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

MANE 6760 (FEM for Fluid Dyn.) Fall 2022: HW2

1. (5 points) Implement and use the following two stabilization parameters: $\tau_{alg,skb} = \tau_{alg1} = \frac{1}{\sqrt{\left(\frac{2|a_x|}{h}\right)^2 + 9\left(\frac{4\kappa}{h^2}\right)^2}}$ and $\tau_{alg,cod} = \tau_{alg2} = \frac{1}{\frac{2|a_x|}{h} + \frac{4\kappa}{h^2}}$. Set $a_x = 1.0$ and $\kappa = 1.0e-4$. Keep all the other settings the same. Provide the plot of the FE solutions (one plot for both settings, or a separate plot for each of the two settings, with clear labels). Also, provide the updated Python code.

   A new function was added to this code called **get_tau_alg()** and it is coded as in Listing 1, and the results are attached in Figure 1.

```
        # choice 0 - shakib, choice 1 - codina
        def get_tau_alg(choice):
            ta_i = (2.0*np.abs(get_ax()))/get_h()
            td_i = (4.0*get_kappa())/(get_h()**2.0)
            if choice==0 :
                tau = 1.0/(np.sqrt(ta_i**2 + 9.0*td_i**2))
            elif choice==1 :
                tau = 1.0/(ta_i + td_i)
        return tau
```

Listing 1: Algorithmic Tau based on choice

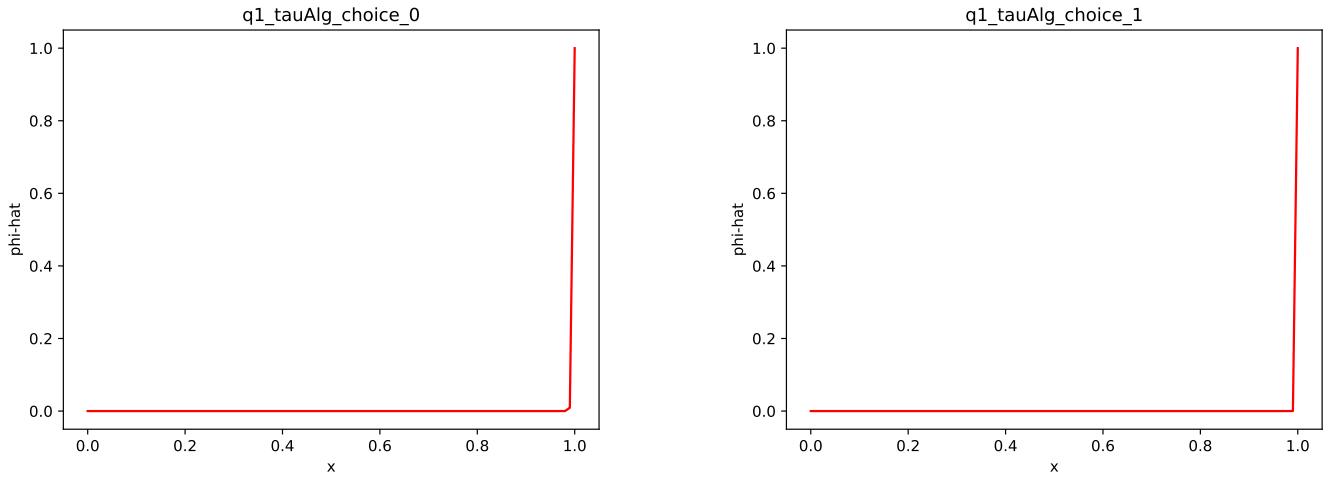

Figure 1: The plot for $\bar{\phi}$ vs x in both cases

2. (15 points) Implement and use the following four stabilization parameters:

   (a) $\tau_{exact1} = \frac{h}{|a_x|}\left(\coth(Pe^e) - \frac{1}{Pe^e}\right)$ with $\coth(Pe^e) = \frac{1+e^{-2Pe^e}}{1-e^{-2Pe^e}}$

   (b) $\tau_{exact1} = \frac{h}{|a_x|}\left(\coth(Pe^e) - \frac{1}{Pe^e}\right)$ with $\coth(Pe^e) = \frac{e^{Pe^e}+e^{-Pe^e}}{e^{Pe^e}+e^{-Pe^e}}$

(c) $\tau_{alg,skb} = \tau_{alg1} = \frac{1}{\sqrt{\left(\frac{2|a_x|}{h}\right)^2 + 9\left(\frac{4\kappa}{h^2}\right)^2}}$

(d) $\tau_{alg,cod} = \tau_{alg2} = \frac{1}{\frac{2|a_x|}{h} + \frac{4\kappa}{h^2}}$

