

AE721 - Boundary Layer Theory

Assignment - 04

Numerical solution of Compressible Couette Flow

Ramkumar S. *

SC22M007, M.Tech. Aerospace - Aerodynamics and Flight Mechanics

This document explains the numerical procedure and outcomes of solving compressible Couette flow where top wall is at rest and the bottom wall is moving with a constant velocity, using shooting method. The computation was performed for two scenarios, one with both walls at equal constant temperature and the other with top wall being adiabatic while bottom wall kept with constant temperature. Non-dimensional form of governing equations were used for the computation and the solution were compared with the given analytical expressions.

I. Introduction

The numerical solution of Compressible Couette Flow is performed in this assignment using nested shooting method. Couette Flow is one of the benchmark problems used in the fluid dynamics for the validation of numerical codes developed. The present problem has analytical solution that has been compared with the numerically obtained solution for the validation of the code developed. The numerical solver code was developed using *Python3* general purpose programming language. The procedure and background theory were obtained from [1].

II. Governing equations

The reduced form of equations are obtained from the full Navier-Stokes equations for the Compressible Couette flow, the reduced momentum equation is given in Equation (1).

$$\frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) = 0 \quad (1)$$

The energy equation is obtained as given below.

$$\frac{\partial}{\partial y} \left(k \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial y} \left(\mu u \frac{\partial u}{\partial y} \right) = 0$$

which can be further simplified by substituting the definition for shear stress, leading to Equation (2).

$$\frac{\partial}{\partial y} \left(k \frac{\partial T}{\partial y} \right) + \tau \frac{\partial u}{\partial y} = 0 \quad (2)$$

Further, the expression for viscosity is taken as a simplified linear dependence model of temperature given in Equation (3).

$$\mu = \mu_0 \left(\frac{T}{T_0} \right) \quad (3)$$

The reference values are taken as $\mu_0 = 1.789e - 5 \text{ Pa.s}$ and $T_0 = 288.16 \text{ K}$.

The thermal conductivity is governed using the Prandtl number definition as given in Equation (4), the gas is taken to be calorically perfect with $C_p = 1005.0 \text{ J/kg.K}$ and $Pr = 0.72$.

$$k = \frac{\mu C_p}{Pr} \quad (4)$$

*SC22M007, M.Tech., Aerospace AFM

Table 1 Boundary conditions for case with equal wall temperatures

parameter	at $y = 0$	at $y = 1$
u	fixed value, $u = 1$	fixed value, $u = 0$
T	fixed value, $T = 1$	fixed value, $T = 1$

Table 2 Boundary conditions for case with adiabatic top wall

parameter	at $y = 0$	at $y = 1$
u	fixed value, $u = 1$	fixed value, $u = 0$
T	fixed value, $T = 1$	fixed gradient, $\frac{\partial T}{\partial y} = 0$

All the parameters were non-dimensionalized with their respective reference values so that the domain in y -direction varies from 0 to 1. The temperatures were normalized using T_0 , viscosity with μ_0 , and the thermal conductivity using k_0 which is given by Equation (5).

$$k_0 = \frac{\mu_0 C_p}{Pr}. \quad (5)$$

After all the non-dimensionalization process, the final governing equations obtained are given in Equations (6) and (7).

$$\frac{\partial}{\partial y'} \left(\mu' \frac{\partial u'}{\partial y'} \right) = 0 \quad (6)$$

$$\frac{\partial}{\partial y'} \left(k' \frac{\partial T'}{\partial y'} \right) + PrEc \times \tau' \frac{\partial u'}{\partial y'} = 0 \quad (7)$$

where, $PrEc = A$ is the product of Prandtl number and Eckert number, the boundary conditions for the scenarios are given in Tables 1 and 2.

III. Numerical procedure

The well known *shooting* method is used for the solution of the equations. Here, there are two unknowns, u, T , hence the equation for temperature is taken to be Equation (7) and the one for velocity is taken from the definition of shear stress given in Equation (8), as the shear stress for this flow problem is a constant, it is obtained from Equation (6).

$$\begin{aligned} \frac{\partial}{\partial y'} \left(\mu' \frac{\partial u'}{\partial y'} \right) &= \frac{\partial \tau'}{\partial y'} = 0 \\ \frac{\tau'}{\mu'} &= \frac{\partial u'}{\partial y'} \end{aligned} \quad (8)$$

Upon nondimensionalization, an expression that relates temperature, viscosity and thermal conductivity in their nondimensional form is given in Equation (9).

$$T' = \mu' = k' \quad (9)$$

The procedure of solving the equations using shooting method, as given in [1] is as follows.

