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

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 MITs 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

List of Figures

List of Tables

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, $\mathrm{W/m^2} \cdot K$

I_{cl} clothing insulation, $\mathrm{m^2} \cdot K/W$

M metabolic rate, $\mathrm{W/m^2}$

p_a water vapor partial pressure, Pa

T_{-n} average outdoor temperature n days befpre 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, $\mathrm{W/m^2}$

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 cant 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)

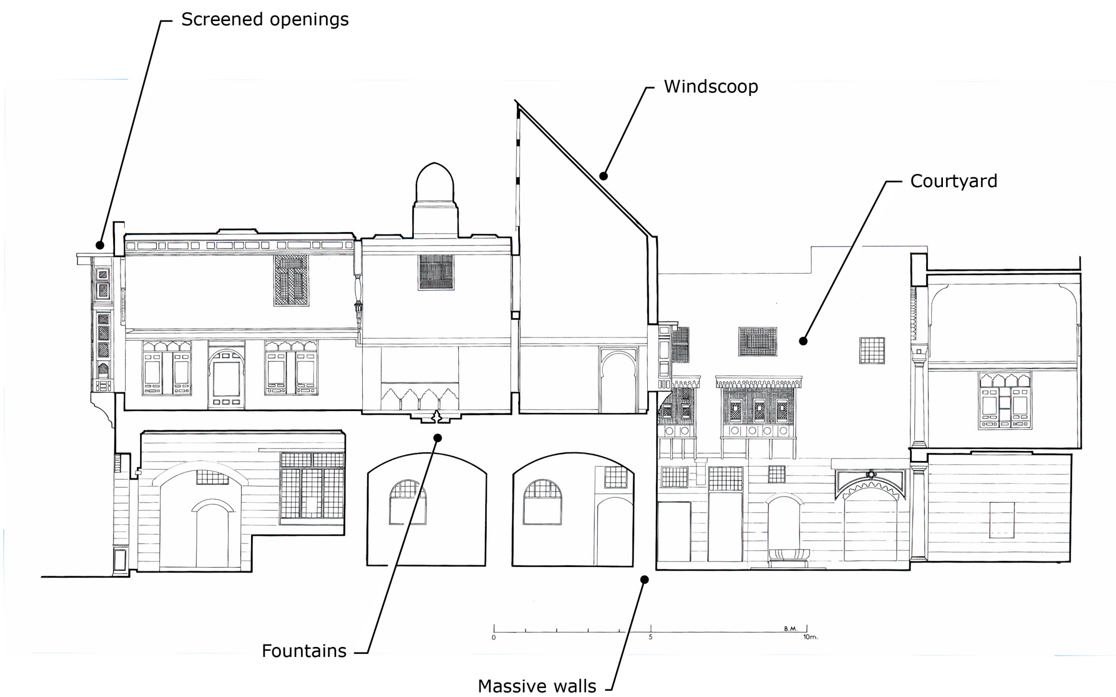
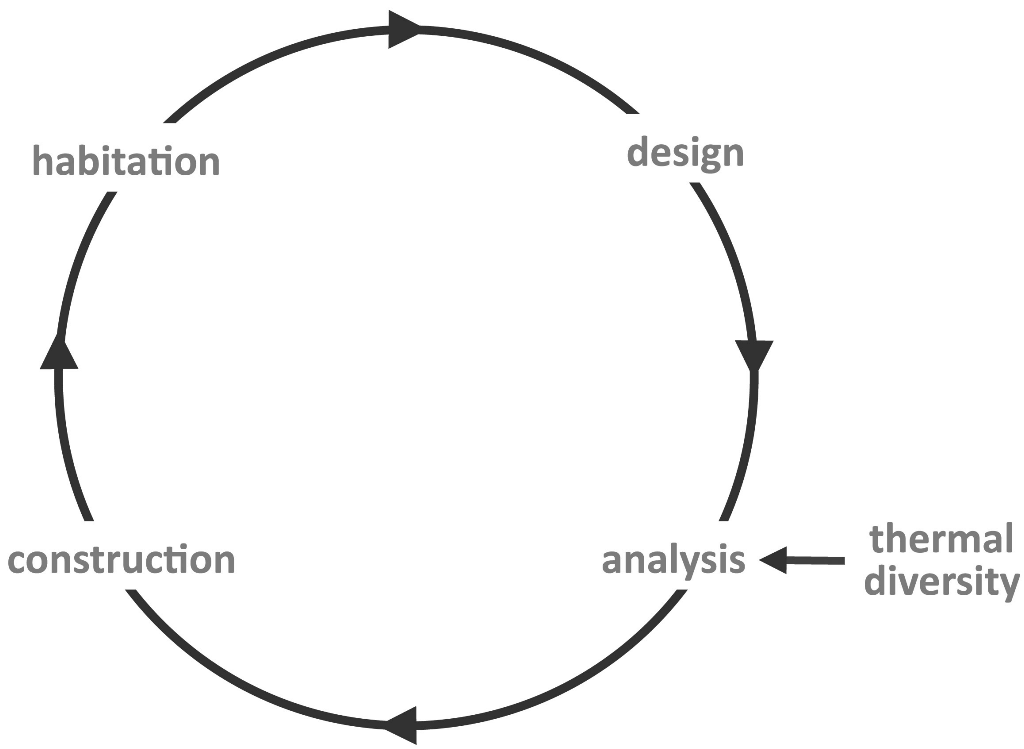


Figure 1-3: Photograph of Cairo from atop the Citadel, 1860. Photo by Frith (20). A close inspection shows the abundance of windscoops across the roofs of the city

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 wont 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?

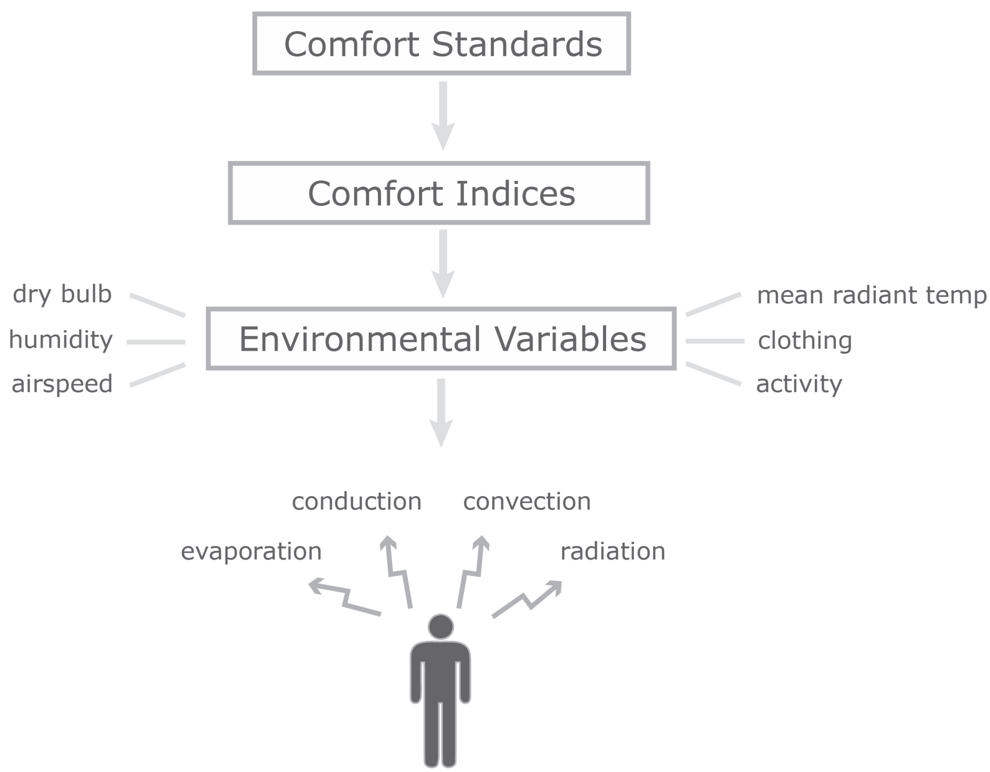
A short discussion on thermal comfort theory, existing thermal comfort analysis tools, and the use of these tools in professional practice follows.

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):

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

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 Dissatisifed, 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)

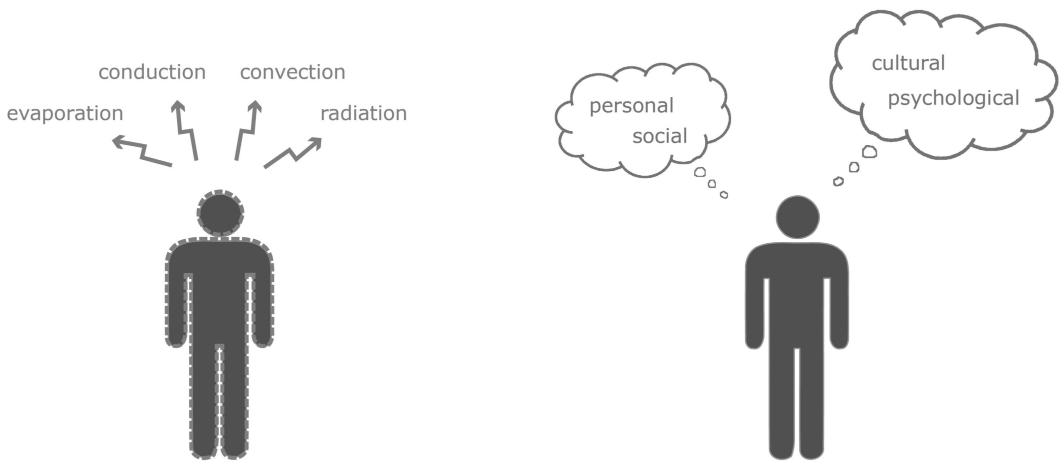
In the 1990s, the adaptive comfort model was developed in response to Fanger's work. In contrast to Fanger's highly controlled climate chamber tests, de Dear and Brager surveyed occupants in actual buildings about their comfort preferences and measured the environmental conditions at the time of survey.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

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

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 $+/-$ 2.5 °C and $+/-$ 3.5 °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 databased 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:

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

where T_{rm7} is calcuated as:

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

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

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

where T_{rm3} is calcuated as:

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

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.

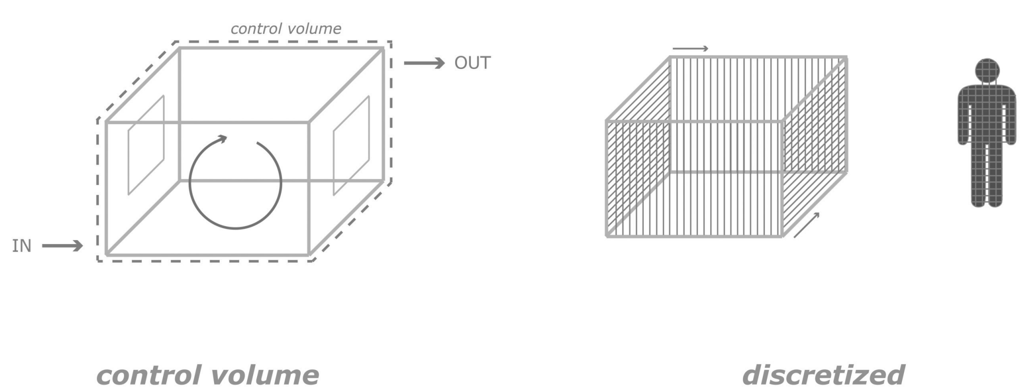


Table 2.2 summarizes these primary characteristics for several existing comfort analysis tools.

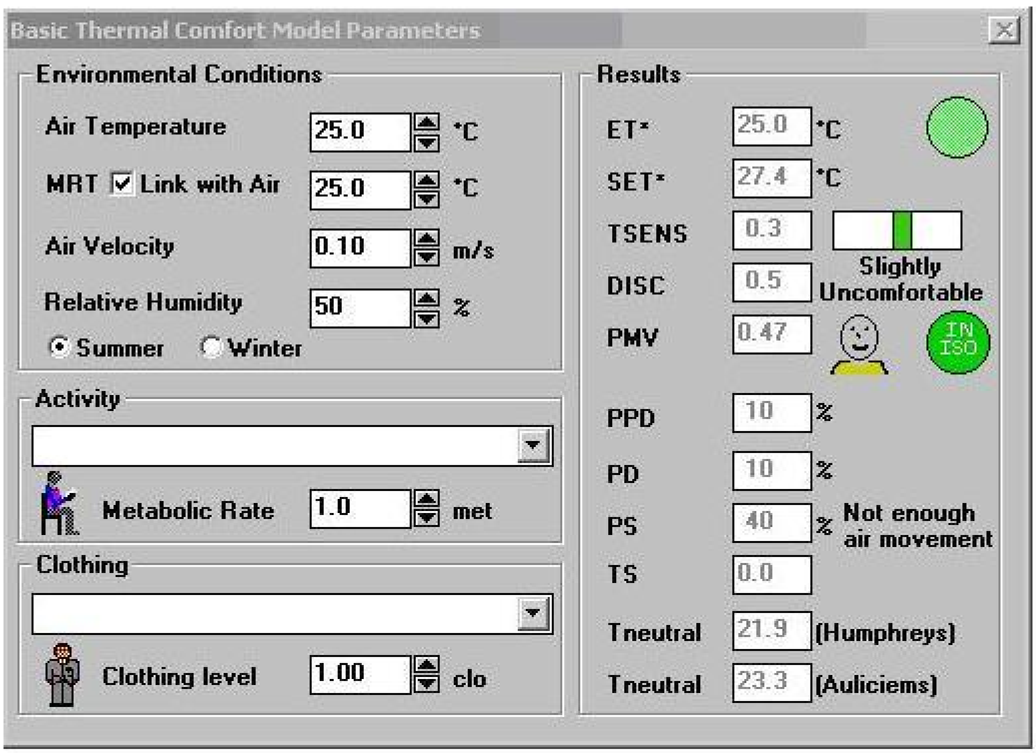
Table 2.2: Characteristics of existing thermal comfort software tools

Tool Name	Method	Scale	Spatial	Temporal
EnergyPlus	control 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

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.

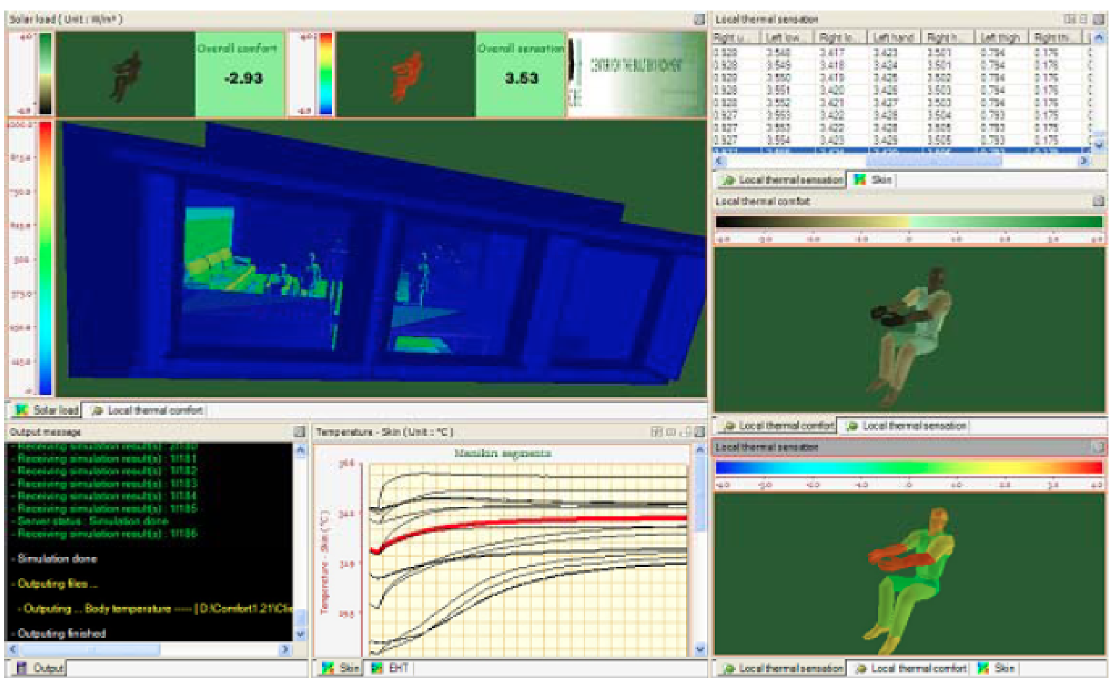
ASHRAE developed its own Thermal Comfort Tool software in the mid 1990s, as part of research project in RP-781 (18) (19). This tool is designed to help HVAC engineers determine whether their design is in compliance with ASHRAE Standard 55. The tool is available for purchase from ASHRAE. In contrast to 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.

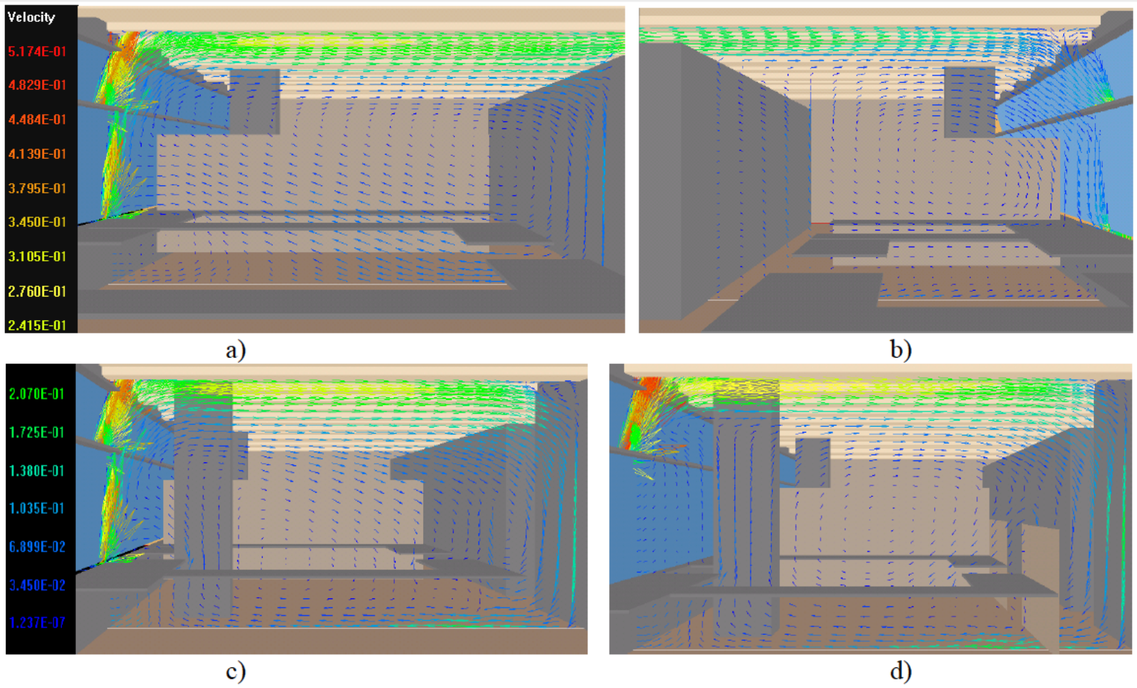
ROOM can produce these results for an average day for each month of the year, but it is not able to produce results for every hour of the year. For information on ROOM, please see White (50). The author also received a demonstration of the ROOM software from Jauni Novak, a Graduate Mechanical Engineer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

While the focus of this thesis in 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 predecent 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:

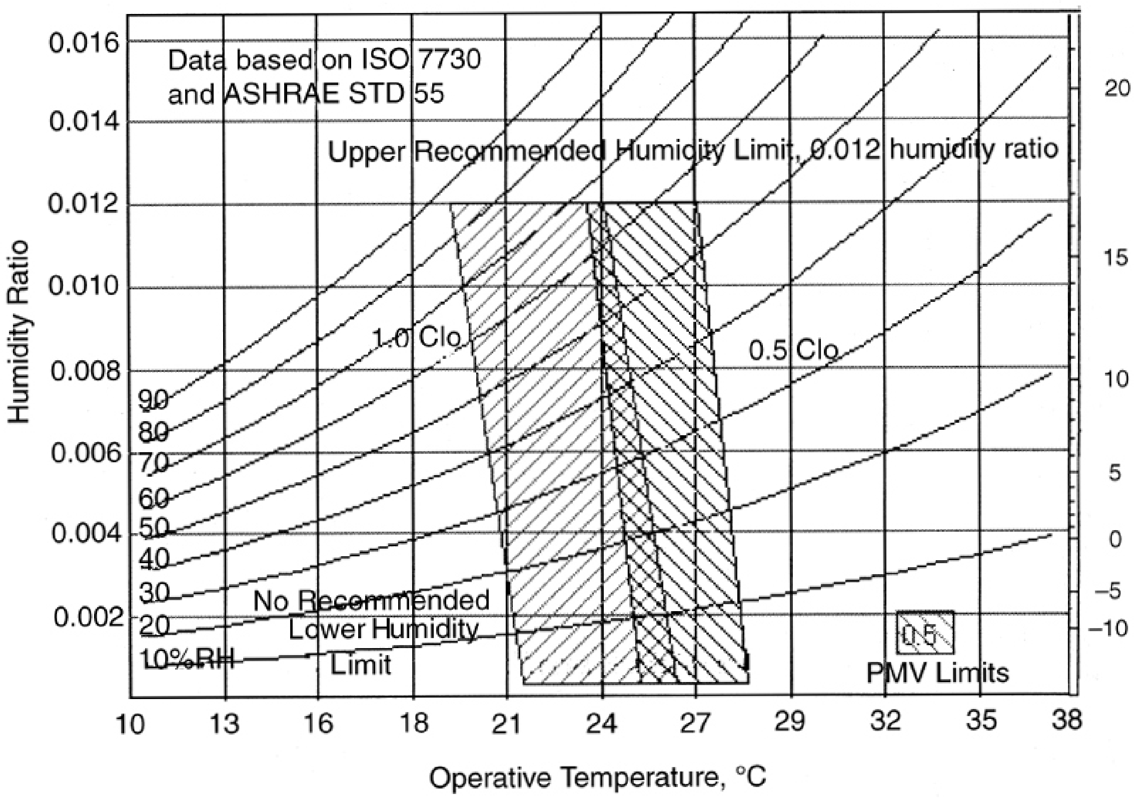
Too time-intensive Tools might be too time-intensive because their computational methods take a long time, e.g., CFD, because the inputs to the tool are non-trival to determine, e.g., angle factors in EnergyPlus, or because they are an entirely separate tool and do not fit within a company's existing 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 emplyed 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.

