Justine John "JJ" A. Serdoncillo
April 7, 2023

Homework 3

Number 2

Upon using the Falkner-Skan laminar boundary layer equations, the following plots were made for different values of α. The figure below shows the change of $f'(\eta)$ or u/U_∞ with respect to η. The domain used is only until $\eta = 10$ and is because if the shear stress was right, the solution converges fast to 1. Other reasons are that the equation is sensitive to the initial condition, so marching along will produce errors which can grow fast that will not reflect the expected behavior. The shear stress was found by using a binary split guess shooting method which gives the an accurate guess within 50 iterations. In order to increase the speed of the code and it's accuracy, the shooting method will stop evaluating the ordinary differential equation (ODE) if f' is above a threshold over 1. After finding the necessary shear stress, the shape factor was calculated by integrating along the entire domain by the use of a left Reimann sum. This doesn't affect the accuracy that much due to the chosen $\Delta\eta = 0.001$.

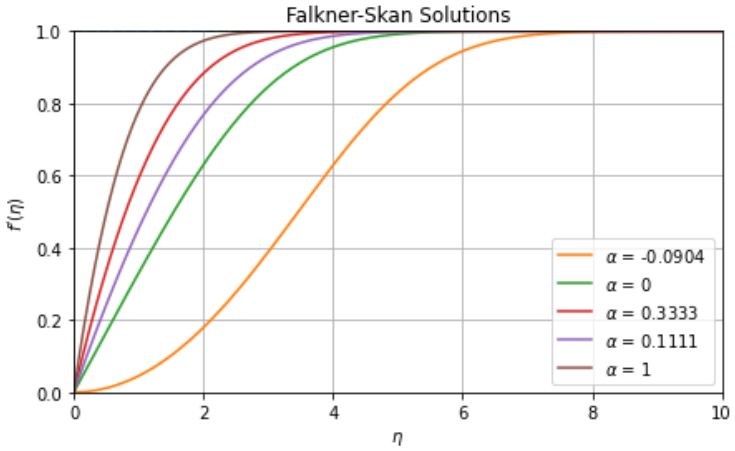

Figure 1: Numerical solution of Falker-Skan in terms of η

α	$f''(0)$	H
-0.0904	0	4.0282
0	0.3321	2.5931
1/3	.7574	2.2893
1/9	0.5118	2.409
1	1.2326	2.2334

Table 1: Vertex Coordinates of the y-direction prism

In order to match the figure 10.8 in Kundu's textbook, it was noted that the x scale isn't along η. Therefore it was necessary to scale it properly. It can be seen from the figure below that the produced plot looked similar to the textbook figures.

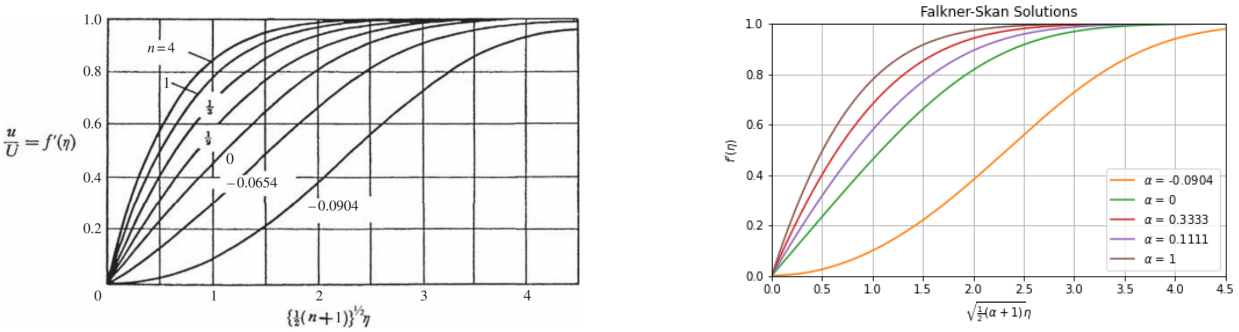

Figure 2: Comparison of Figure 10.8 from KCD and Scaled Numerical solution of Falkner-Skan

Number 3 c)

To find the velocity profile of the Non-newtonian boundary layer, a different ODE was solved. By using the same shooting method, a value of $f''(0) = 0.619$ was found. Afterwards the velocity profile was compared with the Newtonian Blasius boundary layer. It can be seen that the velocity increases faster along the y axis.

