

**Mapping Comfort: An Analysis Method for
Understanding Diversity in the Thermal
Environment**

by

Amanda Laurel Webb

B.A., Yale University (2006)

Submitted to the Department of Architecture
in partial fulfillment of the requirements for the degree of
Master of Science in Architecture Studies

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

© Massachusetts Institute of Technology 2012. All rights reserved.

Author
Department of Architecture
May 24, 2012

Certified by
John Fernandez, MArch
Associate Professor of Architecture and Building Technology and
Engineering Systems
Thesis Supervisor

Accepted by
Takehiko Nagakura
Associate Professor of Design and Computation, Chair of the
Department Committee on Graduate Students

Thesis Readers

John Fernandez

Associate Professor of Architecture and Building Technology and Engineering Systems

Christoph Reinhart

Associate Professor of Architecture and Building Technology

Evelyn Wang

Associate Professor of Mechanical Engineering

Mapping Comfort: An Analysis Method for Understanding Diversity in the Thermal Environment

by

Amanda Laurel Webb

Submitted to the Department of Architecture
on May 24, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science in Architecture Studies

Abstract

Our thermal experience is never neutral. Whether standing near a cold window in the winter, or in the shade on a sunny day, we constantly experience a rich set of thermal stimuli. Yet, many of the tools used in professional practice to analyze and design thermal environments in buildings do not account for the richness of our thermal experience. This disconnect between our analysis tools and our experience results in buildings that use more energy than they should, and that leave occupants dissatisfied with their thermal environment.

This thesis seeks to bridge the gap between our thermal experience and our building thermal analysis tools. A unique methodology has been developed that produces mapping of thermal comfort parameters in all three spatial dimensions, as well as over time. Both heat balance and adaptive comfort indices have been incorporated into the methodology. An accompanying software program, called cMap, has been developed to illustrate the ways that this methodology can be used with existing energy analysis software and to demonstrate how it can fit into existing analysis workflows in professional practice.

Thesis Supervisor: John Fernandez, MArch

Title: Associate Professor of Architecture and Building Technology and Engineering Systems

Acknowledgments

I would like to express my sincere gratitude to my thesis readers, Professor John Fernandez, Professor Christoph Reinhart, and Professor Evelyn Wang. They have each provided me with continued guidance and encouragement, both for this thesis work and for my overall graduate career.

In addition to my readers, I would like to thank Kevin Settlemyre, who has acted as both a mentor and a friend. He has generously lent his knowledge of the building simulation field to my project, and provided me with excellent suggestions during the development of this thesis.

I would also like to thank MIT's Department of Architecture, especially the Building Technology Program, for providing me with invaluable financial and administrative support.

My close friends and family have had immense faith in me and my intellectual project during the past two years. To Vishal, thank you for your ever-present sense of humor. To Clare, thank you for your love and patience.

The inspiration for this thesis came largely from my time in professional practice. Many thanks to my former colleagues at Atelier Ten and to our clients, who always brought new and interesting challenges across my desk.

Finally, I would like to thank the MIT Womens Ice Hockey Club and our supporters. Your camaraderie has meant the world to me these past two years. Go Tech!

Contents

Nomenclature	15
1 Introduction	17
2 Background	25
2.1 Thermal Comfort Theory	25
2.2 Existing Thermal Comfort Analysis Methods and Software Tools	31
2.3 Thermal Comfort Analysis in Practice	37
3 Methodology	41
3.1 Mean Radiant Temperature and Comfort	41
3.2 Angle Factor Calculation	43
3.3 MRT Mapping	43
4 Software	45
4.1 cMap Workflow	45
4.2 cMap Outputs	47
4.3 cMap Capabilities and Limitations	48
5 Conclusions	61
5.1 Thesis Achievements	61
5.2 Discussion	62
5.3 Future Work	64
Bibliography	67
A cMap Source Code	71

List of Figures

1-1	Bayt al-Suhaymi, Cairo	18
1-2	Bayt al-Sinnari, Cairo	20
1-3	Cairo from the Citadel, 1860	21
1-4	Analysis feedback cycle	22
2-1	Comfort concept hierarchy	26
2-2	Fanger comfort model vs. adaptive comfort model	30
2-3	Control volume vs. discretized analysis methods	32
2-4	ASHRAE Thermal Comfort Tool	34
2-5	UC Berkeley Advanced Human Thermal Comfort Model	35
2-6	CFD results from San Francisco Federal Building	37
2-7	ASHRAE comfort zone	38
3-1	Angle factors	42
4-1	cMap interface	50
4-2	cMap workflow	51
4-3	cMap scatterplot, winter example	52
4-4	cMap contour heatmap X-Y slice, winter example	53
4-5	cMap contour heatmap Y-Z slice, winter example	54
4-6	cMap contour heatmap X-Z slice, winter example	55
4-7	cMap scatterplot, summer example range	56
4-8	cMap scatterplot, summer example peak	57
4-9	cMap contour heatmap X-Y slice, summer example	58
4-10	cMap contour heatmap Y-Z slice, summer example	59
4-11	cMap contour heatmap X-Z slice, summer example	60

List of Tables

2.1	ASHRAE 7-point scale	27
2.2	Existing thermal comfort software tools	33
3.1	Control volume variables vs. discretized variables	44
4.1	cMap and existing thermal comfort software tools	47

Nomenclature

f_{cl}	clothing area surface factor
F_{P-n}	angle factor between person P and surface n, °C
h_c	convective heat transfer coefficient, $W/m^2 \cdot K$
I_{cl}	clothing insulation, $m^2 \cdot K/W$
M	metabolic rate, W/m^2
p_a	water vapor partial pressure, Pa
T_{-n}	average outdoor temperature n days before the day in question, °C
t_a	air dry bulb temperature, °C
t_{cl}	clothing surface temperature, °C
T_{mrt}	mean radiant temperature, °C
T_m	mean monthly outdoor temperature, °C
T_n	temperature of surface n, °C
T_{rm3}	3-day running mean outdoor temperature, °C
T_{rm7}	7-day running mean outdoor temperature, °C
t_r	mean radiant temperature, °C
v_{ar}	relative air velocity, m/s
W	effective mechanical power, W/m^2

Chapter 1

Introduction

This thesis begins with the argument that Bayt al-Suhaymi, shown in Figure 1-1, is one of the most compelling buildings ever built. And that it is compelling primarily because it embodies the concept of thermal diversity. The building in Figure 1-1 is merely a proxy; we can extrapolate this argument to state that buildings that thermally diverse buildings are extremely compelling.

What is thermal diversity? Thermal diversity is simply terminology to package the intuitive truth that our thermal experience is not neutral. As Lisa Heschong writes in her short but seminal book *Thermal Delight in Architecture*:

Thermal information is never neutral; it always reflects what is directly happening to the body. This is because the thermal nerve endings are heat flow sensors, not temperature sensors. They can't tell directly what the temperature of something is; rather, they monitor how quickly our bodies are losing or gaining heat. (24).

The thermal environment, to us, is a world of opposites; our bodies are constantly evaluating whether the objects around us - the coffee cup we are holding, the window we are seated next to, the surrounding air - are hotter or colder than we are.

What does this intuitive truth about our thermal experience mean for buildings? In the first place, it means that there are a variety of elements in a building that control the rate of heat gain and loss from our bodies. Such elements include the surfaces that make up a building, the air inside of a building, and the objects within a building. In contrast, the

prevailing way of designing buildings for the past half-century or more has focused only on the volume of air in a building, rather than utilizing all of the components mentioned above. Typical space conditioning systems duct hot or cold air into a space to meet a particular setpoint temperature; the volume of air is assumed to be the same temperature throughout (the well-mixed assumption) and is intended to create a thermally neutral sensation the occupant is neither hot nor cold, is neither gaining nor losing heat. Buildings that embody thermal diversity acknowledge and exploit the fact that our thermal experience is diverse. Rather than just supplying hot or cold air, thermally diverse buildings also use surfaces and other objects within a building to help create a sensation of thermal comfort.

Figure 1-1: Bayt al-Suhaymi, in Cairo, provides an excellent example of a building that embodies thermal diversity. Photo by Hans Munk Hansen, from <http://www.davidmus.dk/assets/972/Bayt-al-Suhaymi-Cairo-Egypten.jpg>



Secondly, the goal in a thermally diverse building is not to create a perfectly uniform volume of air, nor to create a neutral sensation for building occupants. Instead, comfort is typically created by using contrast, providing some air movement on a hot day, for example,

or using a large mass wall to dampen peak temperatures. These buildings seek to 'take the edge off', facilitating heat loss or heat gain from the body just enough to provide relief.

Despite the prevailing concept of comfort as a neutral sensation, there is clear evidence that people do not necessarily want to feel neutral. Humphreys and Hancock (27) have shown through field studies that a persons desired thermal sensation is something other than neutral most of the time. Similarly, the adaptive comfort model (discussed in the Background section below), does not equate comfort with neutrality, but posits a comfort temperature correlated to the outdoor temperature.

If a thermally diverse building uses a variety of strategies to create a non-uniform thermal environment, how does the building shown in Figure 1-1 do this? A section through Bayt al-Suhaymi was not available, but a section through a similar building, Bayt al-Sinnari, is shown in Figure 1-2 below.

The section reveals several key strategies that the building uses to create a thermal environment that is both diverse and comfortable. First, thick, massive walls have a high capacity to store heat, helping to reduce peak surface and air temperatures. Second, the building reduces solar heat gain through small window openings, shaded by a dense wooden *mashrabiyya* screen. Third, the building is organized around a courtyard that provides self-shading, allowing cool night air to sink down into the courtyard and remain there until later in the day. Fourth, a windscoop or *malqaf* reaches up above surrounding buildings to direct airflow down into the building. Fifth, a fountain provides localized evaporative cooling in occupied spaces. None of these strategies attempts to create a uniformly conditioned volume of air. For more on the environmental strategies used in Mamluk and Ottoman era townhouses in Egypt, see Fathy (16) and Webb (49).

This thesis was originally conceived as a comprehensive study of the thermal environments in vernacular buildings. I originally became interested in thermal diversity through my undergraduate thesis, which focused qualitatively on the thermal environment in Bayt al-Suhaymi, and on the historical evolution and urban impacts of windscoops in Cairo, as shown in Figure 1-3 below. In the present work, I wanted to gain a quantitative understanding of the thermal environments created in buildings like Bayt al-Suhaymi. As architects, we often look to vernacular buildings for examples of how passive design can achieve thermal comfort. But there is little quantitative evidence illustrating the thermal conditions in these buildings and comparing to them to our current comfort standards. If we look to

Figure 1-2: Section of Bayt al-Sinnari, a similar building in Cairo, illustrating the features of the building that embody thermal diversity. Diagram by the author. Section of Bayt al-Sinnari from Maury et. al. Planche LXXVII (31)

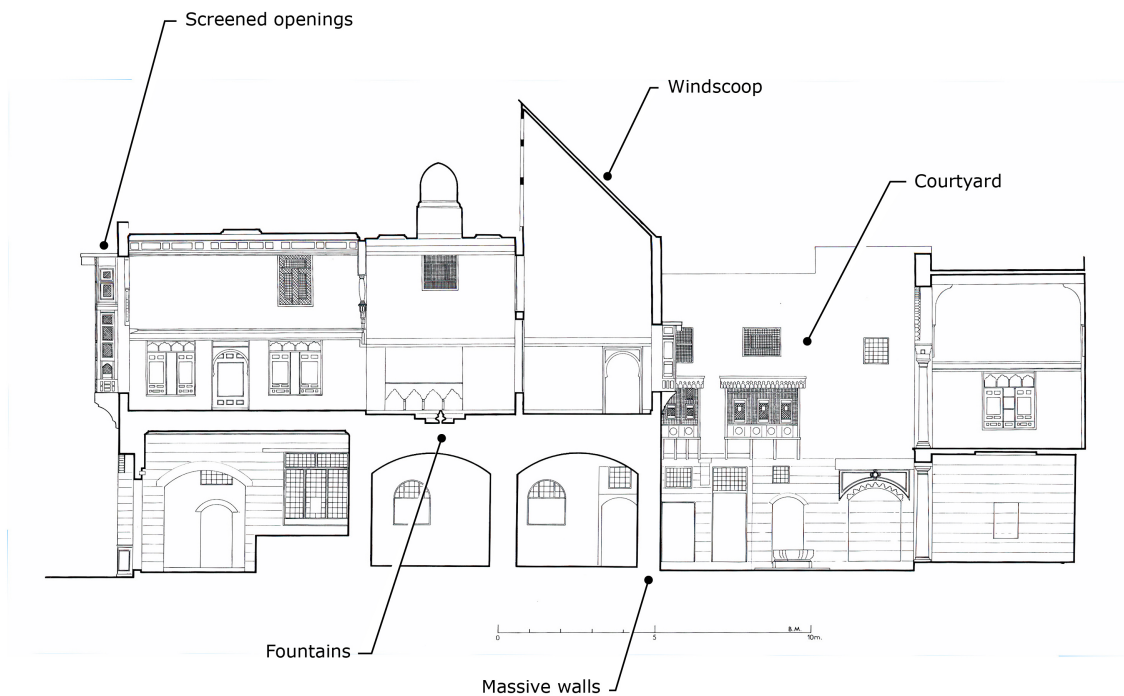


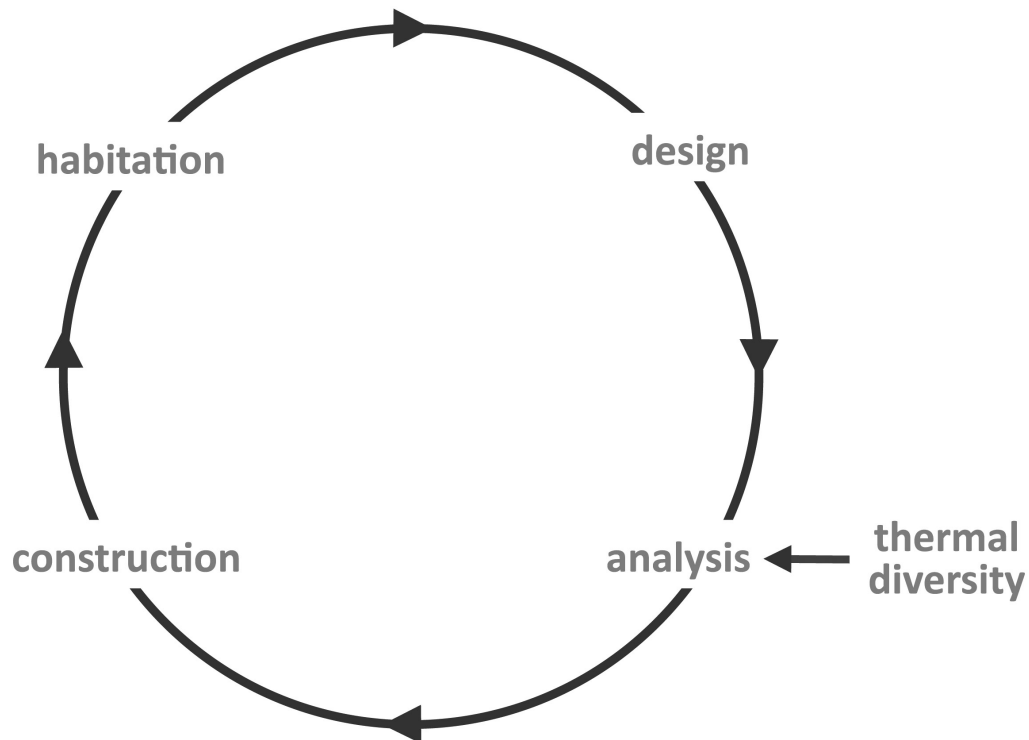
Figure 1-3: Photograph of Cairo from atop the Citadel, 1860. Photo by Frith (20).
A close inspection shows the abundance of windscoops across the roofs of the city



these buildings as precedent, we should know whether the conditions they create accord with our current comfort expectations.

I very quickly became sidetracked by the methods that allow us to quantify thermal diversity, and how those methods are used in professional practice. It turns out that analyzing thermally diverse spaces can be a complex process, and many of the analysis tools used in professional practice have limited capacity to perform such analysis. As a result, this thesis work is aimed at developing a methodology for analyzing thermal diversity that is viable for use in professional practice.

Figure 1-4: Diagram illustrating the key role that analysis tools play in the design process. What our analysis tools lack, our buildings will also lack. Diagram by the author.



Developing a methodology for use in professional practice is important because of the argument that began this thesis that thermally diverse buildings are more compelling. Compelling buildings simultaneously provide for us, educate us and endear themselves to us. They are the kind of buildings in which we recognize a core set of values, and that we want to preserve for future generations. In short, they are the kind of buildings that I

believe we should be building. Analysis methodology is critical to this process, as shown in Figure 1-4 below. Thermal analysis is an essential part of the building design process; if our design tools do not have the capability to analyze thermally diverse spaces, we simply won't build them. An important part of this thesis work is integrating the capability to analyze thermally diverse spaces into professional workflows.

Building thermally diverse spaces could significantly impact our built environment in two ways. First, thermally diverse spaces often use less energy. Rather than conditioning an entire volume of air, diverse spaces typically provide heating and cooling locally. There are clear energy impacts associated with uniformly conditioning a volume of air. Our concept of comfort—including our standards, our design goals, and our analysis methods—all need to be revised to reflect this. A 2008 issue of *Building Research Information* dedicated to the topic of comfort in a low carbon society put it thus:

””The systems of knowledge, and of design and construction that spawned comfort science and air-conditioned buildings, required cheap energy, a planetary atmosphere that could be disregarded, an ascendant engineering elite, technological regulation, powerful corporations, and cooperative governments. Those times are going, if not already gone. (44).

Second, thermally diverse spaces carry cultural significance that should not be lost. Consider the affection that we feel for sitting next to a roaring fire on a cold winter night, or enjoying the shade of a picnic pavilion on a hot summer day. Not only do these spaces conjure certain emotions, they also have the potential to create a distinctive urban form. Consider the unique skyline created by the forest of malqafs atop the roofs of Cairo, shown in Figure 1-3. As Heschong writes:

””The thermal environment also has the potential for such sensuality, cultural roles, and symbolism that need not, indeed should not, be designed out of existence in the name of a thermally neutral world.”” (24).

Chapter 2

Background

Over the past century, the issue of how and when we feel thermally comfortable has been researched, debated and incorporated into our building standards. This section provides context for my work by briefly answering the following questions:

- How do we, as architects and engineers, conceptualize comfort?
- How can we analyze thermal comfort? What methods and tools are available?
- How do we analyze thermal comfort in practice?

A short discussion on thermal comfort theory, existing thermal comfort analysis tools, and the use of these tools in professional practice follows.

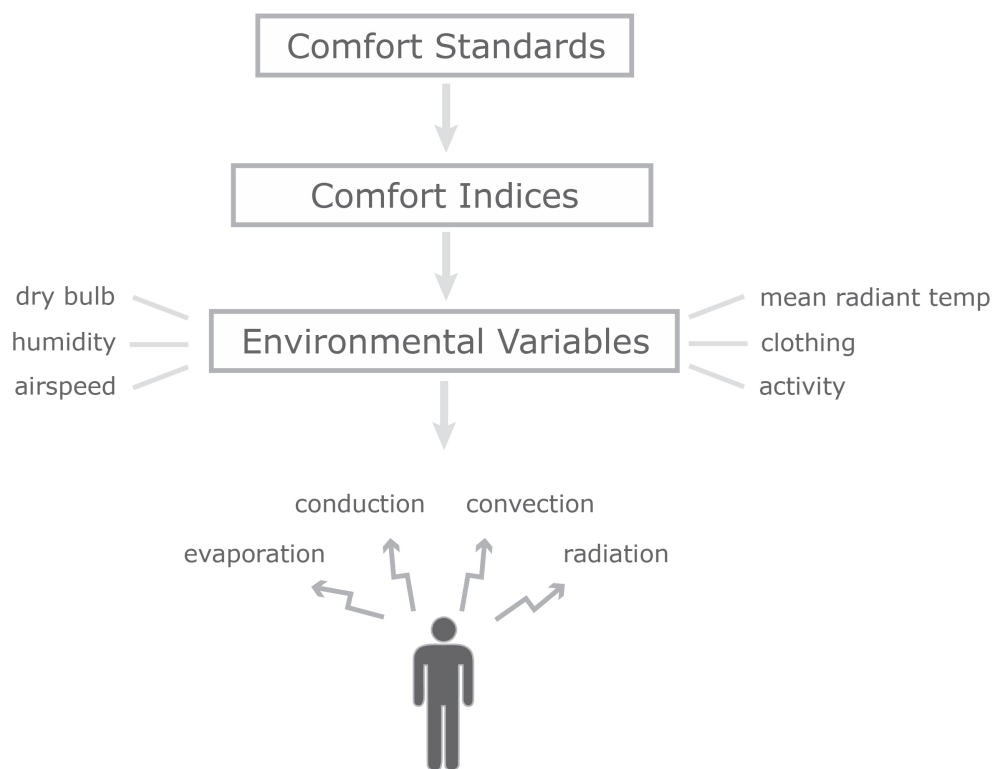
2.1 Thermal Comfort Theory

The study of human thermal comfort is inherently multidisciplinary. Understanding human comfort requires an exploration of both physical and psychosocial factors. These factors include human physiology and the way that it interacts with the built environment, the physics of the built environment, and cultural and behavioral thermal preferences.

We can quantify human thermal sensation at several scales, which, taken together form our current concept of human thermal comfort. At the scale of a single human body, we can evaluate the rate of heat transfer to and from the body. The rate of heat transfer is influenced by the characteristics of the surrounding environment, which can be broken

down into variables like the room air temperature, or relative humidity. These variables can be combined into a single, more convenient comfort index (operative temperature, for instance, is a combination of room air temperature and mean radiant temperature.) Our comfort standards then set acceptable ranges for these indices, establishing the bounds for what is comfortable and what is not.

Figure 2-1: Diagram illustrating the relationship between comfort standards, comfort indices, and heat transfer processes. Taken together, these form our hierarchical concept of thermal comfort. Diagram by the author.



Precisely where these comfort boundaries lie and which comfort index should be used has been the topic of a longstanding debate that still continues to evolve.

Many of the earliest thermal comfort indices were geared towards understanding the effects of extreme thermal conditions (especially extreme heat) on the human body. The early heating, ventilation and air conditioning (HVAC) industry evolved primarily in response to manufacturing needs, and early comfort studies were typically concerned with

setting safety limits for workers exposed to the often severe thermal conditions in factories. The dry bulb temperature and humidity were the main environmental variables explored in these studies. For more on the development of comfort indices and standards in the first half of the 20th century, see Fanger (15) and Cooper (11).

Fanger’s pioneering work in the late 1960s and early 1970s introduced a more thorough comfort index, called Predicted Mean Vote, or PMV. Fanger first derived a comfort equation based on a static heat balance for the human body. This equation accounted for six variables that Fanger asserted affected thermal comfort: dry bulb temperature, relative humidity, mean radiant temperature, airspeed, clothing level and activity level. Fanger then developed the PMV index by combining his comfort equation with experimental data. He seated his subjects in a climate chamber, where he changed each of the six comfort variables and recorded peoples votes on the seven point psycho-physical scale developed by the American Society of Heating, Refrigeration and Air Conditioning Engineers (ASHRAE).

Table 2.1: ASHRAE 7-point psycho-physical scale

cold	cool	slightly cool	neutral	slightly warm	warm	hot
-3	-2	-1	0	1	2	3

The resulting PMV index ranges in value from -3 to +3, and is calculated from the following equation (15) (3):

$$\begin{aligned}
PMV = & \underbrace{[0.303 \cdot (-0.036 \cdot M) + 0.028]}_{\text{thermal sensation coefficient}} \cdot \\
& \underbrace{\{(M - W)\}}_{\text{internal heat production}} \\
& \underbrace{-3.05 \cdot 10^{-3} \cdot [5733 - 6.99 \cdot (M - W) - p_a]}_{\text{heat loss through skin}} \\
& \underbrace{-0.42 \cdot [(M - W) - 58.15]}_{\text{heat loss by sweating}} \\
& \underbrace{-1.7 \cdot 10^{-5} \cdot M \cdot (5867 - p_a)}_{\text{latent respiration heat loss}} \\
& \underbrace{-0.0014 \cdot M \cdot (34 - t_a)}_{\text{dry respiration heat loss}} \\
& \underbrace{-3.96 \cdot 10^{-8} \cdot f_{cl} \cdot [(t_{cl} + 273)^4 + (t_r + 273)^4]}_{\text{heat loss by radiation}} \\
& \underbrace{+ f_{cl} \cdot h_c \cdot (t_{cl} - t_a)}_{\text{term description here}}
\end{aligned} \tag{2.1}$$

The variables t_{cl} , h_c and f_{cl} are determined from the following equations; t_{cl} and h_c are found by iteration.

$$\begin{aligned}
t_{cl} = & \{35.7 - 0.028 \cdot (M - W) - t_{cl} \cdot 3.96 \cdot 10^{-8} \cdot f_{cl} \cdot \\
& [(t_{cl} + 273)^4 + (t_r + 273)^4] + f_{cl} \cdot h_c \cdot (t_{cl} - t_a)\}
\end{aligned} \tag{2.2}$$

$$h_c = \begin{cases} 2.38 \cdot t_{cl} - t_a^{0.25} & \text{for } 2.38 \cdot t_{cl} - t_a^{0.25} > 12.1 \cdot \sqrt{v_{ar}} \\ 12.1 \cdot \sqrt{v_{ar}} & \text{for } 2.38 \cdot t_{cl} - t_a^{0.25} < 12.1 \cdot \sqrt{v_{ar}} \end{cases} \tag{2.3}$$

$$f_{cl} = \begin{cases} 1.00 + 1.290 \cdot I_{cl} & \text{for } I_{cl} \leq 0.078 \\ 1.05 + 0.645 \cdot I_{cl} & \text{for } I_{cl} > 0.078 \end{cases} \tag{2.4}$$

It is important to note that the PMV is a mean, intended to represent an average person

in a space. Therefore, a PMV of -0.3 means that an average person would feel somewhere between neutral and slightly cool under the specified conditions. Since not all people are alike, Fanger also developed the Predicted Percentage of Dissatisfied, or PPD index for provide a clearer measure of discomfort. The PPD index is related to the PMV index as follows:

$$PPD = 100 - 95 \cdot \exp(-0.03353 \cdot PMV^4 - 0.2179 \cdot PMV^2) \quad (2.5)$$

The PPD index suggests that there will always be some number of dissatisfied individuals. At a PMV of zero, i.e., when the average individual in the space is neutral (representing perfectly comfortable), 5% of individuals in the space will still be dissatisfied with the thermal environment.

The current comfort standard in the United States, ASHRAE Standard 55: Thermal Environmental Conditions for Human Occupancy, sets limits of $PPD < 10\%$ and $-0.5 < PMV < +0.5$ for acceptable thermal environments. (2)

In the 1990s, the adaptive comfort model was developed in response to Fanger’s work. In contrast to Fanger’s highly controlled climate chamber tests, de Dear and Brager surveyed occupants in actual buildings about their comfort preferences and measured the environmental conditions at the time of survey. They then developed the adaptive comfort model based on a linear regression of their results. According to their model:

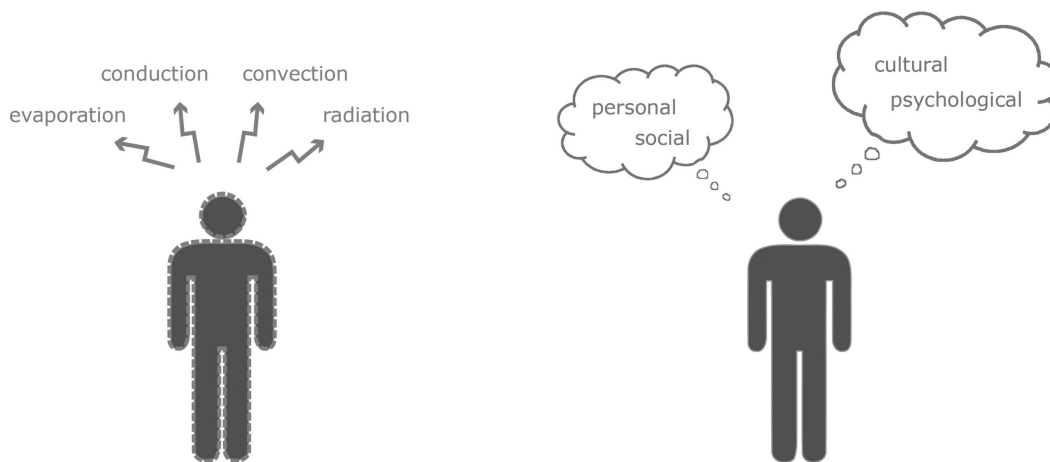
$$T_{comf} = 17.8^{\circ}C + 0.31 \times T_m \quad (2.6)$$

where T_m is the monthly average of the daily average outdoor dry bulb temperatures. The bounds for 80 percent acceptability and 90 percent acceptability are at $\pm 2.5^{\circ}C$ and $\pm 3.5^{\circ}C$, respectively. deDear and Brager’s work was conducted as part of ASHRAE RP-884 and the full scope of their work can be found in deDear (14). Additional summaries of the adaptive model can be found in Brager (8), Humphreys (28), Nicol (35), and de Dear (12)(13).

Whereas Fanger’s PMV index uses six variables to predict comfort, the adaptive model

suggests that comfort is correlated with only two variables - the operative temperature and the mean outdoor temperature. In addition to these three measurable variables, the adaptive standard presupposes that there are psychological, cultural, and personal factors that contribute to an individual's perception of thermal comfort.

Figure 2-2: Conceptual illustration of the difference between the Fanger comfort model (at left below) and the adaptive comfort model (at right below). Fanger's model is based on a heat balance method; the adaptive comfort model assumes a set of psychosocial factors underlie comfort preferences. Diagram by the author.



While ASHRAE Standard 55 includes guidance on both the Fanger model and the adaptive model, the standard states that the adaptive model may only be used in spaces that have operable windows and that do not utilize a mechanical cooling system.

In the past decade, variations to the adaptive comfort model have emerged in different countries. These models differ in two main ways from the original adaptive model that has been incorporated into ASHRAE Standard 55. First, they use different statistical sample sets. Whereas the original adaptive model used a global database of buildings, the adaptive model that has been incorporated into European standard EN 15251 used a database on European buildings only. Second, the models use different mean outdoor temperatures. Whereas the original adaptive model uses the monthly average of the daily average outdoor dry bulb temperatures, the models incorporated into EN 15251 and Dutch standard NPR-CR-1752 uses an exponentially weighted running mean of previous daily outdoor temperatures.

The adaptive model that has been incorporated into the European Standard EN-15251 states that:

$$T_{comf} = 18.8^{\circ}C + 0.33 \times T_{rm7} \quad (2.7)$$

where T_{rm7} is calculated as:

$$T_{rm7} = T_{-1} + 0.8T_{-2} + 0.6T_{-3} + 0.5T_{-4} + 0.4T_{-5} + 0.3T_{-6} + 0.2T_{-7}/3.8 \quad (2.8)$$

The adaptive model that has been incorporated into the Dutch Standard NPR-CR-7251 states that:

$$T_{comf} = 17.8^{\circ}C + 0.31 \times T_{rm3} \quad (2.9)$$

where T_{rm3} is calculated as:

$$T_{rm3} = T_0 + 0.8T_{-2} + 0.4T_{-3} + 0.2T_{-4}/2.4 \quad (2.10)$$

Details on the European and Dutch variations to the original adaptive comfort model are provided in Borgeson (6), McCartney (32), Nicol (36), and van der Linden (47).

2.2 Existing Thermal Comfort Analysis Methods and Software Tools

There are a number of existing software programs that provide explicit support for evaluating thermal comfort in a space. These tools vary widely in their scope, capabilities and

limitations. They can be usefully categorized based on the following:

- Analysis method. Does the tool use control volume analysis or discretized analysis?
- Scale of focus. Does the tool provide analysis of a human body, or of a point in space?
- Spatial output. Does the tool provide spatial mapping of comfort analysis results over an entire space, or does it only provide the comfort conditions at a single point?
- Temporal output. Does the tool provide comfort analysis results over a range of time, or does it only provide the comfort conditions at a single point in time?