- 1) A value for τ is assumed in Equation (7) using the incompressible definition of shear stress. Also the initial velocity profile is assumed to be linear.
- 2) starting at $y = 0$ with the known boundary condition of wall temperature, Equation (7) is integrated till $y = 1$, using first order Euler method of numerical integration, since the equation is second order, two boundary conditions are to be specified. Temperature is fixed at lower wall, and the gradient of temperature is assumed to be some value.

- 3) After reaching the top wall in integration, the temperature obtained at the wall is checked if it matches with the top boundary condition. If not, then the initial temperature gradient assumption at bottom wall is adjusted and re-integration is performed. Newton-Raphson method is implemented for the so called shooting method.
- 4) From the converged temperature profile, the variation of non-dimensional dynamic viscosity and thermal conductivity can be obtained from Equation (9). Using the updated value of μ' , the Equation (8) is numerically integrated for the new velocity profile, from bottom wall till top wall.
- 5) The velocity value obtained at the top wall is checked to match with the boundary condition, and if it does not match, then the value of non-dimensional shear stress (assumed in first step) is adjusted to match the velocity profile, a second shooting method is implemented for this.
- 6) Then the procedure is repeated till both the velocity and temperature profiles does not vary significantly between the iterations.

IV. Numerical solution results and validation

The analytical expressions for temperature and velocity profiles for the case with constant equal wall temperatures are given in Equations (10) and (11).

$$y' = 1 - u' \left(\frac{1 + 0.25 \times A \times u' - \frac{1}{6} A u'^2}{1 + \frac{1}{12} A} \right) \quad (10)$$

$$T' = 1 + \frac{1}{2} A u' (1 - u') \quad (11)$$

where $A = Pr.Ec..$ Similarly the analytical expressions for temperature and velocity profiles for the case with adiabatic top wall and constant temperature bottom wall are given in Equations (12) and (13).

$$y' = 1 - u' \left(\frac{1 + 0.5 \times A \times \left(1 - \frac{u'}{2}\right)}{1 + \frac{1}{4} A} \right) \quad (12)$$

$$T' = 1 + \frac{1}{2} A \left(1 - u'^2\right) \quad (13)$$

The numerically computed results were compared with the analytical solutions given by Equations (10) to (13). The solution for the case with equal constant wall temperatures is given in Figures 1 and 2.

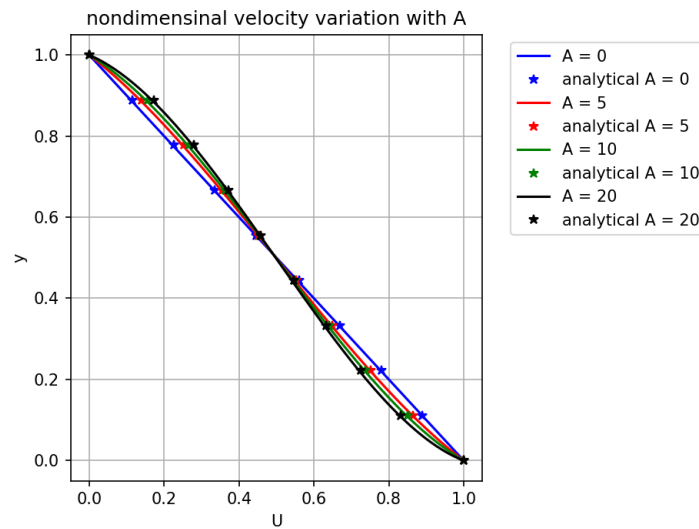


Fig. 1 variation of velocity profile for the different values of A, case with equal wall temperatures

