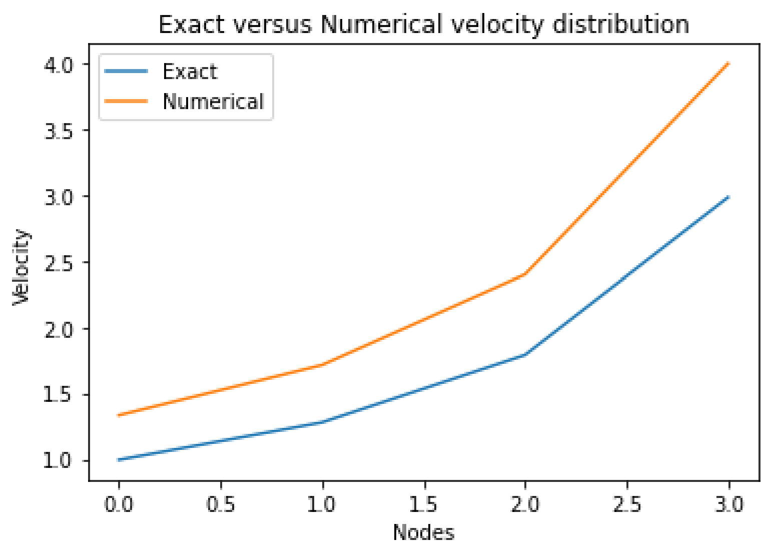
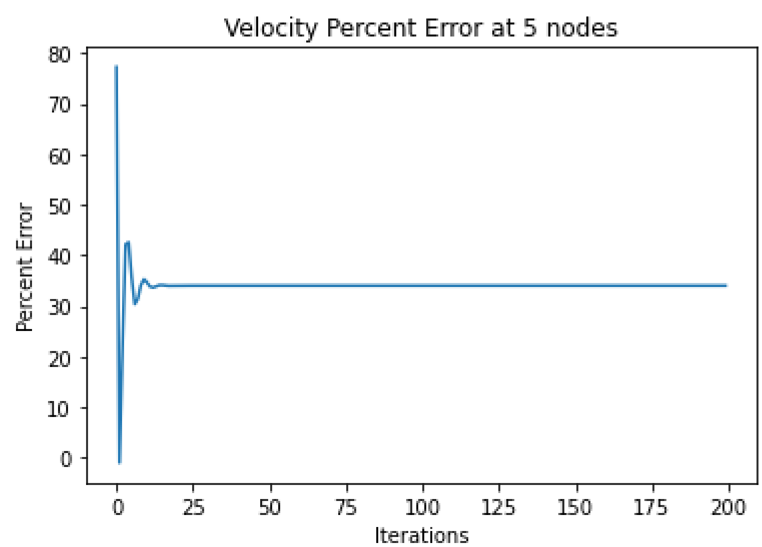
ME4565
Final Project
Ryan Huang

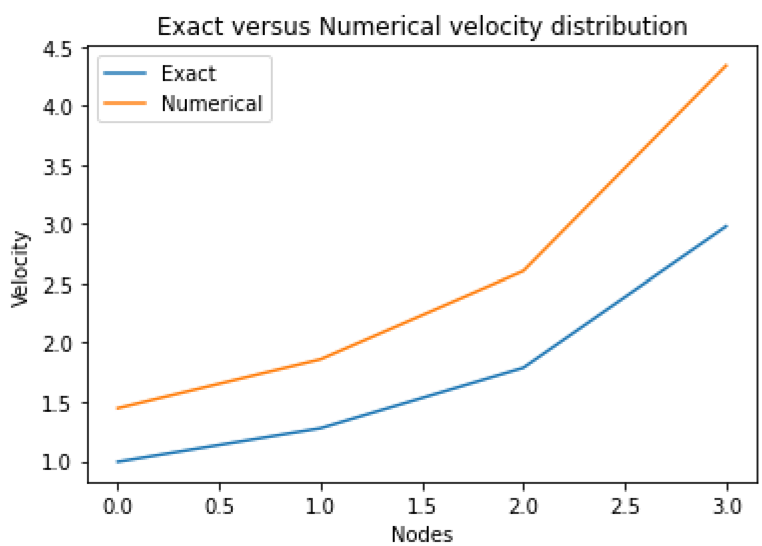
*Final Error is calculated from the exact solution provided by the pdfs
**Under Relaxation Variable naming is explained in the code

