THERMAL MANAGEMENT DESIGN TOOL SYSTEM FOR CUBESAT APPLICATIONS

A Thesis

Presented to

the Faculty of the College of Science

Morehead State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Saikrishna Reddy Kanumuru

April 26, 2019

Accepted by the faculty of the College of Science, Morehead State University, in partial fulfillment of the requirements for the Master of Science degree.

____
Dr. Benjamin K. Malphrus
Director of Thesis

Master's Committee: ________________________________, Chair
Dr. Eric Jerde

____
Jeffery A. Kruth

____
Charles D. Conner

____
Date

THERMAL MANAGEMENT DESIGN TOOL SYSTEM
FOR CUBESAT APPLICATIONS

Saikrishna Reddy Kanumuru
Morehead State University, 2019

Director of Thesis: ____
Dr. Benjamin K. Malphrus

A Thermal Management design tool, also known as PyTherm is a simple thermal analysis tool for small satellites restricted to 1U, 2U and, 3U CubeSats was developed in collaborated with YSPM, LLC, Saratoga, CA. In this study, a comparison of the accuracy of results is made between existing Thermal tools and PyTherm. PyTherm makes assumptions regarding output data due to its limitations and the fact that it does not incorporate CAD models. However, creating an adequate thermal model requires expertise and high costs, and is also time consuming. The process usually takes several weeks to build an accurate model. These long timelines may create delays in the process of thermal modeling and therefore eventually will increases cost. The CubeSat industry is now looking for simple tools to use to analyze thermal conditions that takes less time and are available at low cost. PyTherm can make a large impact in the near future by providing an effective, low cost thermal modeling tool available to the Smallsat industry. Since this tool does not use CAD models, it takes very little a priori

knowledge of thermal systems and thermodynamics to operate this tool. Additionally, results can be obtained in a short period of time- anywhere between a few seconds to a few minutes depending upon the configuration of the CubesSat design. This tool is mainly focused on and developed for small CubeSat companies and universities, based on demand and cost analysis. This study describes the PyTherm tool, explains the underlying thermodynamics upon which the modelling is based, and compares the output to other tools currently available in the aerospace industry.

Accepted by: ______________________________, Chair
Dr. Eric Jerde

____
Jeffery A. Kruth

____
Charles D. Conner

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Boris Yendler for his enormous support and for teaching me the science of thermal analysis and would like to thank Clayton Jayne for help debugging the code. Without their support, this work would not have been possible. I am grateful to Thesis committee chairman Dr. Benjamin K. Malphrus and committee member Jeffery Kruth for their willingness to provide guidance throughout the Thesis. I am also thankful to Victor Clarke for helping me to write the code. Finally, I would like to thank my parents, Bharathi and Raja Ram Reddy for their constant love and support during my stressful situations. Special thanks to my mom for counseling me on how to handle the stress.

Table of Contents

CHAPTER 1 INTRODUCTION 2

1.1 APPROACH: 2

1.2 OVERVIEW: 2

CHAPTER 2 THERMAL ANALYSIS AND DESIGN REVIEW 3

2.1 HEAT ENERGY: 3

2.2 CONDUCTION: 3

2.3 RADIATION: 4

2.4 CONVECTION: 5

2.5 THERMAL DESIGN MODELING: 5

2.5.1 STRUCTURE: 5

2.5.2 FINITE ELEMENT ANALYSIS: 6

2.6 DESIGN REVIEW: 7

2.7.2 THERMAL VACUUM CHAMBER TESTING: 8

2.7.3 THERMAL DESKTOP: 8

2.8 THERMAL CONTROL STRUCTURES: 11

2.9 COMPARISON OF ALTERNATIVES: 12

CHAPTER 3 PYTHERM USER'S MANUAL 14

3.1 DESIGN TRADEOFFS: 14

3.2 USER INPUT: 16

3.4 HEAT RADIATION: 20

3.5 ORBIT MODULE: 21

3.6 THERMAL SOLVER: 23

CHAPTER 4 VALIDATION 27

4.1 TEST CASES: 27

4.1.1 1U CUBE MODEL: 27

4.2.2 2U CUBE MODEL 31

4.2 ADDITIONAL CASES: 38

4.3 RESULTS: 40

CHAPTER 5. CONCLUSION 43

5.1 FUTURE WORK: 44

REFERENCES 45

APPENDIX 48

PROGRAM CODE: 48

List of Figures

Figure 1: Thermal Desktop User Interface (C&R Technologies) 9

Figure 2: Comparison Graph between SatTherm and Thermal desktop (Allison, 2018) 10

Figure 3: Sample Temperature response (MATLAB) 12

Figure 4: Work Flow Diagram (Clayton Jayne) 15

Figure 5: PyTherm User Interface 17

Figure 6: Internal Component Details 18

Figure 7: Conduction network 20

Figure 8: Classical Orbital Elements (Federal Aviation Administration) 22

Figure 9: Orbital Input 23

Figure 10: Comparison of temperature of one side of a 1U cube with 29

Figure 11: Temperature obtained from PyTherm for 1U cubesat model 31

Figure 12: Comparison of temperature of one side of a 2U cube between nodes, by PyTherm and Thermal Desktop 33

Figure 13: Temperature obtained from PyTherm for 2U cubesat model 35

Figure 14: Comparison of temperature of one side of a 3U cube with radiation between nodes, by PyTherm and Thermal Desktop 37

Figure 15: Temperature obtained from PyTherm for 3U cubesat model 38

Figure 16: Temperature predicted by 2U cube model 39

Figure 17: Temperature predicted by 3U cube model 39

Figure 18: Comparison of the lumped model to standard model with high temperature 40

Figure 19: Full model with high temperature 41

Figure 20: Heat radiation 42

Figure 21: Temperature for the orbital periods 42

List of Tables

Table 1: Objectives and Constraints 13

Table 2: 1U construction and surface Details 28

Table 3: 1U orbit details 28

Table 4: t-Test: Two-Sample Assuming Equal Variances for 1U cube model 30

Table 5: 2U construction and surface Details 32

Table 6: 2U orbit details 32

Table 7: t-Test: Two-Sample Assuming Equal Variances for 2U cube model 34

Table 8 : 3U construction and surface Details 36

Table 9: 3U orbit details 36

Table 10: t-Test: Two-Sample Assuming Equal Variances for 3U cube model 37

Chapter 1 Introduction

Satellite thermal control is an important task that can protect a Satellite from an antagonistic thermal envolopeironment, keep it working well and surviving in all mission phases (Huang, 2008). Thermal analysis predicts temperature behavior (absolute and variations) throughout the mission and ensures mission survival. Temperatures outside of survival ranges will cause components to fail in a short period of time. Effective thermal management can extend the operational mission life. If temperatures are not extreme enough to quickly disable electronics, high temperature variation and long-term exposure to extreme temperatures can reduce the lifespan of components (Yendler, 2017). In the development of the Thermal Management Design tool, cost, schedule and technical aspects were considered. In other words, the goal was to develop an inexpensive, rapidly developed and capable thermal control system for use in small satellite design programs. The existing tools requires license activation and open source tools are complex to understand. The PyTherm tool is simple to understand and, its operation does not require a high-level expertise. Since this tool is to write in Python, it is easy to link future packages. This tool is mainly focused on giving a high-level overview of the effectiveness of the entire thermal design. The toolset is based on proven technology with multiple designs, but it requires additional work for Real time operations. Furthermore, PyTherm can be developed into a full-fedged software package in the future and turned to commercial tool concentrating on small companies and universities.

1.1 Approach:

The existing Tools require CAD models that will create a Mesh network, which generates the temperature regions. In contrast, PyTherm makes a set of assumptions regarding the composition

and structure of bus design and electronic components. Once the static thermal network is established, it generates boundary conditions internally and it provides various components to the thermal solver to obtain temperature data. The important tasks of this thermal model to generate a thermal network, generate boundary conditions and to calculate temperature states for the spacecraft for a specified period in a specified thermal envolopeironment. The first step in developing a thermal model in PyTherm is for the user to provide required information for various parts of the program to process. An orbital routine must be input to determines the position and orientation of the spacecraft at each time step, then the information is passed to a heat radiation module which calculates boundary conditions. The internal heat that is generated creates a conductive heat network. Once these data are input into the model, the thermal solver calculates temperature data for the user, provides a temperature graph and the temperature values are saved in an excel file that the user has the option to download and access in the future.

1.2 Overview:

This document starts with a survey of the literature related to thermal investigation and control in little satellites, including elective examination instruments, thermal control structures and variable emissivity surfaces. Following the survey of the literature, this paper identifies specific needs that apply to the CubeSat thermal examination. These needs were addressed by creating PyTherm, a program that captures and models the thermal system. This study starts with the thermal solver and pursues with age of the data sources required for it. The paper closes with a comparison against Thermal Desktop, a recognized benchmark.

Chapter 2 Thermal Analysis and Design Review

2.1 Heat Energy:

As defined in physics, the movement of particles creates heat energy. Heat energy increases as temperature increases because as the temperature rises, atoms move faster, and have more kinetic energy. Heat is transferred from one object to another when the objects are at different temperatures. The amount of heat that is transferred once two objects are brought into contact depends on the discrepancy in temperature between the objects. Heat is transferred only if two objects in contact are at different temperatures. Thermal energy forever moves from hotter to cooler objects, the warmer object loses thermal energy and becomes cooler as the cooler object gains thermal energy and becomes warmer. The heat energy will continue to move from a warmer object to a cooler object until both have the same temperature. Two of the typical three modes of heat transfer can be used within a spacecraft: conduction and radiation. Convection is typically not an option given that it requires a fluidic or gaseous medium.

2.2 Conduction:

Conduction is the transfer of thermal energy through a medium without any flow of the medium (Kombucha, 2014). The transfer is due solely to the atomic and molecular interactions. The particles at the heated end vibrate vigorously having high kinetic energy and collide with neighboring particles and transfer their energy, and eventually the particles at the cooler end are set into more vigorous vibration (Kombucha, 2014). Thus, kinetic energy is transferred from heat atom to the cool side and warming the cooler side. In all solids, thermal energy is transported by collision of particles through vibration.

At first, they collide with atoms in the cooler components of the metal and pass away their energy within the method. Particles in liquid state and gases area Collide between

them and it occurs less frequently, slowing the transfer of kinetic energy and these materials are because of poor conduction of heat.

Conduction rate equation

The measure that is used to quantify conduction is thermal conductivity. The higher the density of the material, the more conductive it is due to the proximity of atoms. Fourier's law gives the conduction rate equation. (wikipedia, n.d.)

$$q_x = -kA \frac{dT}{dX}$$

q_x = Heat Transfer [W or J/S]

K = Thermal conductivity [W/m]

$\frac{dT}{dX}$ = Temperature gradient [C/m or K/m]

A = Cross sectional Area

2.3 Radiation:

Radiation refers to the transfer of thermal energy through electromagnetic radiation and it can occur in a vacuum (Kombucha, 2014). Stefan-Boltzmann equation thermal energy transfer is described by the equation below

$$P_{rad} = \varepsilon * \sigma * T^4 * A$$

ε = emissivity (between 0 and 1)

σ = Stefan-Boltzmann equation = $5.67 * 10^{-8}$ [$W/m^2 k^4$]

T = temperature [K]

A = Surface Area [m^2]

Objects that ideally follow this law are called black bodies. For a similar geometric form, darker colors will absorb more heat and the amount of energy radiated is related to

temperature and surface area only. As an example, a star's color is due to its temperature and the wavelength emitted is derived from Stefan- Boltzmann law combined with other relations (A.Lahrichi, 2017). Solving the Stefan-Boltzmann equation for temperature, we obtain

$$T = \sqrt{\frac{S_{rad}}{\in *6}}$$

$$S_{rad} = \frac{P_{rad}}{A}$$

2.4 Convection:

Convection is referred to the heat transferred by the actual movement of a fluid or gas, such as in a heating system at home, or the earth's atmosphere. Convection is not typically an option for spacecraft thermal systems.

2.5 Thermal Design Modeling:

2.5.1 Structure:

Thermal problems are mathematically stated as a set of restrictions that solutions must verify, some of them given explicitly as data in the statement, plus all the implicit assumed data and equations that constitute the solution itself (Martinez, 2016). It must be kept in mind that both the implicit equations (algebraic, differential, or integral) and the explicit pertinent boundary conditions given in the statement are subjected to uncertainties coming from the assumed pure geometry, assumed material properties, assumed external interactions, etc. In this respect, in modeling a physical problem it is not true that numerical methods are just approximations to the exact differential equations; all models are approximations to real behavior and there is neither an exact model, nor an exact solution to a physical problem; one can just claim to be accurate

enough for the purpose of the modeling (Martinez, 2016). Modeling material properties introduces uncertainties accessible to density, thermal conductivity, thermal capability, emissivity, and so on, depend on the base materials, their impurities, bulk and surface treatments applied, actual temperatures, the results of aging, etc (Martinez, 2016). Mostly, material properties are modeled as uniformly in size, but the accuracy of this model and the right selection of the constant property values require insight.

2.5.2 Finite Element Analysis:

The finite element analysis method (FEA/FEM) is a mathematical technique used to approximate the solutions of systems of partial differential equations (PDE). Most Engineering calculations are done for realistic complex geometries and involve boundary conditions; this often leads to PDEs that don't have an analytical solution. Numerical techniques are therefore the approach used to solve the problems and FEM is one of the most widely used techniques both in industry and research (Zhao, 2016). FEM has a strong theoretical background and has a clear practical methodology that yields, if applied correctly, very accurate results. FEM uses principles from variation calculus and techniques from linear algebra to solve large systems of equations. The studied systems are decomposed into discrete "elements" and equivalent algebraic equations are solved for every element. The process of discretizing the geometry is referred to as meshing and the resulting set of elements a mesh. The higher the number of elements the nearer the approximation is to the time case which ends in additional reliable values. However, increasing the number of elements is limited by computational power and the time requires to run the calculation. Therefore, whenever performing a FEM analysis one should use common engineering practices and a reasonable level of accuracy to address the problem.

2.6 Design Review:

The main objective of satellite thermal design is to ensure that component temperatures remain within precise operational ranges throughout the lifetime of spacecraft. In the past, space programs had to rely on large commercial industries and highly skilled thermal engineers to model spacecraft thermal responses. Today, the space programs are looking at small industries and “nano-class” satellites, which require a unique set of modeling tools and different techniques. Due to time constraints, budget and competitive scenarios, many satellite companies are moving toward satellite specific thermal algorithms for thermal control system design and analysis. While these models are robust and accurate, in situations with complex geometry, complex boundary conditions, or heterogeneous construction, the thermal solution may not be possible all the time. In such cases, another option is the method of finite element analysis (FEA). This model uses nodes with individual heat capacities as well as conductive and radiated connections to neighboring nodes. Examples of CAD models that use FEA are SatTherm, Thermal Vacuum Chamber, and Thermal Desktop.

2.7 Existing Thermal Modeling Approaches:

2.7.1 SatTherm:

SatTherm is a thermal tool, which uses a finite-difference method to solve non-steady temperatures of spacecraft components. This tool utilizes MATLAB scripts, coupled to an Excel user-interface, which can be easily managed by non-thermal experts. At each time-step it determines spacecraft position in space, orientation of exterior surfaces, and net heat flow through each node (VanOutryve, 2008). This tool accurately calculates the spacecraft thermal envolopeironment and can build models within 2 days compared to more detailed models, a

major time saver. Therefore, this design requires high user knowledge; with the increased model generation time and moderate computational requirements SatTherm is not the best option for Non-experts.

2.7.2 Thermal Vacuum Chamber Testing:

Like other designs a thermal vacuum chamber is a testing chamber to simulate the components of Spacecraft. These systems analyze satellite thermal behavior and functionalities to ensure mission success and survivability. These testing chambers were initially designed for large satellites. The chamber cannot mimic orbital conditions even though the operator can adjust the envolopeironment by heating or cooling the chamber walls, which is a drawback for this testing procedure. Also, the tests cannot be performed until the spacecraft is fully constructed.