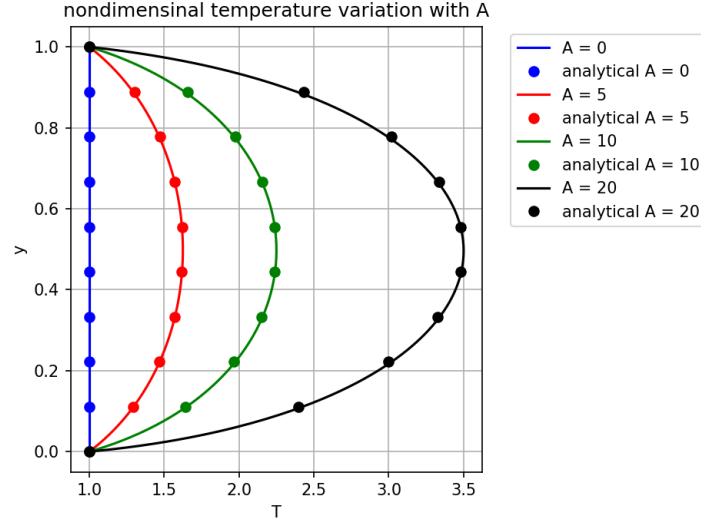


Fig. 2 variation of temperature profile for the different values of A , case with equal wall temperatures

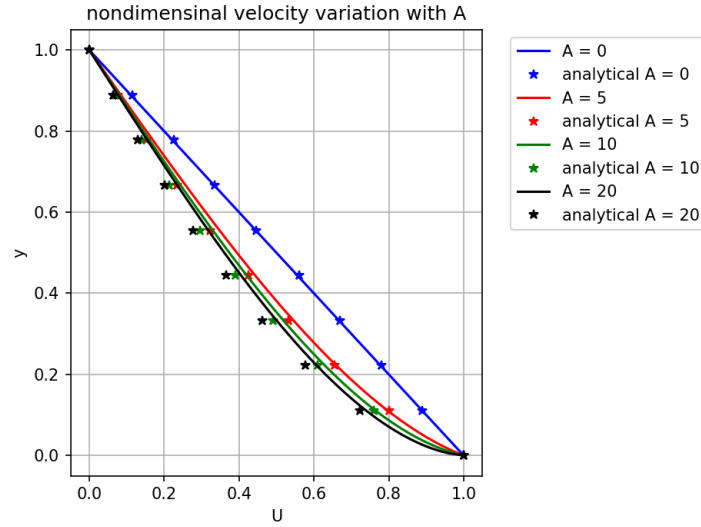


Fig. 3 variation of velocity profile for the different values of A , case with adiabatic top wall

Similarly, the numerically computed results for the case with adiabatic top wall were compared against the analytical solutions and are given in Figures 3 and 4.

From Figures 1 to 4, it can be noted that the numerical solution matches well with the analytical solution, hence the code developed for the solution of these equations is validated.

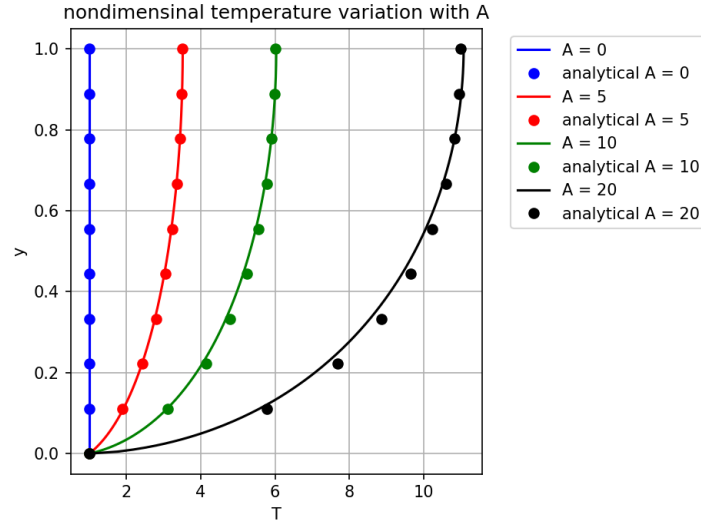


Fig. 4 variation of temperature profile for the different values of A , case with adiabatic top wall

V. Conclusion

The solution of Compressible Couette Flow has been computed numerically using nested shooting method with nondimensional governing equations and the results were compared against the solutions obtained using analytical expressions provided. Two scenarios were taken, one with equal constant wall temperatures and the other with adiabatic top wall and bottom wall being with constant temperature. The code developed for this assignment is validated against the analytical solution and are given in Sections A and B.

References

- [1] Anderson, John. EBOOK: Fundamentals of Aerodynamics (SI units). McGraw hill, 2011.

