Coffee Cooling Challenge #3: This Time It's Personal

Daniel Huantes

October 4, 2019

1 Introduction

In this final Coffee Cooling Challenge submission I added terms of the differential form of Newton's law of cooling, in order to create a coupled differential equation that represents the cooling of a cup of coffee due to both the heat lost to the air above the liquid, and the sides of the cups surrounding it. The equation I used to model the rate of change in the coffee's temperature, T, with respect to time was

$$\frac{\mathrm{d}T}{\mathrm{d}t} = r_{env}(T_{env} - T) + r_c(T_{mug} - T) \tag{1}$$

with T_{mug} being the temperature of the mug, changing with time, T_{env} being the temperature of the environment, assumed to be constant over time. We assumed that the room holding the coffee-mug system was large enough that any heat disappated in it would have a negligible effect on it's total temperature. I modeled $\frac{\mathrm{d}T_{mug}}{t}$, the rate of change of the temperature of the mug using

$$\frac{\mathrm{d}T_{mug}}{\mathrm{d}t} = r_c(T - T_{mug}) + r_{ce}(T_{env} - T_{mug}) \tag{2}$$

Unlike our previous challenges, this version of the coffee cooling problem was more difficult as it involed a coupled differential equation, with both T and T_{mug} varying with time, and affecting each other. Another extra challenge added was the ability for the program to handle a variable volume of coffee entered in a more familiar unit (oz), as well as using measurements of a coffee cup I own in order to provide more reliable estimates of the cooling. Both of these issues improve upon the accuracy of the previous problem by introducing more real life factors that affect cooling, and the utility of the program, with being able to model different cups and volumes of coffee.

2 Discussion

My initial challenge in creating this version of the Coffee Cooling Challenge was creating a program that could take a variable volume of coffee V and with the physical parameters of the mug holding the coffee, could find the necessary volumes and areas required to calculate the 3 r coefficients in Newton's Law of Cooling I used. I was going to use a truncated cone [1] in order to model the geometry of a coffee mug such as my own, which although almost cylindrical, like many cups has a mouth wider than its base. My overall goal was to be able to develop a function in order to find the surface area of coffee in contact with the mug, it makes sense that every distinct volume of coffee that the cup could hold would generate it's own unique height and surface radius, however because of the nature of a cone's geometry, solving for any linear portion from a volume leads to a cubic function. Although the function only has a single meaningful root for our purposes, finding a 1 dimensional quantity, to be used as a linking point, from the surface area of the shape is also a quadratic. Then Patrick Cook heard of my struggles to find this equation, and over the weekend distracted himself from GRE prep with my puzzle, and solved it for me. The full solution is most easily represented in 3 equations, here with full credit and thanks to Patrick Cook, who answered my call: "Patrick, if you figure this out over the weekend don't tell me, I'm trying to work it out on my own"

$$
\begin{aligned}
h &= \frac{\mathbb{H}}{\frac{\mathbb{R}}{\mathbb{r}} - 1} \\
R &= \sqrt[3]{\frac{\mathbb{H}\mathbb{r}^3 - \mathbb{r}v + \mathbb{R}v}{\mathbb{H}}} \\
S_A &= \sqrt{\pi^2 R^4 + 9(\frac{v + \frac{\pi}{3}\mathbb{r}^2 h}{R})^2} \\
&\quad - \pi\mathbb{r}\sqrt{\mathbb{r}^2 + h^2} + \pi\mathbb{r}^2
\end{aligned} \tag{3}
$$

v–	Volume of liquid
h–	"Truncation height" of porion of cone below bottom of cup
$\mathbb{H}$–	Height of mug
$\mathbb{R}$–	Major internal radius of mug
R–	Radius of the surface of the liquid
$\mathbb{r}$–	Minor internal radius of mug
S_A–	Surface area of liquid in contact with mug

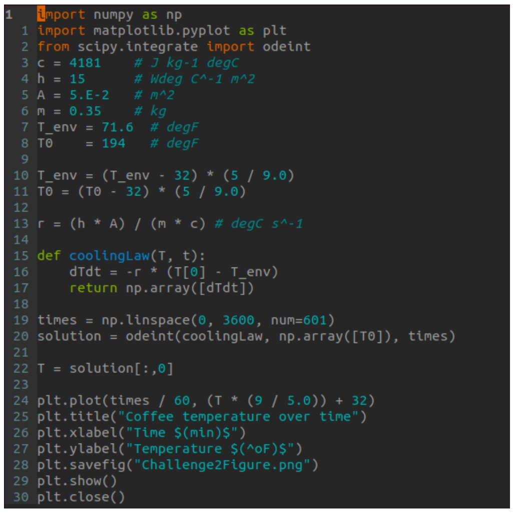
I calculated my r constants using Eq. 2 from the Coffee Cooling handout.