Perhaps the most important distinction between these tools is whether or not they use control volume or discretized analysis methods. Control volume analysis draws a boundary around a volume and solves for the inputs and the outputs. In contrast, discretized methods create a grid of points within the object of interest, and solve for the values at each grid point. Because these two methods set up the analysis problem in very different ways, each method has a very different set of possible outputs.

Figure 2-3: Illustration depicting the conceptual differences between a control volume analysis approach (at left below) and discretized analysis methods (at right below). Diagram by the author.

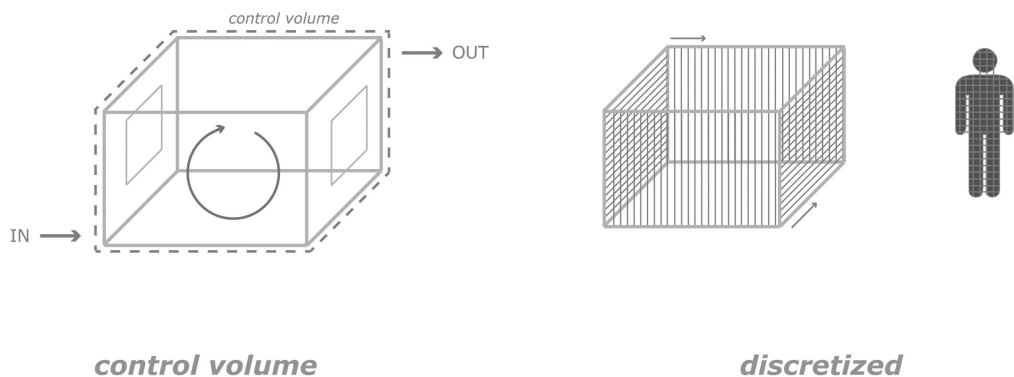


Table 2.2 summarizes these primary characteristics for several existing comfort analysis tools.

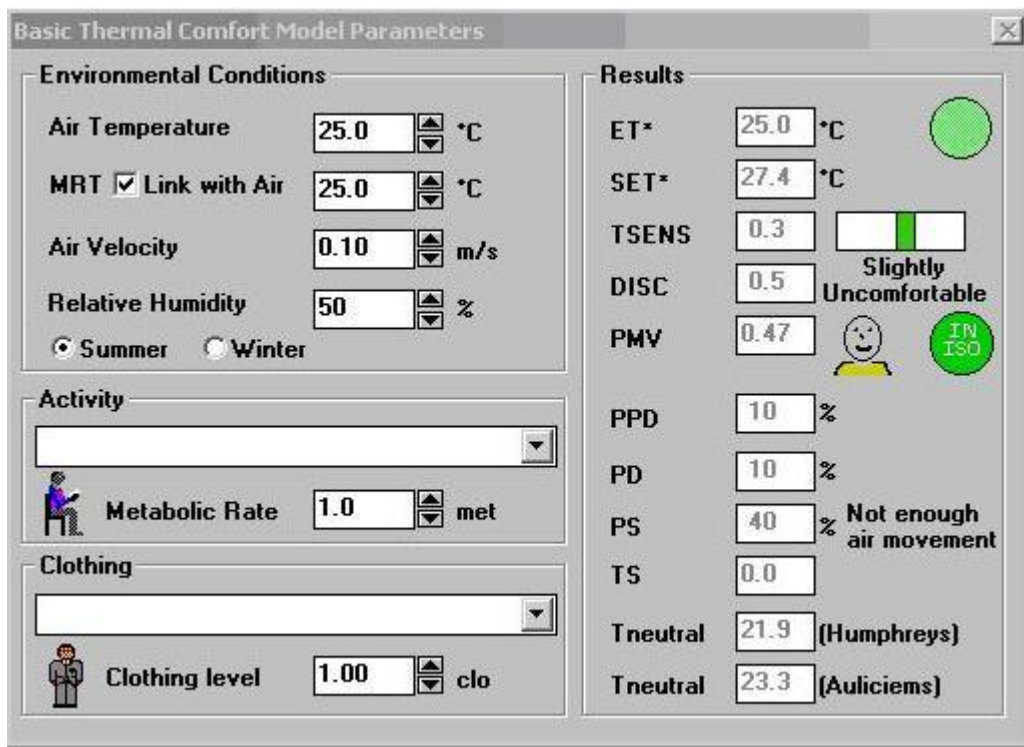
Table 2.2: Characteristics of existing thermal comfort software tools

Tool Name	Method	Scale	Spatial	Temporal
EnergyPlus	control volume	room	point	range
ASHRAE Comfort Tool	control volume	space	point	point
UC Berkeley AHTCM	discretized	body	space	point
Arup ROOM	discretized	room	space	point

EnergyPlus (of Energy), is a whole building energy analysis program used by architects, engineers, and researchers to model building energy and water use at each hour of a typical year. This software is maintained by the U.S. Department of Energy and is free for download. The software provides thermal comfort outputs as part of its People object. The user can specify the mean radiant temperature calculation in three different ways, depending on how much information the user is willing to provide: Zone Averaged, Surface Weighted, and Angle Factor. If the user chooses the Angle Factor method, the user must calculate and input the angle factors. EnergyPlus does not perform this calculation. EnergyPlus can provide analysis output based on several different comfort models, including both the Fanger model and the ASHRAE Standard 55 adaptive model. More information on thermal comfort analysis using EnergyPlus can be found in the programs Input-Output Reference Guide and in the program’s Engineering Reference, available as part of the program download or in the documents section of the program website.

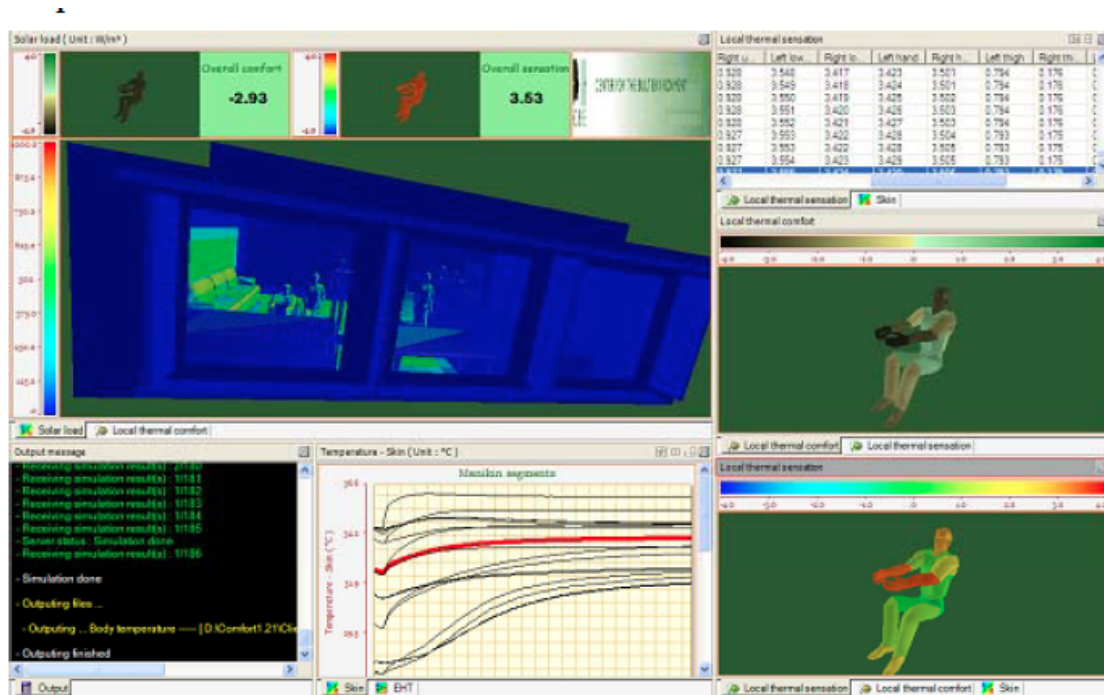
ASHRAE developed its own Thermal Comfort Tool software in the mid 1990s, as part of research project in RP-781 (18) (19). This tool is designed to help HVAC engineers determine whether their design is in compliance with ASHRAE Standard 55. The tool is available for purchase from ASHRAE. In contrast to EnergyPlus, the ASHRAE Thermal Comfort Tool does not provide information about building energy use or thermal conditions; the user must supply the thermal conditions of interest and the tool calculates whether or not PMV or adaptive comfort criteria are met. The tool is only able to provide information about comfort conditions at one set of criteria, that is, at a single point in space and at a single point in time. Version 2.0 of the ASHRAE Thermal Comfort Tool has recently been released. This updated version includes a more detailed mean radiant temperature calculator than the previous version, however, the user still must supply the angle factors to the software. (ASHRAE)

Figure 2-4: Screenshot of the ASHRAE Thermal Comfort Tool user interface, version 1.0



UC Berkeley's Advanced Human Thermal Comfort Model (AHTCM) is a detailed model of the human body and its interactions with the surrounding thermal environment. The model can predict comfort and thermal perception for the human body as a whole, as well as for specific body parts. Like the ASHRAE Thermal Comfort Tool, the AHTCM cannot predict building energy use, and only provides results based on a specific point in time. In contrast to the ASHRAE Thermal Comfort Tool, the AHTCM provides a rendering of a person in a space that includes spatial mapping of temperatures and thermal comfort parameters. The AHTCM is maintained by the Center for the Built Environment at UC Berkeley, and is not available for public use or purchase. See Huizenga (25). A number of similar computational thermal models of the human body have been discussed by Yang (51), van Treek (48), and Rees (39).

Figure 2-5: Screenshot of the UC Berkeley Advanced Human Thermal Comfort Model user interface. Image from Huizenga (26)



The software ROOM is proprietary tool that has been developed by the engineering firm Arup over the past 30 years. ROOM provides all-in-one energy analysis, radiation and shading analysis, and thermal comfort analysis. The thermal comfort module produces 2-D spatial mapping of thermal comfort conditions at a set vertical distance from the floor.

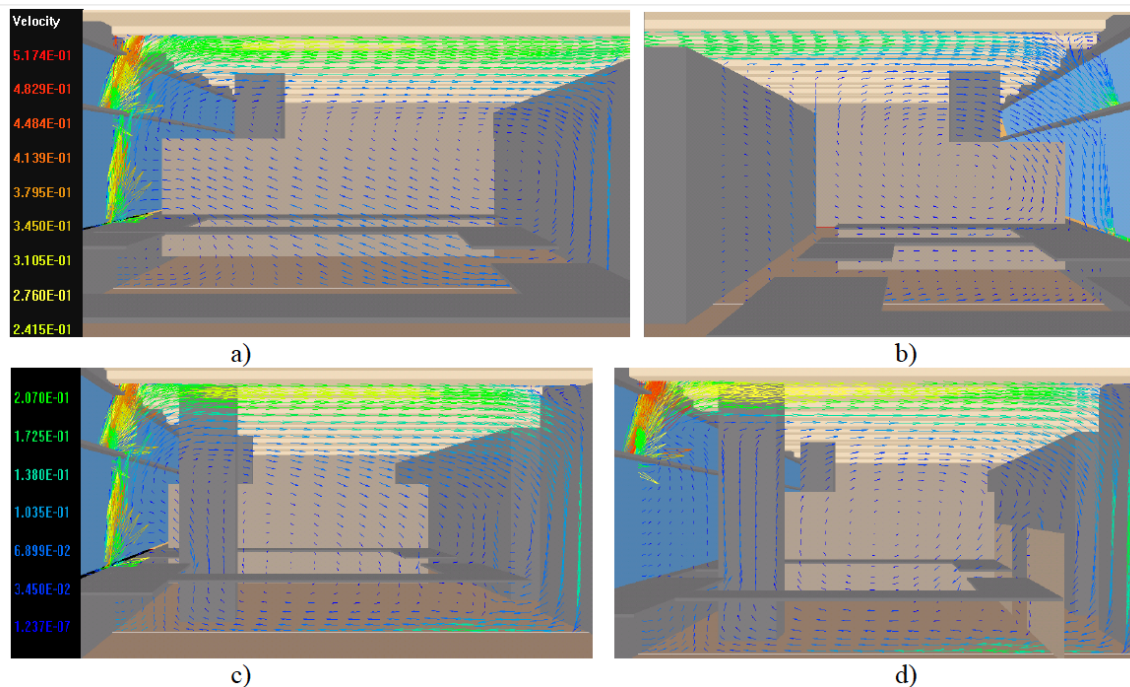
ROOM can produce these results for an average day for each month of the year, but it is not able to produce results for every hour of the year. For information on ROOM, please see White (50). The author also received a demonstration of the ROOM software from Jauni Novak, a Graduate Mechanical Engineer in the Arup Los Angeles office (personal communication, April 11, 2012).

While the focus of this thesis is on indoor thermal comfort analysis, it is worth mentioning the maturing body of literature and range of thermal comfort analysis tools for outdoor thermal comfort analysis. Both the RayMan (30) and ENVI-MET (Bruse) programs apply similar comfort analysis methods to outdoor thermal comfort problems.

While it is an analysis method and not a software tool, computational fluid dynamics (CFD) is increasingly being used to provide detailed analysis of the thermal environment in buildings. CFD utilizes discretized analysis methods to solve the Navier-Stokes equations for fluid flow. As a result, CFD can map temperature and velocity fields throughout a given air volume, as shown in Figure 2-6 below. A variety of studies in recent years have utilized CFD methods to provide spatial mapping of thermal comfort conditions in a space, and these are discussed in more detail in the following section of this paper. While CFD provides information about air temperature and velocity fields, it does not determine other comfort variables, e.g., mean radiant temperature, clothing level, therefore does not explicitly provide information about thermal comfort conditions. While the use of CFD is increasing in professional practice, it is a time-intensive and expertise-intensive process, and is currently much less common than whole building energy modeling. For more on the use of CFD for building analysis applications, see Srebric (45)

In addition to the analysis tools and methods discussed above, two research projects serve as a useful precedent for this thesis. Herkel et. al. (42) developed an interactive tool for the visualization of thermal comfort conditions in a space. This tool produces 2-D slices of comfort conditions within a perspective view of a space. Gan (21) (22) developed a methodology for the full evaluation of thermal comfort in a space, and produced a series of thermal comfort spatial maps using this methodology.

Figure 2-6: CFD results from the design phase analysis of the San Francisco Federal Building depicting the air velocity field. Image from Haves (23)



2.3 Thermal Comfort Analysis in Practice

Despite the variety of thermal comfort analysis tools and methods discussed above, there are a number of major barriers to their use in professional practice:

Too time-intensive Tools might be too time-intensive because their computational methods take a long time, e.g., CFD, because the inputs to the tool are non-trivial to determine, e.g., angle factors in EnergyPlus, or because they are an entirely separate tool and do not fit within a company's existing analysis workflow. In professional practice, time is extremely valuable and analysis that cannot be performed quickly, or sold to a client as important will simply not happen.

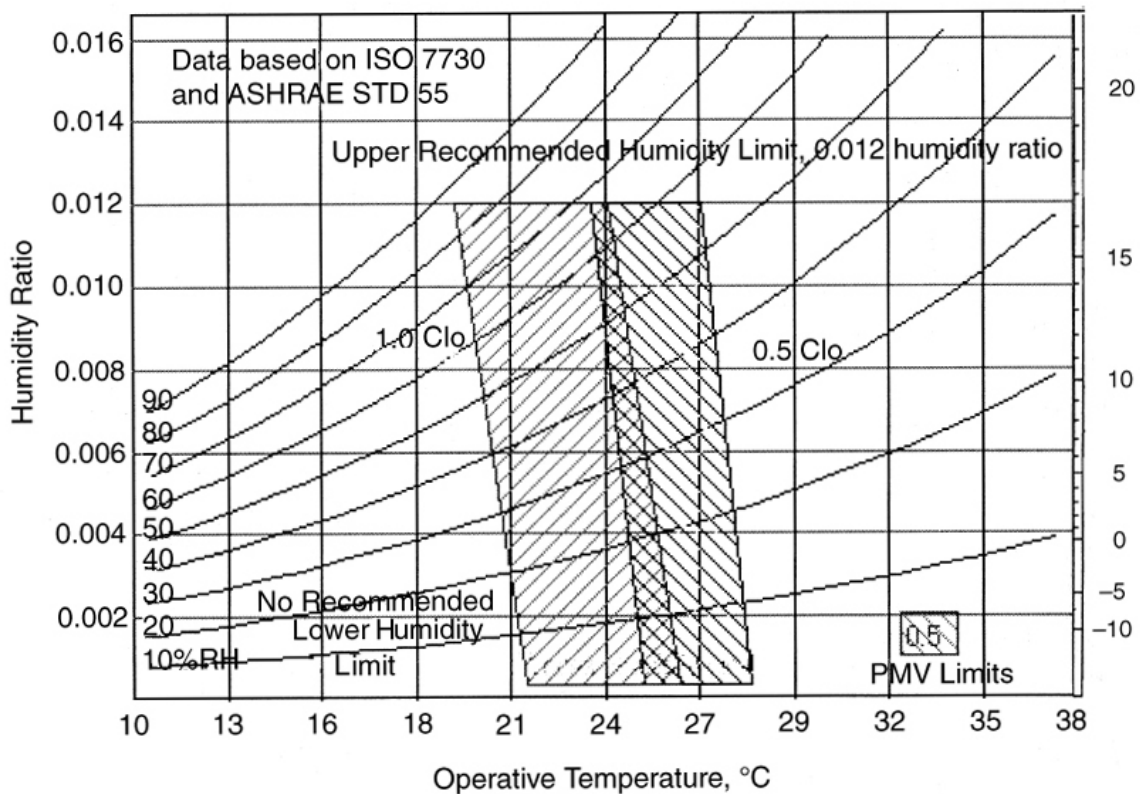
Too experience-intensive Tools or methods that require advanced inputs or user knowledge, e.g., CFD, require expert users to ensure quality results. Design firms may have more difficulty finding prospective employees with such training, and may not want to bear the cost of training existing employees in these skills.

Unavailable Only one of the indoor thermal comfort tools listed above is publicly available

at no cost - EnergyPlus. All of the other tools listed are not publicly available, or are available for a fee.

As a result, these tools are not commonly used in professional practice. Often, comfort is simply approximated by evaluating zone air dry bulb temperature (or operative temperature, if available) and relative humidity using a whole building energy model, and comparing the calculated values to the boundaries of the comfort zone shown in Figure 5.2.1.1 of ASHRAE Standard 55 (2), and reproduced below.

Figure 2-7: Diagram plotting the humidity ratio (y-axis), the operative temperature (x-axis) and delineating the ASHRAE Standard 55 Comfort Zone. Image from ASHRAE Standard 55 (2)



The low occurrence of thermal comfort analysis in practice suggests the need for a cohesive comfort analysis process. Ideally this process would happen at multiple stages during the building design process, and could focus on several different scales - the whole building, a single window, a particular part of the human body - depending on the project

needs.

While there are no existing professional resources, e.g, guides or standards, suggesting such a process, several research studies outline a comfort analysis process as a consequence of their work.

Negrao (34) evaluated thermal comfort for a four-zone sample building in Brazil. He first used a nodal network to evaluate thermal conditions for each zone as a whole for every hour of the year, and then coupled the nodal network results with CFD analysis. The CFD analysis was employed in one of the zones for two points in time – a heating design condition and a cooling design condition. A 7-node model of a human shape was used to evaluate thermal comfort. Spatial mapping of PMV values were produced at one height in the x-y direction.

van Treeck (48) performed an initial simulation at the coarse level, running a whole building energy simulation for the whole year to identify periods where comfort temperatures are not satisfied in a building zone as a whole. The results from the critical periods of potential discomfort are then imported into the "virtual climate chamber" for local analysis using a computational thermal manikin.

Published analysis from the design phases for the San Francisco Federal Building presents perhaps the best example of a comfort analysis process used in practice. Several portions of the building would not have mechanical cooling systems and the team needed to demonstrate that the natural ventilation scheme would produce comfortable indoor conditions. The analysis process first used EnergyPlus to evaluate zonal conditions using a nodal network model. CFD was then used to provide a more detailed analysis, and to help refine opening sizes. For a summary of this analysis work, see Haves (23). For a detailed case study on the design process, see Meguro (33)

What is common to all of these examples is the need for analysis at a zonal (or "coarse") level first, to understand the global comfort conditions for the building. This type of analysis can tell us, over the course of a year, what the thermal conditions are for each zone as a whole. These example all also have a second analysis at a finer level. This may be finer analysis using a human body model, or using a spatially resolved model of a room. While the coarse analysis tells us when a space might be uncomfortable, the finer analysis tells us more precisely where and why.

Chapter 3

Methodology

This thesis seeks to remedy some of the issues with existing thermal comfort analysis tools and to remove the barriers to the use of thermal comfort analysis in professional practice. The goal of this thesis is to develop an analysis methodology that is able to:

- Map thermal comfort parameters over space
- Plot thermal comfort metrics over time, both at a specific hour of the year, and averaged over a specified period
- Fit easily into existing energy analysis workflows in professional practice, i.e., is a computationally lightweight method

3.1 Mean Radiant Temperature and Comfort

All of the comfort indices discussed in section 2.1 have two variables in common: dry bulb temperature and mean radiant temperature, which accounts for radiative exchange between a person and the surroundings. Using one of these two variables, then, as a basis for this methodology has the added benefit of being able to apply it to assess comfort for a range of different comfort standards.

A key objective of this methodology is to be able to map comfort parameters over space. Mapping the dry bulb temperature field in a space is a relatively complex process, generally requiring CFD analysis. Mean radiant temperature, on the other hand, is dependent on

location by its very definition. While it is not a trivial process to plot mean radiant temperature as a function of space, it is less computationally intensive than CFD. Therefore, mean radiant temperature has been selected as the basis for this methodology.

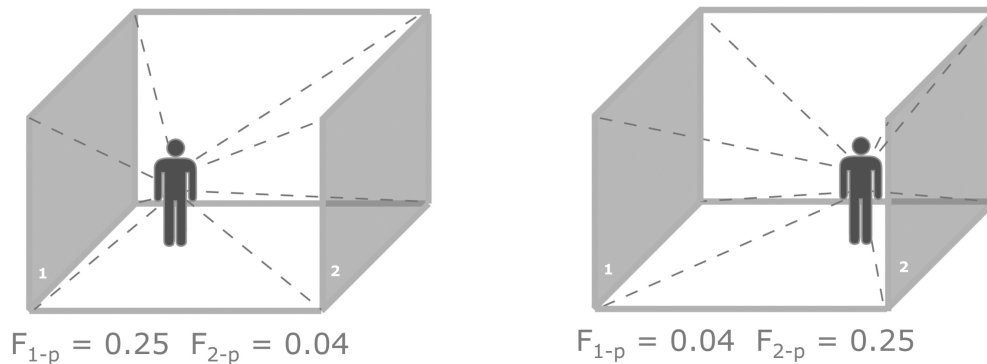
It is worth noting the general importance of mean radiant temperature in the literature. Fanger (15) devoted a significant portion of his work to the calculation of mean radiant temperature. Powitz (38) suggests that radiant temperature asymmetries are a chief cause of comfort complaints in actual buildings.

Mean radiant temperature is calculated using the following equation:

$$T_{mrt} = T_1 F_{P-1} + T_2 F_{P-2} + \dots + T_n F_{P-n} \quad (3.1)$$

These angle factors are highly dependent on the location of a person in a space. Since these angle factors must sum to unity, they are effectively how much of a each surface a body "sees". If a body is standing closer to a surface or if the surface is large, the body will "see" more of that surface than other surfaces in the space. Vice versa for surfaces that are smaller or further away. This concept is illustrated in the figure below.

Figure 3-1: Diagram illustrating angle factors and their dependence on location. In both images below, surface 1 is the left shaded surface and surface 2 is the right shaded surface. Diagram by the author.



3.2 Angle Factor Calculation

While the calculation of mean radiant temperature is relatively straightforward, calculating the angle factor component is much more complex. The angle factor is a function of the surface area and posture of the human body, and the location of a human body within a space. A full derivation of the angle factor is given in Fanger (15). Using the results from a set of experiments, Fanger produced a set of nomograms to allow for the quick calculation of angle factor.

Unfortunately, nomograms cannot be used for computational methods, such as the one proposed here. Cannistraro (10) developed an algorithm for the determination of angle factors based on curve-fitting Fanger's nomograms. This algorithm has been used in subsequent research (5) and has been used for the calculation of angle factors in this work.

3.3 MRT Mapping

The proposed methodology can be described in 4 steps:

Step 1 Discretize the space with a 3-D set of gridpoints

Step 2 Calculate the view factors at each gridpoint

Step 3 From the view factors, calculate the mean radiant temperature at each gridpoint

Step 4 Combine the control volume values for the other comfort parameters with the mean radiant temperature at each point to calculate the comfort indices at each point.

It is important to clarify what is being calculated as a function of space. While mean radiant temperature is being plotted as a function of space, all of the other comfort parameters are being pulled from the control volume method. But, because mean radiant temperature is being plotted as a function of space, the thermal comfort indices can be plotted as a function of space. See Table 3-1 below for a summary. This is a kind of hybrid control volume/discretized method, where some of the values are provided via control volume analysis and mean radiant temperature and comfort indices are discretized.

While this method doesn't plot full temperature and velocity fields, like CFD does, this method is faster and much less computationally intensive, and provides a first order approximation of how comfort changes over the space.

Table 3.1: Listing of which analysis results are determined using control volume methods and which are discretized

Discretized	Control Volume
mean radiant temperature	dry bulb temperature
all comfort indices	relative humidity
	airpeed
	clothing level
	activity level

It should be noted that the use of mean radiant temperature plotting to produce spatial comfort plotting in three dimensions is a logical extension of previous thermal comfort work. Cannistraro suggested that one of the greatest potential results of their algorithms was the ability to be able to determine mean radiant temperature at every point in a room and plot 'iso-comfort' lines (10). Fanger himself produced thermal comfort plots as a part of the full assessment of the thermal environment. See Figure 28 in Fanger (15).

Chapter 4

Software

A software program called Thermal Comfort Spatial Map (cMap) has been developed to demonstrate one possible way to implement the methodology discussed in the previous section. In addition, the cMap outputs are designed to suggest a clear thermal comfort analysis workflow to be used in professional practice.

Please note that all of the figures discussed in this section are located at the end of the chapter.

The cMap software includes a backend script that performs the thermal comfort calculations, and a GUI that produces a series of plots. The GUI is interactive, allowing the user to quickly scroll through the results for different periods of the year. A screenshot of the cMap interface is shown in Figure 4-1 below, highlighting the input, navigation, and output areas.

The software has been written in Python. Numpy and Scipy were used to perform most of the comfort calculations. Matplotlib was used to produce the plots. wxPython was used to create the GUI. The full source code for the software is included in Appendix A.

4.1 cMap Workflow

cMap has been built to work with the EnergyPlus whole building energy simulation software. cMap does not create or run the EnergyPlus model this model must exist already, presumably having been built previously as part of a projects energy modeling needs. Running cMap involves the following three basic steps from the user:

Step 1 The user inputs the location of the EnergyPlus .idf (Input Data File) and .csv (Comma Separated Value results file) file into cMap. cMap parses these files for location on the room geometry and thermal conditions, e.g., surface temperatures, dry bulb temperature, etc. cMap then creates a 3-D set of gridpoints within the space and calculates mean radiant temperature based on the methodology described in the previous section.

Step 2 The user selects the desired thermal comfort metric and timeframe. cMap is capable of producing the results for the following comfort indices and standards:

- Operative Temperature
- Mean Radiant Temperature
- Predicted Mean Vote
- Predicted Percentage Dissatisfied
- ASHRAE 55 Adaptive Comfort
- EN 15251 Adaptive Comfort
- NPR-CR 7251 Adaptive Comfort

The user can request any of these results at a single hour of the year, or averaged over a particular time period.

The comfort indices are calculated based on the equations listed in section 2.1.

Step 3 Based on the users selected metric and timeframe, cMap creates two types of plots. First, a scatterplot is created that plots the room averaged comfort value, for each of the hours of the year. These values are plotted against the acceptable boundaries for the selected metric. Second, a spatial contour heatmap is created that plots these comfort values within the space. cMap produces 2-D slices through the space, and the user can select 2-D slices in any direction (X-Y, X-Z, X-Y) and at any location in the space. These plots can be exported from the program as .png files, which can then be used in a report or presentation.

The steps above are all relatively quick. The software has been designed in anticipation of the user iterating Steps 2 and 3 many times to understand how the comfort conditions change over space and time throughout the year.

A diagram of the cMap software process is shown in Figure 4-2 below.

Various output examples for a summer and winter analysis case are shown in Figures 4-3 through 4-12. These cases are based on a single zone model with a north-facing window and ASHRE 90.1 code minimum envelope properties. The analysis was run using a Boston, MA climate file.

In order to perform the comfort calculations, cMap uses a hybrid control volume and discretized method. The mean radiant temperature is calculated using a discretized method; cMap creates a set of gridpoints within the space and calculates the view factors and mean radiant temperature at each of those points. All of the other variables required to perform the comfort calculations dry bulb temperature, airspeed are pulled from the EnergyPlus .csv results file. All of the other variables, therefore, are calculated using the control volume method.

It is critical to note the importance of mean radiant temperature here. This hybrid method can provide spatial mapping for all of the different comfort variables only because all of them are a function of mean radiant temperature. Since we can map mean radiant temperature as function of space, we can therefore also map PMV, PPD, and the Adaptive models as a function of space.

The capabilities of the cMap software are summarized in comparison to other thermal comfort analysis tools in Table 4-1 below.

Table 4.1: cMap characteristics in comparison to existing thermal comfort software tools

Tool Name	Method	Scale	Spatial	Temporal
EnergyPlus	control colume	room	point	range
ASHRAE Comfort Tool	control volume	space	point	point
UC Berkeley AHTCM	discretized	body	space	point
Arup ROOM	discretized	room	space	point
cMap	discretized	room	space	range

4.2 cMap Outputs

The cMap software and outputs have been intentionally designed to suggest a larger thermal comfort analysis workflow that answers the question How comfortable is this space? The

software has been designed according to the visual information seeking mantra: Overview first, then zoom and filter; details on demand. (41).

Overview first While spatial mapping is important, in order to determine how comfortable a space is, a user would first want to see the average comfort value for the space plotted over the entire period of interest. This period may be all of the occupied hours of the year, or a peak condition. This helps give the user an idea of when the space, on average, meets the comfort criteria, and when it does not. cMap provides a scatterplot of the room averaged comfort condition for all of the user-specified hours of interest.

Zoom and filter Spatial mapping is particularly useful once the user has an idea of when, on average, the space meets or exceeds the acceptable comfort bounds. A second tab on the interface produces a contour heatmap plot that maps comfort variables in all three dimensions in the space.

Details on demand Users may want to know the values for all of the environmental variables at the time of interest. A section of the navigation bar displays all of these relevant parameters indoor and outdoor dry bulb temperature, relative humidity, airspeed, clothing level and activity level at the selected time period of interest. The user can choose to display or hide this menu, depending on the desired level of detail.

The design of the cMap software has also attempted to best practices for data visualization wherever possible. (46)(52)(40)(7).

4.3 cMap Capabilities and Limitations

cMap makes many improvements on existing thermal comfort analysis tools, including the following:

- cMap provides both spatial and temporal mapping of comfort parameters in all three dimensions and at every hour of the year.
- cMap calculates angle factors, and automatically pulls surface temperatures and environmental conditions from EnergyPlus, rather than requiring the user to supply these inputs.

- cMap analysis can be done very quickly, making it more viable for use in professional practice.

However, cMap still has several limitations in its current form:

- cMap can only handle a single zone EnergyPlus model.
- cMap cannot account for direct solar radiation in some parts of the space
- cMap cannot handle complex surfaces
- The user must ask for the correct set of output variables in the EnergyPlus model
- cMap can only view one set of results at a time; the GUI doesnt have comparison capabilities.
- cMap cannot predict air temperature and velocity fields in a space
- cMap cannot predict comfort for different parts of the human body

It is the authors intent that several of the limitations above will be resolved in future versions of the software.