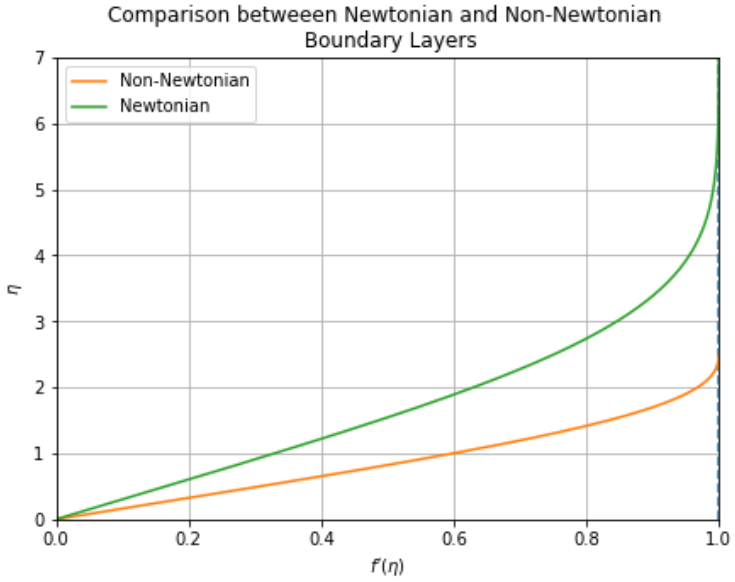

Figure 3: Newtonian and Non-Newtonian Boundary Layers

Appendix)

Python Code for Problems 2

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  6 21:15 2023

@author: jjser
Justine John A. Serdoncillo
AEM 8202 Fluids II
Homework3_P2_v4
    - No more functions
    - Summarized all of the plots
    - Working shooting method
    - shapeFactor()
    - added shooting
"""
import numpy as np
import matplotlib.pyplot as plt

# %%
"""
    Personal Created Functions
    - g1, g2 and g3 represents 3 linear ODEs
    - rk4 is the Runge-Kutta 4th Order Method
    - shoot is the shooting method needed
    - shapeFactor()
"""

# Define personal created functions
def g1(f, df, ddf, a, n):
    """Returns the value of the first ODE"""
    return df

def g2(f, df, ddf, a, n):
    """Returns the value of the second ODE"""
    return ddf

def g3(f, df, ddf, a, n):
    """Returns the value of the third ODE"""
    return -0.5 * (a+1) * f * ddf - a * (1 - df**2)

def rk4(ni, nf, f0, df0, ddf0, dn, a):
    """
    Returns the numerical solutions for the Falkner-Skan equations using
    the fourth-order Runge-Kutta method.
    """
    N = int((nf - ni) / dn)
    n = np.linspace(ni, nf,N+1)
    scaled = ( ( 0.5 * (a+1) )**(1/2) ) * n
    F = np.zeros(len(n))
    dF = np.zeros(len(n))
    ddF = np.zeros(len(n))
    F[0] = f0
    dF[0] = df0
    ddF[0] = ddf0
    for i in range(len(n)-1):
        k01 = g1(F[i], dF[i], ddF[i], a, n[i])
        k02 = g2(F[i], dF[i], ddF[i], a, n[i])
        k03 = g3(F[i], dF[i], ddF[i], a, n[i])

        k11 = g1(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, a, n[i]+dn/2)
        k12 = g2(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, a, n[i]+dn/2)
        k13 = g3(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, a, n[i]+dn/2)

        k21 = g1(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, a, n[i]+dn/2)
        k22 = g2(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, a, n[i]+dn/2)
        k23 = g3(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, a, n[i]+dn/2)

        k31 = g1(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, a, n[i]+dn)
```

```
        k32 = g2(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, a, n[i]+dn)
        k33 = g3(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, a, n[i]+dn)