[1]Actually a right circular frustum of a cone

The calculation methods for my r constants involved pouring over multiple online lists of thermal properties in order to find a strong approximation of the material my mug is made of. Early contenders were glass and brick, but glass was diaqualified as from anecdotal evidence, grabbing a hot glass out of the microwave is harder than a hot mug. Brick was disqualified as the main source of coefficients for brick was from engineering websites listing the thermal properties of building materials and I had no way of being able to tell which of the wide range of bricks listed would be most accurate at modeling my coffee cup. Ultimately I ended with porcelain, which I determined to be the best approximation of the ceramic in my mug with no obvious way of determining the actual material of which the mug was made[2]. The h term in calculating each r coefficient proved to be the hardest to determine. For r_e, I used the r coefficient provided in the handout, the convective heat transfer coefficient between coffee (water) and air, 15 $\frac{W}{m^2K}$. For r_c I used the thermal conductivity of porcelain divided by the thickness of my cup, using the conductive heat loss of a pipe as an approximation of the heat loss of my cup due to the nearly identical geomtry involved [1]. Finally for r_{ce} I used the thermal conductivity of porcelain, multiplied by the surface area of the mug directly outside of the liquid (eliminating the contribution of portions of the mug above the coffee line) and dividing by the entire volume bound by this surface area, in order to reach units of $\frac{W}{m^2K}$. The r values I developed for this model are all very basic approximations, which reflects later in the accuracy of the model, but there is certain aspects that provide promise for more rigorous development of these constants.

[2]see possible future work

3 Code


```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
c = 4181      # J kg-1 degC
h = 15        # Wdeg C^-1 m^2
A = 5.E-2     # m^2
m = 0.35      # kg
T_env = 71.6  # degF
T0    = 194   # degF

T_env = (T_env - 32) * (5 / 9.0)
T0 = (T0 - 32) * (5 / 9.0)

r = (h * A) / (m * c) # degC s^-1

def coolingLaw(T, t):
    dTdt = -r * (T[0] - T_env)
    return np.array([dTdt])

times = np.linspace(0, 3600, num=601)
solution = odeint(coolingLaw, np.array([T0]), times)

T = solution[:,0]

plt.plot(times / 60, (T * (9 / 5.0)) + 32)
plt.title("Coffee temperature over time")
plt.xlabel("Time $(min)$")
plt.ylabel("Temperature $(^oF)$")
plt.savefig("Challenge2Figure.png")
plt.show()
plt.close()
```

Figure 1: My Beautiful Code!

4 Results

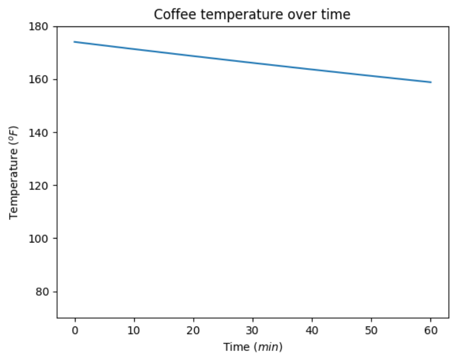


Figure 2: Plot generated from original coffee cooling challenge 2 code, with corrected mass and area

At this point in my work, it became clear that I had reached a natural stopping point, as I was already a week behind from the actual due date of the original coffee cooling problem, and I believe through anecdotal evidence no amount of curve fitting and pouring through lists of thermal coefficients of ceramics will ever get me a date.