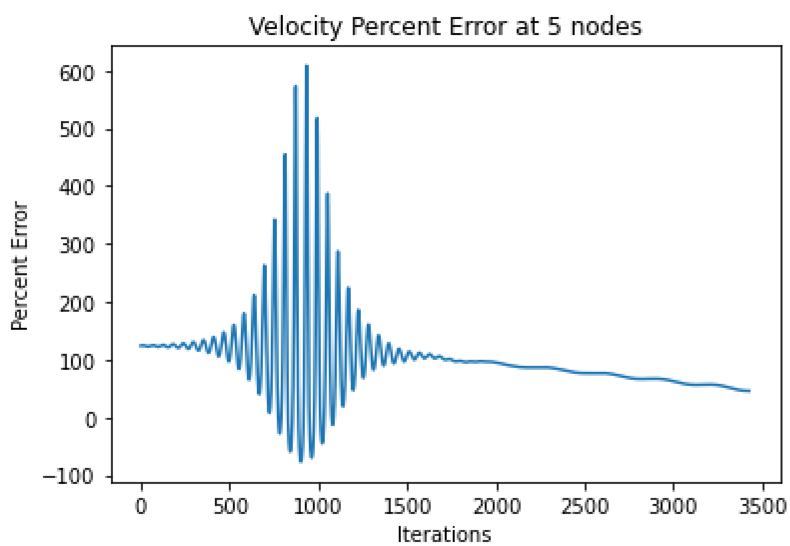
a) Five pressure nodes and four velocity nodes with upwind differencing
Final Velocity: [1.3320748 1.7126676 2.39773464 3.99622439]
Final Pressure: [9.28135858 8.35710938 7.65768656 5.74913138 0.   ]
Final Error: 34.03851857716453%
No Under Relaxation required to converge





b) Five pressure nodes and four velocity nods with central differencing
Final Velocity: [1.44707042 1.86051912 2.60472676 4.34121127]
Final Pressure: [9.81559177e+00 4.19744998e+04 5.22502400e+04 5.16817005e+04 0.00000000e+00]
Final Error: 45.60982326835676%
Under Relaxation Factors:
UR = 0.999999, a = 0.999, aL = 0.999, aR = 0.00001, SU = 0.999, SUL = 0.9999999999, SUR = 0.999, q = 0.9999





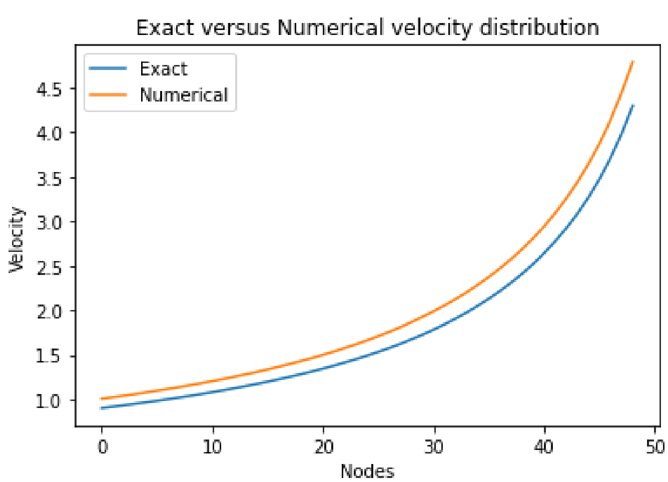
c) Fifty pressure nodes and forty-nine velocity nodes with upwind differencing