2.7.3 Thermal Desktop:

Thermal Desktop is a commercially available thermal package used by most space companies for effective and accurate thermal modeling. It is based on CAD models, which are 3D, a combination of finite element, finite difference and lumped parameter networks (C&R Technologies, 2017). This is a robust package, which is associated with other data libraries. Radiation networks are established by a RadCAD library using a ray-tracing algorithm. Figure 1 shows the user interface of the widely used commercial tool Thermal Desktop.

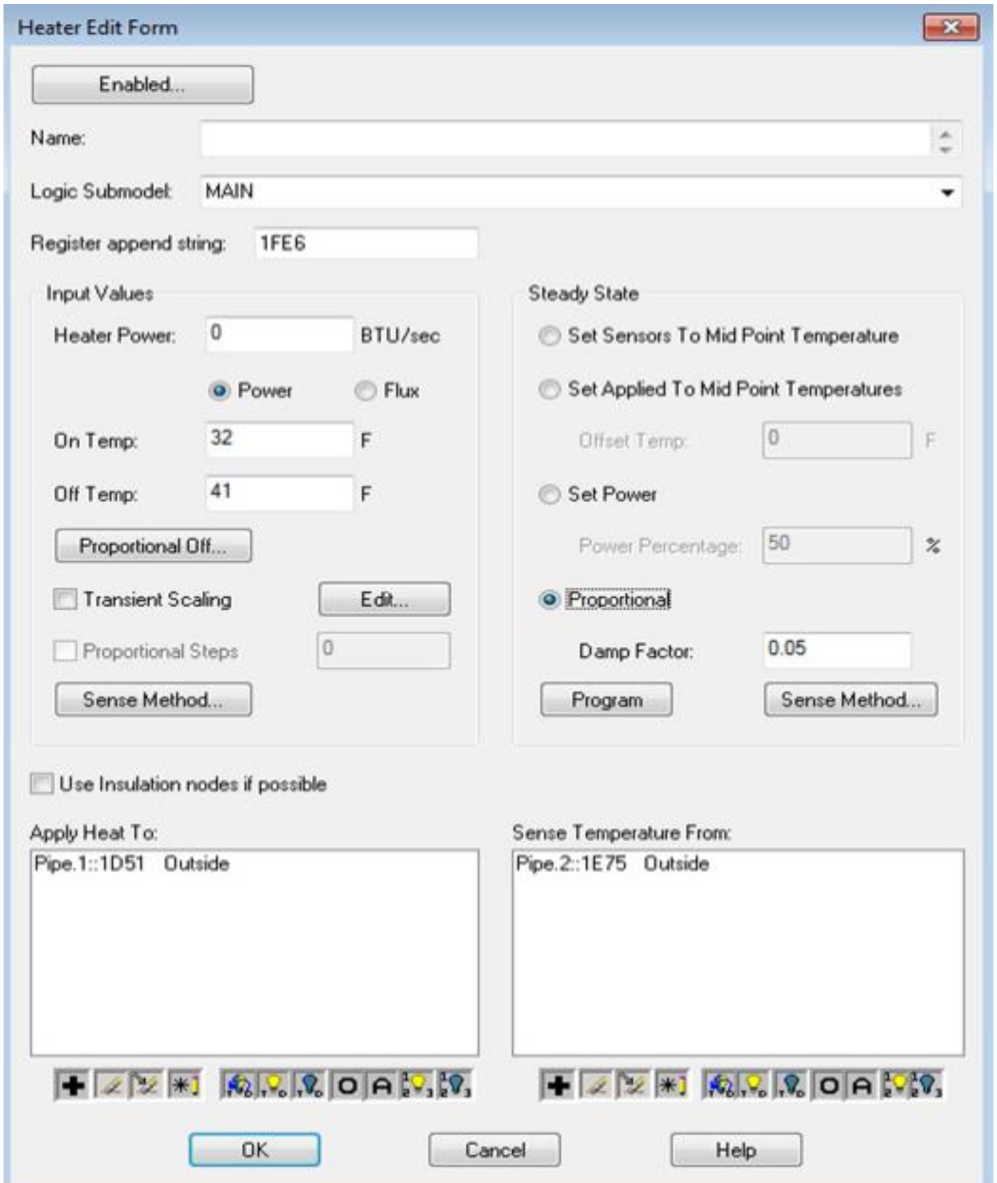

Figure 1: Thermal Desktop User Interface (C&R Technologies, 2017)

The thermal networks are generated by another library (SINDA/FLUINT). By default, thermal Desktop program will generate conductors representing back conduction through the couples. To generate conductors for couples, the user must create a contractor between the cold and hot case or fill the gap between the cold and hot sides with a FD solid brick. Then TEC

dialog box allows the user to add temperature control to either the hot or cold side for transient simulations. This control simulates on/off thermostatic control by default; for steady state simulations, the user can elect to apply a constant input or midpoint control (Grob, 2011). The SINDA model was solved employing a steady-state resolution solver followed by a transient endure four orbits, information was captured over the ultimate a pair of orbits and the min/max temperature combine for every radiator decided. Case Run time for all cases run on dual quad-core processor with 8 GB of RAM and solution time for heating rate calculations approximately takes 1 hour per each case. In summary, due to high knowledge requirements for use, high computational requirements, and high cost for procurement, Thermal Desktop may not be best option for most space companies.

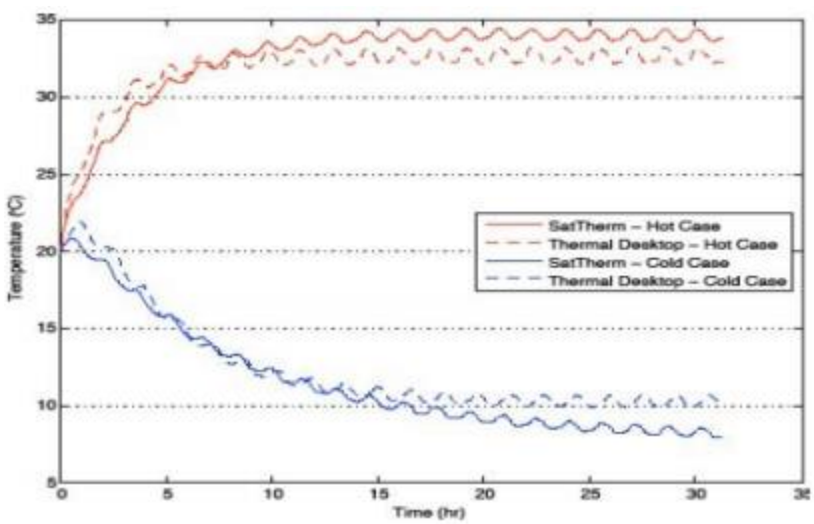

Figure 2: Comparison Graph between SatTherm and Thermal desktop (Allison, 2018)

The graph shown in Figure 2 is taken from a small Earth-orbiting satellite built at NASA Ames and launched in May 2009 (Allison, 2018). This graph is a comparison of generating heat between Thermal Desktop and SatTherm; we can see the slight variation in the graphs. Likewise,

the variation in the heat generation with one Thermal model to another, it is expected that there will be a variation with the Python thermal tool, but in terms of non-complexity this python tool would be much easier to handle. Drawbacks of the python tool are that it is relatively less-accurate and minimally customizable.

2.8 Thermal Control Structures:

Thermal management subsystems support all satellite and payload parts among their essential temperature limits over the complete mission such as operational limits, survival limits, and gradient limits. It can be accomplished by active or passive structures. The system design is a difficult process because of the interplay of heat transfer among the components, and because of the extreme temperatures encountered in space, and because of the changing heat inputs and outputs during the lifespan of a spacecraft. These control system components allow internal and external heat loads to be redistributed, providing a moderate thermal envolopeironment for the payload during all flight phases.

Desired output:

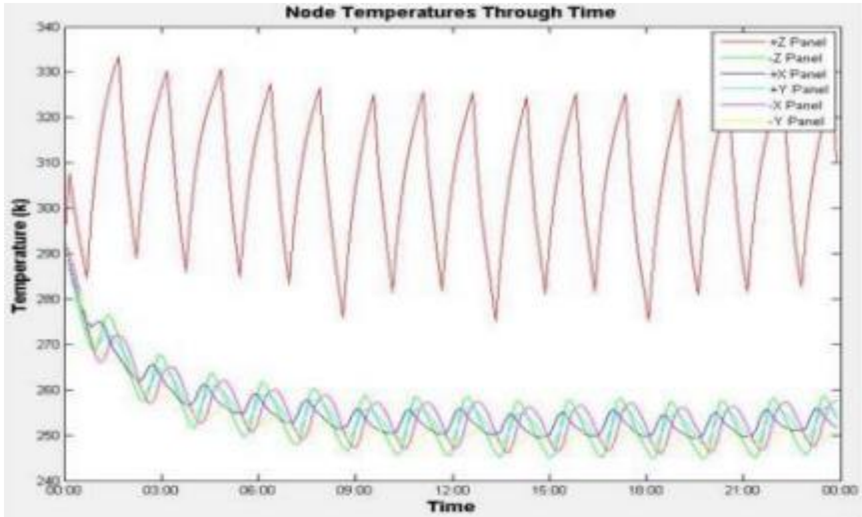

Figure 3: Sample Temperature response (MATLAB)

The sample response shown in Figure 3 was generated in MATLAB; this response is ideal sample for the desired output. This sample output is for a 1U satellite in a thirty-degree inclination with a 6700-kiolometer semi-major axis. Here the thermal control system is using a strip heater to control temperature during eclipse periods. Also, the individual temperature data for conduction network, radiation, heat capacities, and orbital modules will be saved and exported to Excel file that the user can download to their device, which is a major advantage.

2.9 Comparison of Alternatives:

By considering the alternatives for thermal analysis of cubesat class spacecraft, the options are very limited, the tools are very sophisticated and are not accommodating to smaller teams with shorter schedules. SatTherm, while much simpler to use than traditional tools, still requires significant time and needs expertise for generating models. Of all the available tools, the most commonly used means of thermal verification is to perform a thermal vacuum test. Even though, this test is widely used it is ineffective in several ways. The simulation of the space

envolopeironment that is provided by a thermal vacuum is doubtful, since it limits the possibility for correcting alterations unless the spacecraft is fully constructed. With all these limited options, the cubesat industry needs some simple product that is easy to perform and gets simulation much faster, as well as at low cost. Table 1 compares the currently available thermal analysis software tools.

Thermal Analysis	Advantages	Disadvantages
SatTherm	Usually high accuracy, very flexible based upon user needs and their capabilities.	User requires more knowledge, model generation takes quite a lot of time, but computational requirements are moderate.
Thermal Desktop	Standardized product for most companies.	User requires advance knowledge, model generation takes long, computational requirements are high, and it is expensive.
Thermal Vacuum Chamber Testing	Minimal expertise requirement.	Doesn't simulate the orbital radiation envolopeironment, it is not at all possible to perform on a spacecraft until the model is completely constructed, Availability may be limited, and it is expensive.
PyTherm	User requires minimal thermal knowledge, relatively inexpensive, rapid model generation, computational requirement is very low.	Relatively less accurate, minimally customizable.

Table 1: Objectives and Constraints

Chapter 3 PyTherm User's Manual

3.1 Design Tradeoffs:

The defining requirement of this program is to offer accessibility to a wide range of users. This entails building the program to work with limited hardware and entry-level knowledge of thermal physics. The main design tradeoff is building with a reduced node count that represents the constituent components with the minimum resolution possible. While this design reduces computational requirements significantly, it cannot fully represent localized temperature variations which may adversely impact results. This method also significantly reduces memory requirements for the simulation by limiting the range of temperature states and boundary conditions that must be stored.

The primary goal of PyTherm is to provide accessibility to proper thermal analysis for development programs, further the program is designed to generate a thermal network from a simplified user interface. An example of this method is how it generates heat capacities, considering mass differences among components, extra mass can be assigned as electronic components approximated as copper. The workflow for this model design is shown below in Figure 4.

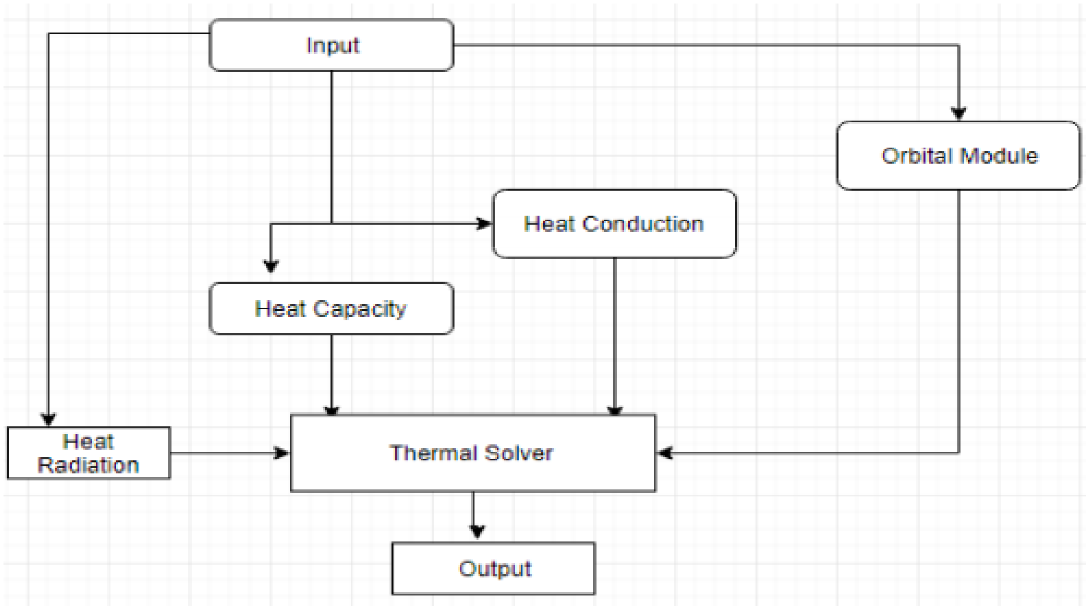

Figure 4: Work Flow Diagram (Clayton Jayne)

The design for this tool has the user enter the Keplerian orbital elements, attitude type for the orbital module, component information, construction and payload details of heat conduction and heat capacity, and component envolopeelopes and placement order for heat radiation. When heat values are generated the thermal solver takes those values and calculates the final temperatures values for the user. Node count for component accuracy is lost due to the invalidity of lumped capacitance assumptions; therefore, does not accurately represent local temperature variations in components such as the elevated temperature of a processor.

To generate internal radiation, ray-tracing algorithms would be difficult to implement within existing architecture and would drastically increase the computational requirements. In addition, errors are introduced in radiative conductors, although these are acceptable as they have a minimal effect on the solution.

