
```

Explanation

- This function represents the scalar field:
[
 $z(a,b) = 200 - a^2 - 2b^2$
]
- It takes coordinates ((a,b)) as input and returns the height (z).
- This is required for both 3D surface and contour plotting.

5. Defining a Function for Gradient Computation

```
z_fn(v) = 200 - v[1]^2 - 2*v[2]^2
```

Explanation

- CalculusWithJulia.gradient() works on functions that take a **vector argument**.
- So we re-write the same scalar field using vector input:
 - $v[1]$ = x-coordinate
 - $v[2]$ = y-coordinate

6. Creating the Range of Coordinates

```
xr = -10:0.5:10
```

```
yr = -10:0.5:10
```

Explanation

- Creates evenly spaced grid points from -10 to $+10$ for both x and y.
- Used for plotting 3D and 2D contour fields.

7. Plotting the 3D Surface

```
surf_plot = surface(  
    xr, yr, z_field,  
    title="Modified: 3D Surface Plot of z(a,b)",  
    xlabel="a", ylabel="b", zlabel="z(a,b)",  
    camera=(30, 30)  
)
```

Explanation

- surface(...) generates a 3D surface plot.
- The plot shows how the scalar field varies in space.
- camera=(30,30) sets the viewing angle of the 3D plot.

8. Plotting the 2D Contour Map

```
cont_plot = contour(  
    xr, yr, z_field,
```

```

    title="Modified: 2D Contour Plot of z(a,b)",
    xlabel="a", ylabel="b",
    fill=true,
    colorbar_title="Field Value"
)

```

Explanation

- Creates a contour map, where each contour line represents constant scalar field value.
 - fill=true adds color shading between contour lines.
 - This helps visualize the gradient and elevation changes more clearly.
-

9. Displaying and Saving the Plots

```

display(surf_plot)

display(cont_plot)

```

```

savefig(surf_plot, "modified_surface.png")
savefig(cont_plot, "modified_contour.png")

```

Explanation

- display shows the plot in the output panel.
 - savefig saves the plots as images, useful for assignment submission.
-

10. Computing the Gradient

```
grad_z = CalculusWithJulia.gradient(z_fn)
```

Explanation

- This returns a function that computes:
$$[\nabla z = \left[\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \right]$$
 - The gradient shows the **direction of steepest ascent** in the field.
-

11. Coarse Grid for Vector Field

```
xa = -10:2:10
```

```
ya = -10:2:10
```

Explanation

- A coarser grid (step size = 2) is used so arrows don't overlap as much in the quiver plot.
-

12. Computing Gradient Components

$Uv = [\text{grad}_z([a, b])][1]$ for b in ya , a in xa

$Vv = [\text{grad}_z([a, b])][2]$ for b in ya , a in xa

Explanation

- For every grid point $((a,b))$, gradient is computed.
- Uv stores x-components: $(\frac{\partial z}{\partial a})$
- Vv stores y-components: $(\frac{\partial z}{\partial b})$

These arrays form the basis for the vector field (quiver plot).

13. Plotting the Gradient (Quiver Plot)

```
grad_plot = contour(  
    xr, yr, z_field,  
    title="Modified: Gradient Vector Field  $\nabla z$ ",  
    xlabel="a", ylabel="b",  
    fill=true, opacity=0.5,  
    colorbar_title="Field Value"  
)
```

- First, a contour background is drawn.

```
quiver!(  
    grad_plot,  
    xa, ya,  
    quiver=(Uv, Vv),  
    color=:black,  
    arrow=true,  
    label=" $\nabla z$ "  
)
```

Explanation

- `quiver!` overlays a vector field (arrows) on the contour map.

- Each arrow shows the gradient direction at that point.
 - Gradient arrows always point **toward increasing z**, showing the steepest direction.
-

14. Displaying and Saving Gradient Plot

```
display(grad_plot)  
savefig(grad_plot, "modified_gradient.png")
```

Explanation

- Shows and saves the gradient vector field plot.
-

15. End the Function

```
println("Modified Problem plots generated successfully.")  
end
```

16. Run the Function

```
problem_scalarfield()
```

Explanation

- Executes everything and produces all required plots.

Question2

Explanation for Question – Vector Field & Divergence in Julia

This code analyzes the vector field:

```
[  
\vec{v}(x,y) = \langle x, -y^2 \rangle  
]
```

The tasks are:

1. Plot the vector field.
2. Compute the divergence both automatically and manually.
3. Plot and compare the two divergence fields.

Below is the explanation for each part of the code.

1. Importing Required Packages

