

BUILDING ENERGY SIMULATION USING CONSTRAINED OPTIMIZATION

A Dissertation

by

CLINTON PAUL DAVIS

Submitted to the Office of Graduate and Professional Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Charles H. Culp
Co-Chair of Committee,	David E. Claridge
Committee Members,	Bryan P. Rasmussen
	Michael B. Pate
Head of Department,	Andreas A. Polycarpou

May 2019

Major Subject: Mechanical Engineering

Copyright 2019 Clinton Paul Davis

## ABSTRACT

This thesis presents an equation-based approach to steady state building energy modeling using constrained optimization. Models using this approach can be built from algebraic equations that describe a building and its HVAC equipment and knowledge of the system's overall control strategy. The equation-based nature of this approach allows arbitrary systems to be represented, and simple rules can be followed to construct models that produce a valid energy balance overall operating conditions, including atypical scenarios such as temperatures drifting above or below their desired set points.

The theory behind constrained optimization modeling of steady state HVAC systems is presented, and methods for constructing typical HVAC systems are given. Mathematical methods for integrating dynamic effects such as thermal mass into these models are also presented. To efficiently solve these models, a prototype program named Beryl is presented along with a performance analysis of the computer resources required for different types of modeling. Finally, future areas of research are given such as modeling improvements and improvements to numerical solvers for the specialized problems being solved.

The solver developed in this work was able to perform yearly building energy simulations on single AHU in 0.5 to 5.0 seconds using hourly time steps on a single thread. Factors that affect simulation runtime were also analyzed. Based on these results and the robustness of the simulations, constrained optimization based building energy modeling was shown to be a valid and potentially useful approach.

## DEDICATION

To my mother, Melanie.

## ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Culp, for his guidance and support throughout the course of this research. I would also like to thank my co-chair, Dr. David E. Claridge, and my committee members Dr. Brian P. Rasmussen and Dr. Michael B. Pate for their input during this work. Also, I would like to thank my mother, father, grandfather, and grandmother for their love and encouragement.

## CONTRIBUTORS AND FUNDING SOURCES

This work was supported by a dissertation committee consisting of Professor Charles H. Culp of the Department of Architecture and David E. Claridge of the Department of Mechanical Engineering as co-advisor, Brian P. Rasmussen of the Department of Mechanical Engineering, and Michael B. Pate of the Department of Mechanical Engineering. All other work conducted for the dissertation was completed by the student independently.

## NOMENCLATURE

$X$	A Fraction that Varies from 0 to 1 (0 – 1)
$Pct$	Percentage (%)
$\alpha$	A Coefficient or Conversion Factor ( <i>Varies</i> )
$\eta$	Efficiency (0 – 1)
$l,w$	Length, Width ( $ft$ )
$A$	Area ( $ft^2$ )
$\widetilde{Occ}$	Occupancy ( $ft^2/Person$ )
$V$	Volume ( $ft^3$ )
$t,\tau,\delta$	Time, Time Constant, Time Step Period ( $h$ )
$ACH$	Air Changes per Hour ( <i>Air Changes/h</i> )
$\rho$	Density ( $lbm/ft^3$ )
$T,\Delta T$	Temperature, Temperature Rise ( $^{\circ}F$ )
$W$	Humidity Ratio ( $lb_w/lb_{da}$ )
$P$	Pressure ( <i>inches <math>H_2O</math>, psi</i> )
$\bar{V}$	Air Volume Flow Rate per $ft^2$ of Floor Area ( $CFM/ft^2$ )
$c_p$	Specific Heat ( $Btu/(lbm \cdot ^{\circ}F)$ )
$R$	R-Value ( $(h \cdot ft^2 \cdot ^{\circ}F)/Btu$ )
$C$	Thermal Capacitance ( $Btu/(ft^2 \cdot ^{\circ}F)$ )
$\overline{UA}$	UA-Value per $ft^2$ of Floor Area ( $Btu/(h \cdot ^{\circ}F)$ )
$\bar{E}$	Electric Usage per $ft^2$ of Floor Area ( $W/ft^2$ )

$\bar{Q}$	Heating Energy Flow ( $Btu/(h \cdot ft^2)$ )
$\widetilde{Hp}$	Fan Horsepower per 1000 CFM ( $Hp/kCFM$ )
$\overline{Hp}$	Horsepower per $ft^2$ of Floor Area ( $Hp/ft^2$ )
$\overline{Bhp}$	Brake Horsepower per $ft^2$ of Floor Area ( $Hp/ft^2$ )
<i>Price</i>	Price of an Energy Source ( $$/kWh$ or $$/MMBtu$ )

## TABLE OF CONTENTS

	Page
ABSTRACT .....	ii
DEDICATION .....	iii
ACKNOWLEDGEMENTS .....	iv
CONTRIBUTORS AND FUNDING SOURCES.....	v
NOMENCLATURE.....	vi
TABLE OF CONTENTS .....	viii
LIST OF FIGURES.....	xii
LIST OF TABLES .....	xvii
CHAPTER I INTRODUCTION .....	1
Background .....	1
Objectives and Scope .....	2
Organization of the Thesis .....	3
CHAPTER II LITERATURE REVIEW .....	4
Building Energy Modeling.....	4
Early Building Energy Analysis.....	5
Steady State and Quasi-Steady State Simulators .....	7
Dynamic, Equation-Based Modeling .....	9
Spawn of EnergyPlus .....	10
Model Optimization .....	11
Constrained Optimization Theory.....	11
Optimization Overview .....	11
Research History .....	14
Specialized Branches of Optimization .....	16
Constrained Optimization in HVAC.....	18
Control Optimization.....	18
Model Predictive Control .....	18
CHAPTER III STEADY STATE MODELING .....	20
Constrained Optimization Based Modeling .....	20
A Simple System.....	21



Equality Constraints Alone .....	23
A Constrained Optimization Model .....	24
Atypical Operating Conditions.....	25
Modeling with Lexicographic Objectives.....	27
<b>CHAPTER IV INTEGRATING DYNAMIC MODELING .....</b>	<b>34</b>
Dynamic Building Processes.....	34
Hybrid Steady State/Dynamic Modeling .....	36
A Simple Hybrid Model.....	38
Integrating Linear Differential Equations to Steady State Models .....	45
Hybrid Wall Models with Multiple Nodes.....	47
<b>CHAPTER V EXAMPLE HVAC MODELS FOR COMMON AIR HANDLER TYPES.....</b>	<b>50</b>
Building Description .....	50
Dimensions and Layout.....	50
Thermal Properties .....	52
Zone Loads .....	55
Energy Sources and Prices .....	57
Weather Data for Simulations .....	58
Constants and Conversion Factors .....	59
Hourly Zone Loads.....	59
Single Duct Constant Air Volume (SDCAV) Systems.....	61
SDCAV Input Parameters .....	61
Precalculated SDCAV Parameters .....	62
The SDCAV Model.....	63
SDCAV Simulation Results .....	65
Modifying the Air Handler Equipment .....	67
Single Duct Variable Air Volume (SDVAV) Systems.....	71
SDVAV Parameters .....	72
Fan Curve Functions .....	75
The SDVAV Model.....	79
Optimizing Energy Cost.....	83
Cooling Coil Setpoint Optimization.....	85
Dual Duct Variable Air Volume (DDVAV) Systems.....	88
Modeling Thermal Mass .....	93
Effect of Time Step Period Length.....	95
<b>CHAPTER VI AN AUTOMATED SOLUTION ALGORITHM .....</b>	<b>99</b>
Overall Algorithm .....	99
Input Models .....	100
Model Preprocessing.....	101

Subproblem Graphs .....	102
Preprocessing Operations .....	103
Basic Subproblem Graph Operations .....	108
Code Generation, Compilation, and Execution.....	108
CHAPTER VII SOLVER PROGRAM IMPLEMENTATION .....	109
Program Structure .....	109
User Interface .....	111
Simulation Manager .....	112
Text Input Parser .....	112
Model Preprocessor.....	113
Code Generation Library.....	113
Numerical Algorithms .....	115
Compilation .....	121
Computer Algebra Library .....	121
Running a Simulation.....	123
Validation and Unit Testing .....	126
Performance .....	127
Limiting Factors in Calculation Speed.....	127
Measured Performance.....	129
CHAPTER VIII CONCLUSIONS AND FUTURE RESEARCH.....	133
Conclusions .....	133
Future Research.....	134
Modeling Improvements .....	134
Solver Improvements .....	135
REFERENCES .....	137
APPENDIX A TIME AVERAGES OF SOLUTIONS TO LINEAR DIFFERENTIAL EQUATIONS .....	145
Time Varying Solution.....	145
Average of Time Varying Solution.....	146
APPENDIX B HVAC MODELING.....	147
Control Error Functions .....	147
Static Pressure Control.....	150
Fan Pressure Curves .....	150
Fixed Static Pressure Set Point Model .....	153
Effect of Changing the Static Pressure Set Point .....	154
Variable Static Pressure Set Point Model.....	156

Psychrometrics .....	159
Water Saturation Pressure .....	159
Saturation Humidity Ratio.....	160
Relative Humidity .....	160
RC Networks for Wall Thermal Mass.....	160
Modeled with a Single Node .....	162
Modeled with Multiple Nodes.....	163
Infiltration and Exfiltration Calculations .....	166
 APPENDIX C CONSTRAINED OPTIMIZATION REFERENCE.....	 167
Nonlinear Programming Reformulations .....	167
Discontinuous Function Approximations.....	167
 APPENDIX D NONLINEAR PROGRAMMING SENSITIVITY ANALYSIS DERIVATION .....	  171
 APPENDIX E BERYL USERS GUIDE.....	 176
User Interface .....	176
Input Model Format .....	176
The Setup Section.....	178
Problem Sections.....	184
Running a Model.....	192

## LIST OF FIGURES

	Page
Figure 1 – Relationship between Literature Review Areas and Thesis Areas .....	4
Figure 2 – Temperature Parameters for a Dual Duct Constant Air Volume System .....	6
Figure 3 – Knebel’s Algorithm to Solve Temperatures in the DDCAV System .....	7
Figure 4 – An unconstrained optimization .....	12
Figure 5 – A constrained optimization .....	12
Figure 6 – Canonical Form of Linear Programs.....	13
Figure 7 – Form of a Nonlinear Program .....	13
Figure 8 – Box-Bound Form of a Nonlinear Program .....	13
Figure 9 – Convex and Non-Convex Functions and Sets .....	14
Figure 10 – A Lexicographic Optimization Problem.....	16
Figure 11 – Lexicographic Decomposition .....	17
Figure 12 – Using Lexicographic Nonlinear Programming for Steady State HVAC .....	21
Figure 13 – A Single-Zoned Single Duct Constant Air Volume System.....	22
Figure 14 – SZCAV Zone Temperature vs. the Required Supply Air Temperature.....	24
Figure 15 – Model of Single-Zoned Single Duct Constant Air Volume Temperatures ..	24
Figure 16 – SDVAV Zone Temperature vs. Supply Air Temperature .....	25
Figure 17 – A SDCAV System That Can’t be Controlled .....	27
Figure 18 – Single-Zoned Variable Air Volume Temperature Balance Equations .....	28
Figure 19 – Zone Temperature Error as a Function of Heating and Cooling Set Points .....	29
Figure 20 – Stage 1 of the SZVAV Model.....	30
Figure 21 – Stage 2 of the SZVAV Model.....	31

Figure 22 – Stage 3 of the SZVAV Model.....	31
Figure 23 – Supply Air Flow and Reheat Usages that Meet Zone Temperature Set Points for the SZVAV Model .....	32
Figure 24 – Solution to Stages 1, 2, and 3, for when the Outside Air Temperature is 100°F .....	33
Figure 25 – Time Scales of HVAC Processes.....	34
Figure 26 – Temperature Lag and Dampening due to Thermal Mass on an Exterior Wall.....	35
Figure 27 – Constructing a Hybrid Model from Static and Dynamic States .....	36
Figure 28 – Mathematical Form for Hybrid Problems.....	37
Figure 29 – Converting a Hybrid Model to a Solvable Constrained Optimization.....	38
Figure 30 – A Steady State Zone Model with Thermal Mass in the Wall .....	39
Figure 31 – Mathematical Formulation of the Zone Model.....	41
Figure 32 – Average and Continuous Wall Temperatures over a Time Step .....	42
Figure 33 – Solvable Hybrid Model of a Zone with Thermal Mass .....	44
Figure 34 – Example Hybrid Model Simulation.....	45
Figure 35 – An RC Network for Modeling Thermal Mass of a Wall .....	47
Figure 36 – Hybrid Model with the Wall Divided into Multiple Layers .....	48
Figure 37 – Air Flow Rates using Different Numbers of Wall Temperature Nodes .....	49
Figure 38 – Example Building Floor Layout .....	50
Figure 39 – Plug and Occupancy Load Ratios for the Example Building .....	56
Figure 40 – Hourly Weekday Plug Load Schedule Values.....	57
Figure 41 – Weather Data Used in Simulations .....	58
Figure 42 – SDCAV System Layout.....	61
Figure 43 – Model of a Single Duct Constant Air Volume System.....	63

