MEEN 2450 Ryan Nalby u0848407 Homework 5 Page 1

Exercise 1:

$$\frac{d^2T}{dx^2} = \frac{hP}{KA}(T - T_\infty)$$

$$z = \frac{dT}{dx}$$

$$\boxed{\begin{array}{ll} \bullet\ \dfrac{dT}{dx} = z & T(x=0) = 600 \\ & z(x=0) = \beta \\ \bullet\ \dfrac{dz}{dx} = \dfrac{hP}{KA}(T - T_\infty) & \end{array}}$$

Aside for Exercise 3

where $h = 20\ W/m^2 \cdot K$

$L = 2.0m$ $K = 200\ W/m \cdot K$

$D = 0.1m$ $T_\infty = 300K$

$T_E = 350K$ $T_B = 600K$

$A = \pi r^2$

$A = \pi \left(\frac{0.1}{2}\right)^2$

$A = .00785\ m^2$

$P = \pi D = \pi(0.1) = .314\ m$

$$\frac{d}{dx}\begin{bmatrix} T \\ z \end{bmatrix} = f\left(x, \begin{bmatrix} T \\ z \end{bmatrix}\right)$$

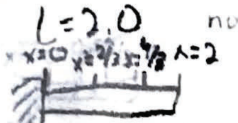
Cooling_fin: returns $\left[z, \frac{dz}{dx}\right]$ given $[x, [T, z], \text{constants}]$

okay: can get $\frac{dT}{dx}$ and $\frac{dz}{dx}$ at a given T, z — (no dependence on x, just then to can be used in Runge Kutta)

which can then be used to get T at a given x (by using Runge Kutta solver for a system of equations finding z)

Exercise 2

$L = 2.0$



node 1: 0m
node 2: 2/3m
node 3: 4/3m
node 4: 2m

$$\frac{dT}{dx} = z \qquad \frac{dz}{dx} = 4T - 1200 \qquad \begin{array}{l} T(x=0) = 600K \\ z(x=0) = \beta \end{array}$$

$$\frac{dz}{dx} = \frac{hP}{KA}(T - T_\infty) = \frac{(20)(.314)}{(200)(.00785)}(T - 300) = 4T - 1200$$

$z(x=0) = \beta_1$ Say $\beta_1 = 0$ find $T(\beta_1)$ using Runge Kutta 4

	n	T^n	z^n	k11	k12	k21	k22	k31	k32	k41	k42	T^{n+1}	z^{n+1}
x=0	0	600	0	0	1200	400	1200	400	1733.3	115.56	2266.67	906.17	1037.03
x=2/3	1	906.17	1037.037	1037.037	2424.69	1845.267	3807.407	2306.17	4885.05	4293.7	8574.5	2421.02	4190.82
x=4/3	2	2421.02	4190.82	4190.82	8484.09	7018.86	14071.86	8881.4	17842.6	16085.8	32168	8207.4	15799.8
x=2	3	8207.4	15799.8										

ME EN 2450 Ryan Dalby u084847 Homework 5 Page 2

Exercise 2 Continued

Say $\beta_2 = 4$ $z(\overset{x=}{0}) = \beta_2$

	n	T^n	z^n	k11	k12	k21	k22	k31	k32	k41	k42	T^{n+1}	z^{n+1}
x=0	0	600	4	4	12000	404	1205.3	405.8	1738.7	1163.1	2282.1	904.63	1045.1
x=2/3	1	904.63	1045.1	1045.1	2438.5	1852.96	383240	9322.41	4915.8	4322.3	8631.7	2434.99	4219.1
x=4/3	2	2434.99	4219.1	4219.1	8539.97	7065.26	11165.44	8440.92	17460.91	16193.1	32382.4	8260.1	15405.24
x=2	3	8260.05	15905.24										

T at boundary (level) should be 350 K

Now: $z^* = \beta_1 + \dfrac{\beta_2 - \beta_1}{T_1^{\beta_2} - T_1^{\beta_1}}\left(T_1 - T_1^{\beta_1}\right) = 0 + \dfrac{4-0}{8260.05 - 8207.4}(350 - 8207.4)$ Intermediate values stored in calculator

Since linear ODE

$\underline{z^* = -596.771}$

Now can solve IVP knowing $T^0 = 600$ $z^0 = -596.771$ at $x=0$

Runge Kutta Again

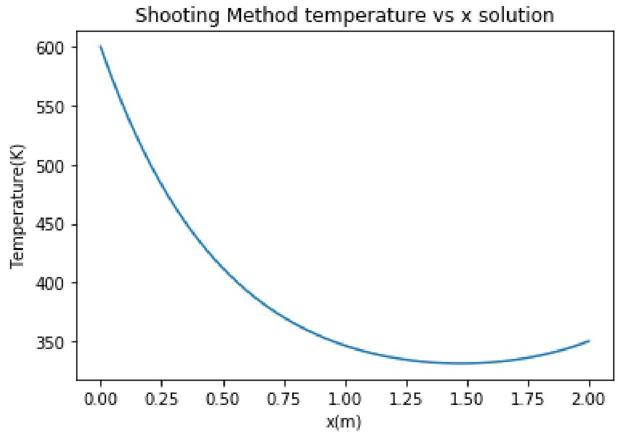
	n	T^n	z^n	k11	k12	k21	k22	k31	k32	k41	k42	T^{n+1}	z^{n+1}
x=0	0	600	-596.771	-596.77	1200	-196.77	404.31	-462.0	937.6	28.32	-32.0	390.4	-168.78
x=2/3	1	390.4	-168.78	-168.78	361.78	-48.19	136.75	-123.205	297.525	29.56	33.23	336.84	-28.34
x=4/3	2	336.84	-28.34	-28.34	147.55	20.79	104.64	8.17	175.277	88.46	164.83	350	70.15
x=2	3	350	70.15										

$$\boxed{\begin{aligned} T(\overset{x=}{0}) &= 600\ K \\ T(2/3) &= 390.44\ K \\ T(4/3) &= 336.89\ K \\ T(2) &= 350\ K \end{aligned}}$$