        F[i+1] = F[i] + dn/6 * (k01 + 2*k11 + 2*k21 + k31)
        dF[i+1] = dF[i] + dn/6 * (k02 + 2*k12 + 2*k22 + k32)
        ddF[i+1] = ddF[i] + dn/6 * (k03 + 2*k13 + 2*k23 + k33)

        if dF[i+1] > 3:
            #print("bruh you high messed up")
            dF[i+1] = 3
            break
        elif dF[i+1] < -1:
            #print("bruh you low messed up")
            dF[i+1] = 0
            break
    return n, F, dF, ddF, scaled

def shoot(ni, nf, f0, df0, dn, a, up, down, show = False, showShow = True):
    ite = 0
    MAXITE = 100
    bad = True
    TOL = 10E-14
    target = 1
    while bad and ite < MAXITE:
        skrt = (up + down)/2
        ite += 1
        n, F, dF, ddF, s = rk4(ni, nf, f0, df0, skrt, dn, a)
        if np.max(dF) > target + TOL:
            down = skrt
        elif np.max(dF) < target - TOL:
            up = skrt
        else:
            bad = False
        if show:
            print(f"I shot with {skrt}")
            print(f"It's been {ite} days since cookies")
            print(f"This is my {np.around(np.max(dF),6)} power \n ")
            print(f"For the last time {np.around(dF[-1],6)} ")
        if showShow:
            plt.plot(n, dF, label=str(ite))
    dff0 = skrt
    if show:
        plt.title(f"This is the {ite} time bro")
        plt.legend()
    return dff0, ite

def shapeFactor(n, dF):
    deltaStar = 0
    theta = 0
    # Left Reimann Sum
    for i in range(len(n)-1):
        dn = n[i+1] - n[i]
        deltaStar += (1 - dF[i]) * dn
        theta += dF[i] * (1 - dF[i]) * dn
    H = deltaStar/theta
    return H

# %%
"""
    Falkner-Skan Similarity Solutions of the Laminar Boundary-Layer Equations
    - alphas = [-0.0904, 0, 1/3, 1/9, 1]
    - fppp + (a+1)/2 * f * fpp - a * fp**2 + a = 0
        - Given: f(0) = 0, fp(0) = 0
        - Shoot with: fpp(0) = idk
        - Target: fp(inf) = 1
"""

# Define problem-specific variables
f0 = 0        # initial value of the dependent variable f
df0 = 0       # initial value of the first derivative of f
ni = 0.0      # initial value of the independent variable n
nf = 10.0     # final value of the independent variable n
```

```
dn = 0.001      # step size
alphas = [-0.0904, 0, 1/3, 1/9, 1] # constant parameter of the Falkner-Skan equations

# Calculate wall shear stress using shooting method
#ddf0s = np.zeros(len(alphas))
#for i in range(len(alphas)-1):
#    ddf0s[i+1], ite = shoot(ni, nf, f0, df0, dn, alphas[i+1], 0, 2)
ddf0s = [0, 0.3321, 0.7574, 0.5118, 1.2326] # initial value of the second derivative of f

# Create two subplots to display the solutions
fig, ax = plt.subplots(figsize=(7,4))
fig1, ax1 = plt.subplots(figsize=(7,4))

