MANE-6960
Vignesh Ramakrishnan
Due: Friday October 23, 2024

Assignment 2

1 Boundary Value Problem and Functional

In this assignment we are looking to solve and use the discrete adjoint form of quasi-1D Euler equations through a converging, diverging nozzle to evaluate the gradient of a nozzle. The quasi-1D Euler equations are given by,

$$R(q,A) \equiv \frac{d}{dx}[F(q,A)] - G(q,A) = 0, \tag{1}$$

where, the flux and the source are,

$$F(q,A) = \begin{bmatrix} \rho u A & \left(\rho u^2 + p\right) A & u(e+p)A \end{bmatrix}^T, \text{ and} \qquad G(q,A) = \begin{bmatrix} 0 & p\frac{dA}{dx} & 0 \end{bmatrix}^T. \tag{2}$$

The unknown state vector is $q = [\rho, \rho u, e]^T$. Pressure is determined using the ideal-gas equation of state: $p(q) = (\gamma - 1)\left(e - \frac{1}{2}\rho u^2\right)$ and e is the energy per unit volume of the fluid. Equation(1) is discretized using a discontinuous spectral element method, where the discrete solutions are stored at the Lobatto-Gauss-Legendre quadrature points. The discrete residual at node i, on element k, takes the form,

$$R_{k,i}(q_h, A_h) = -\sum_{j=1}^{N} Q_{j,i} F_{k,j} + \delta_{i,N}\hat{F}_{k,N} - \delta_{i,1}\hat{F}_{k,1} - G_{k,i} = 0, \tag{3}$$

where $\delta_{i,j}$ is the Kronecker delta, and the flux and source are evaluated as follows:

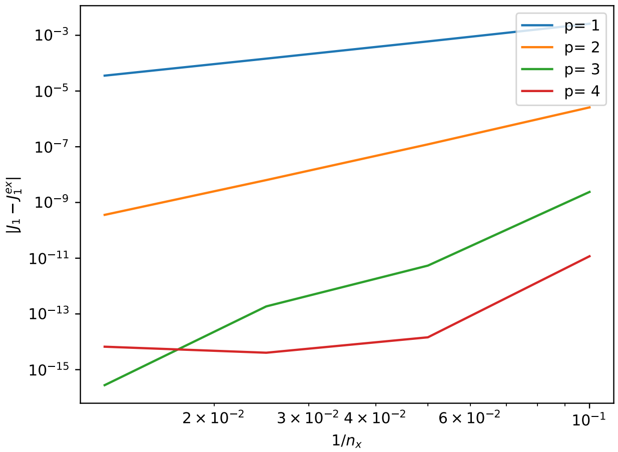
$$F_{k,j} \equiv F(q_{k,j}, A_{k,j}), \text{ and} \qquad G_{k,j} = \begin{bmatrix} 0 \\ p(k,i)\sum_{j=1}^{N} Q_{i,j} A_{k,j} \\ 0 \end{bmatrix}. \tag{4}$$

Roe numerical flux is used to calculate the fluxes $\delta_{i,1}\hat{F}_{i,1}, \delta_{i,N}\hat{F}_{i,N}$ across the element boundaries. $Q_{i,j}$ denotes the $(i,j)^{\text{th}}$ entry in the stiffness matrix $\int_\xi L_i \frac{\partial L_i}{\partial \xi} d\xi$, where L_i is the i^{th} legendre polynomial evaluated on the $[-1,1]$ reference element.
The functionals we are interested in finding out are,

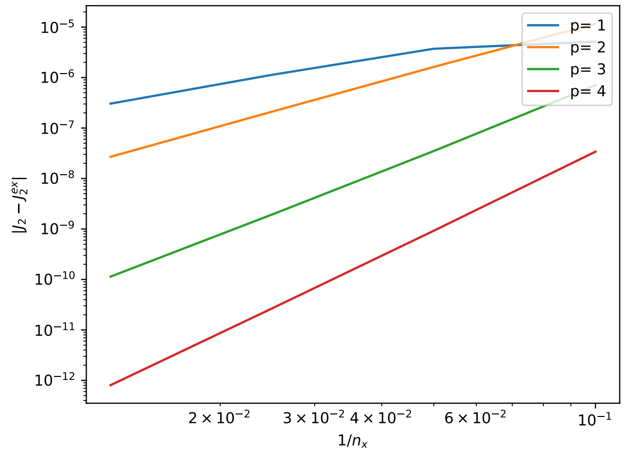
$$J_1(q) = \int_0^1 p\frac{dA}{dx}dx, \text{ and} \qquad J_2(q) = p(q)\Big|_{x=1}. \tag{5}$$

