MANE-6760
Vignesh Ramakrishnan
RIN: 662028006
Due: 11pm September 27, 2022

MANE 6760 (FEM for Fluid Dyn.) Fall 2022: HW1

1. Set $a_x = 1.0$, $\kappa = 0.1$, $\phi_L(x = L) = 0$ and $s = 1.0$.Keepalltheother settings the same. Provide the plot of the FE solution along with the updated Python code.

   In this question, we need to address the right hand side of the equation where we generate the term $\int_{\Omega^e} \bar{w} s \; d\Omega$. In this case, we have $s = 1.0$, hence we are left with numerically integrating the shape function in Ω^e.

$$\int_{\Omega^e} \bar{w} \; d\Omega = \sum_{q=1}^{nq} N_a \left(x(\xi_q)\right) w_q |\det(Jac)|$$

   This will generate a 2×1 b^e vector. Assembly of this is also important since, some elements share degrees of freedom amongst each other. So, the contributions of each element b^e vectors should be added up to the correct row's that corresponds to its equation. This is shown in the Listing 1.

```
    # defining source term
    def source():
        # finding source values at each element
        return 1.0

    def get_right_bdry_value():
        # return right bdry. value (Dirichlet BC)
        return 0.0

    for q in range(neq): # loop index in [0,neq-1]
            wdetj = weq[q]*detj
            s  = source()
            for idx_a in range(nes): # loop index in [0,nes-1]
                be[idx_a] = be[idx_a] + wdetj*shp[idx_a,q]*s // contribution
    to the right index
                for idx_b in range(nes): # loop index in [0,nes-1]
                    Ae[idx_a,idx_b] = Ae[idx_a,idx_b] \
                                      + (shpdgbl[idx_a,q])*ax*shp[idx_b]*wdetj
     \
                                      - (shpdgbl[idx_a,q])*kappa*(shpdgbl[
    idx_b,q])*wdetj

```

Listing 1: code to set up b vector

   The second part of the change is in setting the right boundary condition to $\phi_L(x = L) = 0$. This can be easily modified as shown in Listing 1. The final solution is shown in Fig 1.

2. Set $a_x = 1.0$, $\kappa = 0.1(1.0 + x)$, $\phi_L(x == L) = 1.0$ and $s = 0.0$. Keep all the other settings the same. Provide the plot of the FE solution along with the updated Python code.

   In this problem, κ is calculated at each of the integration points and this can be done by setting up the following bit of code shown in Listing 2.

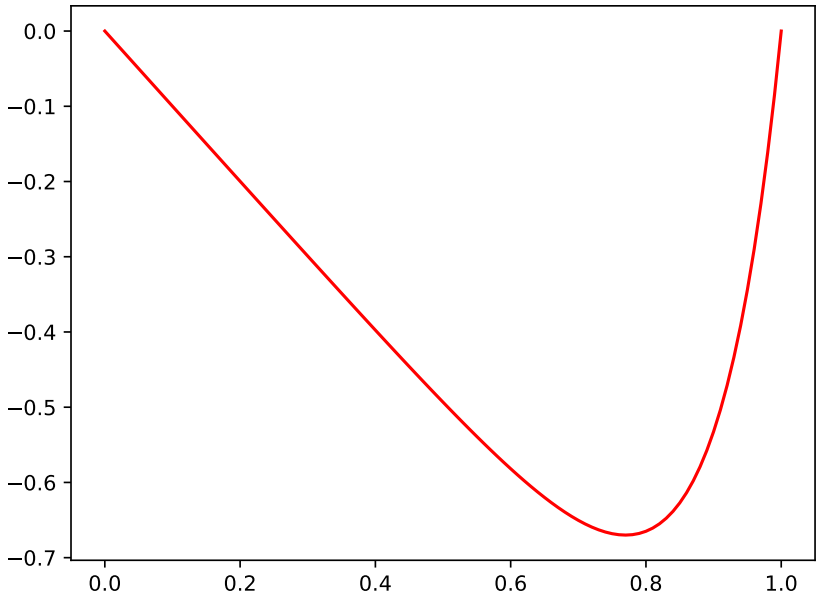

Figure 1: $\phi(x)$ vs x

```
    def get_kappa(x):
        # return kappa value
        kappa = 0.1*(1.0+x)
        # kappa = 1e-1
        assert(kappa>=0)
        return kappa

    def get_right_bdry_value():
        # return right bdry. value (Dirichlet BC)
        return 1.0

    # loop over mesh cells
    for e in range(Ne): # loop index in [0,Ne-1]
        # local/element-level data (matrix and vector)
        assert(nes==nen) # linear elements
        Ae = np.zeros([nen,nen])
        be = np.zeros(nen)

        xl = xmin + e*h
        xr = xmin + (e+1.0)*h

        jac = h/2.0 # 1D and linear elements with uniform spacing
        jacinv = 1/jac # 1D and linear elements
        detj = jac # 1D

        shpdgbl = jacinv*shpdlcl

        for q in range(neq): # loop index in [0,neq-1]
            wdetj = weq[q]*detj
            xq = (xl+xr)/2.0 + detj*xieq[q]
```