# Set the titles and axis labels for the plots
ax.set_title('Falkner-Skan Solutions')
ax.set_xlabel('$ \\eta $')
ax.set_ylabel('$ f\'(\\eta) $')
ax1.set_title('Falkner-Skan Solutions')
ax1.set_xlabel('$ \\sqrt{\\frac{1}{2} (\\alpha + 1)}\\eta $')
ax1.set_ylabel('$ f\'(\\eta) $')

# Set the x and y limits for the plots
ax.set_xlim([0,10])
ax.set_ylim([0,1.0])
ax1.set_xlim([0,4.5])
ax1.set_ylim([0,1.0])

# Plot a horizontal dashed line at y=1 to show the target value
ax.plot(np.linspace(0,100,100,True), np.ones(100), '--')
ax1.plot(np.linspace(0,100,100,True), np.ones(100), '--')

# Loop over the different values of alpha to solve the Falkner-Skan equations
for i in range(len(alphas)):
    # Solve the Falkner-Skan equations using the Runge-Kutta method
    n, F, dF, ddF, scaled = rk4(ni, nf, f0, df0, ddf0s[i], dn, alphas[i])
    H = shapeFactor(n, dF)

    # Print the value of alpha and f''(0) for this solution
    print(f"For alpha = {np.around(alphas[i],4)}, ddf(0) = {np.around(ddf0s[i],4)}")
    # Print the value of the shape factor
    print(f"The corresponding shape factor is {np.around(H,4)}")

    # Add a label for the current value of alpha to the plot
    label = "$\\alpha$ = " + str(np.around(alphas[i],4))

    # Plot the solution for f'(eta) on both subplots
    ax.plot(n, dF, label=label)
    ax1.plot(scaled, dF, label=label)

# Add a legend and gridlines to both subplots
ax.grid()
ax.legend(loc='lower right')
ax1.grid()
ax1.legend(loc='lower right')

# %%
"""
    Shooting
"""
for i in range(len(alphas)-1):
    yay, skrt = shoot(0, 10, 0, 0, 0.001, i+1, 0, 2, True)
    print(f"For alpha = {np.around(alphas[i],4)} \n I got ddF0 = {yay} and it only took me {
    skrt}")