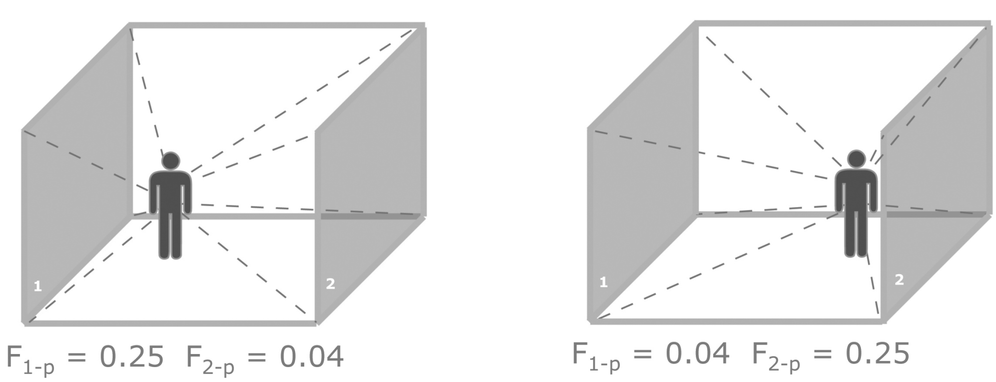
It is worth noting the general importance of mean radiant temperature in the literature. Fanger (15) devoted a significant portion of his work to the calculation of mean radiant temperature. Powitz (38) suggests that radiant temperature asymmetries are a chief cause of comfort complaints in actual buildings.

Mean radiant temperature is calculated using the following equation:

$$T_{mrt} = T_1 F_{P-1} + T_2 F_{P-2} + ... + T_n F_{P-n} \tag{3.1}$$

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 doesnt 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

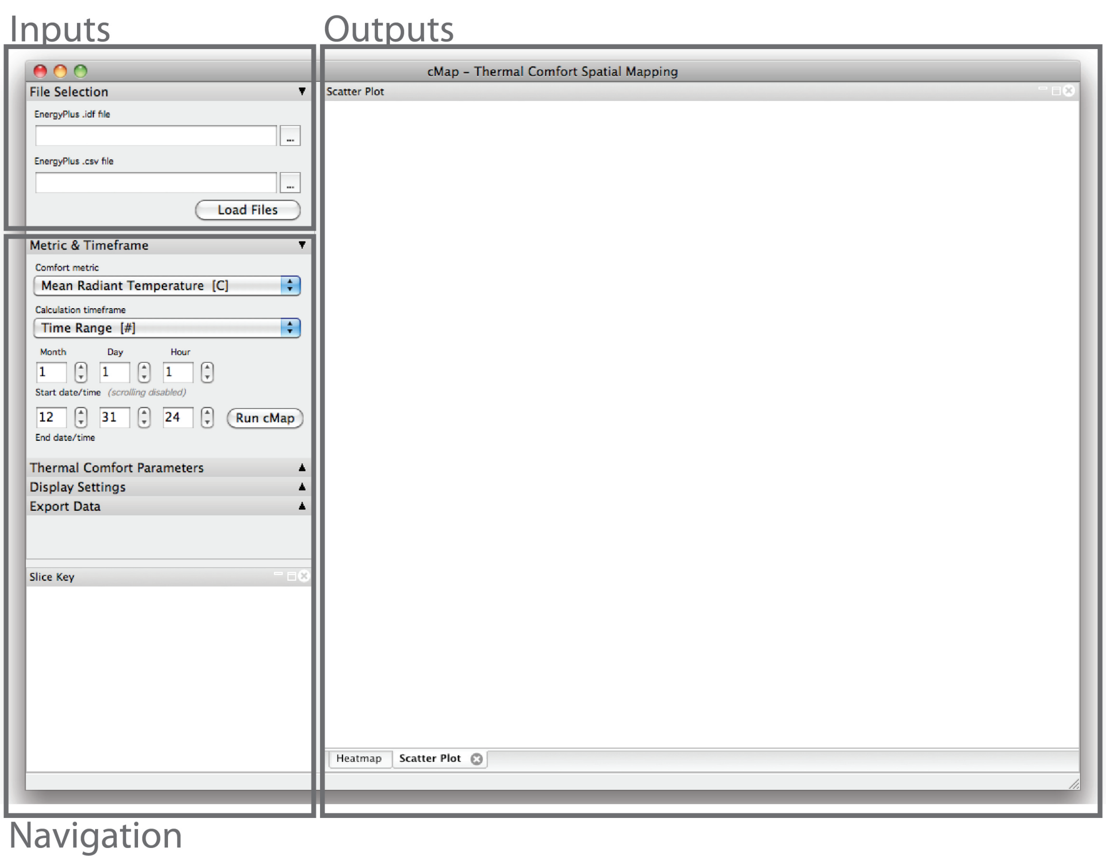


Figure 4-2: Diagram illustrating the analysis workflow within cMap.

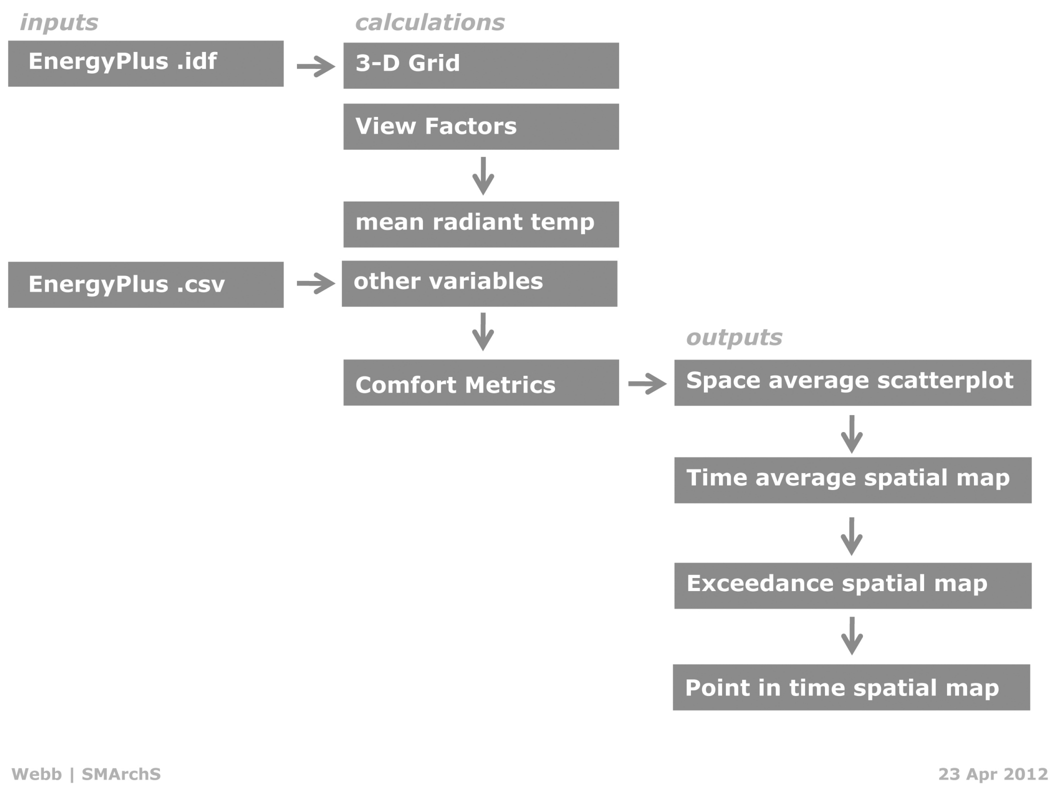


Diagram4.jpg

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

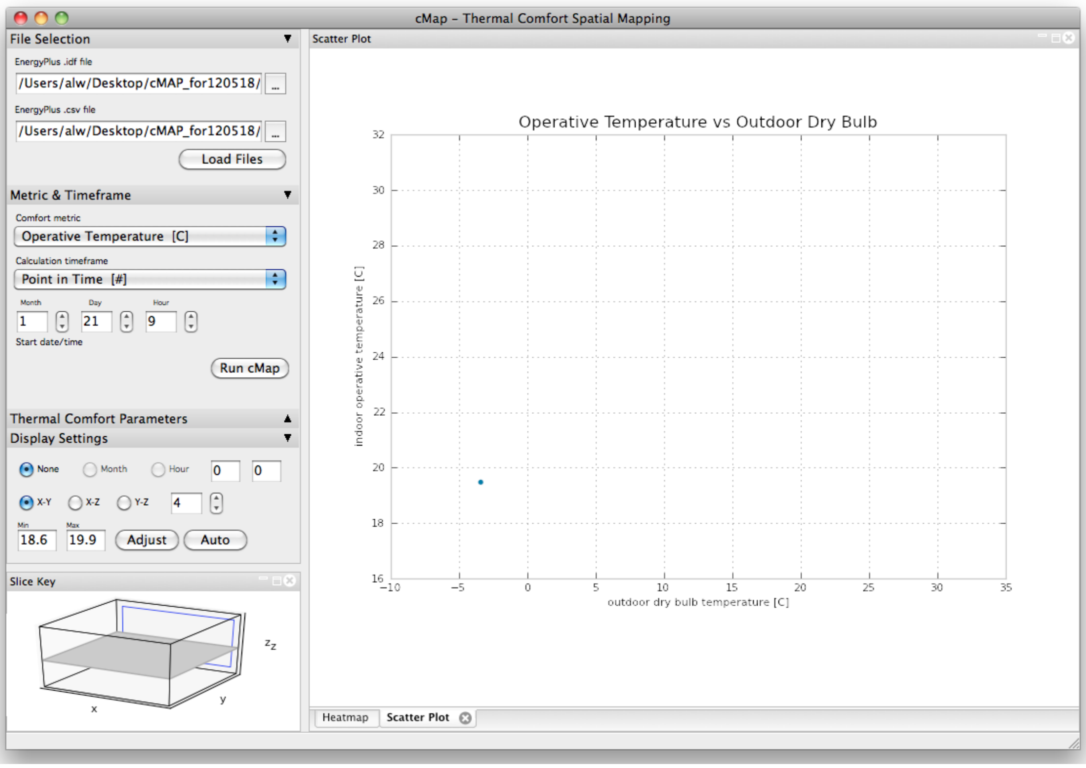


Figure 4-4: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single point in time in January, slice in the X-Y direction

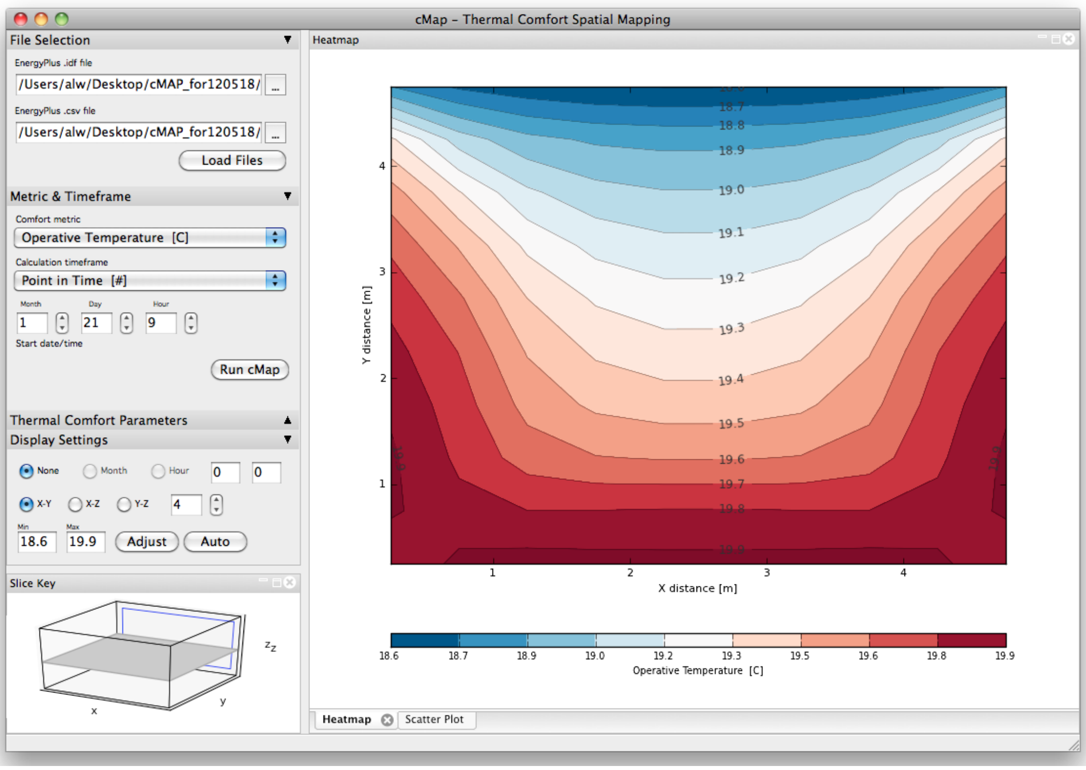


Figure 4-5: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single point in time in January, slice in the Y-Z direction

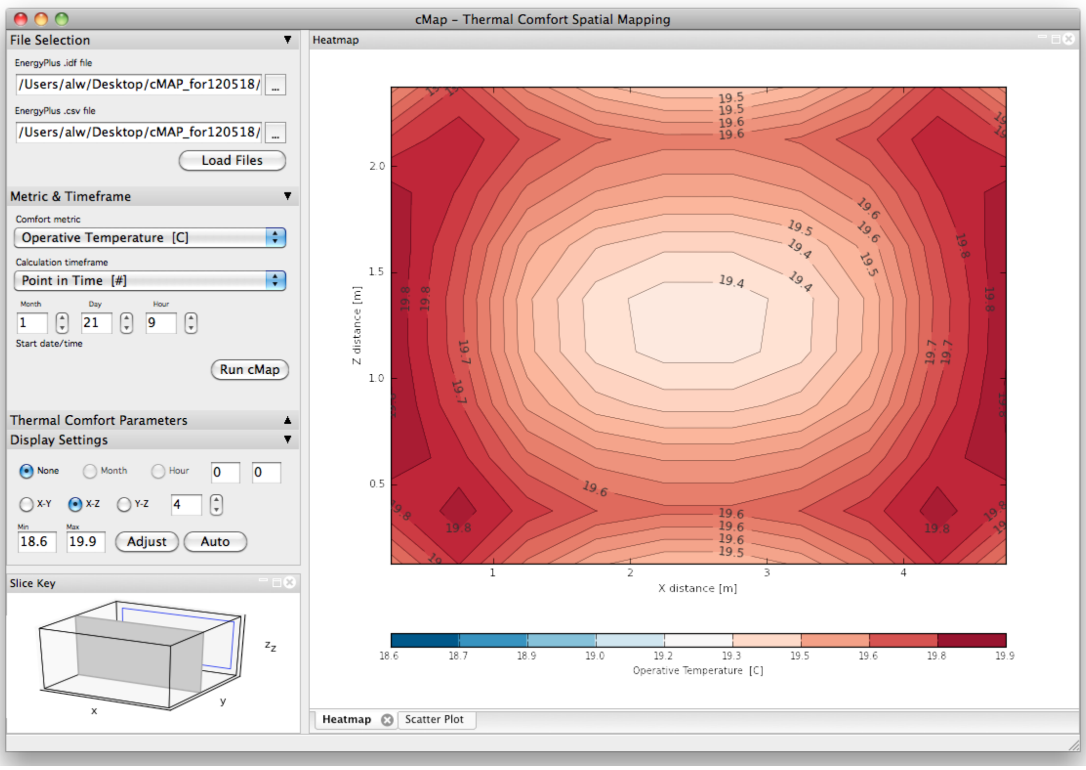


Figure 4-6: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single point in time in January, slice in the X-Z direction

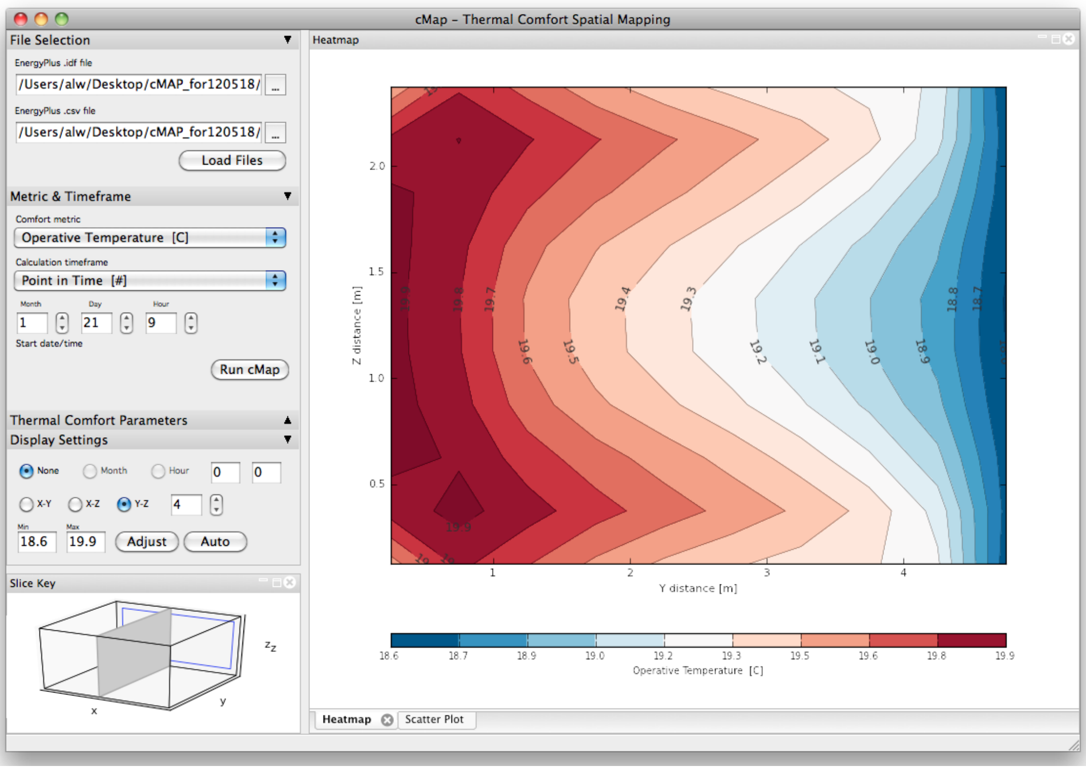


Figure 4-7: Screenshot of cMap showing scatterplot output. Example shown is the ASHRAE 55 adaptive comfort model, for a range of time during the summer. Data points for the month of June are highlighted in red.

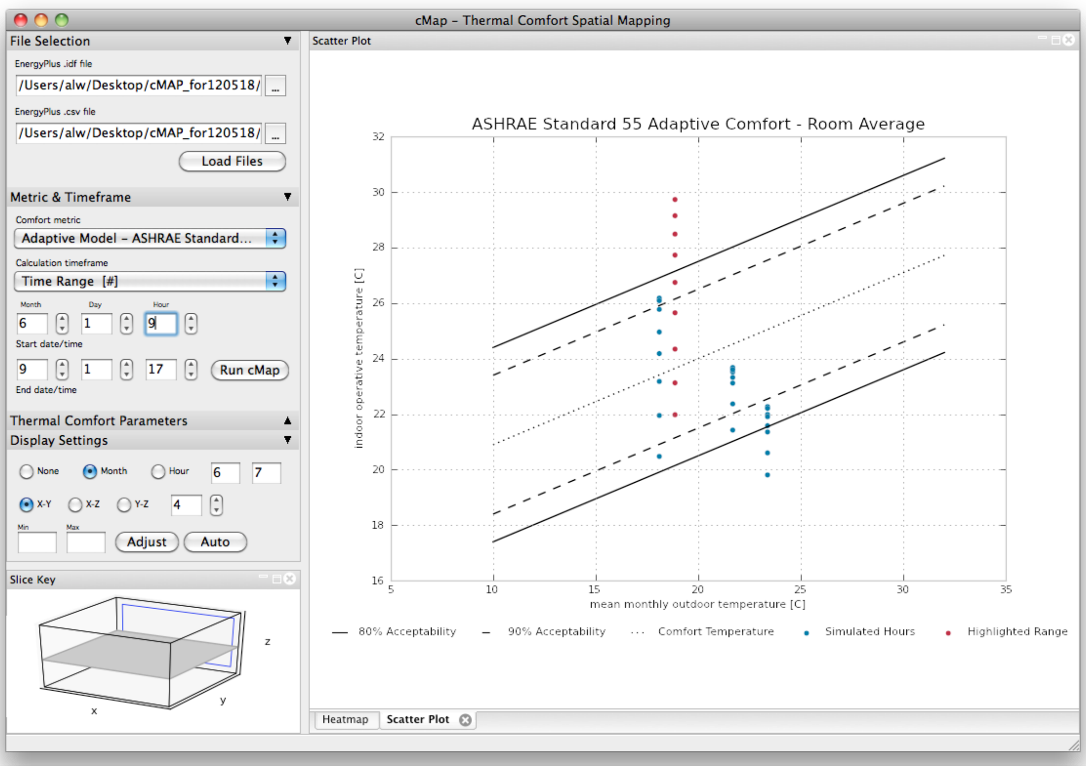


Figure 4-8: Screenshot of cMap showing scatterplot output. Example shown is the ASHRAE 55 adaptive comfort model, for a single peak point in time during the summer.