3.2 User Input:

Unlike general thermal analysis tools, the PyTherm tool does not require a physical model to generate a mesh; this program only requires basic user inputs limited to test, radio buttons and drop-down selections. Orbit specifications may either be designed by a full set of Keplerian elements or a simplified mode. The user is required to provide elevation and inclination, but a circular orbit will be filled by default. This model does not provide for hot cases and cold cases, since there is no physical model present. As such it is difficult to produce a realistic or highly reliable result. However, this drawback is mitigated by differing eclipsed fractions and a supplement to the simplified orbital determination. Future development is required to fully address these problems. The PyTherm is specifically designed for 1U, 2U, and 3U CubeSats. Since there is no physical model to generate temperature, it is unlikely hard to generate accurate results for big satellites. The user will select electrical bus material, density, heat capacity, heat conductivity, and mass. The thickness of the walls must also be entered, as this will calculate porosity, assisting in various thermal network calculations. The user display for the tool is shown in Figure 5

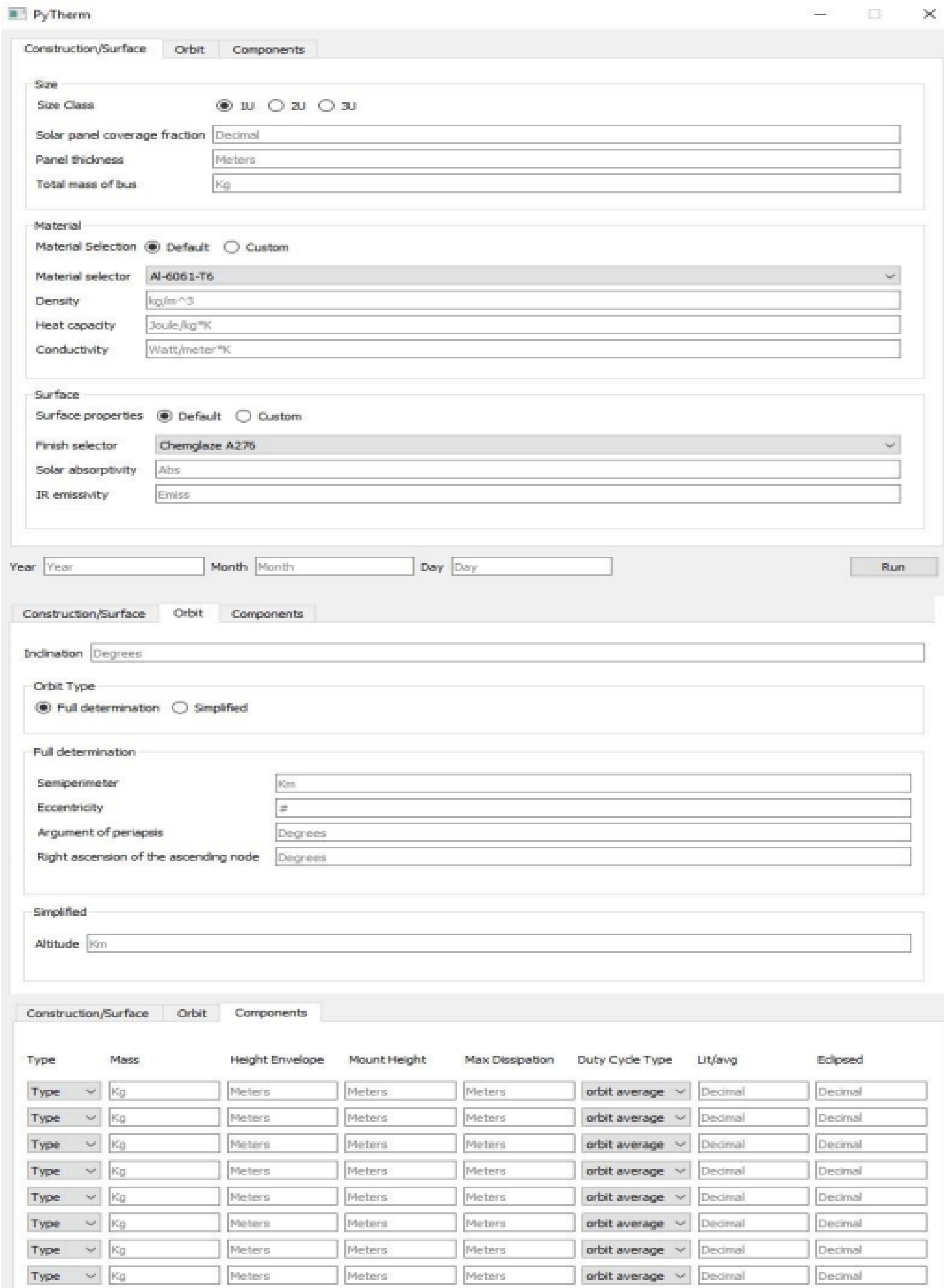

Figure 5: PyTherm User Interface

3.2 Heat Capacity:

The heat capacity reflects the combined effect of mass and composition. Heat capacity and specific heat capacity, both are distinct at their own phases. Heat capacity is an extensive property dependent on certain amount of material, while, specific heat capacity is a property of the composition only. It measures the energy needed to extend the temperature of a unit amount of as elected substance by a selected temperature interval. In the PyTherm tool, the values of heat capacity will be stored in a single column matrix, the first six rows correspond to walls and the remaining elements are assigned to components from bottom to top. The user selects a bus material and mass of walls, which will allow the program to generate a heat capacity for each wall. In the case of batteries, the value is simply determined from the total mass, whereas, in the case of components, they must be represented as a single node. The user is required to provide the mass of the components, also the mass of a standard PCB is subtracted from the total mass to determine the mass of the mounted pieces (NASA JPL). These mounted pieces are represented simply as copper even though they may have higher heat capacity. The heat capacity user input is shown in Figure 6.

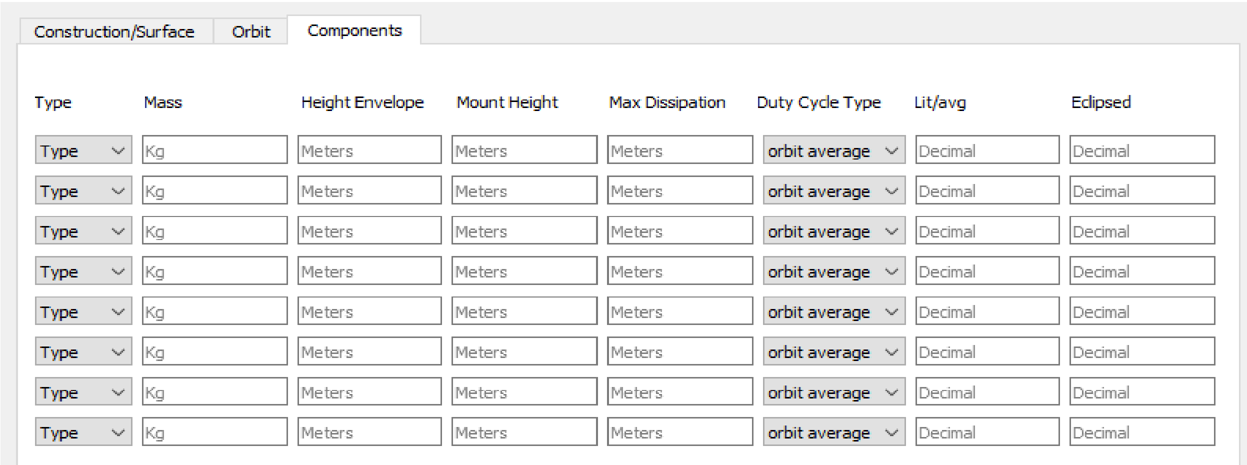

Figure 6: Internal Component Details

3.3 Conduction:

Conduction occurs when thermal energy is transferred from one particle to another (i.e., heat transfer from the more energetic higher temperature particles of a body to the less energetic cooler particles). This implies that a thermal gradient must exist across the body for conduction to occur. The material properties and geometry of a component will play a significant role in its ability to conduct heat. For the thermal solver to generate a solution for all possible configurations without excessive coding, heat network constants must be fed to the program in a structured fashion. This is handled by generating matrices with row to column logic. Specifically, this entails a square matrix where the interface from element A to element B is in row A, column B (Jayne, 2017). Structuring the data in this fashion allows the simulation to scale based upon the number of electronic components located within the bus. Exterior walls fill the first six elements in a specific order with the sizing beyond six depending on the number of electronic components. To fulfill the requirement of minimal user input, several values describing bus 20 construction are chosen. A material is chosen from a preset list to provide density, specific heat and conductivity of the bus. Mass and thickness of walls is also required. From this, a wall porosity value can be calculated to obtain adjusted conductivity of the material. The user interface is shown in Figure 7.

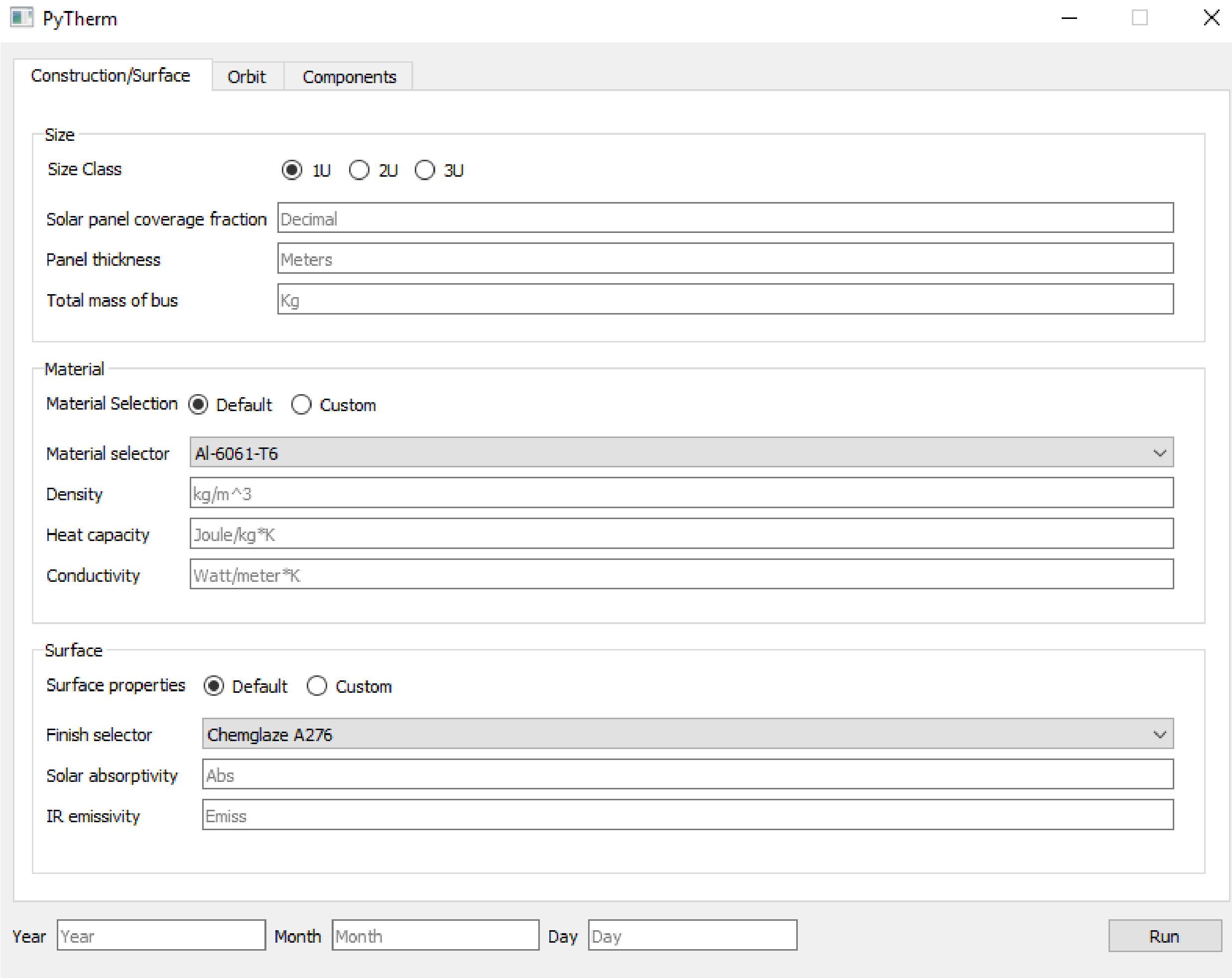

Figure 7: Conduction network

3.4 Heat Radiation:

The rate of thermal energy absorbed by and emitted from a surface through electromagnetic waves is referred to as radiation. Radiation does not require matter to transfer thermal energy and all matter can radiate energy. Most heat transfer in space is because of radiation. The maximum amount of radiation that can be absorbed or emitted by an object is given by the blackbody radiation equation (VanOutryve, 2008).

$$Q_{rad,max} = \sigma A T_{body}^{4}$$

$Q_{rad,max}$ = maximum rate of heat transfers for a black body

σ = Stefan-Boltzmann constant, 5.67xvar10^{-8} W/ K^4m^2

A = Surface area

T_{body}^{4} = absolute temperature of the blackbody (kelvins)

For internal radiation, advanced thermal modeling tools use Monte Carlo ray trace algorithms. This model utilizes traditional formulas and approximations to reduce computational requirements. The values of internal emissivity and the Stefan-Boltzmann constant provide a network of radiative conductors, which is fed directly to the thermal module. All view factors to sidewalls are determined as equal components of the remainder as the view factor for a surface will always sum to one. This leaves a final calculation as sidewalls to other sidewalls. This is impossible to simplify due to the presence of the electronic components. To account for this, the view factors from one sidewall to the others in the absence of the electronic components are calculated. The remaining view factor is then assigned based upon the fraction of the view to each sidewall in the empty bus calculation.

3.5 Orbit Module:

There are several ways to define a Spacecraft's orbit and position in shape. The thermal modeling techniques used in the Adaptive Thermal Modeling Tool (ATMT) are based on the six Keplerian elements or classical orbital elements. The six elements are: semi-major axis (a), eccentricity (e), inclination (i), longitude of ascending node (Ω), argument of periapsis (ω), and the true anomaly (ϑ) (Bishop, 2013). Using these elements, we can define the size, shape, and

orientation of the orbit as well as the position of the spacecraft within that orbit. The figure 8 shows the inclination, longitude of ascending node, argument of periapsis, and the true anomaly are defined with respect to an orbital plane of reference (wikipedia, n.d.).

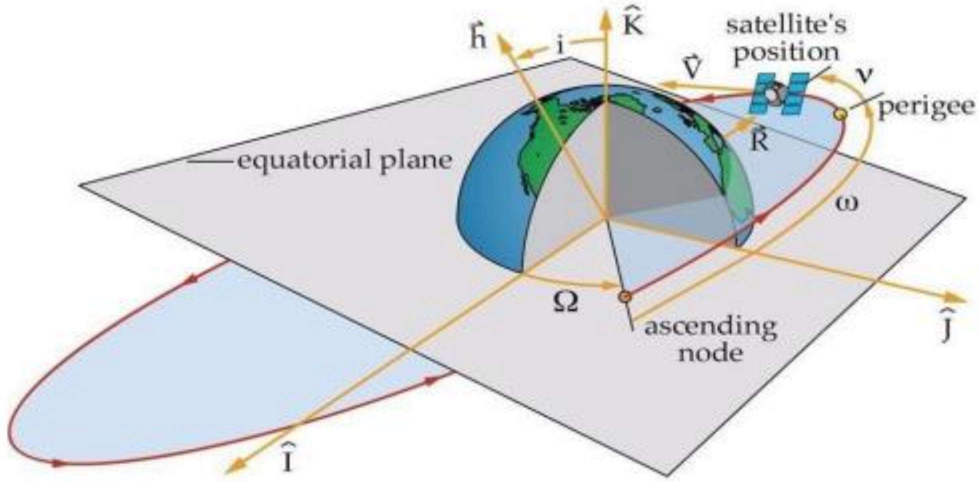

Figure 8: Classical Orbital Elements (Federal Aviation Administration)

The orbital module is the first portion of the program to execute. The main function of this module is to establish the boundary conditions experienced by the satellite during orbit. The process shell responsible for preprocessing will generate hot, cold and nominal orbits for both types of orbit specification. This is achieved by setting the right ascension of the ascending node for maximum and minimum eclipse for cold and hot cases respectively. In the case of a simplified specification, the nominal orbit is set as the intermediate between the hot and cold cases. The orbital module provides a calculation of the heat radiation incident upon the surface. To this end the orientation and intensity of the solar flux must be determined. The user will provide a date to begin simulation, which is processed into a Julian date. The Julian date allows an algorithm to calculate the solar vector, the vector pointing from the center of the earth to the sun in geocentric inertial coordinates (Jayne, 2017). The displacement of this vector provides a

heat flux value, which is held constant over the duration of the simulation. The user input for the orbital module is shown in Figure 9.