Consider the following values of a_x and κ:

(a) Case A: $a_x = 1.0e-8$ and $\kappa = 1.0$ (note: $Pe^G = 1.0e-8$)

(b) Case B: $a_x = 1.0$ and $\kappa = 1.0+8$ (note: $Pe^G = 1.0e-8$)

(c) Case C: $a_x = 1.0e-4$ and $\kappa = 1.0$ (note: $Pe^G = 1.0e-4$)

(d) Case D: $a_x = 1.0e-4$ and $\kappa = 1.0e-8$ (note: $Pe^G = 1.0e+4$)

(e) Case E: $a_x = 1.0$ and $\kappa = 1.0e-8$ (note: $Pe^G = 1.0e+8$)

For 20 combinations of 4 stabilization parameters and 5 cases, determine whether: i) the stabilization parameter is computed properly, and ii) the stabilized FE calculation yields useful a result. For example, no overflow or significant precision/rounding issues. **Specifically check for both, the stabilization parameter and the stabilized FE solution**. Provide the updated Python code and summarize the result for each combination in the following tabular form. Each cell will contain one of the four possibilities: (yes-τ, yes-$\bar{\phi}$), (yes-τ, no-$\bar{\phi}$), (no-τ, yes-$\bar{\phi}$) and (no-τ, no-$\bar{\phi}$). yes-τ implies the stabilization parameter is computed properly while no-τ implies otherwise. Similarly yes-$\bar{\phi}$ implies a useful a useful stabilized FE solution is obtained and vice-versa.

The code has been set up to run for all these 20 cases and the corresponding modifications can be found in Listing 2.

```
        #case 0 - case A, case 1 - case B
        #case 2 - case C, case 3 - case D, case 4 - case E
        def get_ax(case):
            # return advection velocity value
            if case==0:
                ax = 1.0e-8
            elif (case==1 or case==4):
                ax = 1.0e0
            elif (case==2 or case==3):
                ax = 1.0e-4
            assert(np.abs(ax)>0)
            return ax

        def get_kappa(case):
            # return kappa value
            if (case==0 or case==2):
                kappa = 1.0e0
            elif case==1:
                kappa = 1.0e8
            elif (case==3 or case==4):
                kappa = 1.0e-8
            assert(kappa>0)
            return kappa

        # choice 0 - exact1, choice 1 - exact2
        # choice 2 - shakib, choice 3 - codina
        def get_tau(choice,case):
            # return tau value
```

```
            Pee = 0.5*(np.abs(get_ax(case))*get_h())/get_kappa(case)
            ta_i = (2.0*np.abs(get_ax(case)))/get_h()
            td_i = (4.0*get_kappa(case))/(get_h()**2.0)
            if choice==0:
                em2Pee = np.exp(-2.0*Pee)
                cothPee = (1+em2Pee)/(1-em2Pee)
                tau = 0.5*(get_h()/np.abs(get_ax(case)))*(cothPee-1.0/Pee)
            elif choice==1:
                cothPee = (np.exp(Pee)+np.exp(-1.0*Pee))/(np.exp(Pee)-np.exp
    (-1.0*Pee))
                tau = 0.5*(get_h()/np.abs(get_ax(case)))*(cothPee-1.0/Pee)
            elif choice==2:
                tau = 1.0/(np.sqrt(ta_i**2 + 9.0*td_i**2))
            elif choice==3:
                tau = 1.0/(ta_i + td_i)
            print("tau_choice_"+str(choice)+" = ", tau)
            return tau

        def apply_num_scheme(case,choice):
            :
            :
            ax = get_ax(case)
            kappa = get_kappa(case)
            :
            :
            PeG = abs(ax)*(xmax-xmin)/kappa
            Pee = abs(ax)*h/(2*kappa)
            print("case = %d, choice = %d" % (case,choice))
            print("PeG = ", PeG)
            print("Pee = ", Pee)
            print("-------------")
            tau = get_tau(choice,case)

            if(np.isnan(tau)==False):
                if(tau >= 1e-16):
                    print("yes-tau")
                else:
                    print("no-tau")
            else:
                print("no-tau")
            :
            :
            if (np.isnan(tau)==False):
                phi_sfem = solve_banded((1,1),Abanded,b)
                print("yes-phi")
                print("--------------------")
            else :
                print("no-phi")
                print("--------------------")

        # loop over different cases and choices
        for i in range(5):
            for j in range(4):
                apply_num_scheme(i,j)

```

Listing 2: code set up to run 20 cases automatically

If calculation of $\tau \neq$ NaN && $\tau > 1.0e-16$ then, τ has been computed properly. If computed

