Robust open source high performance object orientated program to design chip cooling schemes

Sterling Reynolds

Abstract

Technological advancements in central processing units (CPU) and graphical processing units (GPU) has given an acceleration of engineering research capabilities with respect to numerical application like finite element analysis (FEA). CPU clock speed performance is fundamentally regulated based on temperature associated with the physical maximum safe temperature of the CPU. In this article, the author explores the relationship between diameter, length, and material of a fin system attached to a 12.7mm computer chip. A robust open source high performance object orientated program was developed to assist with determining the maximum rate of heat removal from the fins based upon variable parameters. Analyzing the three materials chosen (Brass, Copper, Bronze), Copper is the most optimum material to use for the fin system.

1 Introduction

In this article, the author develops and applies a robust program to compute the rate of heat removal from a computer chip to determine the optimum configuration with the assumptions made of a cooling fin system with respect to (w.r.t) geometry and material. The user specifies the thermal conductivity k and density ρ of the fin system in Si units ($W/m * K$ and kg/m^3 respectively) of each analysis. Multiprocessing capabilities are added to vary the geometry of multiple materials at once, saving the user design and engineering time.

1.1 Design specifications and standards

The 12.7mm chip analyzed in this article has a $N * N$ array of aligned pin fins that are insulated at the top wall to force airflow in the horizontal direction. The figure below shows a $4 * 4$ configuration although the value of N is arbitrary.

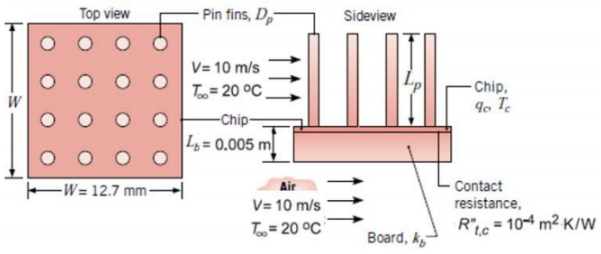

Figure 1: Schematic of $4 * 4$ cooling system

Thin chip is attached to a 5mm circuit board with a thermal contract resistance per unit area $R''_{t,c} = 10^{-4}\frac{m^2K}{W}$. The circuit broad had a thermal conductivity $K_b = 1\frac{W}{mk}$ with convective cooling at the bottom of velocity $V = 10\frac{m}{s}$ and temperature $T_{\infty,b} = 20°C$. Convective cooling velocity and temperature properties through the fin cooling system is the same as the board convection, although the system is varied by the number of pin fins N, diameter D_p, length L_p , and material choice. Maximum pin diameter is a function of the number of find N and a set variable of 9mm expressed in equation 1.

$$NL_p < 9mm \tag{1}$$

1.2 Realistic constraints and design variables

Assumptions with respect to (w.r.t) constraints is steady state flow velocity and temperature during the analysis and the board is flat with isothermal properties. These assumptions can be reasoned by a steady state condition where it is assumed the temperature of the chip will not see large gradients of temperature variations. Fin material is assumed to be perfectly homogeneous w.r.t thermal connectivity and density although in actuality this would not be true but variations are neglected. Table 1 shown below discusses design variables briefly mentioned in section 1.1 in more depth as user defined variables.

Variable	Description	Units	Program Location
N	Number of fins across one side	Unit less	Project_HT/input.py
ρ_f	Fin density	$\frac{Kg}{m^3}$	Project_HT/main.py
K_f	Fin thermal conductivity	$\frac{W}{mK}$	Project_HT/main.py
L_prange	Range of fin length to analyze	m	Project_HT/ analysis_class.py
D_prange	Range of fin diameter to analyze	m	Project_HT/ analysis_class.py

Table 1: User and design input variables for analysis

2 Methodology

Assuming steady-state one-dimensional heat transfer, the developed program will compute the rate of heat removal from the board at the first iteration to solve computational expense. Computation of the heat removal w.r.t the fin system will be computed at each combination of L_p, D_p, and material selection where the results will be stored within a private class variable for later analysis. Methodology section will present how the program works and the programming process w.r.t class objects.

2.1 Board heat removal rate

Following formal programming practices the user will be greeted with a main file to run the analysis [2]. Within the main file the input file will be referenced where the chip geometry properties are initialized and given back within a one-dimensional array. The input array will be used to initialize a objects associated with a private analysis class along with the chosen thermal conductivity and density of the material. This process is repeated for each of the materials w.r.t the main function file.

Once the class object is initialized, the program will call the main definition within the class where a user defined range of L_p and D_p is initialized. Since parameters for the base board of the design are not variable, the rate of heat removal of the base is computed once to save computational cost. This process of computations starts with computing the convective heat transfer coefficient h based upon the Reynolds number R_e and the local Nusselt number N_u where air properties are linearly interpolated from Table A.4 shown below in Figure 2 [1].

TABLE A.4 Thermophysical Properties of Gases at Atmospheric Pressure[a]