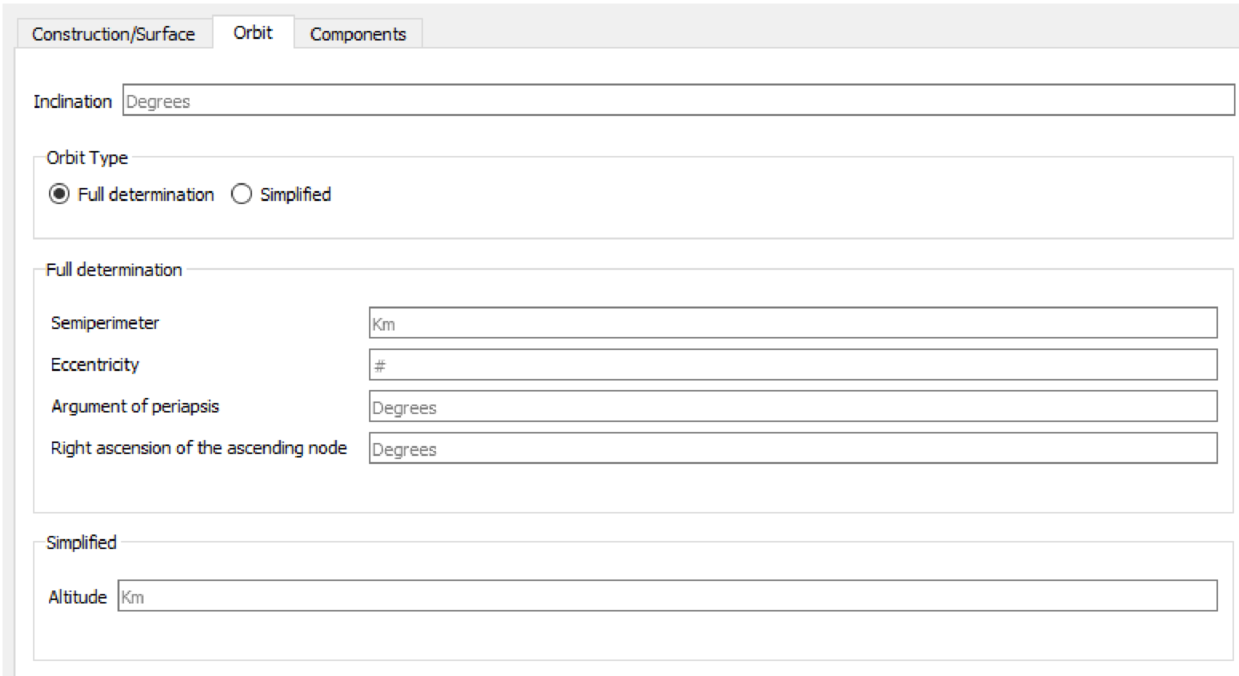

Figure 9: Orbital Input

3.6 Thermal Solver:

The desired output for the thermal solver is a projection of temperature data for the constituent components of the spacecraft, and to that end, the thermal solver is the final step. The basic equation of thermodynamic balance is represented by Equation (1). This solver uses the static network generated in previous modules as well as time dependent heat flux and heat generation values. Temperature values are solved iteratively based upon Equation (2) (Jayne, 2017) with temperatures from the previous time step determining power inbound from connected nodes. These fluxes are multiplied by the program time step and divided by the heat capacity of the node to determine the resultant change in temperature.

$$(1) \qquad \frac{\partial T}{\partial t} = \frac{\sum q_{in} - q_{out} + q_{gen}}{v * \mu * c_p}$$

$$(2) \qquad T_{m,t+1} = T_{m,t} + \frac{\Delta t * \sum_{n=1}^{c+6} \frac{T_{\mathbf{n},t} - T_{\mathbf{m},t}}{R_{\mathbf{m},n}} + \sum_{n=1}^{c+6}\left(\in * \sigma * A_m * B_{m,n} * (T_{n,4} - T_{m,t})\right) + Q_{sen} + \phi_{external}}{C_m}$$

An issue is encountered with interfacing between the thermal program and the orbital routine. The time step for the thermal solver must be kept small enough to prevent instability. To address this the relation shown as Equation (3) is used, a simplified form of the expression representing the iterative temperature solution in Equation (2). The term in parenthesis will become negative if the time step is too large in comparison to the product of the conductive resistance and heat capacity of the node (VanOutryve, 2008). The result is that a higher current temperature results in a lower future temperature, which causes an unstable solution oscillating to infinity. While this necessitates a relatively low time step for the thermal solver, the time step for the orbital solver can be much larger and still generate a stable and meaningful solution, using a larger time step for the orbital solver will reduce computational requirements. The derivation of these stability criteria is shown in Equations (3) and (4) (Jayne, 2017)

$$(3) \qquad T_{m,t+1} = T_{m,t} \frac{\Delta t * \sum_{n=1}^{c+6}\left(\frac{T_{\mathbf{n},t} - T_{\mathbf{m},t}}{R_{\mathbf{m},\mathbf{n}}}\right)}{C_m}$$

$$(4) \qquad T_{m,t+1} = T_{m,t} * \left(1 - \sum_{n=1}^{c+6} \frac{\Delta t}{R_{\mathbf{m},\mathbf{n}} * C_m}\right) + \sum_{n=1}^{c+6}\left(\frac{\Delta t * T_{n,t}}{R_{\mathbf{m},\mathbf{n}} * C_m}\right)$$

To allow a proper interface between these modules, both are set to predefined time-steps. It was decided that the orbital solver would always cover a set number of time steps for the thermal solver. While thermal solvers designed for usage in varied applications must be capable of

variable time-steps, the limited nature of cubesat construction allows for a single time-step to be chosen that will generate a stable solution in all possible configurations. A time-step of one tenth of a second was selected for several reasons. It is well within the range of stability for cubesat configurations, it is easily interpreted as output data, and it provides a basis for interfacing with the orbital functions that is easily modified. For simple convenience, the orbital time step is set at 200 seconds. With the accuracy of the iterative solution of the orbital routine set internally, this time step is well within reasonable bounds. The heat flux data obtained from the orbital solution is interpolated linearly over the 200-second period between steps to provide boundary conditions that change smoothly. The program flow in the thermal section begins with heat transfer from adjacent spacecraft nodes. Each flux source is added to the new temperature in a looping structure. Once all these sources are accounted for, the program proceeds to boundary conditions including radiation to space (Jayne, 2017). Since all external fluxes are handled as a bulk product, the radiation portion of the calculation is handled as though the satellite is in an envolopeironment devoid of any sources. The boundary conditions are then added, which is mathematically equivalent but much easier to process. A derivation of the mathematical justification for separation of external fluxes is shown in Equations (5), (6) and (7).

Solar flux originates from a point source, which can be viewed independently from view factor calculations. Albedo and infrared emitted by the earth are calculated by the view factor from the planet, which leads to a circumstance where solar flux, radiation exchange with space, and radiation exchange with the earth can be separated (Jayne, 2017).

$$(5)\ T_{m,t+1} = T_{m,t+1} + \sum (\varepsilon * \sigma * Am * B_{m,n} * (T_{n,t} - T_{m,t})) + \sum \Phi_{solar} * \epsilon_{solar} * Am$$

$$(6)\ T_{m,t+1} = T' + External\ radiation$$

$$(7)\ External\,radiation = \Phi solar + \Phi albedo * Bm, earth + \epsilon ir * Am * Bm, earth * (TEarth - Tm, t) - (1 - B_{m,\,earth}) * \epsilon ir * Am * T_{m,\,t}$$

Treating exchange of infrared radiation as independent of solar and albedo fluxes as well as assuming space to be at absolute zero yields Equation (8).

$$(8)\ \text{Infrared Exchange} = (B_{m,\,\text{earth}} * T_{Earth} - T_{m,\,t}) * \in_{ir} * Am$$

Once all other heat sources are accounted for, internal generation of the components is introduced. Since all calculations are based upon the previous time step, handling heat exchange in pieces will not affect the solution for the time step being calculated.

Chapter 4 Validation

To validate the results from the PyTherm program, the results of an analysis needs to be compared with an existing thermal analysis tool. The best available source to compare is Thermal desktop; big universities and most companies prefer it. It is capable of both Finite Difference and Finite Element Analysis. This thesis uses a 1U cube, 2U cube and a 3U cube model to examine the validation of PyTherm. For each situation, equal models were worked in both PyTherm and Thermal Desktop, with identical, subjectively chosen orbits and outer situations; preferably the two projects would deliver synchronizing results. Two- sample t-tests are set for each case to investigate the difference between the temperatures. The results indicate no statistically significant difference between the tools.

4.1 Test Cases:

4.1.1 1U Cube Model:

The first test case is a 1U based cube structured model, with same dimensions of 10 cm x 10 cm x 10cm cubic units in both PyTherm and Thermal Desktop. The panel thickness is 0.01, with a material property of Al-6061-T6 and optical property of Chemglaze Z306 coating. In this scenario, the altitude, eccentricity, inclination and other properties are randomly chosen to generate temperature with both tools. This test uses 4 PCB components with 0.2 kg mass, 0.01 meters height, max dissipation of 1 watt and 0.7 eclipsed. The construction, surface and orbit details of the cube model are provided in the Tables 2 and 3.

Name	Value	Range/ units
Size Class	1U	1U, 2U, 3U
Solar panel coverage fraction	0.5	double
Panel Thickness	0.01	Range 0-.03
Total mass of bus	1.5	Range 0-5
Material Selection	0	Default:1, Custom:0
Material	Al-6061-T6	Al-6061-T6, Al-2017-0, Inconel 825, Stainless 308
Density	None	(kg/m**2)
Heat Capacity	None	(Joule/kg*K)
Conductivity	None	Watt/meter*K
Surface Properties	0	Default:1, Custom:0
Finish Selector	ChemglazeZ306	Chemglaze A276, Silvered Teflon Aluminized Kapton, Chemglaze Z306
Solar	0.5	
IR	0.5	

Table 2: 1U construction and surface Details

Name	Value	Range/ Units
Inclination	60	Range 0-180
Full determination	0	Full -0, Simplified -1
Semi parameter (Full)	7000.0	0-1000
Eccentricity (Full)	0.1	0-10
Argument of periapsis (Full)	0.0	0-360
Right ascending node ascension (Full)	0.0	0.360

Table 3: 1U orbit details

The primary step to validate the thermal tool is to ensure that network functions work properly; this verifies the most development of the program. Indeed, although orbital determination and boundary condition application are streamlined relative to other programs, the generation of inactive thermal network from streamlined inputs is the assurance that's expected to help further developers. The results from PyTherm and Thermal desktop are shown in figure below. There is not much discrepancy found in the overall temperature generation. PyTherm gives a good approximation of the temperature conditions of a cubesat, which is useful in initial development stage.

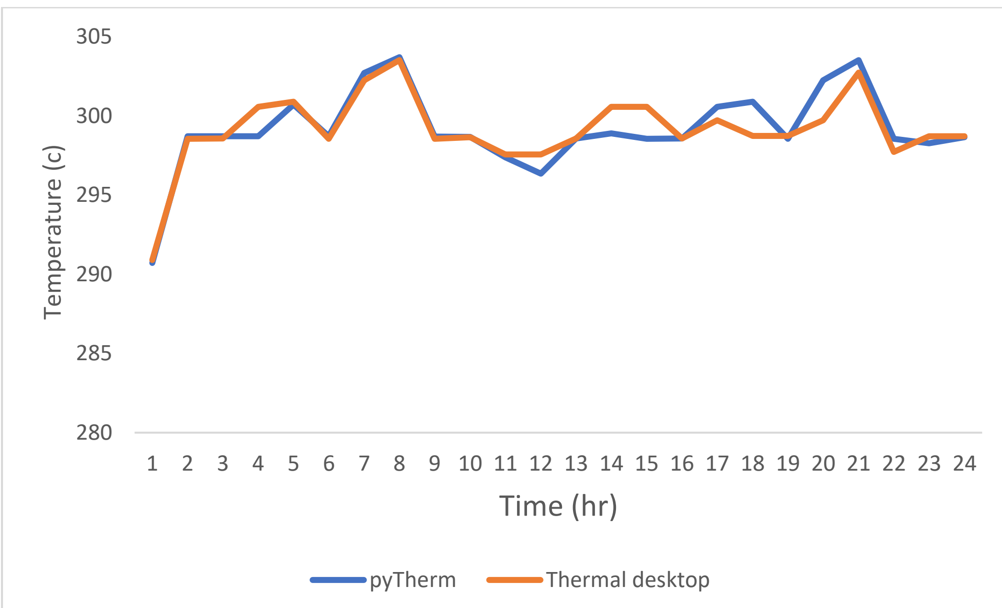

Figure 10: Comparison of temperature of one side of a 1U cube with Conduction between nodes, by PyTherm and Thermal Desktop

To validate the results, a random sample of size 40 was taken and a two-tailed test was set to test the hypothesis that there is no difference between the temperature generated by PyTherm and the temperature generated by Thermal Desktop. At a significance level of 0.05, it was found that there was no evidence to reject the hypothesis. The p-value was 0.47. So statistically speaking, there is no significant difference.

First sample	290.73	290.85
Mean	297.2045	297.28
Variance	16.76052	15.23719
Observations	40	40
Pooled variance	15.99886	
Hypothesized Mean Difference	0	
Df	78	
T stat	-0.0845	
P(T<=t) one-tail	0.466438	
t Critical one-tail	1.664625	
P(T<=t) two-tail	0.473288	
t Critical two-tail	1.990847	

Table 4: t-Test: Two-Sample Assuming Equal Variances for 1U cube model

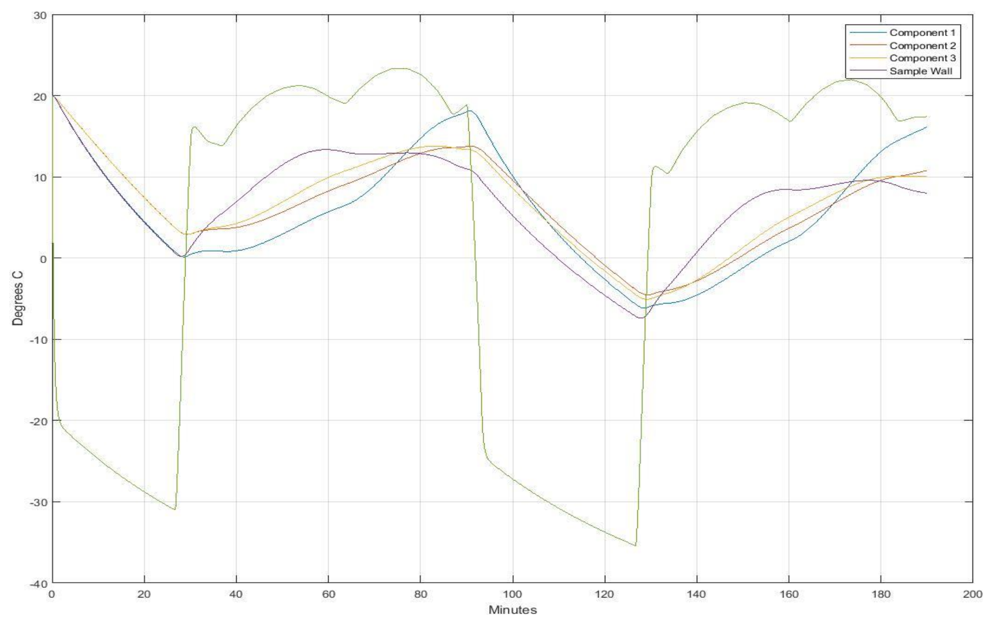

Figure 11: Temperature obtained from PyTherm for 1U cubesat model

4.2.2 2U Cube Model

The second test case is a 2U based cube structured model, with similar dimensions of 10 cm x 10 cm x 20cm cubic units in both PyTherm and Thermal Desktop. The panel thickness is 0.007, with the same material property of Al-6061-T6 but optical property of Chemglaze A276 coating. In this scenario, the altitude, eccentricity, inclination and other properties are randomly chosen to generate the temperature with both tools. This test uses 4 PCB components. Each board has a mass of 0.2 kg, it has a height of 0.01 meters, it has max dissipation 5,2,3,4 watts and eclipse of 0.7,0.9,0.9,1.0 respectively. The construction, surface details and orbit details of the cube model are provided in Tables 5 and 6

Name	Value	Range/ units
Size Class	2U	1U, 2U, 3U
Solar panel coverage fraction	0.7	double
Panel Thickness	0.007	Range 0-.03
Total mass of bus	1.5	Range 0-5
Material Selection	0	Default:1, Custom:0
Material	Al-6061-T6	Al-6061-T6, Al-2017-0, Inconel 825, Stainless 308
Density	None	(kg/m**2)
Heat Capacity	None	(Joule/kg*K)
Conductivity	None	Watt/meter*K
Surface Properties	0	Default:1, Custom:0
Finish Selector	Chemglaze A276	Chemglaze A276, Silvered Teflon Aluminized Kapton, Chemglaze Z306
Solar	0.5	
IR	0.5	

Table 5: 2U construction and surface Details

Name	Value	Range/ Units
Inclination	90	Range 0-180
Full determination	0	Full -0, Simplified -1
Semi parameter (Full)	6670.0	0-1000
Eccentricity (Full)	0.01	0-10
Argument of periapsis (Full)	0.0	0-360
Right ascending node ascension (Full)	0.0	0.360

Table 6: 2U orbit details

The results from PyTherm and Thermal desktop for 2U cube Model are shown in the figure below. In this case the overall temperature output is much better compared to the 1U Cube, but still very minor discrepancy is found as we expected.