It is important to keep in mind what cMap is not. It is not a replacement for CFD analysis. If a design problem requires the precise prediction of air temperature and velocity fields in a space, a CFD package should be used. It is not a comfort model of the human body. If a design problem requires the prediction of comfort for different parts of the human body, UC Berkeleys Advanced Human Thermal Comfort Model or similar software should be used.

Figure 4-1: Screenshot of cMap interface identifying the locations for the required inputs, navigation controls, and analysis outputs

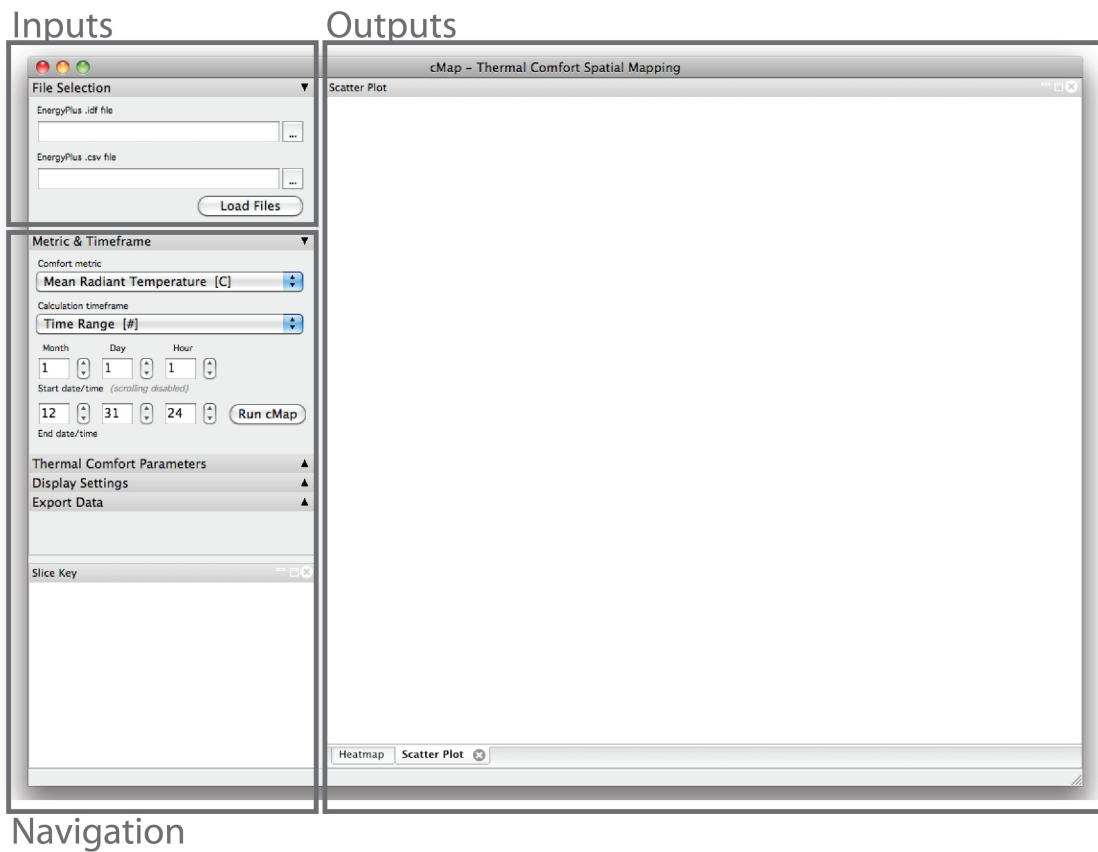


Figure 4-2: Diagram illustrating the analysis workflow within cMap.

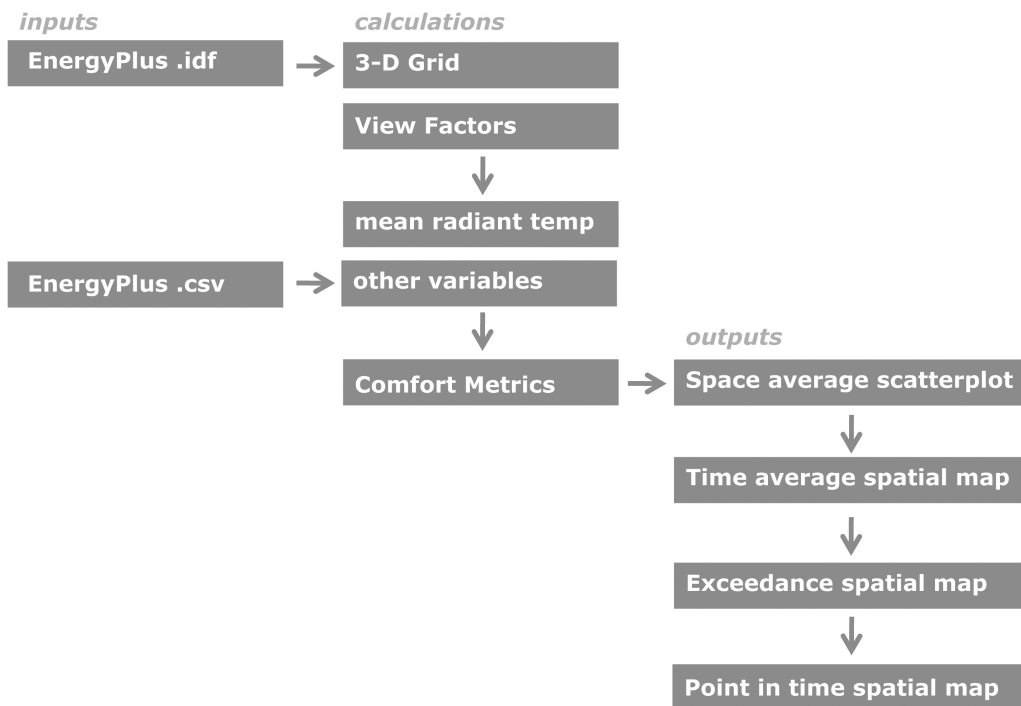


Figure 4-3: Screenshot of cMap showing scatterplot output. Example shown is the operative temperature for a single point in time in January.

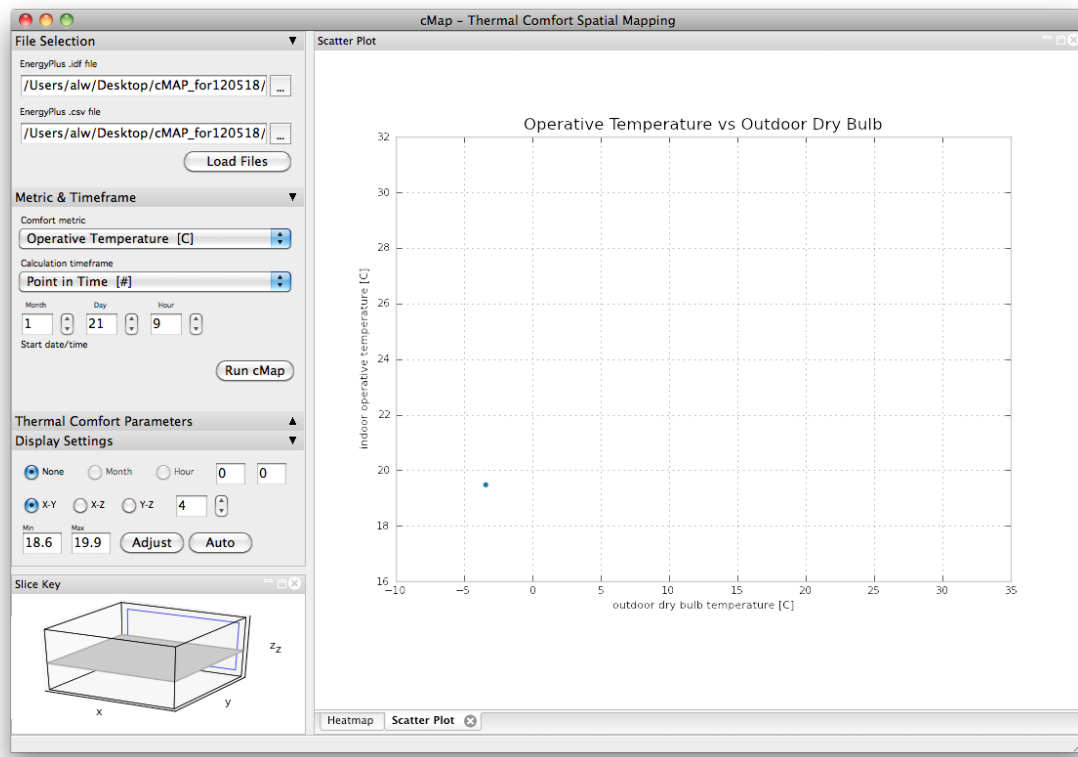


Figure 4-4: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single point in time in January, slice in the X-Y direction

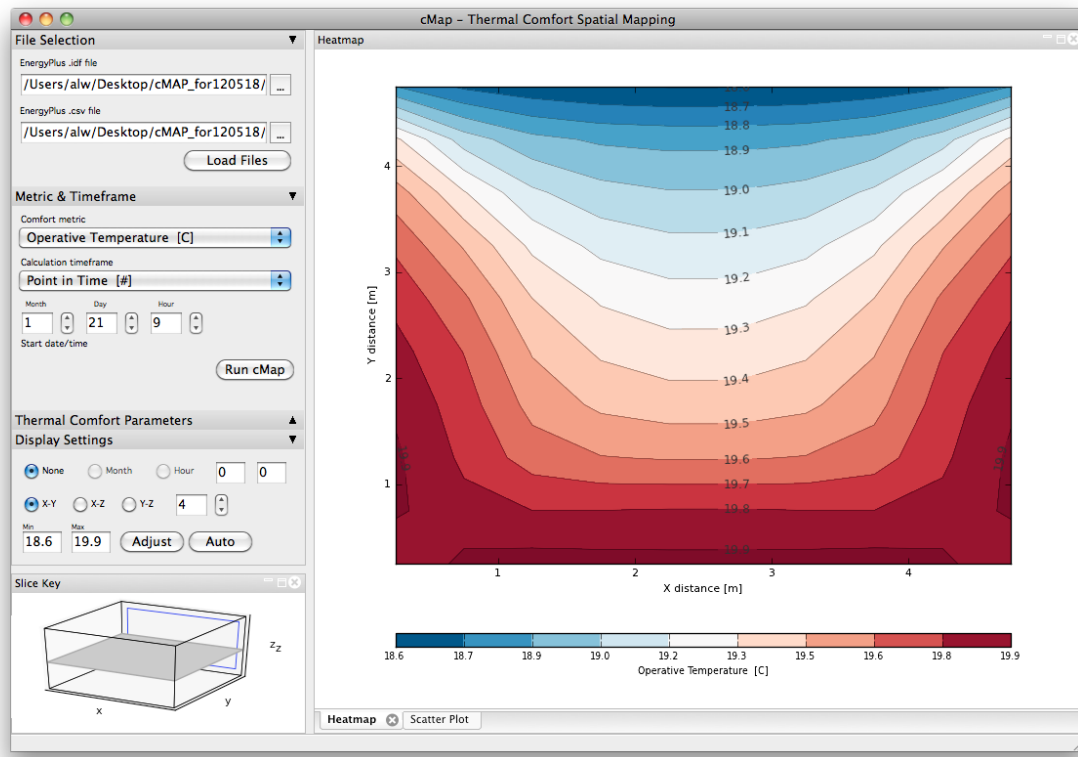


Figure 4-5: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single point in time in January, slice in the Y-Z direction

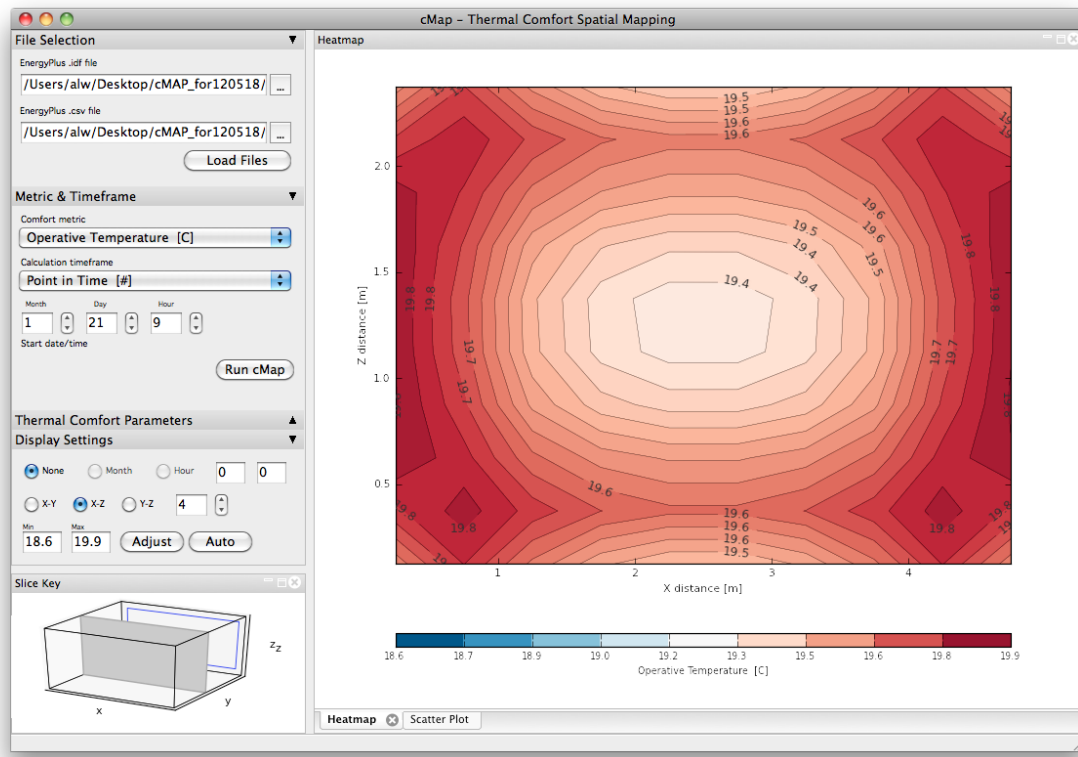


Figure 4-6: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single point in time in January, slice in the X-Z direction

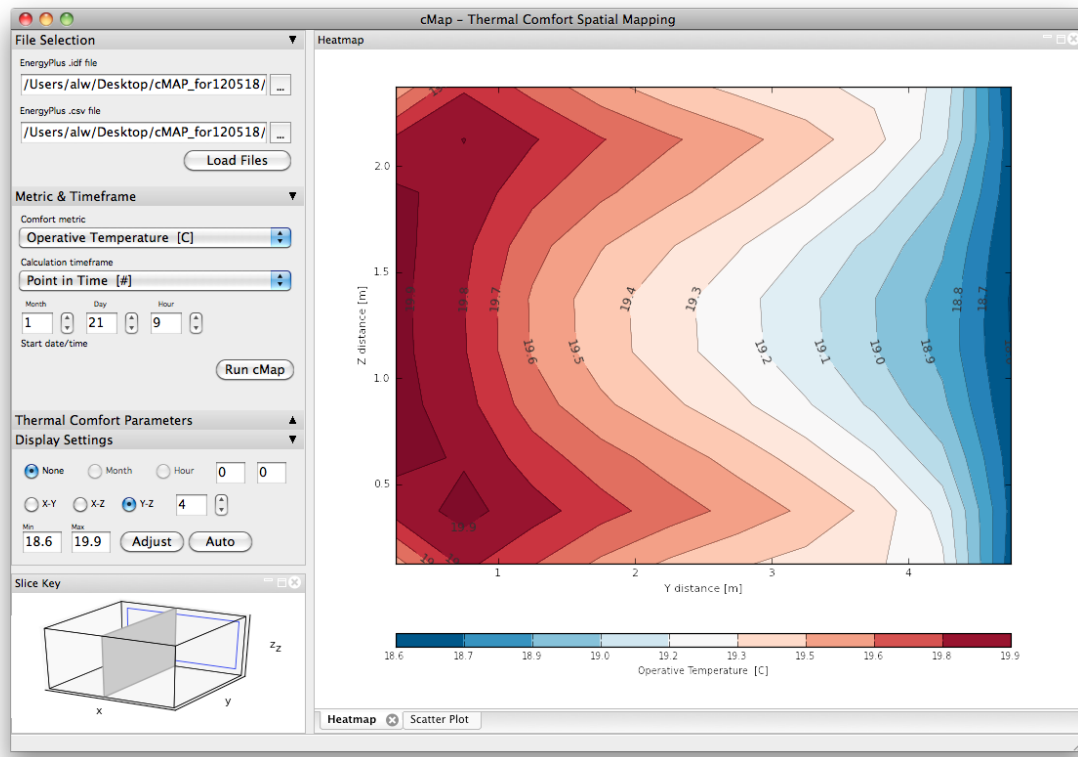


Figure 4-7: Screenshot of cMap showing scatterplot output. Example shown is the ASHRAE 55 adaptive comfort model, for a range of time during the summer. Data points for the month of June are highlighted in red.

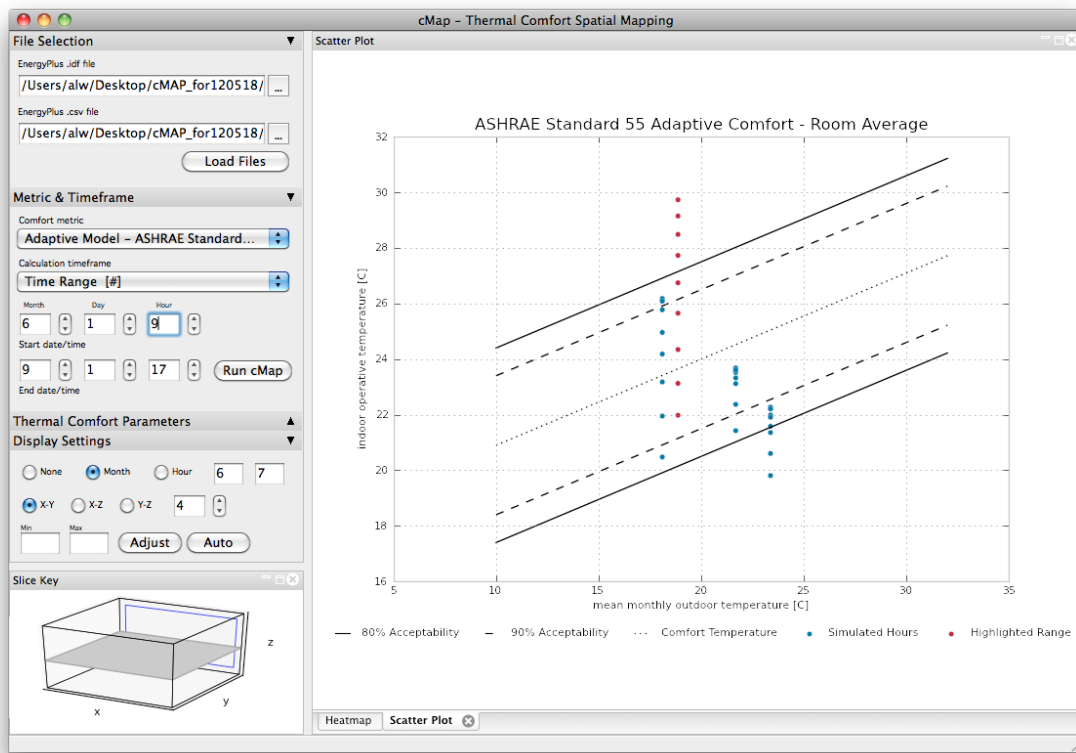


Figure 4-8: Screenshot of cMap showing scatterplot output. Example shown is the ASHRAE 55 adaptive comfort model, for a single peak point in time during the summer.

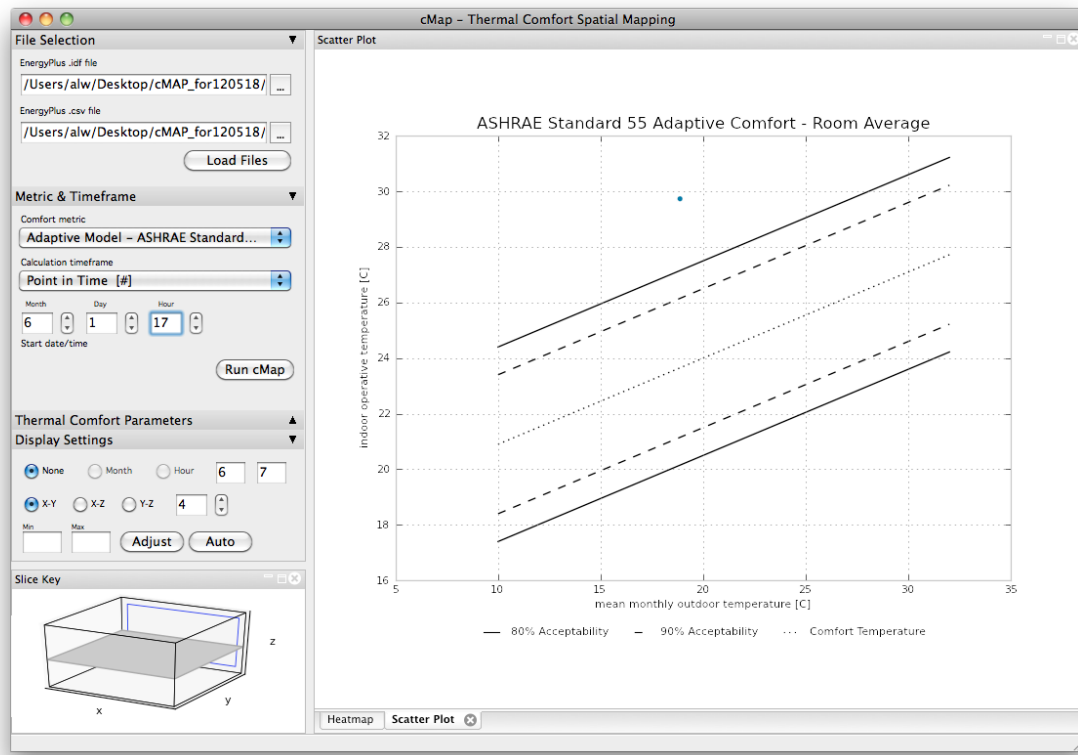


Figure 4-9: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single peak point in time in June, slice in the X-Y direction

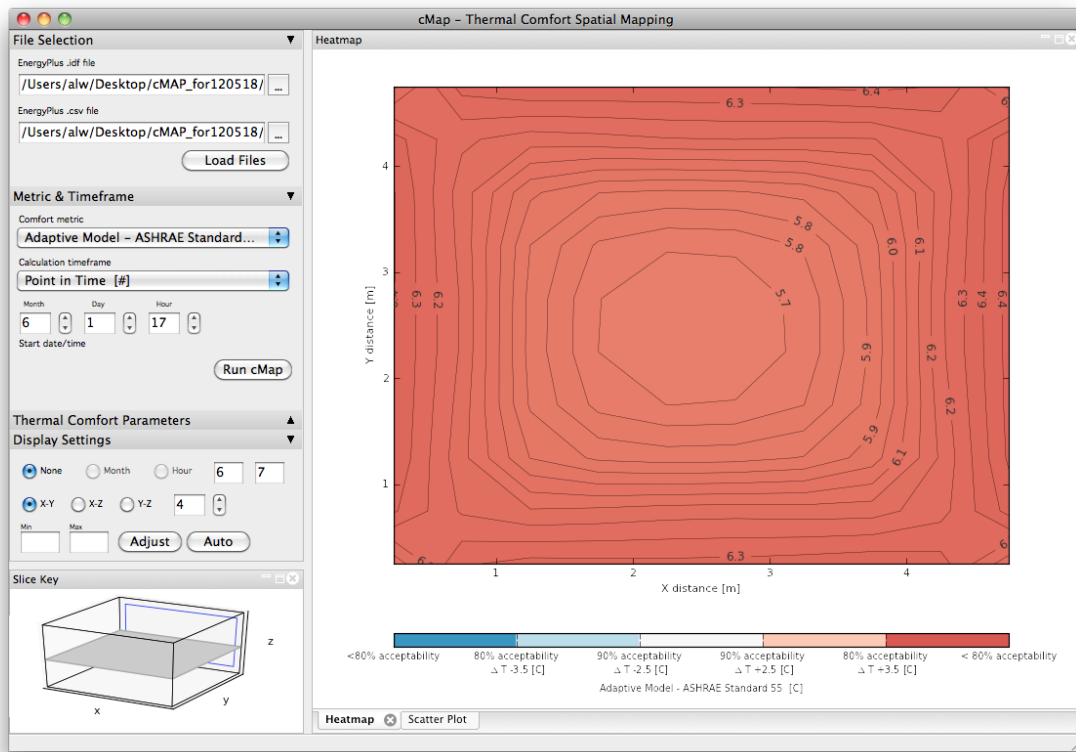


Figure 4-10: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single peak point in time in June, slice in the Y-Z direction

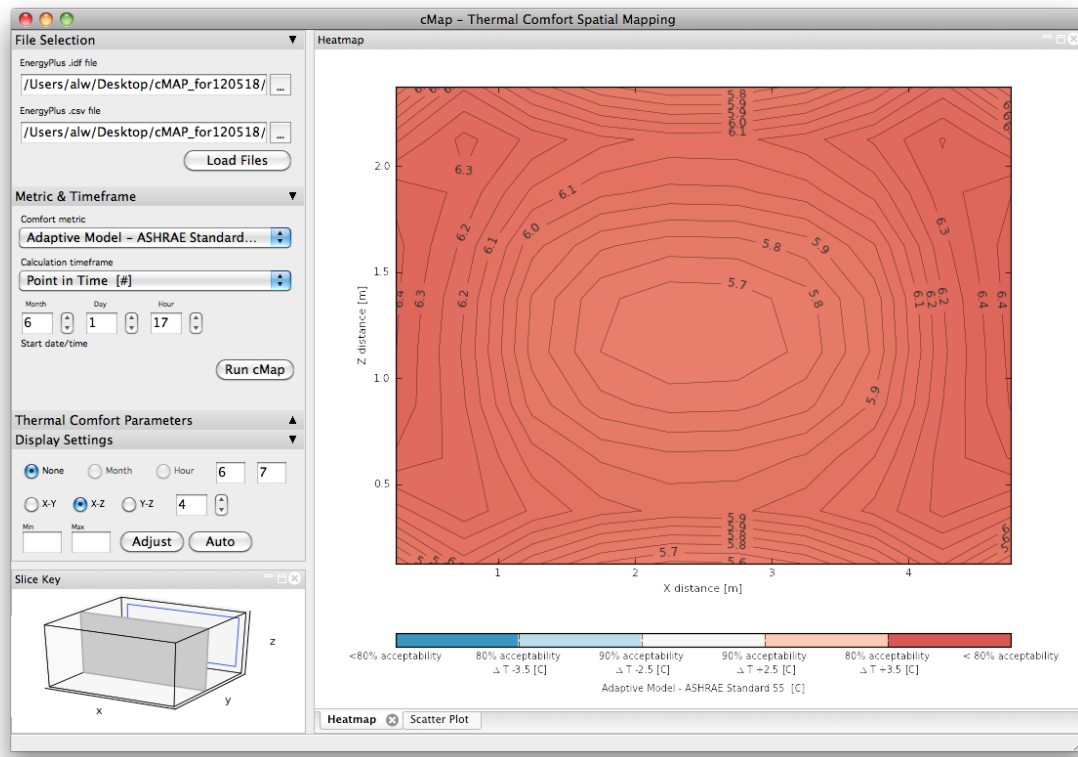
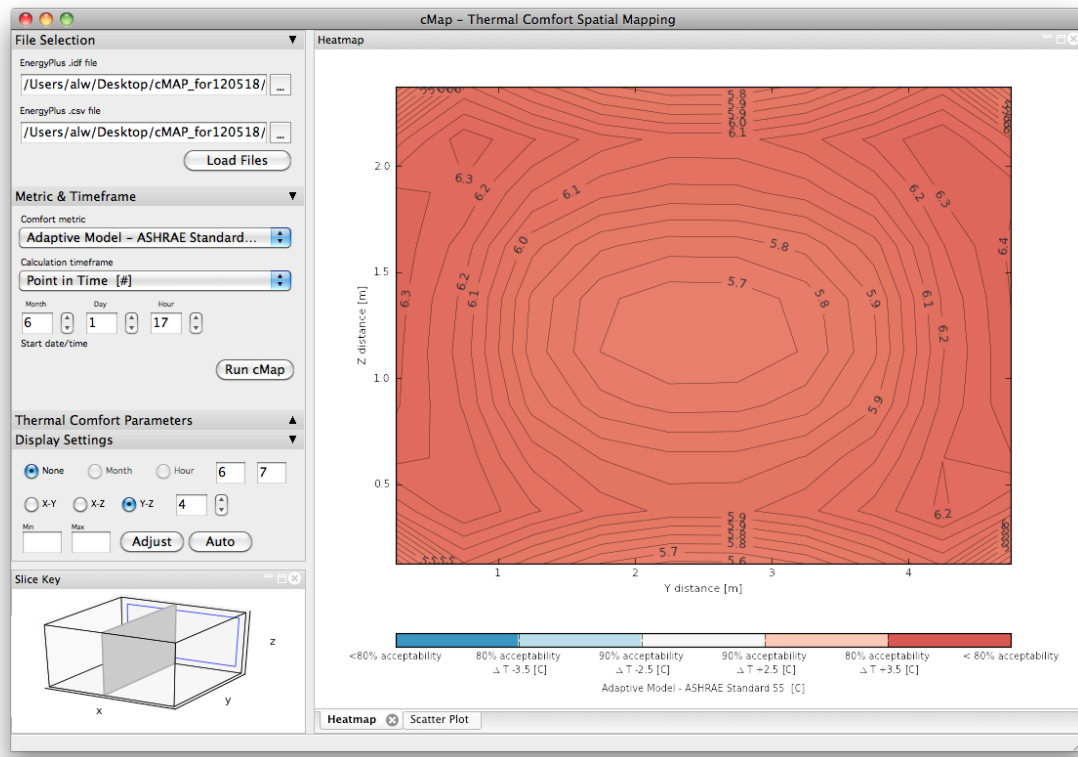


Figure 4-11: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single peak point in time in June, slice in the X-Z direction



Chapter 5

Conclusions

5.1 Thesis Achievements

This overarching goal of this thesis is to bridge the gap between our thermal experience and our building analysis tools. This work provides three primary contributions to the field of thermal comfort research and for the analysis of thermal comfort in practice:

Methodology A unique methodology has been developed that produces mapping of thermal comfort parameters in all three spatial dimensions, as well as over time. Both the Fanger and the adaptive comfort models have been fully incorporated into the methodology.

Software An accompanying software program, called cMap, has been developed to illustrate the ways that this methodology can be used with existing energy analysis software and to demonstrate how it can fit into existing analysis workflows in professional practice. The software is also intended to provide educational benefits, by quantifying and visualizing the thermal environment using a range of thermal comfort metrics.

Dialogue This work is intended to contribute to the larger discussion about thermal comfort and the built environment. As discussed in the background section, the dialogue surrounding the definition of thermal comfort has evolved greatly over the past century. This work is intended to support discussions about the benefits of a diverse thermal environment in our buildings.

5.2 Discussion

This thesis began with the proposition that thermally diverse buildings are compelling. This introductory discussion defined thermal diversity and identified the eventual fate of a compelling building – it becomes beloved. What the initial discussion misses is a clear argument for why thermal diversity makes a building compelling. I want to briefly address that here.

The initial discussion highlighted the role of our thermal receptors as heat flow sensors. These receptors are located in our skin and cover the entire surface area of our bodies. We are constantly awash in a sea of thermal sensations – the thermal environment is intimately connected with our physical being. As discussed in the section on mean radiant temperature, our thermal sensations are function of location. Not only are we awash in a sea of thermal sensations, differentiation in those sensations informs our spatial sense. In sum, the thermal environment plays a critical role in how we locate ourselves in space.