2 Questions

1. Grid convergence study:
   We perform grid convergence studies of the two discrete functionals, $J_{1,h}(q_h)$ and $J_{2,h}(q_h)$. For this study, we have used discretizations of degree $p = 1, 2, 3,$ and 4. The number of elements are increased for every degree of choice from $n_e = 10, 20, 40, 80$. The non-linear residuals are solved to a tolereance of tol $= 1.0e-16$ and we used 50000 iterations of the explicit RK4 time-stepper. Fig 1a and 1b show the results from this convergence study. The order of convergence is found and for each



(a) Mesh convergence study discrete functional $J_{1,h}$



(b) Mesh convergence study discrete functional $J_{2,h}$

Figure 1: Mesh convergence studies plotted in a log log format with respect to both the degree of polynomials and number of elements used.

	$p = 1$	$p = 2$	$p = 3$	$p = 4$
$J_{1,h}$	2.0273318880874727	4.14819617181192	9.392746732420642	
$J_{2,h}$	1.8971128141777056	2.974413346417779	4.090486209474667	5.055653134580847

Table 1: Order of convergence of the computed discrete functionals

functional and show in Table 1. We can notice that the functional $J_{1,h}$ shows super-convergence for polynomials of degree $p > 2$ but the functional $J_{2,h}$ only shows convergence of the order (roughly) $p+1$. This means that functional $J_{2,h}$ is not adjoint consistent, whereas $J_{1,h}$ is. The weird behavior for the convergence of $J_{1,h}$ for a $p = 4$ basis polynomial could be due to reaching ϵ_{mach} in some of the residuals and errors propagating after.

2. Finding and plotting the adjoints of both the functionals $J_{1,h}, J_{2,h}$:

   (a) The Jacobian of the discrete residual R_h with respect to q_h was found using the complex-step method and the code is provided in Listing 1. After the nonlinear residual of the steady-state quasi-1D Euler flow equations are solved, a complex copy of the state vector at all `sbp` nodes at all elements are made. Let $qc_{i,j,k}$ denote the complex copy of the state q_i at `sbp` node j present at element k. This node is perturbed by a complex-step $0. + ih$, where $h = 1E-30$ and the complex residuals are computed at all nodes $Rc_{i,j,k}$, $i = 1, 2, 3,\ j = 1, \dots p+1,\ k = 1, \dots, n_e$. Then the imaginary part of these complex residuals are taken and divided by the complex-step h to find the column of the jacobian pertaining to the state $q_{i,j,k}$, which is $\left.\frac{\partial R_h}{\partial q_h}\right|_{q_{i,j,k}}$. This method follows from the jacobian of a scalar function found using a complex-step method shown below:

$$\frac{\partial f}{\partial u} \approx \frac{\operatorname{Im}\left(f\left(u + ih\right)\right)}{h}. \tag{6}$$

Listing 1: complex-step discrete residual jacobian w.r.t states

```
"""
    calcResidualJacobian!(solver, area, q, dRdqh)
computes the residual jacobian of the non-linear set of equations
at a specified value of q at all nodes
"""
function calcResidualJacobian!(solver::EulerSolver{T},
                area::AbstractMatrix{Tarea},
                q::AbstractArray{Tsol,3},
                dRdqh::AbstractArray{Tres, 2}) where {T, Tarea, Tsol, Tres}

    q_size = size(q, 1) * size(q, 2) * size(q, 3)
    @assert( size(dRdqh, 1) == size(dRdqh, 2) == q_size )

    dRdqh .= 0.0

    # create a complex copy of qh
    q_cmplx = Array{ComplexF64}(undef, size(q, 1), size(q, 2), size(q, 3))
    q_cmplx[:, :, :] = q

    # create a complex version of residual arrays
    r_cmplx = Array{ComplexF64}(undef, size(q, 1), size(q, 2), size(q, 3))
    r_cmplx .= 0.0

    # complex perturbation
    h = 1e-30
    state = 1

    for k = 1:size(q, 3)                        # loop through elements
        for j = 1:size(q, 2)                    # loop through sbp nodes
            for i = 1:size(q, 1)                # go through the state vector
                # perturb state using complex step
                q_cmplx[i, j, k] += complex(0.0, h)

                # find complex step derivative
                calcWeakResidual!(solver, area, q_cmplx, r_cmplx)
                dRdqh[:, state] = imag( vcat(r_cmplx)[:] ) ./ h

                # un-perturb the state
                q_cmplx[i, j, k] -= complex(0.0, h)

                # increment state
                state += 1
            end
        end
    end

end
```

This brute-force technique scales with the number of state-vectors but helps avoid the round-off errors from finite-differencing approaches. We get a matrix of shape $\left.\frac{\partial R_h}{\partial q_h}\right|_{[N,N]}$, where $N = 3 \times n_{\text{sbp}} \times n_e$ being the total number of nodes.