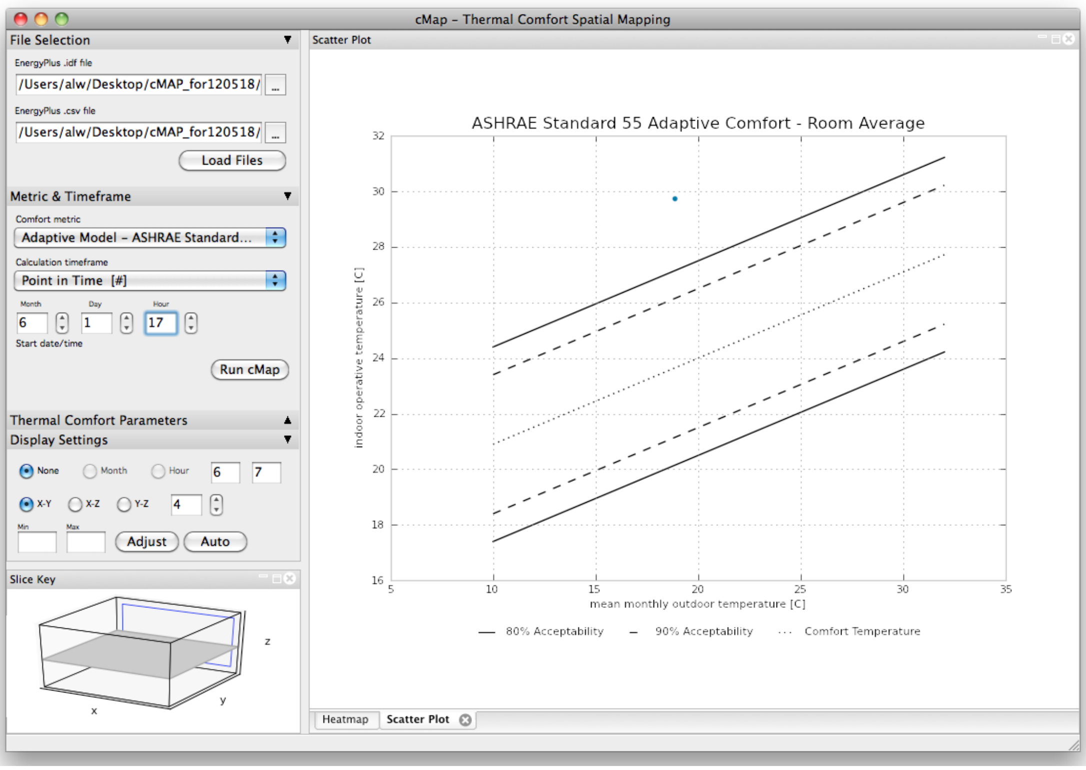


Figure 4-9: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single peak point in time in June, slice in the X-Y direction

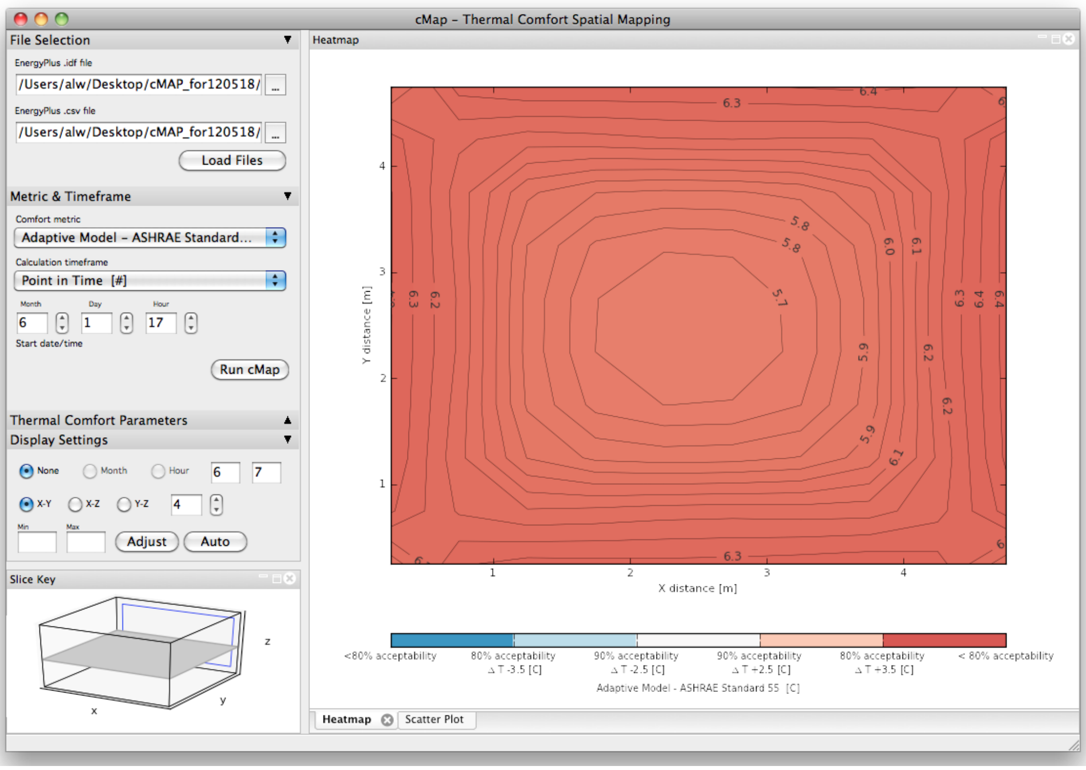


Figure 4-10: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single peak point in time in June, slice in the Y-Z direction

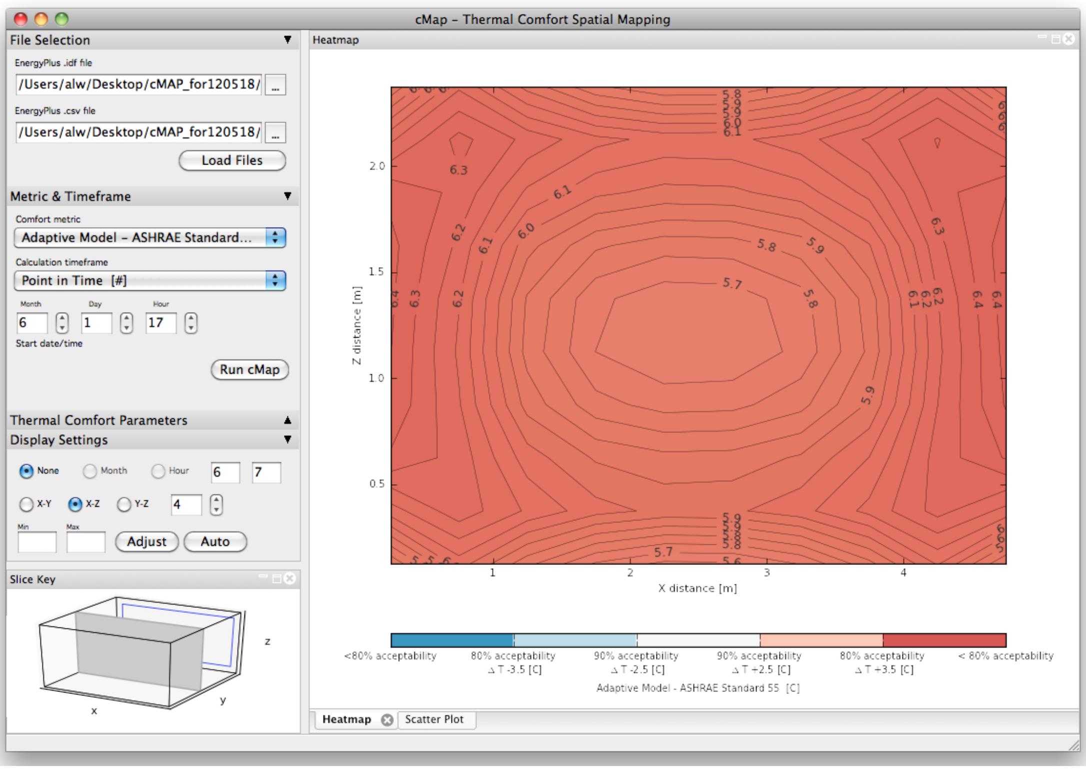
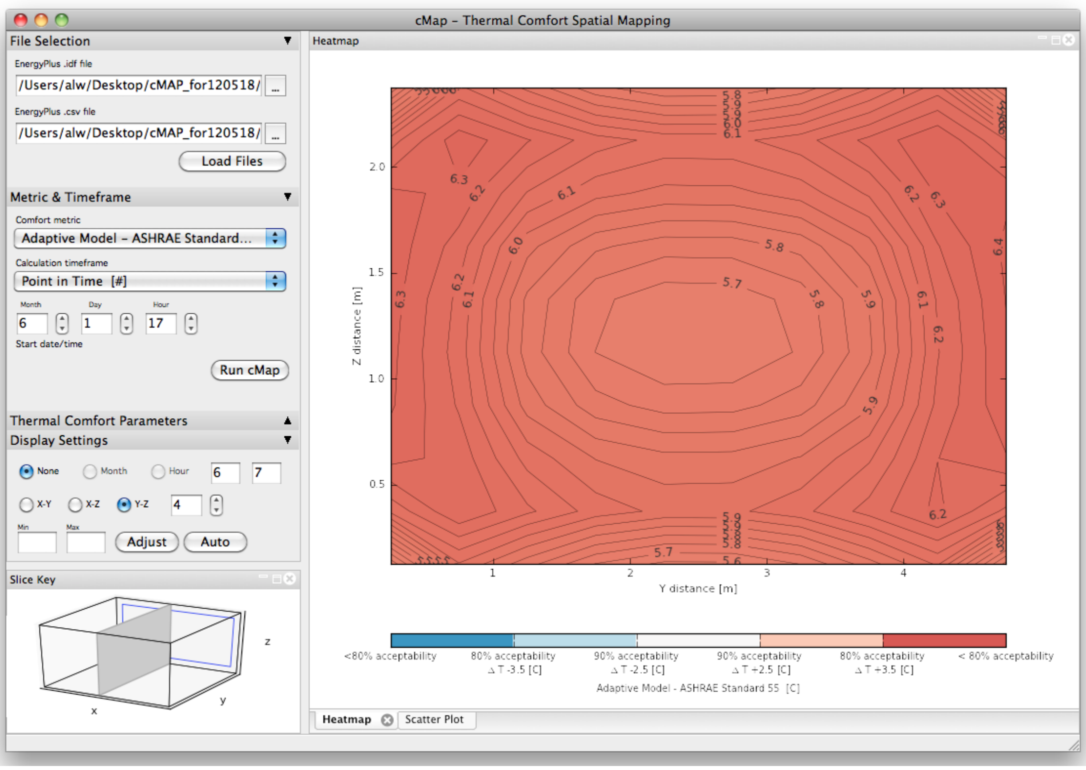


Figure 4-11: Screenshot of cMap showing the contour heatmap output. Example shown is the operative temperature for a single peak point in time in June, slice in the X-Z direction



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-adapatable 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.

```

# !/usr/bin/env python
# Filename: cMap.py

""" Script creates 3-D set of grid points from an EnergyPlus .idf file
Script uses EnergyPlus .csv file to calculate comfort outputs at each point
Script outputs results as .png image file
Script includes interface for controlling raw inputs & visualizing output
"""

# BEGIN INTERFACE -----------------------------------------------------
# Mapping Script Imports ----------------------------------------------
from numpy import *
import csv
import re
import math
from matplotlib import pyplot, mpl, cm
```

```
from matplotlib.colors import *
from scipy.optimize import brentq
from matplotlib.lines import Line2D

# Interface Imports ---------------------------------------------------
import sys
import os
import wx
import wx.lib.agw.aui as aui
import wx.combo
import wx.lib.buttons as buttons
import wx.lib.agw.floatspin as FS
import wx.lib.scrolledpanel as scrolled
from mpl_toolkits.mplot3d import axes3d
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import wx.lib.agw.foldpanelbar as fpb
from matplotlib.figure import Figure
from matplotlib.backends.backend_wxagg import \
    FigureCanvasWxAgg as FigureCanvas
import matplotlib.font_manager as fm

# Interface Components ------------------------------------------------
class MyFrame(wx.Frame):

    def __init__(self, parent, ID, title):
        wx.Frame.__init__(self, parent, ID, title, size=(1100, 760))

        # AUI Panes, Panels, Menu -------------------------------------
        self._mgr = aui.AuiManager(self)

        self.panel1 = wx.Panel(self, -1)
        self.panel2 = wx.Panel(self, -1)
```

```python
        self.panel2.SetBackgroundColour("White")
        self.panel3 = wx.Panel(self, -1)
        self.panel3.SetBackgroundColour("White")
        self.panel4 = wx.Panel(self, -1)
        self.panel4.SetBackgroundColour("White")

        self.CreateStatusBar()

        # add the panes to the manager
        self._mgr.AddPane(self.panel1, aui.AuiPaneInfo().
                Caption("Input Parameters").CaptionVisible(False).Left().
                Layer(2).Floatable(True).Dockable(True).
                MinimizeButton(True).MaximizeButton(True).CloseButton(True).
                BestSize(wx.Size(300, 760)))
        self._mgr.AddPane(self.panel2, aui.AuiPaneInfo().Name("panel2").
                Caption("Heatmap").Center().Floatable(True).Dockable(True).
                MinimizeButton(True).MaximizeButton(True).CloseButton(True).
                BestSize(wx.Size(400, 360)))
        self._mgr.AddPane(self.panel3, aui.AuiPaneInfo().Name("panel3").
                Caption("Scatter Plot").Center().Floatable(True).
                Dockable(True).MinimizeButton(True).MaximizeButton(True).
                CloseButton(True).BestSize(wx.Size(400, 360)),
                target=self._mgr.GetPane("panel2"))
        self._mgr.AddPane(self.panel4, aui.AuiPaneInfo().Name("panel4").
                Caption("Slice Key").Left().Floatable(True).Dockable(True).
                MinimizeButton(True).MaximizeButton(True).CloseButton(True).
                BestSize(wx.Size(200, 200)))

        # Float panel4
        self._mgr.GetPane("panel4").Float().FloatingPosition((940,480)). \
                FloatingSize((200,150))

```

```
        # Tell the manager to commit all the changes just made
        self._mgr.Update()
        self.Bind(wx.EVT_CLOSE, self.OnClose)

        # Default number of gridpoints --------------------------------
        self.gpt = 10

        # Sizers & Sizer Items ----------------------------------------
        # set-up all sizers on panel1
        idfSizer  = wx.BoxSizer(wx.VERTICAL)
        csvSizer  = wx.BoxSizer(wx.VERTICAL)
        calcTextSizer1  = wx.GridBagSizer(hgap=6, vgap=0)
        calcTextSizer  = wx.GridSizer(rows=1, cols=3, hgap=40, vgap=0)
        dateSizer1  = wx.GridBagSizer(hgap=6, vgap=0)
        dateSizer2  = wx.GridBagSizer(hgap=6, vgap=0)
        dateSizer3  = wx.GridBagSizer(hgap=6, vgap=0)

        comfSizer  = wx.GridBagSizer(hgap=6, vgap=0)
        pointTextSizer  = wx.GridSizer(rows=1, cols=2, hgap=110, vgap=0)
        cutSizer  = wx.GridBagSizer(hgap=10, vgap=0)
        colorTextSizer  = wx.GridSizer(rows=1, cols=1)
        colorSizer  = wx.GridBagSizer(hgap=7, vgap=0)
        pointSizer  = wx.GridBagSizer(hgap=0, vgap=0)
        scaleTextSizer  = wx.GridSizer(rows=1, cols=3, hgap=25, vgap=0)
        scaleSizer  = wx.GridBagSizer(hgap=5, vgap=0)
        printSizer  = wx.BoxSizer(wx.VERTICAL)

        # all font & caption bar styles -------------------------------
        titleFont = wx.Font(13, wx.SWISS, wx.NORMAL, wx.NORMAL)
        subFont = wx.Font(9, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
        subFontSM = wx.Font(8, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
        subBoldFont = wx.Font(9, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
```

```
        dateFont = wx.Font(7, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
        scrollingFont = wx.Font(9, wx.DEFAULT, wx.ITALIC, wx.NORMAL)

        cs = fpb.CaptionBarStyle()
        cs.SetCaptionStyle(fpb.CAPTIONBAR_GRADIENT_V)
        color1 = wx.Colour(220, 220, 220)
        color2 = wx.Colour(195, 195, 195)
        cs.SetFirstColour(color1)
        cs.SetSecondColour(color2)
        cs.SetCaptionFont(titleFont)

        # Sizer Items - controls and static texts ---------------------
        # create fold panel instance on main panel
        self.fold_panel = fpb.FoldPanelBar(self.panel1, wx.ID_ANY,
            style=fpb.FPB_VERTICAL)

        # Fold panel bar - file selector -------------------------------
        sectFile = self.fold_panel.AddFoldPanel("File Selection",
            collapsed=False, cbstyle=cs)
        subpanelFile = wx.Panel(sectFile, -1)
        sizerFile = wx.BoxSizer(wx.VERTICAL)

        # texts
        idfTxt = wx.StaticText(subpanelFile, -1, "EnergyPlus .idf file",
            style=wx.ALIGN_LEFT)
        idfTxt.SetFont(subFont)
        csvTxt = wx.StaticText(subpanelFile, -1, "EnergyPlus .csv file",
            style=wx.ALIGN_LEFT)
        csvTxt.SetFont(subFont)

        # controls
        self.idfSel = FileSelectorComboIDF(subpanelFile, size=(280, -1))
```

```
        idfSizer.Add(self.idfSel, 0, wx.LEFT, border=5)
        self.csvSel = FileSelectorComboCSV(subpanelFile, size=(280, -1))
        csvSizer.Add(self.csvSel, 0, wx.LEFT, border=5)

        # load button
        self.load = wx.Button(subpanelFile, -1, 'Load Files',
            size=(110, -1))
        self.load.Bind(wx.EVT_BUTTON, self.loadFiles)

        # put controls on folding sizer
        sizerFile.Add(idfTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 7)
        sizerFile.Add(idfSizer, 0, wx.TOP|wx.BOTTOM, 2)
        sizerFile.AddSpacer(5)
        sizerFile.Add(csvTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 7)
        sizerFile.Add(csvSizer, 0, wx.TOP|wx.BOTTOM, 2)
        sizerFile.Add(self.load, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT, 2)

        sizerFile.AddSpacer(15)

        # set folding sizer on folding panel
        subpanelFile.SetSizer(sizerFile)
        subpanelFile.Fit()

        # add folding panel to foldpanelbar
        self.fold_panel.AddFoldPanelWindow(sectFile, subpanelFile,
            fpb.FPB_ALIGN_LEFT)
        self.fold_panel.AddFoldPanelSeparator(sectFile)

        # Fold panel bar - metric selector ----------------------------
        sectMetric = self.fold_panel.AddFoldPanel("Metric & Timeframe",
            collapsed=False, cbstyle=cs)
        subpanelMetric = wx.Panel(sectMetric, -1)
```

```
        sizerMetric = wx.BoxSizer(wx.VERTICAL)

        # texts
        calcTxt = wx.StaticText(subpanelMetric, -1, "Comfort metric",
            style=wx.ALIGN_LEFT)
        calcTxt.SetFont(subFont)
        timeframeTxt = wx.StaticText(subpanelMetric, -1,
            "Calculation timeframe", style=wx.ALIGN_LEFT)
        timeframeTxt.SetFont(subFont)
        simpMonthTxt = wx.StaticText(subpanelMetric, -1, "Month",
            style=wx.CENTER)
        simpMonthTxt.SetFont(dateFont)
        simpDayTxt = wx.StaticText(subpanelMetric, -1, "Day",
            style=wx.ALIGN_CENTER)
        simpDayTxt.SetFont(dateFont)
        simpHourTxt = wx.StaticText(subpanelMetric, -1, "Hour",
            style=wx.ALIGN_CENTER)
        simpHourTxt.SetFont(dateFont)
        stTxt = wx.StaticText(subpanelMetric, -1, "Start date/time",
            style=wx.ALIGN_LEFT)
        stTxt.SetFont(subFont)
        self.enTxt = wx.StaticText(subpanelMetric, -1, "End date/time",
            style=wx.ALIGN_LEFT)
        self.enTxt.SetFont(subFont)
        calcTextSizer1.Add(stTxt,pos=(0,0))

        # add text to text gridsizers
        calcTextSizer.Add(simpMonthTxt, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL, 0)
        calcTextSizer.Add(simpDayTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 4)
        calcTextSizer.Add(simpHourTxt, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL, 0)

        # controls
```

```
        # calc type selector
        self.calcDrop = wx.Choice(parent=subpanelMetric, size=(280, -1))
        self.calcList = [
            'Mean Radiant Temperature  [C]',
            'Operative Temperature  [C]',
            'Predicted Mean Vote  [+/-]',
            'Percentage People Dissatisfied  [%]',
            'Adaptive Model - ASHRAE Standard 55  [C]',
            'Adaptive Model - EN Standard 15251  [C]',
            'Adaptive Model - NPR-CR 1752 Type Beta  [C]',
            ]
        self.calcDrop.AppendItems(strings=self.calcList)
        self.calcDrop.Select(n=0)

        self.timeframeDrop = wx.Choice(parent=subpanelMetric, size=(280, -1))
        self.timeframeList = [
            'Time Range  [#] ',
            'Point in Time  [#] ',
            ]
        self.timeframeDrop.AppendItems(strings=self.timeframeList)
        self.timeframeDrop.Select(n=0)
        self.timeframeDrop.Bind(wx.EVT_CHOICE, self.DisableAnnual)

        # run button
        self.run = wx.Button(subpanelMetric, -1, 'Run cMap', size=(80, -1))
        self.run.Bind(wx.EVT_BUTTON, self.runCmap)
        dateSizer2.Add(self.run, pos=(0,3),flag=wx.TOP|wx.LEFT|wx.ALIGN_RIGHT,
            border=5)

        # calc start date/time spins
        self.StartMonthSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
        self.StartMonthSpin.SetRange(1,12)
```

```
        self.StartMonthSpin.SetValue(1)
        self.StartMonthSpin.Bind(wx.EVT_SPIN, self.runUpdate)
        self.StartMonthSpin.Bind(wx.EVT_TEXT, self.runUpdate)
        dateSizer1.Add(self.StartMonthSpin, pos=(1,0))

        self.StartDaySpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
        self.StartDaySpin.SetRange(1,31)
        self.StartDaySpin.SetValue(1)
        self.StartDaySpin.Bind(wx.EVT_SPIN, self.runUpdate)
        self.StartDaySpin.Bind(wx.EVT_TEXT, self.runUpdate)
        dateSizer1.Add(self.StartDaySpin, pos=(1,1))

        self.StartHourSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
        self.StartHourSpin.SetRange(1,24)
        self.StartHourSpin.SetValue(1)
        self.StartHourSpin.Bind(wx.EVT_SPIN, self.runUpdate)
        self.StartHourSpin.Bind(wx.EVT_TEXT, self.runUpdate)
        dateSizer1.Add(self.StartHourSpin, pos=(1,2))

        # calc end date/time spins
        self.EndMonthSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
        self.EndMonthSpin.SetRange(1,12)
        self.EndMonthSpin.SetValue(12)
        dateSizer2.Add(self.EndMonthSpin, pos=(0,0))

        self.EndDaySpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
        self.EndDaySpin.SetRange(1,31)
        self.EndDaySpin.SetValue(31)
        dateSizer2.Add(self.EndDaySpin, pos=(0,1))

        self.EndHourSpin = wx.SpinCtrl(subpanelMetric, -1, size=(60, -1))
        self.EndHourSpin.SetRange(1,24)
```

```
        self.EndHourSpin.SetValue(24)
        dateSizer2.Add(self.EndHourSpin, pos=(0,2))

        dateSizer1.Add(simpMonthTxt, pos=(0,0), flag=wx.LEFT, border=8)
        dateSizer1.Add(simpDayTxt, pos=(0,1), flag=wx.LEFT, border=12)
        dateSizer1.Add(simpHourTxt, pos=(0,2), flag=wx.LEFT, border=12)

        # put controls on folding sizer
        sizerMetric.Add(calcTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 8)
        sizerMetric.AddSpacer(2)
        sizerMetric.Add(self.calcDrop, 0, wx.LEFT, border=5)
        sizerMetric.AddSpacer(10)
        sizerMetric.Add(timeframeTxt, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 8)
        sizerMetric.AddSpacer(2)
        sizerMetric.Add(self.timeframeDrop, 0, wx.LEFT, border=5)
        sizerMetric.AddSpacer(10)
        sizerMetric.Add(dateSizer1, 0, wx.LEFT, 5)
        sizerMetric.Add(calcTextSizer1, flag=wx.LEFT, border=8)
        sizerMetric.AddSpacer(5)
        sizerMetric.Add(dateSizer2, 0, wx.LEFT, 5)
        sizerMetric.Add(self.enTxt, flag=wx.LEFT, border=8)
        sizerMetric.AddSpacer(15)

        # set folding sizer on folding panel
        subpanelMetric.SetSizer(sizerMetric)
        subpanelMetric.Fit()

        # add folding panel to foldpanelbar
        self.fold_panel.AddFoldPanelWindow(sectMetric, subpanelMetric,
            fpb.FPB_ALIGN_LEFT)
        self.fold_panel.AddFoldPanelSeparator(sectMetric)