Figure 44 – SDCAV Heating and Cooling Usages .....	66
Figure 45 – Loss of Perimeter Zone Temperature Control in the SDCAV Simulation ...	66
Figure 46 – The SDCAV Model of Figure 42 with Outside Air Preheat and Perimeter Zone Infiltration.....	67
Figure 47 – Preheat Coil Leaving Air Temperature for the Modified SDCAV Model ...	69
Figure 48 – Cooling Coil Chilled Water Usages for the Modified and Unmodified Systems .....	70
Figure 49 – Reheat Coil Hot Water Usages for the Modified and Unmodified Systems .....	71
Figure 50 – A Single Duct Variable Air Volume System with Static Pressure Control..	72
Figure 51 – Fan Curve Inputs for the SDVAV Model .....	74
Figure 52 – Typical Static Pressure and Brake Horsepower Power Curves for a Backward Curved Centrifugal Fan .....	76
Figure 53 – Model of a Single Duct Variable Air Volume System .....	80
Figure 54 – SDVAV Zone Temperatures .....	82
Figure 55 – SDVAV Supply Air and Zone Temperatures .....	83
Figure 56 – SDVAV Supply Air Flow versus Reheat Usage.....	83
Figure 57 – Lexicographic Objective Functions to Optimize Energy Cost for the SDVAV Model .....	84
Figure 58 – Monthly Energy Costs for the Cost Optimized SDVAV Model .....	84
Figure 59 – New Objective Functions for Achieving Relative Humidity Control while Optimizing Cooling Coil Setpoints .....	87
Figure 60 – Optimal Cooling Coil Set Points for the SDVAV Cost Optimization Model.....	88
Figure 61 – DDVAV System Layout .....	89
Figure 62 – Model of a Dual Duct Variable Air Volume System (Continued Below)....	90
Figure 63 – Perimeter Zone DDVAV Air Flow Rates .....	93

Figure 64 – Dynamic Wall Temperature for the DDVAV Model .....	94
Figure 65 – Monthly DDVAV Energy Costs with and without Dynamics .....	95
Figure 66 – Weekday Electric and People Load Schedules for a Time Step of 15 Minutes .....	96
Figure 67 – Monthly Energy Costs for Different Time Step Sizes for the Steady State DDVAV Model .....	97
Figure 68 – Dynamic DDVAV Wall Temperatures for Different Time Step Periods.....	98
Figure 69 – Automated Compilation Process .....	100
Figure 70 – Partitioning a SDCAV Model into Smaller Subproblems .....	101
Figure 71 – Preprocessing Algorithm Pseudocode .....	102
Figure 72 – A Subproblem Graph for Algebraic Equations.....	103
Figure 73 – Basic Subproblem Graph Operations .....	108
Figure 74 – Usage Relationships between Beryl’s Modules.....	111
Figure 75 – A Screenshot of Beryl’s User Interface .....	112
Figure 76 – An Error Message for an Undefined Variable in Beryl.....	113
Figure 77 – Code to Generate an Abstract Syntax Tree to Multiply Two 2D Matrices .....	114
Figure 78 – Code Generation Library of a For Loop, It’s Abstract Syntax Tree, and the Generated Code Output .....	115
Figure 79 – Sparse vs. Dense Matrix Factorization Speed .....	120
Figure 80 – A Linear Function Represented with an Expression Tree .....	122
Figure 81 – The Project Runner Window .....	124
Figure 82 – Presolve Graph Window .....	125
Figure 83 – An Output Chart Window in Beryl.....	126
Figure 84 – Basic Control Error Functions .....	148
Figure 85 – Fan Static Pressure Control in a VAV System .....	150

Figure 86 – Fan Pressure Curves.....	151
Figure 87 – Fan Brake Horsepower Curves .....	152
Figure 88 – System Curve Changes versus Static Pressure Setpoint Changes .....	155
Figure 89 – Pressure Changes in a System with Static Pressure Control .....	155
Figure 90 – Water Saturation Pressure Equation from ASHRAE .....	159
Figure 91 – RC Network with a Single Node .....	162
Figure 92 – An Equally Divided RC Network with Multiple Nodes.....	163
Figure 93 – RC Network for an Interior Node .....	163
Figure 94 – RC Network for an Exterior Node.....	164
Figure 95 – Continuous Approximation of an Absolute Value .....	169
Figure 96 – Continuous Approximation of $\text{Max}(x, 0)$ by Bertsekas.....	170
Figure 97 – A Parametric Nonlinear Programming Problem.....	171
Figure 98 – Beryl's Main User Interface .....	176
Figure 99 – Model Input Structure of Beryl.....	177
Figure 100 – Problem Sections with a Cyclic Dependency .....	192
Figure 101 – Input File Error Window.....	192
Figure 102 – Project Runner Window.....	193



## LIST OF TABLES

	Page
Table 1 – Modeling Time Domains .....	5
Table 2 – Local Minima for Different Problem Types .....	15
Table 3 – Parameters for the SZCAV Example .....	22
Table 4 – Additional Parameters for the SZVAV Example.....	28
Table 5 – Input Parameters for the Example Hybrid Model .....	40
Table 6 – Calculated Parameters for the Example Hybrid Model .....	41
Table 7 – Example Building Dimensions .....	51
Table 8 – Building Zone Parameters .....	51
Table 9 – Calculated Building Areas .....	52
Table 10 – Building Thermal Property Inputs .....	53
Table 11 – Calculated Wall Parameters .....	54
Table 12 – Calculated Zone Parameters.....	55
Table 13 – Lighting, Plug and People Load Inputs.....	55
Table 14 – Zone Load Inputs .....	55
Table 15 – Energy Price Inputs .....	57
Table 16 – Physical Constants and Conversion Factors Used in Modeling.....	59
Table 17 – Calculated Zone Peak Loads .....	60
Table 18 – SDCAV System Parameters.....	62
Table 19 – SDCAV Zone Parameters .....	62
Table 20 – Precalculated SDCAV Parameters .....	63
Table 21 – Added Parameters for SDCAV Model Changes .....	67

Table 22 – Calculated Parameters for SDCAV Model Changes .....	68
Table 23 – SDVAV System Parameters .....	73
Table 24 – SDVAV Zone Parameters .....	75
Table 25 – Denormalized Fan Curve Coefficients for the SDVAV Model .....	79
Table 26 – SDVAV Cooling Coil Setpoint Optimization Additional Parameters .....	85
Table 27 – DDVAV System Parameters .....	89
Table 28 – Calculated DDVAV System Parameters.....	90
Table 29 – Input Model Components.....	100
Table 30 – Preprocessing Operations.....	104
Table 31 – Major Modules of Beryl.....	110
Table 32 – Numerical Algorithms Implemented in the Beryl.....	116
Table 33 – Effect of Model Additions on Runtime.....	129
Table 34 – Full Model Runtimes.....	130
Table 35 – Partitioned Temperature and Humidity Model Runtimes.....	131
Table 36 – Full Model Runtimes with Six Node Wall Dynamic Model.....	131
Table 37 – Full Model Runtimes without Sensitivity Analysis .....	132
Table 38 – Basic Error Function Equations .....	149
Table 39 – Fixed Static Pressure Set Point Model Inputs .....	153
Table 40 – Fixed Static Pressure Set Point Model Outputs .....	153
Table 41 – Variable Static Pressure Set Point Model Inputs .....	157
Table 42 – Variable Static Pressure Set Point Model Outputs.....	157
Table 44 – Coefficients for Saturation Water Pressure Calculations.....	160
Table 44 – Parameters Used in Wall Thermal Modeling.....	161
Table 45 – Infiltration/Exfiltration Variables.....	166

Table 46 – Nonlinear Programming Reformulations .....	167
Table 47 – Parameters Used in Solving Parametric Nonlinear Programming Problem.....	171
Table 48 – Setup Section Line Types.....	178
Table 49 – Beryl User Options.....	181
Table 50 – Problem Section Line Types .....	185
Table 51 – Units Implemented in Beryl .....	189

# CHAPTER I

## INTRODUCTION

### **Background**

Building engineers, operators, and architects analyze building energy flows to minimizing energy usage and costs while maintaining thermal comfort. Modeling is often used to aid this process. Over the past 50 years hundreds of building energy programs have been developed, enhanced, and are in use [1]. The simulations they perform involve multiple domains, such as thermodynamics, fluid dynamics, heat and mass transfer, electrical systems, control systems and communication systems. Describing these areas can require a wide variety of math, including algebraic equations, differential equations, partial differential equations, and constrained optimizations.

Most modern building simulation programs are large, monolithic, and require developers to add extensions in order to simulate new technologies [2]. The algorithms used to solve problems are usually for a specific domain, such as partial differential equation solving or whole building simulation. Modeling with aspects of multiple domains or performing optimizations can require coupling multiple pieces of software together.

The protocols used in whole building energy simulators like EnergyPlus treat HVAC systems as a set of interacting modules [3]. This limits the simplifications that can be applied when solving models and makes a mathematical analysis of a model as a whole difficult. For instance, calibrating or optimizing building energy models can involve the use of derivatives, but these can be difficult to calculate. Errors in the convergence of internal algorithms can result in large changes in output values for small changes in inputs. Also, gradient calculations can require a separate calculation for each variable's derivative.

Differential-algebraic equation solvers such as Modelica express models in terms of mathematical equations instead of algorithms. This allows users to model new equipment while at the same time allowing models to be build based on libraries of

HVAC components [4]. Modelica's equation solving approach also allows for optimizations by simplifying models before solving them with a numerical solver. Models are limited by the type of math that the solver can handle. Optimica extends the Modelica language to handle constrained optimizations [5]. This allows for the direct coupling of models in multiple domains in one tool.

The complexity of building energy systems is rapidly increasing and dealing with this complexity is likely to be a driving force for future building simulation programs [2]. Equation-based modeling aid in dealing with complexity by allowing for new models to be created in a modular way without black box algorithms. It also separates the domains of modeling and mathematical solving, which allows for tools to be created that handle problems that cross multiple domains.

### **Objectives and Scope**

This thesis presents a technique to unify existing building simulation methods under a single equation-based approach. By formulating steady state building energy models as constrained optimizations model variables can be mathematically coupled with equations from other types of modeling approaches. Ultimately the ability to successfully simulate different modeling methods depends on the speed, and robustness of the numerical solver used and the diversity of the mathematical procedures it implements.

Steady state models were implemented to demonstrate using constrained optimization for steady state modeling. Models were developed for three common types of air handling units for a hypothetical office building with three zones and contain approximately 50 to 100 equations and constraints. An analytical treatment for integrating dynamic calculations into a steady state simulation and the theoretical implications of this are given.

A numerical solver named Beryl was created to demonstrate the performance and robustness that can be achieved in solving the developed models. Beryl uses a decomposition algorithm to simplify and decompose models into a sequence of smaller problems before writing a program to solve each problem. These performance results

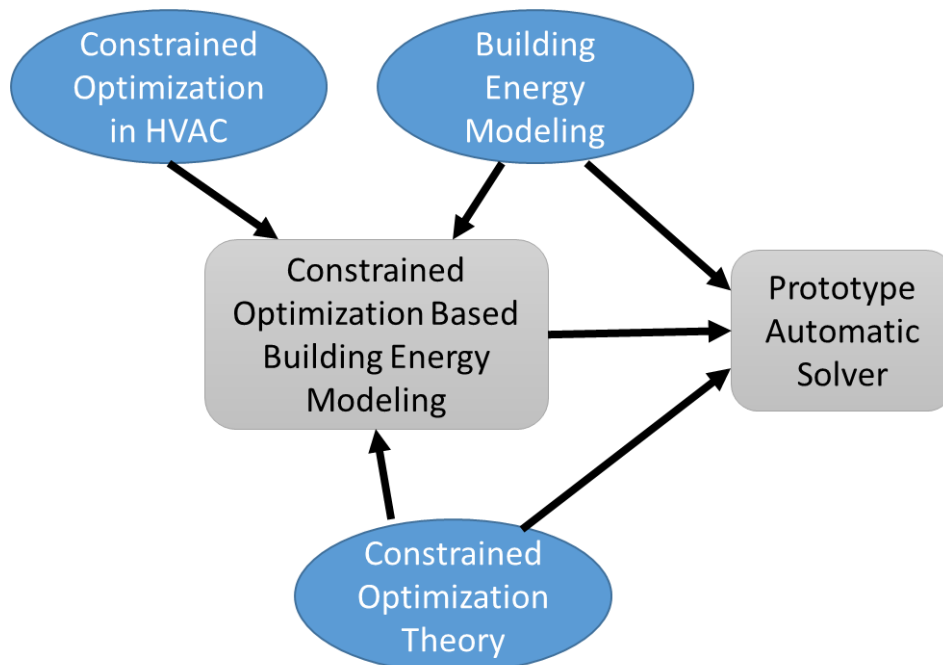
are then combined with theoretical results on the algorithms used to solve them to estimate the runtime performance of larger models.