(b) The Jacobian of the functionals $J_{m,h},\ m = 1, 2$ are also found using complex-step method like described before and shown in Listing 2. We get an array of partials of functional with respect to the state $q_{i,j,k}$, which is $\left.\frac{\partial J_{m,h}}{\partial q_h}\right|_{q_{i,j,k}}$ of size $[1, N]$.

Listing 2: complex-step discrete functional jacobian w.r.t states

```
"""
    calcFunctionalJacobian!(solver, area, q, jopt, dJdqh)
This function uses complex-step method to find how the functionals vary
wrt flow state variables.
"""
function calcFunctionalJacobian!(solver::EulerSolver{T},
            area::AbstractMatrix{Tarea},
            q::AbstractArray{Tsol, 3}, jopt::Int64,
            dJdqh::AbstractArray{Tsol, 1}) where {T, Tarea, Tsol}
    q_size = size(q, 1) * size(q, 2) * size(q, 3)
    @assert( size(dJdqh, 1) == q_size )

    dJdqh .= 0.0

    # create a complex copy of qh
    q_cmplx = Array{ComplexF64}(undef, size(q, 1), size(q, 2), size(q, 3))
    q_cmplx[:, :, :] = q

    h = 1.0e-30
    state = 1

    for k = 1:size(q, 3)          # loop through elements
        for j = 1:size(q, 2)      # loop through sbp nodes
            for i = 1:size(q, 1)  # loop through states
                q_cmplx[i, j, k] += complex(0.0, h)

                if jopt == 1
                    dJdqh[state] = \
                        imag( calcIntegratedSource(solver, area, q_cmplx) )/h
                else
                    dJdqh[state] = \
                        imag( calcOutletPressure(solver, area, q_cmplx) )/h
                end

                state += 1
                q_cmplx[i, j, k] -= complex(0.0, h)
            end
        end
    end

end
```

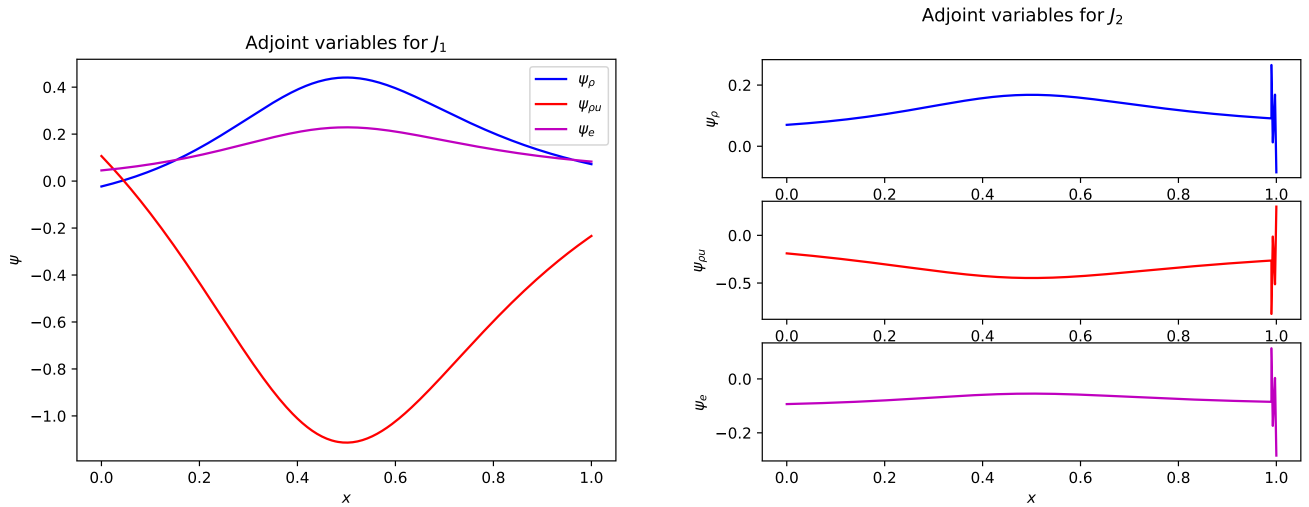
(c) Discrete Lagrangian of the problem is given by,

$$L_{m,h}\left(q_h, A\right) = J_{m,h}\left(q_h, A\right) + \psi^T R_h\left(q_h, A\right), \; m = 1, 2. \tag{7}$$