Second, the thermal environment plays a critical role in how we locate ourselves in time. In fact, the thermal environment, to great extent, defines the very notion of time itself. Consider day and night, defined by the presence or absence of the sun – the ultimate source of heat for our planet. Similarly the passing of the seasons, cycles of growth and death are defined by heatflow from the sun. This is also true for buildings. Our experience of time in the built environment is largely defined by these thermal cycles. We know, for example, that it is daytime when window surfaces are hotter and night when they are cooler.

The role of the thermal environment in our perceptions of space and time cannot be underestimated. It goes beyond the issues of symbolism and sensuality that Heschong identifies; the thermal environment is an essential part of our very existence.

Ultimately, this speaks to my larger intellectual project to change the way we, both as individuals and as a society, relate to the natural environment. The thermal environment has the power to make us feel connected to the natural world and can lead to a positive shift in our environmental attitudes. As humans, we spend most of our time in buildings. A connection to the natural world must happen in our buildings and can happen through the use of thermal diversity.

How can a software tool do this? As illustrated in Figure 1-4, analysis tools play a critical role in the design and construction process. Their ability to quantify performance

determines what can and cannot be built. If we want thermal diversity in our built environment, then our analysis tools absolutely need to be able to quantify it.

But the benefits of software tools do not lie solely in the outcome. Software also has the ability to influence the user throughout the analysis process. The building simulation guide from the Chartered Institute of Building Services Engineers (CIBSE), the equivalent of ASHRAE in the United Kingdom, puts it thus:

Consequently, modelling can not only predict the end result, it can also identify the physical processes that have led to that result. By understanding the reasons rather than just the answers, the designer can carry this knowledge forward to the next project. Modelling also speeds the process of learning, which previously could come only from anecdotal feedback from completed projects.

(1).

Software tools can actually influence and improve the way that concepts are understood and communicated. Given the persistent hold that the Fanger model and thermal neutrality and the well-mixed assumption have on our professional standards and analysis methods, the educational effects of software tools are especially important. In order to build a more thermally diverse environment, architects and engineers need to understand what that means, both in general and for their designs. Software tools can play a very important role in that learning process.

The hope is that the cMap methodology will be used as part of the suite of comfort analysis tools that have been identified in this thesis. Project teams need to understand as much as possible about the capabilities and limitations of each available comfort analysis tool in order to select the best tool for to meet the project needs. The appropriate tool will likely depend on the phase of design, the space type being analyzed, and the simulation experience level of the design team. cMap could be used in any situation where an EnergyPlus model has been created (through native EnergyPlus or through a GUI program like DesignBuilder or COMFEN). Ideally it would be used as early as possible in the design process, and would be a particularly helpful aid in glazing and faade design decisions. It could also add additional perspective to the value engineering process. Consider the case of triple pane glazing, which often saves energy but can also be very expensive. If the design team could also quantify the thermal comfort benefits, such an item may have a better

chance of remaining in the project.

5.3 Future Work

This work emerged, in part, out of the author's experience in professional practice. A number of validation studies and additions to both the methodology and the graphical user interface would help further this work as a viable tool for use in professional practice.

In the short term, a number of features could be implemented into the software:

Direct solar radiation capabilities The current mean radiant temperature calculations do not account for the impact of direct solar radiation. A modified mean radiant temperature calculation that includes direct solar radiation should be implemented into the code.

Complex faade capabilities The current mean radiant temperature calculations are not capable of accounting for the effects of complex surfaces, e.g., multiple windows, small windows. The current calculations assume one temperature per surface, for each of the six surfaces in a zone. A mean radiant temperature calculation method accounting for multiple temperatures per surface should be implemented into the code.

Multi-zone model capabilities The current code can only handle single-zone EnergyPlus models. Since most models in professional practice are multi-zone models, the .idf and .csv file parsing code should be changed to handle multi-zone models.

Simple PMV calculation The PMV and PPD calculations for the current code are extremely slow, as a result of the iterative process required to determine tcl. The current method is to use a bisection search to find tcl. While there are many possible solutions, one potential solution may be to use a simplified PMV calculation. A number of authors have derived a non-iterative comfort equation, including Sherman (43) and Federspiel (17). The viability of these methods for speeding up the PMV calculation should be investigated.

EnergyPlus output viewing While cMap currently has the capability to display ambient and indoor environmental parameters, e.g., dry bulb temperature, airspeed, the tool could benefit from providing the user with a more robust way to view the EnergyPlus

.csv output. When faced with a unexpected plot results, the user should have a quick way to look at the EnergyPlus output for troubleshooting purposes.

Exceedance timeseries heatmap While the scatterplot output is helpful, a heatmap plots indicating the hours in exceedance of comfort conditions would help provide the user with an even broader overview of the comfort conditions in the space. The months of the year would be plotted on the x-axis and the hours of the day would be plotted on the y-axis. The temporal maps shown in Mardaljevic (29) provide an example of this type of plot.

In the authors opinion, all of the above features could be implemented with relatively little difficulty.

In the longer term, a set of more involved studies would help provide additional evidence for the value of this approach, and expand the analysis:

Comfort tools survey A survey of architects and engineers in professional practice could help provide a very clear picture of current comfort analysis practices. Much of the information on current comfort analysis practice comes from the authors experience. Because there are no clear guidelines on the analysis of comfort in practice, a survey could provide an initial step towards developing this type of guidance.

Comparison with CFD cMap does not purport to provide results similar to CFD. However, both tools can be used to quantify comfort. Given this, it would be useful to have a detailed picture of the tradeoffs capabilities, cost, time, expertise - between the two tools. This type of information could help practicing architects and engineers make decisions about which tool to use and when.

Integration with body model cMap does not purport to provide information about comfort levels for different parts of the body. The most complete comfort tool would ideally provide comfort conditions across an entire space and across a human body. It may be possible to integrate the cMap methodology with the UC Berkeley Advanced Human Thermal Comfort model or similar body-based tool.

Better airflow considerations The output from EnergyPlus assumes that the air temperature and velocity is the same throughout a space. Since cMap is interested in

spatial mapping of comfort parameters, it may be possible to use heuristic methods or similar work to provide more resolved information about airflow in a space. Something as simple as assuming a stratification profile would be an improvement over the current well-mixed assumption.

Bibliography

- [1] (1998). *Applications Manual AM11: Building Energy and Environmental Modeling*. Chartered Institution of Building Services Engineers (CIBSE).
- [2] (2004). *Standard 55-2004: Thermal Environmental Conditions for Human Occupancy*. The American Society of Heating, Refrigeration and Air-Conditioning Engineers (ASHRAE).
- [3] (2005). *International Standard ISO 7730: Ergonomic of the thermal environment - Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria*.
- [ASHRAE] ASHRAE. Thermal comfort tool. <http://www.ashrae.org/resources-publications/bookstore/thermal-comfort-tool>.
- [5] Bessoudo, M., Tzempelikos, A., Athienitis, A., and Zmeureanu, R. (2010). Indoor thermal environmental conditions near glazed facades with shading devices - part i: Experiments and building thermal model. *Building and Environment*, 45:2506–2516.
- [6] Borgeson, S. and Brager, G. (2011). Comfort standards and variations in exceedance for mixed-mode buildings. *Building Research and Information*, 32(2):118–133.
- [7] Borland, D. and Taylor, R. (2007). Rainbow color map (still) considered harmful. *GG A Visualization Viewpoints*.
- [8] Brager, G. and de Dear, R. (1998). Thermal adaptation in the built environment: A literature review. *Energy and Buildings*, 27(1):83–96.
- [Bruse] Bruse, M. Envi-met. <http://www.envi-met.com/>.
- [10] Cannistraro, G., Franzitta, G., and Giaconia, C. (1992). Algorithms for the calculation of the view factors between human body and rectangular surfaces in parallelepiped environments. *Energy and Buildings*, 19:51–60.
- [11] Cooper, G. (1998). *Air-conditioning America: Engineers and the controlled environment, 1900-1960*. The Johns Hopkins University Press.
- [12] de Dear, R. and Brager, G. (2001). The adaptive model of thermal comfort and energy conservation in the built environment. *International Journal of Biometeorology*, 45:100–108.
- [13] de Dear, R. and Brager, G. (2002). Thermal comfort in naturally ventilated buildings: Revisions to ashrae standard 55. *Energy and Buildings*, 34:549–561.

- [14] de Dear, R., Brager, G., and Cooper, D. (1997). Ashrae rp-884: Developing an adaptive model of thermal comfort and preference. Technical report, The American Society of Heating, Refrigeration and Air-Conditioning Engineers, Inc., and Environmental Analytics.
- [15] Fanger, P. (1970). *Thermal comfort: Analysis and application in environmental engineering*. McGraw-Hill Book Company.
- [16] Fathy, H. (1986). *Natural Energy and Vernacular Architecture: Principles and Examples with Reference to Hot Arid Climates*. The University of Chicago Press.
- [17] Federspiel, C. C. (1992). *User-adaptable and minimum-power thermal comfort control*. PhD thesis, Massachusetts Institute of Technology.
- [18] Fountain, M. and Huizenga, C. (1995). Ashrae rp-781: A thermal sensation model for use by the engineering profession. Technical report, The American Society of Heating, Refrigeration and Air-Conditioning Engineers, Inc., and Environmental Analytics.
- [19] Fountain, M. and Huizenga, C. (1997). A thermal sensation prediction software tool for use by the profession. In *ASHRAE Transactions*, volume 103, pages 130–136. The American Society of Heating, Refrigeration and Air-Conditioning Engineers.
- [20] Frith, F. (1860). *Egypt, Sinai and Jerusalem: a series of twenty photographic views*. William Mackenzie, London.
- [21] Gan, G. (1994). Numerical method for a full assessment of indoor thermal comfort. *Indoor Air*, 4:154–168.
- [22] Gan, G. (2001). Analysis of mean radiant temperature and thermal comfort. *Building Services Engineering Research and Technology*, 22(2):95–101.
- [23] Haves, P., da Graca, G., and Linden, P. (2003). Use of simulation in the design of a large naturally ventilated commercial office building. Proceedings of Building Simulation 2003 - 8th International IBPSA Conference, Eindhoven, Netherlands. International Building Performance Simulation Association.
- [24] Heschong, L. (1979). *Thermal Delight in Architecture*. MIT Press.
- [25] Huizenga, C., Hui, Z., and Arens, E. (2001). A model of human physiology and comfort for assessing complex thermal environments. *Building and Environment*, 36:691–699.
- [26] Huizenga, C., Zhang, H., Mattelaer, P., Yu, T., Arens, E., and Lyons, P. (2006). Window performance for human thermal comfort: Report to the national fenestration rating council. Technical report, Center for the Built Environment - University of California, Berkeley.
- [27] Humphreys, M. and Hancock, M. (2007). Do people like to feel 'neutral'? exploring the variation of the desired thermal sensation on the ashrae scale. *Energy and Buildings*, 39:867–874.
- [28] Humphreys, M. and Nicol, F. (2002). The validity of iso-pmv for predicting comfort votes in every-day thermal environments. *Energy and Buildings*, 34(6):667–684.

- [29] Mardaljevic, J. (2004). Spatio-temporal dynamics of solar shading for a parametrically defined roof system. *Energy and Buildings*, 36:815–823.
- [30] Matzarakis, A., Rutz, F., and Mayer, H. (2010). Modeling radiation fluxes in simple and complex environments: basics of the rayman model. *International Journal of Biometeorology*, 54:131–139.
- [31] Maury, B., Raymond, A., Revault, J., and Zakariya, M. (1983). *Palais et Maisons du Caire: Epoque Ottomane*. Editions du Centre National de la Recherche Scientifique, Paris.
- [32] McCartney, K. and Nicol, F. (2002). Developing an adaptive control algorithm for europe. *Energy and Buildings*, 34(6):623–635.
- [33] Meguro, W. (2005). Beyond blue and red arrows: Optimizing natural ventilation in large buildings. Master’s thesis, Massachusetts Institute of Technology.
- [34] Negrao, C., Carvalho, C., and Melo, C. (1999). Numerical analysis of human thermal comfort inside occupied spaces. Proceedings of Building Simulation 1999 - 6th International IBPSA Conference, Kyoto, Japan. International Building Performance Simulation Association.
- [35] Nicol, F. and Humphreys, M. (2002). Adaptive thermal comfort and sustainable thermal standards for buildins. *Energy and Buildings*, 34(6):563–572.
- [36] Nicol, F. and Humphreys, M. (2010). Derivation of the adaptive equations for thermal comfort in free-running buildings in european standard en15251. *Building and Environment*, 45(1):11–17.
- [of Energy] of Energy, U. D. Energyplus energy simulation software. <http://apps1.eere.energy.gov/buildings/energyplus/>.
- [38] Powitz, R. and Balsamo, J. (1999). Measuring thermal comfort. *Journal of Environmental Health*, 62(5):37–38.
- [39] Rees, S., Lomas, K., and Fiala, D. (2008). Predicting local thermal discomfort adjacent to glazing. In *ASHRAE Transactions*, volume 114, page 1. The American Society of Heating, Refrigeration and Air-Conditioning Engineers.
- [40] Rogowitz, B. and Treinish, L. (1995). How not to lie with visualization. *IBM Watson*.
- [41] Schneiderman, B. (1996). The eyes have it: a task by data type taxonomy for information visualizations. Proceedings of Visual Languages ’96.
- [42] Sebastian Herkel, Frank Schoffel, J. D. (1999). Interactive three-dimensional visualisation of thermal comfort. Proceedings of Building Simulation 1999 - 6th International IBPSA Conference, Kyoto, Japan. International Building Performance Simulation Association.
- [43] Sherman, M. (1985). A simplified model of thermal comfort. *Energy and Buildings*, 8(37-50).

- [44] Shove, E., Chappells, H., Lutzenhiser, L., and Hackett, B. (2008). Comfort in a lower carbon society. *Building Research and Information*, 36(4):307–311.
- [45] Srebric, J. (2011). *Building Performance Simulation for Design and Optimization*, chapter Chapter 6: Ventilation performance prediction. Routledge.
- [46] Tufte, E. (1997). *Visual and Statistical Thinking: Displays of Evidence for Making Decisions*. Graphics Press.
- [47] van der Linden, A., Boerstra, A., Raue, A., Kurvers, S., and de Dear, R. (2006). Adaptive temperature limits: A new guideline in the netherlands - a new approach for the assessment of building performance with respect to thermal indoor climate. *Energy and Buildings*, 38:8–17.
- [48] van Treeck, C., Frisch, J., Egger, M., and Rank, E. (2009). Model-adaptive analysis of indoor thermal comfort. Proceedings of Building Simulation 2009 - 11th International IBPSA Conference, Glasgow, Scotland. International Building Performance Simulation Association.
- [49] Webb, A. L. (2009). How to win poetic praise and influence architects. *Magazine on Urbanism (MONU)*, (11).
- [50] White, A. and Holmes, M. (2009). Advanced simulation applications using room. Proceedings of Building Simulation 2009 - 11th International IBPSA Conference, Glasgow, Scotland. International Building Performance Simulation Association.
- [51] Yang, T., Cropper, P., Cook, M., Yousaf, R., and Fiala, D. (2007). A new simulation system to predict human-environment thermal interactions in naturally ventilated buildings. Proceedings of Building Simulation 2007 - 10th International IBPSA Conference, Beijing, China. International Building Performance Simulation Association.
- [52] Zeileis, A., Hornik, K., and Murrell, P. (2009). Escaping rgbland: Selecting colors for statistical graphics. *Computational Statistics and Data Analysis*.

Appendix A

cMap Source Code

The full source code for cMap is reproduced below. The program is written in Python. The program relies on Numpy and Scipy for performing the primary calculations. The program uses Matplotlib to create the plots. The program uses wxPython to create the graphical user interface.

```
1
2 # !/usr/bin/env python
3 # Filename: cMap.py
4
5 """ Script creates 3-D set of grid points from an EnergyPlus .idf file
6 Script uses EnergyPlus .csv file to calculate comfort outputs at each point
7 Script outputs results as .png image file
8 Script includes interface for controlling raw inputs & visualizing output
9 """
10
11 # BEGIN INTERFACE -----
12 # Mapping Script Imports -----
13 from numpy import *
14 import csv
15 import re
16 import math
17 from matplotlib import pyplot, mpl, cm
```

```

18 from matplotlib.colors import *
19 from scipy.optimize import brentq
20 from matplotlib.lines import Line2D
21
22 # Interface Imports -----
23 import sys
24 import os
25 import wx
26 import wx.lib.agw.aui as aui
27 import wx.combo
28 import wx.lib.buttons as buttons
29 import wx.lib.agw.floatspin as FS
30 import wx.lib.scrolledpanel as scrolled
31 from mpl_toolkits.mplot3d import axes3d
32 from mpl_toolkits.mplot3d.art3d import Poly3DCollection
33 import wx.lib.agw.foldpanelbar as fpb
34 from matplotlib.figure import Figure
35 from matplotlib.backends.backend_wxagg import \
36     FigureCanvasWxAgg as FigureCanvas
37 import matplotlib.font_manager as fm
38
39 # Interface Components -----
40 class MyFrame(wx.Frame):
41
42     def __init__(self, parent, ID, title):
43         wx.Frame.__init__(self, parent, ID, title, size=(1100, 760))
44
45         # AUI Panes, Panels, Menu -----
46         self._mgr = aui.AuiManager(self)
47
48         self.panel1 = wx.Panel(self, -1)
49         self.panel2 = wx.Panel(self, -1)

```

```

50     self.panel2.SetBackgroundColour("White")
51     self.panel3 = wx.Panel(self, -1)
52     self.panel3.SetBackgroundColour("White")
53     self.panel4 = wx.Panel(self, -1)
54     self.panel4.SetBackgroundColour("White")
55
56     self.CreateStatusBar()
57
58     # add the panes to the manager
59     self._mgr.AddPane(self.panel1, aui.AuiPaneInfo().
60         Caption("Input Parameters").CaptionVisible(False).Left().
61         Layer(2).Floatable(True).Dockable(True).
62         MinimizeButton(True).MaximizeButton(True).CloseButton(True).
63         BestSize(wx.Size(300, 760)))
64     self._mgr.AddPane(self.panel2, aui.AuiPaneInfo().Name("panel2").
65         Caption("Heatmap").Center().Floatable(True).Dockable(True).
66         MinimizeButton(True).MaximizeButton(True).CloseButton(True).
67         BestSize(wx.Size(400, 360)))
68     self._mgr.AddPane(self.panel3, aui.AuiPaneInfo().Name("panel3").
69         Caption("Scatter Plot").Center().Floatable(True).
70         Dockable(True).MinimizeButton(True).MaximizeButton(True).
71         CloseButton(True).BestSize(wx.Size(400, 360)),
72         target=self._mgr.GetPane("panel2"))
73     self._mgr.AddPane(self.panel4, aui.AuiPaneInfo().Name("panel4").
74         Caption("Slice Key").Left().Floatable(True).Dockable(True).
75         MinimizeButton(True).MaximizeButton(True).CloseButton(True).
76         BestSize(wx.Size(200, 200)))
77
78     # Float panel4
79     self._mgr.GetPane("panel4").Float().FloatingPosition((940,480)). \
80         FloatingSize((200,150))
81

```

```

82     # Tell the manager to commit all the changes just made
83     self._mgr.Update()
84     self.Bind(wx.EVT_CLOSE, self.OnClose)
85
86     # Default number of gridpoints -----
87     self.gpt = 10
88
89     # Sizers & Sizer Items -----
90     # set-up all sizers on panell1
91     idfSizer = wx.BoxSizer(wx.VERTICAL)
92     csvSizer = wx.BoxSizer(wx.VERTICAL)
93     calcTextSizer1 = wx.GridBagSizer(hgap=6, vgap=0)
94     calcTextSizer = wx.GridSizer(rows=1, cols=3, hgap=40, vgap=0)
95     dateSizer1 = wx.GridBagSizer(hgap=6, vgap=0)
96     dateSizer2 = wx.GridBagSizer(hgap=6, vgap=0)
97     dateSizer3 = wx.GridBagSizer(hgap=6, vgap=0)
98
99     comfSizer = wx.GridBagSizer(hgap=6, vgap=0)
100    pointTextSizer = wx.GridSizer(rows=1, cols=2, hgap=110, vgap=0)
101    cutSizer = wx.GridBagSizer(hgap=10, vgap=0)
102    colorTextSizer = wx.GridSizer(rows=1, cols=1)
103    colorSizer = wx.GridBagSizer(hgap=7, vgap=0)
104    pointSizer = wx.GridBagSizer(hgap=0, vgap=0)
105    scaleTextSizer = wx.GridSizer(rows=1, cols=3, hgap=25, vgap=0)
106    scaleSizer = wx.GridBagSizer(hgap=5, vgap=0)
107    printSizer = wx.BoxSizer(wx.VERTICAL)
108
109    # all font & caption bar styles -----
110    titleFont = wx.Font(13, wx.SWISS, wx.NORMAL, wx.NORMAL)
111    subFont = wx.Font(9, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
112    subFontSM = wx.Font(8, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
113    subBoldFont = wx.Font(9, wx.DEFAULT, wx.NORMAL, wx.NORMAL)

```

```

114     dateFont = wx.Font(7, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
115     scrollingFont = wx.Font(9, wx.DEFAULT, wx.ITALIC, wx.NORMAL)
116
117     cs = fpb.CaptionBarStyle()
118     cs.SetCaptionStyle(fpb.CAPTIONBAR_GRADIENT_V)
119     color1 = wx.Colour(220, 220, 220)
120     color2 = wx.Colour(195, 195, 195)
121     cs.SetFirstColour(color1)
122     cs.SetSecondColour(color2)
123     cs.SetCaptionFont(titleFont)
124
125     # Sizer Items - controls and static texts -----
126     # create fold panel instance on main panel
127     self.fold_panel = fpb.FoldPanelBar(self.panel1, wx.ID_ANY,
128         style=fpb.FPB_VERTICAL)
129
130     # Fold panel bar - file selector -----
131     sectFile = self.fold_panel.AddFoldPanel("File Selection",
132         collapsed=False, cbstyle=cs)
133     subpanelFile = wx.Panel(sectFile, -1)
134     sizerFile = wx.BoxSizer(wx.VERTICAL)
135
136     # texts
137     idfTxt = wx.StaticText(subpanelFile, -1, "EnergyPlus .idf file",
138         style=wx.ALIGN_LEFT)
139     idfTxt.SetFont(subFont)
140     csvTxt = wx.StaticText(subpanelFile, -1, "EnergyPlus .csv file",
141         style=wx.ALIGN_LEFT)
142     csvTxt.SetFont(subFont)
143
144     # controls
145     self.idfSel = FileSelectorComboIDF(subpanelFile, size=(280, -1))

```

```

146     idfSizer.Add(self.idfSel, 0, wx.LEFT, border=5)
147     self.csvSel = FileSelectorComboCSV(subpanelFile, size=(280, -1))
148     csvSizer.Add(self.csvSel, 0, wx.LEFT, border=5)
149
150     # load button
151     self.load = wx.Button(subpanelFile, -1, 'Load Files',
152         size=(110, -1))
153     self.load.Bind(wx.EVT_BUTTON, self.loadFiles)
154
155     # put controls on folding sizer
156     sizerFile.Add(idfTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 7)
157     sizerFile.Add(idfSizer, 0, wx.TOP|wx.BOTTOM, 2)
158     sizerFile.AddSpacer(5)
159     sizerFile.Add(csvTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 7)
160     sizerFile.Add(csvSizer, 0, wx.TOP|wx.BOTTOM, 2)
161     sizerFile.Add(self.load, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT, 2)
162
163     sizerFile.AddSpacer(15)
164
165     # set folding sizer on folding panel
166     subpanelFile.SetSizer(sizerFile)
167     subpanelFile.Fit()
168
169     # add folding panel to foldpanelbar
170     self.fold_panel.AddFoldPanelWindow(sectFile, subpanelFile,
171         fpb.FPB_ALIGN_LEFT)
172     self.fold_panel.AddFoldPanelSeparator(sectFile)
173
174     # Fold panel bar - metric selector -----
175     sectMetric = self.fold_panel.AddFoldPanel("Metric & Timeframe",
176         collapsed=False, cbstyle=cs)
177     subpanelMetric = wx.Panel(sectMetric, -1)

```



```

178     sizerMetric = wx.BoxSizer(wx.VERTICAL)
179
180     # texts
181     calcTxt = wx.StaticText(subpanelMetric, -1, "Comfort metric",
182         style=wx.ALIGN_LEFT)
183     calcTxt.SetFont(subFont)
184     timeframeTxt = wx.StaticText(subpanelMetric, -1,
185         "Calculation timeframe", style=wx.ALIGN_LEFT)
186     timeframeTxt.SetFont(subFont)
187     simpMonthTxt = wx.StaticText(subpanelMetric, -1, "Month",
188         style=wx.CENTER)
189     simpMonthTxt.SetFont(dateFont)
190     simpDayTxt = wx.StaticText(subpanelMetric, -1, "Day",
191         style=wx.ALIGN_CENTER)
192     simpDayTxt.SetFont(dateFont)
193     simpHourTxt = wx.StaticText(subpanelMetric, -1, "Hour",
194         style=wx.ALIGN_CENTER)
195     simpHourTxt.SetFont(dateFont)
196     stTxt = wx.StaticText(subpanelMetric, -1, "Start date/time",
197         style=wx.ALIGN_LEFT)
198     stTxt.SetFont(subFont)
199     self.enTxt = wx.StaticText(subpanelMetric, -1, "End date/time",
200         style=wx.ALIGN_LEFT)
201     self.enTxt.SetFont(subFont)
202     calcTextSizer1.Add(stTxt, pos=(0,0))
203
204     # add text to text gridsizers
205     calcTextSizer.Add(simpMonthTxt, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL, 0)
206     calcTextSizer.Add(simpDayTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 4)
207     calcTextSizer.Add(simpHourTxt, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL, 0)
208
209     # controls

```

```

210     # calc type selector
211     self.calcDrop = wx.Choice(parent=subpanelMetric, size=(280, -1))
212     self.calcList = [
213         'Mean Radiant Temperature [C]',
214         'Operative Temperature [C]',
215         'Predicted Mean Vote [+/-]',
216         'Percentage People Dissatisfied [%]',
217         'Adaptive Model - ASHRAE Standard 55 [C]',
218         'Adaptive Model - EN Standard 15251 [C]',
219         'Adaptive Model - NPR-CR 1752 Type Beta [C]',
220     ]
221     self.calcDrop.AppendItems(strings=self.calcList)
222     self.calcDrop.Select(n=0)
223
224     self.timeframeDrop = wx.Choice(parent=subpanelMetric, size=(280, -1))
225     self.timeframeList = [
226         'Time Range [#] ',
227         'Point in Time [#] ',
228     ]
229     self.timeframeDrop.AppendItems(strings=self.timeframeList)
230     self.timeframeDrop.Select(n=0)
231     self.timeframeDrop.Bind(wx.EVT_CHOICE, self.DisableAnnual)
232
233     # run button
234     self.run = wx.Button(subpanelMetric, -1, 'Run cMap', size=(80, -1))
235     self.run.Bind(wx.EVT_BUTTON, self.runCmap)
236     dateSizer2.Add(self.run, pos=(0,3), flag=wx.TOP|wx.LEFT|wx.ALIGN_RIGHT,
237         border=5)
238
239     # calc start date/time spins
240     self.StartMonthSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
241     self.StartMonthSpin.SetRange(1,12)

```

```

242     self.StartMonthSpin.SetValue(1)
243     self.StartMonthSpin.Bind(wx.EVT_SPIN, self.runUpdate)
244     self.StartMonthSpin.Bind(wx.EVT_TEXT, self.runUpdate)
245     dateSizer1.Add(self.StartMonthSpin, pos=(1,0))
246
247     self.StartDaySpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
248     self.StartDaySpin.SetRange(1,31)
249     self.StartDaySpin.SetValue(1)
250     self.StartDaySpin.Bind(wx.EVT_SPIN, self.runUpdate)
251     self.StartDaySpin.Bind(wx.EVT_TEXT, self.runUpdate)
252     dateSizer1.Add(self.StartDaySpin, pos=(1,1))
253
254     self.StartHourSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
255     self.StartHourSpin.SetRange(1,24)
256     self.StartHourSpin.SetValue(1)
257     self.StartHourSpin.Bind(wx.EVT_SPIN, self.runUpdate)
258     self.StartHourSpin.Bind(wx.EVT_TEXT, self.runUpdate)
259     dateSizer1.Add(self.StartHourSpin, pos=(1,2))
260
261     # calc end date/time spins
262     self.EndMonthSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
263     self.EndMonthSpin.SetRange(1,12)
264     self.EndMonthSpin.SetValue(12)
265     dateSizer2.Add(self.EndMonthSpin, pos=(0,0))
266
267     self.EndDaySpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
268     self.EndDaySpin.SetRange(1,31)
269     self.EndDaySpin.SetValue(31)
270     dateSizer2.Add(self.EndDaySpin, pos=(0,1))
271
272     self.EndHourSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
273     self.EndHourSpin.SetRange(1,24)

```

```

274     self.EndHourSpin.SetValue(24)
275     dateSizer2.Add(self.EndHourSpin, pos=(0,2))
276
277     dateSizer1.Add(simpMonthTxt, pos=(0,0), flag=wx.LEFT, border=8)
278     dateSizer1.Add(simpDayTxt, pos=(0,1), flag=wx.LEFT, border=12)
279     dateSizer1.Add(simpHourTxt, pos=(0,2), flag=wx.LEFT, border=12)
280
281     # put controls on folding sizer
282     sizerMetric.Add(calcTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 8)
283     sizerMetric.AddSpacer(2)
284     sizerMetric.Add(self.calcDrop, 0, wx.LEFT, border=5)
285     sizerMetric.AddSpacer(10)
286     sizerMetric.Add(timeframeTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 8)
287     sizerMetric.AddSpacer(2)
288     sizerMetric.Add(self.timeframeDrop, 0, wx.LEFT, border=5)
289     sizerMetric.AddSpacer(10)
290     sizerMetric.Add(dateSizer1, 0, wx.LEFT, 5)
291     sizerMetric.Add(calcTextSizer1, flag=wx.LEFT, border=8)
292     sizerMetric.AddSpacer(5)
293     sizerMetric.Add(dateSizer2, 0, wx.LEFT, 5)
294     sizerMetric.Add(self.enTxt, flag=wx.LEFT, border=8)
295     sizerMetric.AddSpacer(15)
296
297     # set folding sizer on folding panel
298     subpanelMetric.SetSizer(sizerMetric)
299     subpanelMetric.Fit()
300
301     # add folding panel to foldpanelbar
302     self.fold_panel.AddFoldPanelWindow(sectMetric, subpanelMetric,
303         fpb.FPB_ALIGN_LEFT)
304     self.fold_panel.AddFoldPanelSeparator(sectMetric)
305

```

```

306 # Fold panel bar - thermal comfort parameters -----
307 sectComfParams = self.fold_panel.AddFoldPanel \
308     ("Thermal Comfort Parameters", collapsed=False, cbstyle=cs)
309 subpanelComfParams = wx.Panel(sectComfParams, -1)
310 sizerComfParams = wx.BoxSizer(wx.VERTICAL)
311
312 # texts & controls
313 outdoorTxt = wx.StaticText(subpanelComfParams, -1, "OUTDOOR",
314     style=wx.ALIGN_RIGHT)
315 outdoorTxt.SetFont(subFont)
316 comfSizer.Add(outdoorTxt, pos=(0,0),
317     flag=wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
318
319 indoorText = wx.StaticText(subpanelComfParams, -1, "INDOOR",
320     style=wx.ALIGN_RIGHT)
321 indoorText.SetFont(subFont)
322 comfSizer.Add(indoorText, pos=(0,3),
323     flag=wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
324
325 peopleText = wx.StaticText(subpanelComfParams, -1, "PEOPLE",
326     style=wx.ALIGN_RIGHT)
327 peopleText.SetFont(subFont)
328 comfSizer.Add(peopleText, pos=(0,6),
329     flag=wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)
330
331 # display items for outdoor dry bulb
332 oDBTxt = wx.StaticText(subpanelComfParams, -1, "dry bulb:",
333     style=wx.ALIGN_RIGHT)
334 oDBTxt.SetFont(subFont)
335 comfSizer.Add(oDBTxt, pos=(1,0),
336     flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, border=1)
337