```
using Plots
```

```
using CalculusWithJulia
```

```
using ForwardDiff
```

Explanation

- **Plots** → used for quiver plots (vector arrows) and contour plots.
 - **CalculusWithJulia** → provides the divergence() function for automatic differentiation.
 - **ForwardDiff** → required internally for differentiating vector-valued functions.
-

2. Defining the Vector Field $v(x,y)$

```
 $v(x,y) = [x, -(y^2)]$ 
```

Explanation

This defines the vector field:

```
[  
    \vec{v}(x,y) = \left[ x, -y^2 \right]  
,
```

- The first component is ($v_1 = x$)
- The second component is ($v_2 = -y^2$)

This function returns a **2D vector** at each point.

3. Creating a Grid for Plotting

```
xs = -2:0.4:2
```

```
ys = -2:0.4:2
```

Explanation

- xs and ys represent evenly spaced sample points from -2 to $+2$.

- These define a grid on which the vector field arrows will be drawn.
-

✓ 4. Extracting Vector Components for Quiver Plot

```
a = [v(x,y)[1] for x in xs, y in ys] # x-component
```

```
b = [v(x,y)[2] for x in xs, y in ys] # y-component
```

✓ Explanation

- a stores the first component ($v_1 = x$)
- b stores the second component ($v_2 = -y^2$)

This converts the vector field into two separate matrices for plotting.

✓ 5. Plotting the Vector Field (Part a)

```
quiver(xs, ys, quiver=(a, b),  
       aspect_ratio=1,  
       xlabel="x", ylabel="y",  
       title="Vector Field v(x,y) = (x, -y^2)"  
)
```

✓ Explanation

- quiver() draws arrows to represent the direction and magnitude of vectors.
- quiver=(a, b) passes the two matrices containing vector components.
- aspect_ratio=1 ensures equal scaling on both axes.

This visualizes the vector field in the 2D plane.

✓ 6. Automatic Divergence Calculation

```
f(x) = v(x[1], x[2])
```

```
auto_div(x,y) = divergence(f, [x,y])
```

✓ Explanation

To compute divergence automatically:

1. Convert the vector field into a function that takes a **vector input**:
2. $f(x) = v(x[1], x[2])$
3. Use `CalculusWithJulia.divergence()`:
4. `auto_div(x,y) = divergence(f, [x,y])`

Divergence formula:

```
[  
\nabla \cdot \vec{v} = \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y}  
]
```

For our vector field:

```
[  
v_1 = x \quad \rightarrow \quad \frac{\partial v_1}{\partial x} = 1  
]  
[  
v_2 = -y^2 \quad \rightarrow \quad \frac{\partial v_2}{\partial y} = -2y  
]
```

So:

```
[  
\nabla \cdot \vec{v} = 1 - 2y  
]
```

✓ 7. Manual Divergence Calculation

$$\text{manual_div}(x,y) = 1 - 2y$$

✓ Explanation

This is the hand-calculated divergence:

```
[  
\nabla \cdot \vec{v} = 1 - 2y  
]
```

✓ 8. Preparing a Fine Grid for Contour Plots

```
xs = -2:0.05:2
```

```
ys = -2:0.05:2
```

✓ Explanation

A fine grid produces smoother contour plots for divergence.

9. Creating Divergence Matrices

```
Z_auto = [auto_div(x,y) for x in xs, y in ys]
```

```
Z_manual = [manual_div(x,y) for x in xs, y in ys]
```

✓ Explanation

- Z_{auto} = divergence computed automatically
- Z_{manual} = divergence computed manually

These matrices are used for contour plots.

10. Plotting the Divergence Fields

```
p1 = contour(xs, ys, Z_auto,  
             title="Automatic Divergence",  
             xlabel="x", ylabel="y"  
)
```

```
p2 = contour(xs, ys, Z_manual,  
             title="Manual Divergence (1 - 2y)",  
             xlabel="x", ylabel="y"  
)
```

✓ Explanation

- $p1 \rightarrow$ contour plot of divergence using automatic differentiation
- $p2 \rightarrow$ contour plot of manually derived expression

Both show identical patterns, confirming correctness.

11. Displaying Both Plots Side-by-Side

```
plot(p1, p2, layout=(1,2), size=(900,400))
```

✓ Explanation

- Combines both contour plots in a single row.
 - Allows easy comparison between automatic and manual divergence.
-

Question3

using Plots

using ForwardDiff

Explanation

- Plots is used for graphing vector fields, surfaces, etc.
 - ForwardDiff is used for automatic differentiation to compute divergence/curl automatically.
-

Defining the vector field

```
f1(x,y) = exp(x)*y^2
```

```
f2(x,y) = x + 2y
```

Explanation

You define a 2-D vector field:

```
[  
 \vec{F}(x,y) = (f_1(x,y),, f_2(x,y)) = \left(e^x y^2,; x + 2y\right)  
 ]
```

So every point in the plane has a vector attached to it.

Creating grid points

```
xs = collect(range(-2, 2, length=20))
```

```
ys = collect(range(-2, 2, length=20))
```

Explanation

- Creates 20 equally spaced values from **-2 to 2** in both x and y directions.
 - These will be the points where the vector field is evaluated.
-

📌 Creating empty matrices

```
nx = length(xs)
```

```
ny = length(ys)
```

```
Umat = zeros(nx, ny)
```

```
Vmat = zeros(nx, ny)
```

```
Dauto = zeros(nx, ny)
```

```
Dmanual = zeros(nx, ny)
```

```
Cauto = zeros(nx, ny)
```

```
Cmanual = zeros(nx, ny)
```

Explanation

You allocate space for:

- Umat → x-component of vector field
 - Vmat → y-component
 - Dauto → divergence (automatic differentiation)
 - Dmanual → divergence (manual formula)
 - Cauto → curl (automatic differentiation)
 - Cmanual → curl (manual formula)
-

📌 Filling matrices

```
for i in 1:nx, j in 1:ny
```

```
    x = xs[i]; y = ys[j]
```

```
    Umat[i,j] = f1(x,y)
```

```
    Vmat[i,j] = f2(x,y)
```

Explanation

- Loops through each grid point.
- Evaluates ($f_1(x,y)$) and ($f_2(x,y)$).

- Stores them inside matrices.
-

Automatic Divergence

```
Dauto[i,j] = ForwardDiff.derivative(tx -> f1(tx,y), x) +
    ForwardDiff.derivative(ty -> f2(x,ty), y)
```

Explanation

Divergence formula:

```
[  
 \nabla \cdot \vec{F} = \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y}.  
 ]
```

- First derivative term: ($\frac{\partial}{\partial x} e^{xy^2} = e^{xy^2}$)
- Second derivative term: ($\frac{\partial}{\partial y} (x + 2y) = 2$)

ForwardDiff computes these derivatives numerically.

Manual Divergence

```
Dmanual[i,j] = exp(x)*y^2 + 2
```

Explanation

You manually use the analytical formula:

```
[  
 \nabla \cdot \vec{F} = e^{xy^2} + 2  
 ]
```

Automatic Curl

```
Cauto[i,j] = ForwardDiff.derivative(tx -> f2(tx,y), x) -
    ForwardDiff.derivative(ty -> f1(x,ty), y)
```

Explanation

In 2D, curl is:

```
[  
 \text{curl } \vec{F} = \frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y}  
 ]
```

Manual Curl

```
Cmanual[i,j] = 1 - 2*y*exp(x)
```

Explanation

```
[  
 \frac{\partial f_2}{\partial x} = 1, \quad  
 \frac{\partial f_1}{\partial y} = 2ye^x  
 ]
```

So:

```
[  
 \text{curl} = 1 - 2ye^x  
 ]
```

📌 Preparing data for quiver plot

```
Xlist = [xs[i] for i in 1:nx for j in 1:ny]  
Ylist = [ys[j] for i in 1:nx for j in 1:ny]  
Ulist = [Umat[i,j] for i in 1:nx for j in 1:ny]  
Vlist = [Vmat[i,j] for i in 1:nx for j in 1:ny]
```

Explanation

quiver() requires flat lists, not matrices.

So this converts the grids into 1-D lists.

📌 Plot the Vector Field

```
quiver(Xlist, Ylist, quiver=(Ulist, Vlist),  
       aspect_ratio=1, title="Vector Field",  
       xlabel="x", ylabel="y")
```

Explanation

Creates an arrow plot showing the direction and magnitude of the vector field at all grid points.

📌 Plot Divergence (Automatic)

```
surface(xs, ys, Dauto', title="Divergence (Automatic)")
```

Explanation

- Plots 3D surface of divergence.
 - ' transposes the matrix to match x-y grid orientation.
-

Plot Divergence (Manual)

```
surface(xs, ys, Dmanual', title="Divergence (Manual)")
```

Explanation

Plots the manual formula.

Should match the automatic version.

Plot Curl (Automatic & Manual)