By setting $\frac{\partial L_{m,h}}{\partial q_h}$ to 0, we can find the adjoint variables solving the linear equation,

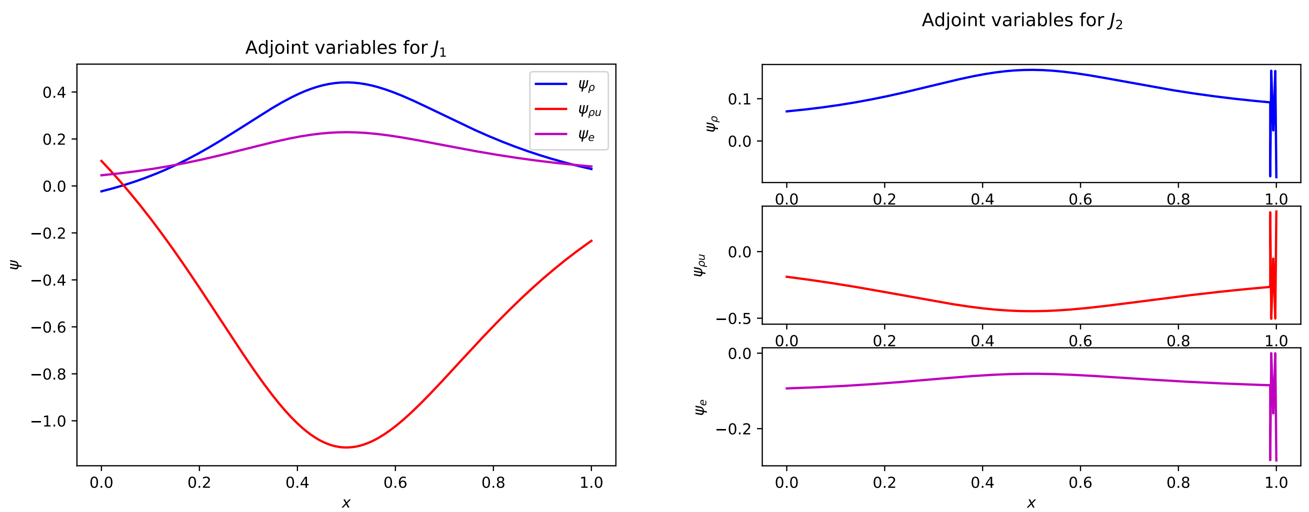
$$\left(\frac{\partial R_h}{\partial q_h}\right)^T \psi = -\left(\frac{\partial J_{m,h}}{\partial q_h}\right)^T. \tag{8}$$

(d) The adjoint variables for both the functionals $J_{1,h}, J_{2,h}$ are plotted in Fig 2 and 3 for discretizations $p = 3, n_e = 100$ and $p = 4, n_e = 80$ respectively. Fig 2a and 3a shows that the functional



(a) Adjoint variables for functional $J_{1,h}$ along domain x (b) Adjoint variables for functional $J_{1,h}$ along domain x

Figure 2: Adjoint variables for both functionals $J_{m,h}$ when the discrete residuals are solved using $p = 3$ and $n_e = 100$.



(a) Adjoint variables for functional $J_{1,h}$ along domain x (b) Adjoint variables for functional $J_{1,h}$ along domain x

Figure 3: Adjoint variables for both functionals $J_{m,h}$ when the discrete residuals are solved using $p = 4$ and $n_e = 80$.

$J_1(q, A)$ which takes the form of $\int_\Omega g'[q] v d\Omega$ is adjoint consistent.
Fig 2b and 3b, are more proof along with the order of convergence study that functional $J_{2,h}$ is not adjoint consistent. Since $J_2\,(q, A)$ does not take the form of neither $\int_\Omega g'[q] v d\Omega$ nor, $\int_\Gamma c'[Cq] C'[q] v d\Gamma$, that functional is not adjoint consistent and all numerical evidences attribute to this.