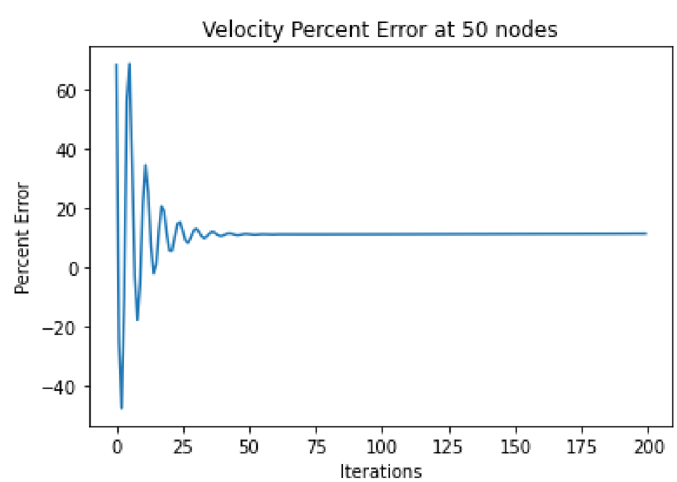
Final Velocity: [1.0054644 1.02229226 1.03969297 1.05769632 1.07633414 1.09564058
1.11565228 1.1364086 1.15795189 1.18032777 1.20358546 1.22777814
1.25296333 1.2792034 1.30656604 1.33512486 1.36496005 1.39615914
1.42881783 1.46304101 1.49894386 1.53665314 1.5763087 1.61806523
1.66209421 1.70858636 1.75775431 1.80983592 1.86509808 1.92384133
1.98640528 2.0531752 2.12458999 2.20115179 2.28343784 2.37211504
2.46795807 2.5718721 2.68492142 2.80836608 2.94370903 3.09275758
3.25770465 3.44123731 3.64668432 3.87821983 4.14114998 4.44232453
4.79074214]

Final Pressure: [9.50273487   8.99715143   14.36139434   25.88332772 43.64258902
67.71641136   98.17893736 135.10042386 178.5463202   228.57619926
285.24251717 348.58917298 418.64983472 495.44599147 578.98468423
669.25585846 766.22927076 869.85086807 980.03854165 1096.67713741
1219.61257942 1348.64493216 1483.52018858 1623.92052272 1769.45268506
1919.63414223 2073.87646542 2231.46534796 2391.53647274 2553.04624427
2714.7361313 2875.08901416 3032.27546492 3184.08726879 3327.85466082
3460.34262073 3577.62001821 3674.89325313 3746.29302579 3784.59860314
3780.87780892 3724.01201197 3600.06211755 3391.41155966 3075.59155126
2623.64561859 1997.81301124 1148.18301803   7.75594025   0.   ]

Final Error: 11.497565683885448%

Under Relaxation Factors: UR = 0.15, a = 0.05, aL = 0.0001, aR = 0.02, SU = 0.99999999, SUL = 0.08, SUR = 0.3, q = 0.005





d) Fifty pressure nodes and forty nine velocity nodes with central differencing

This scenario never converged for any combination of under relaxation factors. I was unable to find a combination of under relaxation values that converged.

The lowest error I could achieve was in the first iteration of 3.644534711994198e+225 before the values were not representable.

Final Velocity: [nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan]
Final Pressure: [nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan 0.]

Final Error: nan%
Minimum Error: 3.644534711994198e+225

Discretization and Formulas used

$\rho u \frac{du}{dx} = -A \frac{dp}{dx}$

$(\rho u A)_e u_e - (\rho u A)_w u_w = \frac{\Delta P}{\Delta x} \Delta V$

Upwind Difference Scheme

$a_P u_P^* = a_W u_W^* + a_E u_E^* + S_u$

$a_W = D_w + max(F_w, 0) = F_w = \rho A_w u_w$

$a_E = D_e + max(0, -F_e) = 0$

$a_P = a_W + a_E + F_e - F_w$

$S_u = \Delta P \cdot A_P$

Central Difference Scheme

$a_P u_P^* = a_W u_W^* + a_E u_E^* + S_u$

$a_W = D_W + \frac{1}{2} F_W = \frac{1}{2} \rho A_w u_w$

$a_E = D_E - \frac{1}{2} F_E = -\frac{1}{2} \rho A_e u_e$

$a_P = a_W + a_E + F_e - F_w$

$S_u = \Delta P \cdot A_P$

Pressure Correction

$(\rho u A)_e - (\rho u A)_w = 0$

$a_P P_P' = a_W P_W' + a_E P_E' + b'$

$a_W = (\rho d A)_w, \quad a_e = (\rho d A)_e$

$b' = F_W^* - F_e^*$

$p = p^* + p'$

$u = u^* + d(p_i' - p_{i+1}')$

The solver is written such that only NODES, the number of nodes, has to be changed to switch between a) and c) or b) and d).
Under Relaxation factors were tuned by inspection and checking whether the velocity error plot would diverge or converge.
To switch between Upwind and Central differencing, only aW and aE were altered to reflect their respective equations.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 29 13:27:35 2022