### **Organization of the Thesis**

The literature review of Chapter II covers topics related to HVAC simulation and constrained optimization. These topics serve as the basis for the mathematical methods and solver that are presented. Chapter III presents a method for performing steady state building energy use calculations using constrained optimization. Next, a method of integrating passive dynamics such as thermal mass into these calculations is presented in Chapter IV. Chapter V gives examples of models of three common air handler types. In Chapter VI an algorithm for decomposing and simplifying constrained optimization problems is presented. A program for performing building energy calculations is presented in Chapter VII. This program implements the algorithm presented in Chapter VI and provides an upper bound on what's achievable in building energy simulations based on this approach. Finally, in Chapter VIII future topics and extensions of this method are discussed.

## CHAPTER II LITERATURE REVIEW

This literature review is divided into four sections: Building Energy Modeling, Constrained Optimization, previous usage of constrained optimization in HVAC, and computer science. Figure 1 shows how these three sections relate to the methods developed in this thesis and the prototype solver developed to demonstrate them. The two sections on building related topics give background for the methods developed, while the section on constrained optimization theory covers non-HVAC topics used in the modeling approach and the prototype solver. Definitions of key terms used in the rest of the thesis appear in bold.



**Figure 1 – Relationship between Literature Review Areas and Thesis Areas**

### **Building Energy Modeling**

Building **models** describe the behavior of buildings and their HVAC systems. A building energy flow model might describe a building's structure and systems. Inputs

such as outside air temperatures and internal loads allow outputs such as energy use to be calculated in a **simulation**. [6]

Most system disturbances that are relevant to a building’s overall energy usage occur at a daily or hourly level [7]. These include changes in occupancy, internal loads, and weather. Table 1 gives definitions of terms related to how a simulation progresses over time. Over a short time period, typically an hour or less, building energy flows can be approximated as steady state. **Steady state** models describe average energy flows over time periods without any dependency on previous times. HVAC control systems that operate with time constants on the order of minutes or faster require **dynamic** models that simulate the system continuously over time.

**Table 1 – Modeling Time Domains**

<b>Time Domain</b>	<b>Description</b>
Steady State	Models equilibrium conditions without time dependence
Dynamic	Models a system over time
Quasi-Dynamic	Using a series of steady state models to model a system’s behavior over time
Quasi-Steady State	Building dynamic calculations into a steady state model

When system dynamics can be ignored, a series of steady state models can be solved to simulate a model over time. This is known as **quasi-dynamic** modeling since time steps of the model don’t depend on each other. Also, slow but important system dynamics can be built into a steady state model, resulting in a **quasi-steady** model.

*Early Building Energy Analysis*

Degree-day methods [8] were developed before computer simulations became available. They provide a way to estimate a building’s thermal energy requirements by looking at the integrated outside air dry-bulb temperatures above or below a balance point temperature. Later, bin methods [9] provided a way to take building part load conditions, structure, and component level features of the HVAC system into account without having to duplicate simulations over similar operating conditions. ASEAM [10,



11] used ASHRAE’s bin calculation method so that building energy analysis could be performed on early microcomputers. Simplified energy analysis methods continue to find use in applications such as building model calibration even after more comprehensive simulation tools became available because of their speed and ability to represent key system parameters [12].

Knebel’s Modified Bin Method [9] provided a way to take part load conditions, structure, and component level features of an HVAC system into account. Algorithms in this method consist of a series of steps where variables are fixed or iterated on until the correct operating conditions are found. For instance, Figure 2 shows a Dual Duct Constant Air Volume (DDCAV) system with two zones. The algorithm of Figure 3 shows how to calculate zone temperatures in the style that Knebel presented. This algorithm is valid when the heating and cooling coils are both being used and the resulting supply fan leaving air temperature falls at or below the heating coil set point and at or above the cooling coil set point. Calculating other operating conditions requires extra steps, and algorithms based on this style are manually developed.

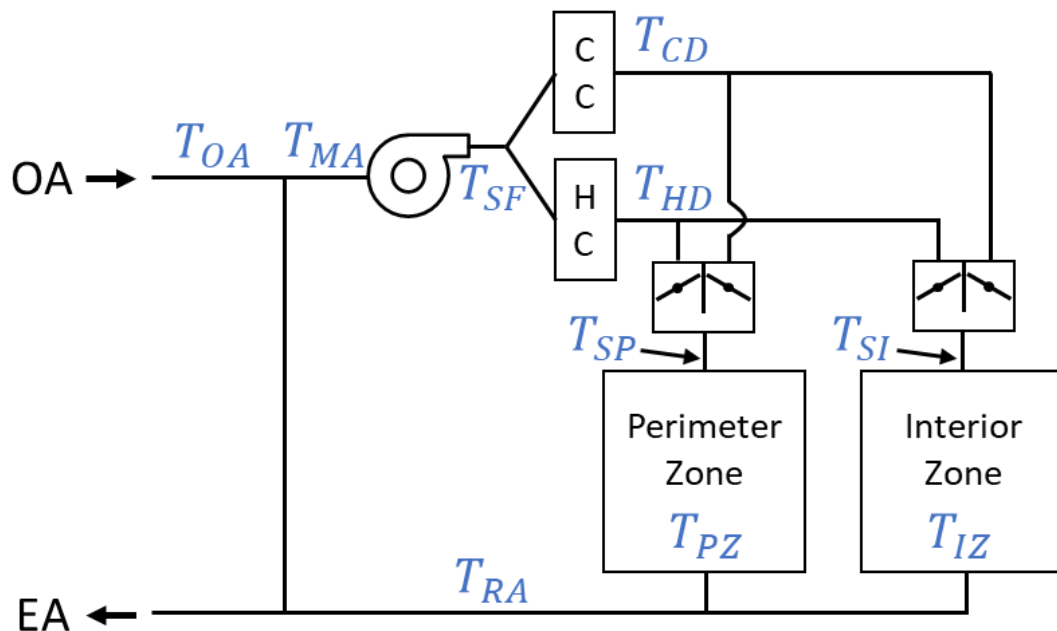
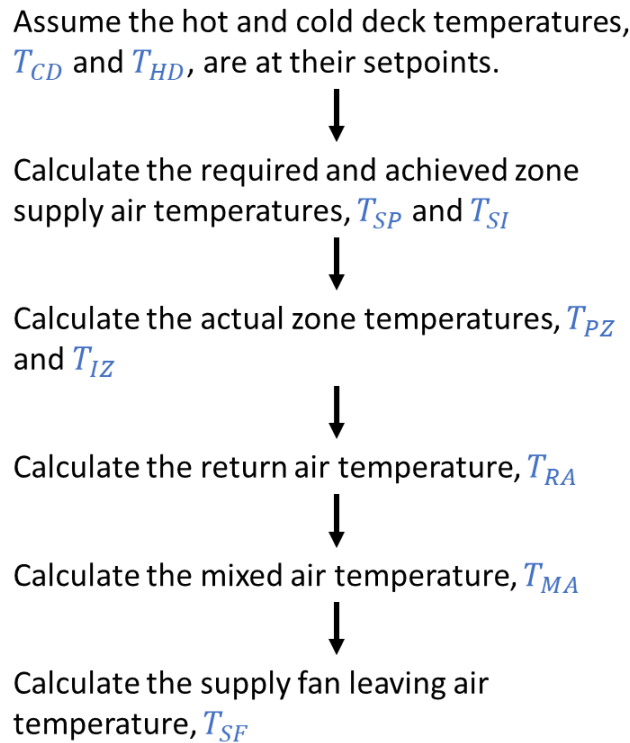


Figure 2 – Temperature Parameters for a Dual Duct Constant Air Volume System



**Figure 3 – Knebel’s Algorithm to Solve Temperatures in the DDCAV System**

*Steady State and Quasi-Steady State Simulators*

In the late 1960s to early 1970s, the U.S. Postal Service in cooperation with ASHRAE began developing algorithms and software to simulate the energy usage of multi-zoned buildings to take into account weather, solar gains, and thermal mass [13]. Many other building energy simulation programs existed in this era, but they were typically proprietary, poorly supported, and expensive to process [14]. Later, Lawrence Berkeley National Laboratory released DOE-2 in 1978 to perform whole building energy simulations using methods developed previously in the public and private sectors [15]. DOE-2 performs hourly simulations with 4 sub-modules for sequentially calculating loads, systems, plants, and economics. Its calculations use a steady state energy balance with the addition of the weighting factor method used to calculate building thermal mass effects [16].

From the early 1970s to 1995, the US ARMY Construction Engineering Research Laboratories developed the program BLAST [17]. Like DOE-2, BLAST uses a collection of sub-models to perform hourly whole building energy use calculations with three sequentially executed subprograms that compute hourly space loads, plant demands, and fuel and electric consumption [18]. Unlike DOE-2, BLAST uses a heat balance approach for zone thermal load calculations based on the first law of thermodynamics [19].

Beginning in 1994, efforts to merge BLAST and DOE-2 into a single program resulted in a whole building energy use simulator known as EnergyPlus [17]. The developers of EnergyPlus sought to merge the best capabilities of DOE-2 and BLAST while rewriting the code base in a modular style. For zone load calculations EnergyPlus uses a derivation of the heat balance approach used in IBLAST, a research version of BLAST [20]. In EnergyPlus, a high-level manager oversees the convergence of iterative sub-system calculations. By treating sub-systems independently of one another, different system-based and component-based calculations can be integrated together in an overall simulation [3]. A thorough review of the history of building energy analysis methods leading to modern methods was presented in a M.S. thesis by Oh [12].

Airflow networks consist of a set of nodes connected by airflow elements such as ducts, fans, doorways, and construction cracks [21]. Developed in 1986, COMIS [22] performs quasi-steady state pressure, air flow, and pollutant transport simulations of buildings using the airflow network method. It can be used as a stand-alone air flow model or as an infiltration module for thermal building simulation programs [23]. In 1988 AIRNET [21] was developed to model building airflow analysis and smoke/contaminant transport. The methods of AIRNET were later integrated into CONTAM [24], which can model mass accumulation and reduction processes such as humidity removal in addition air network calculations.

### *Dynamic, Equation-Based Modeling*

In 1981, Silverman, Jurovics, Low and Sowell [25] developed a concept of modeling HVAC systems as networks of interconnected components. Unlike traditional simulation programs at the time that simulated a fixed “menu” of systems, their component-based approach interconnects component equations to solve with a nonlinear optimization algorithm. Later, using methods of solving nonlinear algebraic equations developed in other industries [26], Sowell, Taghavi, Levy and Low [27] developed a program known as ENET that automatically generated source code for a particular HVAC system configuration.

To facilitate the reuse of building models developed by separate teams, a joint European/US research group created a component-based software architecture known as the Energy Kernel System, or EKS [28]. The US version of EKS, the Simulation Problem ANalysis Kernel (SPANK), evolved into the Simulation Problem Analysis Research Kernel (SPARK). Systems in SPARK consist of interconnected modules containing nonlinear differential and algebraic equations [29]. During a simulation, SPARK decomposes modular systems into a single list of equations before using graph data structures to determine the most efficient solution strategy. For systems of equations with a low level of interconnectivity, SPARK outperforms simulation programs that use sparse matrix techniques or that treat systems as interconnected modules instead of decomposing them into a single set of flattened equations. However, systems with a high degree of connectivity, like discrete 2D heat transfer, are inefficient with SPARK’s approach [30, 31].

First released in 1997, the Modelica language models large, complex, and heterogeneous systems using nonlinear differential-algebraic equations [32]. To support rapid prototyping, design, and operation of building energy and control systems, Lawrence Berkeley National Laboratory released a library of building components implemented in Modelica [4]. While being a rich language in terms of modeling complex dynamic systems, Modelica is unable to handle many concepts used in optimization such as cost functions, constraints, variable bounds, and initial guesses [5].