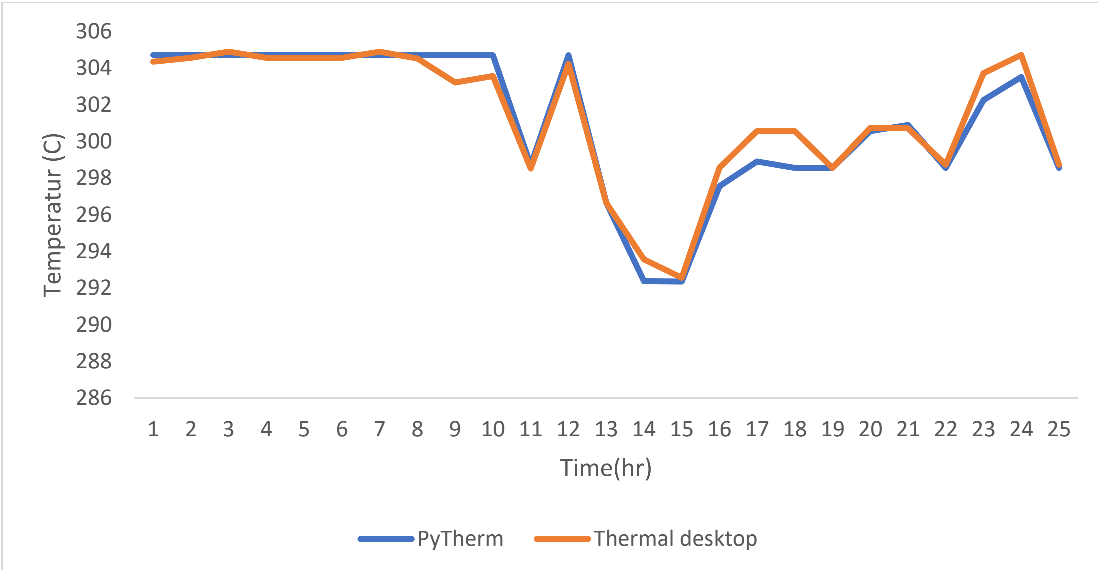

Figure 12: Comparison of temperature of one side of a 2U cube between nodes, by PyTherm and Thermal Desktop

Likewise, in above case to validate the results, a random sample of size 40 was taken and a two-tailed test was set to test the hypothesis that there is no difference between the temperature generated by PyTherm and the temperature generated by Thermal Desktop. At a significance level of 0.05, it was found that there was no evidence to reject the hypothesis. The p-value was 0.63. So statistically speaking, there is no significant difference.

First sample	305.46	306.23
Mean	301.0295	300.6279
Variance	12.85017	16.28634
Observations	40	40
Pooled variance	14.56825	
Hypothesized Mean Difference	0	
Df	78	
T stat	0.470549	
P(T<=t) one-tail	0.319638	
t Critical one-tail	1.664625	
P(T<=t) two-tail	0.639276	
t Critical two-tail	1.990847	

Table 7: t-Test: Two-Sample Assuming Equal Variances for 2U cube model

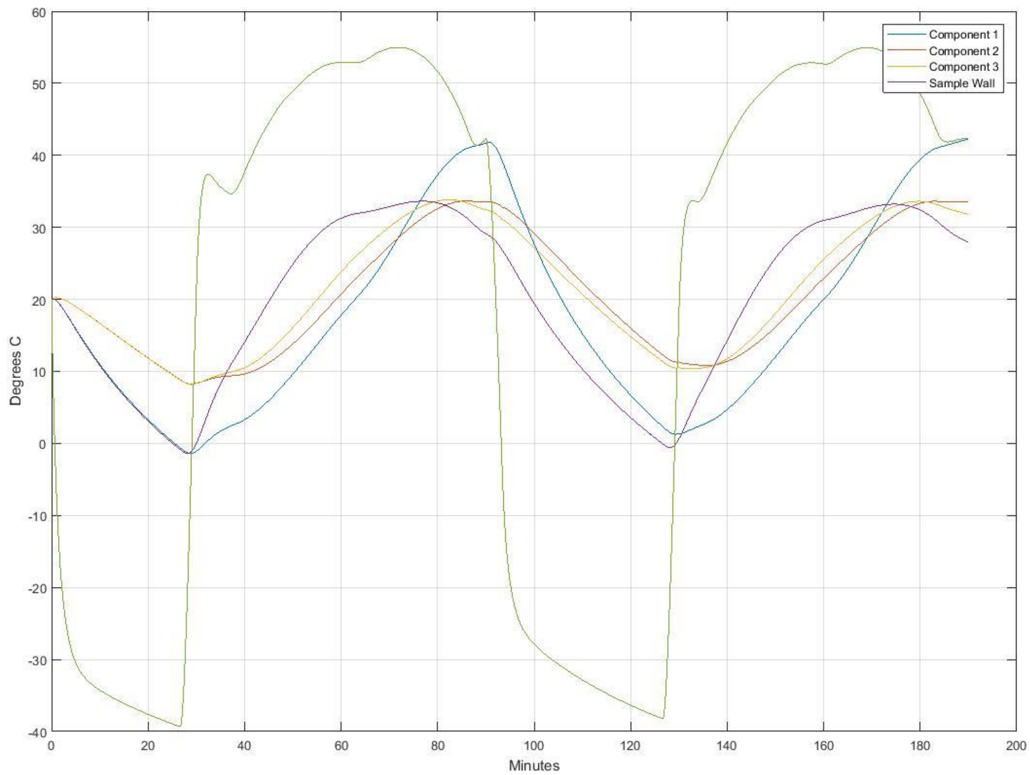

Figure 13: Temperature obtained from PyTherm for 2U cubesat model

4.1.3 3U Cube Model

The third test case is a 3U cube with dimensions of 34cm x 10cm x 10cm which has a panel thickness of 0.007, with the same material property of Al-6061-T6 also optical property of Chemglaze A276 coating. In this scenario, the altitude, eccentricity, inclination and other properties are randomly chosen to generate the temperature with both tools. This test uses 8 PCB components. Each board has a mass of 0.2 kg, it has a height of 0.01 meters, it has max dissipation 3,2,2,1,1,3,3 watts and eclipse of 0.9,0.9,0.9,1.0,1.0,0.7,0.7,0.9 respectively. The construction, surface details and orbit details of the cube model are provided in Tables 8 and 9.

Name	Value	Range/ units
Size Class	3U	1U, 2U, 3U
Solar panel coverage fraction	0.7	double
Panel Thickness	0.007	Range 0-.03
Total mass of bus	3.5	Range 0-5
Material Selection	0	Default:1, Custom:0
Material	Al-6061-T6	Al-6061-T6, Al-2017-0, Inconel 825, Stainless 308
Density	None	(kg/m**2)
Heat Capacity	None	(Joule/kg*K)
Conductivity	None	Watt/meter*K
Surface Properties	0	Default:1, Custom:0
Finish Selector	Chemglaze A276	Chemglaze A276, Silvered Teflon Aluminized Kapton, Chemglaze Z306
Solar	0.5	
IR	0.5	

Table 8 : 3U construction and surface Details

Name	Value	Range/ Units
Inclination	30	Range 0-180
Full determination	0	Full -0, Simplified -1
Semi parameter (Full)	6670.0	0-1000
Eccentricity (Full)	0.01	0-10
Argument of periapsis (Full)	0.0	0-360
Right ascending node ascension (Full)	0.0	0.360

Table 9: 3U orbit details

Comparison of temperatures from PyTherm and Thermal Desktop for 3U cube is shown in the Figure 14 below.

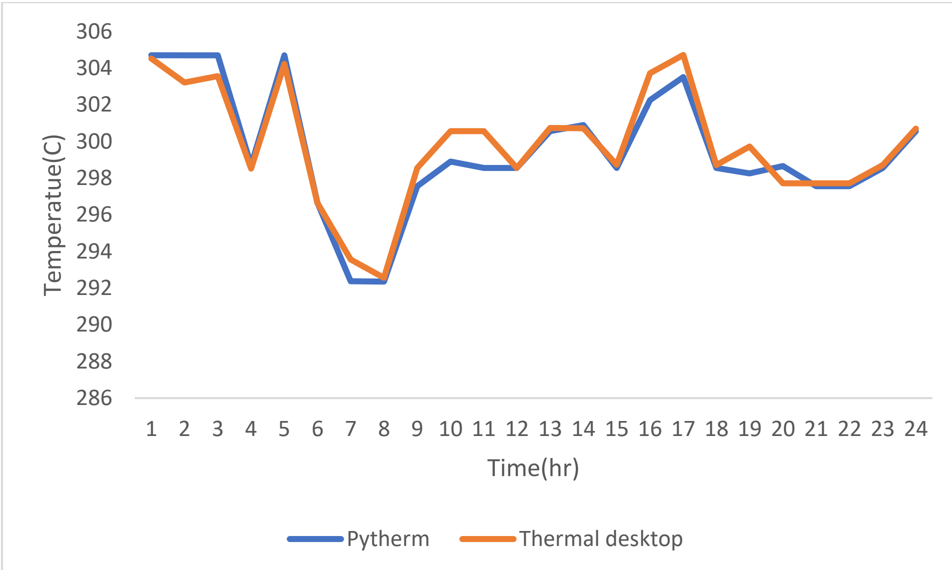

Figure 14: Comparison of temperature of one side of a 3U cube with radiation between nodes, by PyTherm and Thermal Desktop

In this test case the p-value was 0.21. Therefore, statistically speaking, there is no significant difference between these two tools.

First sample	319.692	317.782
Mean	318.7393	318.5017
Variance	0.294886	1.107261
Observations	40	40
Pooled variance	0.701074	
Hypothesized Mean Difference	0	
Df	76	
T stat	1.253238	
P(T<=t) one-tail	0.106981	
t Critical one-tail	1.665151	
P(T<=t) two-tail	0.213961	
t Critical two-tail	1.991673	

Table 10: t-Test: Two-Sample Assuming Equal Variances for 3U cube model

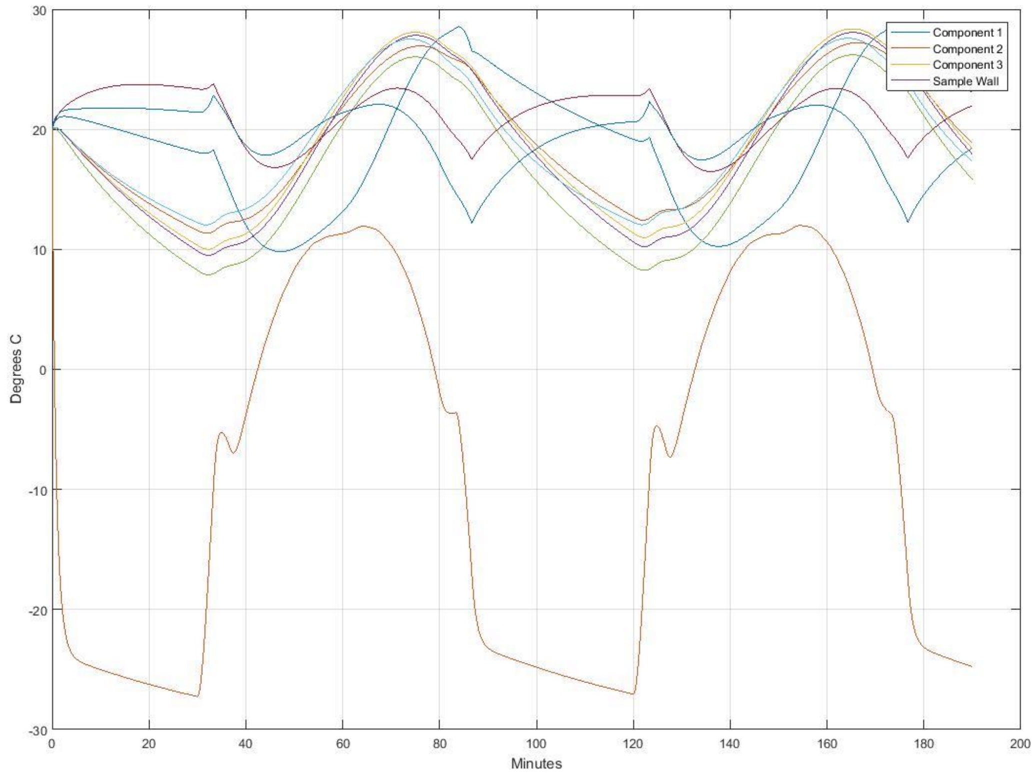

Figure 15: Temperature obtained from PyTherm for 3U cubesat model

4.2 Additional cases:

In addition to above cases, two other cases are taken in PyTherm and generate the temperature to check the efficiency more precise. In consideration of those two cases, one is 2U cube and a 3U CubeSat, has a panel thickness of 0.005, with the same material property of Al-6061-T6 also optical property of Chemglaze A276 coating. Likewise, the above cases, the altitude, eccentricity, inclination and other properties are also randomly chosen to generate the temperature with both tools. One case uses 4 components and other use 5 components. The results from those cases are shown in the Figures 16 and 17 below.