```

```

338     self.oDBSpin = wx.StaticText(subpanelComfParams, -1, "-",
339         style=wx.ALIGN_LEFT)
340     self.oDBSpin.SetFont(subBoldFont)
341     comfSizer.Add(self.oDBSpin, pos=(1,1),
342         flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
343
344     oDBUnit = wx.StaticText(subpanelComfParams, -1, "C",
345         style=wx.ALIGN_RIGHT)
346     oDBUnit.SetFont(subFontSM)
347     comfSizer.Add(oDBUnit, pos=(1,2),
348         flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
349
350     # display items for outdoor relative humidity
351     oRHTxt = wx.StaticText(subpanelComfParams, -1, "humidity:",
352         style=wx.ALIGN_RIGHT)
353     oRHTxt.SetFont(subFont)
354     comfSizer.Add(oRHTxt, pos=(2,0),
355         flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)
356
357     self.oRHSpin = wx.StaticText(subpanelComfParams, -1, "-",
358         style=wx.ALIGN_LEFT)
359     self.oRHSpin.SetFont(subBoldFont)
360     comfSizer.Add(self.oRHSpin, pos=(2,1),
361         flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
362
363     oRHUnit = wx.StaticText(subpanelComfParams, -1, "%",
364         style=wx.ALIGN_RIGHT)
365     oRHUnit.SetFont(subFontSM)
366     comfSizer.Add(oRHUnit, pos=(2,2),
367         flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
368
369     # display items for outdoor airspeed

```

```

370     oAirVeloTxt = wx.StaticText(subpanelComfParams, -1, "airspeed",
371         style=wx.ALIGN_RIGHT)
372     oAirVeloTxt.SetFont(subFont)
373     comfSizer.Add(oAirVeloTxt, pos=(3,0),
374         flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)
375
376     self.oAirVSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
377         style=wx.ALIGN_LEFT)
378     self.oAirVSpin.SetFont(subBoldFont)
379     comfSizer.Add(self.oAirVSpin, pos=(3,1),
380         flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
381
382     oAirVUnit = wx.StaticText(subpanelComfParams, -1, "m/s",
383         style=wx.ALIGN_RIGHT)
384     oAirVUnit.SetFont(subFontSM)
385     comfSizer.Add(oAirVUnit, pos=(3,2),
386         flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=1)
387
388     # display items for indoor dry bulb
389     zDBTxt = wx.StaticText(subpanelComfParams, -1, "dry bulb:",
390         style=wx.ALIGN_RIGHT)
391     zDBTxt.SetFont(subFont)
392     comfSizer.Add(zDBTxt, pos=(1,3),
393         flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)
394
395     self.zDBSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
396         style=wx.ALIGN_LEFT)
397     self.zDBSpin.SetFont(subBoldFont)
398     comfSizer.Add(self.zDBSpin, pos=(1,4),
399         flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
400
401     zDBUnit = wx.StaticText(subpanelComfParams, -1, "C",

```

```

402         style=wx.ALIGN_RIGHT)
403     zDBUnit.SetFont(subFontSM)
404     comfSizer.Add(zDBUnit, pos=(1,5),
405         flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
406
407     # display items for indoor relative humidity
408     zRHtxt = wx.StaticText(subpanelComfParams, -1, "humidity:",
409         style=wx.ALIGN_RIGHT)
410     zRHtxt.SetFont(subFont)
411     comfSizer.Add(zRHtxt, pos=(2,3),
412         flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)
413
414     self.zRHSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
415         style=wx.ALIGN_LEFT)
416     self.zRHSpin.SetFont(subBoldFont)
417     comfSizer.Add(self.zRHSpin, pos=(2,4),
418         flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
419
420     zRHUnit = wx.StaticText(subpanelComfParams, -1, "%",
421         style=wx.ALIGN_RIGHT)
422     zRHUnit.SetFont(subFontSM)
423     comfSizer.Add(zRHUnit, pos=(2,5),
424         flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
425
426     # display items for indoor airspeed
427     zAirVtxt = wx.StaticText(subpanelComfParams, -1, "airspeed:",
428         style=wx.ALIGN_RIGHT)
429     zAirVtxt.SetFont(subFont)
430     comfSizer.Add(zAirVtxt, pos=(3,3),
431         flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)
432
433     self.zAirVSchSpin = wx.StaticText(subpanelComfParams, -1, "-    ",

```



```

434         style=wx.ALIGN_LEFT)
435     self.zAirVSpin.SetFont(subBoldFont)
436     comfSizer.Add(self.zAirVSpin, pos=(3,4),
437                 flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
438
439     zAirVUnit = wx.StaticText(subpanelComfParams, -1, "m/s",
440                             style=wx.ALIGN_RIGHT)
441     zAirVUnit.SetFont(subFontSM)
442     comfSizer.Add(zAirVUnit, pos=(3,5),
443                 flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
444
445     # display items for clothing level
446     zCloSchTxt = wx.StaticText(subpanelComfParams, -1, "clothing:",
447                               style=wx.ALIGN_RIGHT)
448     zCloSchTxt.SetFont(subFont)
449     comfSizer.Add(zCloSchTxt, pos=(1,6),
450                 flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=1)
451
452     self.zCloSchSpin = wx.StaticText(subpanelComfParams, -1, "-   ",
453                                     style=wx.ALIGN_LEFT)
454     self.zCloSchSpin.SetFont(subBoldFont)
455     comfSizer.Add(self.zCloSchSpin, pos=(1,7),
456                 flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
457
458     CloUnit = wx.StaticText(subpanelComfParams, -1, "clo",
459                             style=wx.ALIGN_RIGHT)
460     CloUnit.SetFont(subFontSM)
461     comfSizer.Add(CloUnit, pos=(1,8),
462                 flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
463
464     # display items for activity level
465     zActSchTxt = wx.StaticText(subpanelComfParams, -1, "activity:",

```

```

466         style=wx.ALIGN_RIGHT)
467     zActSchTxt.SetFont(subFont)
468     comfSizer.Add(zActSchTxt, pos=(2,6),
469                 flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)
470
471     self.zActSchSpin = wx.StaticText(subpanelComfParams, -1, "-   ",
472                                     style=wx.ALIGN_LEFT)
473     self.zActSchSpin.SetFont(subBoldFont)
474     comfSizer.Add(self.zActSchSpin, pos=(2,7),
475                 flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)
476
477     MetUnit = wx.StaticText(subpanelComfParams, -1, "met",
478                             style=wx.ALIGN_RIGHT)
479     MetUnit.SetFont(subFontSM)
480     comfSizer.Add(MetUnit, pos=(2,8),
481                 flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)
482
483     # put controls on folding sizer
484     sizerComfParams.AddSpacer(5)
485     sizerComfParams.Add(comfSizer, 0,
486                       wx.LEFT| wx.ALIGN_CENTER_VERTICAL, 5)
487     sizerComfParams.AddSpacer(15)
488
489     # set folding sizer on folding panel
490     subpanelComfParams.SetSizer(sizerComfParams)
491     subpanelComfParams.Fit()
492
493     # add folding panel to foldpanelbar
494     self.fold_panel.AddFoldPanelWindow(sectComfParams,
495                                       subpanelComfParams, fpb.FPB_ALIGN_LEFT)
496     self.fold_panel.AddFoldPanelSeparator(sectComfParams)
497

```

```

498 # Fold panel bar - display settings -----
499 sectDisplay = self.fold_panel.AddFoldPanel("Display Settings",
500     collapsed=False, cbstyle=cs)
501 subpanelDisplay = wx.Panel(sectDisplay, -1)
502 sizerDisplay = wx.BoxSizer(wx.VERTICAL)
503
504 # texts
505 scaleMinTxt = wx.StaticText(subpanelDisplay, -1, "Min",
506     style=wx.CENTER)
507 scaleMinTxt.SetFont(dateFont)
508 scaleMaxTxt = wx.StaticText(subpanelDisplay, -1, "Max",
509     style=wx.CENTER)
510 scaleMaxTxt.SetFont(dateFont)
511 scaleSizer.Add(scaleMinTxt, pos=(0,0),
512     flag=wx.LEFT|wx.ALIGN_CENTER_VERTICAL, border=5)
513 scaleSizer.Add(scaleMaxTxt, pos=(0,1),
514     flag=wx.LEFT|wx.ALIGN_CENTER_VERTICAL, border=5)
515
516 # controls
517 # analysis cut selector
518 self.xy = wx.RadioButton(subpanelDisplay, -1, 'X-Y',
519     style=wx.RB_GROUP)
520 self.xz = wx.RadioButton(subpanelDisplay, -1, 'X-Z')
521 self.yz = wx.RadioButton(subpanelDisplay, -1, 'Y-Z')
522 self.xy.SetFont(subFont)
523 self.xz.SetFont(subFont)
524 self.yz.SetFont(subFont)
525 self.xy.Bind(wx.EVT_RADIOBUTTON, self.OnCheck)
526 self.xz.Bind(wx.EVT_RADIOBUTTON, self.OnCheck)
527 self.yz.Bind(wx.EVT_RADIOBUTTON, self.OnCheck)
528 cutSizer.Add(self.xy, pos=(0,0), flag=wx.LEFT|wx.TOP, border=5)
529 cutSizer.Add(self.xz, pos=(0,1), flag=wx.LEFT|wx.TOP, border=5)

```

```

530     cutSizer.Add(self.yz, pos=(0,2), flag=wx.LEFT|wx.TOP, border=5)
531     self.xy.SetValue(True)
532
533     # slice selector
534     self.sliceSpin = wx.SpinCtrl(subpanelDisplay, -1, size=(60, -1))
535     self.sliceSpin.SetRange(0,self.gpt-1)
536     self.sliceSpin.SetValue(0)
537     self.sliceSpin.Bind(wx.EVT_SPIN, self.OnSlice)
538     self.sliceSpin.Bind(wx.EVT_TEXT, self.OnSlice)
539     cutSizer.Add(self.sliceSpin, pos=(0,3),
540                 flag=wx.LEFT|wx.ALIGN_CENTER_VERTICAL, border=8)
541
542     # scatterplot color selector
543     self.noColor = wx.RadioButton(subpanelDisplay, -1, 'None',
544                                  style=wx.RB_GROUP)
545     self.monthColor = wx.RadioButton(subpanelDisplay, -1, 'Month')
546     self.timeColor = wx.RadioButton(subpanelDisplay, -1, 'Hour')
547     self.noColor.SetFont(subFont)
548     self.monthColor.SetFont(subFont)
549     self.timeColor.SetFont(subFont)
550     self.noColor.Bind(wx.EVT_RADIOBUTTON, self.runCmap)
551     self.monthColor.Bind(wx.EVT_RADIOBUTTON, self.runCmap)
552     self.timeColor.Bind(wx.EVT_RADIOBUTTON, self.runCmap)
553     colorSizer.Add(self.noColor, pos=(0,0), flag=wx.LEFT, border=5)
554     colorSizer.Add(self.monthColor, pos=(0,1), flag=wx.LEFT, border=15)
555     colorSizer.Add(self.timeColor, pos=(0,2), flag=wx.LEFT, border=15)
556     self.colorStart = wx.TextCtrl(subpanelDisplay, value='0',size=(30, -1))
557     self.colorStart.Bind(wx.EVT_TEXT, self.runUpdate)
558     colorSizer.Add(self.colorStart, pos=(0,3), flag = wx.LEFT, border=15)
559     self.colorEnd = wx.TextCtrl(subpanelDisplay, value='0',size=(30, -1))
560     self.colorEnd.Bind(wx.EVT_TEXT, self.runUpdate)
561     colorSizer.Add(self.colorEnd, pos=(0,4),flag = wx.LEFT, border=5)

```

```

562     self.noColor.SetValue(True)
563
564     # colors for plots
565     self.facecolorBlue = '#0E689D'
566     self.facecolorRed = '#AC1D34'
567     self.edgecolor = '#ECE7F2'
568     self.gridcolor = '#737373'
569     self.axescolor = '#BDBDBD'
570     self.alpha = 1
571     self.lw = 0.2
572     self.scattersize = 15
573     self.cmap = cm.RdBu_r
574
575     # colormap scale selector
576     self.cmapMinDisplay = wx.TextCtrl(subpanelDisplay, value=' ',
577         size=(40, -1))
578     scaleSizer.Add(self.cmapMinDisplay, pos=(1,0), flag = wx.LEFT,
579         border=5)
580     self.cmapMaxDisplay = wx.TextCtrl(subpanelDisplay, value=' ',
581         size=(40, -1))
582     scaleSizer.Add(self.cmapMaxDisplay, pos=(1,1), flag = wx.LEFT,
583         border=5)
584     self.adjust = wx.Button(subpanelDisplay, -1, 'Adjust',
585         size=(65, -1))
586     self.adjust.Bind(wx.EVT_BUTTON, self.OnAdjustScale)
587     scaleSizer.Add(self.adjust, pos=(1,2), flag = wx.LEFT,
588         border=5)
589     self.auto = wx.Button(subpanelDisplay, -1, 'Auto',
590         size=(65, -1))
591     self.auto.Bind(wx.EVT_BUTTON, self.OnAutoScale)
592     scaleSizer.Add(self.auto, pos=(1,3), flag = wx.LEFT, border=0)
593

```

```

594     # put controls on folding sizer
595     sizerDisplay.AddSpacer(5)
596     sizerDisplay.Add(colorSizer, 0, wx.LEFT| wx.ALIGN_CENTER_VERTICAL, 5)
597     sizerDisplay.AddSpacer(7)
598     sizerDisplay.Add(cutSizer, 0, wx.LEFT| wx.ALIGN_CENTER_VERTICAL, 5)
599     sizerDisplay.AddSpacer(3)
600     sizerDisplay.Add(scaleTextSizer, 0, wx.LEFT|
601         wx.ALIGN_CENTER_VERTICAL, 10)
602     sizerDisplay.Add(scaleSizer, 0, wx.LEFT, 5)
603     sizerDisplay.AddSpacer(15)
604
605     # set folding sizer on folding panel
606     subpanelDisplay.SetSizer(sizerDisplay)
607     subpanelDisplay.Fit()
608
609     # add folding panel to foldpanelbar
610     self.fold_panel.AddFoldPanelWindow(sectDisplay, subpanelDisplay,
611         fpb.FPB_ALIGN_LEFT)
612     self.fold_panel.AddFoldPanelSeparator(sectDisplay)
613
614     # Fold panel bar - export data -----
615     sectExport = self.fold_panel.AddFoldPanel("Export Data",
616         collapsed=False, cbstyle=cs)
617     subpanelExport = wx.Panel(sectExport, -1)
618     sizerExport = wx.BoxSizer(wx.VERTICAL)
619
620     # controls
621     # print to .png
622     self.printPNG = wx.Button(subpanelExport, -1, 'Export Plots',
623         size=(100, -1))
624     self.printPNG.Bind(wx.EVT_BUTTON, self.OnPrint)
625     printSizer.Add(self.printPNG, 0, wx.ALIGN_RIGHT, border=0)

```

```

626
627     # put controls on folding sizer
628     sizerExport.AddSpacer(5)
629     sizerExport.Add(printSizer, 0, wx.LEFT | wx.ALIGN_LEFT, 5)
630     sizerExport.AddSpacer(10)
631
632     # set folding sizer on folding panel
633     subpanelExport.SetSizer(sizerExport)
634     subpanelExport.Fit()
635
636     # add folding panel to foldpanelbar
637     self.fold_panel.AddFoldPanelWindow(sectExport, subpanelExport,
638         fpb.FPB_ALIGN_LEFT)
639     self.fold_panel.AddFoldPanelSeparator(sectExport)
640
641     # Set Sizers -----
642     # make a sizer for the scrolling panel
643     scroll_sizer = wx.BoxSizer(wx.VERTICAL)
644     scroll_sizer.Add(self.fold_panel, 1, wx.EXPAND)
645
646     self.panel1.SetSizer(scroll_sizer)
647     self.panel1.Layout()
648
649     # panel2 sizer set-up
650     self.graphSizer = wx.BoxSizer(wx.VERTICAL)
651     self.panel2.SetSize((800, 760))
652     self.panel2.SetSizer(self.graphSizer)
653     self.panel2.Fit()
654     self.panel2.Centre()
655
656     # panel3 set-up
657     self.scatterSizer = wx.BoxSizer(wx.VERTICAL)

```

```

658     self.panel3.SetSize((800, 760))
659     self.panel3.SetSizer(self.scatterSizer)
660     self.panel3.Fit()
661     self.panel3.Centre()
662
663     # panel4 sizer set-up
664     self.keySizer = wx.BoxSizer(wx.VERTICAL)
665     self.panel4.SetSize((200, 150))
666     self.panel4.SetSizer(self.keySizer)
667     self.panel4.Fit()
668     self.panel4.Centre()
669
670     # Create matplotlib Object -----
671
672     # plot containers for panel2 heatmap
673     self.chart = wx.Panel(self.panel2)
674     self.fig = Figure(dpi=100, facecolor='none')
675     self.canvas = FigureCanvas(self.chart, -1, self.fig)
676     self.graphSizer.Add(self.chart, 0, wx.ALL|wx.EXPAND, 5)
677
678     # link heatmap plot containers to resize functions
679     self._SetSize()
680     self.canvas.draw()
681     self.chart._resizeflag = False
682     self.chart.Bind(wx.EVT_IDLE, self._onIdle)
683     self.chart.Bind(wx.EVT_SIZE, self._onSize)
684
685     # plot containers for panel3 heatmap
686     self.scatterChart = wx.Panel(self.panel3)
687     self.scatterFig = Figure(dpi=100, facecolor='none')
688     self.scatterCanvas = FigureCanvas(self.scatterChart, -1,
689         self.scatterFig)

```



```

690     self.scatterSizer.Add(self.scatterChart, 0, wx.ALL|wx.EXPAND, 5)
691
692     # link scatter plot containers to resize functions
693     self._SetSizeScatter()
694     self.scatterCanvas.draw()
695     self.scatterChart._resizeflag = False
696     self.scatterChart.Bind(wx.EVT_IDLE, self._onIdleScatter)
697     self.scatterChart.Bind(wx.EVT_SIZE, self._onSizeScatter)
698
699
700     # default variable values
701     self.idf = self.idfSel.GetValue()
702     self.csv = self.csvSel.GetValue()
703     self.cutNum = 0
704     self.ind = 0
705     self.axH = 'X'
706     self.axV = 'Y'
707
708     # Calculation Functions -----
709
710     def readIDF(self):
711         '''Parses .idf file to obtain zone geometry'''
712         values = []
713         valx = []
714         valy = []
715         winx = []
716         winy = []
717         winz = []
718
719         # find line numbers
720         inidf = open(self.idf,'r')
721         textidf = inidf.readlines()

```

```

722     zoneLN = textidf.index(' Zone,\r\n')
723     floorLN = textidf.index \
724         (' FLOOR,                !- Surface Type\r\n')
725     windowLN = textidf.index \
726         (' WINDOW,                !- Surface Type\r\n')
727     for i,line in enumerate(textidf):
728         if i >= zoneLN+3 and i < zoneLN+9 :
729             strip = line.lstrip();
730             split = strip.split(',');
731             new = split.pop(0)
732             values.append(new)
733     for i,line in enumerate(textidf):
734         if i >= floorLN+9 and i < floorLN+13 :
735             strip = line.lstrip();
736             split = strip.split(',');
737             xnew = split.pop(0)
738             valx.append(xnew)
739             ynew = split.pop(0)
740             valy.append(ynew)
741     for i,line in enumerate(textidf):
742         if i >= windowLN+9 and i < windowLN+13 :
743             strip = line.lstrip();
744             a = strip.replace(';',' ',' ')
745             split = a.split(',');
746             winxnew = split.pop(0)
747             winx.append(winxnew)
748             winynew = split.pop(0)
749             winy.append(winynew)
750             winznew = split.pop(0)
751             winz.append(winznew)
752
753     # pull zone dimensions

```

```

754     self.xmin = float(values.pop(0))
755     self.ymin = float(values.pop(0))
756     self.zmin = float(values.pop(0))
757     self.xmax = float(max(valx))
758     self.ymax = float(max(valy))
759     self.zmax = float(values.pop(2))
760     self.winxmax = float(max(winx))
761     self.winyamax = float(max(winy))
762     self.winzmax = float(max(winz))
763     self.winxmin = float(min(winx))
764     self.winymin = float(min(winy))
765     self.winzmin = float(min(winz))
766
767     # create range of gridpoints within zone dimensions
768     # sets start point to match Ecotect grid method:
769     # http://naturalfrequency.com/articles/analysisgrid
770     xspace = self.xmax/self.gpt
771     yspace = self.ymax/self.gpt
772     zspace = self.zmax/self.gpt
773     xstart = xspace/2
774     ystart = yspace/2
775     zstart = zspace/2
776
777     # number of gridpoints same in all dimensions
778     self.pt = mgrid[xstart:self.xmax:xspace,ystart:self.ymax:yspace,
779                   zstart:self.zmax:zspace]
780     self.xcoords = self.pt[0, :, 0, 0]
781     self.ycoords = self.pt[1, 0, :, 0]
782     self.zcoords = self.pt[2, 0, 0, :]
783
784     def getTemps(self):
785         '''Locates surface temps and other variables within .csv file'''

```

```

786     self.numbers = genfromtxt(self.csv, delimiter=',')
787     self.dates = genfromtxt(self.csv, delimiter=',', usecols=(0),
788         dtype='S100')
789     variables = (genfromtxt(self.csv, delimiter=',', dtype='S100'))[0,:]
790     location = [i for i, item in enumerate(variables) \
791         if re.search('Surface Inside Temperature', item)]
792     reorder=[2,3,4,0,6,5]
793     self.relocation = [location[i] for i in reorder]
794     self.loc_oDB = [i for i, item in enumerate(variables) \
795         if re.search('Outdoor Dry Bulb', item)]
796     self.loc_oRH = [i for i, item in enumerate(variables) \
797         if re.search('Outdoor Relative Humidity', item)]
798     self.loc_oAirV = [i for i, item in enumerate(variables) \
799         if re.search('Zone Outdoor Wind Speed', item)]
800     self.loc_zDB = [i for i, item in enumerate(variables) \
801         if re.search('Zone Mean Air Temperature', item)]
802     self.loc_zMRT = [i for i, item in enumerate(variables) \
803         if re.search('Zone Mean Radiant Temperature', item)]
804     self.loc_zOpT = [i for i, item in enumerate(variables) \
805         if re.search('Zone Operative Temperature', item)]
806     self.loc_zRH = [i for i, item in enumerate(variables) \
807         if re.search('Zone Air Relative Humidity', item)]
808     self.loc_zActSch = [i for i, item in enumerate(variables) \
809         if re.search('ACTIVITY_SCH:Schedule Value', item)]
810     self.loc_zCloSch = [i for i, item in enumerate(variables) \
811         if re.search('CLOTHING_SCH:Schedule Value', item)]
812     self.loc_zAirVSch = [i for i, item in enumerate(variables) \
813         if re.search('AIR_VELO_SCH:Schedule Value', item)]
814
815     def setDate_Annual(self):
816         '''Pulls data from appropriate located columns
817         in .csv file for every hour of the year'''

```

```

818     self.DBs = []
819     self.zoneDBs = []
820     self.months = []
821     self.temps = []
822     for position, item in enumerate(self.dates):
823         tem = self.numbers[position,self.relocation]
824         rep = repeat(tem,4)
825         self.temps.append(rep)
826         self.oDB = self.numbers[position,self.loc_oDB]
827         self.DBs.append(self.oDB)
828         self.zDB = self.numbers[position,self.loc_zDB]
829         self.zoneDBs.append(self.zDB)
830         self.months.append(item[0:2])
831     print shape(self.temps)
832     print 'SetDateAnnual done'
833
834 def viewFactors(self):
835     '''Determines view factors at every point in 3D grid'''
836     # locates each point in the space by comparing to max and min
837     xMx = self.xmax - self.pt[0,:,:,:]
838     xMn = self.pt[0,:,:,:] - self.xmin
839     yMx = self.ymax - self.pt[1,:,:,:]
840     yMn = self.pt[1,:,:,:] - self.ymin
841     zMx = self.zmax - self.pt[2,:,:,:]
842     zMn = self.pt[2,:,:,:] - self.zmin
843
844     # for vertical surfaces (walls) -----
845     # array of a/c and b/c for each surface at each point
846     acbcVert = array([
847     # north wall
848     [[ xMn/yMx, zMn/yMx],[ xMn/yMx,  zMx/yMx],[ xMx/yMx,  zMx/yMx],
849     [ xMx/yMx, zMn/yMx]],

```

```

850 # east wall
851 [[ yMx/xMx, zMn/xMx],[ yMx/xMx, zMx/xMx],[ yMn/xMx, zMx/xMx],
852     [ yMn/xMx, zMn/xMx]],
853 # south wall
854 [[ xMn/yMn, zMn/yMn],[ xMn/yMn, zMx/yMn],[ xMx/yMn, zMx/yMn],
855     [ xMx/yMn, zMn/yMn]],
856 # west wall
857 [[ yMn/xMn, zMn/xMn],[ yMn/xMn, zMx/xMn],[ yMx/xMn, zMx/xMn],
858     [ yMx/xMn, zMn/xMn]],
859 ])
860 # for all surfaces, all points, a/c only
861 acV = acbcVert[:, :, 0, :, :, :]
862 # for all surfaces, all points, b/c only
863 bcV = acbcVert[:, :, 1, :, :, :]
864 # calculate angle factors at each point
865 tauV = 1.24186+0.16730*(acV)
866 gammaV = 0.61648+(0.08165*(bcV))+(0.05128*(acV))
867 FV = 0.120*(1-exp(-(acV)/tauV))*(1-exp(-(bcV)/gammaV))
868
869 # for horizontal surfaces (floor, ceiling) -----
870 # ceiling
871 acbcHori = array([
872     [[ xMn/zMx, yMn/zMx],[ xMn/zMx, yMx/zMx],[ xMx/zMx, yMx/zMx],
873         [ xMx/zMx, yMn/zMx]],
874     # floor
875     [[ xMn/zMn, yMn/zMn],[ xMn/zMn, yMx/zMn],[ xMx/zMn, yMx/zMn],
876         [ xMx/zMn, yMn/zMn]],
877 ])
878 # for all surfaces, all points, a/c only
879 acH = acbcHori[:, :, 0, :, :, :]
880 # for all surfaces, all points, b/c only
881 bcH = acbcHori[:, :, 1, :, :, :]

```

```

882
883     # calculate angle factors at each point
884     tauH = 1.59512+0.12788*(acH)
885     gammaH = 1.22643+(0.04621*(bcH))+(0.04434*(acH))
886     FH = 0.116*(1-exp(-(acH)/tauH))*(1-exp(-(bcH)/gammaH))
887     # combine FV and FH
888     self.F = concatenate((FV,FH),axis=0)
889
890     # for reference pull angle factors and find max and min
891     SS = size(self.pt[0,:,:,:])
892     AR = reshape(self.F,(24,SS))
893     aFacs = apply_along_axis(sum, 0, AR)
894     print 'calcViewFactors done'
895
896     # Metric Annual Calculation Functions -----
897     ''' Calculates comfort values at every point at every hour of the
898         year for every metric. Index functions then index off of that
899         for user-requested times.
900     '''
901
902     def calcMRTS(self):
903         print 'Running Mean Radiant Temperature Calculation...'
904         # reshape afacs
905         SX = size(self.pt[0,:,0,0])
906         SY = size(self.pt[0,0,:,0])
907         SZ = size(self.pt[0,0,0,:])
908         RF = reshape(self.F,(24,SX,SY,SZ))
909         # reformat temps
910         AT = asarray(self.temps)
911         # multiply by surface temps and sum to get MRT at each point
912         listMRTS = ([])
913         for position, item in enumerate(AT):

```

```

914         ZR = RF*item[:,newaxis,newaxis,newaxis]
915         MRTResults = sum(ZR, axis=0)
916         listMRTS.append(MRTResults)
917     self.MRTS_Annual = asarray(listMRTS)
918     print 'SHAPE MRTS', shape(self.MRTS_Annual)
919     print 'Finished Mean Radiant Temperature Calculation'
920
921     def calcOpTemp(self):
922         '''Calculates operative temperature according to method in
923         ASHRAE 55-2004'''
924         print 'Starting Operative Temperature Calculation...'
925         collectOpTemps = []
926         for i in range(0,len(self.numbers)):
927             if self.numbers[i,self.loc_zAirVSch] < 0.2:
928                 OpTemp = (0.5*(self.numbers[i,self.loc_zDB]))+ \
929                     (1-0.5)*(self.MRTS_Annual[i])
930             if self.numbers[i,self.loc_zAirVSch] >= 0.2 and \
931                 self.numbers[i,self.loc_zAirVSch] < 0.6:
932                 OpTemp = (0.5*(self.numbers[i,self.loc_zDB]))+ \
933                     (1-0.5)*(self.MRTS_Annual[i])
934             else:
935                 OpTemp = (0.5*(self.numbers[i,self.loc_zDB]))+ \
936                     (1-0.5)*(self.MRTS_Annual[i])
937             collectOpTemps.append(OpTemp)
938         self.OpTemp_Annual = asarray(collectOpTemps)
939         print 'Finished Operative Temperature Calculation'
940
941
942     def calcPMVPPDRoomAverageAnnual(self):
943         ''' provides values for PMV scatter plot for every hour of the year
944         PMV spatial calc done on-demand using 'Run cMap' button
945         from ISO 7730-2005:

```


946 M is metabolic rate [W/m²]
 947 W is effective mechanical power [W/m²]
 948 I_{cl} is clothing insulation [m²K/W]
 949 f_{cl} is clothing surface factor
 950 t_a is air temperature [C]
 951 t_r is mean radiant temperature [C]
 952 var is relative air velocity [m/s]
 953 p_a is water vapour partial pressure [Pa]
 954 h_c is convective heat transfer coefficient [W/m²K]
 955 t_{cl} is clothing surface temperature [C]

956
 957 note: 1 met = 58.2 W/m²; 1 clo = 0.155 m²C/W
 958

959 PMV may be calculated for different combinations of
 960 metabolic rate, clothin insulation, air temperature,
 961 mean radiant temperature, air velocity, and humidity.
 962 The equations for t_{cl} and h_c may be solved by iteration.
 963

964 The PMV index should be used only for values of PMV
 965 between -2 and +2, and when the six main parameters are
 966 within the following intervals:
 967

968 0.8 met < M < 4 met (46 W/m² to 232 W/m²)
 969 0 clo < I_{cl} < 2 clo (0 m²K/W to 0.310 m²K/W)
 970 10 oC < t_a < 30 oC
 971 10 oC < t_r < 30 oC
 972 0 m/s < var < 1 m/s
 973 0 Pa < p_a < 2700 Pa

974 ' ' '

975 self.PMV_RoomAverageAnnual = []
 976 self.PPD_RoomAverageAnnual = []
 977 roomAveragesMRT = []

```

978     for i in range(0,len(self.MRTS_Annual)):
979         results = mean(self.MRTS_Annual[i])
980         roomAveragesMRT.append(results)
981     print 'len roomAverages MRT',len(roomAveragesMRT)
982
983     for position, item in enumerate(self.dates):
984         # find variable values
985         self.zDB = self.numbers[position,self.loc_zDB]
986         self.zRH = self.numbers[position,self.loc_zRH]
987         self.zActSch = self.numbers[position,self.loc_zActSch]
988         self.zCloSch = self.numbers[position,self.loc_zCloSch]
989         self.zAirVSch = self.numbers[position,self.loc_zAirVSch]
990
991         # set variables
992         ta = self.zDB
993         RH = self.zRH/100
994         var = self.zAirVSch
995         met = self.zActSch/58.15
996         clo = self.zCloSch
997         print 'set variables ok'
998
999         # auto calc variables
1000         Icl = clo*0.155
1001         M = met*58.15
1002         W = 0
1003         MW = M-W
1004         print 'auto calc variables ok'
1005
1006         # steam table data - from 2.006 Property Data Tables
1007         steamTableTemps = [
1008             0.01, 5, 10, 15, 20, 25, 30, 35,
1009             40, 45, 50, 55, 60, 65, 70, 75,

```

```

1010     80, 85, 90, 95, 100
1011 ]
1012 steamTablePsat = [
1013     611.66, 872.58, 1228.2, 1705.8, 2339.3, 3169.9, 4247, 5629,
1014     7384.9, 9595, 12352, 15762, 19946, 25042, 31201, 38595,
1015     47414, 57867, 70182, 84608, 101420
1016 ]
1017 # calculate saturation pressure at ta
1018 Psat = interp(ta, steamTableTemps, steamTablePsat)
1019 # calculate water vapor partial pressure
1020 pa = RH*Psat
1021 print 'steam tables ok'
1022
1023 # set fcl
1024 if Icl <= 0.078:
1025     fcl = 1.00 + 1.290*Icl
1026 else:
1027     fcl = 1.05 + 0.645*Icl
1028     print 'set fcl ok'
1029
1030 def findTcl(tcl):
1031     g = 2.38*abs(tcl-ta)**0.25
1032     h = 12.1*sqrt(var)
1033     hc = min(g,h)
1034     b = 35.7-0.028*MW-Icl*(3.96*(10**-8)*fcl*(((tcl+273)**4)- \
1035         ((mrt+273)**4))+fcl*hc*(tcl-ta))-tcl
1036     return b
1037
1038 # determine Tcl from bisection search (brentq method)
1039 mrt = roomAveragesMRT[position]
1040 Tcl = brentq(findTcl, 0, 100)
1041 g = 2.38*abs(Tcl-ta)**0.25

```

```

1042         h = 12.1*sqrt(var)
1043         hc = max(g,h)
1044
1045         # heat loss components by parts -----
1046
1047         # heat loss diff through skin
1048         HL1 = 3.05*0.001*(5733-6.99*MW-pa)
1049         # heat loss by sweating
1050         if MW > 58.15:
1051             HL2 = 0.42*(MW-58.15)
1052         else:
1053             HL2 = 0
1054         # latent respiration heat loss
1055         HL3 = 1.7*0.00001*M*(5867-pa)
1056         # dry respiration heat loss
1057         HL4 = 0.0014*M*(34-ta)
1058         # heat loss by radiation
1059         HL5 = 3.96*0.00000001*fcl*(((Tcl+273)**4)-((mrt+273)**4))
1060         # heat loss by convection
1061         HL6 = fcl*hc*(Tcl-ta)
1062         # thermal sensation trans coeff
1063         TS = 0.303*exp(-0.036*M)+0.028
1064         # calc PMV
1065         PMV = TS*(MW-HL1-HL2-HL3-HL4-HL5-HL6)
1066         self.PMV_RoomAverageAnnual.append(PMV)
1067         PPD = 100-95*exp(-0.03353*PMV**4-0.2179*PMV**2)
1068         self.PPD_RoomAverageAnnual.append(PPD)
1069         print 'shape PMVavg', shape(self.PMV_RoomAverageAnnual)
1070         print 'calc PMV_RoomAverageAnnual ok'
1071
1072     def calcAdaptiveASHRAE55(self):
1073         print 'Starting ASHRAE 55 Adaptive Temperature Calculation...'