```

```
        # Fold panel bar - thermal comfort parameters -----------------
        sectComfParams = self.fold_panel.AddFoldPanel  \
            ("Thermal Comfort Parameters", collapsed=False, cbstyle=cs)
        subpanelComfParams = wx.Panel(sectComfParams, -1)
        sizerComfParams = wx.BoxSizer(wx.VERTICAL)

        # texts & controls
        outdoorTxt = wx.StaticText(subpanelComfParams, -1, "OUTDOOR",
            style=wx.ALIGN_RIGHT)
        outdoorTxt.SetFont(subFont)
        comfSizer.Add(outdoorTxt, pos=(0,0),
            flag=wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)

        indoorText = wx.StaticText(subpanelComfParams, -1, "INDOOR",
            style=wx.ALIGN_RIGHT)
        indoorText.SetFont(subFont)
        comfSizer.Add(indoorText, pos=(0,3),
            flag=wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)

        peopleText = wx.StaticText(subpanelComfParams, -1, "PEOPLE",
            style=wx.ALIGN_RIGHT)
        peopleText.SetFont(subFont)
        comfSizer.Add(peopleText, pos=(0,6),
            flag=wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL)

        # display items for outdoor dry bulb
        oDBTxt = wx.StaticText(subpanelComfParams, -1, "dry bulb:",
            style=wx.ALIGN_RIGHT)
        oDBTxt.SetFont(subFont)
        comfSizer.Add(oDBTxt, pos=(1,0),
            flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=1)

```

```
        self.oDBSpin = wx.StaticText(subpanelComfParams, -1, "-      ",
            style=wx.ALIGN_LEFT)
        self.oDBSpin.SetFont(subBoldFont)
        comfSizer.Add(self.oDBSpin, pos=(1,1),
            flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

        oDBUnit = wx.StaticText(subpanelComfParams, -1, "C",
            style=wx.ALIGN_RIGHT)
        oDBUnit.SetFont(subFontSM)
        comfSizer.Add(oDBUnit, pos=(1,2),
            flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

        # display items for outdoor relative humidity
        oRHTxt = wx.StaticText(subpanelComfParams, -1, "humidity:",
            style=wx.ALIGN_RIGHT)
        oRHTxt.SetFont(subFont)
        comfSizer.Add(oRHTxt, pos=(2,0),
            flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)

        self.oRHSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
            style=wx.ALIGN_LEFT)
        self.oRHSpin.SetFont(subBoldFont)
        comfSizer.Add(self.oRHSpin, pos=(2,1),
            flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

        oRHUnit = wx.StaticText(subpanelComfParams, -1, "%",
            style=wx.ALIGN_RIGHT)
        oRHUnit.SetFont(subFontSM)
        comfSizer.Add(oRHUnit, pos=(2,2),
            flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

        # display items for outdoor airspeed
```

```
        oAirVeloTxt = wx.StaticText(subpanelComfParams, -1, "airspeed",
            style=wx.ALIGN_RIGHT)
        oAirVeloTxt.SetFont(subFont)
        comfSizer.Add(oAirVeloTxt, pos=(3,0),
            flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)

        self.oAirVSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
            style=wx.ALIGN_LEFT)
        self.oAirVSpin.SetFont(subBoldFont)
        comfSizer.Add(self.oAirVSpin, pos=(3,1),
            flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

        oAirVUnit = wx.StaticText(subpanelComfParams, -1, "m/s",
            style=wx.ALIGN_RIGHT)
        oAirVUnit.SetFont(subFontSM)
        comfSizer.Add(oAirVUnit, pos=(3,2),
            flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=1)

        # display items for indoor dry bulb
        zDBTxt = wx.StaticText(subpanelComfParams, -1, "dry bulb:",
            style=wx.ALIGN_RIGHT)
        zDBTxt.SetFont(subFont)
        comfSizer.Add(zDBTxt, pos=(1,3),
            flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)

        self.zDBSpin = wx.StaticText(subpanelComfParams, -1, "-      ",
            style=wx.ALIGN_LEFT)
        self.zDBSpin.SetFont(subBoldFont)
        comfSizer.Add(self.zDBSpin, pos=(1,4),
            flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

        zDBUnit = wx.StaticText(subpanelComfParams, -1, "C",
```

```
        style=wx.ALIGN_RIGHT)
    zDBUnit.SetFont(subFontSM)
    comfSizer.Add(zDBUnit, pos=(1,5),
        flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

    # display items for indoor relative humidity
    zRHTxt = wx.StaticText(subpanelComfParams, -1, "humidity:",
        style=wx.ALIGN_RIGHT)
    zRHTxt.SetFont(subFont)
    comfSizer.Add(zRHTxt, pos=(2,3),
        flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)

    self.zRHSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
        style=wx.ALIGN_LEFT)
    self.zRHSpin.SetFont(subBoldFont)
    comfSizer.Add(self.zRHSpin, pos=(2,4),
        flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

    zRHUnit = wx.StaticText(subpanelComfParams, -1, "%",
        style=wx.ALIGN_RIGHT)
    zRHUnit.SetFont(subFontSM)
    comfSizer.Add(zRHUnit, pos=(2,5),
        flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

    # display items for indoor airspeed
    zAirVTxt = wx.StaticText(subpanelComfParams, -1, "airspeed:",
        style=wx.ALIGN_RIGHT)
    zAirVTxt.SetFont(subFont)
    comfSizer.Add(zAirVTxt, pos=(3,3),
        flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)

    self.zAirVSchSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
```

```
                style=wx.ALIGN_LEFT)
            self.zAirVSchSpin.SetFont(subBoldFont)
            comfSizer.Add(self.zAirVSchSpin, pos=(3,4),
                flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

            zAirVUnit = wx.StaticText(subpanelComfParams, -1, "m/s",
                style=wx.ALIGN_RIGHT)
            zAirVUnit.SetFont(subFontSM)
            comfSizer.Add(zAirVUnit, pos=(3,5),
                flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

            # display items for clothing level
            zCloSchTxt = wx.StaticText(subpanelComfParams, -1, "clothing:",
                style=wx.ALIGN_RIGHT)
            zCloSchTxt.SetFont(subFont)
            comfSizer.Add(zCloSchTxt, pos=(1,6),
                flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=1)

            self.zCloSchSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
                style=wx.ALIGN_LEFT)
            self.zCloSchSpin.SetFont(subBoldFont)
            comfSizer.Add(self.zCloSchSpin, pos=(1,7),
                flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

            CloUnit = wx.StaticText(subpanelComfParams, -1, "clo",
                style=wx.ALIGN_RIGHT)
            CloUnit.SetFont(subFontSM)
            comfSizer.Add(CloUnit, pos=(1,8),
                flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

            # display items for activity level
            zActSchTxt = wx.StaticText(subpanelComfParams, -1, "activity:",
```

```
            style=wx.ALIGN_RIGHT)
        zActSchTxt.SetFont(subFont)
        comfSizer.Add(zActSchTxt, pos=(2,6),
            flag=wx.TOP|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL,border=3)

        self.zActSchSpin = wx.StaticText(subpanelComfParams, -1, "-    ",
            style=wx.ALIGN_LEFT)
        self.zActSchSpin.SetFont(subBoldFont)
        comfSizer.Add(self.zActSchSpin, pos=(2,7),
            flag=wx.TOP|wx.ALIGN_CENTER_VERTICAL,border=3)

        MetUnit = wx.StaticText(subpanelComfParams, -1, "met",
            style=wx.ALIGN_RIGHT)
        MetUnit.SetFont(subFontSM)
        comfSizer.Add(MetUnit, pos=(2,8),
            flag=wx.TOP|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL,border=3)

        # put controls on folding sizer
        sizerComfParams.AddSpacer(5)
        sizerComfParams.Add(comfSizer, 0,
            wx.LEFT| wx.ALIGN_CENTER_VERTICAL, 5)
        sizerComfParams.AddSpacer(15)

        # set folding sizer on folding panel
        subpanelComfParams.SetSizer(sizerComfParams)
        subpanelComfParams.Fit()

        # add folding panel to foldpanelbar
        self.fold_panel.AddFoldPanelWindow(sectComfParams,
            subpanelComfParams, fpb.FPB_ALIGN_LEFT)
        self.fold_panel.AddFoldPanelSeparator(sectComfParams)
```

```
        # Fold panel bar - display settings ---------------------------
        sectDisplay = self.fold_panel.AddFoldPanel("Display Settings",
            collapsed=False, cbstyle=cs)
        subpanelDisplay = wx.Panel(sectDisplay, -1)
        sizerDisplay = wx.BoxSizer(wx.VERTICAL)

        # texts
        scaleMinTxt = wx.StaticText(subpanelDisplay, -1, "Min",
            style=wx.CENTER)
        scaleMinTxt.SetFont(dateFont)
        scaleMaxTxt = wx.StaticText(subpanelDisplay, -1, "Max",
            style=wx.CENTER)
        scaleMaxTxt.SetFont(dateFont)
        scaleSizer.Add(scaleMinTxt, pos=(0,0),
            flag=wx.LEFT|wx.ALIGN_CENTER_VERTICAL, border=5)
        scaleSizer.Add(scaleMaxTxt,  pos=(0,1),
            flag=wx.LEFT|wx.ALIGN_CENTER_VERTICAL, border=5)

        # controls
        # analysis cut selector
        self.xy = wx.RadioButton(subpanelDisplay, -1, 'X-Y',
            style=wx.RB_GROUP)
        self.xz = wx.RadioButton(subpanelDisplay, -1, 'X-Z')
        self.yz = wx.RadioButton(subpanelDisplay, -1, 'Y-Z')
        self.xy.SetFont(subFont)
        self.xz.SetFont(subFont)
        self.yz.SetFont(subFont)
        self.xy.Bind(wx.EVT_RADIOBUTTON, self.OnCheck)
        self.xz.Bind(wx.EVT_RADIOBUTTON, self.OnCheck)
        self.yz.Bind(wx.EVT_RADIOBUTTON, self.OnCheck)
        cutSizer.Add(self.xy, pos=(0,0), flag=wx.LEFT|wx.TOP, border=5)
        cutSizer.Add(self.xz, pos=(0,1), flag=wx.LEFT|wx.TOP, border=5)
```

```
        cutSizer.Add(self.yz, pos=(0,2), flag=wx.LEFT|wx.TOP, border=5)
        self.xy.SetValue(True)

        # slice selector
        self.sliceSpin = wx.SpinCtrl(subpanelDisplay, -1, size=(60, -1))
        self.sliceSpin.SetRange(0,self.gpt-1)
        self.sliceSpin.SetValue(0)
        self.sliceSpin.Bind(wx.EVT_SPIN, self.OnSlice)
        self.sliceSpin.Bind(wx.EVT_TEXT, self.OnSlice)
        cutSizer.Add(self.sliceSpin, pos=(0,3),
            flag=wx.LEFT|wx.ALIGN_CENTER_VERTICAL, border=8)

        # scatterplot color selector
        self.noColor = wx.RadioButton(subpanelDisplay, -1, 'None',
            style=wx.RB_GROUP)
        self.monthColor = wx.RadioButton(subpanelDisplay, -1, 'Month')
        self.timeColor = wx.RadioButton(subpanelDisplay, -1, 'Hour')
        self.noColor.SetFont(subFont)
        self.monthColor.SetFont(subFont)
        self.timeColor.SetFont(subFont)
        self.noColor.Bind(wx.EVT_RADIOBUTTON, self.runCmap)
        self.monthColor.Bind(wx.EVT_RADIOBUTTON, self.runCmap)
        self.timeColor.Bind(wx.EVT_RADIOBUTTON, self.runCmap)
        colorSizer.Add(self.noColor, pos=(0,0), flag=wx.LEFT, border=5)
        colorSizer.Add(self.monthColor, pos=(0,1), flag=wx.LEFT, border=15)
        colorSizer.Add(self.timeColor, pos=(0,2), flag=wx.LEFT, border=15)
        self.colorStart = wx.TextCtrl(subpanelDisplay, value='0',size=(30, -1))
        self.colorStart.Bind(wx.EVT_TEXT, self.runUpdate)
        colorSizer.Add(self.colorStart, pos=(0,3), flag = wx.LEFT, border=15)
        self.colorEnd = wx.TextCtrl(subpanelDisplay, value='0',size=(30, -1))
        self.colorEnd.Bind(wx.EVT_TEXT, self.runUpdate)
        colorSizer.Add(self.colorEnd, pos=(0,4),flag = wx.LEFT, border=5)
```

```
        self.noColor.SetValue(True)

        # colors for plots
        self.facecolorBlue = '#0E689D'
        self.facecolorRed = '#AC1D34'
        self.edgecolor = '#ECE7F2'
        self.gridcolor = '#737373'
        self.axescolor = '#BDBDBD'
        self.alpha = 1
        self.lw = 0.2
        self.scattersize = 15
        self.cmap = cm.RdBu_r

        # colormap scale selector
        self.cmapMinDisplay = wx.TextCtrl(subpanelDisplay, value=' ',
            size=(40, -1))
        scaleSizer.Add(self.cmapMinDisplay, pos=(1,0), flag = wx.LEFT,
            border=5)
        self.cmapMaxDisplay = wx.TextCtrl(subpanelDisplay, value=' ',
            size=(40, -1))
        scaleSizer.Add(self.cmapMaxDisplay, pos=(1,1), flag = wx.LEFT,
            border=5)
        self.adjust = wx.Button(subpanelDisplay, -1, 'Adjust',
            size=(65, -1))
        self.adjust.Bind(wx.EVT_BUTTON, self.OnAdjustScale)
        scaleSizer.Add(self.adjust, pos=(1,2), flag = wx.LEFT,
            border=5)
        self.auto = wx.Button(subpanelDisplay, -1, 'Auto',
            size=(65, -1))
        self.auto.Bind(wx.EVT_BUTTON, self.OnAutoScale)
        scaleSizer.Add(self.auto, pos=(1,3), flag = wx.LEFT, border=0)