5 Conclusion

We can tell that numerical methods to solve differentials are an extremely useful tool, because although the analytical solution to this specific function is fairly easily found, it does save us as programmers some work, as well as give us a tool to use in the future for more differentials. In further work, I plan to model thetemperatures of both the liquid inside the coffee and the cup in which it is held, as well as introducing ideas like variable volume of coffee.

6 Source Code

```
import numpy as np
from numpy import pi as PI
from numpy import e
from numpy.polynomial.polynomial import Pol
import matplotlib.pyplot as plt
from scipy.integrate import odeint
#
def newtCool(r, T_self, T_other):
    dTdt = r * (T_other - T_self)
    return dTdt

def dT_ldt(r_env, T_l, T_env, r_c, T_c):
    #        heat lost to environment
heat lost from cup
    dT_ldt = newtCool(r_env, T_l, T_env) +
    return dT_ldt

def dT_cdt(r_ce, T_l, T_env, r_c, T_c):
    #        heat gained from liquid
heat lost to sides
    dT_cdt = newtCool(r_c, T_c, T_l) + newt
    return dT_cdt

def cool(T_l, T_c, T_env, r_env, r_c, r_ce,
    liquidTemp  = T_l
    cupTemp     = T_c
    envTemp     = T_env
    liquidTempList, cupTempList, envTempList = [liquidTemp], [cupTemp],
    DeltaT = t_list[1] - t_list[0]
    for t in t_list:
        liquidTempChange
= dT_ldt(r_env, liquidTemp, envTemp, r_c, cupTemp)
        cupTempChange
= dT_cdt(r_ce, liquidTemp, envTemp, r_c, cupTemp)
        liquidTemp  += liquidTempChange * DeltaT
        cupTemp     += cupTempChange * DeltaT
        liquidTempList.append(liquidTemp)
        cupTempList.append(cupTemp)
        envTempList.append(envTemp)
    return (liquidTempList, cupTempList, envTempList)

def getR(cupProperties, Volume):
    H, r_a, r_b = cupProperties
    return (((H * r_a**3) - (r_a * Volume) + (r_b * Volume)) / H)**(1 /

def geth(cupProperties):
    H, r_a, r_b = cupProperties
    return H / ((r_b / r_a) - 1)

#Calculated by Patrick Cook
def SurfaceAreafromVolume(cupProperties, Volume):
    H, r_a, r_b = cupProperties
    R = getR(cupProperties, Volume)
    h = geth(cupProperties)
    return (PI**2 * R**4 + 9 * ((Volume + (PI / 3) * r_a**2 * h) / R)**2

def frustumVolume(h, r1, r2):
    return (1 / 3.0) * PI * h * (r1**2 + r2**2 + (r1 * r2))

def r(h, A, p, v, c):
    return (h * A) / (p * v * c)

def expFit(x, y):
    logFit = np.polyfit(np.array(x), np.log(np.array(y)), 1, w=np.sqrt(
    return np.array([e**logFit[1], logFit[0]])
```

```
#
                          # coffee cup parameters
max_l = 11.27             # cm
max_height = 11.34        # cm
rA = 3                    # cm
rA_ext = 3.3              # cm
rB = 3.86                 # cm
rB_ext = 4.2              # cm
thickness = (rB_ext - rB + rA_ext - rA) / 2

c_water = 4181            # J kg-1 degC
c_ceramic = 1085          # J kg-1 degC
h_water = 15              # Wdeg C^-1 m^-2
k_pork  = 1.5             # Wdeg C^-1 m^-2
p_water = 997             # kg / m^3
p_pork  = 2403            # kg / m^3

T_env = 23                # degC
T_cup = T_env             # degC
T_coffee = 74             # degC
#
max_l /= 100             # m
max_height /= 100        # m
rA /= 100                # m
rB /= 100                # m
rB_ext /= 100            # m
rA_ext /= 100            # m
thickness /= 100         # m

#calculating physical parameters of our cup
cup            = (max_height, rA, rB)
ext            = (max_height + thickness, rA_