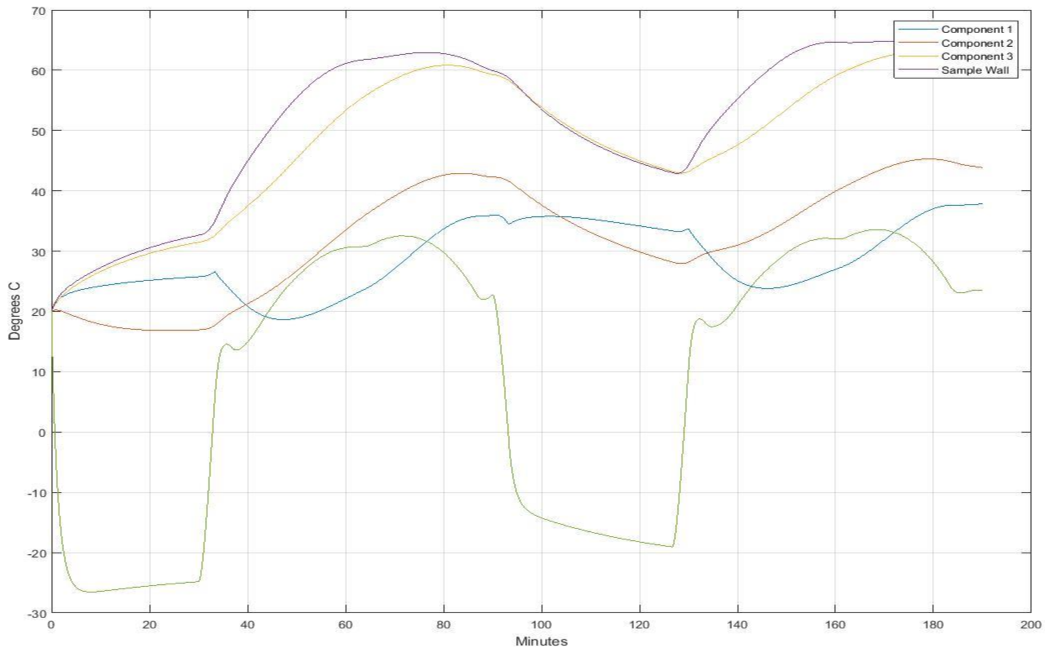

Figure 16: Temperature predicted by 2U cube model

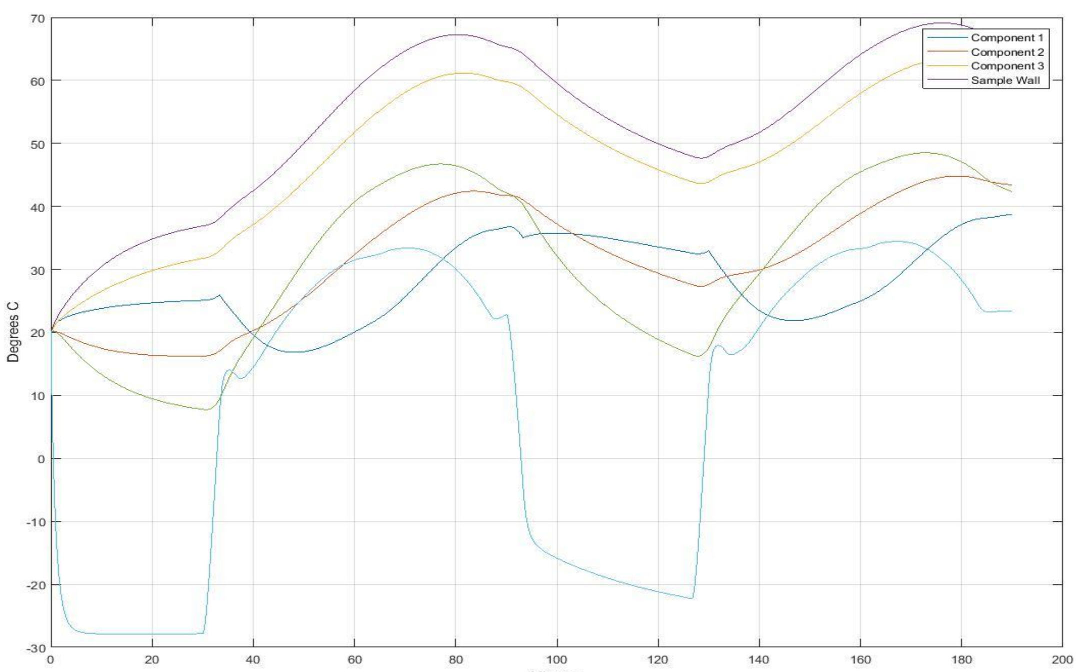

Figure 17: Temperature predicted by 3U cube model

4.3 Results:

In the absence of regular Thermal Desktop data, a valid test for the thermal model is to compare against a single node lumped model. This compares the data in the model and validates the node network functions properly whether justifies the multi-node model. In this test we initialize at a higher temperature of 400 K, with the envolopeironment at a steady temperature of 220 K, the results are shown in Figure 18. When compared to the full model, lumped model temperature falls steadily between the interior temperatures. In the instance of cooling, the lumped model drops below the temperature due to outer walls of the multi-node model but happens slowly due to the fourth power scaling of heat radiation transfer with absolute temperature. The results from the full model are shown in the Figure 19

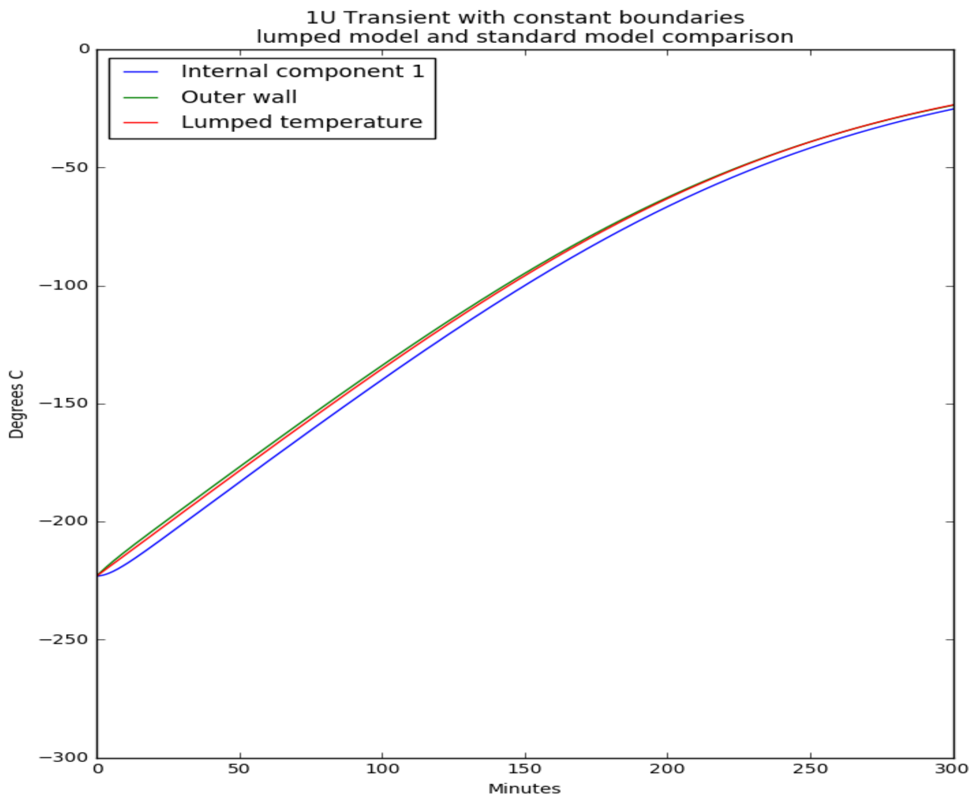

Figure 18: Comparison of the lumped model to standard model with high temperature

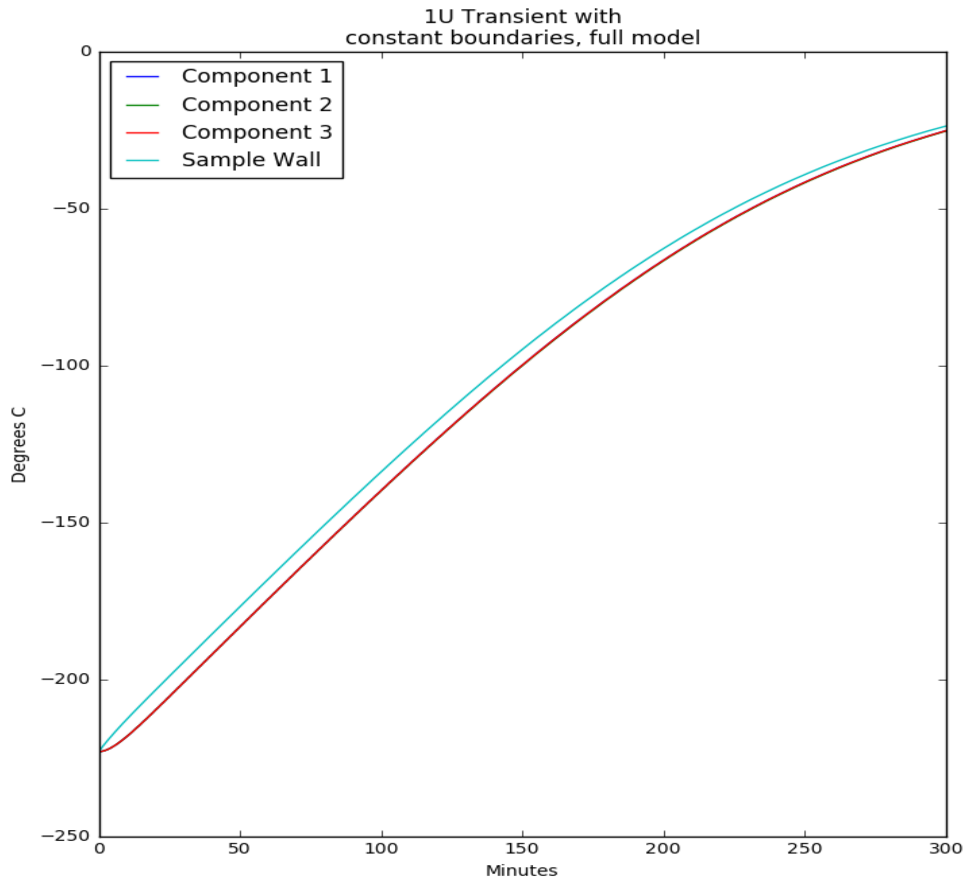

Figure 19: Full model with high temperature

While developing the PyTherm the heat conditions are predefined to check the temperature prediction. The mass of the bus is 2.7 kg, panel thickness is 0.005, an eccentricity of 0.0005887, inclination of 51.6369, altitude of 404 and argument of periapsis is 85.68. The results from the heat radiation, and orbital properties are shown in the Figures 20 and 21.

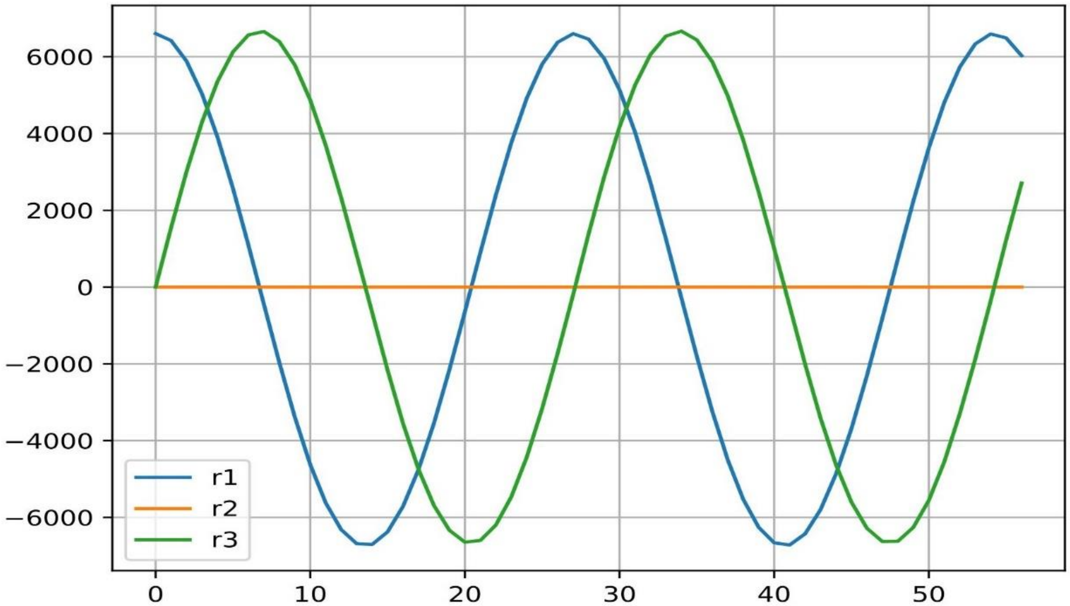

Figure 20: Heat radiation

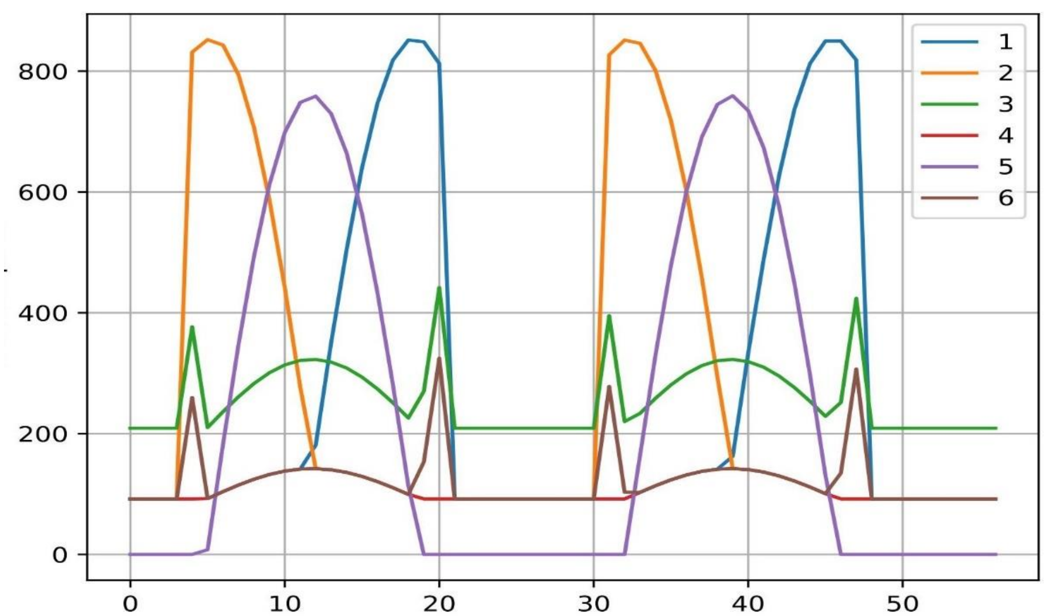

Figure 21: Temperature for the orbital periods

Chapter 5. Conclusion

The goal of this thesis was to produce and validate a thermal network from minimal inputs for the PyTherm software toolset. This is often vital because it fulfills the necessity of the program making thermal displaying accessible to clients with restricted time and experience. This incorporates the mass distribution properties that are utilized for calculating heat capacity, conduction, and the rationale-based task of inward view factors. Some prototype cases have been presented, some cases were compared to the commercially used tool of Thermal Desktop and it was found that there was no significant difference in the outputs of these two tools. These cases involved 1U, 2U, 3U cube models, and PyTherm calculates differential heat equations for given models. The temperatures predicted by the PyTherm agrees with the temperatures predicted by the Thermal Desktop within 6 $°C$ or less.

The use of Python language proved useful to develop a tool that can generates temperatures for cubesat models with least possible inputs from the user. The discrepancy between two tools are very minimal likely of as ±6 $°C$. This is primarily because of absence of CAD models and calculates temperature with just bus components. Even some available Thermal tools available in market like SatTherm models 3-D objects as 2-D flat surfaces, therefore, some variation can be found. Moreover, using different mathematical calculations in different tools would give some error but should not be constrained. PyTherm is an effective thermal analysis tool for 1U, 2U and 3U CubeSats, providing accurate results to within ±6 $°C$. In addition, PyTherm has the advantage of being a user-friendlier tool.

5.1 Future Work:

Although, the PyTherm takes little longer to complete one run compared to available commercial tools, ideally it saves time in learning the program and builds a model. In its present term, PyTherm does have certain limitations due to various complications. In the future, it can be improved in several ways.