```

```

1074 # pull outdoor dry bulb temps and month names from CSV
1075 csvOutdoorDryBulbs=[]
1076 csvMonths=[]
1077 for i in range(0,len(self.dates)):
1078     oDB = self.numbers[i,self.loc_oDB]
1079     csvOutdoorDryBulbs.append(oDB)
1080     csvMonths.append(self.dates[i][0:2])
1081 print "len csvOutdoorDryBulb", len(csvOutdoorDryBulbs)
1082 print "len csvMonths", len(csvMonths)
1083
1084 # chunk data into months and days to find the monthly average
1085 # of the daily averages
1086 u = unique(csvMonths)
1087 chunks = []
1088 for i in range(0,len(u)-1):
1089     itemindex = csvMonths.index(u[i])
1090     print "itemindex", itemindex
1091     chunks.append(itemindex)
1092 chunks.append(len(csvMonths))
1093 meanMonthlyOutdoorDryBulbs = []
1094 for i in range(0,len(chunks)-1):
1095     x = (csvOutdoorDryBulbs[chunks[i]:chunks[i+1]])
1096     days = zip(*(iter(x),) * 24)
1097     dailyAverages = []
1098     for k in days:
1099         dailyAverages.append(mean(k))
1100     meanMonthlyOutdoorDryBulbs.append(mean(dailyAverages))
1101
1102 # replace months with corresponding average dry bulb
1103 dictionary = dict(zip(u, meanMonthlyOutdoorDryBulbs))
1104 self.replacedODBforMean = [dictionary.get(x,x) for x in csvMonths]
1105 self.replacedODBforMean = asarray(self.replacedODBforMean[1:],

```

```

1106         dtype=np.float)
1107     print "len replacedODBforMean", len(self.replacedODBforMean)
1108     dummyRow = np.array([0])
1109     self.concatReplacedODBforMean = \
1110         concatenate((dummyRow,self.replacedODBforMean))
1111     print self.concatReplacedODBforMean
1112     print "len concatenate", len(self.concatReplacedODBforMean)
1113
1114     # calculate Tcomf
1115     Tcomfs = []
1116     for i in range(0,len(self.concatReplacedODBforMean)):
1117         if self.concatReplacedODBforMean[i] < 10:
1118             Tcomf = 22
1119         else:
1120             Tcomf = 0.31*self.concatReplacedODBforMean[i]+17.8
1121         Tcomfs.append(Tcomf)
1122     print "len Tcomfs", len(Tcomfs)
1123
1124     # find deltaT
1125     collectDeltaT = []
1126     for i in range(0, len(Tcomfs)):
1127         deltaT = subtract(self.OpTemp_Annual[i],Tcomfs[i])
1128         collectDeltaT.append(deltaT)
1129     self.AdaptiveASHRAE55_Annual = asarray(collectDeltaT)
1130     print 'Finished ASHRAE 55 Adaptive Temperature Calculation'
1131
1132
1133     def calcAdaptiveEN15251(self):
1134         print 'Starting EN15251 Adaptive Temperature Calculation...'
1135         # chunk outdoor dry bulb temps from CSV into 24 hour blocks
1136         x = self.numbers[1:,self.loc_oDB]
1137         print len(x)

```

```

1138     print x[0]
1139     days = zip(*(iter(x),) * 24)
1140     print len(days)
1141     print days[0]
1142     print len(days[0])
1143     # for each day, find average temps
1144     dailyAverages = []
1145     for i in days:
1146         dailyAverages.append(mean(i))
1147     print len(dailyAverages)
1148     print "DA zero",dailyAverages[0]
1149     # perform weighted running mean every 7 days
1150     collectTrm7 = []
1151     for i in range(0,len(dailyAverages)):
1152         T1m = dailyAverages[i-1]
1153         T2m = dailyAverages[i-2]
1154         T3m = dailyAverages[i-3]
1155         T4m = dailyAverages[i-4]
1156         T5m = dailyAverages[i-5]
1157         T6m = dailyAverages[i-6]
1158         T7m = dailyAverages[i-7]
1159         Trm7 = (T1m + 0.8*T2m + 0.6*T3m + 0.5*T4m + 0.4*T5m + \
1160                0.3*T6m + 0.2*T7m)/3.8
1161         collectTrm7.append(Trm7)
1162     self.Trm7s = repeat(collectTrm7, 24)
1163     print "LEN self.Trm7s", len(self.Trm7s)
1164     # calculate Tcomf
1165     Tcomfs = []
1166     for i in range(0,len(collectTrm7)):
1167         if collectTrm7[i] < 10:
1168             Tcomf = 22
1169         else:

```

```

1170         Tcomf = 0.33*collectTrm7[i]+18.8
1171         Tcomfs.append(Tcomf)
1172     print "len OLD Tcomfs", len(Tcomfs)
1173     # repeat 24 times to make it the same length as the OpTemps
1174     Tcomfs = repeat(Tcomfs, 24)
1175     print "len NEW Tcomfs", len(Tcomfs)
1176     dummyRow = array([0])
1177     Tcomfs = concatenate((dummyRow,Tcomfs))
1178     print "len NEW CONCAT Tcomfs", len(Tcomfs)
1179
1180     # find deltaT
1181     collectDeltaT = []
1182     for i in range(0, len(Tcomfs)):
1183         deltaT = subtract(self.OpTemp_Annual[i],Tcomfs[i])
1184         collectDeltaT.append(deltaT)
1185     print 'Shape collect Delta T', shape(collectDeltaT)
1186     self.AdaptiveEN15251_Annual = asarray(collectDeltaT)
1187     print 'Finished EN15251 Adaptive Temperature Calculation'
1188
1189
1190     def calcAdaptiveNPRCR1752Beta(self):
1191         print 'Starting NPR-CR 1752 Adaptive Temperature Calculation...'
1192         # chunk outdoor dry bulb temps from CSV into 24 hour blocks
1193         x = self.numbers[1:,self.loc_oDB]
1194         print len(x)
1195         print x[0]
1196         days = zip(*(iter(x),) * 24)
1197         print len(days)
1198         print days[0]
1199         print len(days[0])
1200         # for each day, find average temps
1201         dailyAverageofMaxMin = []

```



```

1202     for i in days:
1203         mx = max(i)
1204         mn = min(i)
1205         dailyAverageofMaxMin.append(mean([mx, mn]))
1206     print len(dailyAverageofMaxMin)
1207     print "DA zero",dailyAverageofMaxMin[0]
1208     # perform weighted running mean every 7 days
1209     collectTrm3 = []
1210     for i in range(0,len(dailyAverageofMaxMin)):
1211         T1m = dailyAverageofMaxMin[i]
1212         T2m = dailyAverageofMaxMin[i-1]
1213         T3m = dailyAverageofMaxMin[i-2]
1214         T4m = dailyAverageofMaxMin[i-3]
1215         Trm3 = (T1m + 0.8*T2m + 0.4*T3m + 0.2*T4m)/2.4
1216         collectTrm3.append(Trm3)
1217     self.Trm3s = repeat(collectTrm3, 24)
1218     print "LEN self.Trm3s", len(self.Trm3s)
1219     # calculate Tcomf
1220     Tcomfs = []
1221     for i in range(0,len(collectTrm3)):
1222         if collectTrm3[i] < 10:
1223             Tcomf = 22
1224         else:
1225             Tcomf = 0.31*collectTrm3[i]+17.8
1226         Tcomfs.append(Tcomf)
1227     print "len OLD Tcomfs", len(Tcomfs)
1228     # repeat 24 times to make it the same length as the OpTemps
1229     Tcomfs = repeat(Tcomfs, 24)
1230     print "len NEW Tcomfs", len(Tcomfs)
1231     dummyRow = array([0])
1232     Tcomfs = concatenate((dummyRow,Tcomfs))
1233     print "len NEW CONCAT Tcomfs", len(Tcomfs)

```

```

1234
1235     # find deltaT
1236     collectDeltaT = []
1237     for i in range(0, len(Tcomfs)):
1238         deltaT = subtract(self.OpTemp_Annual[i],Tcomfs[i])
1239         collectDeltaT.append(deltaT)
1240     print 'Shape collect Delta T', shape(collectDeltaT)
1241     self.AdaptiveNPRCR1752Beta_Annual = asarray(collectDeltaT)
1242     print 'Finished NPR-CR 1752 Adaptive Temperature Calculation'
1243
1244     # Metric Indexing & Display Calculation Functions -----
1245
1246     # indexing functions for single point in time -----
1247     def setDate_Point(self):
1248         self.StartMonth = '%02d' % (int(self.StartMonthSpin.GetValue()))
1249         self.StartDay = '%02d' % (int(self.StartDaySpin.GetValue()))
1250         self.StartHour = '%02d' % (int(self.StartHourSpin.GetValue()))
1251         i = self.calcDrop.GetSelection()
1252         # note that for PMV and PPD, currently reference off MRTS
1253         calctype = [ self.MRTS_Annual,
1254                     self.OpTemp_Annual,
1255                     self.MRTS_Annual,
1256                     self.MRTS_Annual,
1257                     self.AdaptiveASHRAE55_Annual,
1258                     self.AdaptiveEN15251_Annual,
1259                     self.AdaptiveNPRCR1752Beta_Annual
1260                 ]
1261         calcselection = calctype[i]
1262         calcselectionOpTemps = calctype[1]
1263         self.outdoorDB = []
1264         for position, item in enumerate(self.dates):
1265             if (item[0:2] == self.StartMonth) and \

```

```

1266         (item[3:5] == self.StartDay) and \
1267         (item[7:9] == self.StartHour) :
1268             self.position = position
1269             print self.position
1270             self.calcSelection_Point = calcselection[position,:,:,:]
1271             self.calcSelection_Point_OpTemps = calcselectionOpTemps \
1272                 [position,:,:,:]
1273             # comfort parameters
1274             self.oDB = self.numbers[position,self.loc_oDB]
1275             self.outdoorDB.append(self.oDB)
1276             self.oRH = self.numbers[position,self.loc_oRH]
1277             self.oAirV = self.numbers[position,self.loc_oAirV]
1278             self.zDB = self.numbers[position,self.loc_zDB]
1279             self.zMRT = self.numbers[position,self.loc_zMRT]
1280             self.zOpT = self.numbers[position,self.loc_zOpT]
1281             self.zRH = self.numbers[position,self.loc_zRH]
1282             self.zActSch = self.numbers[position,self.loc_zActSch]
1283             self.zCloSch = self.numbers[position,self.loc_zCloSch]
1284             self.zAirVSch = self.numbers[position,self.loc_zAirVSch]
1285             self.monthlyOutdoorMean = \
1286                 self.concatReplacedODBforMean[position]
1287             self.runningMean7day = self.Trm7s[position]
1288             self.runningMean3day = self.Trm3s[position]
1289         print 'SetDate_Point done'
1290
1291     def indexMRTS_Point(self):
1292         # call indexing function
1293         self.setDate_Point()
1294         # define heatmap and scatter results
1295         self.heatmapCalcResults = self.calcSelection_Point
1296         self.scatterCalcResults = mean(self.calcSelection_Point)
1297         # display settings

```

```

1298     self.format = '%2.1f'
1299     self.xlabel = 'outdoor dry bulb temperature [C]'
1300     self.ylabel = 'indoor mean radiant temperature [C]'
1301     self.scattertitle = "Mean Radiant Temperature vs Outdoor Dry Bulb"
1302     self.scatterXlim = [-10,35]
1303     self.scatterYlim = [16,32]
1304
1305     def indexOpTemp_Point(self):
1306         # call indexing function
1307         self.setDate_Point()
1308         # define heatmap and scatter results
1309         self.heatmapCalcResults = self.calcSelection_Point
1310         self.scatterCalcResults = mean(self.calcSelection_Point)
1311         # display settings
1312         self.format = '%2.1f'
1313         self.xlabel = 'outdoor dry bulb temperature [C]'
1314         self.ylabel = 'indoor operative temperature [C]'
1315         self.scattertitle = "Operative Temperature vs Outdoor Dry Bulb"
1316         self.scatterXlim = [-10,35]
1317         self.scatterYlim = [16,32]
1318
1319     def calcPMV_Point(self):
1320         '''provides values for PMV heatmap for a single point
1321            in time
1322         '''
1323         self.setDate_Point()
1324         self.scatterCalcResultsPMV = self.PMV_RoomAverageAnnual[self.position]
1325         self.scatterCalcResultsPPD = self.PPD_RoomAverageAnnual[self.position]
1326
1327         # set variables
1328         ta = self.zDB
1329         RH = self.zRH/100

```

```

1330     var = self.zAirVSch
1331     met = self.zActSch/58.15
1332     clo = self.zCloSch
1333
1334     # auto calc variables
1335     Icl = clo*0.155
1336     M = met*58.15
1337     W = 0
1338     MW = M-W
1339
1340     # steam table data - from 2.006 Property Data Tables
1341     steamTableTemps = [
1342     0.01, 5, 10, 15, 20, 25, 30, 35,
1343     40, 45, 50, 55, 60, 65, 70, 75,
1344     80, 85, 90, 95, 100
1345     ]
1346     steamTablePsat = [
1347     611.66, 872.58, 1228.2, 1705.8, 2339.3, 3169.9, 4247, 5629,
1348     7384.9, 9595, 12352, 15762, 19946, 25042, 31201, 38595,
1349     47414, 57867, 70182, 84608, 101420
1350     ]
1351     # calculate saturation pressure at ta
1352     Psat = interp(ta, steamTableTemps, steamTablePsat)
1353     # calculate water vapor partial pressure
1354     pa = RH*Psat
1355
1356     # set fcl
1357     if Icl <= 0.078:
1358         fcl = 1.00 + 1.290*Icl
1359     else:
1360         fcl = 1.05 + 0.645*Icl
1361

```

```

1362     def findTcl(tcl):
1363         g = 2.38*abs(tcl-ta)**0.25
1364         h = 12.1*sqrt(var)
1365         hc = min(g,h)
1366         b = 35.7-0.028*MW-Icl*(3.96*(10**-8)*fcl*(((tcl+273)**4)- \
1367             ((i+273)**4))+fcl*hc*(tcl-ta))-tcl
1368         return b
1369
1370     # determine Tcl
1371     collectTcl = []
1372     collectHC = []
1373     for i in ravel(self.calcSelection_Point):
1374         Tcl = brentq(findTcl, 0, 100)
1375         collectTcl.append(Tcl)
1376         g = 2.38*abs(Tcl-ta)**0.25
1377         h = 12.1*sqrt(var)
1378         hc = max(g,h)
1379         collectHC.append(hc)
1380
1381     # heat loss components by parts -----
1382
1383     self.collectPMV = []
1384     self.collectPPD = []
1385     for i in range(0,len(ravel(self.calcSelection_Point))):
1386         tr = (ravel(self.calcSelection_Point))[i]
1387         Tcl = collectTcl[i]
1388         hc = collectHC[i]
1389
1390         # heat loss diff through skin
1391         HL1 = 3.05*0.001*(5733-6.99*MW-pa)
1392         # heat loss by sweating
1393         if MW > 58.15:

```

```

1394         HL2 = 0.42*(MW-58.15)
1395     else:
1396         HL2 = 0
1397     # latent respiration heat loss
1398     HL3 = 1.7*0.00001*M*(5867-pa)
1399     # dry respiration heat loss
1400     HL4 = 0.0014*M*(34-ta)
1401     # heat loss by radiation
1402     HL5 = 3.96*0.00000001*fcl*(((Tcl+273)**4)-((tr+273)**4))
1403     # heat loss by convection
1404     HL6 = fcl*hc*(Tcl-ta)
1405     # thermal sensation trans coeff
1406     TS = 0.303*exp(-0.036*M)+0.028
1407     # calc PMV
1408     PMV = TS*(MW-HL1-HL2-HL3-HL4-HL5-HL6)
1409     self.collectPMV.append(PMV)
1410     PPD = 100-95*exp(-0.03353*PMV**4-0.2179*PMV**2)
1411     self.collectPPD.append(PPD)
1412
1413     self.heatmapCalcResults = reshape(self.collectPMV,
1414         shape(self.calcSelection_Point))
1415
1416     # display items
1417     r = around(self.heatmapCalcResults, decimals=2)
1418     u = unique(r)
1419     self.Tickvalues = u
1420     self.Tickvalues = [-3.0, -2.0, -1.0, -0.5, 0, 0.5, 1.0, 2.0, 3.0]
1421     self.Ticklabels = ['-3', '-2', '-1', '-0.5', '0', '+0.5',
1422         '+1', '+2', '+3']
1423     self.Levels = len(u)
1424     self.format = '%2.2f'
1425     self.scattertitle = \

```

```

1426         "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
1427     self.xlabel = 'predicted mean vote (PMV)'
1428     self.ylabel = 'predicted percentage of dissatisfied (PPD)'
1429     self.scatterXlim = [-3,3]
1430     self.scatterYlim = [0,100]
1431
1432     def calcPPD_Point(self):
1433         '''provides values for PPD heatmap for a single point
1434             in time
1435         '''
1436         self.setDate_Point()
1437         self.calcPMV_Point()
1438         self.scatterCalcResultsPMV = \
1439             self.PMV_RoomAverageAnnual[self.position]
1440         self.scatterCalcResultsPPD = \
1441             self.PPD_RoomAverageAnnual[self.position]
1442         self.heatmapCalcResults = \
1443             reshape(self.collectPPD,shape(self.calcSelection_Point))
1444
1445         # display items
1446         r = around(self.heatmapCalcResults, decimals=2)
1447         u = unique(r)
1448         self.Tickvalues = u
1449         self.Tickvalues = [0, 5, 10, 15, 20, 50]
1450         self.Ticklabels = ['0%', '5%', '10%', '15%', '20%', '>20%']
1451         self.Levels = len(u)
1452         self.format = '%2.2f'
1453         self.scattertitle = \
1454             "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
1455         self.xlabel = 'predicted mean vote (PMV)'
1456         self.ylabel = 'predicted percentage of dissatisfied (PPD)'
1457         self.scatterXlim = [-3,3]

```



```

1458     self.scatterYlim = [0,100]
1459
1460 def indexASHRAE55Adaptive_Point(self):
1461     # call indexing function
1462     self.setDate_Point()
1463     # define heatmap and scatter results
1464     self.heatmapCalcResults = self.calcSelection_Point
1465     self.scatterCalcResults = mean(self.calcSelection_Point_OpTemps)
1466     # display settings
1467     r = around(self.heatmapCalcResults, decimals=1)
1468     u = unique(r)
1469     self.Tickvalues = [-10, -3.5, -2.5, 2.5, 3.5, 10]
1470     self.Ticklabels = [
1471         '<80% acceptability',
1472         '80% acceptability \n $\Delta$ T -3.5 [C]',
1473         '90% acceptability \n $\Delta$ T -2.5 [C]',
1474         '90% acceptability \n $\Delta$ T +2.5 [C]',
1475         '80% acceptability \n $\Delta$ T +3.5 [C]',
1476         '< 80% acceptability']
1477     self.Levels = len(u)
1478     self.format = '%2.1f'
1479     # scatter plot lines
1480     self.scattertitle = \
1481         "ASHRAE Standard 55 Adaptive Comfort - Room Average"
1482     self.xlabel = 'mean monthly outdoor temperature [C]'
1483     self.ylabel = 'indoor operative temperature [C]'
1484     self.scatterXlim = [5,35]
1485     self.scatterYlim = [16,32]
1486     # scatter plot boundary lines
1487     self.outdoor = arange(10,33)
1488     self.comfortASHRAE = (0.31*self.outdoor+17.8)
1489     self.lower80ASHRAE = (0.31*self.outdoor+17.8)-3.5

```

```

1490     self.lower90ASHRAE = (0.31*self.outdoor+17.8)-2.5
1491     self.upper90ASHRAE = (0.31*self.outdoor+17.8)+2.5
1492     self.upper80ASHRAE = (0.31*self.outdoor+17.8)+3.5
1493
1494     def indexEN15251Adaptive_Point(self):
1495         # call indexing function
1496         self.setDate_Point()
1497         # define heatmap and scatter results
1498         self.heatmapCalcResults = self.calcSelection_Point
1499         self.scatterCalcResults = mean(self.calcSelection_Point_OpTemps)
1500         # display settings
1501         r = around(self.heatmapCalcResults, decimals=1)
1502         u = unique(r)
1503         self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
1504         self.Ticklabels = [
1505             '<80% acceptability',
1506             '80% acceptability \n $\Delta$ T -3.0 [C]',
1507             '90% acceptability \n $\Delta$ T -2.0 [C]',
1508             '90% acceptability \n $\Delta$ T +2.0 [C]',
1509             '80% acceptability \n $\Delta$ T +3.0 [C]',
1510             '< 80% acceptability']
1511         self.Levels = len(u)
1512         self.format = '%2.1f'
1513         # scatter plot lines
1514         self.scattertitle = \
1515             "European Standard EN 15251 Adaptive Comfort - Room Average"
1516         self.xlabel = '7-day running mean outdoor temperature [C]'
1517         self.ylabel = 'indoor operative temperature [C]'
1518         self.scatterXlim = [5,35]
1519         self.scatterYlim = [16,32]
1520         # scatter plot boundary lines
1521         self.outdoor = arange(10,33)

```

```

1522     self.comfortEN15251 = (0.33*self.outdoor+18.8)
1523     self.lower80EN15251 = (0.33*self.outdoor+18.8)-3.0
1524     self.lower90EN15251 = (0.33*self.outdoor+18.8)-2.0
1525     self.upper90EN15251 = (0.33*self.outdoor+18.8)+2.0
1526     self.upper80EN15251 = (0.33*self.outdoor+18.8)+3.0
1527
1528     def indexNPRCR1752BetaAdaptive_Point(self):
1529         # call indexing function
1530         self.setDate_Point()
1531         # define heatmap and scatter results
1532         self.heatmapCalcResults = self.calcSelection_Point
1533         self.scatterCalcResults = mean(self.calcSelection_Point_OpTemps)
1534         # display settings
1535         r = around(self.heatmapCalcResults, decimals=1)
1536         u = unique(r)
1537         self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
1538         self.Ticklabels = [
1539             '<80% acceptability',
1540             '80% acceptability \n $\Delta$ T -3.0 [C]',
1541             '90% acceptability \n $\Delta$ T -2.0 [C]',
1542             '90% acceptability \n $\Delta$ T +2.0 [C]',
1543             '80% acceptability \n $\Delta$ T +3.0 [C]',
1544             '< 80% acceptability']
1545         self.Levels = len(u)
1546         self.format = '%2.1f'
1547         # scatter plot lines
1548         self.scattertitle = \
1549             "Dutch Standard NPR-CR 1752 Type Beta Adaptive Comfort"
1550         self.xlabel = '3-day running mean outdoor temperature [C]'
1551         self.ylabel = 'indoor operative temperature [C]'
1552         self.scatterXlim = [5,35]
1553         self.scatterYlim = [16,32]

```

```

1554     # scatter plot boundary lines
1555     self.outdoor = arange(10,33)
1556     self.comfortNPRCR1752 = (0.31*self.outdoor+17.8)
1557     self.lower80NPRCR1752 = (0.31*self.outdoor+17.8)-3.0
1558     self.lower90NPRCR1752 = (0.31*self.outdoor+17.8)-2.0
1559     self.upper90NPRCR1752 = (0.31*self.outdoor+17.8)+2.0
1560     self.upper80NPRCR1752 = (0.31*self.outdoor+17.8)+3.0
1561
1562
1563     # indexing functions for date/time range -----
1564     def setDate_Range(self):
1565         self.StartMonth = '%02d' % (int(self.StartMonthSpin.GetValue()))
1566         self.StartDay = '%02d' % (int(self.StartDaySpin.GetValue()))
1567         self.StartHour = '%02d' % (int(self.StartHourSpin.GetValue()))
1568         self.EndMonth = '%02d' % (int(self.EndMonthSpin.GetValue()))
1569         self.EndDay = '%02d' % (int(self.EndDaySpin.GetValue()))
1570         self.EndHour = '%02d' % (int(self.EndHourSpin.GetValue()))
1571         i = self.calcDrop.GetSelection()
1572         calctype = [ self.MRTS_Annual,
1573                     self.OpTemp_Annual,
1574                     self.MRTS_Annual,
1575                     self.MRTS_Annual,
1576                     self.AdaptiveASHRAE55_Annual,
1577                     self.AdaptiveEN15251_Annual,
1578                     self.AdaptiveNPRCR1752Beta_Annual
1579                 ]
1580         calcselection = calctype[i]
1581         calcselectionOpTemps = calctype[1]
1582         print 'CALCSELECTION', calcselection
1583         self.calcSelection_Range = []
1584         self.calcSelection_Range_OpTemps = []
1585         self.monthlyOutdoorMean = []

```

```

1586     ##
1587     self.positions = []
1588     self.Hr = []
1589     self.Mn = []
1590     self.outdoorDB = []
1591     self.zoneDB = []
1592     self.zoneRH = []
1593     self.zoneActSch = []
1594     self.zoneCloSch = []
1595     self.zoneAirVSch = []
1596     self.months = []
1597     self.runningMean7day = []
1598     self.runningMean3day = []
1599     for position, item in enumerate(self.dates):
1600         if (item[0:2] >= self.StartMonth) and \
1601             (item[0:2] <= self.EndMonth) and \
1602             (item[3:5] >= self.StartDay) and \
1603             (item[3:5] <= self.EndDay) and \
1604             (item[7:9] >= self.StartHour) and \
1605             (item[7:9] <= self.EndHour) :
1606             self.Hr.append(int(item[7:9]))
1607             self.Mn.append(int(item[0:2]))
1608             self.positions.append(position)
1609             self.calcSelection_Point = calcselection[position,:,:,:]
1610             self.calcSelection_Range.append(self.calcSelection_Point)
1611             self.calcSelection_Point_OpTemps = \
1612                 calcselectionOpTemps[position,:,:,:]
1613             self.calcSelection_Range_OpTemps.append \
1614                 (self.calcSelection_Point_OpTemps)
1615             monthlyOutdoorMean = self.concatReplacedODBforMean[position]
1616             self.monthlyOutdoorMean.append(monthlyOutdoorMean)
1617             self.runningMean7day.append(self.Trm7s[position])

```

```

1618         self.runningMean3day.append(self.Trm3s[position])
1619         # pull parameters
1620         self.oDB = self.numbers[position,self.loc_oDB]
1621         self.outdoorDB.append(self.oDB)
1622         self.zDB = self.numbers[position,self.loc_zDB]
1623         self.zoneDB.append(self.zDB)
1624         self.zRH = self.numbers[position,self.loc_zRH]
1625         self.zoneRH.append(self.zRH)
1626         self.zActSch = self.numbers[position,self.loc_zActSch]
1627         self.zoneActSch.append(self.zActSch)
1628         self.zCloSch = self.numbers[position,self.loc_zCloSch]
1629         self.zoneCloSch.append(self.zCloSch)
1630         self.zAirVSch = self.numbers[position,self.loc_zAirVSch]
1631         self.zoneAirVSch.append(self.zAirVSch)
1632         self.months.append(item[0:2])
1633     print 'SetDate_Range done'
1634     # bin data for scatters
1635     self.binStart = (int(self.colorStart.GetValue()))
1636     self.binEnd = (int(self.colorEnd.GetValue()))
1637     MBins = array([0,self.binStart,self.binEnd,13])
1638     HBins = array([0,self.binStart,self.binEnd,25])
1639     self.MonthBins = digitize(self.Mn, MBins)
1640     self.HourBins = digitize(self.Hr, HBins)
1641
1642     def scatterResults(self):
1643         '''Bin scatter results into user-specified periods
1644         '''
1645         self.scatterCalcResults = []
1646         # for months
1647         self.scatterCalcResultsMonthBin1 = []
1648         self.MonthBin1 = []
1649         self.scatterCalcResultsMonthBin2 = []

```

```

1650     self.MonthBin2 = []
1651     self.scatterCalcResultsMonthBin3 = []
1652     self.MonthBin3 = []
1653     # for hours
1654     self.scatterCalcResultsHourBin1 = []
1655     self.HourBin1 = []
1656     self.scatterCalcResultsHourBin2 = []
1657     self.HourBin2 = []
1658     self.scatterCalcResultsHourBin3 = []
1659     self.HourBin3 = []
1660     i = self.calcDrop.GetSelection()
1661     if i >= 0 and i <= 1:
1662         variable1 = self.calcSelection_Range
1663         variable2 = self.outdoorDB
1664     if i == 4:
1665         variable1 = self.calcSelection_Range_0pTemps
1666         variable2 = self.monthlyOutdoorMean
1667     if i == 5:
1668         variable1 = self.calcSelection_Range_0pTemps
1669         variable2 = self.runningMean7day
1670     if i == 6:
1671         variable1 = self.calcSelection_Range_0pTemps
1672         variable2 = self.runningMean3day
1673     for i in range(0,len(variable1)):
1674         results = mean(variable1[i])
1675         self.scatterCalcResults.append(results)
1676     # for months
1677     if self.MonthBins[i]==1:
1678         self.scatterCalcResultsMonthBin1.append(results)
1679         self.MonthBin1.append(variable2[i])
1680     if self.MonthBins[i]==2:
1681         self.scatterCalcResultsMonthBin2.append(results)