@author: ryanhuang
"""
import numpy as np
import matplotlib.pyplot as plt

'''
Converged scenarios with a decent amount of error
-------------------------------------------------------------------------------------------
For 5 Nodes, 200 Iterations, and without Under-Relaxing

Final Velocity: [1.3320748  1.7126676  2.39773464 3.99622439]
Final Pressure: [9.28135858 8.35710938 7.65768656 5.74913138 0.        ]

-------------------------------------------------------------------------------------------
For 5 Nodes, 200 Iterations, and Under-Relaxing

UR = 0.2, a = SU = SUL = SUR = 0.4, aL = 0.1, aR = 0.2, q = 0.1

Final Velocity: [1.35765631 1.74555811 2.44378135 4.07296892]
Final Pressure: [9.25340322 8.29324523 7.94834747 5.96707075 0.        ]

-------------------------------------------------------------------------------------------
For 50 nodes, 200 Iterations, and Under-Relaxing

UR = 0.15, a = 0.05, aL = 0.0001, aR = 0.02, SU = 0.99999999, SUL = 0.08, SUR = 0.3, q = 0.005

Velocity Error: 11.4975656%

'''


# THESE PARAMETERS CAN BE CHANGED TO TEST DIFFERENT CASES
NUM_ITER = 200                                      # number of iterations
NODES = 50                                           # number of nodes to use
test_converge = False                                # break the loop if error reaches a low enough amount
under_relax = True                                   # must adjust the underrelaxation constants if True

# UNDER RELAXATION CONSTANTS
UR = 0.15                                            # for V and P
a = 0.05                                             # for aP
aL = 0.0001                                            # for aP_right
aR = 0.02                                            # for aP_left
SU = 0.99999999                                            # for Su
SUL = 0.08                                           # for Su_left
SUR = 0.3                                           # for Su_right
q = 0.005                                            # for d


# THESE PARAMETERS ARE THE INITIAL CONDITIONS
POINTS = NODES - 1
DENS = 1                                            # density
P_inlet = 10                                        # inlet pressure
P_outlet = 0                                        # outlet pressure

# THESE VARIABLES REPRESENT THE INITIAL GUESSES MADE IN THE ALGORITHM
P_i = np.linspace(P_inlet, P_outlet, num=NODES)     # initial pressure guesses
area = np.linspace(0.5, 0.1, num=(NODES+POINTS))                            # area per point with linear spacing
AREA_nodes = np.array([area[i] for i in range(len(area)) if i % 2 == 0])        # area per node with linear spacing
AREA_points = np.array([area[i] for i in range(len(area)) if i % 2 !=0])        # area per point with linear spacing


# THESE VARIABLES ARE FOR CALCULATING PERCENT ERROR AT THE END
exact_mass_flow_rate = 0.44721                      # exact mass flow rate
guessed_mass_flow_rate = 1                          # guessed mass flow_rate
exact_vel = np.array([exact_mass_flow_rate/(DENS * AREA_points[i]) for i in range(POINTS)])

```

```


def main():
    # variables for underrelaxation calculations
    UR_aP = 0
    UR_Su = 0
    UR_aP_left = 0
    UR_aP_right = 0
    UR_Su_right = 0
    UR_Su_left = 0
    UR_d = []



    # lists to track convergence
    aP_plt = []
    aP_R_plt = []
    aP_L_plt = []


    vel_error = []                                  # store the velocity error per iteration

    vel_mat = np.zeros((NODES-1,NODES-1))           # declare velocity matrix
    source_mat = np.zeros((NODES-1, 1))             # declare source matrix

    pressure_mat = np.zeros((NODES,NODES))          # declare pressure correction matrix
    temp_mat = np.zeros((NODES,1))                  # this matrix is used in conjunction with the pressure correction matrix