## using (0,2) and 10E-14, (-1, 3)
# 0 -> 0.332057337203 in 43 days
# 1 -> 1.232587656820 in 45 days
# 1/3 -> 0.7574475807215 in 47 days
# 1/9 -> 0.511842057835 in 45 days
#  -0.0904 -> 0
```

Python Code for Problems 3

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  3 21:44:28 2023

@author: jjser
Justine John A. Serdoncillo
AEM 8202 Fluids II
Homework 3 Problem 3 v2
    - Solve the Non-Newtonian Similarity ODE
    - Working shooting method
    - added shooting
"""
import numpy as np
import matplotlib.pyplot as plt
import warnings

# %%
"""
    Personal Created Functions
    - g1, g2 and g3 represents 3 linear ODEs
    - rk4 is the Runge-Kutta 4th Order Method
    - shoot is the shooting method needed
"""
def g1(f, df, ddf, n):
    """Returns the value of the first ODE"""
    return df

def g2(f, df, ddf, n):
    """Returns the value of the second ODE"""
    return ddf

def g3(f, df, ddf, n):
    """Returns the value of the third ODE"""
    try:
        with warnings.catch_warnings(record=True) as w:
            warnings.simplefilter("always")
            result = -f * ddf / (ddf**(1/2))
            if w:
                print(f"Warning occurred with: \n f={np.around(f,6)},\n df={np.around(df,6)},\n ddf={np.around(ddf,6)},\n n={np.around(n,6)}")
                print(f"Warning message: {w[0].message}")
    except ZeroDivisionError:
        print(f"ZeroDivisionError occurred with  \n f={np.around(f,6)},\n df={np.around(df,6)},\n ddf={np.around(ddf,6)},\n n={np.around(n,6)}")
        result = 0  # or set a suitable value when a division by zero occurs
    return result

def rk4(ni, nf, f0, df0, ddf0, dn):
    N = int((nf - ni) / dn)
    n = np.linspace(ni, nf, N+1)
    F = np.ones(len(n))
    dF = np.ones(len(n))
    ddF = np.ones(len(n))
    F[0] = f0
    dF[0] = df0
    ddF[0] = ddf0
    for i in range(len(n)-1):
        k01 = g1(F[i], dF[i], ddF[i], n[i])
        k02 = g2(F[i], dF[i], ddF[i], n[i])
        k03 = g3(F[i], dF[i], ddF[i], n[i])

        k11 = g1(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, n[i]+dn/2)
        k12 = g2(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, n[i]+dn/2)
        k13 = g3(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, n[i]+dn/2)

        k21 = g1(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, n[i]+dn/2)
        k22 = g2(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, n[i]+dn/2)
        k23 = g3(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, n[i]+dn/2)

        k31 = g1(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, n[i]+dn)
```

```
        k32 = g2(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, n[i]+dn)
        k33 = g3(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, n[i]+dn)

        F[i+1] = F[i] + dn/6 * (k01 + 2*k11 + 2*k21 + k31)
        dF[i+1] = dF[i] + dn/6 * (k02 + 2*k12 + 2*k22 + k32)
        ddF[i+1] = ddF[i] + dn/6 * (k03 + 2*k13 + 2*k23 + k33)

        if dF[i+1] > 3:
            #print("bruh you high messed up")
            dF[i+1] = 3
            break
        elif dF[i+1] < -1:
            #print("bruh you low messed up")
            dF[i+1] = 0
            break

    return n, F, dF, ddF

def shoot(ni, nf, f0, df0, dn, up, down, show = False):
    ite = 0
    MAXITE = 100
    bad = True
    TOL = 10E-14
    target = 1
    while bad and ite < MAXITE:
        skrt = (up + down)/2
        ite += 1
        n, F, dF, ddF = rk4(ni, nf, f0, df0, skrt, dn)
        if np.max(dF) > target + TOL:
            down = skrt
        elif np.max(dF) < target - TOL:
            up = skrt
        else:
            bad = False
        if show:
            print(f"I shot with {skrt}")
            print(f"It's been {ite} days since cookies")
            print(f"This is my {np.around(np.max(dF),6)} power \n ")
            #print(f"For the last time {np.around(dF[-1],6)} ")
            plt.plot(n, dF, label=str(ite))
    dff0 = skrt
    if show:
        plt.title(f"This is the {ite} time bro")
        plt.legend()
    return dff0, ite


# %%
"""
   Non-Newtonian Boundary Layer Profile
   - dddf * ddf^(1/2) + f * ddf = 0
"""