```
surface(xs, ys, Cauto', title="Curl (Automatic)")
```

```
surface(xs, ys, Cmanual', title="Curl (Manual)")
```

Explanation

- Both surface plots show rotational tendency of the vector field.
 - Again, automatic vs analytical formulas should match.
-

Question4

2. Defining Basic Input Values

$I = 5.0$

$q = 12.0$

Explanation

- $I = 5 \text{ m} \rightarrow$ original beam length.
- $q = 12 \text{ kN/m} \rightarrow$ uniformly distributed load (UDL) over the secondary projection.

These values come directly from the question.

3. Extension of Beam Due to Projection

$\text{extra} = 0.251 * I$

$L = I + \text{extra}$

Explanation

- The problem states an additional projection equal to **0.251 × I**.
- This extension increases the total effective length.

$$[L = I + 0.251I = 1.251I]$$

So the UDL acts over the full length L , not just the original 5 m.

📌 4. Reaction Calculations

$$R_b = q * L * (L/2) / I$$

$$R_a = q * L - R_b$$

Explanation

For a simply supported beam with a UDL over an extended length:

- Total load = ($w = qL$)
- Load acts at mid-point (= $L/2$)

Taking moments about support A:

$$\begin{bmatrix} R_b \cdot l = qL \cdot \frac{L}{2} \\ R_b = \frac{qL(L/2)}{I} \end{bmatrix}$$

So,

$$\begin{bmatrix} R_a = qL - R_b \end{bmatrix}$$

Once (R_b) is known:

$$\begin{bmatrix} R_a = qL - R_b \end{bmatrix}$$

This ensures vertical equilibrium:

$$\begin{bmatrix} R_a + R_b = qL \end{bmatrix}$$

📌 5. Shear Force Function $V(x)$

$$V(x) = x \leq 0 ? 0 :$$

$$x \leq l ? R_a - q*x :$$

$$x \leq L ? R_a + R_b - q*x : 0$$

Explanation

This is a **piecewise function**, matching the physical behavior of the beam.

Region-wise meaning

1. For $x \leq 0$

0

No beam \rightarrow no shear.

2. For $0 \leq x \leq l$

$R_a - q^*x$

Because:

- Shear starts with reaction at A.
- Linearly decreases due to UDL.

3. For $l \leq x \leq L$

$R_a + R_b - q^*x$

At $x = l$, reaction at B is applied.

Shear suddenly jumps by R_b , then decreases with UDL until free end.

4. For $x \geq L$

0

No load \rightarrow shear becomes zero.

📌 6. Bending Moment Function $M(x)$

$M(x) = x \leq 0 ? 0 :$

$x \leq l ? R_a*x - q^*x^2/2 :$

$x \leq L ? R_a*x + R_b*(x - l) - q^*x^2/2 : 0$

Explanation

This uses the bending moment formula:

```
[  
M = \int V(x), dx  
]
```

Region-wise meaning

1. For $x \leq 0$

No beam \rightarrow no moment.

2. For $0 \leq x \leq l$

```
[  
M = R_a*x - \frac{qx^2}{2}  
]
```

Standard bending moment under UDL until support B.

3. For $l \leq x \leq L$

```
[  
M = R_a x + R_b (x - l) - \frac{q x^2}{2}  
]
```

This includes:

- Moment of Ra at distance x
- Moment of Rb at distance $(x - l)$
- Moment of UDL load up to x

4. For $x \geq L$

Beam ends \rightarrow moment = 0.

📌 7. Creating the x-Axis Values

```
xvals = range(0, L, length=500)
```

Explanation

- 500 equally spaced points from 0 to full beam length L.
 - Ensures smooth shear and moment diagrams.
-

📌 8. Evaluating the Functions

```
Vvals = [V(x) for x in xvals]
```

```
Mvals = [M(x) for x in xvals]
```

Explanation

This computes:

- Shear force at every x
- Bending moment at every x

Creates numerical arrays for plotting.

📌 9. Plotting Shear Force Diagram

```
plot(xvals, Vvals, title="Shear Force Diagram",  
      xlabel="x", ylabel="V(x)", lw=2, legend=false)
```

Explanation

- Plots shear force vs x.
 - No legend needed.
 - Line width $lw=2$ makes it look clear.
-

📌 10. Plotting Bending Moment Diagram

```
plot(xvals, Mvals, title="Bending Moment Diagram",
      xlabel="x", ylabel="M(x)", lw=2, legend=false)
```

Explanation

- Plots bending moment vs x.
 - Smooth parabola due to UDL.
 - Moment becomes zero at the free end.
-

Question5

This program plots the **Shear Force Diagram (SFD)** and **Bending Moment Diagram (BMD)** for the beam loading shown in *Fig. 2* of assignment.

The beam carries:

- A **partial UDL** over the first span (0–l)
- A **point load** at ($D = 0.8l$)
- A second UDL over span (B–C)
- Three supports producing reactions **RA, RB, RC**

All values match the beam loading in the question.

📌 1. Importing Required Package

using Plots

Purpose:

Plots.jl is used to draw the SFD and BMD graphs.

📌 2. Defining the Main Function