1. PyTherm can be used currently for 1U, 2U, 3U; it can further be developed to incorporate 6U cubesat models. Also, PyTherm currently works using the python platform. Future work can make it available using other platforms and it can be developed into desktop software.

2. Inclusion of the flexibility to outline completely different optical properties on different sides of one surface.

3. An additional reasonable check and duty assignment of the time-step if the initial time-step is too large, which can cause an unbalanced solution.

4. As per initial design of PyTherm, the temperature data is saved in excel file after the tool gives the temperature graph. This feature is disabled in current development to avoid program crashing but can be developed in the future.

References

Allison, C. (2018). SatTherm: A Thermal Analysis and Design Tool for Small Spacecraft. 23rd Annual AIAA/USU Conference on Small Satellites, (pp. 7-8).

A.Lahrichi. (2017). Heat Transfer Modeling and Simulation of MASATVAR1.

Bishop, R. (2013). Propagation of CubeSats in LEO using NORAD two line element sets: Accuracy and update frequency. AIAA Guidance, Navigation, and Control (GNC) Conference.

Cassandra Belle VanOutryve. A Thermal Analysis and Design Tool for Small Spacecraft. Master's thesis, San Jose State University, 2008.

Cengel, Y. A. (1998). Heat transfer: A practical approach. San Francisco: McGrawHill.

C&R Technologies. (2017, 5 17). *crtech.com.* Retrieved from crtech.com: https://www.crtech.com/sites/default/files/files/Guides-Manuals/Protected/AdvModTech.pdf

David A. Vallado. Fundamentals of Astrodynamics and Applications. Space Technology Library, 2001.

Space Vehicles Directorate. Nanosat-6 User's Guide: University Nanosat-6 Program. Air Force Research Laboratory, 1 edition, January 2009.

Robert Siegel and John R. Howell. Thermal Radiation Heat Transfer. McGraw-Hill, 1972.

S. Schick, USU Small Satellite conference (2011), Isothermal Structural Panels for Spacecraft Thermal Management.

Federal Aviation Administration. (n.d.). *Federal Aviation Administration.* Retrieved from www.faa.gov: https://www.faa.gov/about/office_org/headquarters_offices/avs/offices/aam/cami/library/online_libraries/aerospace_medicine/tutorial/media/III.4.1.4_Describing_Orbits.pdf

Gluck, D. F., & Baturkin, V. (2002). Mountings and interfaces. In D. G. Gilmore (Ed.), Spacecraft thermal control handbook. Vol. I: Fundamental technologies. El Segundo, CA: The Aerospace Press.

Grob, E. W. (2011, 8 15-19). Thermo-Electric Coolers. Newport News, VA, United states.

Huang, J. (2008, 10 16). Powershow.com. Retrieved from https://www.powershow.com/view4/5895c9-MGYxM/Picosat_System_Design_Course_-_powerpoint_ppt_presentation

Jayne, W. C. (2017). A Simplified Thermal Design Tool for CubeSat Applications.

J. DiPalma , USU Small Satellite conference (2004), Applications of Multifunctional Structures to Small Spacecraft.

Kombucha, P. (2014, 8 14). Retrieved from slideshare.net: https://www.slideshare.net/kombuchamushroom/conduction-ppt

Martinez, I. (2016). Heat Transfer and Thermal Radiation Modelling.

Basics of Space Flight (n.d.). NASA JPL. *solarsystem.nasa.gov.* Retreived from https://solarsystem.nasa.gov/basics/

Q. Young, USU Small Satellite conference (2008), Implications of Advanced Thermal Control Architecture for Modular Spacecraft

SatTherm: A Thermal Analysis and Design Tool for Small Spacecraft (2009), Cassandra Allison, Millan Diaz-Aguado, Belgacem Jaroux.

VanOutryve, C. B. (2008). A Thermal Analysis and Design Tool for Small Spacecraft.

W. Clayton Jayne, A Simplified Thermal Design Tool for CubeSat Applications, Master's thesis, Saint Louis University, 2017.

"Thermal Conduction" wikipedia(n.d.). The Free Encyclopedia. Wikimedia Foundation.

Yendler, D. (2017). Retrieved from mstl.atl.calpoly.edu: http://mstl.atl.calpoly.edu/~workshop/archive/2017/Spring/Alternates/Boris%20Yendler.pdf

Zhao, J. (2016). Deformation measurement using digital image correlation by adaptively adjusting the parameters.

APPENDIX

Program Code:

Development of GUI

```
from PyQt5 import uic, QtWidgets, QtGui, QtCore
from PyQt5.QtCore import pyqtSignal, QThread, QSettings

def show_detailed_warning(title, text, details):
    msg = QtWidgets.QMessageBox()
    msg.setIcon(QtWidgets.QMessageBox.Warning)
    msg.setText(text)
    msg.setWindowTitle(title)
    msg.setDetailedText(details)
    msg.setStandardButtons(QtWidgets.QMessageBox.Ok)
    msg.exec_()


class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, parent=None):

        # Init and load the UI file
        super(MainWindow, self).__init__(parent)
        uic.loadUi('main.ui', self)

        # Init config
        self.init_config()

        # Init other interesting non-UI members
        self.init_members()

        # Init events
        self.init_signals()

        self.show()

    def init_config(self):
```

```
        self.setFixedSize(800, 605)
        self.statusBar().setSizeGripEnabled(False)
        self.setWindowTitle("PyTherm")

    def init_members(self):
        # Initialize radio buttons IDs
        self.sizeClass.setId(self._1u, 1)
        self.sizeClass.setId(self._2u, 2)
        self.sizeClass.setId(self._3u, 3)

        self.matSel.setId(self.msdef, 0)
        self.matSel.setId(self.mscus, 1)

        self.surfProp.setId(self.spdef, 0)
        self.surfProp.setId(self.spcus, 1)

        self.orbitType.setId(self.otfd, 0)
        self.orbitType.setId(self.otsim, 1)

# Get input from all of the controls except Components tab's controls
        yr = self.yr.text()
        mo = self.mth.text()
        d = self.day.text()
        sclass = self.sizeClass.checkedId() # size class
        panel_cov = self.spcf.text() # solar panel coverage
        tempwall = self.pt.text() # panel
        mbus = self.tmob.text() # total mus bus bus
        cust_mat = self.matSel.checkedId() # mat selector
        mat_contents  = self.ms.currentText() # MAterian drop down
        cust_bus_dens = self.den.text()# desbusty
        cust_bus_hcap = self.hc.text()# heat capacity
        cust_bus_cond = self.cond.text()# Conductivity
        cust_surf = self.surfProp.checkedId() # Surface properites
        surf_contents = self.fs.currentText() # Finish drop down
        cust_solabs = self.sa.text()# Solar Abs
```

```
cust_iremiss = self.ire.text()# IR Emiss
simp_orbit = self.orbitType.checkedId() # Orbit type radio button
inc = self.incl.text()# Inclicnation
a = self.sp.text()# Semipermeter
ecc = self.ecc.text()# Eccentricity Argument of peri
argP = self.aop.text()# Argument of prepase
raan = self.raan.text()# Right ascension of the ascending node
altitude = self.alt.text()# altitude

# Get input from Components tabs' controls
# For maximum compatibility, we're not going to make an array out of those here

ctype1 = self.ctype_1.currentText()
masscom1 = self.cmass_1.text()
envolope1 = self.chenvolope_1.text()
hm1 = self.cmh_1.text()
tgen1 = self.cmdis_1.text()
dcyctype1 = self.cdct_1.currentText()
dcyc11 = self.clit_1.text()
dcyc21 = self.cecl_1.text()
ctype2 = self.ctype_2.currentText()
masscom2 = self.cmass_2.text()
envolope2 = self.chenvolope_2.text()
hm2 = self.cmh_2.text()
tgen2 = self.cmdis_2.text()
dcyctype2 = self.cdct_2.currentText()
dcyc12 = self.clit_2.text()
dcyc22 = self.cecl_2.text()

ctype3 = self.ctype_3.currentText()
masscom3 = self.cmass_3.text()
envolope3 = self.chenvolope_3.text()
hm3 = self.cmh_3.text()
tgen3 = self.cmdis_3.text()
dcyctype3 = self.cdct_3.currentText()
```

```
dcyc13 = self.clit_3.text()
dcyc23 = self.cecl_3.text()

ctype4 = self.ctype_4.currentText()
masscom4 = self.cmass_4.text()
envolope4 = self.chenvolope_4.text()
hm4 = self.cmh_4.text()
tgen4 = self.cmdis_4.text()
dcyctype4 = self.cdct_4.currentText()
dcyc14 = self.clit_4.text()
dcyc24 = self.cecl_4.text()

ctype5 = self.ctype_5.currentText()
masscom5 = self.cmass_5.text()
envolope5 = self.chenvolope_5.text()
hm5 = self.cmh_5.text()
tgen5 = self.cmdis_5.text()
dcyctype5 = self.cdct_5.currentText()
dcyc15 = self.clit_5.text()
dcyc25 = self.cecl_5.text()

ctype6 = self.ctype_6.currentText()
masscom6 = self.cmass_6.text()
envolope6 = self.chenvolope_6.text()
hm6 = self.cmh_6.text()
tgen6 = self.cmdis_6.text()
dcyctype6 = self.cdct_6.currentText()
dcyc16 = self.clit_6.text()
dcyc26 = self.cecl_6.text()

ctype7 = self.ctype_7.currentText()
masscom7 = self.cmass_7.text()
envolope7 = self.chenvolope_7.text()
hm7 = self.cmh_7.text()
tgen7 = self.cmdis_7.text()
```

```
        dcyctype7 = self.cdct_7.currentText()
        dcyc17 = self.clit_7.text()
        dcyc27 = self.cecl_7.text()

        ctype8 = self.ctype_8.currentText()
        masscom8 = self.cmass_8.text()
        envolope8 = self.chenvolope_8.text()
        hm8 = self.cmh_8.text()
        tgen8 = self.cmdis_8.text()
        dcyctype8 = self.cdct_8.currentText()
        dcyc18 = self.clit_8.text()
        dcyc28 = self.cecl_8.text()

       # Build a list out of the inputs
        args_list = [yr, mo, d, sclass, panel_cov, tempwall, mbus, cust_mat, mat_contents ,
cust_bus_dens, cust_bus_hcap, cust_bus_cond, cust_surf, surf_contents, cust_solabs,
cust_iremiss, simp_orbit, inc, a, ecc, argP, raan, altitude, ctype1, masscom1, envolope1, hm1,
tgen1, dcyctype1, dcyc11, dcyc21, ctype2, masscom2, envolope2, hm2, tgen2, dcyctype2,
dcyc12, dcyc22, ctype3, masscom3, envolope3, hm3, tgen3, dcyctype3, dcyc13, dcyc23, ctype4,
masscom4, envolope4, hm4, tgen4, dcyctype4, dcyc14, dcyc24, ctype5, masscom5, envolope5,
hm5, tgen5, dcyctype5, dcyc15, dcyc25, ctype6, masscom6, envolope6, hm6, tgen6, dcyctype6,
dcyc16, dcyc26, ctype7, masscom7, envolope7, hm7, tgen7, dcyctype7, dcyc17, dcyc27, ctype8,
masscom8, envolope8, hm8, tgen8, dcyctype8, dcyc18, dcyc28]

       if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()

sys.exit(app.exec_())
```

Calculate Heat Capacity

```
class Heatcapacityf:
# Define bus variables and mass of the components
    def heatcapacityf(htbus, tempwall, masscom, ctype, por, wallfixed, ccustom, mucustom):
        capindex=np.array([896,1,1])
        muindex=np.array([2700,1,1])
        if wallfixed==0:
```

```
            cal=ccustom
            mual=mucustom
        else:
            cal=capindex(wallfixed)
            mual=muindex(wallfixed)
       cpcb = 950
        cec=385*0.7
        cbat=1
        masspcb=0.09*0.09*0.002*1850
# calculate the heat capacity for the components
        tcapacity = np.zeros(((6+len(masscom)),1), dtype = np.double)
        tcapacity[0]=.1*.1*tempwall*mual*cal*(1-por)*1.1
        tcapacity[1]=.1*.1*tempwall*mual*cal*(1-por)*1.1
        tcapacity[2]=.1*htbus*tempwall*mual*cal*(1-por)*1.1
        tcapacity[3]=.1*htbus*tempwall*mual*cal*(1-por)*1.1
        tcapacity[4]=.1*htbus*tempwall*mual*cal*(1-por)*1.1
        tcapacity[5]=.1*htbus*tempwall*mual*cal*(1-por)*1.1

        ii=0
        while ii<(len(tcapacity)-6):
            #print(ii)
            if ctype[ii]==1:
                tcapacity[ii+6]=masspcb*cpcb+((masscom[ii]-masspcb)*cec)
            elif ctype[ii]==2:
                tcapacity[ii+6]=masscom[ii]*cbat
            ii=ii+1
            #print(tcapacity[ii+5])
        print('\ntcapacity:\n\n', tcapacity)
```

Calculate conduction

```
class Conduction:
    def conduction(envolope,tempwall,por,htbus,ctype,wallfixed,kcustom):   # define variables
        if wallfixed==0:
            kal=kcustom
```

```
    else:
        kal = kalindex(wallfixed)
    kal = kal*(((2*kal)-(2*por*kal))/(2*kal-por*kal))

    kbolt = 0.1
    kpcb = 17.4
    temppcb = 0.002
    rjoint = 0
    kbat = 1

# calculate the heat conduction for the components
    kfixedarray = ((len(envolope)+6), (len(envolope)+6))
    kfixed = np.zeros(kfixedarray, dtype = np.double)
    ksidev = (0.5*htbus)/(kal*(tempwall*wdbus))
    ksideh = (0.5*wdbus)/(kal*(tempwall*htbus))

    ktop = (0.5*wdbus)/(kal*(tempwall*wdbus))
    kboard = (1/kbolt)+(.5*wdboard/(kpcb*1.1*temppcb*wdboard))
    kside2side = (ksideh*2)+(rjoint/(htbus*tempwall))
    kside2top = ksidev+ktop+(rjoint/(wdbus*tempwall))
    kpcb2side = ksideh+kboard

    kfixed[0,2]=kside2top
    kfixed[0,3]=kside2top
    kfixed[0,4]=kside2top
    kfixed[0,5]=kside2top

    kfixed[1,2]=kside2top
    kfixed[1,3]=kside2top
    kfixed[1,4]=kside2top
    kfixed[1,5]=kside2top

    kfixed[2,0]=kside2top
    kfixed[3,0]=kside2top
    kfixed[4,0]=kside2top
```

```
kfixed[5,0]=kside2top

kfixed[2,1]=kside2top
kfixed[3,1]=kside2top
kfixed[4,1]=kside2top
kfixed[5,1]=kside2top

kfixed[2,3]=kside2side
kfixed[2,5]=kside2side

kfixed[3,4]=kside2side
kfixed[3,2]=kside2side

kfixed[4,5]=kside2side
kfixed[4,3]=kside2side

kfixed[5,2]=kside2side
kfixed[5,4]=kside2side
ii=0
while ii<(len(envolope)):

    if ctype[ii]==1:

        kfixed[ii+6,2] = kpcb2side
        kfixed[ii+6,3] = kpcb2side
        kfixed[ii+6,4] = kpcb2side
        kfixed[ii+6,5] = kpcb2side

        kfixed[2, ii+6] = kpcb2side
        kfixed[3, ii+6] = kpcb2side
        kfixed[4, ii+6] = kpcb2side
        kfixed[5, ii+6] = kpcb2side

    elif ctype[ii]==2:
```

```
kbat2side=1/((((.5*htbus*(1+por))/(kal*(tempwall*wdbus)))+(1/kbolt)+(.5*wdboard/(kbat*envol
ope[ii]*wdboard)))
            kfixed[ii+6,2] = kbat2side
            kfixed[ii+6,3] = kbat2side
            kfixed[ii+6,4] = kbat2side
            kfixed[ii+6,5] = kbat2side

            kfixed[2,ii+6] = kbat2side
            kfixed[3,ii+6] = kbat2side
            kfixed[4,ii+6] = kbat2side
            kfixed[5,ii+6] = kbat2side
         ii = ii+1
      print('\nkfixed:\n\n', kfixed)

Conduction.conduction(envolope,tempwall,por,htbus,ctype,wallfixed,kcustom)
```