T (K)	ρ (kg/m^3)	c_p (kJ/kg · K)	$\mu \cdot 10^7$ (N · s/m^2)	$\nu \cdot 10^6$ (m^2/s)	$k \cdot 10^3$ (W/m · K)	$\alpha \cdot 10^6$ (m^2/s)	Pr
Air, $\mathcal{M}$ = 28.97 kg/kmol							
100	3.5562	1.032	71.1	2.00	9.34	2.54	0.786
150	2.3364	1.012	103.4	4.426	13.8	5.84	0.758
200	1.7458	1.007	132.5	7.590	18.1	10.3	0.737
250	1.3947	1.006	159.6	11.44	22.3	15.9	0.720
300	1.1614	1.007	184.6	15.89	26.3	22.5	0.707
350	0.9950	1.009	208.2	20.92	30.0	29.9	0.700
400	0.8711	1.014	230.1	26.41	33.8	38.3	0.690
450	0.7740	1.021	250.7	32.39	37.3	47.2	0.686
500	0.6964	1.030	270.1	38.79	40.7	56.7	0.684
550	0.6329	1.040	288.4	45.57	43.9	66.7	0.683
600	0.5804	1.051	305.8	52.69	46.9	76.9	0.685
650	0.5356	1.063	322.5	60.21	49.7	87.3	0.690
700	0.4975	1.075	338.8	68.10	52.4	98.0	0.695
750	0.4643	1.087	354.6	76.37	54.9	109	0.702
800	0.4354	1.099	369.8	84.93	57.3	120	0.709
850	0.4097	1.110	384.3	93.80	59.6	131	0.716
900	0.3868	1.121	398.1	102.9	62.0	143	0.720
950	0.3666	1.131	411.3	112.2	64.3	155	0.723
1000	0.3482	1.141	424.4	121.9	66.7	168	0.726
1100	0.3166	1.159	449.0	141.8	71.5	195	0.728
1200	0.2902	1.175	473.0	162.9	76.3	224	0.728
1300	0.2679	1.189	496.0	185.1	82	257	0.719
1400	0.2488	1.207	530	213	91	303	0.703
1500	0.2322	1.230	557	240	100	350	0.685
1600	0.2177	1.248	584	268	106	390	0.688
1700	0.2049	1.267	611	298	113	435	0.685
1800	0.1935	1.286	637	329	120	482	0.683
1900	0.1833	1.307	663	362	128	534	0.677
2000	0.1741	1.337	689	396	137	589	0.672
2100	0.1658	1.372	715	431	147	646	0.667
2200	0.1582	1.417	740	468	160	714	0.655
2300	0.1513	1.478	766	506	175	783	0.647
2400	0.1448	1.558	792	547	196	869	0.630
2500	0.1389	1.665	818	589	222	960	0.613
3000	0.1135	2.726	955	841	486	1570	0.536

Figure 2: Table A.4 **REFERENCE**

Reynolds number is determined using,

$$R_e = \frac{pU_\infty x}{\mu} \tag{2}$$

where p, U_∞, x, μ represent density, flow velocity, location w.r.t x direction, and viscosity respectively. The script will check for the user if the flow is laminar using the flat plate criteria of Reynolds number being less than $5 * 10^5$. The Nusselt number is computed with,

$$N_u = 0.332 R_e^{\frac{1}{2}} P_r^{\frac{1}{3}} \tag{3}$$

with P_r representing the interpolated Prandtl number from Figure 2. Convective heat transfer coefficient h is determined by,

$$h_b = N_u \frac{k_b}{x} \tag{4}$$

where k_b is the interpolated thermal conductivity from Figure 2. These variables will be stored associated with the class object, where the variables will be implemented in computing the board heat rate where a thermal circuit is implemented described in Figure 3.

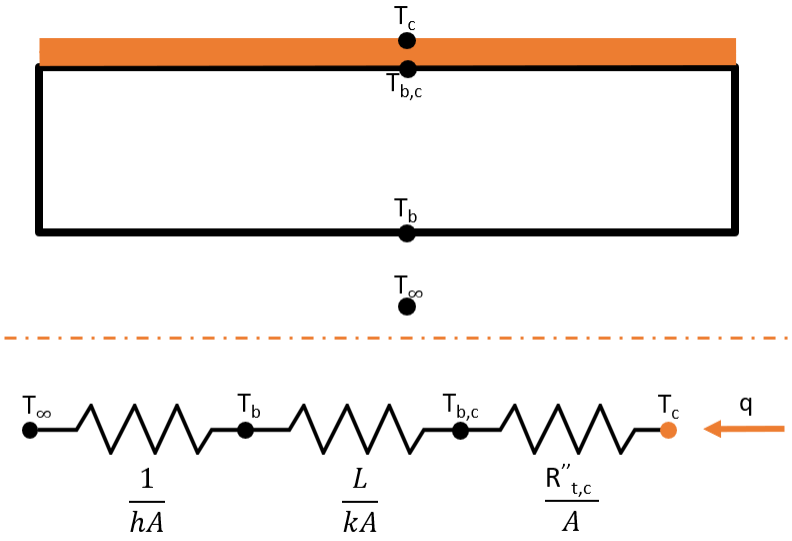

Figure 3: Board thermal circuit used to compute the heat rate

Thermal circuit resistances can be rearranged and solving for heat rate,

$$q_b = \frac{A_c(T_c - T_\infty)}{R'' + \frac{L_b}{K_b} + \frac{1}{h_b}} \tag{5}$$

where A_c is the cross sectional area of the chip.

2.2 Fin heat removal rate

Computation of the fin heat removal rate starts with interpolating the air tables shown in Figure 2 based on the T_∞ where density, specific heat, viscosity, kinematic viscosity, thermal conductivity, and Prandtl number are obtained. Figure 4 displays a top view of the chip where the spacing variables $S_L$and S_T computed in this example analysis by

$$S_T = \frac{1}{N+1} w \tag{6}$$

where w is the chip width.

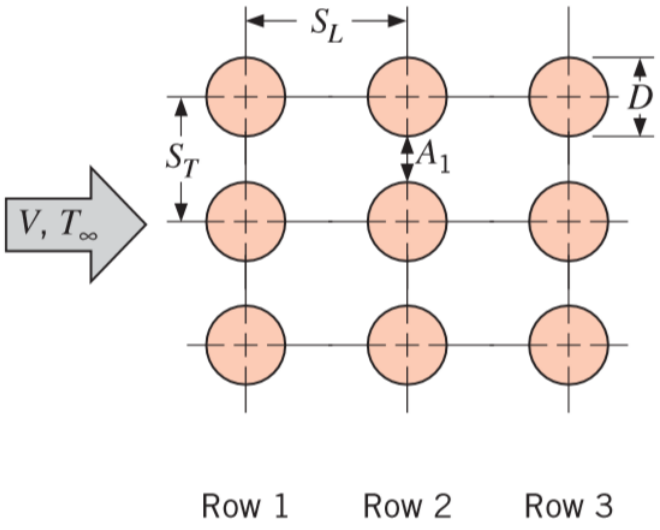

Figure 4: Tube arrangements in a bank [1]

Computing the average Nusselt number required the maximum Reynolds number which is a function of maximum velocity V_{max} described by,

$$V_{max} = \frac{S_T}{S_t - D_p} U_\infty \tag{7}$$

and the Reynolds number is defined as,

$$R_{e,max} = \frac{U_\infty D_p}{\nu} \tag{8}$$

where ν is kinematic viscosity. Based on the Reynolds number and N the Nusselt constants for the analysis the program will read Table 7.5 [1], if N is less than 20 a secondary constant is interpolated using Table 7.6 [1] both are displayed in Figure 5.