    V = np.array([guessed_mass_flow_rate/(DENS * AREA_points[i]) for i in range(POINTS)])

    print(f"Velocities: {exact_vel}\n")

    #----------------------------------------------------------------------
    #solve for the interior nodes

    for i in range(NUM_ITER):

        d = []                                          # parameter used in pressure correction

        for node in range(1, POINTS-1):                  # only consider the interior nodes and not the boundaries
            Fw = DENS * (0.5*(V[node-1] + V[node])) * AREA_nodes[node]
            Fe = DENS * (0.5*(V[node] + V[node+1])) * AREA_nodes[node+1]
            aW = Fw
            aE = 0

            # underrelaxation
            if i == 0 or under_relax == False:
                aP = aW + aE + Fe - Fw
                Su = (P_i[node] - P_i[node+1]) * AREA_points[node]
            else:
                aP = a*UR_aP + (1-a)*(aW + aE + Fe - Fw)
                Su = SU*UR_Su + (1-SU)*((P_i[node] - P_i[node+1]) * AREA_points[node])

            d.append(AREA_points[node]/aP)

            vel_mat[node][node-1] = -aW
            vel_mat[node][node] = aP
            source_mat[node] = Su

        #----------------------------------------------------------------------
        # solve for the boundary nodes
        #----------------------------------------------------------------------
        # leftmost boundary

        Fe_left = DENS * (0.5*(V[0] + V[1])) * AREA_nodes[1]

        uA = V[0] * (AREA_points[0]/AREA_nodes[0])
        Fw_left = DENS * uA * AREA_nodes[0]

        aW_left = aE_left = 0

        # underrelaxation
        if i == 0 or under_relax == False:
```

```
            aP_left = Fe_left + Fw_left * 0.5 * (AREA_points[0]/AREA_nodes[0])**2
            Su_left = (P_i[0] - P_i[1]) * AREA_points[0] + Fw_left * (AREA_points[0]/AREA_nodes[0]) * V[0]
        else:
            aP_left = aL*UR_aP_left + (1-aL)*(Fe_left + Fw_left * 0.5 * (AREA_points[0]/AREA_nodes[0])**2)
            Su_left = SUL*UR_Su_left + (1-SUL)*((P_i[0] - P_i[1]) * AREA_points[0] + Fw_left * (AREA_points[0]/AREA_nodes[0]) * ol


        vel_mat[0][0] = aP_left
        source_mat[0] = Su_left
        d.insert(0, AREA_points[0]/aP_left)

        #--------------------------------------------------------------------
        # rightmost boundary

        if i == 0:
            Fe_right = guessed_mass_flow_rate
        else:
            Fe_right = DENS * V[-1] * AREA_points[-1]

        Fw_right = DENS * (0.5*(V[-2] + V[-1])) * AREA_nodes[-2]
        aW_right = Fw_right
        aE_right = 0

        #underrelaxation
        if i == 0 or under_relax == False:
            aP_right = aW_right + aE_right + Fe_right - Fw_right
            Su_right = (P_i[-2] - P_i[-1]) * AREA_points[-1]
        else:
            aP_right = aR*UR_aP_right + (1-aR)*(aW_right + aE_right + Fe_right - Fw_right)
            Su_right = SUR*UR_Su_right + (1-SUR)*((P_i[-2] - P_i[-1]) * AREA_points[-1])

        vel_mat[-1][-2] = -aW_right
        vel_mat[-1][-1] = aP_right
        source_mat[-1] = Su_right
        d.append(AREA_points[-1]/aP_right)

        # underrelaxation for d
        temp_d = d

        if under_relax == True:
            if i != 0:
                for ele in range(len(d)):
                    d[ele] = q*UR_d[ele] + (1-q)*temp_d[ele]

        # print(vel_mat)
        # print(source_mat)
        # print(d)

        #--------------------------------------------------------------------
        #solve for the velocity field

        sol = np.linalg.solve(vel_mat, source_mat)
        # print(f"{sol}\n")