Other commercial component-based HVAC simulation programs also exist. HVACSIM+ [33] uses a hierarchical concept of interconnected components. Components themselves are represented with algebraic and differential equations that are solved using a nonlinear equation solver. Similarly, TRNSYS [34] simulates the dynamics of HVAC systems using component based models.

#### *Spawn of EnergyPlus*

A prototype to the successor of EnergyPlus known as Spawn-of-EnergyPlus (SOEP) is currently under development [35]. The SOEP team broke EnergyPlus into a set of component models for simulating in a discrete event simulator. These component models are decoupled from the numerical solver and can be co-simulated with models from other tools or component models generated from Modelica. This modular structure of SOEP is designed to allow the scope of building energy modeling to evolve more rapidly and robustly than its current form in EnergyPlus.

SOEP uses the Functional Mockup Interface (FMI) standard [36] to couple component models together. This standard was developed as a collaboration between simulation tool vendors and research institutes. It describes models in terms of differential, algebraic, and discrete equations with time, state, and step events. This enables C-Code to be generated in the form of an input/output block for a model. Simulations using the FMI standard couple multiple FMI components into a single simulation using master algorithms that synchronize and control the data exchange between the components.

EnergyPlus currently uses rule-based supervisory control sequences based on calculated loads. SOEP's modular structure allows models to be simulated with control schemes based on predefined and user defined control sequences. FMI components for these control functions are generated from Modelica based models. Modelica based models can also be used for new HVAC modeling while keeping existing EnergyPlus models for envelope heat transfer, lighting, and airflow.

Recently developed quantized-state system (QSS) methods are used to integrate ordinary differential equations (ODEs) [37] in SOEP. QSS methods discretize system

states and keep time as the calculated output, which allows control events to be explicitly scheduled [35]. In contrast, a classical ODE simulator that uses a constant time step size there can be uncertainty when event transitions occur.

### *Model Optimization*

GenOpt [38, 39] couples simulation programs that have a text-based input and output with mathematical optimization algorithms. It was developed for optimization problems where calculating cost functions are computationally expensive and derivatives may not exist. Available algorithms include pattern searches, particle swarm optimization, and the simplex algorithm of Nelder and Mead.

While Modelica simulates differential-algebraic equations, Optimica [5, 40] extends the Modelica language to handle steady state and dynamic optimization problems. For steady state optimizations, derivatives of dynamic Modelica models can be set to zero. For dynamic optimizations, Optimica automatically transcribes continuous variables into discrete problems for solving with a steady state nonlinear programming algorithm.

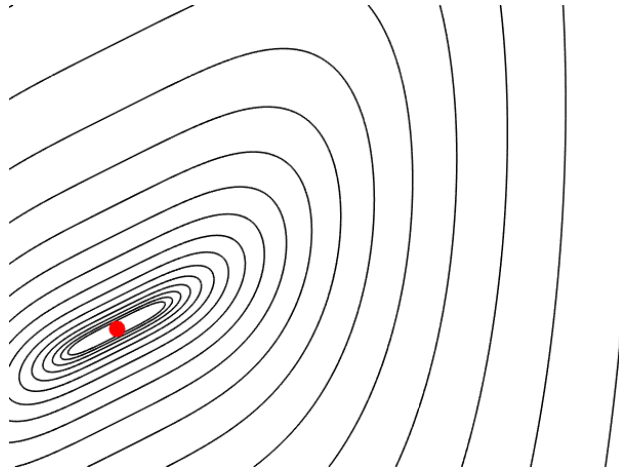
### **Constrained Optimization Theory**

This thesis demonstrates a method of performing steady state building energy modeling using constrained optimization as well as a prototype simulation engine. Both depend on constrained optimization solvers and theory for their performance. This section discusses the constrained optimization theory, its history, as well as minor subfields that have use in building energy modeling.

### *Optimization Overview*

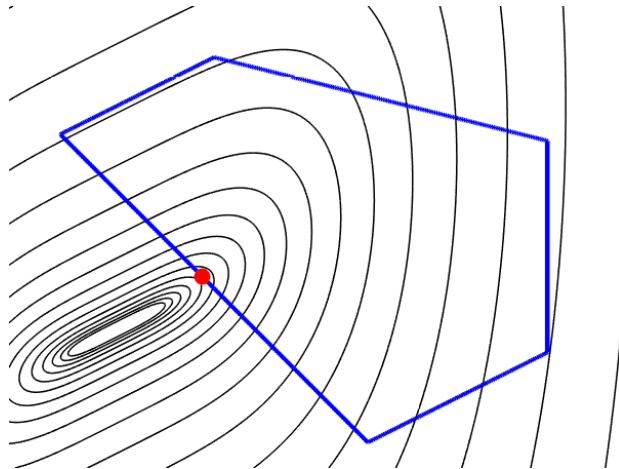
### **Unconstrained and Constrained Optimization**

Unconstrained optimization finds a minimum or maximum of a function. Figure 4 shows an unconstrained optimization, where contour lines show the level curves of a function and the red dot marks the optimal value.



**Figure 4 – An unconstrained optimization**

Constrained optimization optimizes a function subject to constraints on the variables. Figure 5 shows a constrained optimization with the same optimization goal contours as Figure 4 except that the variables are constrained to lie within the blue polygon.



**Figure 5 – A constrained optimization**

### **Linear and Nonlinear Programming**

Constrained optimization can be divided into linear and nonlinear programming based on the type of objective function and constraints used. Linear programs, with a standard form shown in Figure 6, optimize a linear function subject to linear constraints.

Nonlinear programs, with a form of Figure 7, can have any functions for the objective or constraints. Figure 8 gives an alternate form of nonlinear program that any problem in the form of Figure 7 can be reformulated into. Solvers such as IPOPT [41] use the form of Figure 8 as inputs. Nonlinear programs are parametric when a sequence of nonlinear problems is solved with parameters that vary on each simulation. Sensitivity analysis [42] can be used in parametric nonlinear programming to estimate the solution on one time step from the solution of the previous time step.

minimize:	$\mathbf{c}^T \mathbf{x}$
subject to:	$\mathbf{Ax} \leq \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$

**Figure 6 – Canonical Form of Linear Programs**

minimize:	$f(\mathbf{x})$
subject to:	$\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ $\mathbf{h}(\mathbf{x}) = \mathbf{0}$

**Figure 7 – Form of a Nonlinear Program**

minimize:	$f(\mathbf{x})$
subject to:	$\mathbf{h}(\mathbf{x}) = \mathbf{0}$ $\mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}$

**Figure 8 – Box-Bound Form of a Nonlinear Program**

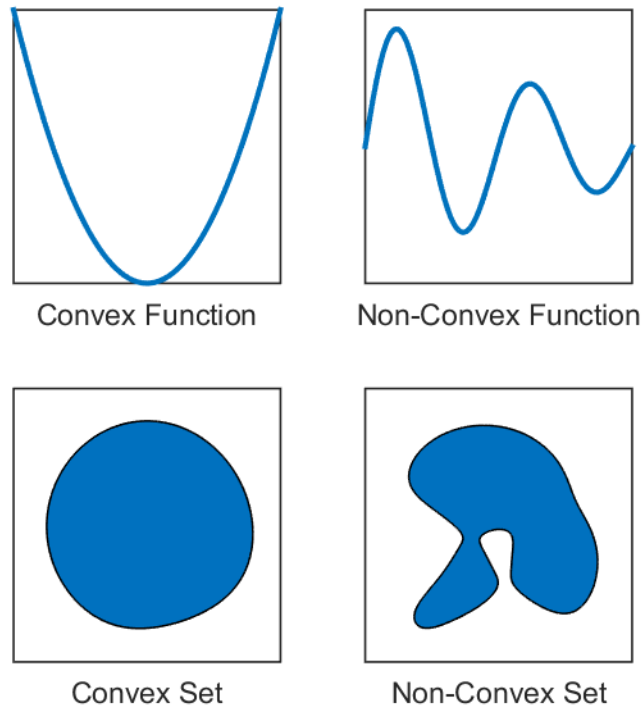
### Convex and Non-Convex Optimization

Convex optimization problems [43] minimize convex functions over convex sets, shown in Figure 9. The properties of convex functions and sets guarantee that all local minima to a convex optimization problem are also global minima, which makes them easier to solve in general than non-convex optimizations. Formulating an HVAC system



model with convexity in mind helps ensure the model's robustness and accuracy when it's simulated in a numerical solver.

Specialized branches of convex optimization with well-developed theory and algorithms also exist. For instance, piecewise linear programming [44] extends linear programming by optimizing a piecewise linear function. Quadratic programming [45] finds the optimal value of a quadratic function subject to linear inequality constraints. These branches can be used to efficiently approximate more complex problems or as sub-problems in non-convex optimization algorithms.



**Figure 9 – Convex and Non-Convex Functions and Sets**

### *Research History*

Pierre de Fermat first developed a method to find minima and maxima of functions [46]. He found that local minima of an unconstrained function occur where the gradient equals zero. In 1788 Joseph-Louis Lagrange extended this to functions constrained by equality constraints with the theory of Lagrange multipliers [47]. In 1939

William Karush extended this to inequality constraints [48], which was rediscovered in 1951 by Harold Kuhn and Albert Tucker [49]. These conditions are known as the Karush-Kuhn-Tucker conditions, or KKT conditions. Table 2 summarizes the solvability conditions for these different types of problems.

**Table 2 – Local Minima for Different Problem Types**

Problem		Local Minima Conditions	Discovered
minimize:	$f(\mathbf{x})$	$\nabla f(\mathbf{x}) = \mathbf{0}$	1636 (Fermat)
minimize:	$f(\mathbf{x})$	$\nabla f(\mathbf{x}) + \sum_{j=1}^n \lambda_j \nabla h_j(\mathbf{x}) = \mathbf{0}$ $h_j(\mathbf{x}) = 0, \text{ for all } j = 1, \dots, l$	1788 (Lagrange)
subject to:	$\mathbf{h}(\mathbf{x}) = \mathbf{0}$		
minimize:	$f(\mathbf{x})$	$\nabla f(\mathbf{x}) + \sum_{i=1}^m \mu_i \nabla g_i(\mathbf{x}) + \sum_{j=1}^n \lambda_j \nabla h_j(\mathbf{x}) = \mathbf{0}$ $g_i(\mathbf{x}) \leq 0, \text{ for all } i = 1, \dots, m$ $h_j(\mathbf{x}) = 0, \text{ for all } j = 1, \dots, l$ $\mu_i \geq 0, \text{ for all } i = 1, \dots, m$ $\mu_i g_i(\mathbf{x}) = 0, \text{ for all } i = 1, \dots, m$	1938 (Karush) and 1951 (Kuhn/Tucker)
subject to:	$\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ $\mathbf{h}(\mathbf{x}) = \mathbf{0}$		

In 1939 Leonid Kantorovitch of the USSR began researching the use of linear programming for economics. This resulted in him winning 1975 Nobel Prize in Economics [50]. At the time, economic planning decisions were made based on ad-hoc rules and people manually working to find solutions. The first algorithm to solve these problems effectively, the simplex method for linear programming, was developed in 1947 by Dantzig [51]. The simplex method allowed for economic decisions with thousands of variables to be calculated exactly instead of using ad-hoc rules.

Initially, research into linear and nonlinear programming developed along different paths [52]. Then, in 1984, Karmarkar [53] introduced a class of algorithms known as interior point methods. Since the 1990s, interior point methods have been

competitive for solving linear programs as well as more general problems such as convex optimizations.

*Specialized Branches of Optimization*

**Multi-Objective and Lexicographic Optimization**

Multi-objective optimization problems involve optimizing multiple, competing objectives at the same time. Pareto optimal solutions are points where all objectives cannot be made better without one being made worse, and preferences of which objectives to maximize over others dictate which Pareto optimal solution to take as the final overall solution [54]. Goal programming [55] extends linear programming for multiple, usually conflicting objectives.

Nonlinear programs in the form of Figure 7 optimize a single objective function subject to constraints. However, HVAC control systems often have a hierarchy of objectives. For instance, a control system might first seek to maintain a zone’s temperature and as a lower priority keep the relative humidity below a certain percentage. Lexicographic optimization problems, seen in Figure 10, minimize a sequence of objective functions  $f_1(x), f_2(x), \dots, f_k(x)$  subject to constraints. [56]

minimize: (in order)	$f(x) = [f_1(x), f_2(x), \dots, f_k(x)]^T$
subject to:	$g(x) \leq 0$ $h(x) = 0$

**Figure 10 – A Lexicographic Optimization Problem**

Lexicographic optimization problems can be decomposed into a sequence of single-objective optimizations where the minimized objective of a higher goal becomes a constraint to later goals [57]. Figure 11 shows this process, where a new optimal value is calculated for each objective. After performing an optimization for one stage, the optimized objective value and function constrains later stages. The solution to the last stage is taken to be the overall result.