# Define problem-specific variables
f0 = 0        # initial value of the dependent variable f
df0 = 0       # initial value of the first derivative of f
ni = 0.0      # initial value of the independent variable n
nf = 2.5      # final value of the independent variable n
dn = 0.001     # step size

# Calculate wall shear stress using shooting method
#ddf0, ite = shoot(ni, nf, f0, df0, dn, 0.5, 0.75)
ddf0 = 0.619  # initial value of the second derivative of f

# Solve the Falkner-Skan equations using the Runge-Kutta method
n, F, dF, ddF = rk4(ni, nf, f0, df0, ddf0, dn)
print(f"For Non-Newtonian Boundary Layers, ddf(0) = {np.around(ddf0,4)}")

# Create a plot to display the solution
fig, ax = plt.subplots(figsize=(7,5))

# Set the title and axis labels for the plot
```

```
ax.set_title('Comparison betweeen Newtonian and Non-Newtonian \n Boundary Layers')
ax.set_ylabel('$ \\eta $')
ax.set_xlabel('$ f\'(\\eta) $')

# Set the x and y limits for the plot
ax.set_ylim([0,7.0])
ax.set_xlim([0,1.0])

# Plot a horizontal dashed line at y=1 to show the target value
ax.plot(np.ones(100), np.linspace(0,100,100,True), '--')

# Plot the solution for f'(eta)
ax.plot(dF, n, label='Non-Newtonian')

# Add gridlines to the plot
ax.grid()

# %%
"""
    Blasius Newtonian Solution Taken from Problem 2
"""
# Define personal created functions
def g1(f, df, ddf, a, n):
    """Returns the value of the first ODE"""
    return df

def g2(f, df, ddf, a, n):
    """Returns the value of the second ODE"""
    return ddf

def g3(f, df, ddf, a, n):
    """Returns the value of the third ODE"""
    return -0.5 * (a+1) * f * ddf - a * (1 - df**2)

def rk4(ni, nf, f0, df0, ddf0, dn, a):
    """
    Returns the numerical solutions for the Falkner-Skan equations using
    the fourth-order Runge-Kutta method.
    """
    N = int((nf - ni) / dn)
    n = np.linspace(ni, nf,N+1)
    scaled = ( ( 0.5 * (a+1) )**(1/2) ) * n
    F = np.zeros(len(n))
    dF = np.zeros(len(n))
    ddF = np.zeros(len(n))
    F[0] = f0
    dF[0] = df0
    ddF[0] = ddf0
    for i in range(len(n)-1):
        k01 = g1(F[i], dF[i], ddF[i], a, n[i])
        k02 = g2(F[i], dF[i], ddF[i], a, n[i])
        k03 = g3(F[i], dF[i], ddF[i], a, n[i])

        k11 = g1(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, a, n[i]+dn/2)
        k12 = g2(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, a, n[i]+dn/2)
        k13 = g3(F[i]+dn/2*k01, dF[i]+dn/2*k02, ddF[i]+dn/2*k03, a, n[i]+dn/2)

        k21 = g1(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, a, n[i]+dn/2)
        k22 = g2(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, a, n[i]+dn/2)
        k23 = g3(F[i]+dn/2*k11, dF[i]+dn/2*k12, ddF[i]+dn/2*k13, a, n[i]+dn/2)

        k31 = g1(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, a, n[i]+dn)
        k32 = g2(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, a, n[i]+dn)
        k33 = g3(F[i]+dn*k21, dF[i]+dn*k22, ddF[i]+dn*k23, a, n[i]+dn)

        F[i+1] = F[i] + dn/6 * (k01 + 2*k11 + 2*k21 + k31)
        dF[i+1] = dF[i] + dn/6 * (k02 + 2*k12 + 2*k22 + k32)
        ddF[i+1] = ddF[i] + dn/6 * (k03 + 2*k13 + 2*k23 + k33)
    return n, F, dF, ddF, scaled

# Define problem-specific variables
f0 = 0          # initial value of the dependent variable f
```

```
df0 = 0         # initial value of the first derivative of f
ni  = 0.0       # initial value of the independent variable n
nf  = 10.0      # final value of the independent variable n
dn  = 0.001     # step size
a   = 0         # constant parameter of the Falkner-Skan equations

ddf0 = 0.3321  # initial value of the second derivative of f

# Solve the Falkner-Skan equations using the Runge-Kutta method
n, F, dF, ddF, scaled = rk4(ni, nf, f0, df0, ddf0, dn, a)
print(f"For Newtonian Boundary Layers, ddf(0) = {np.around(ddf0,4)}")

# Plot the solution for f'(eta)
ax.plot(dF, n, label='Newtonian')

ax.legend(loc='upper left')


# %%
"""
    Shooting
"""
yay, skrt = shoot(0, 2.5, 0, 0, 0.001, 0.5, 0.75, True)
print(f"After shooting \n I got ddF0 = {yay} and it only took me {skrt}")
## using (0.5,0.75) and 10E-14, (-1, 3)
# -> 0.618964749629 in 41 days
```