$\tau \neq$ NaN && $\tau < 0$, then τ is wrongly calculated due to some precision and rounding issues but we can still try to see if $\bar{\phi}$ can be computed. If $\tau ==$ NaN, then all of our solution values will be $\bar{\phi} =$ NaN. Using this logic, the code above was written. The final solution table is as follows:

	τ_{exact1}	τ_{exact2}	τ_{alg1}	τ_{alg2}
Case A	no-τ, yes-$\bar{\phi}$	no-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$
Case B	no-τ, yes-$\bar{\phi}$	no-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$
Case C	yes-τ, yes-$\bar{\phi}$	no-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$
Case D	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$
Case E	yes-τ, yes-$\bar{\phi}$	no-τ, no-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$	yes-τ, yes-$\bar{\phi}$

3. (10 points) Implement and use the following three stabilization parameters:
$\tau_{exact1} = \frac{h}{|a_x|}\left(\coth(Pe^e) - \frac{1}{Pe^e}\right)$ with $\coth(Pe^e) = \frac{1+e^{-2Pe^e}}{1-e^{-2Pe^e}}$,
$\tau_{alg,skb} = \tau_{alg1} = \frac{1}{\sqrt{\left(\frac{2|a_x|}{h}\right)^2 + 9\left(\frac{4\kappa}{h^2}\right)^2}}$,
and $\tau_{alg,cod} = \tau_{alg2} = \frac{1}{\frac{2|a_x|}{h} + \frac{4\kappa}{h^2}}$.
Set $a_x = 1.0$, $\kappa = 2.0e-3$. Keep all the other settings the same.
Note that the exact solution for this case is given as: $\phi^{exact} = \frac{e^{-Pe^G\left(1-\frac{x}{L}\right)} - e^{-Pe^G}}{1-e^{-Pe^G}}$.
For each of the three stabilization parameters, provide a semi-log plot (log scale only in the vertical axis) for quantity: $\left|\phi^{exact} - \bar{\phi}\right|$, at mesh vertices/nodes (i.e., absolute value of the nodal error, or difference between the exact solution and FE solution evaluated at mesh nodes, in log scale). Three plots in total. Also, provide the updated Python code.

In addition to computing τ_{alg}, an extra bit of code is necessary to compute the exact solution at these nodal locations. This is done through the funciton **get_exact()** and it is shown in Listing 3,

```
        def get_tau():
            # return tau value
            Pee = 0.5*(np.abs(get_ax())*get_h())/get_kappa()
            em2Pee = np.exp(-2.0*Pee)
            cothPee = (1+em2Pee)/(1-em2Pee)
            tau = 0.5*(get_h()/np.abs(get_ax()))*(cothPee-1.0/Pee)
            return tau

        # choice 0 - shakib, choice 1 - codina
        def get_tau_alg(choice):
            ta_i = (2.0*np.abs(get_ax()))/get_h()
            td_i = (4.0*get_kappa())/(get_h()**2.0)
            if choice==0 :
                tau = 1.0/(np.sqrt(ta_i**2 + 9.0*td_i**2))
            elif choice==1 :
                tau = 1.0/(ta_i + td_i)
            return tau

        def get_exact(PeG, x):
            # PeG = abs(ax)*(xmax-xmin)/kappa
            L = get_L()
            emPeG = np.exp(-1*PeG)
```

```
            emPeG_x = np.exp(-1.0*PeG*(1.0-(x/L)))
            phi_exact = (emPeG_x-emPeG)/(1-emPeG)
            return phi_exact

        def apply_num_scheme(choice):
            :
            :
            :
            if choice==0 or choice==1:
                tau = get_tau_alg(choice)
            else:
                tau = get_tau()
            phi_ex = np.zeros(Nn)
            err    = np.zeros(Nn)
            for i in range(Nn):
                x = xmin+i*h
                phi_ex[i] = get_exact(PeG, x)
                print(phi_sfem[i]-phi_ex[i])
                err[i]    = abs(phi_ex[i]-phi_sfem[i])
            s1 = "q3_tauChoice_"+str(choice)
            s2 = s1+".pdf"
            s3 = s1+"_err.pdf"

            if (display_phi_plot):
                plt.plot(xpoints,phi_sfem,'r')
                plt.xlabel('x')
                plt.ylabel('phi-hat')
                plt.title(s1)
                plt.savefig(s2)
                plt.show()

            if (display_phi_plot):
                plt.semilogy(xpoints,err,'b')
                plt.xlabel('x')
                plt.ylabel('err')
                plt.title(s1)
                plt.savefig(s3)
                plt.show()

```

Listing 3: code to compute ϕ^{exact}

Nodal exactness can be observed in the first case where, τ_{exact1} is used and we don't observe the same when τ is calculated through the two numerical algorithmic approximations. This is proved through the **semilogy()** plot of the $||\text{err}||_1$ vs x_{node} shown in Figure 2. In this case, as you can see the set up, the first two solutions are not nodally exact as the error between $|\phi^{exact} - \bar{\phi}| \not< 1.0e-16$. Whereas, the last case, where the exact τ is used, the maximum value of this error is in the order of $1.0e-31 < 1e-16$. Therefore, that solution is nodally exact.