Stage 1	minimize	$f_1(x_1)$
	subject to	$\mathbf{g}(x_1) \leq \mathbf{0}$ $\mathbf{h}(x_1) = \mathbf{0}$
Stage 2	minimize	$f_2(x_2)$
	subject to	$\mathbf{g}(x_2) \leq \mathbf{0}$ $\mathbf{h}(x_2) = \mathbf{0}$ $f_1(x_2) = f_1(x_1)$
⋮	⋮	
Stage k	minimize	$f_k(x_k)$
	subject to	$\mathbf{g}(x_k) \leq \mathbf{0}$ $\mathbf{h}(x_k) = \mathbf{0}$ $f_1(x_k) = f_1(x_1)$ ⋮ $f_{k-1}(x_k) = f_{k-1}(x_{k-1})$

**Figure 11 – Lexicographic Decomposition**

### Non-Convex Optimization

In an HVAC model, a single nonlinear fan curve or VAV zone temperature balance makes the optimization problem non-convex. This requires the use of non-convex solution algorithms. In one class of non-convex solution algorithms, penalty and interior point methods, constrained problems are converted to unconstrained problems by modifying the cost function [58]. In the context of an HVAC simulation, penalty methods start out by allowing variables like VAV flow rates to hold any value, but gradually ensure that constraints hold as the simulation goes on. Conversely, in interior point methods all variables hold valid values in the beginning, but as the calculation goes on the solution is allowed to get closer to the constraint boundary.

Solvers may also use convex approximations of non-convex problems to take advantage of convex optimization algorithms [43]. Sequential quadratic programming

(SQP), used in Ma et al. [59], iteratively finds a solution to nonlinear and non-convex optimization problems. On each iteration it approximates the optimization problem as a quadratic program around the current best solution. Then it uses this quadratic program to calculate a more optimal solution [60].

### **Integer and Global Optimization**

Integer programming and mixed integer programming [61] refer to the cases when all or some parameters must be integers. For example, in the area of HVAC modeling a control sequence that turns multiple chillers on or off can be formulated as a mixed integer program. To find a known global optimal solution to integer programs, branch and bound algorithms [62] divide up a search space, find upper and lower bounds of these subspaces, and prune spaces with than cannot contain the optimal solution from further consideration.

### **Constrained Optimization in HVAC**

#### *Control Optimization*

Constrained optimization finds use in optimizing HVAC control set points. Zheng and Zaheer-Uddin [63] modeled a chilled water loop connected to a variable air volume AHU using steady state nonlinear equations with nonlinear constraints. With 10 control variables and discrete intervals of 10 minutes, they were able to show the benefits of humidity control and optimizing outside air usage in terms of air quality and efficiency. Later, Zheng [64] developed a strategy for constructing dynamic HVAC models for use in control optimization. Models created with this strategy were used to investigate the global optimal operation of CAV and VAV AHUs.

#### *Model Predictive Control*

Model predictive control (MPC) uses a plant model and an optimization routine to choose an optimal control strategy based on predicted future conditions. Mathematically, the optimization routines it performs are very similar to the type of simulation proposed for this thesis since both use constrained optimization applied to HVAC models. Where examples in this thesis use constrained optimization to solve for

instantaneous steady state control conditions, MPC optimizes a control cost function as a system operates over time to select an immediate control strategy.

Ma et al. [65] applied MPC to a cooling thermal energy storage system. In their MPC control strategy, every hour command signals change based on an optimal control problem that looks 24 hours into the future. Later, [66] Ma et al. refined their method by expanding the system model to incorporate a model of the buildings that the cooling system serves that includes thermal mass modeling, solar modeling, and a fan coil unit model. After testing their control method in a real central plant, operators manually applied the calculated optimal strategy to the plant to improve its COP by 11.9%.

In addition, this approach was tailored by Ma et al. to control a single duct VAV AHU [59]. The system model neglected dynamics for the AHU components while using a RC network model for zonal thermal capacitance. Discretizing the system dynamics over a 24-hour time period combined with constraints on operating parameters resulted in a nonlinear program. This was solved using a sequential quadratic programming algorithm. In Ma et al. [67] statistical distributions of parameter uncertainties and allowable violations of constraints results were taken into account, resulting in a stochastic model predictive control problem. This problem was solved using a customized interior point method that took advantage of the problem's overall sparseness. The analysis of Kelman et al [68] on the optimization algorithms used for the single duct VAV system as well as a dual duct CAV system showed that local minima occur that correspond to different system control strategies. As system complexity grows, the number of local minima increase making global optimizations more difficult.

## CHAPTER III

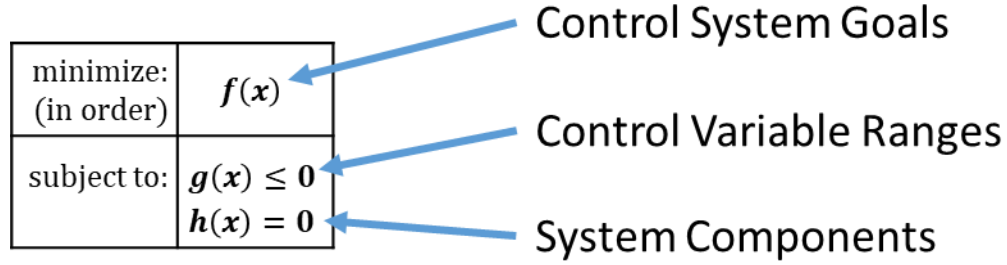
### STEADY STATE MODELING

Steady state building energy modeling directly represents building and HVAC operating conditions under static equilibrium. These models can be useful in estimating building energy usage, since energy usage depends on fluid flows and equipment operation that varies slowly over time more than control dynamics. Also, ignoring the effects of control dynamics allows time steps of an hour or longer to be used. However, steady state algorithms used in commercial software are complex, monolithic, and can't take advantage of symbolic simplifications, making an equation-based approach desirable.

This chapter presents a method of formulating steady state building energy models using constrained optimization. Models can be created by piecing together algebraic equipment models and defining functions to optimize so that the system behaves identically to a dynamic model that has reached steady state. Optimization functions can be created from a qualitative description of the control system, so detailed descriptions of control system dynamics aren't required.

#### **Constrained Optimization Based Modeling**

Formulating a steady state building energy model requires knowledge of the building, HVAC system equipment, and control strategy. Figure 12 shows where this information fits in the equality constraints, inequality constraints, and objectives of a nonlinear program. Equality constraints in this approach describe the relations between state variables from components such as fans, zones, and cooling coils. Inequality constraints describe limitations on the controlled variables. Finally, the optimization objective describes control system goals, where minimizing the objective results in the steady state operating conditions of the system.



**Figure 12 – Using Lexicographic Nonlinear Programming for Steady State HVAC**

### Modeling

This approach focuses on the goals that a control system seeks to achieve and ignores details of the control system’s mechanical operation and control algorithms. Examples of common control system goals include zone temperature control, humidity control, and minimizing air flow or energy costs. The mathematical functions to use can be derived from a description of the system’s operation, as shown in the models of Chapter V. These models are useful for estimating building energy usage, since energy usage depends on fluid flows and equipment operation that varies slowly over time more than control dynamics.

### A Simple System

Figure 13 shows a Single-Zoned Single Duct Constant Air Volume (SZCAV) system with reheat and a single zone. This system controls the zone temperature to its set point by reheating a constant flow of supply air, where the supply air is cooled by a cooling coil with a fixed temperature set point. The simplified model of this system used here assumes a constant temperature rise across the fan, infinite cooling and reheat coil capacities, and that the zone has no thermal mass. Table 3 gives the parameters used in modeling this system, with volume flow rates, the UA value, and loads normalized to the zone’s floor area. Variables normalized to floor area are written with an overbar.



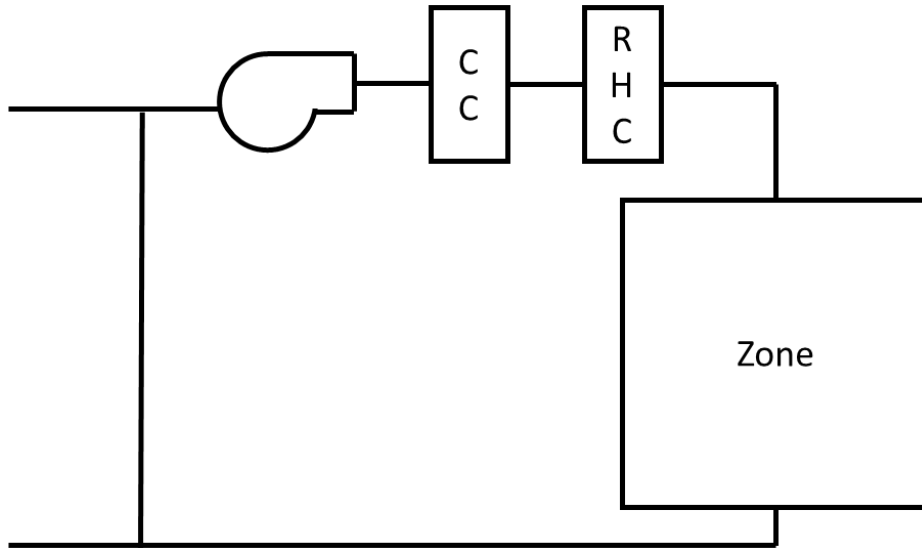


Figure 13 – A Single-Zoned Single Duct Constant Air Volume System

Table 3 – Parameters for the SZCAV Example

Parameter	Symbol	Value	Unit
Zone Set Point Temperature	$T_{ZA,SP}$	70	$^{\circ}F$
Outside Air Flow per ft <sup>2</sup> of Floor Area	$\bar{V}_{OA}$	0.2	$\frac{CFM}{ft^2}$
Supply Fan Delta-T	$\Delta T_{SF}$	2.0	$^{\circ}F$
Cooling Coil Set Point Temperature	$T_{CC,SP}$	55	$^{\circ}F$
Air Density Times Specific Heat	$\rho c_p$	1.08	$\frac{Btu}{h \cdot ^{\circ}F \cdot CFM}$
Wall UA Value per ft <sup>2</sup> of Floor Area	$\bar{UA}$	0.6	$\frac{Btu}{h \cdot ft^2 \cdot ^{\circ}F}$
Zone Load per ft <sup>2</sup> of Floor Area	$\bar{Q}_s$	6.0	$\frac{Btu}{h \cdot ft^2}$
Supply Air Flow per ft <sup>2</sup> of Floor Area	$\bar{V}_{SA}$	1.0	$\frac{CFM}{ft^2}$
Outside Air Temperature	$T_{OA}$	50	$^{\circ}F$

### Equality Constraints Alone

Equations 3.1–3.5 below give the energy balance equations for calculating temperatures in the model. Equation 3.1 gives the mixed air temperature balance, equation 3.2 gives the supply fan leaving air temperature, equation 3.3 gives the cooling coil leaving air temperature, equation 3.4 gives the supply air temperature from the cooling coil leaving air temperature and the amount of reheat used, and equation 3.5 gives the zone energy balance. These five equations have six unknown variables: the mixed air temperature  $T_{MA}$ , the supply fan leaving air temperature  $T_{SF}$ , the cooling coil leaving air temperature  $T_{CC}$ , the reheat coil energy usage  $\bar{Q}_{RH}$ , the supply air temperature  $T_{SA}$ , and the zone temperature  $T_{ZA}$ .