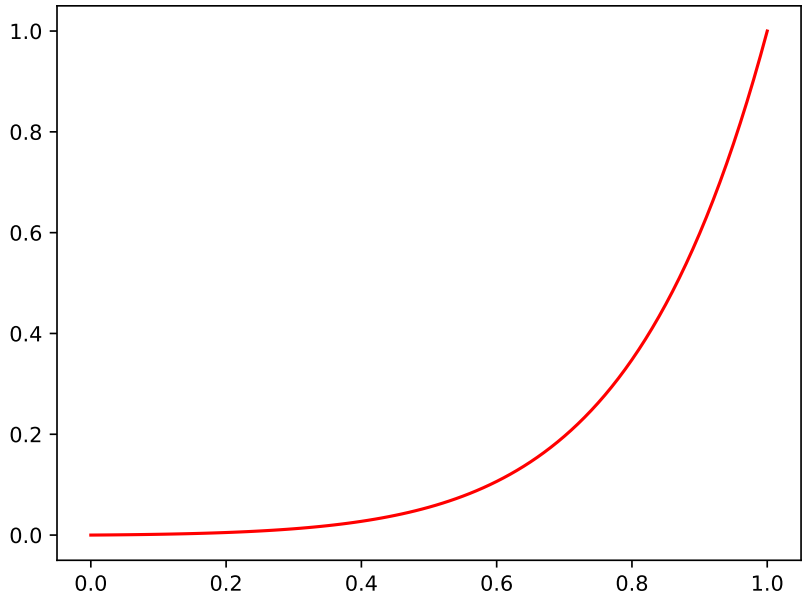

Figure 2: $\phi(x)$ vs x

```
            kappa = get_kappa(xq)
            for idx_a in range(nes): # loop index in [0,nes-1]
                be[idx_a] = 0 # no source term
                for idx_b in range(nes): # loop index in [0,nes-1]
                    Ae[idx_a,idx_b] = Ae[idx_a,idx_b] \
                                      + (shpdgbl[idx_a,q])*ax*shp[idx_b]*wdetj
    \
                                      - (shpdgbl[idx_a,q])*kappa*(shpdgbl[
    idx_b,q])*wdetj

```

Listing 2: code to set up linearly changing κ over the element

Setting the boundary condition back to 1.0, we get the following plot on Fig 2 for this Advection-Diffusion problem.

3. Set $a_x = (1.0 + x)$, $\kappa = 0.1$, $\phi_L(x = L) = 1.0$, and $s = 1.0$. Make sure to determine and use/encode the appropriate numerical integration rule/scheme (i.e., weights and points). Keep all the other settings the same. Provide the plot of the FE solution along with the updated Python code.

   Instead of calculating κ, we now calculate a_x the same way and also set up the source term like we set it up in Q1. The code is attached in Listing 3.

```
    def get_ax(x):
        # return advection velocity value
        ax = (1.0+x)
        return ax

```

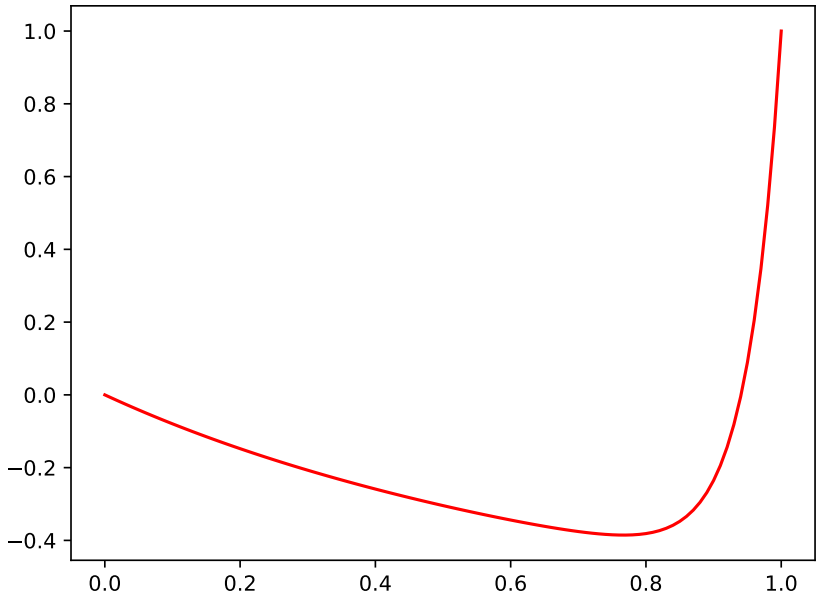

Figure 3: $\phi(x)$ vs x

```
    # defining source term
    def source():
        # finding source values at each element
        return 1.0

    def get_right_bdry_value():
        # return right bdry. value (Dirichlet BC)
        return 1.0

    # loop over mesh cells
    for e in range(Ne): # loop index in [0,Ne-1]
        # local/element-level data (matrix and vector)
        assert(nes==nen) # linear elements
        Ae = np.zeros([nen,nen])
        be = np.zeros(nen)

        xl = xmin + e*h
        xr = xmin + (e+1.0)*h

        jac = h/2.0 # 1D and linear elements with uniform spacing
        jacinv = 1/jac # 1D and linear elements
        detj = jac # 1D

        shpdgbl = jacinv*shpdlcl

        for q in range(neq): # loop index in [0,neq-1]
            wdetj = weq[q]*detj
            xq = (xl+xr)/2.0 + detj*xieq[q]
            ax = get_ax(xq)
            s  = source()
```

```
            for idx_a in range(nes): # loop index in [0,nes-1]
                be[idx_a] = be[idx_a] + wdetj*shp[idx_a,q]*s
                for idx_b in range(nes): # loop index in [0,nes-1]
                    Ae[idx_a,idx_b] = Ae[idx_a,idx_b] \
                                      + (shpdgbl[idx_a,q])*ax*shp[idx_b]*wdetj
    \
                                      - (shpdgbl[idx_a,q])*kappa*(shpdgbl[
   idx_b,q])*wdetj

```

Listing 3: code to set up linearly changing a_x and set up b vector

The final solution looks like whats shown in Fig 3.