```

```
        # put controls on folding sizer
        sizerDisplay.AddSpacer(5)
        sizerDisplay.Add(colorSizer, 0, wx.LEFT| wx.ALIGN_CENTER_VERTICAL, 5)
        sizerDisplay.AddSpacer(7)
        sizerDisplay.Add(cutSizer, 0, wx.LEFT| wx.ALIGN_CENTER_VERTICAL, 5)
        sizerDisplay.AddSpacer(3)
        sizerDisplay.Add(scaleTextSizer, 0, wx.LEFT|
            wx.ALIGN_CENTER_VERTICAL, 10)
        sizerDisplay.Add(scaleSizer, 0, wx.LEFT, 5)
        sizerDisplay.AddSpacer(15)

        # set folding sizer on folding panel
        subpanelDisplay.SetSizer(sizerDisplay)
        subpanelDisplay.Fit()

        # add folding panel to foldpanelbar
        self.fold_panel.AddFoldPanelWindow(sectDisplay, subpanelDisplay,
            fpb.FPB_ALIGN_LEFT)
        self.fold_panel.AddFoldPanelSeparator(sectDisplay)

        # Fold panel bar - export data --------------------------------
        sectExport = self.fold_panel.AddFoldPanel("Export Data",
            collapsed=False, cbstyle=cs)
        subpanelExport = wx.Panel(sectExport, -1)
        sizerExport = wx.BoxSizer(wx.VERTICAL)

        # controls
        # print to .png
        self.printPNG = wx.Button(subpanelExport, -1, 'Export Plots',
            size=(100, -1))
        self.printPNG.Bind(wx.EVT_BUTTON, self.OnPrint)
        printSizer.Add(self.printPNG, 0, wx.ALIGN_RIGHT, border=0)
```

```
        # put controls on folding sizer
        sizerExport.AddSpacer(5)
        sizerExport.Add(printSizer, 0, wx.LEFT | wx.ALIGN_LEFT, 5)
        sizerExport.AddSpacer(10)

        # set folding sizer on folding panel
        subpanelExport.SetSizer(sizerExport)
        subpanelExport.Fit()

        # add folding panel to foldpanelbar
        self.fold_panel.AddFoldPanelWindow(sectExport, subpanelExport,
            fpb.FPB_ALIGN_LEFT)
        self.fold_panel.AddFoldPanelSeparator(sectExport)

        # Set Sizers ---------------------------------------------------
        # make a sizer for the scrolling panel
        scroll_sizer = wx.BoxSizer(wx.VERTICAL)
        scroll_sizer.Add(self.fold_panel,1,wx.EXPAND)

        self.panel1.SetSizer(scroll_sizer)
        self.panel1.Layout()

        # panel2 sizer set-up
        self.graphSizer = wx.BoxSizer(wx.VERTICAL)
        self.panel2.SetSize((800, 760))
        self.panel2.SetSizer(self.graphSizer)
        self.panel2.Fit()
        self.panel2.Centre()

        # panel3 set-up
        self.scatterSizer = wx.BoxSizer(wx.VERTICAL)
```

```
        self.panel3.SetSize((800, 760))
        self.panel3.SetSizer(self.scatterSizer)
        self.panel3.Fit()
        self.panel3.Centre()

        # panel4 sizer set-up
        self.keySizer = wx.BoxSizer(wx.VERTICAL)
        self.panel4.SetSize((200, 150))
        self.panel4.SetSizer(self.keySizer)
        self.panel4.Fit()
        self.panel4.Centre()

        # Create matplotlib Object ------------------------------------

        # plot containers for panel2 heatmap
        self.chart = wx.Panel(self.panel2)
        self.fig = Figure(dpi=100, facecolor='none')
        self.canvas = FigureCanvas(self.chart, -1, self.fig)
        self.graphSizer.Add(self.chart, 0, wx.ALL|wx.EXPAND, 5)

        # link heatmap plot containers to resize functions
        self._SetSize()
        self.canvas.draw()
        self.chart._resizeflag = False
        self.chart.Bind(wx.EVT_IDLE, self._onIdle)
        self.chart.Bind(wx.EVT_SIZE, self._onSize)

        # plot containers for panel3 heatmap
        self.scatterChart = wx.Panel(self.panel3)
        self.scatterFig = Figure(dpi=100, facecolor='none')
        self.scatterCanvas = FigureCanvas(self.scatterChart, -1,
            self.scatterFig)
```

```
        self.scatterSizer.Add(self.scatterChart, 0, wx.ALL|wx.EXPAND, 5)

        # link scatter plot containers to resize functions
        self._SetSizeScatter()
        self.scatterCanvas.draw()
        self.scatterChart._resizeflag = False
        self.scatterChart.Bind(wx.EVT_IDLE, self._onIdleScatter)
        self.scatterChart.Bind(wx.EVT_SIZE, self._onSizeScatter)


        # default variable values
        self.idf = self.idfSel.GetValue()
        self.csv = self.csvSel.GetValue()
        self.cutNum = 0
        self.ind = 0
        self.axH = 'X'
        self.axV = 'Y'

    # Calculation Functions ------------------------------------------

    def readIDF(self):
        '''Parses .idf file to obtain zone geometry'''
        values = []
        valx = []
        valy = []
        winx = []
        winy = []
        winz = []

        # find line numbers
        inidf = open(self.idf,'r')
        textidf = inidf.readlines()
```

```
        zoneLN = textidf.index('  Zone,\r\n')
        floorLN = textidf.index \
            ('    FLOOR,                   !- Surface Type\r\n')
        windowLN = textidf.index \
            ('    WINDOW,                  !- Surface Type\r\n')
        for i,line in enumerate(textidf):
            if i >= zoneLN+3 and i < zoneLN+9 :
                strip = line.lstrip();
                split = strip.split(',');
                new = split.pop(0)
                values.append(new)
        for i,line in enumerate(textidf):
            if i >= floorLN+9 and i < floorLN+13 :
                strip = line.lstrip();
                split = strip.split(',');
                xnew = split.pop(0)
                valx.append(xnew)
                ynew = split.pop(0)
                valy.append(ynew)
        for i,line in enumerate(textidf):
            if i >= windowLN+9 and i < windowLN+13 :
                strip = line.lstrip();
                a = strip.replace(';', ', ')
                split = a.split(',');
                winxnew = split.pop(0)
                winx.append(winxnew)
                winynew = split.pop(0)
                winy.append(winynew)
                winznew = split.pop(0)
                winz.append(winznew)

        # pull zone dimensions
```

```
        self.xmin = float(values.pop(0))
        self.ymin = float(values.pop(0))
        self.zmin = float(values.pop(0))
        self.xmax = float(max(valx))
        self.ymax = float(max(valy))
        self.zmax = float(values.pop(2))
        self.winxmax = float(max(winx))
        self.winymax = float(max(winy))
        self.winzmax = float(max(winz))
        self.winxmin = float(min(winx))
        self.winymin = float(min(winy))
        self.winzmin = float(min(winz))

        # create range of gridpoints within zone dimensions
        # sets start point to match Ecotect grid method:
        # http://naturalfrequency.com/articles/analysisgrid
        xspace = self.xmax/self.gpt
        yspace = self.ymax/self.gpt
        zspace = self.zmax/self.gpt
        xstart = xspace/2
        ystart = yspace/2
        zstart = zspace/2

        # number of gridpoints same in all dimensions
        self.pt = mgrid[xstart:self.xmax:xspace,ystart:self.ymax:yspace,
            zstart:self.zmax:zspace]
        self.xcoords = self.pt[0,:,0,0]
        self.ycoords = self.pt[1,0,:,0]
        self.zcoords = self.pt[2,0,0,:]

    def getTemps(self):
        '''Locates surface temps and other variables within .csv file'''
```

```
        self.numbers = genfromtxt(self.csv, delimiter=',')
        self.dates = genfromtxt(self.csv, delimiter=',', usecols=(0),
            dtype='S100')
        variables = (genfromtxt(self.csv, delimiter=',', dtype='S100'))[0,:]
        location = [i for i, item in enumerate(variables) \
            if re.search('Surface Inside Temperature', item)]
        reorder=[2,3,4,0,6,5]
        self.relocation = [location[i] for i in reorder]
        self.loc_oDB = [i for i, item in enumerate(variables) \
            if re.search('Outdoor Dry Bulb', item)]
        self.loc_oRH = [i for i, item in enumerate(variables) \
            if re.search('Outdoor Relative Humidity', item)]
        self.loc_oAirV = [i for i, item in enumerate(variables) \
            if re.search('Zone Outdoor Wind Speed', item)]
        self.loc_zDB = [i for i, item in enumerate(variables) \
            if re.search('Zone Mean Air Temperature', item)]
        self.loc_zMRT = [i for i, item in enumerate(variables) \
            if re.search('Zone Mean Radiant Temperature', item)]
        self.loc_zOpT = [i for i, item in enumerate(variables) \
            if re.search('Zone Operative Temperature', item)]
        self.loc_zRH = [i for i, item in enumerate(variables) \
            if re.search('Zone Air Relative Humidity', item)]
        self.loc_zActSch = [i for i, item in enumerate(variables) \
            if re.search('ACTIVITY_SCH:Schedule Value', item)]
        self.loc_zCloSch = [i for i, item in enumerate(variables) \
            if re.search('CLOTHING_SCH:Schedule Value', item)]
        self.loc_zAirVSch = [i for i, item in enumerate(variables) \
            if re.search('AIR_VELO_SCH:Schedule Value', item)]

    def setDate_Annual(self):
        '''Pulls data from appropriate located columns
        in .csv file for every hour of the year'''
```

```
        self.DBs = []
        self.zoneDBs = []
        self.months = []
        self.temps = []
        for position, item in enumerate(self.dates):
            tem = self.numbers[position,self.relocation]
            rep = repeat(tem,4)
            self.temps.append(rep)
            self.oDB = self.numbers[position,self.loc_oDB]
            self.DBs.append(self.oDB)
            self.zDB = self.numbers[position,self.loc_zDB]
            self.zoneDBs.append(self.zDB)
            self.months.append(item[0:2])
        print shape(self.temps)
        print 'SetDateAnnual done'

    def viewFactors(self):
        '''Determines view factors at every point in 3D grid'''
        # locates each point in the space by comparing to max and min
        xMx = self.xmax - self.pt[0,:,:,:]
        xMn = self.pt[0,:,:,:] - self.xmin
        yMx = self.ymax - self.pt[1,:,:,:]
        yMn = self.pt[1,:,:,:] - self.ymin
        zMx = self.zmax - self.pt[2,:,:,:]
        zMn = self.pt[2,:,:,:] - self.zmin

        # for vertical surfaces (walls) -------------------------------
        # array of a/c and b/c for each surface at each point
        acbcVert = array([
        # north wall
        [[ xMn/yMx, zMn/yMx],[ xMn/yMx,  zMx/yMx],[ xMx/yMx,  zMx/yMx],
            [ xMx/yMx, zMn/yMx]],
```

```
        # east wall
        [[ yMx/xMx, zMn/xMx],[ yMx/xMx,  zMx/xMx],[ yMn/xMx,  zMx/xMx],
            [ yMn/xMx, zMn/xMx]],
        # south wall
        [[ xMn/yMn, zMn/yMn],[ xMn/yMn,  zMx/yMn],[ xMx/yMn,  zMx/yMn],
            [ xMx/yMn, zMn/yMn]],
        # west wall
        [[ yMn/xMn, zMn/xMn],[ yMn/xMn,  zMx/xMn],[ yMx/xMn,  zMx/xMn],
            [ yMx/xMn, zMn/xMn]],
        ])
        # for all surfaces, all points, a/c only
        acV = acbcVert[:,:,0,:,:,:]
        # for all surfaces, all points, b/c only
        bcV = acbcVert[:,:,1,:,:,:]
        # calculate angle factors at each point
        tauV = 1.24186+0.16730*(acV)
        gammaV = 0.61648+(0.08165*(bcV))+(0.05128*(acV))
        FV = 0.120*(1-exp(-(acV)/tauV))*(1-exp(-(bcV)/gammaV))

        # for horizontal surfaces (floor, ceiling) --------------------
        # ceiling
        acbcHori = array([
        [[ xMn/zMx, yMn/zMx],[ xMn/zMx,  yMx/zMx],[ xMx/zMx,  yMx/zMx],
            [ xMx/zMx,  yMn/zMx]],
        # floor
        [[ xMn/zMn, yMn/zMn],[ xMn/zMn,  yMx/zMn],[ xMx/zMn,  yMx/zMn],
            [ xMx/zMn,  yMn/zMn]],
        ])
        # for all surfaces, all points, a/c only
        acH = acbcHori[:,:,0,:,:,:]
        # for all surfaces, all points, b/c only
        bcH = acbcHori[:,:,1,:,:,:]
```

```
        # calculate angle factors at each point
        tauH = 1.59512+0.12788*(acH)
        gammaH = 1.22643+(0.04621*(bcH))+(0.04434*(acH))
        FH = 0.116*(1-exp(-(acH)/tauH))*(1-exp(-(bcH)/gammaH))
        # combine FV and FH
        self.F = concatenate((FV,FH),axis=0)

        # for reference pull angle factors and find max and min
        SS = size(self.pt[0,:,:,:])
        AR = reshape(self.F,(24,SS))
        aFacs = apply_along_axis(sum, 0, AR)
        print 'calcViewFactors done'

    # Metric Annual Calculation Functions -----------------------------
    ''' Calculates comfort values at every point at every hour of the
        year for every metric.  Index functions then index off of that
        for user-requested times.
    '''

    def calcMRTS(self):
        print 'Running Mean Radiant Temperature Calculation...'
        # reshape afacs
        SX = size(self.pt[0,:,0,0])
        SY = size(self.pt[0,0,:,0])
        SZ = size(self.pt[0,0,0,:])
        RF = reshape(self.F,(24,SX,SY,SZ))
        # reformat temps
        AT = asarray(self.temps)
        # multiply by surface temps and sum to get MRT at each point
        listMRTS = ([])
        for position, item in enumerate(AT):
```

```
            ZR = RF*item[:,newaxis,newaxis,newaxis]
            MRTResults = sum(ZR, axis=0)
            listMRTS.append(MRTResults)
        self.MRTS_Annual = asarray(listMRTS)
        print 'SHAPE MRTS', shape(self.MRTS_Annual)
        print 'Finished Mean Radiant Temperature Calculation'

    def calcOpTemp(self):
        '''Calculates operative temperature according to method in
        ASHRAE 55-2004'''
        print 'Starting Operative Temperature Calculation...'
        collectOpTemps = []
        for i in range(0,len(self.numbers)):
            if self.numbers[i,self.loc_zAirVSch] < 0.2:
                OpTemp = (0.5*(self.numbers[i,self.loc_zDB]))+ \
                    (1-0.5)*(self.MRTS_Annual[i])
            if self.numbers[i,self.loc_zAirVSch] >= 0.2 and \
                self.numbers[i,self.loc_zAirVSch] < 0.6:
                OpTemp = (0.5*(self.numbers[i,self.loc_zDB]))+ \
                    (1-0.5)*(self.MRTS_Annual[i])
            else:
                OpTemp = (0.5*(self.numbers[i,self.loc_zDB]))+ \
                    (1-0.5)*(self.MRTS_Annual[i])
            collectOpTemps.append(OpTemp)
        self.OpTemp_Annual = asarray(collectOpTemps)
        print 'Finished Operative Temperature Calculation'


    def calcPMVPPDRoomAverageAnnual(self):
        ''' provides values for PMV scatter plot for every hour of the year
            PMV spatial calc done on-demand using 'Run cMap' button
            from ISO 7730-2005:
```

```
            M is metabolic rate [W/m2]
            W is effective mechanical power [W/m2]
            Icl is clothing insulation [m2K/W]
            fcl is clothing surface factor
            ta is air temperature [C]
            tr is mean radiant temperature [C]
            var is relative air velocity [m/s]
            pa is water vapour partial pressure [Pa]
            hc is convective heat transfer coefficient [W/m2K]
            tcl is clothing surface temperature [C]

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

            PMV may be calculated for different combinations of
            metabolic rate, clothin insulation, air temperature,
            mean radiant temperature, air velocity, and humidity.
            The equations for tcl and hc may be solved by iteration.

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

                0.8 met < M < 4 met (46 W/m2 to 232 W/m2)
                0 clo < Icl < 2 clo (0 m2K/W to 0.310 m2K/W)
                10 oC < ta < 30 oC
                10 oC < tr < 30 oC
                0 m/s < var < 1 m/s
                0 Pa < Pa < 2700 Pa
        '''
        self.PMV_RoomAverageAnnual = []
        self.PPD_RoomAverageAnnual = []
        roomAveragesMRT = []
```

```
        for i in range(0,len(self.MRTS_Annual)):
            results = mean(self.MRTS_Annual[i])
            roomAveragesMRT.append(results)
        print 'len roomAverages MRT',len(roomAveragesMRT)

        for position, item in enumerate(self.dates):
            # find variable values
            self.zDB = self.numbers[position,self.loc_zDB]
            self.zRH = self.numbers[position,self.loc_zRH]
            self.zActSch = self.numbers[position,self.loc_zActSch]
            self.zCloSch = self.numbers[position,self.loc_zCloSch]
            self.zAirVSch = self.numbers[position,self.loc_zAirVSch]

            # set variables
            ta = self.zDB
            RH = self.zRH/100
            var = self.zAirVSch
            met = self.zActSch/58.15
            clo = self.zCloSch
            print 'set variables ok'

            # auto calc variables
            Icl = clo*0.155
            M = met*58.15
            W = 0
            MW = M-W
            print 'auto calc variables ok'

            # steam table data - from 2.006 Property Data Tables
            steamTableTemps = [
            0.01, 5, 10, 15, 20, 25, 30, 35,
            40, 45, 50, 55, 60, 65, 70, 75,
```

```
        80, 85, 90, 95, 100
        ]
        steamTablePsat = [
        611.66, 872.58, 1228.2, 1705.8, 2339.3, 3169.9, 4247, 5629,
        7384.9, 9595, 12352, 15762, 19946, 25042, 31201, 38595,
        47414, 57867, 70182, 84608, 101420
        ]
        # calculate saturation pressure at ta
        Psat = interp(ta, steamTableTemps, steamTablePsat)
        # calculate water vapor partial pressure
        pa = RH*Psat
        print 'steam tables ok'

        # set fcl
        if Icl <= 0.078:
            fcl = 1.00 + 1.290*Icl
        else:
            fcl = 1.05 + 0.645*Icl
            print 'set fcl ok'

        def findTcl(tcl):
            g = 2.38*abs(tcl-ta)**0.25
            h = 12.1*sqrt(var)
            hc = min(g,h)
            b = 35.7-0.028*MW-Icl*(3.96*(10**-8)*fcl*(((tcl+273)**4)- \
                ((mrt+273)**4))+fcl*hc*(tcl-ta))-tcl
            return b

        # determine Tcl from bisection search (brentq method)
        mrt = roomAveragesMRT[position]
        Tcl = brentq(findTcl, 0, 100)
        g = 2.38*abs(Tcl-ta)**0.25
```

```
            h = 12.1*sqrt(var)
            hc = max(g,h)

            # heat loss components by parts --------

            # heat loss diff through skin
            HL1 = 3.05*0.001*(5733-6.99*MW-pa)
            # heat loss by sweating
            if MW > 58.15:
                HL2 = 0.42*(MW-58.15)
            else:
                HL2 = 0
            # latent respiration heat loss
            HL3 = 1.7*0.00001*M*(5867-pa)
            # dry respiration heat loss
            HL4 = 0.0014*M*(34-ta)
            # heat loss by radiation
            HL5 = 3.96*0.00000001*fcl*(((Tcl+273)**4)-((mrt+273)**4))
            # heat loss by convection
            HL6 = fcl*hc*(Tcl-ta)
            # thermal sensation trans coeff
            TS = 0.303*exp(-0.036*M)+0.028
            # calc PMV
            PMV = TS*(MW-HL1-HL2-HL3-HL4-HL5-HL6)
            self.PMV_RoomAverageAnnual.append(PMV)
            PPD = 100-95*exp(-0.03353*PMV**4-0.2179*PMV**2)
            self.PPD_RoomAverageAnnual.append(PPD)
        print 'shape PMVavg', shape(self.PMV_RoomAverageAnnual)
        print 'calc PMV_RoomAverageAnnual ok'

    def calcAdaptiveASHRAE55(self):
        print 'Starting ASHRAE 55 Adaptive Temperature Calculation...'
```

```
        # pull outdoor dry bulb temps and month names from CSV
        csvOutdoorDryBulbs=[]
        csvMonths=[]
        for i in range(0,len(self.dates)):
            oDB = self.numbers[i,self.loc_oDB]
            csvOutdoorDryBulbs.append(oDB)
            csvMonths.append(self.dates[i][0:2])
        print "len csvOutdoorDryBulb", len(csvOutdoorDryBulbs)
        print "len csvMonths", len(csvMonths)

        # chunk data into months and days to find the monthly average
        # of the daily averages
        u = unique(csvMonths)
        chunks = []
        for i in range(0,len(u)-1):
            itemindex = csvMonths.index(u[i])
            print "itemindex", itemindex
            chunks.append(itemindex)
        chunks.append(len(csvMonths))
        meanMonthlyOutdoorDryBulbs = []
        for i in range(0,len(chunks)-1):
            x = (csvOutdoorDryBulbs[chunks[i]:chunks[i+1]])
            days = zip(*(iter(x),) * 24)
            dailyAverages = []
            for k in days:
                dailyAverages.append(mean(k))
            meanMonthlyOutdoorDryBulbs.append(mean(dailyAverages))

        # replace months with corresponding average dry bulb
        dictionary = dict(zip(u, meanMonthlyOutdoorDryBulbs))
        self.replacedODBforMean = [dictionary.get(x,x) for x in csvMonths]
        self.replacedODBforMean = asarray(self.replacedODBforMean[1:],
```

```
            dtype=np.float)
        print "len replacedODBforMean", len(self.replacedODBforMean)
        dummyRow = np.array([0])
        self.concatReplacedODBforMean = \
            concatenate((dummyRow,self.replacedODBforMean))
        print self.concatReplacedODBforMean
        print "len concatenate", len(self.concatReplacedODBforMean)

        # calculate Tcomf
        Tcomfs = []
        for i in range(0,len(self.concatReplacedODBforMean)):
           if self.concatReplacedODBforMean[i] < 10:
               Tcomf = 22
           else:
               Tcomf = 0.31*self.concatReplacedODBforMean[i]+17.8
           Tcomfs.append(Tcomf)
        print "len Tcomfs", len(Tcomfs)

        # find deltaT
        collectDeltaT = []
        for i in range(0, len(Tcomfs)):
            deltaT = subtract(self.OpTemp_Annual[i],Tcomfs[i])
            collectDeltaT.append(deltaT)
        self.AdaptiveASHRAE55_Annual = asarray(collectDeltaT)
        print 'Finished ASHRAE 55 Adaptive Temperature Calculation'


    def calcAdaptiveEN15251(self):
        print 'Starting EN15251 Adaptive Temperature Calculation...'
        # chunk outdoor dry bulb temps from CSV into 24 hour blocks
        x = self.numbers[1:,self.loc_oDB]
        print len(x)
```

```
        print x[0]
        days = zip(*(iter(x),) * 24)
        print len(days)
        print days[0]
        print len(days[0])
        # for each day, find average temps
        dailyAverages = []
        for i in days:
            dailyAverages.append(mean(i))
        print len(dailyAverages)
        print "DA zero",dailyAverages[0]
        # perform weighted running mean every 7 days
        collectTrm7 = []
        for i in range(0,len(dailyAverages)):
            T1m = dailyAverages[i-1]
            T2m = dailyAverages[i-2]
            T3m = dailyAverages[i-3]
            T4m = dailyAverages[i-4]
            T5m = dailyAverages[i-5]
            T6m = dailyAverages[i-6]
            T7m = dailyAverages[i-7]
            Trm7 = (T1m + 0.8*T2m + 0.6*T3m + 0.5*T4m + 0.4*T5m + \
                0.3*T6m + 0.2*T7m)/3.8
            collectTrm7.append(Trm7)
        self.Trm7s = repeat(collectTrm7, 24)
        print "LEN self.Trm7s", len(self.Trm7s)
        # calculate Tcomf
        Tcomfs = []
        for i in range(0,len(collectTrm7)):
           if collectTrm7[i] < 10:
              Tcomf = 22
           else:
```

```
               Tcomf = 0.33*collectTrm7[i]+18.8
           Tcomfs.append(Tcomf)
        print "len OLD Tcomfs", len(Tcomfs)
        # repeat 24 times to make it the same length as the OpTemps
        Tcomfs = repeat(Tcomfs, 24)
        print "len NEW Tcomfs", len(Tcomfs)
        dummyRow = array([0])
        Tcomfs = concatenate((dummyRow,Tcomfs))
        print "len NEW CONCAT Tcomfs", len(Tcomfs)

        # find deltaT
        collectDeltaT = []
        for i in range(0, len(Tcomfs)):
            deltaT = subtract(self.OpTemp_Annual[i],Tcomfs[i])
            collectDeltaT.append(deltaT)
        print 'Shape collect Delta T', shape(collectDeltaT)
        self.AdaptiveEN15251_Annual = asarray(collectDeltaT)
        print 'Finished EN15251 Adaptive Temperature Calculation'


    def calcAdaptiveNPRCR1752Beta(self):
        print 'Starting NPR-CR 1752 Adaptive Temperature Calculation...'
        # chunk outdoor dry bulb temps from CSV into 24 hour blocks
        x = self.numbers[1:,self.loc_oDB]
        print len(x)
        print x[0]
        days = zip(*(iter(x),) * 24)
        print len(days)
        print days[0]
        print len(days[0])
        # for each day, find average temps
        dailyAverageofMaxMin = []
```

```
        for i in days:
            mx = max(i)
            mn = min(i)
            dailyAverageofMaxMin.append(mean([mx, mn]))
        print len(dailyAverageofMaxMin)
        print "DA zero",dailyAverageofMaxMin[0]
        # perform weighted running mean every 7 days
        collectTrm3 = []
        for i in range(0,len(dailyAverageofMaxMin)):
            T1m = dailyAverageofMaxMin[i]
            T2m = dailyAverageofMaxMin[i-1]
            T3m = dailyAverageofMaxMin[i-2]
            T4m = dailyAverageofMaxMin[i-3]
            Trm3 = (T1m + 0.8*T2m + 0.4*T3m + 0.2*T4m)/2.4
            collectTrm3.append(Trm3)
        self.Trm3s = repeat(collectTrm3, 24)
        print "LEN self.Trm3s", len(self.Trm3s)
        # calculate Tcomf
        Tcomfs = []
        for i in range(0,len(collectTrm3)):
           if collectTrm3[i] < 10:
               Tcomf = 22
           else:
               Tcomf = 0.31*collectTrm3[i]+17.8
           Tcomfs.append(Tcomf)
        print "len OLD Tcomfs", len(Tcomfs)
        # repeat 24 times to make it the same length as the OpTemps
        Tcomfs = repeat(Tcomfs, 24)
        print "len NEW Tcomfs", len(Tcomfs)
        dummyRow = array([0])
        Tcomfs = concatenate((dummyRow,Tcomfs))
        print "len NEW CONCAT Tcomfs", len(Tcomfs)
```

```
        # find deltaT
        collectDeltaT = []
        for i in range(0, len(Tcomfs)):
            deltaT = subtract(self.OpTemp_Annual[i],Tcomfs[i])
            collectDeltaT.append(deltaT)
        print 'Shape collect Delta T', shape(collectDeltaT)
        self.AdaptiveNPRCR1752Beta_Annual = asarray(collectDeltaT)
        print 'Finished NPR-CR 1752 Adaptive Temperature Calculation'

    # Metric Indexing & Display Calculation Functions -----------------

    # indexing functions for single point in time ---------------------
    def setDate_Point(self):
        self.StartMonth = '%02d' % (int(self.StartMonthSpin.GetValue()))
        self.StartDay = '%02d' % (int(self.StartDaySpin.GetValue()))
        self.StartHour = '%02d' % (int(self.StartHourSpin.GetValue()))
        i = self.calcDrop.GetSelection()
        # note that for PMV and PPD, currently reference off MRTS
        calctype = [ self.MRTS_Annual,
                     self.OpTemp_Annual,
                     self.MRTS_Annual,
                     self.MRTS_Annual,
                     self.AdaptiveASHRAE55_Annual,
                     self.AdaptiveEN15251_Annual,
                     self.AdaptiveNPRCR1752Beta_Annual
                    ]
        calcselection = calctype[i]
        calcselectionOpTemps = calctype[1]
        self.outdoorDB = []
        for position, item in enumerate(self.dates):
            if (item[0:2] == self.StartMonth) and \
```

```
                (item[3:5] == self.StartDay) and \
                (item[7:9] == self.StartHour) :
                 self.position = position
                 print self.position
                 self.calcSelection_Point = calcselection[position,:,:,:]
                 self.calcSelection_Point_OpTemps = calcselectionOpTemps \
                     [position,:,:,:]
                 # comfort parameters
                 self.oDB = self.numbers[position,self.loc_oDB]
                 self.outdoorDB.append(self.oDB)
                 self.oRH = self.numbers[position,self.loc_oRH]
                 self.oAirV = self.numbers[position,self.loc_oAirV]
                 self.zDB = self.numbers[position,self.loc_zDB]
                 self.zMRT = self.numbers[position,self.loc_zMRT]
                 self.zOpT = self.numbers[position,self.loc_zOpT]
                 self.zRH = self.numbers[position,self.loc_zRH]
                 self.zActSch = self.numbers[position,self.loc_zActSch]
                 self.zCloSch = self.numbers[position,self.loc_zCloSch]
                 self.zAirVSch = self.numbers[position,self.loc_zAirVSch]
                 self.monthlyOutdoorMean = \
                     self.concatReplacedODBforMean[position]
                 self.runningMean7day = self.Trm7s[position]
                 self.runningMean3day = self.Trm3s[position]
        print 'SetDate_Point done'

    def indexMRTS_Point(self):
        # call indexing function
        self.setDate_Point()
        # define heatmap and scatter results
        self.heatmapCalcResults = self.calcSelection_Point
        self.scatterCalcResults = mean(self.calcSelection_Point)
        # display settings
```

```
        self.format = '%2.1f'
        self.xlabel = 'outdoor dry bulb temperature [C]'
        self.ylabel = 'indoor mean radiant temperature [C]'
        self.scattertitle = "Mean Radiant Temperature vs Outdoor Dry Bulb"
        self.scatterXlim = [-10,35]
        self.scatterYlim = [16,32]

    def indexOpTemp_Point(self):
        # call indexing function
        self.setDate_Point()
        # define heatmap and scatter results
        self.heatmapCalcResults = self.calcSelection_Point
        self.scatterCalcResults = mean(self.calcSelection_Point)
        # display settings
        self.format = '%2.1f'
        self.xlabel = 'outdoor dry bulb temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scattertitle = "Operative Temperature vs Outdoor Dry Bulb"
        self.scatterXlim = [-10,35]
        self.scatterYlim = [16,32]

    def calcPMV_Point(self):
        '''provides values for PMV heatmap for a single point
           in time
        '''
        self.setDate_Point()
        self.scatterCalcResultsPMV = self.PMV_RoomAverageAnnual[self.position]
        self.scatterCalcResultsPPD = self.PPD_RoomAverageAnnual[self.position]

        # set variables
        ta = self.zDB
        RH = self.zRH/100
```

```
        var = self.zAirVSch
        met = self.zActSch/58.15
        clo = self.zCloSch

        # auto calc variables
        Icl = clo*0.155
        M = met*58.15
        W = 0
        MW = M-W

        # steam table data - from 2.006 Property Data Tables
        steamTableTemps = [
        0.01, 5, 10, 15, 20, 25, 30, 35,
        40, 45, 50, 55, 60, 65, 70, 75,
        80, 85, 90, 95, 100
        ]
        steamTablePsat = [
        611.66, 872.58, 1228.2, 1705.8, 2339.3, 3169.9, 4247, 5629,
        7384.9, 9595, 12352, 15762, 19946, 25042, 31201, 38595,
        47414, 57867, 70182, 84608, 101420
        ]
        # calculate saturation pressure at ta
        Psat = interp(ta, steamTableTemps, steamTablePsat)
        # calculate water vapor partial pressure
        pa = RH*Psat

        # set fcl
        if Icl <= 0.078:
            fcl = 1.00 + 1.290*Icl
        else:
            fcl = 1.05 + 0.645*Icl