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW5/HW5b.py', wdir='C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW5')
```




```
In [2]:
```

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 19 15:56:27 2019
HW5b
@author: Ryan Dalby
"""

import math
import numpy as np
import matplotlib.pyplot as plt
from cooling_fin import cooling_fin
import scipy.integrate as integrate

import pandas as pd

#table = pd.DataFrame(columns = ["n", "y1n", "y2n", "k11", "k12", "k21", "k22", "k31", "k32", "k41

def runge_kutta_four(func, yInitial, xInitial, xFinal, numSteps):
    """
    Given a function(dy[]/dx = func(x,y[])) and yInitial[] values at an xInitial will find
    y[] at xFinal with a given number of steps between xInitial and xFinal
    """
    h = abs(xFinal - xInitial) / numSteps
    x = xInitial
    y = yInitial
#     row = np.empty(13)
    for i in range(numSteps):
#         print(x,y)
#         row[0] = i
#         row[1] = y[0]
#         row[2] = y[1]
        k1 = func(x, y)
#         row[3] = k1[0]
#         row[4] = k1[1]
        k2 = func(x + h/2, y + k1*h/2)
#         row[5] = k2[0]
#         row[6] = k2[1]
        k3 = func(x + h/2, y + k2*h/2)
#         row[7] = k3[0]
#         row[8] = k3[1]
        k4 = func(x + h, y + k3*h)
#         row[9] = k4[0]
#         row[10] = k4[1]
        y = y + (k1 + 2*k2 + 2*k3 + k4)/6 * h
        x = x + h
#         row[11] = y[0]
#         row[12] = y[1]
#         table.loc[i] = row
#     print(x,y)

    return y


##Define test ODE
#f = lambda t,y: 2 * (325/850) * (math.sin(t)**2) - (200*(1+y)**(3/2))/850
#print(rungeKutta4(f, 2, 0, 10, 50))
```

```
#w = integrate.solve_ivp(f, (0.0,10.0), np.array([2]))
#print(w)


def shootingMethodForCooling_fin(func, firstx, firstT, secondx, secondT, numberOfNodes, beta1, beta
    """
    From cooling_fin(passed in as func) which represents two linear ODEs say dT/dx = f(z) and dz/dx
    and two boundary conditions (x1, T1) and (x2, T2), and number of nodes(essentially how many ste
    and beta1 and beta2 guesses for a IVP that will satisfy a BVP
    Will return the z(firstx) that allows problem to be IVP and yet satisfies BVP
    """
    xInitial = firstx #initial x value
    xFinal = secondx #x value we will integrate to

    z1Initial = beta1 #z = dT/dx; first guess for our IVP; z(firstx) = beta1
    z2Initial = beta2 #z = dT/dx; second guess for our IVP; z(firstx) = beta2

    Tb = firstT #K temperature at beginning
    Te = secondT #K temperature at end

    T1Final,_ = runge_kutta_four(func, np.array([Tb, z1Initial]), xInitial, xFinal, numberOfNodes-1
#     print(table.to_string())
#     print()
    T2Final,_ = runge_kutta_four(func, np.array([Tb, z2Initial]), xInitial, xFinal, numberOfNodes-1
#     print(table.to_string())

#     print(T1Final, T2Final)
    zInitial = z1Initial + ((z2Initial - z1Initial) / (T2Final - T1Final)) * (Te - T1Final) #Now we

#     print(runge_kutta_four(func, np.array([Tb, zInitial]), xInitial, xFinal, numberOfNodes)) #Test

    #Now that we have zInitial just solve like IVP and can change xFinal to get temperature at diff
    #Now use this zInital value in another script to get an array of Ts at a given array of xs (cal
    return zInitial


def find_temperature_profile_shooting(firstx, firstT, secondx, secondT, numberOfNodes, shouldPlot =
    """
    From cooling_fin which represents two linear ODEs say dT/dx = f(z) and dz/dx = f(T)
    and two boundary conditions (x1, T1) and (x2, T2), and number of nodes(essentially how many ste
    and an xDesired where the temperature will be calcuated at
    Will display a plot of T vs x(if indicated to do so) and return np arrays of the x values and t
    This is essentially a driver for shootingMethodForCooling_fin
    """
    d = 0.1 #m
    h = 20 #W/m^2*K
    k = 200 #W/m * K
    Too = 300 #K surrounding temperature, time at t = infinity
    func = lambda x,y: cooling_fin(x, y, d, h, k, Too) #Cooling fin returns [dT/dx, dz/dx] given [x

    zInitialAtfirstx = shootingMethodForCooling_fin(func, 0, 600, 2.0, 350, numberOfNodes, 0, 4) #t
    xVals = np.linspace(firstx, secondx, numberOfNodes)
    TVals = []
    solverIterations = 30 #how many iterations our solver will take per value it is solving for
    for xVal in xVals:
```

```
        TVal, _= runge_kutta_four(func, [firstT, zInitialAtfirstx], firstx, xVal, solverIterations)
        TVals.append(TVal)
#    print(table.to_string())
    if(shouldPlot):
        fig, ax = plt.subplots()
        ax.plot(xVals, TVals)
        ax.set_title("Shooting Method temperature vs x solution")
        ax.set_xlabel("x(m)")
        ax.set_ylabel("Temperature(K)")
        
    return xVals, TVals

shootingx, shootingT = find_temperature_profile_shooting(0, 600, 2.0, 350, 51, shouldPlot=True)
```