Julia-Assignment 1

Akanksha Kurre

22CE02013

```
1.using Plots
using SymPy
gr()  # or plotlyjs() for interactive plots

function q1_class()
   println("Running Problem 1: Scalar Field h(x,y) = 200 - x^2 - 2y^2")
   x, y = symbols("x y")
   h_expr = 200 - x^2 - 2*y^2
   dh_dx = diff(h_expr, x)
   dh_dy = diff(h_expr, y)
   println("∂h/∂x = $dh_dx ,   ∂h/∂y = $dh_dy")
   h_fun   = lambdify(h_expr, [x, y])
   dhx_fun = lambdify(dh_dx, [x, y])
   dhy_fun = lambdify(dh_dy, [x, y])
   x_range = -10:0.5:10
   y_range = -10:0.5:10
   p1_3d = surface(x_range, y_range, (xx, yy) -> h_fun(xx, yy),
               title="Q1(a): 3D Surface Plot of h(x,y)",
               xlabel="x", ylabel="y", zlabel="h(x,y)",
               camera=(30, 30))
   display(p1_3d)
   savefig(p1_3d, "Q1_surface.png")
   p1_2d = contour(x_range, y_range, (xx, yy) -> h_fun(xx, yy),
               title="Q1(b): 2D Contour Plot of h(x,y)",
               xlabel="x", ylabel="y", fill=true, colorbar_title="h(x,y)")
   display(p1_2d)
   savefig(p1_2d, "Q1_contour.png")
```

```
    x_grid = -10:2:10
    y_grid = -10:2:10
    U = [dhx_fun(xx, yy) for yy in y_grid, xx in x_grid]
    V = [dhy_fun(xx, yy) for yy in y_grid, xx in x_grid]

    p1_grad = contour(x_range, y_range, (xx, yy) -> h_fun(xx, yy),
                  fill=true, opacity=0.5, xlabel="x", ylabel="y",
                  title="Q1(c): Gradient Vector Field ∇h", colorbar_title="h(x,y)")
    quiver!(p1_grad, x_grid, y_grid, quiver=(U, V), color=:black, arrow=true, label="∇h")
    display(p1_grad)
    savefig(p1_grad, "Q1_gradient_field.png")
end
q1_class()
```

2.using Plots

```
using SymPy
gr()

function q2_class()
    println("Running Problem 2: Vector Field v(x,y) = x*e₁ - y²*e₂")
    x, y = symbols("x y")
    v₁ = x
    v₂ = -y^2
    div_v = diff(v₁, x) + diff(v₂, y)
    curl_v = diff(v₂, x) - diff(v₁, y)

    println("∇·v = $div_v")
    println("∇×v = $curl_v")
    v1_fun = lambdify(v₁, [x, y])
    v2_fun = lambdify(v₂, [x, y])
    div_fun = lambdify(div_v, [x, y])
    curl_fun = lambdify(curl_v, [x, y])
    x_range = -5:0.5:5
    y_range = -5:0.5:5
    U = [v1_fun(xx, yy) for yy in y_range, xx in x_range]
    V = [v2_fun(xx, yy) for yy in y_range, xx in x_range]

    p2_vec = quiver(x_range, y_range, quiver=(U, V),
                title="Q2(a): Vector Field v(x,y) = [x, -y²]",
                xlabel="x", ylabel="y", aspect_ratio=:equal)
```

```
    display(p2_vec)
    savefig(p2_vec, "Q2_vector_field.png")
    div_data = [div_fun(xx, yy) for yy in y_range, xx in x_range]
    p2_div = contourf(x_range, y_range, div_data,
                      title="Q2(b): Divergence ∇·v = 1 - 2y",
                      xlabel="x", ylabel="y", colorbar_title="∇·v")
    display(p2_div)
    savefig(p2_div, "Q2_divergence.png")
    curl_data = [curl_fun(xx, yy) for yy in y_range, xx in x_range]
    p2_curl = contourf(x_range, y_range, curl_data,
                       title="Q2(c): Curl ∇×v = 0",
                       xlabel="x", ylabel="y", colorbar_title="∇×v")
    display(p2_curl)
    savefig(p2_curl, "Q2_curl.png")
end
q2_class()
```

```
3. using Plots
using SymPy
gr()

function q3_class()
    println("Running Problem 3: Vector Field f(x,y) = (e^x * y²)e₁ + (x + 2y)e₂")
    x, y = symbols("x y")
    f₁ = exp(x) * y^2
    f₂ = x + 2*y
    div_f = diff(f₁, x) + diff(f₂, y)
    curl_f = diff(f₂, x) - diff(f₁, y)

    println("∇·f = $div_f")
    println("∇×f = $curl_f")
    f1_fun = lambdify(f₁, [x, y])
    f2_fun = lambdify(f₂, [x, y])
    div_fun = lambdify(div_f, [x, y])
    curl_fun = lambdify(curl_f, [x, y])
    x_range = -2:0.2:2
    y_range = -2:0.2:2
    U = [f1_fun(xx, yy) for yy in y_range, xx in x_range]
    V = [f2_fun(xx, yy) for yy in y_range, xx in x_range]

    p3_vec = quiver(x_range, y_range, quiver=(U, V),
                title="Q3(a): Vector Field f(x,y)",
                xlabel="x", ylabel="y", aspect_ratio=:equal)
    display(p3_vec)
```

```
    savefig(p3_vec, "Q3_vector_field.png")
    div_data = [div_fun(xx, yy) for yy in y_range, xx in x_range]
    p3_div = contourf(x_range, y_range, div_data,
                    title="Q3(b): Divergence ∇·f = e^x·y² + 2",
                    xlabel="x", ylabel="y", colorbar_title="∇·f")
    display(p3_div)
    savefig(p3_div, "Q3_divergence.png")
    curl_data = [curl_fun(xx, yy) for yy in y_range, xx in x_range]
    p3_curl = contourf(x_range, y_range, curl_data,
                     title="Q3(c): Curl ∇×f = 1 - 2y·e^x",
                     xlabel="x", ylabel="y", colorbar_title="∇×f")
    display(p3_curl)
    savefig(p3_curl, "Q3_curl.png")
end
q3_class()
```