```

```
        def findTcl(tcl):
            g = 2.38*abs(tcl-ta)**0.25
            h = 12.1*sqrt(var)
            hc = min(g,h)
            b = 35.7-0.028*MW-Icl*(3.96*(10**-8)*fcl*(((tcl+273)**4)- \
                ((i+273)**4))+fcl*hc*(tcl-ta))-tcl
            return b

        # determine Tcl
        collectTcl = []
        collectHC = []
        for i in ravel(self.calcSelection_Point):
            Tcl = brentq(findTcl, 0, 100)
            collectTcl.append(Tcl)
            g = 2.38*abs(Tcl-ta)**0.25
            h = 12.1*sqrt(var)
            hc = max(g,h)
            collectHC.append(hc)

        # heat loss components by parts ------------------------------

        self.collectPMV = []
        self.collectPPD = []
        for i in range(0,len(ravel(self.calcSelection_Point))):
            tr = (ravel(self.calcSelection_Point))[i]
            Tcl = collectTcl[i]
            hc = collectHC[i]

            # heat loss diff through skin
            HL1 = 3.05*0.001*(5733-6.99*MW-pa)
            # heat loss by sweating
            if MW > 58.15:
```

```
                HL2 = 0.42*(MW-58.15)
            else:
                HL2 = 0
            # latent respiration heat loss
            HL3 = 1.7*0.00001*M*(5867-pa)
            # dry respiration heat loss
            HL4 = 0.0014*M*(34-ta)
            # heat loss by radiation
            HL5 = 3.96*0.00000001*fcl*(((Tcl+273)**4)-((tr+273)**4))
            # heat loss by convection
            HL6 = fcl*hc*(Tcl-ta)
            # thermal sensation trans coeff
            TS = 0.303*exp(-0.036*M)+0.028
            # calc PMV
            PMV = TS*(MW-HL1-HL2-HL3-HL4-HL5-HL6)
            self.collectPMV.append(PMV)
            PPD = 100-95*exp(-0.03353*PMV**4-0.2179*PMV**2)
            self.collectPPD.append(PPD)

        self.heatmapCalcResults = reshape(self.collectPMV,
            shape(self.calcSelection_Point))

        # display items
        r = around(self.heatmapCalcResults, decimals=2)
        u = unique(r)
        self.Tickvalues = u
        self.Tickvalues = [-3.0, -2.0, -1.0, -0.5, 0, 0.5, 1.0, 2.0, 3.0]
        self.Ticklabels = ['-3', '-2', '-1', '-0.5', '0', '+0.5',
            '+1','+2','+3']
        self.Levels = len(u)
        self.format = '%2.2f'
        self.scattertitle = \
```

```
            "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
        self.xlabel = 'predicted mean vote (PMV)'
        self.ylabel = 'predicted percentage of dissatisfied (PPD)'
        self.scatterXlim = [-3,3]
        self.scatterYlim = [0,100]

    def calcPPD_Point(self):
        '''provides values for PPD heatmap for a single point
           in time
        '''
        self.setDate_Point()
        self.calcPMV_Point()
        self.scatterCalcResultsPMV = \
            self.PMV_RoomAverageAnnual[self.position]
        self.scatterCalcResultsPPD = \
            self.PPD_RoomAverageAnnual[self.position]
        self.heatmapCalcResults = \
            reshape(self.collectPPD,shape(self.calcSelection_Point))

        # display items
        r = around(self.heatmapCalcResults, decimals=2)
        u = unique(r)
        self.Tickvalues = u
        self.Tickvalues = [0, 5, 10, 15, 20, 50]
        self.Ticklabels = ['0%', '5%', '10%', '15%', '20%', '>20%']
        self.Levels = len(u)
        self.format = '%2.2f'
        self.scattertitle = \
            "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
        self.xlabel = 'predicted mean vote (PMV)'
        self.ylabel = 'predicted percentage of dissatisfied (PPD)'
        self.scatterXlim = [-3,3]
```

```
        self.scatterYlim = [0,100]

    def indexASHRAE55Adaptive_Point(self):
        # call indexing function
        self.setDate_Point()
        # define heatmap and scatter results
        self.heatmapCalcResults = self.calcSelection_Point
        self.scatterCalcResults = mean(self.calcSelection_Point_OpTemps)
        # display settings
        r = around(self.heatmapCalcResults, decimals=1)
        u = unique(r)
        self.Tickvalues = [-10, -3.5, -2.5, 2.5, 3.5, 10]
        self.Ticklabels = [
            ’<80% acceptability’,
            ’80% acceptability \n $\Delta$ T -3.5 [C]’,
            ’90% acceptability \n $\Delta$ T -2.5 [C]’,
            ’90% acceptability \n $\Delta$ T +2.5 [C]’,
            ’80% acceptability \n $\Delta$ T +3.5 [C]’,
            ’< 80% acceptability’]
        self.Levels = len(u)
        self.format = ’%2.1f’
        # scatter plot lines
        self.scattertitle = \
            "ASHRAE Standard 55 Adaptive Comfort - Room Average"
        self.xlabel = ’mean monthly outdoor temperature [C]’
        self.ylabel = ’indoor operative temperature [C]’
        self.scatterXlim = [5,35]
        self.scatterYlim = [16,32]
        # scatter plot boundary lines
        self.outdoor = arange(10,33)
        self.comfortASHRAE = (0.31*self.outdoor+17.8)
        self.lower80ASHRAE = (0.31*self.outdoor+17.8)-3.5
```

```
        self.lower90ASHRAE = (0.31*self.outdoor+17.8)-2.5
        self.upper90ASHRAE = (0.31*self.outdoor+17.8)+2.5
        self.upper80ASHRAE = (0.31*self.outdoor+17.8)+3.5

    def indexEN15251Adaptive_Point(self):
        # call indexing function
        self.setDate_Point()
        # define heatmap and scatter results
        self.heatmapCalcResults = self.calcSelection_Point
        self.scatterCalcResults = mean(self.calcSelection_Point_OpTemps)
        # display settings
        r = around(self.heatmapCalcResults, decimals=1)
        u = unique(r)
        self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
        self.Ticklabels = [
            '<80% acceptability',
            '80% acceptability \n $\Delta$ T -3.0 [C]',
            '90% acceptability \n $\Delta$ T -2.0 [C]',
            '90% acceptability \n $\Delta$ T +2.0 [C]',
            '80% acceptability \n $\Delta$ T +3.0 [C]',
            '< 80% acceptability']
        self.Levels = len(u)
        self.format = '%2.1f'
        # scatter plot lines
        self.scattertitle = \
            "European Standard EN 15251 Adaptive Comfort - Room Average"
        self.xlabel = '7-day running mean outdoor temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scatterXlim = [5,35]
        self.scatterYlim = [16,32]
        # scatter plot boundary lines
        self.outdoor = arange(10,33)
```

```
        self.comfortEN15251 = (0.33*self.outdoor+18.8)
        self.lower80EN15251 = (0.33*self.outdoor+18.8)-3.0
        self.lower90EN15251 = (0.33*self.outdoor+18.8)-2.0
        self.upper90EN15251 = (0.33*self.outdoor+18.8)+2.0
        self.upper80EN15251 = (0.33*self.outdoor+18.8)+3.0

    def indexNPRCR1752BetaAdaptive_Point(self):
        # call indexing function
        self.setDate_Point()
        # define heatmap and scatter results
        self.heatmapCalcResults = self.calcSelection_Point
        self.scatterCalcResults = mean(self.calcSelection_Point_OpTemps)
        # display settings
        r = around(self.heatmapCalcResults, decimals=1)
        u = unique(r)
        self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
        self.Ticklabels = [
            '<80% acceptability',
            '80% acceptability \n $\Delta$ T -3.0 [C]',
            '90% acceptability \n $\Delta$ T -2.0 [C]',
            '90% acceptability \n $\Delta$ T +2.0 [C]',
            '80% acceptability \n $\Delta$ T +3.0 [C]',
            '< 80% acceptability']
        self.Levels = len(u)
        self.format = '%2.1f'
        # scatter plot lines
        self.scattertitle = \
            "Dutch Standard NPR-CR 1752 Type Beta Adaptive Comfort"
        self.xlabel = '3-day running mean outdoor temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scatterXlim = [5,35]
        self.scatterYlim = [16,32]
```

```
        # scatter plot boundary lines
        self.outdoor = arange(10,33)
        self.comfortNPRCR1752 = (0.31*self.outdoor+17.8)
        self.lower80NPRCR1752 = (0.31*self.outdoor+17.8)-3.0
        self.lower90NPRCR1752 = (0.31*self.outdoor+17.8)-2.0
        self.upper90NPRCR1752 = (0.31*self.outdoor+17.8)+2.0
        self.upper80NPRCR1752 = (0.31*self.outdoor+17.8)+3.0


    # indexing functions for date/time range --------------------------
    def setDate_Range(self):
        self.StartMonth = '%02d' % (int(self.StartMonthSpin.GetValue()))
        self.StartDay = '%02d' % (int(self.StartDaySpin.GetValue()))
        self.StartHour = '%02d' % (int(self.StartHourSpin.GetValue()))
        self.EndMonth = '%02d' % (int(self.EndMonthSpin.GetValue()))
        self.EndDay = '%02d' % (int(self.EndDaySpin.GetValue()))
        self.EndHour = '%02d' % (int(self.EndHourSpin.GetValue()))
        i = self.calcDrop.GetSelection()
        calctype = [ self.MRTS_Annual,
                     self.OpTemp_Annual,
                     self.MRTS_Annual,
                     self.MRTS_Annual,
                     self.AdaptiveASHRAE55_Annual,
                     self.AdaptiveEN15251_Annual,
                     self.AdaptiveNPRCR1752Beta_Annual
                    ]
        calcselection = calctype[i]
        calcselectionOpTemps = calctype[1]
        print 'CALCSELECTION', calcselection
        self.calcSelection_Range = []
        self.calcSelection_Range_OpTemps = []
        self.monthlyOutdoorMean = []
```

```
        ##
        self.positions = []
        self.Hr = []
        self.Mn = []
        self.outdoorDB = []
        self.zoneDB = []
        self.zoneRH = []
        self.zoneActSch = []
        self.zoneCloSch = []
        self.zoneAirVSch = []
        self.months = []
        self.runningMean7day = []
        self.runningMean3day = []
        for position, item in enumerate(self.dates):
            if (item[0:2] >= self.StartMonth) and \
                (item[0:2] <= self.EndMonth) and \
               (item[3:5] >= self.StartDay) and \
                (item[3:5] <= self.EndDay) and \
               (item[7:9] >= self.StartHour) and \
                (item[7:9] <= self.EndHour) :
                self.Hr.append(int(item[7:9]))
                self.Mn.append(int(item[0:2]))
                self.positions.append(position)
                self.calcSelection_Point = calcselection[position,:,:,:]
                self.calcSelection_Range.append(self.calcSelection_Point)
                self.calcSelection_Point_OpTemps = \
                    calcselectionOpTemps[position,:,:,:]
                self.calcSelection_Range_OpTemps.append \
                    (self.calcSelection_Point_OpTemps)
                monthlyOutdoorMean = self.concatReplacedODBforMean[position]
                self.monthlyOutdoorMean.append(monthlyOutdoorMean)
                self.runningMean7day.append(self.Trm7s[position])
```

```
                self.runningMean3day.append(self.Trm3s[position])
                # pull parameters
                self.oDB = self.numbers[position,self.loc_oDB]
                self.outdoorDB.append(self.oDB)
                self.zDB = self.numbers[position,self.loc_zDB]
                self.zoneDB.append(self.zDB)
                self.zRH = self.numbers[position,self.loc_zRH]
                self.zoneRH.append(self.zRH)
                self.zActSch = self.numbers[position,self.loc_zActSch]
                self.zoneActSch.append(self.zActSch)
                self.zCloSch = self.numbers[position,self.loc_zCloSch]
                self.zoneCloSch.append(self.zCloSch)
                self.zAirVSch = self.numbers[position,self.loc_zAirVSch]
                self.zoneAirVSch.append(self.zAirVSch)
                self.months.append(item[0:2])
        print 'SetDate_Range done'
        # bin data for scatters
        self.binStart = (int(self.colorStart.GetValue()))
        self.binEnd = (int(self.colorEnd.GetValue()))
        MBins = array([0,self.binStart,self.binEnd,13])
        HBins = array([0,self.binStart,self.binEnd,25])
        self.MonthBins = digitize(self.Mn, MBins)
        self.HourBins = digitize(self.Hr, HBins)

    def scatterResults(self):
        '''Bin scatter results into user-specified periods
        '''
        self.scatterCalcResults = []
        # for months
        self.scatterCalcResultsMonthBin1 = []
        self.MonthBin1 = []
        self.scatterCalcResultsMonthBin2 = []
```

```
        self.MonthBin2 = []
        self.scatterCalcResultsMonthBin3 = []
        self.MonthBin3 = []
        # for hours
        self.scatterCalcResultsHourBin1 = []
        self.HourBin1 = []
        self.scatterCalcResultsHourBin2 = []
        self.HourBin2 = []
        self.scatterCalcResultsHourBin3 = []
        self.HourBin3 = []
        i = self.calcDrop.GetSelection()
        if i >= 0 and i <= 1:
            variable1 = self.calcSelection_Range
            variable2 = self.outdoorDB
        if i == 4:
            variable1 = self.calcSelection_Range_OpTemps
            variable2 = self.monthlyOutdoorMean
        if i == 5:
            variable1 = self.calcSelection_Range_OpTemps
            variable2 = self.runningMean7day
        if i == 6:
            variable1 = self.calcSelection_Range_OpTemps
            variable2 = self.runningMean3day
        for i in range(0,len(variable1)):
            results = mean(variable1[i])
            self.scatterCalcResults.append(results)
            # for months
            if self.MonthBins[i]==1:
                self.scatterCalcResultsMonthBin1.append(results)
                self.MonthBin1.append(variable2[i])
            if self.MonthBins[i]==2:
                self.scatterCalcResultsMonthBin2.append(results)
```

```
                self.MonthBin2.append(variable2[i])
            if self.MonthBins[i]==3:
                self.scatterCalcResultsMonthBin3.append(results)
                self.MonthBin3.append(variable2[i])
            # for hours
            if self.HourBins[i]==1:
                self.scatterCalcResultsHourBin1.append(results)
                self.HourBin1.append(variable2[i])
            if self.HourBins[i]==2:
                self.scatterCalcResultsHourBin2.append(results)
                self.HourBin2.append(variable2[i])
            if self.HourBins[i]==3:
                self.scatterCalcResultsHourBin3.append(results)
                self.HourBin3.append(variable2[i])

    def indexMRTS_Range(self):
        # call indexing function
        self.setDate_Range()
        # define heatmap and scatter results
        self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
        self.scatterResults()
        # display settings
        self.format = '%2.1f'
        self.xlabel = 'outdoor dry bulb temperature [C]'
        self.ylabel = 'indoor mean radiant temperature [C]'
        self.scattertitle = "Mean Radiant Temperature vs Outdoor Dry Bulb"
        self.scatterXlim = [-10,35]
        self.scatterYlim = [16,32]

    def indexOpTemp_Range(self):
        # call indexing function
        self.setDate_Range()
```

```
        # define heatmap and scatter results
        self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
        self.scatterResults()
        # display settings
        self.format = '%2.1f'
        self.format = '%2.1f'
        self.xlabel = 'outdoor dry bulb temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scattertitle = "Operative Temperature vs Outdoor Dry Bulb"
        self.scatterXlim = [-10,35]
        self.scatterYlim = [16,32]

    def calcPMV_Range(self):
        '''provides values for PMV heatmap over a range of time
            note that this calculation can take a very long time
            because of the bisection search
        '''
        self.setDate_Range()

        self.scatterCalcResultsPMV = \
            self.PMV_RoomAverageAnnual[self.positions[0]:self.positions[-1]]
        self.scatterCalcResultsPPD = \
            self.PPD_RoomAverageAnnual[self.positions[0]:self.positions[-1]]

        self.collectPMVarrays = []
        self.collectPPDarrays = []
        for position, item in enumerate(self.positions):
            # set variables
            ta = self.zoneDB[position]
            RH = self.zoneRH[position]/100
            var = self.zoneAirVSch[position]
            met = self.zoneActSch[position]/58.15
```

```
            clo = self.zoneCloSch[position]

            # auto calc variables
            Icl = clo*0.155
            M = met*58.15
            W = 0
            MW = M-W
            print 'set variables ok'

            # steam table data - from 2.006 Property Data Tables
            steamTableTemps = [
            0.01, 5, 10, 15, 20, 25, 30, 35,
            40, 45, 50, 55, 60, 65, 70, 75,
            80, 85, 90, 95, 100
            ]
            steamTablePsat = [
            611.66, 872.58, 1228.2, 1705.8, 2339.3, 3169.9, 4247, 5629,
            7384.9, 9595, 12352, 15762, 19946, 25042, 31201, 38595,
            47414, 57867, 70182, 84608, 101420
            ]
            # calculate saturation pressure at ta
            Psat = interp(ta, steamTableTemps, steamTablePsat)
            # calculate water vapor partial pressure
            pa = RH*Psat
            print 'set steam tables ok'

            # set fcl
            if Icl <= 0.078:
                fcl = 1.00 + 1.290*Icl
            else:
                fcl = 1.05 + 0.645*Icl
            print 'set fcl ok'
```

```
    def findTcl(tcl):
        g = 2.38*abs(tcl-ta)**0.25
        h = 12.1*sqrt(var)
        hc = min(g,h)
        b = 35.7-0.028*MW-Icl*(3.96*(10**-8)*fcl*(((tcl+273)**4)- \
            ((i+273)**4))+fcl*hc*(tcl-ta))-tcl
        return b

    # determine Tcl by bisection search (brentq method)
    collectTcl = []
    collectHC = []
    for i in ravel(self.calcSelection_Range[position]):
        Tcl = brentq(findTcl, 0, 100)
        collectTcl.append(Tcl)
        g = 2.38*abs(Tcl-ta)**0.25
        h = 12.1*sqrt(var)
        hc = max(g,h)
        collectHC.append(hc)

    # heat loss components by parts ---------------------------

    self.collectPMV = []
    self.collectPPD = []
    for i in range(0,len(ravel(self.calcSelection_Range[position]))):
        tr = (ravel(self.calcSelection_Range))[i]
        Tcl = collectTcl[i]
        hc = collectHC[i]

        # heat loss diff through skin
        HL1 = 3.05*0.001*(5733-6.99*MW-pa)
        # heat loss by sweating
```

```
                if MW > 58.15:
                    HL2 = 0.42*(MW-58.15)
                else:
                    HL2 = 0
                # latent respiration heat loss
                HL3 = 1.7*0.00001*M*(5867-pa)
                # dry respiration heat loss
                HL4 = 0.0014*M*(34-ta)
                # heat loss by radiation
                HL5 = 3.96*0.00000001*fcl*(((Tcl+273)**4)-((tr+273)**4))
                # heat loss by convection
                HL6 = fcl*hc*(Tcl-ta)
                # thermal sensation trans coeff
                TS = 0.303*exp(-0.036*M)+0.028
                # calc PMV
                PMV = TS*(MW-HL1-HL2-HL3-HL4-HL5-HL6)
                self.collectPMV.append(PMV)
                PPD = 100-95*exp(-0.03353*PMV**4-0.2179*PMV**2)
                self.collectPPD.append(PPD)
            self.collectPMVarrays.append(self.collectPMV)
            self.collectPPDarrays.append(self.collectPPD)

        reshapeCollectArrays = \
            reshape(self.collectPMVarrays,shape(self.calcSelection_Range))
        self.heatmapCalcResults = mean(reshapeCollectArrays, axis=0)

        # display items
        r = around(self.heatmapCalcResults, decimals=2)
        u = unique(r)
        self.Tickvalues = u
        self.Tickvalues = [-3.0, -2.0, -1.0, -0.5, 0, 0.5, 1.0, 2.0, 3.0]
        self.Ticklabels = [’-3’, ’-2’, ’-1’, ’-0.5’, ’0’, \
```

```
            '+0.5', '+1','+2','+3']
        self.Levels = len(u)
        self.format = '%2.2f'
        self.scattertitle = \
            "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
        self.xlabel = 'predicted mean vote (PMV)'
        self.ylabel = 'predicted percentage of dissatisfied (PPD)'
        self.scatterlim = [-3,3]

    def calcPPD_Range(self):
        # provides values for PMV scatter plot for every hour of the year
        # PMV spatial calc done on-demand using 'Run cMap' button
        self.setDate_Range()
        self.calcPMV_Range()

        self.scatterCalcResultsPMV = \
            self.PMV_RoomAverageAnnual[self.positions[0]:self.positions[-1]]
        self.scatterCalcResultsPPD = \
            self.PPD_RoomAverageAnnual[self.positions[0]:self.positions[-1]]

        reshapeCollectArrays = \
            reshape(self.collectPPDarrays,shape(self.calcSelection_Range))
        self.heatmapCalcResults = mean(reshapeCollectArrays, axis=0)

        # display items
        r = around(self.heatmapCalcResults, decimals=2)
        u = unique(r)
        self.Tickvalues = u
        self.Tickvalues = [0, 5, 10, 15, 20, 50]
        self.Ticklabels = ['0%', '5%', '10%', '15%', '20%', '>20%']
        self.Levels = len(u)
#         self.cmap = self.cmap5Step
```

```
        self.format = '%2.2f'
        self.scattertitle = \
            "ASHRAE Standard 55 PPD as a function of PMV - Room Average"
        self.xlabel = 'predicted mean vote (PMV)'
        self.ylabel = 'predicted percentage of dissatisfied (PPD)'
        self.scatterlim = [-3,3]

    def indexASHRAE55Adaptive_Range(self):
        # call indexing function
        self.setDate_Range()
        # define heatmap and scatter results
        self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
        self.scatterResults()
        # display settings
        r = around(self.heatmapCalcResults, decimals=1)
        u = unique(r)
        self.Tickvalues = [-10, -3.5, -2.5, 2.5, 3.5, 10]
        self.Ticklabels = [
            '<80% acceptability',
            '80% acceptability \n $\Delta$ T -3.5 [C]',
            '90% acceptability \n $\Delta$ T -2.5 [C]',
            '90% acceptability \n $\Delta$ T +2.5 [C]',
            '80% acceptability \n $\Delta$ T +3.5 [C]',
            '< 80% acceptability']
        self.Levels = len(u)
        self.format = '%2.1f'
        # scatter plot lines
        self.scattertitle = \
            "ASHRAE Standard 55 Adaptive Comfort - Room Average"
        self.xlabel = 'mean monthly outdoor temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scatterXlim = [5,35]
```

```
        self.scatterYlim = [16,32]
        # scatter plot boundary lines
        self.outdoor = arange(10,33)
        self.comfortASHRAE = (0.31*self.outdoor+17.8)
        self.lower80ASHRAE = (0.31*self.outdoor+17.8)-3.5
        self.lower90ASHRAE = (0.31*self.outdoor+17.8)-2.5
        self.upper90ASHRAE = (0.31*self.outdoor+17.8)+2.5
        self.upper80ASHRAE = (0.31*self.outdoor+17.8)+3.5


    def indexEN15251Adaptive_Range(self):
        # call indexing function
        self.setDate_Range()
        # define heatmap and scatter results
        self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
        self.scatterResults()
        # display settings
        r = around(self.heatmapCalcResults, decimals=1)
        u = unique(r)
        self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
        self.Ticklabels = [
            '<80% acceptability',
            '80% acceptability \n $\Delta$ T -3.0 [C]',
            '90% acceptability \n $\Delta$ T -2.0 [C]',
            '90% acceptability \n $\Delta$ T +2.0 [C]',
            '80% acceptability \n $\Delta$ T +3.0 [C]',
            '< 80% acceptability']
        self.Levels = len(u)
        self.format = '%2.1f'
        # scatter plot lines
        self.scattertitle = \
            "European Standard EN 15251 Adaptive Comfort - Room Average"
```

```
        self.xlabel = '7-day running mean outdoor temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scatterXlim = [5,35]
        self.scatterYlim = [16,32]
        # scatter plot boundary lines
        self.outdoor = arange(10,33)
        self.comfortEN15251 = (0.33*self.outdoor+18.8)
        self.lower80EN15251 = (0.33*self.outdoor+18.8)-3.0
        self.lower90EN15251 = (0.33*self.outdoor+18.8)-2.0
        self.upper90EN15251 = (0.33*self.outdoor+18.8)+2.0
        self.upper80EN15251 = (0.33*self.outdoor+18.8)+3.0


    def indexNPRCR1752BetaAdaptive_Range(self):
        # call indexing function
        self.setDate_Range()
        # define heatmap and scatter results
        self.heatmapCalcResults = mean(self.calcSelection_Range, axis=0)
        self.scatterResults()
        # display settings
        r = around(self.heatmapCalcResults, decimals=1)
        u = unique(r)
        self.Tickvalues = [-10, -3.0, -2.0, 2.0, 3.0, 10]
        self.Ticklabels = [
            '<80% acceptability',
            '80% acceptability \n $\Delta$ T -3.0 [C]',
            '90% acceptability \n $\Delta$ T -2.0 [C]',
            '90% acceptability \n $\Delta$ T +2.0 [C]',
            '80% acceptability \n $\Delta$ T +3.0 [C]',
            '< 80% acceptability']
        self.Levels = len(u)
        self.format = '%2.1f'
```

```
        # scatter plot lines
        self.scattertitle = \
            "Dutch Standard NPR-CR 1752 Type Beta Adaptive Comfort"
        self.xlabel = '3-day running mean outdoor temperature [C]'
        self.ylabel = 'indoor operative temperature [C]'
        self.scatterXlim = [5,35]
        self.scatterYlim = [16,32]
        # scatter plot boundary lines
        self.outdoor = arange(10,33)
        self.comfortNPRCR1752 = (0.31*self.outdoor+17.8)
        self.lower80NPRCR1752 = (0.31*self.outdoor+17.8)-3.0
        self.lower90NPRCR1752 = (0.31*self.outdoor+17.8)-2.0
        self.upper90NPRCR1752 = (0.31*self.outdoor+17.8)+2.0
        self.upper80NPRCR1752 = (0.31*self.outdoor+17.8)+3.0

    # Control Functions -----------------------------------------------

    # run controls
    def loadFiles(self, event):
        self.idf = self.idfSel.GetValue()
        self.csv = self.csvSel.GetValue()
        # call calc functions
        self.readIDF()
        self.getTemps()
        self.setDate_Annual()
        self.viewFactors()
        # call annual metric calc functions
        self.calcMRTS()
        self.calcOpTemp()
        self.calcPMVPPDRoomAverageAnnual()
        self.calcAdaptiveASHRAE55()
        self.calcAdaptiveEN15251()
```

```
        self.calcAdaptiveNPRCR1752Beta()

    def runCmap(self, event):
        self.GetCalcType()
#        self.comfParamsSet()
        if self.noColor.GetValue() == True:
            self.makeScatter()
        if self.monthColor.GetValue() == True:
            self.makeScatterMonth()
        if self.timeColor.GetValue() == True:
            self.makeScatterHour()
        self.makePlot()

    def runUpdate(self, event):
        self.GetCalcType()
#        self.comfParamsSet()
        if self.noColor.GetValue() == True:
            self.makeScatter()
        if self.monthColor.GetValue() == True:
            self.makeScatterMonth()
        if self.timeColor.GetValue() == True:
            self.makeScatterHour()
        self.updatePlot()

    def GetCalcType(self):
        i = self.calcDrop.GetSelection()
        j = self.timeframeDrop.GetSelection()
        # point in time calculation functions
        if i == 0 and j == 1:
            self.indexMRTS_Point()
        if i == 1 and j == 1:
            self.indexOpTemp_Point()
```

```
        if i == 2 and j == 1:
            self.calcPMV_Point()
        if i == 3 and j == 1:
            self.calcPPD_Point()
        if i == 4 and j == 1:
            self.indexASHRAE55Adaptive_Point()
        if i == 5 and j == 1:
            self.indexEN15251Adaptive_Point()
        if i == 6 and j == 1:
            self.indexNPRCR1752BetaAdaptive_Point()
        # range calculation functions
        if i == 0 and j == 0:
            self.indexMRTS_Range()
        if i == 1 and j == 0:
            self.indexOpTemp_Range()
        if i == 2 and j == 0:
            self.calcPMV_Range()
        if i == 3 and j == 0:
            self.calcPPD_Range()
        if i == 4 and j == 0:
            self.indexASHRAE55Adaptive_Range()
        if i == 5 and j == 0:
            self.indexEN15251Adaptive_Range()
        if i == 6 and j == 0:
            self.indexNPRCR1752BetaAdaptive_Range()

    def comfParamsSet(self):
        # get values
        self.oDBSpin.SetLabel((str(around(self.oDB,decimals=1))). \
            strip('[').strip(']'))
        self.oRHSpin.SetLabel((str(around(self.oRH,decimals=0))). \
            strip('[').strip(']').strip('.'))
```

```
        self.oAirVSpin.SetLabel((str(around(self.oAirV,decimals=1))). \
            strip('[').strip(']'))
        self.zDBSpin.SetLabel((str(around(self.zDB,decimals=1))). \
            strip('[').strip(']'))
        self.zRHSpin.SetLabel((str(around(self.zRH,decimals=0))). \
            strip('[').strip(']').strip('.'))
        self.zAirVSchSpin.SetLabel((str(around(self.zAirVSch,decimals=1))). \
            strip('[').strip(']'))
        self.zCloSchSpin.SetLabel((str(around(self.zCloSch,decimals=1))). \
            strip('[').strip(']'))
        self.zActSchSpin.SetLabel((str(around(self.zActSch/58.2,decimals=1))). \
            strip('[').strip(']'))

    # scale controls
    def scaleSet(self):
        # get min & max to clip heatmaps
        i = self.calcDrop.GetSelection()
        if i >= 0 and i <= 1:
            r = around(self.heatmapCalcResults, decimals=1)
            u = unique(r)
            self.Tickvalues = around(linspace(min(u), max(u), num=10,
                endpoint=True), decimals=1)
            self.Ticklabels = self.Tickvalues
            self.Levels = self.Tickvalues
            self.Ticks = self.Tickvalues
            self.Norm = mpl.colors.Normalize(vmin = min(self.Tickvalues),
                vmax = max(self.Tickvalues), clip = False)
            self.cmapMaxDisplay.SetValue(str(max(self.Tickvalues)))
            self.cmapMinDisplay.SetValue(str(min(self.Tickvalues)))
        else:
            self.Ticks = self.Tickvalues
            self.Norm = mpl.colors.Normalize(vmin = min(self.Tickvalues),
```

```
                vmax = max(self.Tickvalues), clip = False)

    def OnAdjustScale(self, event):
        clipMax = float(self.cmapMaxDisplay.GetValue())
        clipMin = float(self.cmapMinDisplay.GetValue())
        self.Tickvalues = around(linspace(clipMin, clipMax, num=10,
            endpoint=True), decimals=1)
        self.Ticklabels = self.Tickvalues
        self.Levels = self.Tickvalues
        self.Ticks = self.Tickvalues
        self.Norm = mpl.colors.Normalize(vmin = min(self.Tickvalues),