TABLE 7.5 Constants of Equation 7.58 for the tube bank in cross flow [17]

Configuration	$Re_{D,max}$	C_1	m
Aligned	$10–10^2$	0.80	0.40
Staggered	$10–10^2$	0.90	0.40
Aligned	$10^2–10^3$	Approximate as a single (isolated) cylinder	
Staggered	$10^2–10^3$		
Aligned $(S_T/S_L > 0.7)^a$	$10^3–2 \times 10^5$	0.27	0.63
Staggered $(S_T/S_L < 2)$	$10^3–2 \times 10^5$	$0.35(S_T/S_L)^{1/5}$	0.60
Staggered $(S_T/S_L > 2)$	$10^3–2 \times 10^5$	0.40	0.60
Aligned	$2 \times 10^5–2 \times 10^6$	0.021	0.84
Staggered	$2 \times 10^5–2 \times 10^6$	0.022	0.84

[a]For $S_T/S_L < 0.7$, heat transfer is inefficient and aligned tubes should not be used.

TABLE 7.6 Correction factor C_2 of Equation 7.59 for $N_L < 20$ $(Re_{D,max} \gtrsim 10^3)$ [17]

N_L	1	2	3	4	5	7	10	13	16
Aligned	0.70	0.80	0.86	0.90	0.92	0.95	0.97	0.98	0.99
Staggered	0.64	0.76	0.84	0.89	0.92	0.95	0.97	0.98	0.99

Figure 5: Table 7.5 and 7.6 [1]

Average Nusselt number across the array of pins is described by,

$$[N < 20] \qquad N_u = C2 * C1 * \bar{R}^m_{e,max} * P_r^{0.36}(\frac{P_r}{P_{r_s}})^{\frac{1}{4}} \tag{9}$$

$$[N \geqslant 20] \qquad N_u = C1 * \bar{R}^m_{e,max} * P_r^{0.36}(\frac{P_r}{P_{r_s}})^{\frac{1}{4}} \tag{10}$$

The average convective heat transfer coefficient is computed with equation 4, with the average Nusselt number computed either with equation 9 or 10 based upon the configuration. Using the formulation of a straight fin with uniform cross-section and adiabatic insulated tip,

$$n_o = 1 - \frac{N_t A_f}{A_t}(1 - n_f) \tag{11}$$

$$n_f = \frac{\tanh(mL_p)}{mL_p} \tag{12}$$

$$m = \sqrt{\frac{4h}{kD_p}} \tag{13}$$

where,

$$A_f = \pi D_p L_c \tag{14}$$

$$A_t = (N_t A_f) + A_b \tag{15}$$

$$A_b = W^2 - N_t A_c \tag{16}$$

is used to compute the heat rate. Using the same formulation of the thermal circuit shown in figure 3, the heat rate removal from the fin system is formulated as,

$$q_f = \frac{T_c - T_\infty}{R_f} \tag{17}$$

$$R_f = \frac{1}{n_0 \overline{h} A_t} \tag{18}$$

where T_∞ is the entrance temperature to the fin system. The total heat removal rate is described by,

$$q_c = q_b + q_f \tag{19}$$

which is stored within a python list along with diameter and length for each analysis.

2.3 Programming methods

Using object orientated programming and a unix/unix-like system offers great benefits given the programs repetitive characteristic, where the same computation is being performed with changing parameters [2]. Using private class objects given to the user specified material, one could use multiprocessing to compute each material at the same time. A trivial implementation of python multiprocessing is setup using the built in module where a process will be defined for each material. Once the process is started within shared memory, a CPU will pick up the job and execute the commands; once completed the results will be gained from it's object [1]. Figure 6 shows a global flowchart neglecting the details on object orientated class structure details.

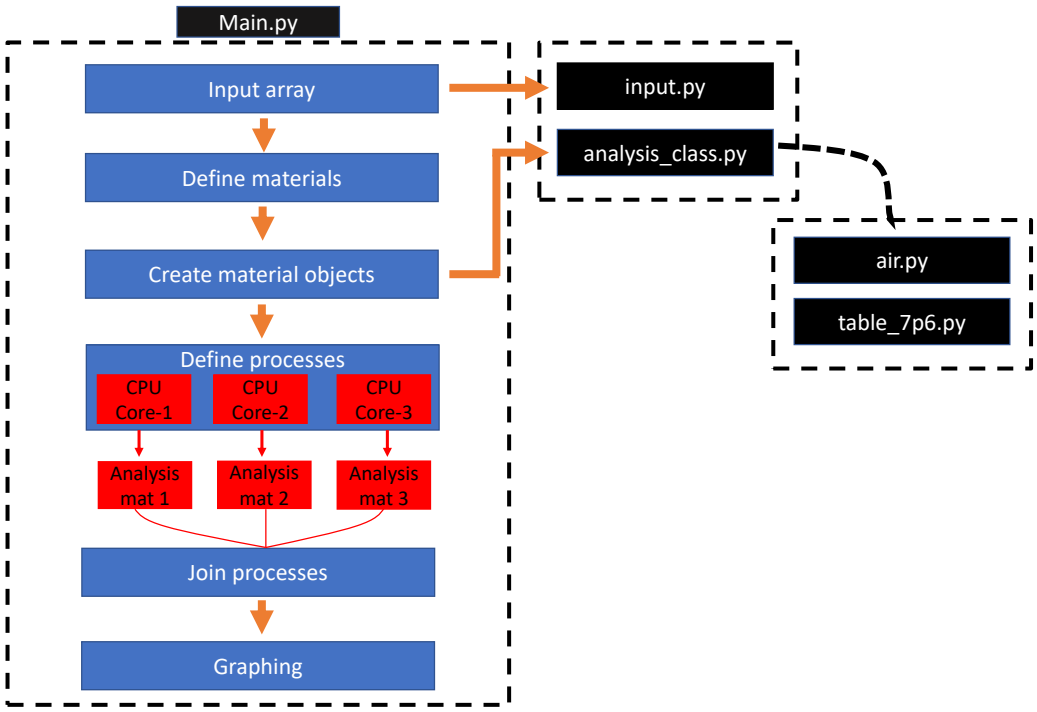

Figure 6: Global multiprocessing flowchart

3 Results

Three materials were chosen to demonstrate in the results section, although the code is robust to compute any material with a known thermal conductivity coefficient. These selected materials are Brass, Copper, and Bronze to demonstrate the usefulness of the program and performance analysis on realistic materials. Results indicate a large fluctuation in the total heat remove rate w.r.t the length of the fin system L_p,although the diameter D_p is also import. Figure 7 shows the heat removal rate in 3 dimensions where the D_p, L_p and q_c represent x,y,z directions respectability.