```

```

1682         self.MonthBin2.append(variable2[i])
1683     if self.MonthBins[i]==3:
1684         self.scatterCalcResultsMonthBin3.append(results)
1685         self.MonthBin3.append(variable2[i])
1686     # for hours
1687     if self.HourBins[i]==1:
1688         self.scatterCalcResultsHourBin1.append(results)
1689         self.HourBin1.append(variable2[i])
1690     if self.HourBins[i]==2:
1691         self.scatterCalcResultsHourBin2.append(results)
1692         self.HourBin2.append(variable2[i])
1693     if self.HourBins[i]==3:
1694         self.scatterCalcResultsHourBin3.append(results)
1695         self.HourBin3.append(variable2[i])
1696
1697 def indexMRTS_Range(self):
1698     # call indexing function
1699     self.setDate_Range()
1700     # define heatmap and scatter results
1701     self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
1702     self.scatterResults()
1703     # display settings
1704     self.format = '%2.1f'
1705     self.xlabel = 'outdoor dry bulb temperature [C]'
1706     self.ylabel = 'indoor mean radiant temperature [C]'
1707     self.scattertitle = "Mean Radiant Temperature vs Outdoor Dry Bulb"
1708     self.scatterXlim = [-10,35]
1709     self.scatterYlim = [16,32]
1710
1711 def indexOpTemp_Range(self):
1712     # call indexing function
1713     self.setDate_Range()

```



```

1714     # define heatmap and scatter results
1715     self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
1716     self.scatterResults()
1717     # display settings
1718     self.format = '%2.1f'
1719     self.format = '%2.1f'
1720     self.xlabel = 'outdoor dry bulb temperature [C]'
1721     self.ylabel = 'indoor operative temperature [C]'
1722     self.scattertitle = "Operative Temperature vs Outdoor Dry Bulb"
1723     self.scatterXlim = [-10,35]
1724     self.scatterYlim = [16,32]
1725
1726 def calcPMV_Range(self):
1727     '''provides values for PMV heatmap over a range of time
1728         note that this calculation can take a very long time
1729         because of the bisection search
1730     '''
1731     self.setDate_Range()
1732
1733     self.scatterCalcResultsPMV = \
1734         self.PMV_RoomAverageAnnual[self.positions[0]:self.positions[-1]]
1735     self.scatterCalcResultsPPD = \
1736         self.PPD_RoomAverageAnnual[self.positions[0]:self.positions[-1]]
1737
1738     self.collectPMVarrays = []
1739     self.collectPPDarrays = []
1740     for position, item in enumerate(self.positions):
1741         # set variables
1742         ta = self.zoneDB[position]
1743         RH = self.zoneRH[position]/100
1744         var = self.zoneAirVSch[position]
1745         met = self.zoneActSch[position]/58.15

```

```

1746     clo = self.zoneCloSch[position]
1747
1748     # auto calc variables
1749     Icl = clo*0.155
1750     M = met*58.15
1751     W = 0
1752     MW = M-W
1753     print 'set variables ok'
1754
1755     # steam table data - from 2.006 Property Data Tables
1756     steamTableTemps = [
1757         0.01, 5, 10, 15, 20, 25, 30, 35,
1758         40, 45, 50, 55, 60, 65, 70, 75,
1759         80, 85, 90, 95, 100
1760     ]
1761     steamTablePsat = [
1762         611.66, 872.58, 1228.2, 1705.8, 2339.3, 3169.9, 4247, 5629,
1763         7384.9, 9595, 12352, 15762, 19946, 25042, 31201, 38595,
1764         47414, 57867, 70182, 84608, 101420
1765     ]
1766     # calculate saturation pressure at ta
1767     Psat = interp(ta, steamTableTemps, steamTablePsat)
1768     # calculate water vapor partial pressure
1769     pa = RH*Psat
1770     print 'set steam tables ok'
1771
1772     # set fcl
1773     if Icl <= 0.078:
1774         fcl = 1.00 + 1.290*Icl
1775     else:
1776         fcl = 1.05 + 0.645*Icl
1777     print 'set fcl ok'

```

```

1778
1779     def findTcl(tcl):
1780         g = 2.38*abs(tcl-ta)**0.25
1781         h = 12.1*sqrt(var)
1782         hc = min(g,h)
1783         b = 35.7-0.028*MW-Icl*(3.96*(10**-8)*fcl*(((tcl+273)**4)- \
1784             ((i+273)**4))+fcl*hc*(tcl-ta))-tcl
1785         return b
1786
1787     # determine Tcl by bisection search (brentq method)
1788     collectTcl = []
1789     collectHC = []
1790     for i in ravel(self.calcSelection_Range[position]):
1791         Tcl = brentq(findTcl, 0, 100)
1792         collectTcl.append(Tcl)
1793         g = 2.38*abs(Tcl-ta)**0.25
1794         h = 12.1*sqrt(var)
1795         hc = max(g,h)
1796         collectHC.append(hc)
1797
1798     # heat loss components by parts -----
1799
1800     self.collectPMV = []
1801     self.collectPPD = []
1802     for i in range(0,len(ravel(self.calcSelection_Range[position]))):
1803         tr = (ravel(self.calcSelection_Range))[i]
1804         Tcl = collectTcl[i]
1805         hc = collectHC[i]
1806
1807         # heat loss diff through skin
1808         HL1 = 3.05*0.001*(5733-6.99*MW-pa)
1809         # heat loss by sweating

```

```

1810         if MW > 58.15:
1811             HL2 = 0.42*(MW-58.15)
1812         else:
1813             HL2 = 0
1814         # latent respiration heat loss
1815         HL3 = 1.7*0.00001*M*(5867-pa)
1816         # dry respiration heat loss
1817         HL4 = 0.0014*M*(34-ta)
1818         # heat loss by radiation
1819         HL5 = 3.96*0.00000001*fcl*(((Tcl+273)**4)-((tr+273)**4))
1820         # heat loss by convection
1821         HL6 = fcl*hc*(Tcl-ta)
1822         # thermal sensation trans coeff
1823         TS = 0.303*exp(-0.036*M)+0.028
1824         # calc PMV
1825         PMV = TS*(MW-HL1-HL2-HL3-HL4-HL5-HL6)
1826         self.collectPMV.append(PMV)
1827         PPD = 100-95*exp(-0.03353*PMV**4-0.2179*PMV**2)
1828         self.collectPPD.append(PPD)
1829         self.collectPMVarrays.append(self.collectPMV)
1830         self.collectPPDarrays.append(self.collectPPD)
1831
1832     reshapeCollectArrays = \
1833         reshape(self.collectPMVarrays,shape(self.calcSelection_Range))
1834     self.heatmapCalcResults = mean(reshapeCollectArrays, axis=0)
1835
1836     # display items
1837     r = around(self.heatmapCalcResults, decimals=2)
1838     u = unique(r)
1839     self.Tickvalues = u
1840     self.Tickvalues = [-3.0, -2.0, -1.0, -0.5, 0, 0.5, 1.0, 2.0, 3.0]
1841     self.Ticklabels = ['-3', '-2', '-1', '-0.5', '0', \

```

```

1842         '+0.5', '+1', '+2', '+3']
1843     self.Levels = len(u)
1844     self.format = '%2.2f'
1845     self.scattertitle = \
1846         "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
1847     self.xlabel = 'predicted mean vote (PMV)'
1848     self.ylabel = 'predicted percentage of dissatisfied (PPD)'
1849     self.scatterlim = [-3,3]
1850
1851     def calcPPD_Range(self):
1852         # provides values for PMV scatter plot for every hour of the year
1853         # PMV spatial calc done on-demand using 'Run cMap' button
1854         self.setDate_Range()
1855         self.calcPMV_Range()
1856
1857         self.scatterCalcResultsPMV = \
1858             self.PMV_RoomAverageAnnual[self.positions[0]:self.positions[-1]]
1859         self.scatterCalcResultsPPD = \
1860             self.PPD_RoomAverageAnnual[self.positions[0]:self.positions[-1]]
1861
1862         reshapeCollectArrays = \
1863             reshape(self.collectPPDarrays, shape(self.calcSelection_Range))
1864         self.heatmapCalcResults = mean(reshapeCollectArrays, axis=0)
1865
1866         # display items
1867         r = around(self.heatmapCalcResults, decimals=2)
1868         u = unique(r)
1869         self.Tickvalues = u
1870         self.Tickvalues = [0, 5, 10, 15, 20, 50]
1871         self.Ticklabels = ['0%', '5%', '10%', '15%', '20%', '>20%']
1872         self.Levels = len(u)
1873     #     self.cmap = self.cmap5Step

```

```

1874     self.format = '%2.2f'
1875     self.scattertitle = \
1876         "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
1877     self.xlabel = 'predicted mean vote (PMV)'
1878     self.ylabel = 'predicted percentage of dissatisfied (PPD)'
1879     self.scatterlim = [-3,3]
1880
1881     def indexASHRAE55Adaptive_Range(self):
1882         # call indexing function
1883         self.setDate_Range()
1884         # define heatmap and scatter results
1885         self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
1886         self.scatterResults()
1887         # display settings
1888         r = around(self.heatmapCalcResults, decimals=1)
1889         u = unique(r)
1890         self.Tickvalues = [-10, -3.5, -2.5, 2.5, 3.5, 10]
1891         self.Ticklabels = [
1892             '<80% acceptability',
1893             '80% acceptability \n  $\Delta T -3.5$  [C]',
1894             '90% acceptability \n  $\Delta T -2.5$  [C]',
1895             '90% acceptability \n  $\Delta T +2.5$  [C]',
1896             '80% acceptability \n  $\Delta T +3.5$  [C]',
1897             '< 80% acceptability']
1898         self.Levels = len(u)
1899         self.format = '%2.1f'
1900         # scatter plot lines
1901         self.scattertitle = \
1902             "ASHRAE Standard 55 Adaptive Comfort - Room Average"
1903         self.xlabel = 'mean monthly outdoor temperature [C]'
1904         self.ylabel = 'indoor operative temperature [C]'
1905         self.scatterXlim = [5,35]

```

```

1906     self.scatterYlim = [16,32]
1907     # scatter plot boundary lines
1908     self.outdoor = arange(10,33)
1909     self.comfortASHRAE = (0.31*self.outdoor+17.8)
1910     self.lower80ASHRAE = (0.31*self.outdoor+17.8)-3.5
1911     self.lower90ASHRAE = (0.31*self.outdoor+17.8)-2.5
1912     self.upper90ASHRAE = (0.31*self.outdoor+17.8)+2.5
1913     self.upper80ASHRAE = (0.31*self.outdoor+17.8)+3.5
1914
1915
1916     def indexEN15251Adaptive_Range(self):
1917         # call indexing function
1918         self.setDate_Range()
1919         # define heatmap and scatter results
1920         self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
1921         self.scatterResults()
1922         # display settings
1923         r = around(self.heatmapCalcResults, decimals=1)
1924         u = unique(r)
1925         self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
1926         self.Ticklabels = [
1927             '<80% acceptability',
1928             '80% acceptability \n  $\Delta T -3.0 [C]$ ',
1929             '90% acceptability \n  $\Delta T -2.0 [C]$ ',
1930             '90% acceptability \n  $\Delta T +2.0 [C]$ ',
1931             '80% acceptability \n  $\Delta T +3.0 [C]$ ',
1932             '< 80% acceptability']
1933         self.Levels = len(u)
1934         self.format = '%2.1f'
1935         # scatter plot lines
1936         self.scattertitle = \
1937             "European Standard EN 15251 Adaptive Comfort - Room Average"

```

```

1938     self.xlabel = '7-day running mean outdoor temperature [C]'
1939     self.ylabel = 'indoor operative temperature [C]'
1940     self.scatterXlim = [5,35]
1941     self.scatterYlim = [16,32]
1942     # scatter plot boundary lines
1943     self.outdoor = arange(10,33)
1944     self.comfortEN15251 = (0.33*self.outdoor+18.8)
1945     self.lower80EN15251 = (0.33*self.outdoor+18.8)-3.0
1946     self.lower90EN15251 = (0.33*self.outdoor+18.8)-2.0
1947     self.upper90EN15251 = (0.33*self.outdoor+18.8)+2.0
1948     self.upper80EN15251 = (0.33*self.outdoor+18.8)+3.0
1949
1950
1951     def indexNPRCR1752BetaAdaptive_Range(self):
1952         # call indexing function
1953         self.setDate_Range()
1954         # define heatmap and scatter results
1955         self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
1956         self.scatterResults()
1957         # display settings
1958         r = around(self.heatmapCalcResults, decimals=1)
1959         u = unique(r)
1960         self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
1961         self.Ticklabels = [
1962             '<80% acceptability',
1963             '80% acceptability \n  $\Delta T -3.0$  [C]',
1964             '90% acceptability \n  $\Delta T -2.0$  [C]',
1965             '90% acceptability \n  $\Delta T +2.0$  [C]',
1966             '80% acceptability \n  $\Delta T +3.0$  [C]',
1967             '< 80% acceptability']
1968         self.Levels = len(u)
1969         self.format = '%2.1f'

```



```

1970     # scatter plot lines
1971     self.scattertitle = \
1972         "Dutch Standard NPR-CR 1752 Type Beta Adaptive Comfort"
1973     self.xlabel = '3-day running mean outdoor temperature [C]'
1974     self.ylabel = 'indoor operative temperature [C]'
1975     self.scatterXlim = [5,35]
1976     self.scatterYlim = [16,32]
1977     # scatter plot boundary lines
1978     self.outdoor = arange(10,33)
1979     self.comfortNPRCR1752 = (0.31*self.outdoor+17.8)
1980     self.lower80NPRCR1752 = (0.31*self.outdoor+17.8)-3.0
1981     self.lower90NPRCR1752 = (0.31*self.outdoor+17.8)-2.0
1982     self.upper90NPRCR1752 = (0.31*self.outdoor+17.8)+2.0
1983     self.upper80NPRCR1752 = (0.31*self.outdoor+17.8)+3.0
1984
1985     # Control Functions -----
1986
1987     # run controls
1988     def loadFiles(self, event):
1989         self.idf = self.idfSel.GetValue()
1990         self.csv = self.csvSel.GetValue()
1991         # call calc functions
1992         self.readIDF()
1993         self.getTemps()
1994         self.setDate_Annual()
1995         self.viewFactors()
1996         # call annual metric calc functions
1997         self.calcMRTS()
1998         self.calcOpTemp()
1999         self.calcPMVPPDRoomAverageAnnual()
2000         self.calcAdaptiveASHRAE55()
2001         self.calcAdaptiveEN15251()

```

```

2002         self.calcAdaptiveNPRCR1752Beta()
2003
2004     def runCmap(self, event):
2005         self.GetCalcType()
2006     #         self.comfParamsSet()
2007         if self.noColor.GetValue() == True:
2008             self.makeScatter()
2009         if self.monthColor.GetValue() == True:
2010             self.makeScatterMonth()
2011         if self.timeColor.GetValue() == True:
2012             self.makeScatterHour()
2013         self.makePlot()
2014
2015     def runUpdate(self, event):
2016         self.GetCalcType()
2017     #         self.comfParamsSet()
2018         if self.noColor.GetValue() == True:
2019             self.makeScatter()
2020         if self.monthColor.GetValue() == True:
2021             self.makeScatterMonth()
2022         if self.timeColor.GetValue() == True:
2023             self.makeScatterHour()
2024         self.updatePlot()
2025
2026     def GetCalcType(self):
2027         i = self.calcDrop.GetSelection()
2028         j = self.timeframeDrop.GetSelection()
2029         # point in time calculation functions
2030         if i == 0 and j == 1:
2031             self.indexMRTS_Point()
2032         if i == 1 and j == 1:
2033             self.indexOpTemp_Point()

```

```

2034     if i == 2 and j == 1:
2035         self.calcPMV_Point()
2036     if i == 3 and j == 1:
2037         self.calcPPD_Point()
2038     if i == 4 and j == 1:
2039         self.indexASHRAE55Adaptive_Point()
2040     if i == 5 and j == 1:
2041         self.indexEN15251Adaptive_Point()
2042     if i == 6 and j == 1:
2043         self.indexNPRCR1752BetaAdaptive_Point()
2044     # range calculation functions
2045     if i == 0 and j == 0:
2046         self.indexMRTS_Range()
2047     if i == 1 and j == 0:
2048         self.indexOpTemp_Range()
2049     if i == 2 and j == 0:
2050         self.calcPMV_Range()
2051     if i == 3 and j == 0:
2052         self.calcPPD_Range()
2053     if i == 4 and j == 0:
2054         self.indexASHRAE55Adaptive_Range()
2055     if i == 5 and j == 0:
2056         self.indexEN15251Adaptive_Range()
2057     if i == 6 and j == 0:
2058         self.indexNPRCR1752BetaAdaptive_Range()
2059
2060     def comfParamsSet(self):
2061         # get values
2062         self.oDBSpin.SetLabel((str(around(self.oDB,decimals=1))). \
2063             strip('[').strip(']'))
2064         self.oRHSpin.SetLabel((str(around(self.oRH,decimals=0))). \
2065             strip('[').strip(']')).strip('.')

```

```

2066     self.oAirVSpin.SetLabel((str(around(self.oAirV,decimals=1))). \
2067         strip('[').strip(']'))
2068     self.zDBSpin.SetLabel((str(around(self.zDB,decimals=1))). \
2069         strip('[').strip(']'))
2070     self.zRHSpin.SetLabel((str(around(self.zRH,decimals=0))). \
2071         strip('[').strip(']')).strip('.'))
2072     self.zAirVSchSpin.SetLabel((str(around(self.zAirVSch,decimals=1))). \
2073         strip('[').strip(']'))
2074     self.zCloSchSpin.SetLabel((str(around(self.zCloSch,decimals=1))). \
2075         strip('[').strip(']'))
2076     self.zActSchSpin.SetLabel((str(around(self.zActSch/58.2,decimals=1))). \
2077         strip('[').strip(']'))
2078
2079     # scale controls
2080     def scaleSet(self):
2081         # get min & max to clip heatmaps
2082         i = self.calcDrop.GetSelection()
2083         if i >= 0 and i <= 1:
2084             r = around(self.heatmapCalcResults, decimals=1)
2085             u = unique(r)
2086             self.Tickvalues = around(linspace(min(u), max(u), num=10,
2087                 endpoint=True), decimals=1)
2088             self.Ticklabels = self.Tickvalues
2089             self.Levels = self.Tickvalues
2090             self.Ticks = self.Tickvalues
2091             self.Norm = mpl.colors.Normalize(vmin = min(self.Tickvalues),
2092                 vmax = max(self.Tickvalues), clip = False)
2093             self.cmapMaxDisplay.SetValue(str(max(self.Tickvalues)))
2094             self.cmapMinDisplay.SetValue(str(min(self.Tickvalues)))
2095         else:
2096             self.Ticks = self.Tickvalues
2097             self.Norm = mpl.colors.Normalize(vmin = min(self.Tickvalues),

```

```

2098         vmax = max(self.Tickvalues), clip = False)
2099
2100     def OnAdjustScale(self, event):
2101         clipMax = float(self.cmapMaxDisplay.GetValue())
2102         clipMin = float(self.cmapMinDisplay.GetValue())
2103         self.Tickvalues = around(linspace(clipMin, clipMax, num=10,
2104             endpoint=True), decimals=1)
2105         self.Ticklabels = self.Tickvalues
2106         self.Levels = self.Tickvalues
2107         self.Ticks = self.Tickvalues
2108         self.Norm = mpl.colors.Normalize(vmin = min(self.Tickvalues),
2109             vmax = max(self.Tickvalues), clip = False)
2110         self.updatePlot()
2111
2112     def OnAutoScale(self, event):
2113         self.scaleSet()
2114         self.updatePlot()
2115
2116     # cut controls
2117     def OnCheck(self, event):
2118         self.v = event.GetEventObject().GetLabel()
2119         cn = { 'X-Y' : '0', 'Y-Z' : '1', 'X-Z' : '2', }
2120         self.cutNum = int(cn[self.v])
2121         self.axH = self.v[0]
2122         self.axV = self.v[2]
2123         self.updatePlot()
2124
2125     # grid controls
2126     def OnGrid(self, event):
2127         self.gpt = self.gridSpin.GetValue()
2128         self.sliceSpin.SetRange(0,self.gpt-1)
2129         self.readIDF()

```

```

2130     self.getTemps()
2131     self.GetCalcType()
2132     self.updatePlot()
2133
2134     def DisableAnnual(self, event):
2135         a = self.StartMonthSpin.GetValue()
2136         b = self.StartDaySpin.GetValue()
2137         c = self.StartHourSpin.GetValue()
2138         z = self.StartHourSpin.GetValue()+1
2139         d = self.EndMonthSpin.GetValue()
2140         e = self.EndDaySpin.GetValue()
2141         f = self.EndHourSpin.GetValue()
2142         j = self.timeframeDrop.GetSelection()
2143         if j == 1:
2144             self.EndMonthSpin.Hide()
2145             self.EndDaySpin.Hide()
2146             self.EndHourSpin.Hide()
2147             self.enTxt.Hide()
2148             self.noColor.SetValue(True)
2149             self.monthColor.Disable()
2150             self.timeColor.Disable()
2151         else:
2152             self.EndMonthSpin.Show()
2153             self.EndDaySpin.Show()
2154             self.EndHourSpin.Show()
2155             self.enTxt.Show()
2156             self.monthColor.Enable()
2157             self.timeColor.Enable()
2158
2159     # slice controls
2160     def OnSlice(self, event):
2161         self.ind = self.sliceSpin.GetValue()

```

```

2162         self.updatePlot()
2163
2164     # scatter plot controls
2165     def makeScatter(self):
2166         print 'Making scatter plot...'
2167         self.scatterFig.clf()
2168         self.scatteraxes = self.scatterFig.add_subplot(1,1,1)
2169         self.scatterFig.subplots_adjust(top=0.875)
2170         self.scatterFig.subplots_adjust(bottom=0.2)
2171         self.scatterFig.subplots_adjust(left=0.1)
2172         self.scatterFig.subplots_adjust(right=0.9)
2173         self.scatteraxes.set_ylim(self.scatterYlim)
2174         self.scatteraxes.set_xlim(self.scatterXlim)
2175         i = self.calcDrop.GetSelection()
2176         j = self.timeframeDrop.GetSelection()
2177         if i >= 0 and i <= 1:
2178             self.scatteraxes.scatter(self.outdoorDB,self.scatterCalcResults,
2179                                     color=self.facecolorBlue,edgecolor=self.edgecolor,
2180                                     lw = self.lw,alpha=self.alpha,s=self.scattersize)
2181         if i >= 2 and i <= 3:
2182             self.curvePMVS = arange(-3,3.1,0.1)
2183             self.curvePPDS = \
2184                 100-95*exp(-0.03353*self.curvePMVS**4-0.2179*self.curvePMVS**2)
2185             self.scatteraxes.plot(self.curvePMVS,self.curvePPDS, 'k')
2186             self.scatteraxes.plot(self.scatterCalcResultsPMV,
2187                                   self.scatterCalcResultsPPD,'o',
2188                                   color=self.facecolorBlue,markersize=5)
2189         if i == 4:
2190             d = Line2D(self.outdoor,self.lower80ASHRAE, color='black')
2191             e = Line2D(self.outdoor,self.lower90ASHRAE, color='black',
2192                       linestyle = '--')
2193             f = Line2D(self.outdoor,self.comfortASHRAE, color='black',

```

```

2194         linestyle = ':'')
2195     g = Line2D(self.outdoor,self.upper90ASHRAE, color='black',
2196             linestyle = '--')
2197     h = Line2D(self.outdoor,self.upper80ASHRAE, color='black')
2198     self.scatteraxes.add_line(d)
2199     self.scatteraxes.add_line(e)
2200     self.scatteraxes.add_line(f)
2201     self.scatteraxes.add_line(g)
2202     self.scatteraxes.add_line(h)
2203     self.scatteraxes.scatter(self.monthlyOutdoorMean,
2204                             self.scatterCalcResults,color=self.facecolorBlue,
2205                             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2206                             s=self.scattersize)
2207     # make legend and labels
2208     prop = fm.FontProperties(size=8)
2209     legendlabels = ('80% Acceptability','90% Acceptability',
2210                   'Comfort Temperature')
2211     legendseries = (d,e,f)
2212     self.scatteraxes.legend(legendseries, legendlabels,
2213                             'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2214                             ncol=3, prop=prop).draw_frame(False)
2215     if i == 5:
2216         d = Line2D(self.outdoor,self.lower80EN15251, color='black')
2217         e = Line2D(self.outdoor,self.lower90EN15251, color='black',
2218             linestyle = '--')
2219         f = Line2D(self.outdoor,self.comfortEN15251, color='black',
2220             linestyle = ':'')
2221         g = Line2D(self.outdoor,self.upper90EN15251, color='black',
2222             linestyle = '--')
2223         h = Line2D(self.outdoor,self.upper80EN15251, color='black')
2224         self.scatteraxes.add_line(d)
2225         self.scatteraxes.add_line(e)

```



```

2226         self.scatteraxes.add_line(f)
2227         self.scatteraxes.add_line(g)
2228         self.scatteraxes.add_line(h)
2229         self.scatteraxes.scatter(self.runningMean7day,
2230             self.scatterCalcResults,color=self.facecolorBlue,
2231             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2232             s=self.scattersize)
2233         # make legend and labels
2234         prop = fm.FontProperties(size=8)
2235         legendlabels = ('80% Acceptability','90% Acceptability',
2236             'Comfort Temperature')
2237         legendseries = (d,e,f)
2238         self.scatteraxes.legend(legendseries, legendlabels,
2239             'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2240             ncol=3, prop=prop).draw_frame(False)
2241     if i == 6:
2242         d = Line2D(self.outdoor,self.lower80NPRCR1752, color='black')
2243         e = Line2D(self.outdoor,self.lower90NPRCR1752, color='black',
2244             linestyle = '--')
2245         f = Line2D(self.outdoor,self.comfortNPRCR1752, color='black',
2246             linestyle = ':')
2247         g = Line2D(self.outdoor,self.upper90NPRCR1752, color='black',
2248             linestyle = '--')
2249         h = Line2D(self.outdoor,self.upper80NPRCR1752, color='black')
2250         self.scatteraxes.add_line(d)
2251         self.scatteraxes.add_line(e)
2252         self.scatteraxes.add_line(f)
2253         self.scatteraxes.add_line(g)
2254         self.scatteraxes.add_line(h)
2255         self.scatteraxes.scatter(self.runningMean3day,
2256             self.scatterCalcResults,color=self.facecolorBlue,
2257             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,

```

```

2258         s=self.scattersize)
2259     # make legend and labels
2260     prop = fm.FontProperties(size=8)
2261     legendlabels = ('80% Acceptability','90% Acceptability',
2262         'Comfort Temperature')
2263     legendseries = (d,e,f)
2264     self.scatteraxes.legend(legendseries, legendlabels,
2265         'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2266         ncol=3, prop=prop).draw_frame(False)
2267     self.scatteraxes.grid(True,color=self.gridcolor)
2268     self.scatteraxes.spines['bottom'].set_color(self.axescolor)
2269     self.scatteraxes.spines['top'].set_color(self.axescolor)
2270     self.scatteraxes.spines['right'].set_color(self.axescolor)
2271     self.scatteraxes.spines['left'].set_color(self.axescolor)
2272     self.scatteraxes.set_title(self.scattertitle,fontsize=12)
2273     self.scatteraxes.set_xlabel(self.xlabel, fontsize=8)
2274     self.scatteraxes.set_ylabel(self.ylabel, fontsize=8)
2275     # update the font size of the x and y axes
2276     for tick in self.scatteraxes.xaxis.get_major_ticks():
2277         tick.label1.set_fontsize(8)
2278     for tick in self.scatteraxes.yaxis.get_major_ticks():
2279         tick.label1.set_fontsize(8)
2280     # send resize event to refresh panel
2281     pix = tuple( self.panel3.GetClientSize() )
2282     set = (pix[0]*1.01, pix[1]*1.01)
2283     self.scatterChart.SetClientSize( set )
2284     self.scatterCanvas.SetClientSize( set )
2285     print 'Finished making scatter plot'
2286
2287 # scatter plot controls
2288 def makeScatterMonth(self):
2289     print 'Making scatter plot...'