        #--------------------------------------------------------------------
        # pressure correction


        for node in range(1, NODES-1):
            aW1 = DENS * d[node-1] * AREA_points[node-1]
            aE1 = DENS * d[node] * AREA_points[node]
            Fw1 = DENS * sol[node-1] * AREA_points[node-1]
            Fe1 = DENS * sol[node] * AREA_points[node]
            aP1 = aW1 + aE1
            b = Fw1 - Fe1

            pressure_mat[node][node-1] = -aW1
            pressure_mat[node][node] = aP1
            pressure_mat[node][node+1] = -aE1


            temp_mat[node] = b

        # set the correction pressures at the boundaries equal to 0 by deleting the edges of the matrix

```

```
        #these two reassignments are two preserve the dimensions of pressure_mat and temp_mat
        press_mat = pressure_mat
        temp1_mat = temp_mat

        press_mat = np.delete(press_mat, 0, axis = 0)
        press_mat = np.delete(press_mat, -1, axis = 0)
        press_mat = np.delete(press_mat, 0, axis = 1)
        press_mat = np.delete(press_mat, -1, axis = 1)
        temp1_mat = np.delete(temp1_mat, 0)
        temp1_mat = np.delete(temp1_mat, -1)


        press_corr = np.linalg.solve(press_mat, temp1_mat)
        #--------------------------------------------------------------------
        # correct pressures

        oldP = P_i

        for p in range(1,NODES-1):
            P_i[p] += press_corr[p-1]

        # add 0 to both ends of the pressure correction matrix to preserve length after the edges were deleted earlier
        press_corr = np.insert(press_corr, 0, 0)
        press_corr = np.append(press_corr, 0)

        #--------------------------------------------------------------------
        # correct velocities
        for v in range(0, POINTS):
            sol[v] += d[v] * (press_corr[v] - press_corr[v+1])


        #--------------------------------------------------------------------
        # solve for nodal pressure at A

        P_i[0] = P_inlet - 0.5 * DENS*((sol[0] * AREA_points[0])/AREA_nodes[0])**2

        #--------------------------------------------------------------------
        # underelaxation factor

        oldV = V


        for val in range(len(V)):
            V[val] = UR * oldV[val] + (1-UR) * sol[val]
            P_i[val] = UR * oldP[val] + (1-UR) * P_i[val]

        # print(f"Velocities: {V}")
        # print(f"Pressures: {P_i}\n")

        UR_aP = aP
        UR_Su = Su
        UR_aP_left = aP_left
        UR_aP_right = aP_right
        UR_Su_right = Su_right
        UR_Su_left = Su_left
        UR_d = d


        aP_plt.append(UR_aP)
        aP_R_plt.append(UR_aP_left)
        aP_L_plt.append(UR_aP_right)

        vel_error.append((V[0]/exact_vel[0] - 1)*100)


        # breaks the loop if an error reaches a certain point
        if test_converge:
            if len(vel_error) > 2:
                if abs(vel_error[-1]) < 0.01 or (abs(vel_error[-1] - vel_error[-2])) < 0.001:
                    print("converged?")
                    break

    # display the final velocities, pressures, and errors
    print(f"Final Velocity: {V}")
    print(f"Final Pressure: {P_i}\n")
```

```
    print(f"Final Error: {vel_error[-1]}%")


    # take the absolute value of all the errors to find the index of the minimum error
    min_err = [abs(err) for err in vel_error]
    min_err_index = min_err.index(min(min_err))

    print(f"Minimum Error: {vel_error[min_err_index]}")
    print(f"Iteration of Minimum Error : {min_err_index}")
    print(f"Last error difference: {vel_error[-1] - vel_error[-2]}")


    #plot the percent error per iteration
    plt.figure(0)
    plt.plot(list(range(len(vel_error))), vel_error)
    plt.xlabel("Iterations")
    plt.ylabel("Percent Error")
    plt.title(f"Velocity Percent Error at {NODES} nodes")


    #plot velocity distribution
    plt.figure(1)
    plt.title("Exact versus Numerical velocity distribution")
    plt.xlabel("Nodes")
    plt.ylabel("Velocity")
    plt.plot(list(range(len(exact_vel))), exact_vel, label="Exact")
    plt.plot(list(range(len(V))), V, label="Numerical")
    plt.legend()





if __name__ == "__main__":
    main()
```

PDF document made with CodePrint using [Prism](#)