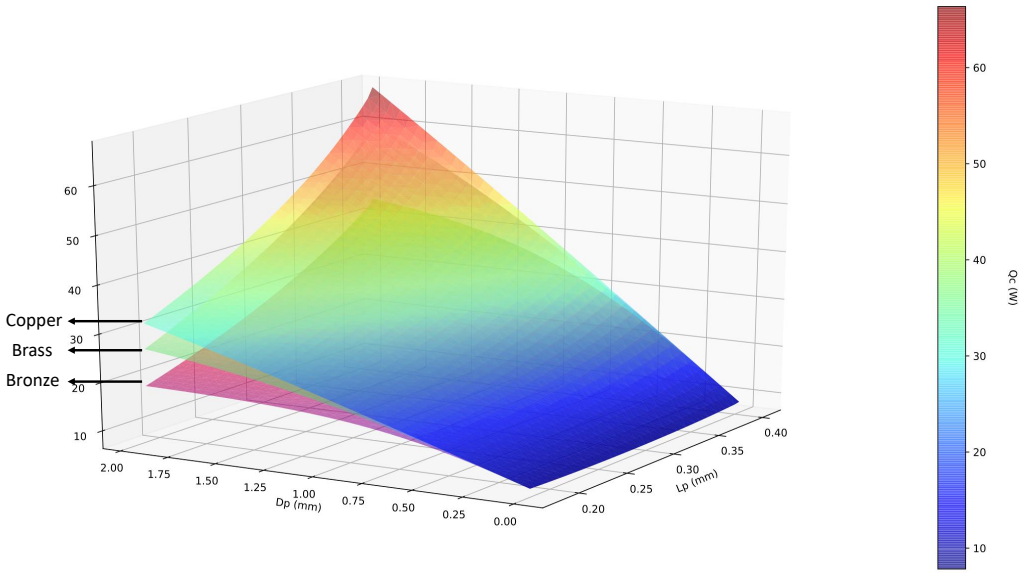

Figure 7: Heat removal rate q_c

Where the thermal conductivity coefficient of each material significantly changes the heat removal rate, although each material follows the same trend. Figure 8 shows a two dimensional plot of the heat removal rate vs the length of fin for the largest allowable D_p.

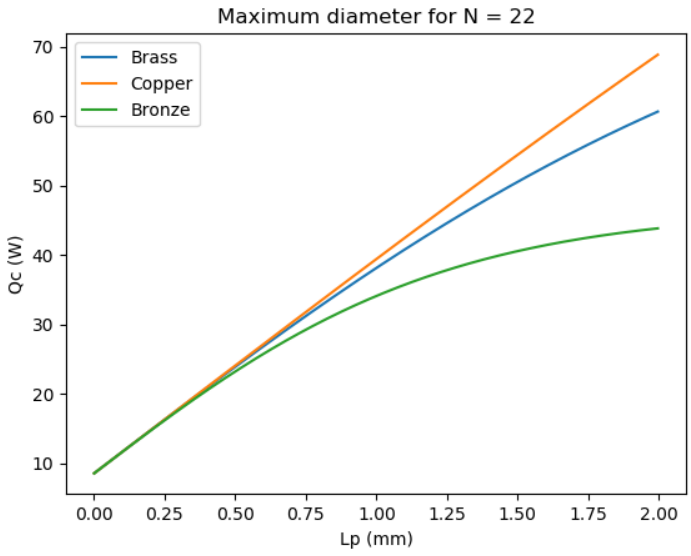

Figure 8: Largest diameter $q_c(w)$

As shown in the two dimensional graph, Copper has a more linear response w.r.t Bronze and Brass and could be a optimum choice between material selection. Other design constraints have to be the mass of the system which is a function of the materials density and configuration of geometry. The mass of each fin system configuration is represented in a three dimensional graph, like seen in Figure 7.

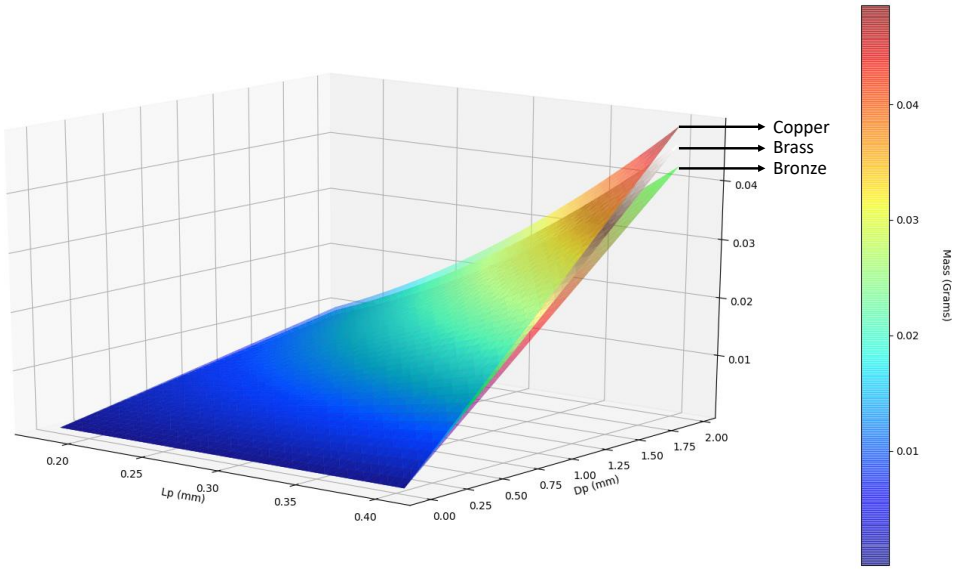

Figure 9: Mass of each configuration

4 Conclusion

A open source high performance object orientated program was developed in a robust way to compute the heat transfer removal rate of a cooling scheme applied to a computer chip. Users have full control on material and geometry is used during the analysis by modifying the simple main and input files respectability. Each material can be computed using it's own CPU, using the multiprocessing scheme shown in section 2.3 reducing computational time. Viewing the data gained from the three materials, Copper seems to be the best choice with the $D_p$being equal to the maximum allowable diameter and L_p equal to 2mm, given the mass does not differ enough to make significantly design alterations.