vol_coffee     = 12 # oz
vol_coffee    *= 2.95735e-5 #oz to m^3 conver
vol_container= frustumVolume(vol_coffee / (
vol_cup        = vol_container - vol_coffee
m_coffee       = p_water * vol_coffee #kg
sa_coffee      = SurfaceAreafromVolume(cup, v
sa_ext          = SurfaceAreafromVolume(ext, vol_container) # m^2

# for the h value for r_env I used the average value of 15 W/m^2 K range
#r_env = r(h_water, PI * getR(cup, vol_coffee)**2, p_water, vol_coffee, c_water
r_env = r(5, PI * getR(cup, vol_coffee)**2, p_water, vol_coffee, c_water
# for the h value of r_cup I used it's thermal conductivity / the thickn
#r_cup = r(k_pork / thickness, sa_coffee, p_water, vol_coffee, c_water
r_cup = r(12.87, sa_coffee, p_water, vol_coffee, c_water
# for the h value of r_ce I multiplied it's linear thermal conductivity
#r_ce  = r(k_pork * sa_ext / vol_container, sa_ext, p_pork, vol_cup, c_pork
r_ce  = r(1.528, sa_ext, p_pork, vol_cup, c_pork)

print("h_water: %g" %(h_water))
print("h_pork: %g" %(k_pork / thickness))
print("h_pork to Env:  %g"
%(k_pork * sa_ext / vol_container))

print("r_env: %g" %(r_env))
print("r_cup: %g" %(r_cup))
print("r_ce:  %g"  %(r_ce))

print("r_env: %g" %(r_env))
print("r_cup: %g" %(r_cup))
print("r_ce:  %g"  %(r_ce))

print("Coffee Area: %g" %(PI * getR(cup, vol_coffee)**2))
print("Coffee mass: %g" %(vol_coffee * p_water))

times = np.arange(0, 3600, 0.1)

lTemps, cTemps, eTemps = cool(T_coffee, T_env, T_cup, r_env, r_cup, r_c
times = times / 60

experimentalX  = np.array([1, 2, 3, 4, 5, 6, 10, 15, 20, 25, 30, 35, 40
experimentalY  = np.array([165, 163, 161, 159, 156, 154, 146, 139, 132
```

```
minErr = 9e9
minErrnumi = -1
minErrnumj = -1
minErrnumk = -1
for i in np.arange(1, 1.5, 0.1): #water range
    for j in np.arange(-1, 1, 0.1): #coffee range
        for k in np.arange(-1, 1, 0.1): #pork
            [illegible]
            h_water = [illegible]
            r_env = r(i, PI * getR(cup, vol
            r_cup = r(j, sa_coffee, p_water
            r_ce = r(k, sa_ext, p_pork, vol_cup
            lTemps, cTemps, eTemps = cool(T
            lTemps = np.array(lTemps[0:-1])
            cTemps = np.array(cTemps[0:-1])
            eTemps = np.array(eTemps[0:-1])
            coeff
= expFit(times, lTemps)
            expectedY
= coeff[0] * e**(coeff[1] * experimentalX)
            chisq
= np.sum((experimentalY - expectedY)**2)
            if minErr > chisq:
                minErr = chisq
                minErrnumi = i
                minErrnumj = j
                minErrnumk = k
print("A minimum of %g occured at i =
print("bye bye")
exit()
lTemps = np.array(lTemps[0:-1])
cTemps = np.array(cTemps[0:-1])
eTemps = np.array(eTemps[0:-1])

coeff           = expFit(times, lTemps)
experimentalX   = np.array([1, 2, 3, 4, 5,
experimentalY   = np.array([165, 163, 161,
expectedY       = coeff[0] * e**(coeff[1] *
chisq             = np.sum((experimentalY - expectedY)**2 / expectedY)
print("Chi␣Squared:␣%g" %(chisq))
plt.plot(times, lTemps * (9 / 5.0) + 32)
plt.plot(times, cTemps * (9 / 5.0) + 32)
plt.plot(times, eTemps * (9 / 5.0) + 32)
plt.legend(("Liquid␣temperature", "Cup␣temperature", "Environment␣Tempe
plt.savefig("coolingfig2.png")
plt.xlabel("Time␣(min)")
plt.ylabel("Temperature␣($^o$F)")
plt.ylim(70, 180.01)
plt.title("Heat␣loss␣of␣coffee␣in␣a␣mug␣over␣time")
plt.show()
plt.close()
```

References

[1] Engineering ToolBox overall heat transfer coefficient. Accessed: 2019-09-23.