A. Python code for Equal wall temperature case

This section contains the Python code developed for the case with equal constant wall temperatures.

```
#!/bin/python3
"""
AE721 — Boundary Layer Theory
Assignment — 04: Numerical simulation of Compressible Couette Flow
case 1 : equal and constant wall temperatures

Name : Ramkumar S
SCNO : SC22M007
"""

# importing needed modules
import numpy as np
import pandas as pd
from copy import copy as cp
import matplotlib.pyplot as plt

# computation parameters definition
Pr = 0.72 # prandtl number
mu0 = 1.789e-5 # dynamic viscosity reference
T0 = 288.16 # reference temperature
Cp = 1005.0 # specific heats at constant pressure
k0 = mu0*Cp/Pr # reference thermal conductivity value for air
N = 201 # number of steps in y direction

A = [0,5,10,20] # A = Pr*Ec
T_bot = 288.16/T0 # top and bottom wall temperatures
T_top = 288.16/T0
U_bot = 1.0 # top and bottom wall velocities
U_top = 0.0

# function definitions
# variables definition
T = np.linspace(T_bot, T_top*0.9, N)
U = np.linspace(U_bot, U_top, N)
S = np.ones(N)*(T_top-T_bot)/1.0
y = np.linspace(0,1,N)
mu = np.ones(N)
dy = 1/(N-1)

# test function for T variable
def T_testfunc(tau, PrEc, Sw):
    global mu, S, U, T
    S[0] = Sw
    # begining loop
    for i in range(N-1):
        # computing K, T and mu
        K = mu[i]

        # solving equations
        T[i+1] = T[i] + S[i]/K*dy
        S[i+1] = S[i] - PrEc*tau*(U[i+1]-U[i])/dy*dy

    return T[-1]

# test function for U variable
def U_testfunc(tau):
    global mu, S, U, T
    # integrating
    for i in range(N-1):
        U[i+1] = U[i] + tau/mu[i]*dy

    return U[-1]

# solution computation
```

```

66 # creating lists to store values
   T_list = []
68 U_list = []

70 # looping through A values
   for A_index in range(len(A)):
72
74       print("Solving for A = ",A[A_index])
76
78       tau = 1.0 # initial guess for nondim shearstress
76
78       for itr in range(100):
78           print("iteration : ",itr+1)
80               # solving for T
80               S_a = 0
82               S_b = 1.0
82               S_c = (S_a+S_b)/2
84
84               while abs(T_testfunc(tau,A[A_index],S_b)-T_top) > 1e-7:
86                   # solving for T
86                   S_c = S_b - (T_testfunc(tau,A[A_index],S_b)-T_top)/((T_testfunc(tau,A[A_index],S_b)-
T_testfunc(tau,A[A_index],S_a))/(S_b-S_a))
88                   S_a = S_b*1.0
88                   S_b = S_c*1.0
90
90                   S[0] = S_c*1.0
92                   print("\tsolved for T")
92
94                   mu = T*1.0
94
96                   # solving for U
96                   tau_a = 0
98                   tau_b = 1.0
98                   tau_c = (tau_a + tau_b)/2.0
100
100                   while abs(U_testfunc(tau_c) - U_top) > 1e-7:
102                       # solving for U
102                       tau_c = tau_b - (U_testfunc(tau_b)-U_top)/((U_testfunc(tau_b)-U_testfunc(tau_a))/(
tau_b-tau_a))
104                       tau_a = tau_b*1.0
104                       tau_b = tau_c*1.0
106
106                       tau = tau_c*1.0
108                       print("\tsolved for U")
110
110                       T_list.append(list(T))
110                       U_list.append(list(U))
112
112 # analytical solution computation
114 # preparing lists to store computed values
   T_A_list = []; U_A_list = []
116
116 # defining lambda functions for U and T
118 res_U = lambda y,u,Aval: 1-u*(1+0.25*Aval*u-1/6*Aval*u**2)/(1+1/12*Aval)-y
   Tval = lambda u,Aval: 1 + 0.5*Aval*u*(1-u)
120
120 N_vals = np.linspace(0,N-1,10, dtype=int)
122 Y_A = np.linspace(0,1,10)
124
124 # looping through A values
   for A_index in range(len(A)):
126     # initializing empty list to store T and U values
126     T_A = []; U_A = []
128
128     print("computing analytical solution for A = ", A[A_index])
130
130     # looping through N values