References

[1] Bergman, T. L., et al. Fundamentals of Heat and Mass Transfer. John Wiley & Sons, Inc., 2019.

[2] Knuth, Donald Ervin. The Art of Computer Programming. Addison-Wesley Publ., 2005.

5 Code

5.1 Main.py

In [ ]:
```
#*********************************************************************
# NAME OF THE PROJECT:  main.py
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PURPOSE OF THIS MODULE :
#                          Main input script to execute the heat trans
#                          project.
#
# REQIREMENTS :            (Linux/Mac/Windows) Python 3.x
#
# Developer: Sterling Reynolds , Undergraduate student
#            Contact: icu327@my.utsa.edu
#            Department of Mechanical Engineering, UTSA Texas
#
# License:  MIT
#           If you're using this for academic work, a donation of coff
#           developer would be much appreciated.
# DATE:     March 2020 (SR)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import numpy as np
import matplotlib.pyplot as plt
from multiprocessing import Process
import pandas as pd
from Project_HT.analysis_class import *
from Project_HT.input  import *
```

In [ ]:
```
def main():
    #*****************************************************************
    #                          Insulated
    #      _     _      _      _      _      _      _
    #       |    | |    | |    | |    | |    | |    | |    / ______ Vf
    #       |    | |    | |    | |    | |    | |    | |    \
    #    Lp |    | |    | |    | |    | |    | |    | |    / ______ Tf_in
    #       |    | |    | |    | |    | |    | |    | |    \
    #       |    | |    | |    | |    | |    | |    | |
    #     -   *-------------------------------------*
    #         |/////////////////////////////////////|   R_pp = 10^-4 (
    #     -   *-------------------------------------*
    #         |                                     |
    #  0.005m |                                     |
    #         |                                     |
    #     -   *-------------------------------------*
    #                                                 / ______ Vb
    #                                                 \
    #                                                 / ______ Tb_in
    #                                                 \
    #                          12.7mm
    #         |~~~~~~~~~~~~~~~~~~ W ~~~~~~~~~~~~~~~~~~|
```

```
#*****************************************************************

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Start: Retrieve properties from input file
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
init_prop = init_properties()
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Start:  Retrieve properties from input file
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Brass ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thermal_conductivity  = 64  # W/(m*k)
density               = 8400  # (Kg/m^3)
Brass_analysis= analysis(init_prop,thermal_conductivity,density)
# Brass ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Copper ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thermal_conductivity  = 413  # W/(m*k)
density               = 8940 # (Kg/m^3)
CU_analysis= analysis(init_prop,thermal_conductivity,density)
# Copper ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Bronze ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thermal_conductivity  = 15  # W/(m*k)
density               = 7700  # (Kg/m^3)
Bronze_analysis= analysis(init_prop,thermal_conductivity,density)
# Bronze ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


# Start analysis ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
p_Brass = Process(target=Brass_analysis.main()  , args=())
p_CU = Process(target=CU_analysis.main()  , args=())
p_Bronze = Process(target= Bronze_analysis.main(), args=())
p_Brass.start()
p_CU.start()
p_Bronze.start()
p_Brass.join()
p_CU.join()
p_Bronze .join()
# End of analysis ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Get data from objects
results_Brass  =  np.concatenate(Brass_analysis.results   )
results_CU     =  np.concatenate(CU_analysis.results   )
results_Bronze =  np.concatenate(Bronze_analysis.results)

fig    = plt.figure(1)
ax1     = fig.add_subplot(111, projection='3d')
#surf   = ax1.plot_trisurf(results_Brass[:,1], results_Brass[:,2],
#surf11 = ax1.plot_trisurf(results_CU[:,1], results_CU[:,2], resul
#surf1  = ax1.plot_trisurf(results_Bronze[:,1], results_Bronze[:,2
```

```
    #surf1  = ax1.plot_trisurf(results_Bronze[:,1], results_Bronze[:,2
    #cbar = fig.colorbar(surf11)
    #cbar.set_label('Qc (W)',labelpad=30, rotation=270)

    surf   = ax1.plot_trisurf(results_Brass[:,1], results_Brass[:,2]
    surf11 = ax1.plot_trisurf(results_CU[:,1], results_CU[:,2]
    surf1  = ax1.plot_trisurf(results_Bronze[:,1], results_Bronze[:,2]
    cbar = fig.colorbar(surf11)
    cbar.set_label('Mass (Grams)',labelpad=30, rotation=270)

    plt.xlabel('Lp (mm)')
    plt.ylabel('Dp (mm)')

    #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    two_d_results_Brass  =  np.concatenate( Brass_analysis.last_res
    two_d_results_CU     =  np.concatenate(    CU_analysis.last_res
    two_d_results_Bronze =  np.concatenate(Bronze_analysis.last_res

    fig1 = plt.figure(3)
    plt.plot(two_d_results_Brass[:,2], two_d_results_Brass[:,0]  ,labe
    plt.plot(two_d_results_CU[:,2], two_d_results_CU[:,0]        ,labe
    plt.plot(two_d_results_Bronze[:,2], two_d_results_Bronze[:,0],labe
    plt.legend()
    plt.xlabel('Lp (mm)')
    plt.ylabel('Qc (W)')
    title = 'Maximum diameter for N = ' + str(init_prop[0])
    plt.title(title)
    plt.show()




    df_results_Brass  = pd.DataFrame(results_Brass )
    df_results_CU     = pd.DataFrame(results_CU    )
    df_results_Bronze = pd.DataFrame(results_Bronze)
    df_results_Brass.to_excel('Excel/Brass.xlsx')
    df_results_CU.to_excel('Excel/Copper.xlsx')
    df_results_Bronze.to_excel('Excel/Bronze.xlsx')
```