Heat Radiation

```
class Voltfcalc:
   def voltfcalc(hm,envolope):
      emiss = 0.9
      sb = 5.67e-8
      aboard = .09**2
      abside = .09*envolope
      atboard = 2*aboard+4*abside
      atop = .1**2
      wtop = .1
      wdboard = .09
      htop = .1
      awall = .1*htop
      radj = 0.235401618249073
      rpar = 0.529196763501854

      voltfarray=((6+len(hm)), (6+len(hm)))
      voltf = np.zeros(voltfarray, dtype=np.double)
```

```
hus = hm[0]
width1 = wdboard/hus
width2 = wtop/hus
pvar1 = ((width1**2)+(width2**2)+2)**2
xvar1 = (width2-width1)
yvar1 = (width2+width1)
qvar1 = ((xvar1**2)+2)*((yvar1**2)+2)
uvar1 = np.sqrt(4+xvar1**2)
vvar1 = np.sqrt(4+yvar1**2)
svar1 = uvar1*((xvar1*np.arctan(xvar1/uvar1))-(yvar1*np.arctan(yvar1/uvar1)))
tvar1 = vvar1*((2*np.arctan(xvar1/vvar1))-(yvar1*np.arctan(yvar1/vvar1)))
fucsp = (1/(np.pi*(width1**2)))*(np.log(pvar1/qvar1)+svar1-tvar1)
voltf[6,0] = fucsp*(aboard/atboard[0])
voltf[0,6] = fucsp*(aboard/atop)
voltf[0,2:6] = (1-fucsp*(aboard/atop))/4
voltf[2:6,0] = ((1-fucsp*(aboard/atop))/4)*(atop/awall)


hus = htop-hm[len(hm)-1]-envolope[len(hm)-1]
width1 = wdboard/hus
width2 = wtop/hus
pvar1 = ((width1**2)+(width2**2)+2)**2
xvar1 = (width2-width1)
yvar1 = (width2+width1)
qvar1 = ((xvar1**2)+2)*((yvar1**2)+2)
uvar1 = np.sqrt(4+xvar1**2)
vvar1 = np.sqrt(4+yvar1**2)
svar1 = uvar1*((xvar1*np.arctan(xvar1/uvar1))-(yvar1*np.arctan(yvar1/uvar1)))
tvar1 = vvar1*((2*np.arctan(xvar1/vvar1))-(yvar1*np.arctan(yvar1/vvar1)))
fucsp = (1/(np.pi*(width1**2)))*(np.log(pvar1/qvar1)+svar1-tvar1)
voltf[5+len(hm),1]=fucsp*(aboard/atboard[len(hm)-1])
voltf[1,(5+len(hm))]=fucsp*(aboard/atop)
voltf[1,2:6]=(1-fucsp*(aboard/atop))/4
voltf[2:6,1]=((1-fucsp*(aboard/atop))/4)*(atop/awall)
```

```
ii=0
while ii<(len(hm)-1):

    d = hm[ii+1]-(hm[ii]+envolope[ii])
    xbar = wdboard/d
    ybar=xbar

fij=(2/(np.pi*xbar*ybar))*((np.log((((1+xbar**2)*(1+ybar**2))/((1+xbar**2)+ybar**2))**.5)+(xbar*((1+ybar**2)**.5)*np.arctan(xbar/((1+ybar**2)**.5)))+(ybar*((1+xbar**2)**.5)*np.arctan(ybar/((1+xbar**2)**.5)))-(xbar*np.arctan(xbar))-(ybar*np.arctan(ybar)))
    voltf[ii+7,ii+6]=fij*(aboard/(2*aboard+4*abside[ii+1]))
    voltf[ii+6,ii+7]=fij*(aboard/(2*aboard+4*abside[ii]))
    voltf[ii+6,2:6]=(1-sum(voltf[ii+6,:]))/4
    voltf[2:6, ii+6]=(voltf[ii+6,2])*(atboard[ii]/awall)
    ii=ii+1

voltf[len(hm)+5,2:6]=(1-sum(voltf[len(hm)+5,:]))/4
voltf[2:6,len(hm)+5]=voltf[len(hm)+5,2]*(atboard[len(hm)-1]/awall)

print('4',voltf)

rem=sum(voltf[2:6,:]);
voltf[2,3]=rem[0]*radj;
voltf[2,4]=rem[0]*rpar;
voltf[2,5]=rem[0]*radj;

voltf[3,2]=rem[1]*radj;
voltf[3,4]=rem[1]*radj;
voltf[3,5]=rem[1]*rpar;

voltf[4,2]=rem[2]*rpar;
voltf[4,3]=rem[2]*radj;
voltf[4,5]=rem[2]*radj;

voltf[5,2]=rem[3]*radj;
voltf[5,3]=rem[3]*rpar;
```

```
    voltf[5,4]=rem[3]*radj;

    print('5',voltf)
  radcond=voltf
   radcond[0,:]=radcond[0,:]*atop*emiss*sb
    radcond[1,:]=radcond[1,:]*atop*emiss*sb
    radcond[2,:]=radcond[2,:]*awall*emiss*sb
    radcond[3,:]=radcond[3,:]*awall*emiss*sb
    radcond[4,:]=radcond[4,:]*awall*emiss*sb
    radcond[5,:]=radcond[5,:]*awall*emiss*sb

    jj=6
    while jj<len(radcond):
        radcond[jj,:]=radcond[jj,:]*atboard[jj-6]*emiss*sb;
        jj=jj+1

    print('\nvoltf:\n\n', voltf)
    print('\nradcond:\n\n', radcond)
```

Calculate Orbit and Solar flux

```
class orbital:
    def code(a,ec,taInit,jj,n,argP,inc,raan,acc):
        p=a*(1-ec**2) #semi-parameter
        mu = 3.986004e5
        Tm=np.arange(0,(301*200),200)
        Tm=Tm.astype(np.double) #time vector (56)
        ta=np.zeros((1,len(Tm)),dtype = np.double)      #vectors that will store true anomaly and
ecc. anomaly
        eA=np.zeros((1,len(Tm)),dtype = np.double)    #inital ecc. anomaly gives mean anomaly
        period=(2*np.pi)*np.sqrt((a**3)/mu)
        ta[0,0]=taInit
        eA[0,0]=np.arccos((ec+np.cos(ta[0,0]))/(1+ec*np.cos(ta[0,0])))
        mA=eA[0,0]-ec*np.sin(eA[0,0])
```

```
    print('',)
    ii=0
    while ii<len(Tm):
      mAt=mA+(2*np.pi)*(Tm[ii]/period)        #increments Mean anomaly (Tm(1)=0)
      if mAt>2*np.pi:
        mAt=mAt-(2*np.pi)
        mA=mA-(2*np.pi)
      eAOut,taOut = Ta.find(mAt,ec,acc)        #sends incremented value to True anomaly
finder
      eA[0,ii]=eAOut
      ta[0,ii]=taOut          #writes the ecc and true anomaly vectors
      ii=ii+1
    ri,rj,rk,vi,vj,vk = coe2.rv(p,ta,ec,mu,raan,argP,inc,len(Tm))
    return ri,rj,rk,vi,vj,vk

class Ta:
  def find(mAt,ec,acc):
    if mAt>np.pi:          #starting values depend on mean anomoly
      e0=mAt-ec/2
      err=(e0-ec*np.sin(e0)-mAt)/(1-ec*np.cos(e0))
      while err > acc:
        f=e0-ec*np.sin(e0)-mAt
        fp=1-ec*np.cos(e0)
        e0=e0-f/fp
        err=f/fp
    elif mAt<np.pi:
      e0=mAt+ec/2
      err=(e0-ec*np.sin(e0)-mAt)/(1-ec*np.cos(e0))
      while err>acc:
        f=e0-ec*np.sin(e0)-mAt
        fp=1-ec*np.cos(e0)
        e0=e0-f/fp
        err=f/fp
    else:                              # if Mean anomoly. is pi, eccentricity and true are also pi
      e0=np.pi
```

```
        eAOut=e0                    #ecc anomoly is the main output
        if eAOut>np.pi:
# true anomoly is a secondary calculation
            taOut=(2*np.pi)-np.arccos((np.cos(eAOut)-ec)/(1-ec*np.cos(eAOut)))
else:
            taOut=np.arccos((np.cos(eAOut)-ec)/(1-ec*np.cos(eAOut)))
        return eAOut, taOut

class solar:
    def flux(rsun,rsunabs,rsat,vsat,lit,solabs):
        print('',)
        fsurf = np.zeros((len(rsat), 6),dtype = np.double)

        rr=0
        while rr<len(rsat):
            rsatabs=np.sqrt((rsat[rr,0]**2)+(rsat[rr,1]**2)+(rsat[rr,2]**2))
            rsatu=rsat[rr,:]/rsatabs
            vsatabs=np.sqrt((vsat[rr,0]**2)+(vsat[rr,1]**2)+(vsat[rr,2]**2))
            vsatu=vsat[rr,:]/vsatabs # (unit vector of a satellite velocity)
            rsunAU=rsun/1.496e+8  # (sun radius converts back to Astronomical Unit)
            rsunu=rsunAU/rsunabs
            tutvar1=(jd-2451545)/36525
            msun=357.5291092+35999.05034*tutvar1
            sunabs=1.000140612-.016708617*np.cos(msun*np.pi/180)-
.000139589*np.cos((2*msun)*np.pi/180)
            flux=(1367.5/((rsunabs)**2))*(solabs)      #w/m**2

# side walls starting from nadir and going clockwise as viewed from negative Z
            s3=-1*rsatu
            s5=rsatu
            svar1=-1*vsatu
            s2=vsatu # Needs to be rewritten as an internal loop
            s4=np.cross(s2,s3)
            s6=-1*s5
```

```
if rr==300:
    print('rsunu',rsunu.shape)
    print(svar1.shape)
    print('svar1',svar1)
    print('s2',s2)
    print('s3',s3)
    print('s4',s4)
    print('s5',s5)
    print('s6',s6)
if lit[rr]==1: #check sun light state
    a1=np.dot(svar1,rsunu)
    if a1>0:
        fsurf[rr,0]=a1*flux
    a2=np.dot(s2,rsunu)
    if a2>0:
        fsurf[rr,1]=a2*flux
    a3=np.dot(s3,rsunu)
    if a3>0:
        fsurf[rr,2]=a3*flux
    a4=np.dot(s4,rsunu)
    if a4>0:
        fsurf[rr,3]=a4*flux
    a5=np.dot(s5,rsunu)
    if a5>0:
        fsurf[rr,4]=a5*flux
    a6=np.dot(s6,rsunu)
    if a6>0:
        fsurf[rr,5]=a6*flux
a=.2        #setting this arbitrarily for the moment at a reasonable value
emiss=1     #consider the earth as a blackbody
flag=0
if np.dot(rsatu,rsunu)>0:
    flag=1
tearth=293
qir=(5.67e-8)*(tearth**4)
```

```
            thetas=np.arccos(np.dot(rsatu,rsunu))
            qalbedo=flux*a*np.cos(thetas)
            fsurf[rr,2]=fsurf[rr,2]+(qir+qalbedo*flag)*.5
            fsurf[rr,0]=fsurf[rr,0]+(qir+qalbedo*flag)*.22
            fsurf[rr,3]=fsurf[rr,3]+(qir+qalbedo*flag)*.22

            fsurf[rr,1]=fsurf[rr,1]+(qir+qalbedo*flag)*.22
            fsurf[rr,5]=fsurf[rr,5]+(qir+qalbedo*flag)*.22
            #print('rr',rr)
            #print('qir',qir)
            #print('thetas',thetas)
            #print('qalbedo',qalbedo)
            #print(lit[rr])
            #print(fsurf[rr,:])
            rr=rr+1
        print('fsurf', fsurf)
        return fsurf

class sun:
    def v(jd):
        tutvar1=(jd-2451545)/36525
        lamdam=280.46+36000.771*tutvar1
        msun=357.5291092+35999.05034*tutvar1

lamdaecl=lamdam+1.914666*np.sin(msun*np.pi/180)+.019994643*np.sin((2*msun)*np.pi/180)
        rsunabs=1.000140612-.016708617*np.cos(msun*np.pi/180)-
.000139589*np.cos((2*msun)*np.pi/180)
        ep=23.439291-.0130042*tutvar1

rsun=np.array(([rsunabs*np.cos(lamdaecl*np.pi/180)],[rsunabs*np.cos(ep*np.pi/180)*np.sin(la
mdaecl*np.pi/180)],[rsunabs*np.sin(ep*np.pi/180)*np.sin(lamdaecl*np.pi/180)]),dtype =
np.double)
        rsun=rsun*1.496e+8

        return rsun, rsunabs
```

```
class sl:
    def check(rsat,rsun):
        lit=np.zeros((len(rsat),1), np.double)
        rsunabs=np.sqrt((rsun[0]**2)+(rsun[1]**2)+(rsun[2]**2))
        ii=0
        while ii<len(rsat):
            rsatabs=np.sqrt((rsat[ii,0]**2)+(rsat[ii,1]**2)+(rsat[ii,2]**2))
            rsatunit=rsat[ii,:]/rsatabs
            rsununit=rsun/rsunabs
            ssdot=np.dot(rsatunit,rsununit)
            angledif=np.arccos(ssdot*np.pi/180)
            if angledif>107:
                lit[ii]=0
            if angledif<=107:
                lit[ii]=1
            ii=ii+1
        return lit
class c:
    def gen(dcyc,lit,tgen):
        gen=np.zeros((len(dcyc[0]),len(lit)),dtype = np.double)
        ii=0
        while ii<len(lit):
            if lit[ii]==1:
                jj=0
                while jj<len(dcyc):
                    gen[jj,ii]=tgen[jj]*dcyc[1,jj]
                    jj=jj+1
            ii=ii+1
        return gen

ri,rj,rk,vi,vj,vk = orbital.code(a,ec,taInit,jj,n,argP,inc,raan,acc)
rsat=np.zeros((len(ri[0]),3),dtype = np.double)
rsat[:,0]=ri
rsat[:,1]=rj
rsat[:,2]=rk
```

```
vsat=np.zeros((len(vi[0]),3), dtype = np.double)
vsat[:,0]=vi
vsat[:,1]=vj
vsat[:,2]=vk

jd=calendar.julian(yr,mo,d,h,mnt,s)
rsun,rsunabs=sun.v(jd)
litcompact=sl.check(rsat,rsun)    #slcheckalt

fsurfcompact=solar.flux(rsun,rsunabs,rsat,vsat,litcompact,solabs)
fsurfcompact=np.nan_to_num(fsurfcompact)
fsurf=np.zeros((2000*len(fsurfcompact),6),dtype = np.double)
lit=np.zeros((2000*len(fsurfcompact),1),dtype = np.double)

qq=0
while qq<len(litcompact):
    lit[(1+(2000*(qq-1)))]
    qq=qq+1
kk=0
while kk<len(fsurfcompact[0]):
    if kk<len(fsurfcompact[0]):
        qvar1=fsurfcompact[kk,:]
        q2=fsurfcompact[kk+1,:]
        mm=0
        while mm<2000:
            fsurf[(mm+2000*(kk-1)),:]=((mm-1)*((q2-qvar1)/2000))+qvar1
            mm=mm+1
    kk=kk+1

gen = c.gen(dcyc,lit,tgen)
```