$$T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{ZA} \quad 3.1$$

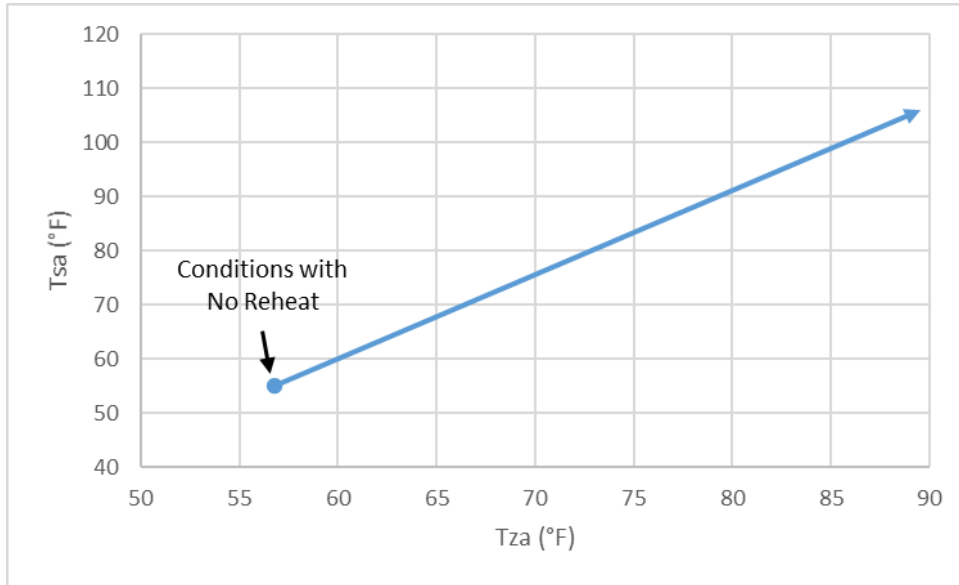
$$T_{SF} = T_{MA} + \Delta T_{SF} \quad 3.2$$

$$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP}) \quad 3.3$$

$$\bar{Q}_{RH} = \rho c_p \bar{V}_{SA}(T_{SA} - T_{CC}) \quad 3.4$$

$$\rho c_p \bar{V}_{SA}(T_{SA} - T_{ZA}) + \bar{U}\bar{A}(T_{OA} - T_{ZA}) + \bar{Q}_S = 0 \quad 3.5$$

The five equations and six unknown variables of the example model result in an underdefined set of equations because they don't take the control system's actions into account. These equations can be reduced to describe any variable in terms of another. Figure 14 shows the supply air temperatures required to achieve different zone temperatures for the given operating conditions. The leftmost point on the ray represents the system state at the lowest achievable supply air temperature, 55°F, which occurs when no reheat is used. As the supply air temperature increases the zone temperature increases.



**Figure 14 – SZCAV Zone Temperature vs. the Required Supply Air Temperature**

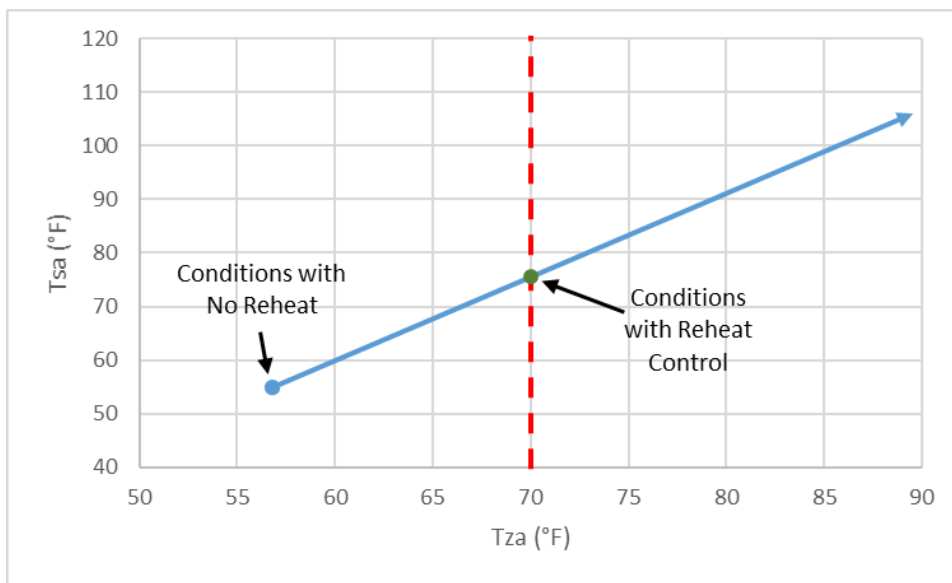
*A Constrained Optimization Model*

The SZCAV example system reheats air to control zone temperature by controlling the reheat coil’s set point temperature. This can be represented mathematically using constrained optimization by adding an inequality constraint and an objective function to equations 3.1–3.5 above, resulting in the model of Figure 15. The inequality constraint in equation 3.12 of this model limits the reheat coil set point when insufficient cooling exists. The objective function of equation 3.6 ensures that the used reheat coil set point achieves the zone temperature set point to the highest degree possible.

minimize:	$ T_{ZA} - T_{ZA,SP} $	3.6
subject to:	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{ZA}$	3.7
	$T_{SF} = T_{MA} + \Delta T_{SF}$	3.8
	$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP})$	3.9
	$\bar{Q}_{RH} = \rho c_p \bar{V}_{SA}(T_{SA} - T_{CC})$	3.10
	$\rho c_p \bar{V}_{SA}(T_{SA} - T_{ZA}) + \bar{U}A(T_{OA} - T_{ZA}) + \bar{Q}_S = 0$	3.11
	$\bar{Q}_{RH} \geq 0$	3.12

**Figure 15 – Model of Single-Zoned Single Duct Constant Air Volume Temperatures**

Figure 16 shows the zone temperature versus supply temperature for the same operating conditions used in Figure 14. The equality and inequality constraints define the 1D ray of operating points that the system can achieve. The leftmost point on the ray gives the zone and supply air temperature when no reheat is used, and the dotted red line marks the zone temperature set point of 70°F. The intersection of the ray and the dotted line gives the zone and supply air temperatures when reheat is used to control the space at an outside air temperature of 50°F.



**Figure 16 – SDVAV Zone Temperature vs. Supply Air Temperature**

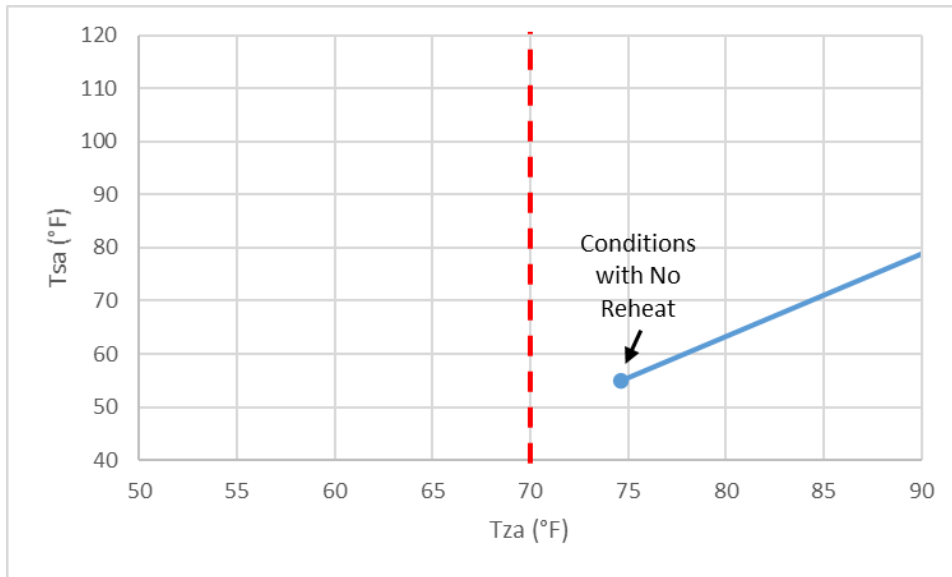
### *Atypical Operating Conditions*

Manually developed building energy modeling algorithms often assume that systems operate under typical operating conditions. For instance, DOE-2 assumes that zone temperatures operate at their set point for load calculations. Applying this assumption to equations 3.1–3.5 results in a model with an equal number of equations and unknown variables, but this is only valid when the system has enough cooling capacity. This model would be incomplete since it's possible for the initial assumptions of the model to be violated. For instance, a system without enough cooling capacity may

have zones that rise above their set point temperatures. A complete model would correctly represent atypical operating conditions as well as normal operating conditions.

Having complete models that represent atypical operating conditions is important because the process of developing, calibrating, and optimizing a model can easily result in atypical simulations that need to be refined. Even if a building never experiences atypical operating conditions in real life, invalid inputs or bad guesses for calibrations and optimizations can cause a simulation where a system is unable to handle zone loads or equipment is operated outside of its valid operating range. Complete models can also be debugged more easily by getting rid of the need to manually account for a model's faulty assumptions. For instance, if the model of Figure 15 has too little cooling capacity and a zone temperature of 85°F, a complete model would show the hot zone while an incomplete model that assumes the zone is at the correct set point would not balance out.

Constrained optimization based models provide a natural way to describe the complete operation of a system. This is because systems minimize their optimization goals, but only as far as the constraints of the system allows. For example, the model of Figure 15 correctly models situations when the zone temperature setpoint cannot be achieved. Figure 17 shows this system for an outside air temperature of 100°F. In this case the system cannot achieve the zone set temperature point. However, the constrained optimization problem takes this into account by cooling the zone as much as it can, resulting in the leftmost point on the ray as the solution.



**Figure 17 – A SDCAV System That Can’t be Controlled**

### **Modeling with Lexicographic Objectives**

The SZCAV system of the previous section can be modified into a Single-Zoned Single Duct Variable Air Volume (SZVAV) system by changing the controls and using a variable speed drive. Figure 18 shows a constrained optimization formulation of this model with objectives that have a priority. This lexicographic optimization uses the parameters from the SDCAV example in Table 3 as well as new parameters given in Table 4. Loads in this model are constant, but quasi-dynamic simulations can be performed by treating loads as time dependent inputs and repeating calculations for each time step.

minimize: (in order)	$Max(0, T_{ZA,HeatingSP} - T_{ZA}) + Max(0, T_{ZA} - T_{ZA,CoolingSP})$	3.13
	$\bar{V}_{SA}$	3.14
	$\bar{Q}_{RH}$	3.15
subject to:	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{ZA}$	3.16
	$T_{SF} = T_{MA} + \Delta T_{SF,Max} \left( \frac{\bar{V}_{SA}}{\bar{V}_{SA,Max}} \right)^2$	3.17
	$T_{CC} = Min(T_{SF}, T_{CC,SP})$	3.18
	$\bar{Q}_{RH} = \rho c_p \bar{V}_{SA} (T_{SA} - T_{CC})$	3.19
	$\rho c_p \bar{V}_{SA} (T_{SA} - T_{ZA}) + \bar{U}A(T_{OA} - T_{ZA}) + \bar{Q}_S = 0$	3.20
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA}$	3.21
	$\bar{V}_{SA} \leq \bar{V}_{SA,Max}$	3.22
	$\bar{Q}_{RH} \geq 0$	3.23

**Figure 18 – Single-Zoned Variable Air Volume Temperature Balance Equations**

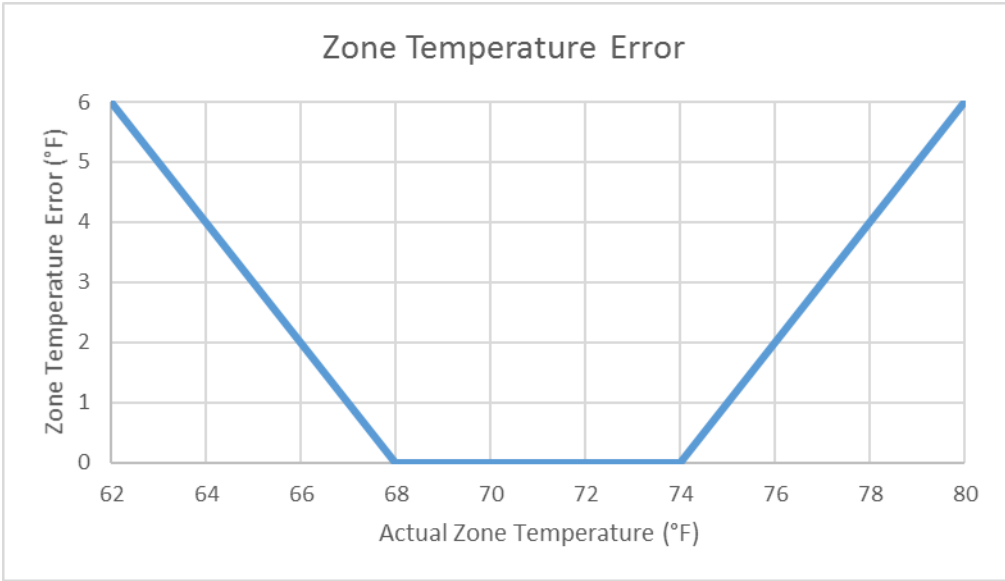
**Table 4 – Additional Parameters for the SZVAV Example**

Parameter	Symbol	Value	Unit
Zone Heating Set Point Temperature	$T_{ZA,HeatingSP}$	68	$^{\circ}F$
Zone Cooling Set Point Temperature	$T_{ZA,CoolingSP}$	74	$^{\circ}F$
Maximum Supply Fan Delta-T	$\Delta T_{SF,Max}$	2.0	$^{\circ}F$
Minimum Supply Air Flow	$\bar{V}_{SA,Min}$	0.4	$\frac{CFM}{ft^2}$
Maximum Supply Air Flow	$\bar{V}_{SA,Max}$	1.0	$\frac{CFM}{ft^2}$

Three changes are made to the constraints. First, the supply air flow is allowed to vary between min and max values instead of being held at a constant value. The inequalities of equations 3.21 and 3.22 represents this. Second, a variable speed drive causes the temperature rise across the supply fan to vary as a function of its flow, which is reflected in equation 3.17. Finally, reheat usage is explicitly written into equations 3.19 and 3.23, and negative reheat usage isn't allowed. This lets the supply air temperature float above the cooling coil leaving air temperature to provide zone temperature control.