```

```

132     for i in N_vals:
133         # solving for U_A with bisection method
134         U_a = -0.1
135         U_b = 1.1
136         U_c = (U_a+U_b)/2
137         while abs(res_U(y[i],U_c,A[A_index])) > 1e-6:
138             resVal = res_U(y[i],U_c,A[A_index])
139             if resVal > 0:
140                 U_a = U_c*1.0
141             else:
142                 U_b = U_c*1.0
143
144             U_c = (U_a+U_b)/2.0
145         U_A.append(U_c)
146     U_A_list.append(U_A)
147
148     # computing temperature
149     for i in range(10):
150         T_A.append(Tval(U_A[i],A[A_index]))
151     T_A_list.append(T_A)
152
153     # post-processing
154     # writing data to file
155
156     # creating dataframe for T
157     fid = pd.DataFrame(np.transpose(T_list), columns = ["A="+str(val) for val in A])
158     fid['Y'] = y
159     fid.to_csv("T_values.csv", index = None)
160
161     # creating dataframe for T_analytical
162     fid = pd.DataFrame(np.transpose(T_A_list), columns = ["A="+str(val) for val in A])
163     fid['Y'] = Y_A
164     fid.to_csv("T_Analytical_values.csv", index = None)
165
166     # creating dataframe for U
167     fid = pd.DataFrame(np.transpose(U_list), columns = ["A="+str(val) for val in A])
168     fid['Y'] = y
169     fid.to_csv("U_values.csv", index = None)
170
171     # creating dataframe for U_analytical
172     fid = pd.DataFrame(np.transpose(U_A_list), columns = ["A="+str(val) for val in A])
173     fid['Y'] = Y_A
174     fid.to_csv("U_Analytical_values.csv", index = None)
175
176     # plotting velocity graph
177     plt.figure()
178     # for i in range(len(A)):
179     #     plt.plot(U_list[i],y,label='A = '+str(A[i]))
180     #     plt.plot(U_A_list[i],Y_A,'*',label='analytical A = '+str(A[i]))
181
182     plt.plot(U_list[0],y,'-b',label='A = '+str(A[0]))
183     plt.plot(U_A_list[0],Y_A,'*b',label='analytical A = '+str(A[0]))
184     plt.plot(U_list[1],y,'-r',label='A = '+str(A[1]))
185     plt.plot(U_A_list[1],Y_A,'*r',label='analytical A = '+str(A[1]))
186     plt.plot(U_list[2],y,'-g',label='A = '+str(A[2]))
187     plt.plot(U_A_list[2],Y_A,'*g',label='analytical A = '+str(A[2]))
188     plt.plot(U_list[3],y,'-k',label='A = '+str(A[3]))
189     plt.plot(U_A_list[3],Y_A,'*k',label='analytical A = '+str(A[3]))
190
191     plt.grid()
192     plt.xlabel('U')
193     plt.ylabel('y')
194     plt.legend(loc='upper left',bbox_to_anchor=[1.05,1])
195     plt.title("nondimensinal velocity variation with A")
196     plt.tight_layout()
197     plt.savefig("U_profiles.png", dpi = 150)
198
199     # plotting temperature graph

```



```

200 plt.figure()
# for i in range(len(A)):
202 #     plt.plot(T_list[i],y,label='A = '+str(A[i]))
#     plt.plot(T_A_list[i],Y_A,'o',label='analytical A = '+str(A[i]))
204 plt.plot(T_list[0],y,'-b',label='A = '+str(A[0]))
plt.plot(T_A_list[0],Y_A,'ob',label='analytical A = '+str(A[0]))
206 plt.plot(T_list[1],y,'-r',label='A = '+str(A[1]))
plt.plot(T_A_list[1],Y_A,'or',label='analytical A = '+str(A[1]))
208 plt.plot(T_list[2],y,'-g',label='A = '+str(A[2]))
plt.plot(T_A_list[2],Y_A,'og',label='analytical A = '+str(A[2]))
210 plt.plot(T_list[3],y,'-k',label='A = '+str(A[3]))
plt.plot(T_A_list[3],Y_A,'ok',label='analytical A = '+str(A[3]))
212 plt.grid()
plt.xlabel('T')
214 plt.ylabel('y')
plt.legend(loc='best',bbox_to_anchor=[1.05,1])
216 plt.title("nondimensinal temperature variation with A")
plt.tight_layout()
218 plt.savefig("T_profiles.png", dpi = 150)
220
plt.show()