In [ ]:
```
if __name__ == "__main__":
    main()
```

In [ ]:
```

```

5.2 Input.py

In [ ]:
```
#*********************************************************************
# NAME OF THE PROJECT:  input.py
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PURPOSE OF THIS MODULE :
#                          Input file for heat transfer project
#                          : Initialize analysis properties
#
# REQIREMENTS :            (Linux/Mac/Windows) Python 3.x
#
# Developer: Sterling Reynolds , Undergraduate student
#            Contact: icu327@my.utsa.edu
#            Department of Mechanical Engineering, UTSA Texas
#
# DATE:     March 2020 (SR)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import numpy as np
```

In [ ]:
```
def init_properties():
    #*********************************************************************
    #                          Insulated
    #      _     _     _     _     _     _     _
    #       | |   | |   | |   | |   | |   | |   | |    / _______ Vf
    #       | |   | |   | |   | |   | |   | |   | |    \
    #    Lp | |   | |   | |   | |   | |   | |   | |    / _______ Tf_in
    #       | |   | |   | |   | |   | |   | |   | |    \
    #       | |   | |   | |   | |   | |   | |
    #     _   *-------------------------------------*
    #         |/////////////////////////////////////|   R_pp = 10^-4 (
    #     _   *-------------------------------------*
    #         |                                     |
    #  0.005m |                                     |
    #         |                                     |
    #     _   *-------------------------------------*
    #                                               / _______ Vb
    #                                               \
    #                                               / _______ Tb_in
    #                                               \
    #                          12.7mm
    #         |~~~~~~~~~~~~~~~~~ W ~~~~~~~~~~~~~~~~~|
    #*********************************************************************

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Start: Define parameters for analysis
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Geometry
    N        = 22     # number of fins (Reynolds)
    W        = 0.0127  # (m) width of base
```

```
    Rpp_chip = 10**-4  # ((m^2K / W)) thermal contact resistance per u
    Lb       = 0.005   # (m) Height of base
    Kb       = 1       # (W/m.K) Thermal conductivity

    # Velocity and temperature of convection cooling
    Vf      = 10              # (m/s) Velocity over fins
    Tf_inf  = 20 + 273.15     # (K)   T infinity over fins
    Vb      = 10              # (m/s) Velocity at base
    Tb_inf  = 20 + 273.15     # (K)   T infinity at base


    # Temperature of chip
    Tc = 75 + 273.15          # (K)


    # Varied parameters
    # Dp = Pin diameter
    # Lp = Length of pin
    #
    # Number of pins multiplied by the diameter of the pins cannot exc
    # Thus: (N)Dp = 9mm
    Dp_max = (0.009) / (N)  # (m Max)
    Lp = None           # (m) Unknown parameters
    Qc = None           # (W) Unknown parameters
    Qb = None           # (W) Unknown parameters
    Qf = None           # (W) Unknown parameters
    hb = None           # (W/m*2*K) Unknown parameters

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # End: Define parameters for analysis
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    init_prop = np.array((N, W, Rpp_chip, Lb, Kb, Vf, Tf_inf, Vb, Tb_i

    # return initialized analysis properties main function
    return(init_prop)
```

5.3 analysis_class.py

In [ ]:
```
#*****************************************************************
# NAME OF THE PROJECT: analysis_class.py
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PURPOSE OF THIS MODULE :
#                          Private class to hold parameters for projec
#
#
# REQIREMENTS :            (Linux/Mac/Windows) Python 3.x
#
# Developer: Sterling Reynolds , Undergraduate student
#            Contact: icu327@my.utsa.edu
#            Department of Mechanical Engineering, UTSA Texas
#
# DATE:     March 2020 (SR)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import sys
import math
from numba import jit
import time
from Project_HT.Analysis_properties.air import *
from Project_HT.Analysis_properties.table_7p6 import *
```