The system controls the fan and reheat coil to use as little electricity and heating as possible while keeping the zone temperature as close as possible to the zone heating/cooling setpoint range. This involves three lexicographic objective functions: equation 3.13 for the zone temperature error and, equation 3.14 to minimize the air flow rate, and equation 3.15 to minimize the amount of reheat use. Figure 19 shows a graph of the zone temperature error function from the objective in equation 3.13. It has a value of zero between the heating and cooling set points and increases linearly away from them.

A powerful feature of this lexicographic formulation is that the problem can be described verbally based on a description of what the control system needs to achieve. The control system's priorities determine the ordering of lexicographic objectives. For this model minimizing the zone temperature error in equation 3.13 is prioritized over minimizing the air flow rate in equation 3.14 and reheat usage in equation 3.15 since the purpose of the system is temperature control. After that, equations 3.14 and 3.15 can be ordered either way since both result in the same answer.



**Figure 19 – Zone Temperature Error as a Function of Heating and Cooling Set Points**



This lexicographic model can be solved in three stages, one for each objective [57]. Figure 20, Figure 21, and Figure 22 show the problems solved for each stage. The first stage minimizes the zone temperature error, the second stage minimizes the supply air flow rate subject to the minimized temperature error from the previous stage, and the third stage minimizes reheat usage subject to the minimized zone temperature error and supply air flow. The ordering of these objectives causes the model to prioritize temperature control first, supply air flow second, and reheat usage third. Temperature control has the highest priority since controlling zone conditions is the main purpose of HVAC systems. This system uses as little air flow and reheat as possible, but never at the expense of a zone that's too hot or cold.

minimize:	$f_1 = \text{Max}(0, T_{ZA,HeatingSP} - T_{ZA}) + \text{Max}(0, T_{ZA} - T_{ZA,CoolingSP})$	3.24
subject to:	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{ZA}$	3.25
	$T_{SF} = T_{MA} + \Delta T_{SF,Max} \left( \frac{\bar{V}_{SA}}{\bar{V}_{SA,Max}} \right)^2$	3.26
	$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP})$	3.27
	$\bar{Q}_{RH} = \rho c_p \bar{V}_{SA} (T_{SA} - T_{CC})$	3.28
	$\rho c_p \bar{V}_{SA} (T_{SA} - T_{ZA}) + \bar{U}\bar{A}(T_{OA} - T_{ZA}) + \bar{Q}_S = 0$	3.29
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA}$	3.30
	$\bar{V}_{SA} \leq \bar{V}_{SA,Max}$	3.31
	$\bar{Q}_{RH} \geq 0$	3.32

**Figure 20 – Stage 1 of the SZVAV Model**

minimize:	$f_2 = \bar{V}_{SA}$	3.33
subject to:	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{ZA}$	3.34
	$T_{SF} = T_{MA} + \Delta T_{SF,Max} \left( \frac{\bar{V}_{SA}}{\bar{V}_{SA,Max}} \right)^2$	3.35
	$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP})$	3.36
	$\bar{Q}_{RH} = \rho c_p \bar{V}_{SA} (T_{SA} - T_{CC})$	3.37
	$\rho c_p \bar{V}_{SA} (T_{SA} - T_{ZA}) + \bar{U}A(T_{OA} - T_{ZA}) + \bar{Q}_S = 0$	3.38
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA}$	3.39
	$\bar{V}_{SA} \leq \bar{V}_{SA,Max}$	3.40
	$\bar{Q}_{RH} \geq 0$	3.41
	$\text{Max}(0, T_{ZA,HeatingSP} - T_{ZA}) + \text{Max}(0, T_{ZA} - T_{ZA,CoolingSP}) = f_1$	3.42

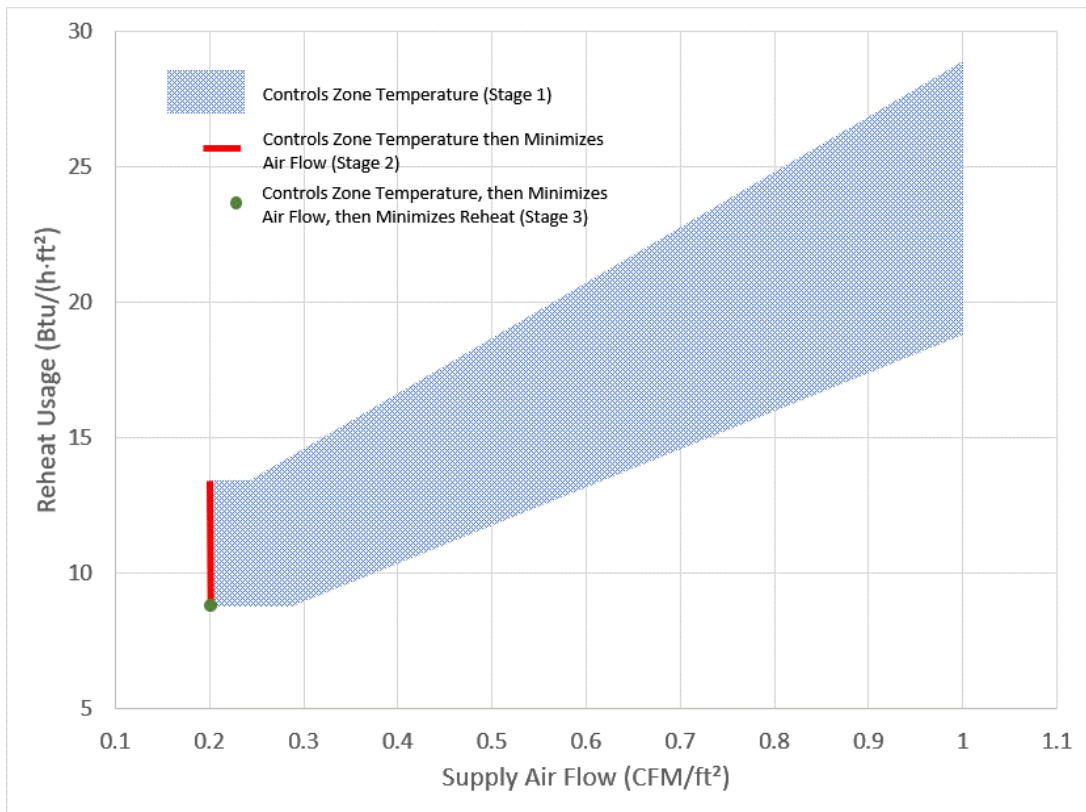
**Figure 21 – Stage 2 of the SZVAV Model**

minimize:	$\bar{Q}_{RH}$	3.43
subject to:	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{ZA}$	3.44
	$T_{SF} = T_{MA} + \Delta T_{SF,Max} \left( \frac{\bar{V}_{SA}}{\bar{V}_{SA,Max}} \right)^2$	3.45
	$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP})$	3.46
	$\bar{Q}_{RH} = \rho c_p \bar{V}_{SA} (T_{SA} - T_{CC})$	3.47
	$\rho c_p \bar{V}_{SA} (T_{SA} - T_{ZA}) + \bar{U}A(T_{OA} - T_{ZA}) + \bar{Q}_S = 0$	3.48
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA}$	3.49
	$\bar{V}_{SA} \leq \bar{V}_{SA,Max}$	3.50
	$\bar{Q}_{RH} \geq 0$	3.51
	$\text{Max}(0, T_{ZA,HeatingSP} - T_{ZA}) + \text{Max}(0, T_{ZA} - T_{ZA,CoolingSP}) = f_1$	3.52
	$\bar{V}_{SA} = f_2$	3.53

**Figure 22 – Stage 3 of the SZVAV Model**

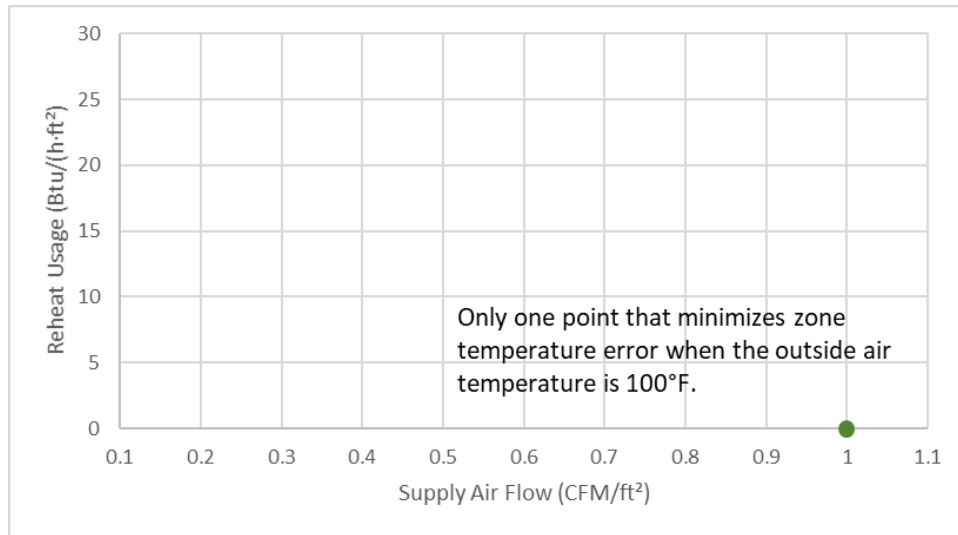
These calculations can be shown visually in terms of the two controlled parameters: the supply air temperature and the amount of reheat performed. Fixing both parameters results in a set of equations that can be solved for a unique solution. Figure 23 shows all possible states that satisfy the three stages of the solution from Figure 20. The non-convex polytope that ranges from  $0.2 \frac{CFM}{ft^2}$  to  $1.0 \frac{CFM}{ft^2}$  gives the range of all valid supply air flows and reheat usages that ensure temperature control of the space. The solution of the stage 1 problem can lie anywhere within this space. The stage 2

algorithm minimizes the supply air flow while only considering points that are valid solutions to stage 1. This solves for the supply air flow rate that's used. Finally, stage 3 minimizes the reheat usage subject to the solutions of stages 1 and 2. This reduces the solution to a single point and gives the steady state solution to the overall problem.



**Figure 23 – Supply Air Flow and Reheat Usages that Meet Zone Temperature Set Points for the SZVAV Model**

Temperature control can only be achieved for the SZVAV system when the outside air temperature is less than 100°F. In this case the solution to stage 1 of the model is a single point that achieves a zone temperature of 74.6 °F. Stages 2 and 3 reach the same solution since stage 1 reduces space of possible solutions to a single point.



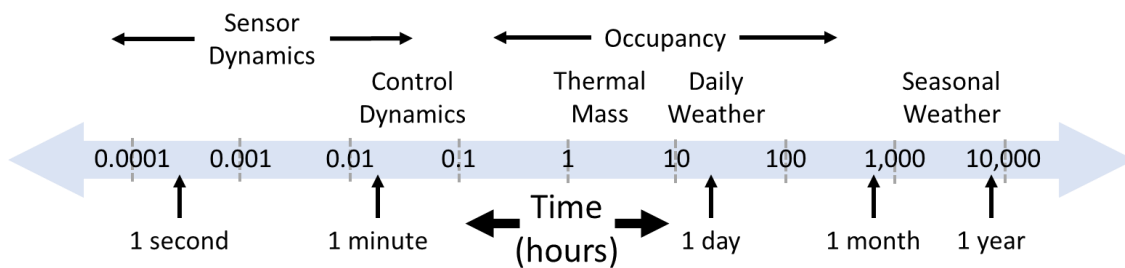
**Figure 24 – Solution to Stages 1, 2, and 3, for when the Outside Air Temperature is 100°F**

CHAPTER IV  
INTEGRATING DYNAMIC MODELING

Dynamic processes in a building can have a large effect on its energy use. For example, thermal mass dampens and delays the impact of outside air conditions, which can reduce cooling and heating loads by 20% [69]. In this case steady state assumptions can't be made for an entire system. However, modeling steady state and dynamic parts of a system together can give the speed benefits of steady state building energy use modeling without having to use small time steps. The following process describes how to create hybrid steady state/dynamic models by embedding dynamic modeling within a steady state constrained optimization model.