            vmax = max(self.Tickvalues), clip = False)
        self.updatePlot()

    def OnAutoScale(self, event):
        self.scaleSet()
        self.updatePlot()

    # cut controls
    def OnCheck(self, event):
        self.v = event.GetEventObject().GetLabel()
        cn = { 'X-Y' : '0', 'Y-Z' : '1', 'X-Z' : '2', }
        self.cutNum = int(cn[self.v])
        self.axH = self.v[0]
        self.axV = self.v[2]
        self.updatePlot()

    # grid controls
    def OnGrid(self, event):
        self.gpt = self.gridSpin.GetValue()
        self.sliceSpin.SetRange(0,self.gpt-1)
        self.readIDF()
```

```
        self.getTemps()
        self.GetCalcType()
        self.updatePlot()

    def DisableAnnual(self,event):
        a = self.StartMonthSpin.GetValue()
        b = self.StartDaySpin.GetValue()
        c = self.StartHourSpin.GetValue()
        z = self.StartHourSpin.GetValue()+1
        d = self.EndMonthSpin.GetValue()
        e = self.EndDaySpin.GetValue()
        f = self.EndHourSpin.GetValue()
        j = self.timeframeDrop.GetSelection()
        if j == 1:
            self.EndMonthSpin.Hide()
            self.EndDaySpin.Hide()
            self.EndHourSpin.Hide()
            self.enTxt.Hide()
            self.noColor.SetValue(True)
            self.monthColor.Disable()
            self.timeColor.Disable()
        else:
            self.EndMonthSpin.Show()
            self.EndDaySpin.Show()
            self.EndHourSpin.Show()
            self.enTxt.Show()
            self.monthColor.Enable()
            self.timeColor.Enable()

    # slice controls
    def OnSlice(self, event):
        self.ind = self.sliceSpin.GetValue()
```

```
        self.updatePlot()

    # scatter plot controls
    def makeScatter(self):
        print 'Making scatter plot...'
        self.scatterFig.clf()
        self.scatteraxes = self.scatterFig.add_subplot(1,1,1)
        self.scatterFig.subplots_adjust(top=0.875)
        self.scatterFig.subplots_adjust(bottom=0.2)
        self.scatterFig.subplots_adjust(left=0.1)
        self.scatterFig.subplots_adjust(right=0.9)
        self.scatteraxes.set_ylim(self.scatterYlim)
        self.scatteraxes.set_xlim(self.scatterXlim)
        i = self.calcDrop.GetSelection()
        j = self.timeframeDrop.GetSelection()
        if i >= 0 and i <= 1:
            self.scatteraxes.scatter(self.outdoorDB,self.scatterCalcResults,
                color=self.facecolorBlue,edgecolor=self.edgecolor,
                lw = self.lw,alpha=self.alpha,s=self.scattersize)
        if i >= 2 and i <= 3:
            self.curvePMVS = arange(-3,3.1,0.1)
            self.curvePPDS = \
                100-95*exp(-0.03353*self.curvePMVS**4-0.2179*self.curvePMVS**2)
            self.scatteraxes.plot(self.curvePMVS,self.curvePPDS, 'k')
            self.scatteraxes.plot(self.scatterCalcResultsPMV,
                self.scatterCalcResultsPPD,'o',
                color=self.facecolorBlue,markersize=5)
        if i == 4:
            d = Line2D(self.outdoor,self.lower80ASHRAE, color='black')
            e = Line2D(self.outdoor,self.lower90ASHRAE, color='black',
                linestyle = '--')
            f = Line2D(self.outdoor,self.comfortASHRAE, color='black',
```

```
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90ASHRAE, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80ASHRAE, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            self.scatteraxes.scatter(self.monthlyOutdoorMean,
                self.scatterCalcResults,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
                'Comfort Temperature')
            legendseries = (d,e,f)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=3, prop=prop).draw_frame(False)
        if i == 5:
            d = Line2D(self.outdoor,self.lower80EN15251, color='black')
            e = Line2D(self.outdoor,self.lower90EN15251, color='black',
                linestyle = '--')
            f = Line2D(self.outdoor,self.comfortEN15251, color='black',
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90EN15251, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80EN15251, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
```

```
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            self.scatteraxes.scatter(self.runningMean7day,
                self.scatterCalcResults,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
                'Comfort Temperature')
            legendseries = (d,e,f)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
            ncol=3, prop=prop).draw_frame(False)
        if i == 6:
            d = Line2D(self.outdoor,self.lower80NPRCR1752, color='black')
            e = Line2D(self.outdoor,self.lower90NPRCR1752, color='black',
                linestyle = '--')
            f = Line2D(self.outdoor,self.comfortNPRCR1752, color='black',
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90NPRCR1752, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80NPRCR1752, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            self.scatteraxes.scatter(self.runningMean3day,
                self.scatterCalcResults,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
```

```
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
                'Comfort Temperature')
            legendseries = (d,e,f)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=3, prop=prop).draw_frame(False)
        self.scatteraxes.grid(True,color=self.gridcolor)
        self.scatteraxes.spines['bottom'].set_color(self.axescolor)
        self.scatteraxes.spines['top'].set_color(self.axescolor)
        self.scatteraxes.spines['right'].set_color(self.axescolor)
        self.scatteraxes.spines['left'].set_color(self.axescolor)
        self.scatteraxes.set_title(self.scattertitle,fontsize=12)
        self.scatteraxes.set_xlabel(self.xlabel, fontsize=8)
        self.scatteraxes.set_ylabel(self.ylabel, fontsize=8)
        # update the font size of the x and y axes
        for tick in self.scatteraxes.xaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        for tick in self.scatteraxes.yaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        # send resize event to refresh panel
        pix = tuple( self.panel3.GetClientSize() )
        set = (pix[0]*1.01, pix[1]*1.01)
        self.scatterChart.SetClientSize( set )
        self.scatterCanvas.SetClientSize( set )
        print 'Finished making scatter plot'

    # scatter plot controls
    def makeScatterMonth(self):
        print 'Making scatter plot...'
```

```
        self.scatterFig.clf()
        self.scatteraxes = self.scatterFig.add_subplot(1,1,1)
        self.scatterFig.subplots_adjust(top=0.875)
        self.scatterFig.subplots_adjust(bottom=0.2)
        self.scatterFig.subplots_adjust(left=0.1)
        self.scatterFig.subplots_adjust(right=0.9)
        self.scatteraxes.set_ylim(self.scatterYlim)
        self.scatteraxes.set_xlim(self.scatterXlim)
        i = self.calcDrop.GetSelection()
        j = self.timeframeDrop.GetSelection()
        if i >= 0 and i <= 1:
            a = self.scatteraxes.scatter(self.MonthBin1,
                self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            b = self.scatteraxes.scatter(self.MonthBin2,
                self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            c = self.scatteraxes.scatter(self.MonthBin3,
                self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('Simulated Hours','Highlighted Range')
            legendseries = (a,b)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=2, prop=prop).draw_frame(False)
        # need PMV and PPD
        if i == 4:
```

```
        d = Line2D(self.outdoor,self.lower80ASHRAE, color='black')
        e = Line2D(self.outdoor,self.lower90ASHRAE, color='black',
            linestyle = '--')
        f = Line2D(self.outdoor,self.comfortASHRAE, color='black',
            linestyle = ':')
        g = Line2D(self.outdoor,self.upper90ASHRAE, color='black',
            linestyle = '--')
        h = Line2D(self.outdoor,self.upper80ASHRAE, color='black')
        self.scatteraxes.add_line(d)
        self.scatteraxes.add_line(e)
        self.scatteraxes.add_line(f)
        self.scatteraxes.add_line(g)
        self.scatteraxes.add_line(h)
        a = self.scatteraxes.scatter(self.MonthBin1,
            self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
            edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
            s=self.scattersize)
        b = self.scatteraxes.scatter(self.MonthBin2,
            self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
            edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
            s=self.scattersize)
        c = self.scatteraxes.scatter(self.MonthBin3,
            self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
            edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
            s=self.scattersize)
        # make legend and labels
        prop = fm.FontProperties(size=8)
        legendlabels = ('80% Acceptability','90% Acceptability',
            'Comfort Temperature','Simulated Hours','Highlighted Range')
        legendseries = (d,e,f,a,b)
        self.scatteraxes.legend(legendseries, legendlabels,
            'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
```

```
                ncol=5, prop=prop).draw_frame(False)
        if i == 5:
            d = Line2D(self.outdoor,self.lower80EN15251, color='black')
            e = Line2D(self.outdoor,self.lower90EN15251, color='black',
                linestyle = '--')
            f = Line2D(self.outdoor,self.comfortEN15251, color='black',
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90EN15251, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80EN15251, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            a = self.scatteraxes.scatter(self.MonthBin1,
                self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            b = self.scatteraxes.scatter(self.MonthBin2,
                self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            c = self.scatteraxes.scatter(self.MonthBin3,
                self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
                'Comfort Temperature','Simulated Hours','Highlighted Range')
            legendseries = (d,e,f,a,b)
```

```
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=5, prop=prop).draw_frame(False)
        if i == 6:
            d = Line2D(self.outdoor,self.lower80NPRCR1752, color='black')
            e = Line2D(self.outdoor,self.lower90NPRCR1752, color='black',
                linestyle = '--')
            f = Line2D(self.outdoor,self.comfortNPRCR1752, color='black',
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90NPRCR1752, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80NPRCR1752, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            a = self.scatteraxes.scatter(self.MonthBin1,
                self.scatterCalcResultsMonthBin1,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            b = self.scatteraxes.scatter(self.MonthBin2,
                self.scatterCalcResultsMonthBin2,color=self.facecolorRed,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            c = self.scatteraxes.scatter(self.MonthBin3,
                self.scatterCalcResultsMonthBin3,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
```

```
                'Comfort Temperature','Simulated Hours','Highlighted Range')
            legendseries = (d,e,f,a,b)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=5, prop=prop).draw_frame(False)
        self.scatteraxes.set_title(self.scattertitle,fontsize=12)
        self.scatteraxes.set_xlabel(self.xlabel, fontsize=8)
        self.scatteraxes.set_ylabel(self.ylabel, fontsize=8)
        self.scatteraxes.grid(True,color=self.gridcolor)
        self.scatteraxes.spines['bottom'].set_color(self.axescolor)
        self.scatteraxes.spines['top'].set_color(self.axescolor)
        self.scatteraxes.spines['right'].set_color(self.axescolor)
        self.scatteraxes.spines['left'].set_color(self.axescolor)
        # update the font size of the x and y axes
        for tick in self.scatteraxes.xaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        for tick in self.scatteraxes.yaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        # send resize event to refresh panel
        pix = tuple( self.panel3.GetClientSize() )
        print "PIX", pix
        set = (pix[0]*1.01, pix[1]*1.01)
        self.scatterChart.SetClientSize( set )
        self.scatterCanvas.SetClientSize( set )
        print 'Finished making scatter plot'

    # scatter plot controls
    def makeScatterHour(self):
        print 'Making scatter plot...'
        self.scatterFig.clf()
        self.scatteraxes = self.scatterFig.add_subplot(1,1,1)
        self.scatterFig.subplots_adjust(top=0.875)
```

```
        self.scatterFig.subplots_adjust(bottom=0.2)
        self.scatterFig.subplots_adjust(left=0.1)
        self.scatterFig.subplots_adjust(right=0.9)
        self.scatteraxes.set_ylim(self.scatterYlim)
        self.scatteraxes.set_xlim(self.scatterXlim)
        i = self.calcDrop.GetSelection()
        j = self.timeframeDrop.GetSelection()
        if i >= 0 and i <= 1:
            a = self.scatteraxes.scatter(self.HourBin1,
                self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            b = self.scatteraxes.scatter(self.HourBin2,
                self.scatterCalcResultsHourBin2,color=self.facecolorRed,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            c = self.scatteraxes.scatter(self.HourBin3,
                self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('Simulated Hours','Highlighted Range')
            legendseries = (a,b)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=2, prop=prop).draw_frame(False)
        # need PMV and PPD
        if i == 4:
            d = Line2D(self.outdoor,self.lower80ASHRAE, color='black')
            e = Line2D(self.outdoor,self.lower90ASHRAE, color='black',
                linestyle = '--')
```

```
            f = Line2D(self.outdoor,self.comfortASHRAE, color='black',
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90ASHRAE, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80ASHRAE, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            a = self.scatteraxes.scatter(self.HourBin1,
                self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            b = self.scatteraxes.scatter(self.HourBin2,
                self.scatterCalcResultsHourBin2,color=self.facecolorRed,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            c = self.scatteraxes.scatter(self.HourBin3,
                self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
                'Comfort Temperature','Simulated Hours','Highlighted Range')
            legendseries = (d,e,f,a,b)
            self.scatteraxes.legend(legendseries, legendlabels,
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=5, prop=prop).draw_frame(False)
        if i == 5:
            d = Line2D(self.outdoor,self.lower80EN15251, color='black')
```

```
        e = Line2D(self.outdoor,self.lower90EN15251, color='black',
            linestyle = '--')
        f = Line2D(self.outdoor,self.comfortEN15251, color='black',
            linestyle = ':')
        g = Line2D(self.outdoor,self.upper90EN15251, color='black',
            linestyle = '--')
        h = Line2D(self.outdoor,self.upper80EN15251, color='black')
        self.scatteraxes.add_line(d)
        self.scatteraxes.add_line(e)
        self.scatteraxes.add_line(f)
        self.scatteraxes.add_line(g)
        self.scatteraxes.add_line(h)
        a = self.scatteraxes.scatter(self.HourBin1,
            self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
            edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
            s=self.scattersize)
        b = self.scatteraxes.scatter(self.HourBin2,
            self.scatterCalcResultsHourBin2,color=self.facecolorRed,
            edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
            s=self.scattersize)
        c = self.scatteraxes.scatter(self.HourBin3,
            self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
            edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
            s=self.scattersize)
        # make legend and labels
        prop = fm.FontProperties(size=8)
        legendlabels = ('80% Acceptability','90% Acceptability',
            'Comfort Temperature','Simulated Hours','Highlighted Range')
        legendseries = (d,e,f,a,b)
        self.scatteraxes.legend(legendseries, legendlabels,
            'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
            ncol=5, prop=prop).draw_frame(False)
```

```
        if i == 6:
            d = Line2D(self.outdoor,self.lower80NPRCR1752, color='black')
            e = Line2D(self.outdoor,self.lower90NPRCR1752, color='black',
                linestyle = '--')
            f = Line2D(self.outdoor,self.comfortNPRCR1752, color='black',
                linestyle = ':')
            g = Line2D(self.outdoor,self.upper90NPRCR1752, color='black',
                linestyle = '--')
            h = Line2D(self.outdoor,self.upper80NPRCR1752, color='black')
            self.scatteraxes.add_line(d)
            self.scatteraxes.add_line(e)
            self.scatteraxes.add_line(f)
            self.scatteraxes.add_line(g)
            self.scatteraxes.add_line(h)
            a = self.scatteraxes.scatter(self.HourBin1,
                self.scatterCalcResultsHourBin1,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            b = self.scatteraxes.scatter(self.HourBin2,
                self.scatterCalcResultsHourBin2,color=self.facecolorRed,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            c = self.scatteraxes.scatter(self.HourBin3,
                self.scatterCalcResultsHourBin3,color=self.facecolorBlue,
                edgecolor=self.edgecolor,lw = self.lw,alpha=self.alpha,
                s=self.scattersize)
            # make legend and labels
            prop = fm.FontProperties(size=8)
            legendlabels = ('80% Acceptability','90% Acceptability',
                'Comfort Temperature','Simulated Hours','Highlighted Range')
            legendseries = (d,e,f,a,b)
            self.scatteraxes.legend(legendseries, legendlabels,
```

```
                'upper center', scatterpoints=1, bbox_to_anchor=(0.5, -0.085),
                ncol=5, prop=prop).draw_frame(False)
        self.scatteraxes.set_title(self.scattertitle,fontsize=12)
        self.scatteraxes.set_xlabel(self.xlabel, fontsize=8)
        self.scatteraxes.set_ylabel(self.ylabel, fontsize=8)
        self.scatteraxes.grid(True,color=self.gridcolor)
        self.scatteraxes.spines['bottom'].set_color(self.axescolor)
        self.scatteraxes.spines['top'].set_color(self.axescolor)
        self.scatteraxes.spines['right'].set_color(self.axescolor)
        self.scatteraxes.spines['left'].set_color(self.axescolor)
        # update the font size of the x and y axes
        for tick in self.scatteraxes.xaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        for tick in self.scatteraxes.yaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        # send resize event to refresh panel
        pix = tuple( self.panel3.GetClientSize() )
        print "PIX", pix
        set = (pix[0]*1.01, pix[1]*1.01)
        self.scatterChart.SetClientSize( set )
        self.scatterCanvas.SetClientSize( set )
        print 'Finished making scatter plot'

    # plot controls
    def makePlot(self):
        print 'Making heatmap plot...'
        self.scaleSet()
        self.fig.clf()
        self.axes = self.fig.add_subplot(1,1,1)
        self.fig.subplots_adjust(top=0.95)
        self.fig.subplots_adjust(bottom=0.225)
        self.fig.subplots_adjust(left=0.1)
```

```
        self.graphCut1 = [self.pt[0,:,:,0], self.pt[1,0,:,:], self.pt[0,:,0,:]]
        self.graphCut2 = [self.pt[1,:,:,0], self.pt[2,0,:,:], self.pt[2,:,0,:]]
        self.graphCut3 = [
            self.heatmapCalcResults[:,:,self.ind],
            self.heatmapCalcResults[self.ind,:,:],
            self.heatmapCalcResults[:,self.ind,:]]
        self.t = self.axes.contourf \
            (self.graphCut1[self.cutNum],
            self.graphCut2[self.cutNum],
            self.graphCut3[self.cutNum],
            15,alpha=1, cmap=cm.get_cmap(self.cmap), norm = self.Norm)
        self.s = self.axes.contour \
            (self.graphCut1[self.cutNum],
            self.graphCut2[self.cutNum],
            self.graphCut3[self.cutNum],
            15, linewidths=0.25, colors='k', norm = self.Norm)
        pyplot.clabel(self.s, fmt = self.format, colors = '0.15', fontsize=9)
        self.axes.set_xlabel('%s distance [m]'%(self.axH), fontsize=8)
        self.axes.set_ylabel('%s distance [m]'%(self.axV), fontsize=8)
        # update the font size of the x and y axes
        for tick in self.axes.xaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        for tick in self.axes.yaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        # left,bottom,width,height
        cax = self.fig.add_axes([0.1, 0.1, 0.8, 0.02])
        cb = mpl.colorbar.ColorbarBase \
            (cax, cmap=cm.get_cmap(self.cmap), norm=self.Norm,
            orientation = 'horizontal', boundaries = self.Ticks,
            format = '%2.1f')
        cb.set_label((self.calcList[self.calcDrop.GetCurrentSelection()]),
            fontsize=7)
```

```
        cb.ax.set_xticklabels(self.Ticklabels)
        for t in cb.ax.get_xticklabels():
            t.set_fontsize(7)
        # send resize event to refresh panel
        pix = tuple( self.panel2.GetClientSize() )
        set = (pix[0]*1.01, pix[1]*1.01)
        self.chart.SetClientSize( set )
        self.canvas.SetClientSize( set )
        self.makeKey()
        print 'Finished making heatmap plot'

    def updatePlot(self):
        self.fig.clf()
        self.axes = self.fig.add_subplot(1,1,1)
        self.fig.subplots_adjust(top=0.95)
        self.fig.subplots_adjust(bottom=0.225)
        self.fig.subplots_adjust(left=0.1)
        self.graphCut1 = [self.pt[0,:,:,0], self.pt[1,0,:,:], self.pt[0,:,0,:]]
        self.graphCut2 = [self.pt[1,:,:,0], self.pt[2,0,:,:], self.pt[2,:,0,:]]
        self.graphCut3 = [
            self.heatmapCalcResults[:,:,self.ind],
            self.heatmapCalcResults[self.ind,:,:],
            self.heatmapCalcResults[:,self.ind,:]]
        self.t = self.axes.contourf \
            (self.graphCut1[self.cutNum],
            self.graphCut2[self.cutNum],
            self.graphCut3[self.cutNum],
            15, alpha=1, cmap=cm.get_cmap(self.cmap), norm = self.Norm)
        self.s = self.axes.contour \
            (self.graphCut1[self.cutNum],
            self.graphCut2[self.cutNum],
            self.graphCut3[self.cutNum],
```

```
            15, linewidths=0.25, colors='k', norm = self.Norm)
        pyplot.clabel(self.s, fmt = self.format, colors = '0.15', fontsize=9)
        self.axes.set_xlabel('%s distance [m]'%(self.axH), fontsize=8)
        self.axes.set_ylabel('%s distance [m]'%(self.axV), fontsize=8)
        # update the font size of the x and y axes
        for tick in self.axes.xaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        for tick in self.axes.yaxis.get_major_ticks():
            tick.label1.set_fontsize(8)
        #left,bottom,width,height
        cax = self.fig.add_axes([0.1, 0.1, 0.8, 0.02])
        cb = mpl.colorbar.ColorbarBase \
            (cax, cmap=cm.get_cmap(self.cmap), norm=self.Norm,
            orientation = 'horizontal', boundaries = self.Ticks,
            format = '%2.1f')
        cb.set_label((self.calcList[self.calcDrop.GetCurrentSelection()]),
            fontsize=7)
        cb.ax.set_xticklabels(self.Ticklabels)
        for t in cb.ax.get_xticklabels():
            t.set_fontsize(7)
        # send resize event to refresh panel
        pix = tuple( self.panel2.GetClientSize() )
        set = (pix[0]*1.01, pix[1]*1.01)
        self.chart.SetClientSize( set )
        self.canvas.SetClientSize( set )
        self.makeKey()
        print 'updatePlot fine'

    def makeKey(self):
        '''function for creating 3-D slice key'''
        # plot containers for slice key
        self.key = wx.Panel(self.panel4)
```

```
        self.keyfig = Figure(dpi=100, facecolor='none')
        self.keycanvas = FigureCanvas(self.key, -1, self.keyfig)
        self.keySizer.Add(self.key, 0, wx.ALL|wx.EXPAND, 5)
        self.keyaxes = axes3d.Axes3D(self.keyfig)
        self.keyaxes.disable_mouse_rotation()
        xL = [self.xmin,self.xmax]
        yL = [self.ymin,self.ymax]
        zL = [self.zmin,self.zmax]
        xJ = [self.winxmin,self.winxmax]
        yJ = [self.winymin,self.winymax]
        zJ = [self.winzmin,self.winzmax]
        # set plot limits
        self.keyaxes.set_xlim((xL[0],xL[1]))
        self.keyaxes.set_ylim((yL[0],yL[1]))
        self.keyaxes.set_zlim((zL[0],zL[1]))
        # plot room facades
        xN = [xL[0],xL[0],xL[1],xL[1]]
        yN = [yL[1],yL[1],yL[1],yL[1]]
        zN = [zL[0],zL[1],zL[1],zL[0]]
        verts = [zip(xN,yN,zN)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('1'),edgecolors = ('k'), linewidths=(0.5),
            alpha=0.05))
        # east wall
        xE = [xL[1],xL[1],xL[1],xL[1]]
        yE = [yL[0],yL[0],yL[1],yL[1]]
        zE = [zL[0],zL[1],zL[1],zL[0]]
        verts = [zip(xE,yE,zE)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('1'),edgecolors = ('k'), linewidths=(0.5),
            alpha=0.05))
        # south wall
```

```
        xS = [xL[0],xL[0],xL[1],xL[1]]
        yS = [yL[0],yL[0],yL[0],yL[0]]
        zS = [zL[0],zL[1],zL[1],zL[0]]
        verts = [zip(xS,yS,zS)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
            alpha=0.05))
        # west wall
        xW = [xL[0],xL[0],xL[0],xL[0]]
        yW = [yL[0],yL[0],yL[1],yL[1]]
        zW = [zL[0],zL[1],zL[1],zL[0]]
        verts = [zip(xW,yW,zW)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
            alpha=0.05))
        # floor
        xF = [xL[0],xL[0],xL[1],xL[1]]
        yF = [yL[0],yL[1],yL[1],yL[0]]
        zF = [zL[0],zL[0],zL[0],zL[0]]
        verts = [zip(xF,yF,zF)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
            alpha=0.05))
        # ceiling
        xC = [xL[0],xL[0],xL[1],xL[1]]
        yC = [yL[0],yL[1],yL[1],yL[0]]
        zC = [zL[1],zL[1],zL[1],zL[1]]
        verts = [zip(xC,yC,zC)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('1'), edgecolors = ('k'), linewidths=(0.5),
            alpha=0.05))
        # window
```

```
        xWin = [xJ[0],xJ[0],xJ[1],xJ[1]]
        yWin = [yJ[1],yJ[1],yJ[1],yJ[1]]
        zWin = [zJ[0],zJ[1],zJ[1],zJ[0]]
        verts = [zip(xWin,yWin,zWin)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('w'), edgecolors = ('b'), linewidths=(0.65),
            alpha=0.05))
        # cut slice
        ct = [self.pt[2,0,0,self.ind], self.pt[0,self.ind,0,0],
            self.pt[1,0,self.ind,0]]
        self.TT = ct[self.cutNum]
        cn = ['X-Y', 'Y-Z', 'X-Z']
        self.SS = cn[self.cutNum]
        if self.SS == 'X-Y':
            xCut = [xL[0],xL[1],xL[1],xL[0]]
            yCut = [yL[0],yL[0],yL[1],yL[1]]
            zCut = [self.TT,self.TT,self.TT,self.TT]
        if self.SS == 'Y-Z':
            xCut = [self.TT,self.TT,self.TT,self.TT]
            yCut = [yL[0],yL[0],yL[1],yL[1]]
            zCut = [zL[0],zL[1],zL[1],zL[0]]
        if self.SS == 'X-Z':
            xCut = [xL[0],xL[0],xL[1],xL[1]]
            yCut = [self.TT,self.TT,self.TT,self.TT]
            zCut = [zL[0],zL[1],zL[1],zL[0]]
        verts = [zip(xCut,yCut,zCut)]
        self.keyaxes.add_collection3d(Poly3DCollection(verts,
            facecolor = ('#BDBDBD'), edgecolors = ('#969696'), alpha=1))
        # format plot
        self.keyaxes.set_xticks([])
        self.keyaxes.set_yticks([])
        self.keyaxes.set_zticks([])
```

```
        self.keyaxes.set_xlabel('x', fontsize=8)
        self.keyaxes.set_ylabel('y', fontsize=8)
        self.keyaxes.set_zlabel('z', fontsize=8 )
        # link key containers to resize functions
        self._SetSizeKey()
        self.keycanvas.draw()
        self.key._resizeflag = False
        self.key.Bind(wx.EVT_IDLE, self._onIdleKey)
        self.key.Bind(wx.EVT_SIZE, self._onSizeKey)

    def OnChoice(self, event):
        choice = event.GetString()
        print choice

    def OnClose(self, event):
        # deinitialize the frame manager
        self._mgr.UnInit()
        # delete the frame
        self.Destroy()

    def OnPrint(self, event):
        '''function for exporting images as .png files'''
        cwd = os.getcwd()
        #make destination folders
        dirs = [os.path.join(cwd, 'images')]
        for i in dirs:
            try:
                os.makedirs(i)
            except OSError:
                pass
        ct = [self.pt[2,0,0,self.ind], self.pt[0,self.ind,0,0],
            self.pt[1,0,self.ind,0]]
```

```
        self.TT = ct[self.cutNum]
        cn = ['XY', 'YZ', 'XZ']
        self.SS = cn[self.cutNum]
        i = str(self.calcDrop.GetSelection())
        cn = { '0':'MRT', '1':'OpTemp', '2':'PMV', '3':'PPD',
            '4':'AdaptASHRAE', '5':'AdaptEN15251', '6':'AdaptNPRCR1752', }
        calc = (cn[i])
        j = self.timeframeDrop.GetSelection()
        if j == 0:
            timeStart = '%s%s%s' % \
                (self.StartMonth,self.StartDay,self.StartHour)
            timeEnd = '-%s%s%s' % \
                (self.EndMonth,self.EndDay,self.EndHour)
        else:
            timeStart = '%s%s%s' % \
                (self.StartMonth,self.StartDay,self.StartHour)
            timeEnd = ''
        heatmapFname = cwd + '/images' + '/hmap%s_%s%s_%s%sm.png' % \
            (calc,timeStart,timeEnd,self.SS,self.TT)
        scatterFname = cwd + '/images' + '/sctr%s_%s%s_RmAvg.png' % \
            (calc,timeStart,timeEnd)
        print 'Saving image', heatmapFname
        print 'Saving image', scatterFname
        self.fig.savefig(heatmapFname, dpi = 300, format='png')
        self.scatterFig.savefig(scatterFname, dpi = 300, format='png')

    # Resize Plot Functions - heatmap plot ---------------------------
    # plots are updated using these resize events

    def _onSize( self, event ):
        self.chart._resizeflag = True