3. Finding gradient of the integrated-source functional with respect to the area, $DJ_{1,h}/DA_h$
From the Lagrangian defined in Equation (7), we can derive the total derivatives,

$$\frac{DL_{1,h}}{DA_h} = \frac{\partial J_{1,h}}{\partial A_h} + \psi^T \frac{\partial R_h}{\partial A_h} = \frac{DJ_{1,h}}{DA_h}. \tag{9}$$

The un-sorted `area` array is set-up with the of shape $[n_{\text{sbp}} \times n_e]$ in the code. When this array is sorted to be organized along the spatial direction x, it still takes the same shape,

$$A^{\text{sorted}} = \begin{bmatrix} a^{\text{s}}_{1,1} & a^{\text{s}}_{1,2} & \cdots & a^{\text{s}}_{1,n_e} \\ \vdots & \vdots & \ddots & \vdots \\ a^{\text{s}}_{n_{\text{sbp}},1} & \cdots & \cdots & a^{\text{s}}_{n_{\text{sbp}},n_e} \end{bmatrix}. \tag{10}$$

It is important to note that $a^{\text{s}}_{n_{\text{sbp}},k}$ (where k denotes the element) is the same as area $a^{\text{s}}_{1,k+1}$, since this is present at the element interface boundaries. Hence, when complex-step method is used to find the jacobians $\left.\frac{\partial R_h}{\partial A_h}\right|_{[N,N_a]}$ (where $N_a = n_{\text{sbp}} \times n_e$), and $\left.\frac{\partial J_{1,h}}{\partial A_h}\right|_{[1,N_a]}$, both sorted areas, $a^{\text{s}}_{1,k+1}$ and $a^{\text{s}}_{n_{\text{sbp}},k}$ should be perturbed at the same time. This adjustment can be found in lines 24-61 of Listing 3 shown below to calculate the jacobian of the discrete residuals with respect to discrete area.

Listing 3: complex-step discrete residual jacobian w.r.t discrete area

```
1  """
2      calcResidualGradient!(solver, area, q, dRdA)
3  This function computes the partial of flow-residual wrt to discrete
4  area along the nozzle nodes
5  """
6  function calcResidualGradient!(solver::EulerSolver{T},
7                  area::AbstractMatrix{Tarea},
8                  q::AbstractArray{Tsol,3},
9                  dRdA::AbstractArray{Tres, 2}) where {T, Tarea, Tsol, Tres}
10     q_size = size(q, 1) * size(q, 2) * size(q, 3)
11     a_size = size(area, 1) * size(area, 2)
12 
13     @assert( size(dRdA, 1) == q_size )
14     @assert( size(dRdA, 2) == a_size )
15 
16     dRdA .= 0.0
17     state = 1
18 
19 
20     # create a complex copy of area
21     a_cmplx = Array{ComplexF64}(undef, size(area, 1), size(area, 2))
22     a_cmplx[:, :] = area
23 
24     # get the sorted indices
25     idx = sortperm(vec(solver.x[1, :, 1]))
26 
27     # create a complex copy of residual
28     r_cmplx = Array{ComplexF64}(undef, size(q, 1), size(q, 2), size(q, 3))
29     r_cmplx .= 0.0
30 
31     h = 1.0e-40
32 
33     for k = 1:size(q, 3)      # loop over elements
34         for j = 1:size(q, 2) # loop over sbp nodes
```

```
35             # finding the sorted index to check for boundary nodes
36             # once found, the adjacent boundary nodes are also perturbed
37             # because the area on the boundaries are continuous
38             # As_{N, k} = As_{1, k+1}
39             if k==1
40                 if j==idx[size(q, 2)]   # checking for As_{N, k}
41                     a_cmplx[1, k+1] += complex(0.0, h)
42                 else
43                     a_cmplx[j, k] += complex(0.0, h)
44                 end
45             elseif k==size(q, 3)
46                 if j==1                # checking for As_{1, k}
47                     a_cmplx[idx[size(q, 2)], k-1] += complex(0.0, h)
48                 else
49                     a_cmplx[j, k] += complex(0.0, h)
50                 end
51             else
52                 if j==1
53                     a_cmplx[idx[size(q, 2)], k-1] += complex(0.0, h)
54                 elseif j==idx[size(q, 2)]
55                     a_cmplx[1, k+1] += complex(0.0, h)
56                 else
57                     a_cmplx[j, k] += complex(0.0, h)
58                 end
59             end
60             # perturbing it twice to add the contributions
61             a_cmplx[j, k] += complex(0.0, h)
62
63             # find complex step derivative
64             calcWeakResidual!(solver, a_cmplx, q, r_cmplx)
65             dRdA[:, state] = 0.5*imag( vcat(r_cmplx)[:] ) ./ h
66
67             a_cmplx[:, :] .= area
68
69             state += 1
70         end
71     end
72
73 end
```