```

B. Python code for adiabatic top wall case

This section contains the Python code developed for the case with adiabatic top wall case.

```

#!/bin/python3
"""
2 AE721 — Boundary Layer Theory
4 Assignment — 04: Numerical simulation of Compressible Couette Flow
case 2 : adiabatic top wall and constant tempeature bottom wall
6
Name : Ramkumar S
8 SCNO : SC22M007
"""
10
# importing needed modules
12 import numpy as np
import pandas as pd
14 from copy import copy as cp
import matplotlib.pyplot as plt
16
# computation parameters definition
18 Pr = 0.72 # prandtl number
mu0 = 1.789e-5 # dynamic viscosity reference
20 T0 = 288.16 # reference temperature
Cp = 1005.0 # specific heats at constant pressure
22 k0 = mu0*Cp/Pr # reference thermal conductivity value for air
N = 201 # number of steps in y direction
24
A = [0,5,10,20] # A = Pr*Ec
26 T_bot = 288.16/T0 # top and bottom wall temperatures
T_top = 288.16/T0
28 U_bot = 1.0 # top and bottom wall velocities
U_top = 0.0
30
# function definitions
32 # variables definition
T = np.linspace(T_bot, T_top*0.9, N)
34 U = np.linspace(U_bot,U_top,N)
S = np.ones(N)*(T_top-T_bot)/1.0
36 y = np.linspace(0,1,N)
mu = np.ones(N)
38 dy = 1/(N-1)
40
# test function for T variable

```

```

def T_testfunc (tau ,PrEc ,Sw):
42     global mu,S,U,T
    S[0] = Sw
44     # begining loop
    for i in range(N-1):
46         # computing K, T and mu
        K = mu[i]

48         # solving equations
50         T[i+1] = T[i] + S[i]/K*dy
        S[i+1] = S[i] - PrEc*tau*(U[i+1]-U[i])/dy*dy

52     return S[-1]

54 # test function for U variable
def U_testfunc (tau):
    global mu,S,U,T
56     # integrating
    for i in range(N-1):
58         U[i+1] = U[i] + tau/mu[i]*dy

60     return U[-1]

62 # solution computation
64 # creating lists to store values
T_list = []
66 U_list = []

70 # looping through A values
for A_index in range(len(A)):
72     print("Solving for A = ",A[A_index])

74     tau = 1.0 # initial guess for nondim shearstress

76     for itr in range(100):
78         print("iteration : ",itr+1)
        # solving for T
80         S_a = 0
        S_b = 1.0
82         S_c = (S_a+S_b)/2

84         while abs(T_testfunc (tau ,A[A_index] ,S_b)-0) > 1e-7:
            # solving for T
86             S_c = S_b - ((T_testfunc (tau ,A[A_index] ,S_b)-0)/
                ((T_testfunc (tau ,A[A_index] ,S_b)-
88                  T_testfunc (tau ,A[A_index] ,S_a))/(S_b-S_a)))
            S_a = S_b*1.0
90             S_b = S_c*1.0

92         S[0] = S_c*1.0
        print("\tsolved for T")

94         mu = T*1.0

96         # solving for U
98         tau_a = 0
        tau_b = 1.0
100        tau_c = (tau_a + tau_b)/2.0

102        while abs(U_testfunc (tau_c) - U_top) > 1e-7:
            # solving for U
104            tau_c = tau_b - ((U_testfunc (tau_b)-U_top)/
                ((U_testfunc (tau_b)-U_testfunc (tau_a))/(tau_b-tau_a)))
106            tau_a = tau_b*1.0
            tau_b = tau_c*1.0