```

```
    def _onIdle( self, evt ):
        if self.chart._resizeflag:
            self.chart._resizeflag = False
            self._SetSize()

    def _SetSize( self ):
        pixels = tuple( self.panel2.GetClientSize() )
        self.chart.SetSize( pixels )
        self.canvas.SetSize( pixels )
        self.fig.set_size_inches \
            ( float( pixels[0] )/self.fig.get_dpi(),
                float( pixels[1] )/self.fig.get_dpi() )

    def draw(self): pass # abstract, to be overridden by child classes

    # Resize Plot Functions - diagram key -----------------------------

    def _onSizeKey( self, event ):
        self.key._resizeflag = True

    def _onIdleKey( self, evt ):
        if self.key._resizeflag:
            self.key._resizeflag = False
            self._SetSizeKey()

    def _SetSizeKey( self ):
        pixels = tuple( self.panel4.GetClientSize() )
        self.key.SetSize( [pixels[0]/1.1,pixels[1]/1.1] )
        self.keycanvas.SetSize( [pixels[0]/1.1,pixels[1]/1.1] )
        self.keyfig.set_size_inches \
            ( float( pixels[0]/1.1 )/self.keyfig.get_dpi(),
                float( pixels[1]/1.1 )/self.keyfig.get_dpi() )
```

```

    def draw(self): pass # abstract, to be overridden by child classes

    # Resize Plot Functions - scatter plot ----------------------------

    def _onSizeScatter( self, event ):
        self.scatterChart._resizeflag = True

    def _onIdleScatter( self, evt ):
        if self.scatterChart._resizeflag:
            self.scatterChart._resizeflag = False
            self._SetSizeScatter()

    def _SetSizeScatter( self ):
        pixels = tuple( self.panel3.GetClientSize() )
        self.scatterChart.SetSize( pixels )
        self.scatterCanvas.SetSize( pixels )
        self.scatterFig.set_size_inches \
            ( float( pixels[0] )/self.scatterFig.get_dpi(),
                float( pixels[1] )/self.scatterFig.get_dpi() )

    def draw(self): pass # abstract, to be overridden by child classes

# FileSelectorCombo class -------------------------------------------
class FileSelectorComboIDF(wx.combo.ComboCtrl):
    '''class for control for selecting .idf file'''
    def __init__(self, *args, **kw):
        wx.combo.ComboCtrl.__init__(self, *args, **kw)

        # make a custom bitmap showing "..."
        bw, bh = 14, 16
        bmp = wx.EmptyBitmap(bw,bh)
```

```
        dc = wx.MemoryDC(bmp)

        # clear to a specific background colour
        bgcolor = wx.Colour(255,254,255)
        dc.SetBackground(wx.Brush(bgcolor))
        dc.Clear()

        # draw the label onto the bitmap
        label = "..."
        font = wx.SystemSettings.GetFont(wx.SYS_DEFAULT_GUI_FONT)
        font.SetWeight(wx.FONTWEIGHT_BOLD)
        dc.SetFont(font)
        tw,th = dc.GetTextExtent(label)
        dc.DrawText(label, (bw-tw)/2, (bw-tw)/2)
        del dc

        # now apply a mask using the bgcolor
        bmp.SetMaskColour(bgcolor)

        # and tell the ComboCtrl to use it
        self.SetButtonBitmaps(bmp, True)

    # Overridden from ComboCtrl, called when the combo button is clicked
    def OnButtonClick(self):
        path = ""
        name = ""
        if self.GetValue():
            path, name = os.path.split(self.GetValue())

        dlg = wx.FileDialog(self, "Choose File", path, name,
                            ".idf files (*.idf)|*.idf", wx.FD_OPEN)
        if dlg.ShowModal() == wx.ID_OK:
```

```
                self.SetValue(dlg.GetPath())
            dlg.Destroy()
            self.SetFocus()

    # Overridden from ComboCtrl to avoid assert since there is no ComboPopup
    def DoSetPopupControl(self, popup):
        pass

class FileSelectorComboCSV(wx.combo.ComboCtrl):
    '''class for control for selecting .csv file'''
    def __init__(self, *args, **kw):
        wx.combo.ComboCtrl.__init__(self, *args, **kw)

        # make a custom bitmap showing "..."
        bw, bh = 14, 16
        bmp = wx.EmptyBitmap(bw,bh)
        dc = wx.MemoryDC(bmp)

        # clear to a specific background colour
        bgcolor = wx.Colour(255,254,255)
        dc.SetBackground(wx.Brush(bgcolor))
        dc.Clear()

        # draw the label onto the bitmap
        label = "..."
        font = wx.SystemSettings.GetFont(wx.SYS_DEFAULT_GUI_FONT)
        font.SetWeight(wx.FONTWEIGHT_BOLD)
        dc.SetFont(font)
        tw,th = dc.GetTextExtent(label)
        dc.DrawText(label, (bw-tw)/2, (bw-tw)/2)
        del dc

```

```
        # now apply a mask using the bgcolor
        bmp.SetMaskColour(bgcolor)

        # and tell the ComboCtrl to use it
        self.SetButtonBitmaps(bmp, True)


    # Overridden from ComboCtrl, called when the combo button is clicked
    def OnButtonClick(self):
        path = ""
        name = ""
        if self.GetValue():
            path, name = os.path.split(self.GetValue())

        dlg = wx.FileDialog(self, "Choose File", path, name,
                            ".csv files (*.csv)|*.csv", wx.FD_OPEN)
        if dlg.ShowModal() == wx.ID_OK:
            self.SetValue(dlg.GetPath())
        dlg.Destroy()
        self.SetFocus()

    # Overridden from ComboCtrl to avoid assert since there is no ComboPopup
    def DoSetPopupControl(self, popup):
        pass

# End FileSelectorCombo class------------------------------------------

app = wx.PySimpleApp()
frame = MyFrame(None, -1, "cMap - Thermal Comfort Spatial Mapping")
frame.Show()
app.MainLoop()
```

```
# END INTERFACE -------------------------------------------------------
```