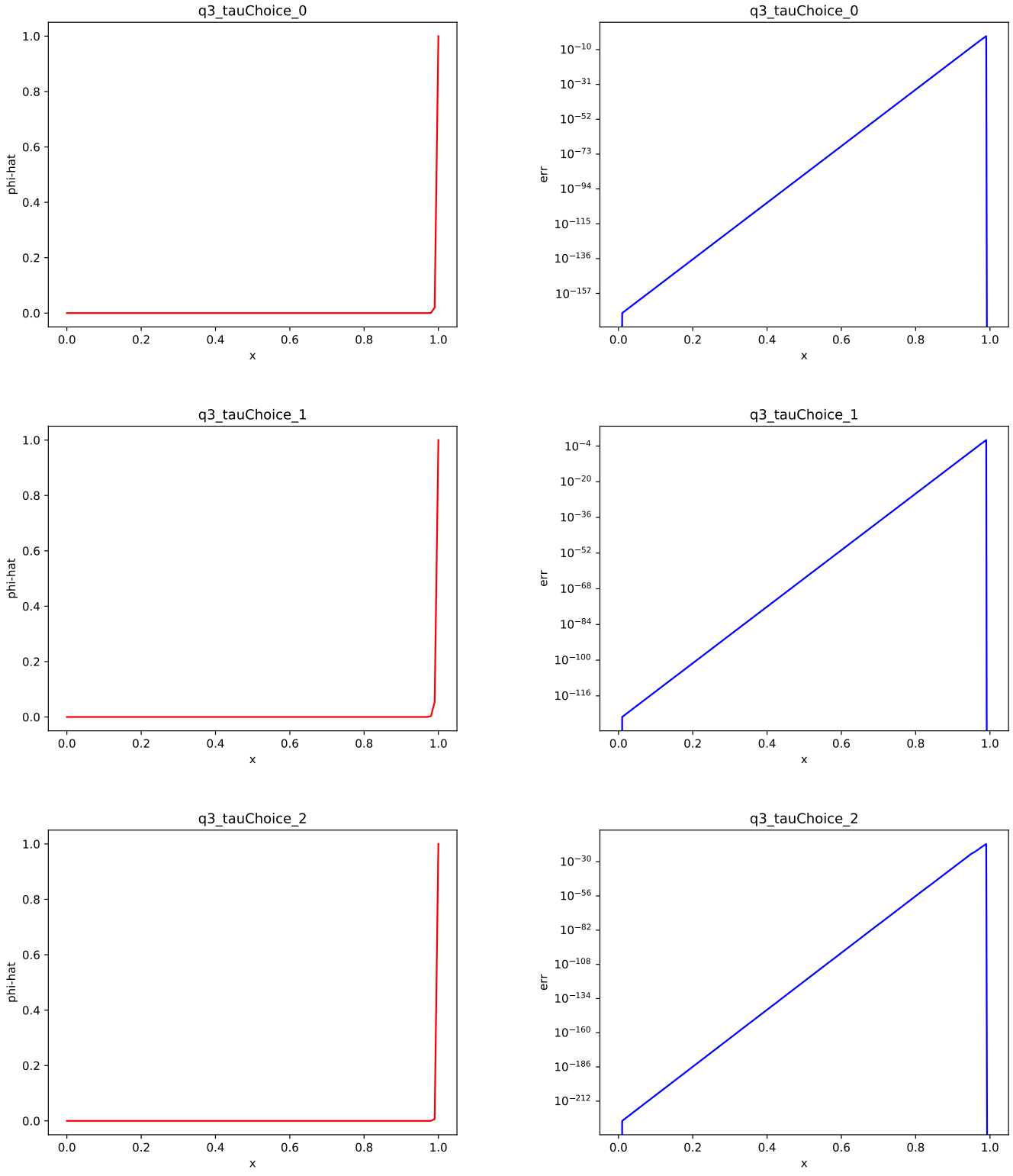

Figure 2: Solution and 1-norm of the error vs nodal location