```

```

2290     self.scatterFig.clf()
2291     self.scatteraxes = self.scatterFig.add_subplot(1,1,1)
2292     self.scatterFig.subplots_adjust(top=0.875)
2293     self.scatterFig.subplots_adjust(bottom=0.2)
2294     self.scatterFig.subplots_adjust(left=0.1)
2295     self.scatterFig.subplots_adjust(right=0.9)
2296     self.scatteraxes.set_ylim(self.scatterYlim)
2297     self.scatteraxes.set_xlim(self.scatterXlim)
2298     i = self.calcDrop.GetSelection()
2299     j = self.timeframeDrop.GetSelection()
2300     if i >= 0 and i <= 1:
2301         a = self.scatteraxes.scatter(self.MonthBin1,
2302                                     self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
2303                                     edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2304                                     s=self.scattersize)
2305         b = self.scatteraxes.scatter(self.MonthBin2,
2306                                     self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
2307                                     edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2308                                     s=self.scattersize)
2309         c = self.scatteraxes.scatter(self.MonthBin3,
2310                                     self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
2311                                     edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2312                                     s=self.scattersize)
2313         # make legend and labels
2314         prop = fm.FontProperties(size=8)
2315         legendlabels = ('Simulated Hours','Highlighted Range')
2316         legendseries = (a,b)
2317         self.scatteraxes.legend(legendseries, legendlabels,
2318                                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2319                                ncol=2, prop=prop).draw_frame(False)
2320     # need PMV and PPD
2321     if i == 4:

```

```

2322     d = Line2D(self.outdoor,self.lower80ASHRAE, color='black')
2323     e = Line2D(self.outdoor,self.lower90ASHRAE, color='black',
2324               linestyle = '--')
2325     f = Line2D(self.outdoor,self.comfortASHRAE, color='black',
2326               linestyle = ':')
2327     g = Line2D(self.outdoor,self.upper90ASHRAE, color='black',
2328               linestyle = '--')
2329     h = Line2D(self.outdoor,self.upper80ASHRAE, color='black')
2330     self.scatteraxes.add_line(d)
2331     self.scatteraxes.add_line(e)
2332     self.scatteraxes.add_line(f)
2333     self.scatteraxes.add_line(g)
2334     self.scatteraxes.add_line(h)
2335     a = self.scatteraxes.scatter(self.MonthBin1,
2336                                 self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
2337                                 edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2338                                 s=self.scattersize)
2339     b = self.scatteraxes.scatter(self.MonthBin2,
2340                                 self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
2341                                 edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2342                                 s=self.scattersize)
2343     c = self.scatteraxes.scatter(self.MonthBin3,
2344                                 self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
2345                                 edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2346                                 s=self.scattersize)
2347     # make legend and labels
2348     prop = fm.FontProperties(size=8)
2349     legendlabels = ('80% Acceptability','90% Acceptability',
2350                   'Comfort Temperature','Simulated Hours','Highlighted Range')
2351     legendseries = (d,e,f,a,b)
2352     self.scatteraxes.legend(legendseries, legendlabels,
2353                             'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),

```

```

2354         ncol=5, prop=prop).draw_frame(False)
2355     if i == 5:
2356         d = Line2D(self.outdoor,self.lower80EN15251, color='black')
2357         e = Line2D(self.outdoor,self.lower90EN15251, color='black',
2358             linestyle = '--')
2359         f = Line2D(self.outdoor,self.comfortEN15251, color='black',
2360             linestyle = ':')
2361         g = Line2D(self.outdoor,self.upper90EN15251, color='black',
2362             linestyle = '--')
2363         h = Line2D(self.outdoor,self.upper80EN15251, color='black')
2364         self.scatteraxes.add_line(d)
2365         self.scatteraxes.add_line(e)
2366         self.scatteraxes.add_line(f)
2367         self.scatteraxes.add_line(g)
2368         self.scatteraxes.add_line(h)
2369         a = self.scatteraxes.scatter(self.MonthBin1,
2370             self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
2371             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2372             s=self.scattersize)
2373         b = self.scatteraxes.scatter(self.MonthBin2,
2374             self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
2375             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2376             s=self.scattersize)
2377         c = self.scatteraxes.scatter(self.MonthBin3,
2378             self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
2379             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2380             s=self.scattersize)
2381         # make legend and labels
2382         prop = fm.FontProperties(size=8)
2383         legendlabels = ('80% Acceptability','90% Acceptability',
2384             'Comfort Temperature','Simulated Hours','Highlighted Range')
2385         legendseries = (d,e,f,a,b)

```

```

2386         self.scatteraxes.legend(legendseries, legendlabels,
2387             'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2388             ncol=5, prop=prop).draw_frame(False)
2389     if i == 6:
2390         d = Line2D(self.outdoor,self.lower80NPRCR1752, color='black')
2391         e = Line2D(self.outdoor,self.lower90NPRCR1752, color='black',
2392             linestyle = '--')
2393         f = Line2D(self.outdoor,self.comfortNPRCR1752, color='black',
2394             linestyle = ':')
2395         g = Line2D(self.outdoor,self.upper90NPRCR1752, color='black',
2396             linestyle = '--')
2397         h = Line2D(self.outdoor,self.upper80NPRCR1752, color='black')
2398         self.scatteraxes.add_line(d)
2399         self.scatteraxes.add_line(e)
2400         self.scatteraxes.add_line(f)
2401         self.scatteraxes.add_line(g)
2402         self.scatteraxes.add_line(h)
2403         a = self.scatteraxes.scatter(self.MonthBin1,
2404             self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
2405             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2406             s=self.scattersize)
2407         b = self.scatteraxes.scatter(self.MonthBin2,
2408             self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
2409             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2410             s=self.scattersize)
2411         c = self.scatteraxes.scatter(self.MonthBin3,
2412             self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
2413             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2414             s=self.scattersize)
2415         # make legend and labels
2416         prop = fm.FontProperties(size=8)
2417         legendlabels = ('80% Acceptability','90% Acceptability',

```

```

2418         'Comfort Temperature', 'Simulated Hours', 'Highlighted Range')
2419     legendseries = (d,e,f,a,b)
2420     self.scatteraxes.legend(legendseries, legendlabels,
2421         'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2422         ncol=5, prop=prop).draw_frame(False)
2423     self.scatteraxes.set_title(self.scattertitle, fontsize=12)
2424     self.scatteraxes.set_xlabel(self.xlabel, fontsize=8)
2425     self.scatteraxes.set_ylabel(self.ylabel, fontsize=8)
2426     self.scatteraxes.grid(True, color=self.gridcolor)
2427     self.scatteraxes.spines['bottom'].set_color(self.axescolor)
2428     self.scatteraxes.spines['top'].set_color(self.axescolor)
2429     self.scatteraxes.spines['right'].set_color(self.axescolor)
2430     self.scatteraxes.spines['left'].set_color(self.axescolor)
2431     # update the font size of the x and y axes
2432     for tick in self.scatteraxes.xaxis.get_major_ticks():
2433         tick.label1.set_fontsize(8)
2434     for tick in self.scatteraxes.yaxis.get_major_ticks():
2435         tick.label1.set_fontsize(8)
2436     # send resize event to refresh panel
2437     pix = tuple( self.panel3.GetClientSize() )
2438     print "PIX", pix
2439     set = (pix[0]*1.01, pix[1]*1.01)
2440     self.scatterChart.SetClientSize( set )
2441     self.scatterCanvas.SetClientSize( set )
2442     print 'Finished making scatter plot'
2443
2444     # scatter plot controls
2445     def makeScatterHour(self):
2446         print 'Making scatter plot...'
2447         self.scatterFig.clf()
2448         self.scatteraxes = self.scatterFig.add_subplot(1,1,1)
2449         self.scatterFig.subplots_adjust(top=0.875)

```

```

2450     self.scatterFig.subplots_adjust(bottom=0.2)
2451     self.scatterFig.subplots_adjust(left=0.1)
2452     self.scatterFig.subplots_adjust(right=0.9)
2453     self.scatteraxes.set_ylim(self.scatterYlim)
2454     self.scatteraxes.set_xlim(self.scatterXlim)
2455     i = self.calcDrop.GetSelection()
2456     j = self.timeframeDrop.GetSelection()
2457     if i >= 0 and i <= 1:
2458         a = self.scatteraxes.scatter(self.HourBin1,
2459                                     self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
2460                                     edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2461                                     s=self.scattersize)
2462         b = self.scatteraxes.scatter(self.HourBin2,
2463                                     self.scatterCalcResultsHourBin2,color=self.facecolorRed,
2464                                     edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2465                                     s=self.scattersize)
2466         c = self.scatteraxes.scatter(self.HourBin3,
2467                                     self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
2468                                     edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2469                                     s=self.scattersize)
2470         # make legend and labels
2471         prop = fm.FontProperties(size=8)
2472         legendlabels = ('Simulated Hours','Highlighted Range')
2473         legendseries = (a,b)
2474         self.scatteraxes.legend(legendseries, legendlabels,
2475                                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2476                                ncol=2, prop=prop).draw_frame(False)
2477     # need PMV and PPD
2478     if i == 4:
2479         d = Line2D(self.outdoor,self.lower80ASHRAE, color='black')
2480         e = Line2D(self.outdoor,self.lower90ASHRAE, color='black',
2481                   linestyle = '--')

```



```

2482     f = Line2D(self.outdoor,self.comfortASHRAE, color='black',
2483               linestyle = ':')
2484     g = Line2D(self.outdoor,self.upper90ASHRAE, color='black',
2485               linestyle = '--')
2486     h = Line2D(self.outdoor,self.upper80ASHRAE, color='black')
2487     self.scatteraxes.add_line(d)
2488     self.scatteraxes.add_line(e)
2489     self.scatteraxes.add_line(f)
2490     self.scatteraxes.add_line(g)
2491     self.scatteraxes.add_line(h)
2492     a = self.scatteraxes.scatter(self.HourBin1,
2493                                 self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
2494                                 edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2495                                 s=self.scattersize)
2496     b = self.scatteraxes.scatter(self.HourBin2,
2497                                 self.scatterCalcResultsHourBin2,color=self.facecolorRed,
2498                                 edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2499                                 s=self.scattersize)
2500     c = self.scatteraxes.scatter(self.HourBin3,
2501                                 self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
2502                                 edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2503                                 s=self.scattersize)
2504     # make legend and labels
2505     prop = fm.FontProperties(size=8)
2506     legendlabels = ('80% Acceptability','90% Acceptability',
2507                   'Comfort Temperature','Simulated Hours','Highlighted Range')
2508     legendseries = (d,e,f,a,b)
2509     self.scatteraxes.legend(legendseries, legendlabels,
2510                             'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2511                             ncol=5, prop=prop).draw_frame(False)
2512 if i == 5:
2513     d = Line2D(self.outdoor,self.lower80EN15251, color='black')

```

```

2514     e = Line2D(self.outdoor,self.lower90EN15251, color='black',
2515               linestyle = '--')
2516     f = Line2D(self.outdoor,self.comfortEN15251, color='black',
2517               linestyle = ':')
2518     g = Line2D(self.outdoor,self.upper90EN15251, color='black',
2519               linestyle = '--')
2520     h = Line2D(self.outdoor,self.upper80EN15251, color='black')
2521     self.scatteraxes.add_line(d)
2522     self.scatteraxes.add_line(e)
2523     self.scatteraxes.add_line(f)
2524     self.scatteraxes.add_line(g)
2525     self.scatteraxes.add_line(h)
2526     a = self.scatteraxes.scatter(self.HourBin1,
2527                                  self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
2528                                  edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2529                                  s=self.scattersize)
2530     b = self.scatteraxes.scatter(self.HourBin2,
2531                                  self.scatterCalcResultsHourBin2,color=self.facecolorRed,
2532                                  edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2533                                  s=self.scattersize)
2534     c = self.scatteraxes.scatter(self.HourBin3,
2535                                  self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
2536                                  edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2537                                  s=self.scattersize)
2538     # make legend and labels
2539     prop = fm.FontProperties(size=8)
2540     legendlabels = ('80% Acceptability','90% Acceptability',
2541                    'Comfort Temperature','Simulated Hours','Highlighted Range')
2542     legendseries = (d,e,f,a,b)
2543     self.scatteraxes.legend(legendseries, legendlabels,
2544                              'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2545                              ncol=5, prop=prop).draw_frame(False)

```

```

2546     if i == 6:
2547         d = Line2D(self.outdoor,self.lower80NPRCR1752, color='black')
2548         e = Line2D(self.outdoor,self.lower90NPRCR1752, color='black',
2549             linestyle = '--')
2550         f = Line2D(self.outdoor,self.comfortNPRCR1752, color='black',
2551             linestyle = ':')
2552         g = Line2D(self.outdoor,self.upper90NPRCR1752, color='black',
2553             linestyle = '--')
2554         h = Line2D(self.outdoor,self.upper80NPRCR1752, color='black')
2555         self.scatteraxes.add_line(d)
2556         self.scatteraxes.add_line(e)
2557         self.scatteraxes.add_line(f)
2558         self.scatteraxes.add_line(g)
2559         self.scatteraxes.add_line(h)
2560         a = self.scatteraxes.scatter(self.HourBin1,
2561             self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
2562             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2563             s=self.scattersize)
2564         b = self.scatteraxes.scatter(self.HourBin2,
2565             self.scatterCalcResultsHourBin2,color=self.facecolorRed,
2566             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2567             s=self.scattersize)
2568         c = self.scatteraxes.scatter(self.HourBin3,
2569             self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
2570             edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
2571             s=self.scattersize)
2572         # make legend and labels
2573         prop = fm.FontProperties(size=8)
2574         legendlabels = ('80% Acceptability','90% Acceptability',
2575             'Comfort Temperature','Simulated Hours','Highlighted Range')
2576         legendseries = (d,e,f,a,b)
2577         self.scatteraxes.legend(legendseries, legendlabels,

```

```

2578         'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
2579         ncol=5, prop=prop).draw_frame(False)
2580     self.scatteraxes.set_title(self.scattertitle, fontsize=12)
2581     self.scatteraxes.set_xlabel(self.xlabel, fontsize=8)
2582     self.scatteraxes.set_ylabel(self.ylabel, fontsize=8)
2583     self.scatteraxes.grid(True, color=self.gridcolor)
2584     self.scatteraxes.spines['bottom'].set_color(self.axescolor)
2585     self.scatteraxes.spines['top'].set_color(self.axescolor)
2586     self.scatteraxes.spines['right'].set_color(self.axescolor)
2587     self.scatteraxes.spines['left'].set_color(self.axescolor)
2588     # update the font size of the x and y axes
2589     for tick in self.scatteraxes.xaxis.get_major_ticks():
2590         tick.label1.set_fontsize(8)
2591     for tick in self.scatteraxes.yaxis.get_major_ticks():
2592         tick.label1.set_fontsize(8)
2593     # send resize event to refresh panel
2594     pix = tuple( self.panel3.GetClientSize() )
2595     print "PIX", pix
2596     set = (pix[0]*1.01, pix[1]*1.01)
2597     self.scatterChart.SetClientSize( set )
2598     self.scatterCanvas.SetClientSize( set )
2599     print 'Finished making scatter plot'
2600
2601 # plot controls
2602 def makePlot(self):
2603     print 'Making heatmap plot...'
2604     self.scaleSet()
2605     self.fig.clf()
2606     self.axes = self.fig.add_subplot(1,1,1)
2607     self.fig.subplots_adjust(top=0.95)
2608     self.fig.subplots_adjust(bottom=0.225)
2609     self.fig.subplots_adjust(left=0.1)

```

```

2610 self.graphCut1 = [self.pt[0,:,:,0], self.pt[1,0,:,:), self.pt[0,:,0,:]]
2611 self.graphCut2 = [self.pt[1,:,:,0], self.pt[2,0,:,:), self.pt[2,:,0,:]]
2612 self.graphCut3 = [
2613     self.heatmapCalcResults[:, :, self.ind],
2614     self.heatmapCalcResults[self.ind, :, :],
2615     self.heatmapCalcResults[:, self.ind, :]]
2616 self.t = self.axes.contourf \
2617     (self.graphCut1[self.cutNum],
2618     self.graphCut2[self.cutNum],
2619     self.graphCut3[self.cutNum],
2620     15, alpha=1, cmap=cm.get_cmap(self.cmap), norm = self.Norm)
2621 self.s = self.axes.contour \
2622     (self.graphCut1[self.cutNum],
2623     self.graphCut2[self.cutNum],
2624     self.graphCut3[self.cutNum],
2625     15, linewidths=0.25, colors='k', norm = self.Norm)
2626 pyplot.xlabel(self.s, fmt = self.format, colors = '0.15', fontsize=9)
2627 self.axes.set_xlabel('%s distance [m]'%(self.axH), fontsize=8)
2628 self.axes.set_ylabel('%s distance [m]'%(self.axV), fontsize=8)
2629 # update the font size of the x and y axes
2630 for tick in self.axes.xaxis.get_major_ticks():
2631     tick.label1.set_fontsize(8)
2632 for tick in self.axes.yaxis.get_major_ticks():
2633     tick.label1.set_fontsize(8)
2634 # left,bottom,width,height
2635 cax = self.fig.add_axes([0.1, 0.1, 0.8, 0.02])
2636 cb = mpl.colorbar.ColorbarBase \
2637     (cax, cmap=cm.get_cmap(self.cmap), norm=self.Norm,
2638     orientation = 'horizontal', boundaries = self.Ticks,
2639     format = '%2.1f')
2640 cb.set_label((self.calcList[self.calcDrop.GetCurrentSelection()]),
2641     fontsize=7)

```

```

2642     cb.ax.set_xticklabels(self.Ticklabels)
2643     for t in cb.ax.get_xticklabels():
2644         t.set_fontsize(7)
2645     # send resize event to refresh panel
2646     pix = tuple( self.panel2.GetClientSize() )
2647     set = (pix[0]*1.01, pix[1]*1.01)
2648     self.chart.SetClientSize( set )
2649     self.canvas.SetClientSize( set )
2650     self.makeKey()
2651     print 'Finished making heatmap plot'
2652
2653     def updatePlot(self):
2654         self.fig.clf()
2655         self.axes = self.fig.add_subplot(1,1,1)
2656         self.fig.subplots_adjust(top=0.95)
2657         self.fig.subplots_adjust(bottom=0.225)
2658         self.fig.subplots_adjust(left=0.1)
2659         self.graphCut1 = [self.pt[0,:,:0], self.pt[1,0,:,:], self.pt[0,:,0,:]]
2660         self.graphCut2 = [self.pt[1,:,:0], self.pt[2,0,:,:], self.pt[2,:,0,:]]
2661         self.graphCut3 = [
2662             self.heatmapCalcResults[:,:,self.ind],
2663             self.heatmapCalcResults[self.ind,:,:],
2664             self.heatmapCalcResults[:,self.ind,:]]
2665         self.t = self.axes.contourf \
2666             (self.graphCut1[self.cutNum],
2667              self.graphCut2[self.cutNum],
2668              self.graphCut3[self.cutNum],
2669              15, alpha=1, cmap=cm.get_cmap(self.cmap), norm = self.Norm)
2670         self.s = self.axes.contour \
2671             (self.graphCut1[self.cutNum],
2672              self.graphCut2[self.cutNum],
2673              self.graphCut3[self.cutNum],

```

```

2674         15, linewidths=0.25, colors='k', norm = self.Norm)
2675     pyplot.xlabel(self.s, fmt = self.format, colors = '0.15', fontsize=9)
2676     self.axes.set_xlabel('%s distance [m]'%(self.axH), fontsize=8)
2677     self.axes.set_ylabel('%s distance [m]'%(self.axV), fontsize=8)
2678     # update the font size of the x and y axes
2679     for tick in self.axes.xaxis.get_major_ticks():
2680         tick.label1.set_fontsize(8)
2681     for tick in self.axes.yaxis.get_major_ticks():
2682         tick.label1.set_fontsize(8)
2683     #left,bottom,width,height
2684     cax = self.fig.add_axes([0.1, 0.1, 0.8, 0.02])
2685     cb = mpl.colorbar.ColorbarBase \
2686         (cax, cmap=cm.get_cmap(self.cmap), norm=self.Norm,
2687          orientation = 'horizontal', boundaries = self.Ticks,
2688          format = '%2.1f')
2689     cb.set_label((self.calcList[self.calcDrop.GetCurrentSelection()]),
2690                fontsize=7)
2691     cb.ax.set_xticklabels(self.Ticklabels)
2692     for t in cb.ax.get_xticklabels():
2693         t.set_fontsize(7)
2694     # send resize event to refresh panel
2695     pix = tuple( self.panel2.GetClientSize() )
2696     set = (pix[0]*1.01, pix[1]*1.01)
2697     self.chart.SetClientSize( set )
2698     self.canvas.SetClientSize( set )
2699     self.makeKey()
2700     print 'updatePlot fine'
2701
2702     def makeKey(self):
2703         '''function for creating 3-D slice key'''
2704         # plot containers for slice key
2705         self.key = wx.Panel(self.panel4)

```

```

2706     self.keyfig = Figure(dpi=100, facecolor='none')
2707     self.keycanvas = FigureCanvas(self.key, -1, self.keyfig)
2708     self.keySizer.Add(self.key, 0, wx.ALL|wx.EXPAND, 5)
2709     self.keyaxes = axes3d.Axes3D(self.keyfig)
2710     self.keyaxes.disable_mouse_rotation()
2711     xL = [self.xmin,self.xmax]
2712     yL = [self.ymin,self.ymax]
2713     zL = [self.zmin,self.zmax]
2714     xJ = [self.winxmin,self.winxmax]
2715     yJ = [self.winymin,self.winyamax]
2716     zJ = [self.winzmin,self.winzmax]
2717     # set plot limits
2718     self.keyaxes.set_xlim((xL[0],xL[1]))
2719     self.keyaxes.set_ylim((yL[0],yL[1]))
2720     self.keyaxes.set_zlim((zL[0],zL[1]))
2721     # plot room facades
2722     xN = [xL[0],xL[0],xL[1],xL[1]]
2723     yN = [yL[1],yL[1],yL[1],yL[1]]
2724     zN = [zL[0],zL[1],zL[1],zL[0]]
2725     verts = [zip(xN,yN,zN)]
2726     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2727         facecolor = ('1'),edgecolors = ('k'), linewidths=(0.5),
2728         alpha=0.05))
2729     # east wall
2730     xE = [xL[1],xL[1],xL[1],xL[1]]
2731     yE = [yL[0],yL[0],yL[1],yL[1]]
2732     zE = [zL[0],zL[1],zL[1],zL[0]]
2733     verts = [zip(xE,yE,zE)]
2734     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2735         facecolor = ('1'),edgecolors = ('k'), linewidths=(0.5),
2736         alpha=0.05))
2737     # south wall

```



```

2738     xS = [xL[0],xL[0],xL[1],xL[1]]
2739     yS = [yL[0],yL[0],yL[0],yL[0]]
2740     zS = [zL[0],zL[1],zL[1],zL[0]]
2741     verts = [zip(xS,yS,zS)]
2742     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2743         facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
2744         alpha=0.05))
2745     # west wall
2746     xW = [xL[0],xL[0],xL[0],xL[0]]
2747     yW = [yL[0],yL[0],yL[1],yL[1]]
2748     zW = [zL[0],zL[1],zL[1],zL[0]]
2749     verts = [zip(xW,yW,zW)]
2750     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2751         facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
2752         alpha=0.05))
2753     # floor
2754     xF = [xL[0],xL[0],xL[1],xL[1]]
2755     yF = [yL[0],yL[1],yL[1],yL[0]]
2756     zF = [zL[0],zL[0],zL[0],zL[0]]
2757     verts = [zip(xF,yF,zF)]
2758     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2759         facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
2760         alpha=0.05))
2761     # ceiling
2762     xC = [xL[0],xL[0],xL[1],xL[1]]
2763     yC = [yL[0],yL[1],yL[1],yL[0]]
2764     zC = [zL[1],zL[1],zL[1],zL[1]]
2765     verts = [zip(xC,yC,zC)]
2766     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2767         facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
2768         alpha=0.05))
2769     # window

```

```

2770     xWin = [xJ[0],xJ[0],xJ[1],xJ[1]]
2771     yWin = [yJ[1],yJ[1],yJ[1],yJ[1]]
2772     zWin = [zJ[0],zJ[1],zJ[1],zJ[0]]
2773     verts = [zip(xWin,yWin,zWin)]
2774     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2775         facecolor = ('w'), edgecolors = ('b'), linewidths=(0.65),
2776         alpha=0.05))
2777     # cut slice
2778     ct = [self.pt[2,0,0,self.ind], self.pt[0,self.ind,0,0],
2779         self.pt[1,0,self.ind,0]]
2780     self.TT = ct[self.cutNum]
2781     cn = ['X-Y', 'Y-Z', 'X-Z']
2782     self.SS = cn[self.cutNum]
2783     if self.SS == 'X-Y':
2784         xCut = [xL[0],xL[1],xL[1],xL[0]]
2785         yCut = [yL[0],yL[0],yL[1],yL[1]]
2786         zCut = [self.TT,self.TT,self.TT,self.TT]
2787     if self.SS == 'Y-Z':
2788         xCut = [self.TT,self.TT,self.TT,self.TT]
2789         yCut = [yL[0],yL[0],yL[1],yL[1]]
2790         zCut = [zL[0],zL[1],zL[1],zL[0]]
2791     if self.SS == 'X-Z':
2792         xCut = [xL[0],xL[0],xL[1],xL[1]]
2793         yCut = [self.TT,self.TT,self.TT,self.TT]
2794         zCut = [zL[0],zL[1],zL[1],zL[0]]
2795     verts = [zip(xCut,yCut,zCut)]
2796     self.keyaxes.add_collection3d(Poly3DCollection(verts,
2797         facecolor = ('#BDBDBD'), edgecolors = ('#969696'), alpha=1))
2798     # format plot
2799     self.keyaxes.set_xticks([])
2800     self.keyaxes.set_yticks([])
2801     self.keyaxes.set_zticks([])

```

```

2802     self.keyaxes.set_xlabel('x', fontsize=8)
2803     self.keyaxes.set_ylabel('y', fontsize=8)
2804     self.keyaxes.set_zlabel('z', fontsize=8 )
2805     # link key containers to resize functions
2806     self._SetSizeKey()
2807     self.keycanvas.draw()
2808     self.key._resizeflag = False
2809     self.key.Bind(wx.EVT_IDLE, self._onIdleKey)
2810     self.key.Bind(wx.EVT_SIZE, self._onSizeKey)
2811
2812     def OnChoice(self, event):
2813         choice = event.GetString()
2814         print choice
2815
2816     def OnClose(self, event):
2817         # deinitialize the frame manager
2818         self._mgr.UnInit()
2819         # delete the frame
2820         self.Destroy()
2821
2822     def OnPrint(self, event):
2823         '''function for exporting images as .png files'''
2824         cwd = os.getcwd()
2825         #make destination folders
2826         dirs = [os.path.join(cwd, 'images')]
2827         for i in dirs:
2828             try:
2829                 os.makedirs(i)
2830             except OSError:
2831                 pass
2832         ct = [self.pt[2,0,0,self.ind], self.pt[0,self.ind,0,0],
2833             self.pt[1,0,self.ind,0]]

```

```

2834     self.TT = ct[self.cutNum]
2835     cn = ['XY', 'YZ', 'XZ']
2836     self.SS = cn[self.cutNum]
2837     i = str(self.calcDrop.GetSelection())
2838     cn = { '0':'MRT', '1':'OpTemp', '2':'PMV', '3':'PPD',
2839           '4':'AdaptASHRAE', '5':'AdaptEN15251', '6':'AdaptNPRCR1752', }
2840     calc = (cn[i])
2841     j = self.timeframeDrop.GetSelection()
2842     if j == 0:
2843         timeStart = '%s%s%s' % \
2844             (self.StartMonth,self.StartDay,self.StartHour)
2845         timeEnd = '-%s%s%s' % \
2846             (self.EndMonth,self.EndDay,self.EndHour)
2847     else:
2848         timeStart = '%s%s%s' % \
2849             (self.StartMonth,self.StartDay,self.StartHour)
2850         timeEnd = ''
2851     heatmapFname = cwd + '/images' + '/hmap%s_%s_%s%sm.png' % \
2852         (calc,timeStart,timeEnd,self.SS,self.TT)
2853     scatterFname = cwd + '/images' + '/sctr%s_%s%sm.png' % \
2854         (calc,timeStart,timeEnd)
2855     print 'Saving image', heatmapFname
2856     print 'Saving image', scatterFname
2857     self.fig.savefig(heatmapFname, dpi = 300, format='png')
2858     self.scatterFig.savefig(scatterFname, dpi = 300, format='png')
2859
2860     # Resize Plot Functions - heatmap plot -----
2861     # plots are updated using these resize events
2862
2863     def _onSize( self, event ):
2864         self.chart._resizeflag = True
2865

```

```

2866     def _onIdle( self, evt ):
2867         if self.chart._resizeflag:
2868             self.chart._resizeflag = False
2869             self._SetSize()
2870
2871     def _SetSize( self ):
2872         pixels = tuple( self.panel2.GetClientSize() )
2873         self.chart.SetSize( pixels )
2874         self.canvas.SetSize( pixels )
2875         self.fig.set_size_inches \
2876             ( float( pixels[0] )/self.fig.get_dpi(),
2877              float( pixels[1] )/self.fig.get_dpi() )
2878
2879     def draw(self): pass # abstract, to be overridden by child classes
2880
2881     # Resize Plot Functions - diagram key -----
2882
2883     def _onSizeKey( self, event ):
2884         self.key._resizeflag = True
2885
2886     def _onIdleKey( self, evt ):
2887         if self.key._resizeflag:
2888             self.key._resizeflag = False
2889             self._SetSizeKey()
2890
2891     def _SetSizeKey( self ):
2892         pixels = tuple( self.panel4.GetClientSize() )
2893         self.key.SetSize( [pixels[0]/1.1,pixels[1]/1.1] )
2894         self.keycanvas.SetSize( [pixels[0]/1.1,pixels[1]/1.1] )
2895         self.keyfig.set_size_inches \
2896             ( float( pixels[0]/1.1 )/self.keyfig.get_dpi(),
2897              float( pixels[1]/1.1 )/self.keyfig.get_dpi() )

```

```

2898
2899     def draw(self): pass # abstract, to be overridden by child classes
2900
2901     # Resize Plot Functions - scatter plot -----
2902
2903     def _onSizeScatter( self, event ):
2904         self.scatterChart._resizeflag = True
2905
2906     def _onIdleScatter( self, evt ):
2907         if self.scatterChart._resizeflag:
2908             self.scatterChart._resizeflag = False
2909             self._SetSizeScatter()
2910
2911     def _SetSizeScatter( self ):
2912         pixels = tuple( self.panel3.GetClientSize() )
2913         self.scatterChart.SetSize( pixels )
2914         self.scatterCanvas.SetSize( pixels )
2915         self.scatterFig.set_size_inches \
2916             ( float( pixels[0] )/self.scatterFig.get_dpi(),
2917              float( pixels[1] )/self.scatterFig.get_dpi() )
2918
2919     def draw(self): pass # abstract, to be overridden by child classes
2920
2921     # FileSelectorCombo class -----
2922     class FileSelectorComboIDF(wx.combo.ComboCtrl):
2923         '''class for control for selecting .idf file'''
2924         def __init__(self, *args, **kw):
2925             wx.combo.ComboCtrl.__init__(self, *args, **kw)
2926
2927             # make a custom bitmap showing "...
2928             bw, bh = 14, 16
2929             bmp = wx.EmptyBitmap(bw,bh)

```

```

2930     dc = wx.MemoryDC bmp)
2931
2932     # clear to a specific background colour
2933     bgcolor = wx.Colour(255,254,255)
2934     dc.SetBackground(wx.Brush(bgcolor))
2935     dc.Clear()
2936
2937     # draw the label onto the bitmap
2938     label = "..."
2939     font = wx.SystemSettings.GetFont(wx.SYS_DEFAULT_GUI_FONT)
2940     font.SetWeight(wx.FONTWEIGHT_BOLD)
2941     dc.SetFont(font)
2942     tw,th = dc.GetTextExtent(label)
2943     dc.DrawText(label, (bw-tw)/2, (bw-tw)/2)
2944     del dc
2945
2946     # now apply a mask using the bgcolor
2947     bmp.SetMaskColour(bgcolor)
2948
2949     # and tell the ComboCtrl to use it
2950     self.SetButtonBitmaps bmp, True)
2951
2952     # Overridden from ComboCtrl, called when the combo button is clicked
2953     def OnButtonClick(self):
2954         path = ""
2955         name = ""
2956         if self.GetValue():
2957             path, name = os.path.split(self.GetValue())
2958
2959         dlg = wx.FileDialog(self, "Choose File", path, name,
2960                             ".idf files (*.idf)|*.idf", wx.FD_OPEN)
2961         if dlg.ShowModal() == wx.ID_OK:

```

```

2962         self.SetValue(dlg.GetPath())
2963     dlg.Destroy()
2964     self.SetFocus()
2965
2966     # Overridden from ComboCtrl to avoid assert since there is no ComboPopup
2967     def DoSetPopupControl(self, popup):
2968         pass
2969
2970 class FileSelectorComboCSV(wx.combo.ComboCtrl):
2971     '''class for control for selecting .csv file'''
2972     def __init__(self, *args, **kw):
2973         wx.combo.ComboCtrl.__init__(self, *args, **kw)
2974
2975         # make a custom bitmap showing "..."
2976         bw, bh = 14, 16
2977         bmp = wx.EmptyBitmap(bw, bh)
2978         dc = wx.MemoryDC(bmp)
2979
2980         # clear to a specific background colour
2981         bgcolor = wx.Colour(255, 254, 255)
2982         dc.SetBackground(wx.Brush(bgcolor))
2983         dc.Clear()
2984
2985         # draw the label onto the bitmap
2986         label = "..."
2987         font = wx.SystemSettings.GetFont(wx.SYS_DEFAULT_GUI_FONT)
2988         font.SetWeight(wx.FONTWEIGHT_BOLD)
2989         dc.SetFont(font)
2990         tw, th = dc.GetTextExtent(label)
2991         dc.DrawText(label, (bw-tw)/2, (bh-th)/2)
2992         del dc
2993

```



```

2994         # now apply a mask using the bgcolor
2995         bmp.SetMaskColour(bgcolor)
2996
2997         # and tell the ComboCtrl to use it
2998         self.SetButtonBitmaps(bmp, True)
2999
3000
3001     # Overridden from ComboCtrl, called when the combo button is clicked
3002     def OnButtonClick(self):
3003         path = ""
3004         name = ""
3005         if self.GetValue():
3006             path, name = os.path.split(self.GetValue())
3007
3008             dlg = wx.FileDialog(self, "Choose File", path, name,
3009                               ".csv files (*.csv)|*.csv", wx.FD_OPEN)
3010             if dlg.ShowModal() == wx.ID_OK:
3011                 self.SetValue(dlg.GetPath())
3012             dlg.Destroy()
3013             self.SetFocus()
3014
3015     # Overridden from ComboCtrl to avoid assert since there is no ComboPopup
3016     def DoSetPopupControl(self, popup):
3017         pass
3018
3019 # End FileSelectorCombo class-----
3020
3021 app = wx.PySimpleApp()
3022 frame = MyFrame(None, -1, "cMap - Thermal Comfort Spatial Mapping")
3023 frame.Show()
3024 app.MainLoop()
3025

```

3026 # END INTERFACE -----