In [ ]:
```
class analysis():

        def __init__(self, init_prop,Kf,pf):

            self.__N         = init_prop[0]   # number of fins (Reynol
            self.__W         = init_prop[1]   # (m) width of base
            self.__Rpp_chip  = init_prop[2]   # ((m^2K / W)) thermal c
            self.__Lb        = init_prop[3]   # (m) Height of base
            self.__Kb        = init_prop[4]   # (W/m.K) Thermal conduc
            self.__Vf        = init_prop[5]   # (m/s) Velocity over fi
            self.__Tf_inf    = init_prop[6]   # (K)   T infinity over
            self.__Vb        = init_prop[7]   # (m/s) Velocity at base
            self.__Tb_inf    = init_prop[8]   # (K)   T infinity at ba
            self.__Tc        = init_prop[9]   # (K) Temperature of chi
            self.__Dp_max    = init_prop[10]  # (m) Max diameter of fi
            self.__Dp        = 0              # init value
            self.__Lp        = init_prop[11]  # (m) Unknown parameters
            self.__Qc        = init_prop[12]  # (W) Unknown parameters
            self.__Qb        = init_prop[13]  # (W) Unknown parameters
            self.__Qf        = init_prop[14]  # (W) Unknown parameters
            self.__hb        = init_prop[15]  # (W/m*2*K) Unknown para
```

```
        self.__Kf        = Kf            # (W/m.K)  Thermal condu
        self.__pf        = pf            # (Kg/m^3) Density
        self.__results   = []            # Empty list to hold res
        self.last_res    = []            # for 2d plot

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Purpose of function: run analysis within class
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def main(self):

        self.base_conv_heat_transfer_coeff()
        self.base_heat_rate()

        # Optimize parameters via gradient decent to find local mi
        Dp_range = np.arange((0.009/50) , self.__Dp_max, (5*10**-6
        Lp_range = np.arange((0.001 * 10**-3) , (2* 10**-3), (5*10

        for i in range(len(Dp_range)):
            self.__Dp = Dp_range[i]
            for x in range(len(Lp_range)):
                self.__Lp = Lp_range[x]

                self.fin_heat_rate()

                mass = ((self.__Lp * ((math.pi /4) * self.__Dp**2)

                self.__results.append([(self.__Qc[0],self.__Dp * 1
                if i == len(Dp_range)-1:
                    self.last_res.append([(self.__Qc[0],self.__Dp


    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Purpose: Compute convection heat transfer coefficent at the
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def base_conv_heat_transfer_coeff(self):
        # Base convection heat transfer coefficent at base
        #  Assumptions: Steady state, isothermal, flat plate
        #
        #         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        #  0.005m |                                     |
        #         |                                     |
        #     -   *-------------------------------------*
        #                                                / ____
        #                                                \
        #                                                / ____
        #                                                \
        # Interpolate air tables for Density and Viscosity based o
        air_tablea_4 = air_a4()
        density_base   = np.interp([self.__Tb_inf], air_tablea_4[:
        viscosity_base = np.interp([self.__Tb_inf], air_tablea_4[:
```

```
        viscosity_base = np.interp([self.__Tb_inf], air_tablea_4[:
        K_base         = np.interp([self.__Tb_inf], air_tablea_4[:
        Prandtl_base   = np.interp([self.__Tb_inf], air_tablea_4[:

        # Compute Reynolds number ******************************
        # Re = (VpX / U)
        Re_base = (self.__Vb * density_base * self.__W) / (viscosi
        if Re_base >= (5*10**5):
            print('\n\nTurbulent flow Qf is not programmed: you ca
            print('Breaking program: please reduce parameters to e
            print('Bye')
            sys.exit()

        # Compute Nusselt number *******************************
        # Local_Nu_base = 0.332 * Re^(0.5) * Pr^(1/3)
        local_Nu_base = 0.332 * (Re_base**(1/2)) * (Prandtl_base**

        # Compute convection heat transfer coefficent at base
        # h = Nu * (k/x)
        self.__hb = local_Nu_base * (K_base / self.__W)

        return()


    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Purpose of function: Compute Qc of the chip
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def base_heat_rate(self):
        # Thermal circuit of the base:
        #
        #   Tb_inf           Tb              Tb1               Tc
        #    *---\/\/\/\/\/---*---\/\/\/\/\/---*---\/\/\/\/\/---*
        #          RConv            RCond1             Cond
        #          1/hb              Lb/K              Rpp
        #
        # Thus Qpp = (Tc - Tb_inf) / (Rconv + Rcond + Rcond)
        #      Converting to Qc
        #      Qc = Ac(Tc - Tb_inf) / (Rconv + Rcond + Rcond)
        #           where Ac = W**2

        Ac = self.__W**2

        RConv = (1/self.__hb)
        RCond = (self.__Lb / self.__Kb)

        self.__Qb = (Ac * (self.__Tc - self.__Tb_inf) ) / ((self._


    def fin_heat_rate(self):
        #**************************************************
        #                        Insulated
```

```
#     _     _     _     _     _     _     _
#          | |   | |   | |   | |   | |   | |    /
#          | |   | |   | |   | |   | |   | |    \
#    Lp    | |   | |   | |   | |   | |   | |    /
#          | |   | |   | |   | |   | |   | |    \
#          | |   | |   | |   | |   | |   | |
#     -   *-------------------------------------*
#                Chip and board below
#*****************************************************
# Computing Qf of the fin configuration

# Compute Reynolds Number for Flow
air_tablea_4    = air_a4()
density_fin     = np.interp([self.__Tf_inf], air_table
Cp_fin          = np.interp([self.__Tf_inf], air_table
viscosity_fin   = np.interp([self.__Tf_inf], air_table
K_fin           = np.interp([self.__Tf_inf], air_table
Pr_fin_air      = np.interp([self.__Tf_inf], air_table
Pr_fin_s        = np.interp([self.__Tc]    , air_table


Lc = self.__Lp + (self.__Dp / 4 )
Af = math.pi * self.__Dp * Lc
Ac = (math.pi * (self.__Dp ** 2)) / 4
Ab = (self.__W**2) - ((self.__N**2) * Ac)
At = ((self.__N**2) * Af) + Ab



# Compute St, SL, and max velocity ******************
St = (1/ (self.__N + 1)) * self.__W
SL = St
if St <= self.__Dp:
    print('This config is not logical, diameter is lar
    sys.exit()

Vmax = (St / (St - self.__Dp)) * self.__Vf

# Compute Reynolds number ***************************
# Re = (VpD / U)
Re_fin_max = (Vmax * density_fin * self.__Dp) / (visco

# Determine Constants of Equation 7.58 for the tube ba
# These if statments are not good although I would lik
# Thus, don't look at them and you'll be happier :)
if (Re_fin_max >= 10 and Re_fin_max < 100):
    C1 = 0.80
    m = 0.40
elif (Re_fin_max >= 100 and Re_fin_max < 10**3):
    C1 = 0.51
```

```
    m = 0.5
elif (Re_fin_max >= 10**3 and Re_fin_max < 2*10**5):
    C1 = 0.21
    m = 0.63
elif (Re_fin_max >= 2*10**5 and Re_fin_max < 2*10**6):
    C1 = 0.021
    m = 0.84
else:
    print('Your Reynolds number is pretty big')
    print('Please check your parameters')
    print('Breaking the program, bye')
    sys.exit()


# Compute Nusselt number ****************************
if self.__N < 20:
    Correction_factor_C2_table = table_7p6_C2()
    C2   = np.interp([self.__N], Correction_factor_C2_
    Nu_bar_fin = C2 * C1 * (Re_fin_max**m) * (Pr_fin_a
else:
    Nu_bar_fin = C1 * (Re_fin_max**m) * (Pr_fin_air**0

# Compute average heat transfer coeff ***************
#h_bar_fin = Nu_bar_fin * (self.__Kf / self.__Dp)
h_bar_fin = Nu_bar_fin * (K_fin / self.__Dp)

# Compute parameters for n_o ************************
# Lc = L + (D/4)
# Af = πD(Lc)
# Ac = (πD^2) /4
# Ab = (W^2) - (N^2 * Ac)
# At = ( N^2 * Af) + Ab
#
# m   = sqrt((hbar*p / k*Ac))
# nf  = tanh(m*Lc) / (m*Lc)
# n_o = (1 - (N*Af / At))(1-nf)

#m = math.sqrt( (h_bar_fin * self.__pf) / (self.__Kf *
m = math.sqrt( ( 4 * h_bar_fin) / (self.__Kf * self.__



# Check if ml is larger than 2.65 pg 163 in textbook
if self.__Lp > (2.65 / m):
    print('Max length is reached, returning to main')
    return()

nf = math.tanh(m * self.__Lp) /( m * self.__Lp)
ex = ((self.__N**2 * Af)/(At))*(1 - nf)
n_o = (1 - ex)
```

```
            res = (n_o * h_bar_fin * At)**-1

            self.__Qf = (self.__Tc - self.__Tf_inf) / res
            self.__Qc = self.__Qf + self.__Qb
            return()


    @property
    def N(self):
        return(self.__N)
    @property
    def W(self):
        return(self.__W)
    @property
    def Rpp_chip(self):
        return(self.__Rpp_chip)
    @property
    def Lb(self):
        return(self.__Lb)
    @property
    def Kb(self):
        return(self.__Kb)
    @property
    def Vf(self):
        return(self.__Vf)
    @property
    def Tf_inf(self):
        return(self.__Tf_inf)
    @property
    def Vb(self):
        return(self.__Vb)
    @property
    def Tb_inf(self):
        return(self.__Tb_inf)
    @property
    def Tc(self):
        return(self.__Tc)
    @property
    def Dp_max(self):
        return(self.__Dp_max)
    @property
    def Dp(self):
        return(self.__Dp)
    @property
    def Lp(self):
        return(self.__Lp)
    @property
    def Qc(self):
        return(self.__Qc)
```