```

```

110     tau = tau_c*1.0
111     print("\tsolved for U")
112
113     T_list.append(list(T))
114     U_list.append(list(U))
115
116 # analytical solution computation
117 # preparing lists to store computed values
118 T_A_list = []; U_A_list = []
119
120 # defining lambda functions for U and T
121 res_U = lambda y,u,Aval: 1-u*(1+0.5*Aval*(1-u/2))/(1+0.25*Aval) - y
122 Tval = lambda u,Aval: 1 + 0.5*Aval*(1-u**2)
123
124 N_vals = np.linspace(0,N-1,10, dtype=int)
125 Y_A = np.linspace(0,1,10)
126
127 # looping through A values
128 for A_index in range(len(A)):
129     # initializing empty list to store T and U values
130     T_A = []; U_A = []
131
132     print("computing analytical solution for A = ", A[A_index])
133
134     # looping through N values
135     for i in N_vals:
136         # solving for U_A with bisection method
137         U_a = -0.1
138         U_b = 1.1
139         U_c = (U_a+U_b)/2
140         while abs(res_U(y[i],U_c,A[A_index])) > 1e-6:
141             resVal = res_U(y[i],U_c,A[A_index])
142             if resVal > 0:
143                 U_a = U_c*1.0
144             else:
145                 U_b = U_c*1.0
146
147             U_c = (U_a+U_b)/2.0
148         U_A.append(U_c)
149         U_A_list.append(U_A)
150
151     # computing temperature
152     for i in range(10):
153         T_A.append(Tval(U_A[i],A[A_index]))
154     T_A_list.append(T_A)
155
156 # post-processing
157 # writing data to file
158
159 # creating dataframe for T
160 fid = pd.DataFrame(np.transpose(T_list), columns = ["A="+str(val) for val in A])
161 fid['Y'] = y
162 fid.to_csv("T_values.csv", index = None)
163
164 # creating dataframe for T_analytical
165 fid = pd.DataFrame(np.transpose(T_A_list), columns = ["A="+str(val) for val in A])
166 fid['Y'] = Y_A
167 fid.to_csv("T_Analytical_values.csv", index = None)
168
169 # creating dataframe for U
170 fid = pd.DataFrame(np.transpose(U_list), columns = ["A="+str(val) for val in A])
171 fid['Y'] = y
172 fid.to_csv("U_values.csv", index = None)
173
174 # creating dataframe for U_analytical
175 fid = pd.DataFrame(np.transpose(U_A_list), columns = ["A="+str(val) for val in A])
176 fid['Y'] = Y_A

```

```

fid.to_csv("U_Analytical_values.csv", index = None)

178

180 # plotting velocity graph
plt.figure()
182 # for i in range(len(A)):
#     plt.plot(U_list[i],y,label='A = '+str(A[i]))
#     plt.plot(U_A_list[i],Y_A,'*',label='analytical A = '+str(A[i]))
184 plt.plot(U_list[0],y,'-b',label='A = '+str(A[0]))
186 plt.plot(U_A_list[0],Y_A,'*b',label='analytical A = '+str(A[0]))
plt.plot(U_list[1],y,'-r',label='A = '+str(A[1]))
188 plt.plot(U_A_list[1],Y_A,'*r',label='analytical A = '+str(A[1]))
plt.plot(U_list[2],y,'-g',label='A = '+str(A[2]))
190 plt.plot(U_A_list[2],Y_A,'*g',label='analytical A = '+str(A[2]))
plt.plot(U_list[3],y,'-k',label='A = '+str(A[3]))
192 plt.plot(U_A_list[3],Y_A,'*k',label='analytical A = '+str(A[3]))

194 plt.grid()
plt.xlabel('U')
196 plt.ylabel('y')
plt.legend(loc='upper left',bbox_to_anchor=[1.05,1])
198 plt.title("nondimensional velocity variation with A")
plt.tight_layout()
200 plt.savefig("U_profiles.png", dpi = 150)

202 # plotting temperature graph
plt.figure()
204 # for i in range(len(A)):
#     plt.plot(T_list[i],y,label='A = '+str(A[i]))
#     plt.plot(T_A_list[i],Y_A,'o',label='analytical A = '+str(A[i]))
206 plt.plot(T_list[0],y,'-b',label='A = '+str(A[0]))
208 plt.plot(T_A_list[0],Y_A,'ob',label='analytical A = '+str(A[0]))
plt.plot(T_list[1],y,'-r',label='A = '+str(A[1]))
210 plt.plot(T_A_list[1],Y_A,'or',label='analytical A = '+str(A[1]))
plt.plot(T_list[2],y,'-g',label='A = '+str(A[2]))
212 plt.plot(T_A_list[2],Y_A,'og',label='analytical A = '+str(A[2]))
plt.plot(T_list[3],y,'-k',label='A = '+str(A[3]))
214 plt.plot(T_A_list[3],Y_A,'ok',label='analytical A = '+str(A[3]))
plt.grid()
216 plt.xlabel('T')
plt.ylabel('y')
218 plt.legend(loc='best',bbox_to_anchor=[1.05,1])
plt.title("nondimensional temperature variation with A")
220 plt.tight_layout()
plt.savefig("T_profiles.png", dpi = 150)
222 plt.show()

```