4. using Plots

```
using SymPy
gr()

function q4_class()
    println("Running Problem 4 : Beam with Overhang (UDL = q over 1.25 l)")
    x, l, q, RA, RB = symbols("x l q RA RB")
    eq1 = Eq(RA + RB, q*(1.25*l))                   # ΣFy = 0
    eq2 = Eq(RB*l - q*(1.25*l)*(1.25*l/2), 0)       # ΣM_A = 0

    sol = solve([eq1, eq2], [RA, RB])
    RA_expr, RB_expr = sol[RA], sol[RB]
    println("Reactions:")
    println("RA = $RA_expr")
    println("RB = $RB_expr")
    # Region 1: 0 ≤ x ≤ l
    V1 = RA_expr - q*x
    M1 = RA_expr*x - q*x^2/2
    # Region 2: l ≤ x ≤ 1.25l
    V2 = RA_expr + RB_expr - q*x
    M2 = RA_expr*x + RB_expr*(x - l) - q*x^2/2
    l_val = 10.0
    q_val = 5.0
    subs_pairs = Dict(l => l_val, q => q_val)

    RA_val = N(subs(RA_expr, subs_pairs))
    RB_val = N(subs(RB_expr, subs_pairs))
```

```
    println("Numeric values → RA = $RA_val , RB = $RB_val")
    V1f = lambdify(subs(V1, subs_pairs), [x])
    V2f = lambdify(subs(V2, subs_pairs), [x])
    M1f = lambdify(subs(M1, subs_pairs), [x])
    M2f = lambdify(subs(M2, subs_pairs), [x])
    x1 = 0:0.01:l_val
    x2 = l_val:0.01:1.25*l_val
    V1v, V2v = V1f.(x1), V2f.(x2)
    M1v, M2v = M1f.(x1), M2f.(x2)
    p4s = plot(x1, V1v, label="0–l", fillrange=0, fillalpha=0.3,
            title="Q4 (a): Shear Force Diagram", xlabel="x (m)", ylabel="V (kN)")
    plot!(p4s, x2, V2v, label="l–1.25l", fillrange=0, fillalpha=0.3)
    hline!(p4s, [0], linestyle=:dash, label=false)
    display(p4s)
    savefig(p4s, "Q4_Shear_Diagram.png")
    p4m = plot(x1, M1v, label="0–l", fillrange=0, fillalpha=0.3,
            title="Q4 (b): Bending Moment Diagram", xlabel="x (m)", ylabel="M (kNm)")
    plot!(p4m, x2, M2v, label="l–1.25l", fillrange=0, fillalpha=0.3)
    hline!(p4m, [0], linestyle=:dash, label=false)
    display(p4m)
    savefig(p4m, "Q4_Moment_Diagram.png")
end

q4_class()
```