5.4 air.py

In [ ]:
```
#*********************************************************************
# NAME OF THE PROJECT:  air.py
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PURPOSE OF THIS MODULE :
#
#                         Hold air table
#
# REQIREMENTS :           (Linux/Mac/Windows) Python 3.x
#
# Developer: Sterling Reynolds , Undergraduate student
#            Contact: icu327@my.utsa.edu
#            Department of Mechanical Engineering, UTSA Texas
#
# DATE:     March 2020 (SR)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import numpy as np
```

In [ ]:
```
def air_a4():
    #*********************************************************************
    #    Table A4 from "Fundamentals of Heat and Mass Transfer 8th ED"
    #               ( Photo of table provided in same directory :) )
    #    Column 0 = Temperature          (K)
    #    Column 1 = Density              (Kg/m^3)
    #    Column 2 = Specific heat        (kJ/kg * K)
    #    Column 3 = Viscosity            (N · s/m2)
    #    Column 4 = Kinematic viscosity (m^2 / s)
    #    Column 5 = Thermal Conductivity (W/m · K)
    #    Column 6 = Alpha                (m^2/s)
    #    Column 7 = Prandtl Number
    #*********************************************************************

    air_tablea_4 = np.array([ [100 ,  3.5562,    1.032,    71.1 ,   2.
                              [150 ,  2.3364,    1.012,    103.4,   4.
                              [200 ,  1.7458,    1.007,    132.5,   7.
                              [250 ,  1.3947,    1.006,    159.6,   11
                              [300 ,  1.1614,    1.007,    184.6,   15
                              [350 ,  0.9950,    1.009,    208.2,   20
                              [400 ,  0.8711,    1.014,    230.1,   26
                              [450 ,  0.7740,    1.021,    250.7,   32
                              [500 ,  0.6964,    1.030,    270.1,   38
                              [550 ,  0.6329,    1.040,    288.4,   45
                              [600 ,  0.5804,    1.051,    305.8,   52
                              [650 ,  0.5356,    1.063,    322.5,   60
                              [700 ,  0.4975,    1.075,    338.8,   68
                              [750 ,  0.4643,    1.087,    354.6,   76
                              [800 ,  0.4354,    1.099,    369.8,   84
                              [850 ,  0.4097,    1.110,    384.3,   93
```

```
                                [900 ,  0.3868,    1.121,    398.1,   10
                                [950 ,  0.3666,    1.131,    411.3,   11
                                [1000,  0.3482,    1.141,    424.4,   12
                                [1100,  0.3166,    1.159,    449.0,   14
                                [1200,  0.2902,    1.175,    473.0,   16
                                [1300,  0.2679,    1.189,    496.0,   18
                                [1400,  0.2488,    1.207,    530  ,   21
                                [1500,  0.2322,    1.230,    557  ,   24
                                [1600,  0.2177,    1.248,    584  ,   26
                                [1700,  0.2049,    1.267,    611  ,   29
                                [1800,  0.1935,    1.286,    637  ,   32
                                [1900,  0.1833,    1.307,    663  ,   36
                                [2000,  0.1741,    1.337,    689  ,   39
                                [2100,  0.1658,    1.372,    715  ,   43
                                [2200,  0.1582,    1.417,    740  ,   46
                                [2300,  0.1513,    1.478,    766  ,   50
                                [2400,  0.1448,    1.558,    792  ,   54
                                [2500,  0.1389,    1.665,    818  ,   58
                                [3000,  0.1135,    2.726,    955  ,   84

    air_tablea_4[:,3] = air_tablea_4[:,3] * 10**-7 # correct magnitude
    air_tablea_4[:,4] = air_tablea_4[:,4] * 10**-6 # correct magnitude
    air_tablea_4[:,5] = air_tablea_4[:,5] * 10**-3 # correct magnitude
    air_tablea_4[:,6] = air_tablea_4[:,6] * 10**-6 # correct magnitude

    # return initialized analysis properties main function
    return(air_tablea_4)
```

5.5 table_7p6.py

In [ ]:
```
#*********************************************************************
# NAME OF THE PROJECT:  table_7p6.py
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PURPOSE OF THIS MODULE :
#
#                          Hold table 7.6
#
# REQIREMENTS :            (Linux/Mac/Windows) Python 3.x
#
# Developer: Sterling Reynolds , Undergraduate student
#            Contact: icu327@my.utsa.edu
#            Department of Mechanical Engineering, UTSA Texas
#
# DATE:     March 2020 (SR)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import numpy as np
```

In [ ]:
```
def table_7p6_C2():
    #*********************************************************************
    #    Table 7.6 from "Fundamentals of Heat and Mass Transfer 8th ED
    #               ( Photo of table provided in same directory :) )
    #*********************************************************************
    Correction_factor_C2 = np.array([ [1, 2, 3, 4, 5, 7, 10, 13, 16],
                                      [0.70, 0.80, 0.86, 0.90, 0.92, 0

    # return initialized analysis properties main function
    return(Correction_factor_C2)
```