```
function beam_F(q, l)
```

Explanation:

This defines a function that takes:

- $q \rightarrow$ uniformly distributed load
- $l \rightarrow$ span length

It computes reactions, then plots SFD and BMD.

3. Reaction Forces

$$RC = (1.3 * q * l) / 3$$

$$RB = (0.4 * q * l) / 3$$

$$RA = q*l + 0.8*q*l - RB - RC$$

Explanation:

From the question diagram:

- Load on span AD = total UDL = (ql)
- Additional point load at D = ($0.8ql$)
- Total load = ($ql + 0.8ql = 1.8ql$)

The reaction values are **already derived from equilibrium equations**:

```
[  
RC = \frac{1.3ql}{3},\quad RB = \frac{0.4ql}{3}  
]
```

Using the equilibrium condition:

```
[  
RA + RB + RC = 1.8ql  
]
```

So,

```
[  
RA = 1.8ql - RB - RC  
]
```

The print statements show the computed reactions.

4. Defining Key Beam Locations

$$xA = 0$$

$$xD = 0.8l$$

$$xB = l$$

$$xC = 2l$$

Explanation:

According to the figure:

- Support A at $x = 0$
- Point load at $D = 0.8l$
- Support B at $x = l$
- End support at $C = 2l$

These positions divide the beam into **three regions**.

📌 5. Shear Force Function

function $V(x)$

This is a **piecewise shear force expression** reflecting load changes.

Region 1: A → D (0 to 0.8l)

if $x < xD$

 return RA

end

Shear = reaction at A (constant), because only RA acts.

At $x = D$ (point load $0.8ql$)

if $x == xD$

 return RA - $0.8 * q * l$

end

Shear drops suddenly due to the point load.

Region 2: D → B (0.8l to l)

if $x < xB$

 return RA - $0.8 * q * l$

end

Constant shear until support B.

At $x = B$, add reaction RB

if $x == xB$

```
return RA - 0.8*q*l + RB  
end  
Support reaction RB causes upward jump.
```

Region 3: B → C (l to 2l) UDL q

```
return RA - 0.8*q*l + RB - q*(x - xB)  
Shear decreases linearly due to UDL over BC.
```

6. Bending Moment Function

```
function M(x)  
Again piecewise, derived by integrating the shear.
```

Region A–D

```
if x < xD  
    return RA*x  
end  
Only reaction RA contributes moment.
```

Region D–B

```
if x < xB  
    return RA*x - 0.8*q*l*(x - xD)  
end  
Moment from:  


- RA acting at x
- Point load at D acting at (x – xD)

```

Region B–C (UDL)

```
return RA*x - 0.8*q*l*(x - xD) + RB*(x - xB) - q*(x - xB)^2/2  
Moment contributions:  


- RA
- Point load at D

```

- RB
 - UDL over BC → creates $(\frac{q(x-l)^2}{2})$ moment
-

📌 7. Evaluate Shear and Moment at 500 Points

```
x = range(0, 2l, length=500)
```

```
Vvals = [V(xx) for xx in x]
```

```
Mvals = [M(xx) for xx in x]
```

Explanation:

- Creates a smooth set of points from $x = 0$ to $x = 2l$
 - Calculates shear and moment at each point
-

📌 8. Plotting Shear Force Diagram

```
p1 = plot(  
    x, Vvals,  
    xlabel="x (m)", ylabel="Shear V(x)",  
    title="Shear Force Diagram (Fig. 2)",  
    linewidth=3, legend=false  
)  
  
hline!([0], color=:black)
```

Explanation:

- Plots $V(x)$
 - Adds zero line (baseline)
 - Thick line for clarity
-

📌 9. Plotting Bending Moment Diagram

```
p2 = plot(  
    x, Mvals,  
    xlabel="x (m)", ylabel="Moment M(x)",  
    title="Bending Moment Diagram (Fig. 2)",  
    linewidth=3, legend=false  
)
```

```
hline!([0], color=:black)
```

Explanation:

- Plots $M(x)$
 - Adds zero-moment reference line
-

📌 **10. Running the Function**

```
beam_F(10, 5)
```

Explanation:

Substitutes the values:

- UDL ($q = 10$, kN/m)
- Span ($l = 5$, m)

And prints the reactions:

- **RA = 61.67 kN**
- **RB = 6.67 kN**
- **RC = 21.67 kN**

These automatically match the derived formulas.