Since in the code, the perturbation is done twice, the derivative for a scalar function analogy is calculated as,

$$\frac{\partial f}{\partial u} \approx \frac{\text{Im}\left(f\left(u + 2ih\right)\right)}{2h}. \tag{11}$$

This can be seen in line 65 of Listing 3. The same treatment of complex perturbation of areas at the element interface boundaries is performed to calculate the functional jacobian $\frac{\partial J_{1,h}}{\partial A_h}$. Fig 4 shows the graident $DJ_{1,h}/DA_h$ plotted along x. From the continuous residual in Equation (1), we can note that,

$$\int_{x=0}^{x=1} \frac{d}{dx}\left[F\left(q, A\right)\right] dx = \int_{x=0}^{x=1} G\left(q, A\right) dx. \tag{12}$$

If we isolate the second non-linear equation separately, we can observe that,

$$\int_{x=0}^{x=1} \frac{d}{dx}\left(\left(\rho u^2 + p\right) A\right) dx = \int_{x=0}^{x=1} p \frac{dA}{dx} dx = J_1\left(q, A\right). \tag{13}$$

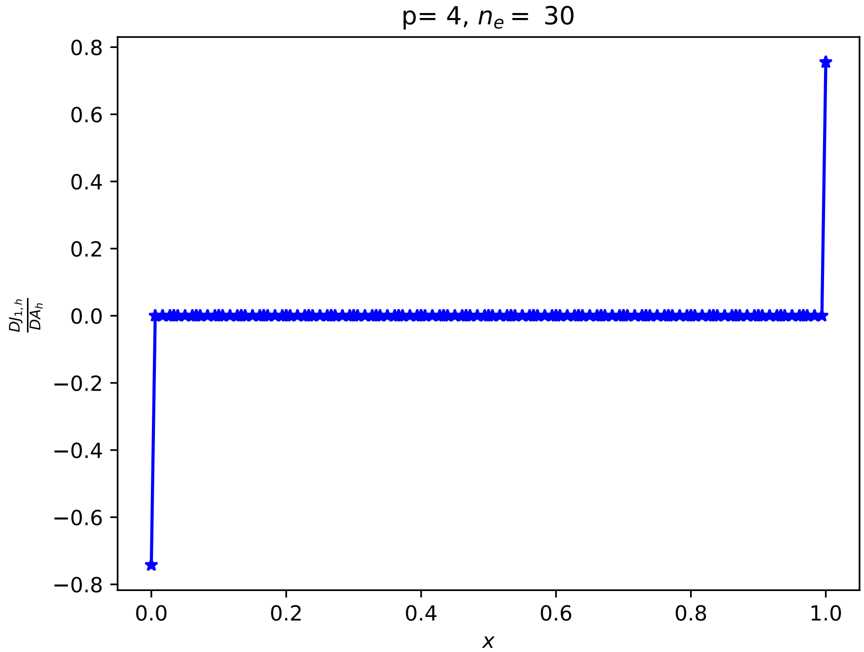


Figure 4: Gradient $DJ_{1,h}/DA_h$ plotted for adjoint consistent discretization using $p = 4, n_e = 30$.

Therefore,

$$J_1(q, A) = \int_0^1 p \frac{dA}{dx} dx = \left[\left(\rho u^2 + p\right) A\right]_{x=0}^{x=1} \tag{14}$$

$$= \left(\left(\rho u^2 + p\right) A\right)\Big|_{x=1} - \left(\left(\rho u^2 + p\right) A\right)\Big|_{x=0}. \tag{15}$$

This means that the functional $J_{1,h}$ is sensitive only to the states at the inlet $(x = 0)$ and outlet $(x = 1)$ of this converging-diverging nozzle. This behavior is also seen in Fig 4 where the gradient is 0 at all node locations except at the inlet and the outlet.

3 Acknowledgements

I would like to thank Habeeb Idris for his timely help with coding in Julia. I also extend my thanks to Zachary Knowlan for working together on the analysis part of the assignment in Question 3.