5. using Plots

```
using SymPy
gr()

function q5_class()
    println("Running Problem 5 : Compound Beam with Hinge")
    x, l, q, RA, RB, RC, Dy = symbols("x l q RA RB RC Dy")
    # Moment at D = 0 → RA*(0.8l) - q*(0.8l)*(0.8l/2) = 0
    eq_MD = Eq(RA*(0.8*l) - (q*0.8*l)*(0.8*l/2), 0)
    RA_expr = solve(eq_MD, RA)[1]

    # Vertical equilibrium of left span (A–D)
    # RA - q*(0.8l) + Dy = 0
    eq_AD = Eq(RA_expr - q*(0.8*l) + Dy, 0)
    Dy_expr = solve(eq_AD, Dy)[1]
    # Moment about B = 0 → (-Dy)*(l - 0.8l) - (q*l)*(l/2) + RC*(l) = 0
    eq_MB = Eq(-Dy_expr*(l - 0.8*l) - (q*l)*(l/2) + RC*(l), 0)
    RC_expr = solve(eq_MB, RC)[1]

    # Vertical equilibrium of right span (B–C)
    # -Dy + RB + RC - q*l = 0
    eq_BC = Eq(-Dy_expr + RB + RC_expr - q*l, 0)
    RB_expr = solve(eq_BC, RB)[1]

    println("Symbolic Reactions:")
    println("RA = $RA_expr")
    println("RB = $RB_expr")
```

```
println("RC = $RC_expr")
println("Dy = $Dy_expr")
l_val = 10.0
q_val = 5.0
subs_pairs = Dict(l => l_val, q => q_val)

RA_val = N(subs(RA_expr, subs_pairs))
RB_val = N(subs(RB_expr, subs_pairs))
RC_val = N(subs(RC_expr, subs_pairs))
Dy_val = N(subs(Dy_expr, subs_pairs))

println("Numeric Values:")
println("RA = $RA_val , RB = $RB_val , RC = $RC_val , Dy = $Dy_val")
# A–D: 0 ≤ x ≤ 0.8l
V_AD(xv) = RA_val - q_val*xv
M_AD(xv) = RA_val*xv - q_val*xv^2/2

# D–B: 0.8l ≤ x ≤ l
V_DB(xv) = -Dy_val
M_DB(xv) = -Dy_val*(xv - 0.8*l_val)

# B–C: l ≤ x ≤ 2l
V_BC(xv) = -Dy_val + RB_val - q_val*(xv - l_val)
M_BC(xv) = -Dy_val*(xv - 0.8*l_val) + RB_val*(xv - l_val) - q_val*(xv - l_val)^2/2
x1 = 0:0.01:(0.8*l_val)
x2 = (0.8*l_val):0.01:l_val
x3 = l_val:0.01:(2*l_val)

V1, V2, V3 = V_AD.(x1), V_DB.(x2), V_BC.(x3)
M1, M2, M3 = M_AD.(x1), M_DB.(x2), M_BC.(x3)
p5s = plot(x1, V1, label="AD", fillrange=0, fillalpha=0.3,
```

```
            title="Q5 (a): Shear Force Diagram", xlabel="x (m)", ylabel="V (kN)")
    plot!(p5s, x2, V2, label="DB", fillrange=0, fillalpha=0.3)
    plot!(p5s, x3, V3, label="BC", fillrange=0, fillalpha=0.3)
    hline!(p5s, [0], linestyle=:dash, label=false)
    display(p5s)
    savefig(p5s, "Q5_Shear_Diagram.png")
    p5m = plot(x1, M1, label="AD", fillrange=0, fillalpha=0.3,
            title="Q5 (b): Bending Moment Diagram", xlabel="x (m)", ylabel="M (kNm)")
    plot!(p5m, x2, M2, label="DB", fillrange=0, fillalpha=0.3)
    plot!(p5m, x3, M3, label="BC", fillrange=0, fillalpha=0.3)
    hline!(p5m, [0], linestyle=:dash, label=false)
    display(p5m)
    savefig(p5m, "Q5_Moment_Diagram.png")
end
q5_class()
```