**Dynamic Building Processes**

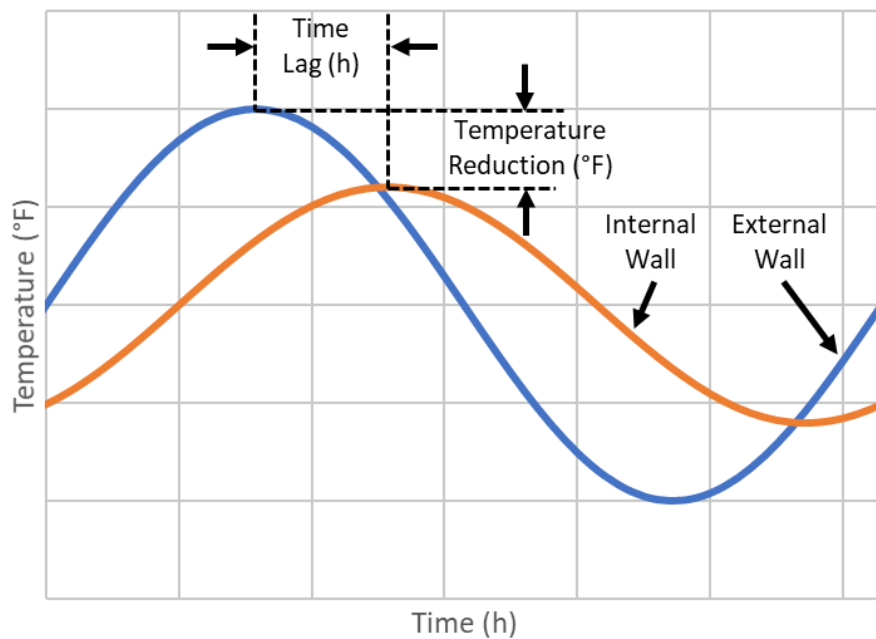
Processes in buildings take place over a wide range of time scales. Weather follows a daily cycle, walls have time constants on the order of minutes hours, and HVAC controls can have time constants on the order of seconds to minutes. Figure 25 shows approximate time scales that different HVAC processes act on.



**Figure 25 – Time Scales of HVAC Processes**

Ignoring control dynamics allows steady state building energy models to use time steps of an hour or more. This reduces the time required to perform simulations over an entire year of weather and operating conditions. However, thermal mass dynamics take place on a larger time scale and may be too important to ignore.

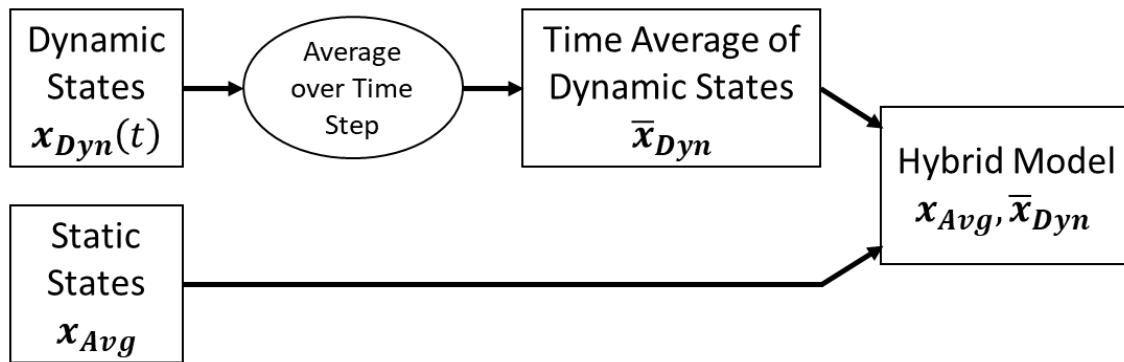
Figure 26 shows the effect of thermal mass on the internal and external surfaces of an exterior wall. A wall's outside surface temperature cycles over a 24-hour period because of outside air conditions, and thermal mass causes the internal surface temperature to lag. The time frame of this lag can be on the order of hours. This reduces the temperature variation of the interior wall, which tends to reduce the energy required to condition the interior space.



**Figure 26 – Temperature Lag and Dampening due to Thermal Mass on an Exterior Wall**

## Hybrid Steady State/Dynamic Modeling

Steady state models directly represent average conditions over time step periods while the states of dynamic models states change continuously with time. Dynamic and steady state models can be coupled together by balancing energy and mass flows over each steady state time step period. This results in a hybrid steady state/dynamic model. Figure 27 shows this process.



**Figure 27 – Constructing a Hybrid Model from Static and Dynamic States**

Mathematically, hybrid models can be represented as parametric, lexicographic, constrained optimizations that include systems of differential-algebraic equations and conservation equations, as seen in Figure 28. In this form the  $x_{Avg}$  vector represents the average states of the steady state part of the model over time step period  $\delta$ . For system dynamics, the  $x_{Dyn}$  vector represents dynamic energy and mass flows that change over time, and the  $\bar{x}_{Dyn}$  vector represents the average of  $x_{Dyn}$  variables over the time step period. The  $y$  vector represents input parameters such as loads and outside air temperatures that remain constant over a time step. However, these inputs can change from time step to time step to represent varying weather and loads.

minimize: (in order)	$f(x_{Avg}, \bar{x}_{Dyn}, y)$	4.1
subject to:	$g(x_{Avg}, \bar{x}_{Dyn}, y) \leq \mathbf{0}$	4.2
	$h(x_{Avg}, \bar{x}_{Dyn}, y) = \mathbf{0}$	4.3
	$j(x_{Avg}, x_{Dyn}(t), \dot{x}_{Dyn}(t), y) = \mathbf{0}$	4.4
	$\bar{x}_{Dyn} = \frac{1}{\delta} \int_0^{\delta} x_{Dyn}(\tau) d\tau$	4.5

**Figure 28 – Mathematical Form for Hybrid Problems**

Equations 4.1–4.3 of Figure 28 cover steady state aspects of the model, while equation 4.4 covers dynamic aspects of the model. These differential-algebraic equations must have a unique solution for  $x_{Dyn}(t)$  to be valid. Equation 4.5 connects the steady state and dynamic aspects of the model by calculating the average of each dynamic variable over a time step.

Hybrid models in the form of Figure 28 can be solved by eliminating dynamic variables and solving the resulting constrained optimization. Figure 29 below shows this process. First, time-average variables are solved for as a function of time from their differential equations. Exact analytical solutions or approximate numerical solutions can be used. Next, functions for time averages are calculated. Finally, the time-average solutions are used as equality constraints in the final formulation.

Starting conditions for the dynamic variables are required for running a model since they usually appear in the dynamic variable's time averaged equations. At the end of each time step the final dynamic states are calculated for use as the initial conditions on the next time step. The tracking of the initial and final states of dynamic variables creates a dependence on each time step to the previous time step. These initial and final conditions or the state of a dynamic variable in between them can also be explicitly added as variables to a model for outputting within a solver. This enables visualizing the time-average of the dynamic states as well as the dynamic states themselves versus time.



## Original Hybrid Model

minimize: (in order)	$f(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y})$
subject to:	$g(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y}) \leq \mathbf{0}$ $h(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y}) = \mathbf{0}$ $j(\mathbf{x}_{Avg}, \mathbf{x}_{Dyn}(t), \dot{\mathbf{x}}_{Dyn}(t), \mathbf{y}) = \mathbf{0}$ $\bar{\mathbf{x}}_{Dyn} = \frac{1}{\delta} \int_0^{\delta} \mathbf{x}_{Dyn}(\tau) d\tau$



## Solve for $\mathbf{x}_{Dyn}(t)$

minimize: (in order)	$f(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y})$
subject to:	$g(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y}) \leq \mathbf{0}$ $h(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y}) = \mathbf{0}$ $\mathbf{x}_{Dyn}(t) = \mathbf{k}(\mathbf{x}_{Avg}, \mathbf{y}, t)$ $\bar{\mathbf{x}}_{Dyn} = \frac{1}{\delta} \int_0^{\delta} \mathbf{x}_{Dyn}(\tau) d\tau$



## Find Time Averages

$$\bar{\mathbf{x}}_{Dyn} = \frac{1}{\delta} \int_0^{\delta} \mathbf{x}_{Dyn}(\tau) d\tau$$



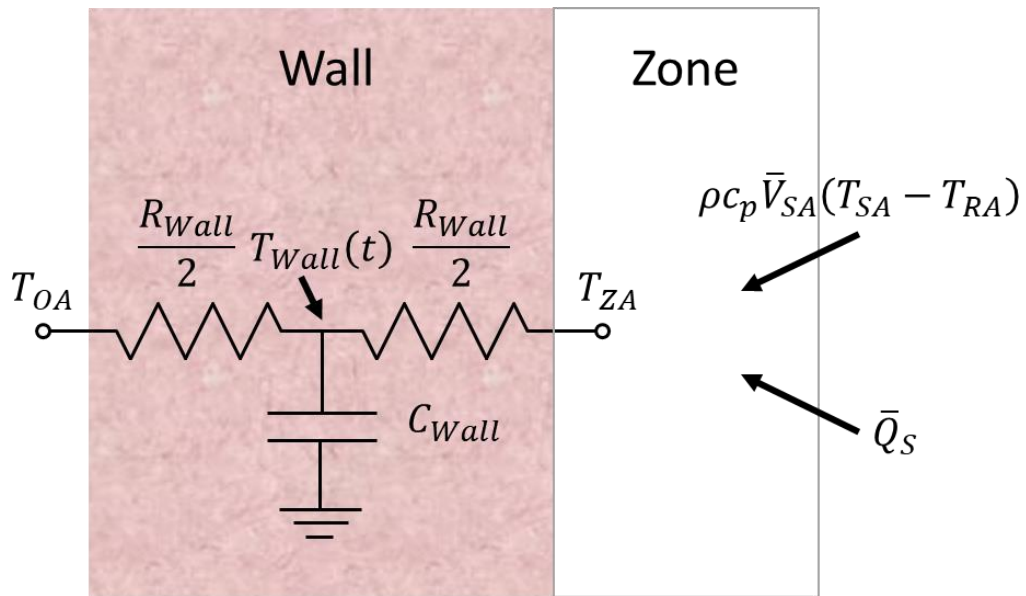
## Final Hybrid Model

minimize: (in order)	$f(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y})$
subject to:	$g(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y}) \leq \mathbf{0}$ $h(\mathbf{x}_{Avg}, \bar{\mathbf{x}}_{Dyn}, \mathbf{y}) = \mathbf{0}$ $\bar{\mathbf{x}}_{Dyn} = \mathbf{l}(\mathbf{x}_{Avg}, \mathbf{y})$

Figure 29 – Converting a Hybrid Model to a Solvable Constrained Optimization

### A Simple Hybrid Model

Figure 30 shows a model for a zone where the only thermal contact with the outside is a brick wall modeled as an RC network with a single thermal capacitance. The wall temperature in this model varies continuously while the zone temperature and supply air flow are assumed to be constant over a time step. For zone temperature control the supply air flow is varied to keep the zone at a constant set point.



**Figure 30 – A Steady State Zone Model with Thermal Mass in the Wall**

Table 5 shows input parameter values for the model of Figure 30. The wall's thermal parameters and thickness are set to typical values for a brick structure with no insulation. These fundamental parameters are translated to the wall resistance, thermal capacitance, time constant, and UA value parameters used in the RC circuit model and zone temperature balance using equations 4.1–4.4. Table 6 lists their values.

**Table 5 – Input Parameters for the Example Hybrid Model**

Parameter	Symbol	Value	Unit
Wall Density	$\rho_{Wall}$	120	$\frac{lbm}{ft^3}$
Wall Thermal Conductivity	$k_{Wall}$	0.42	$\frac{Btu}{h \cdot ft \cdot ^\circ F}$
Wall Specific Heat	$c_{p,Wall}$	0.20	$\frac{Btu}{lbm \cdot ^\circ F}$
Wall Thickness	$l_{Wall}$	0.36	$ft$
Wall Area	$A_{Wall}$	5,600	$ft^2$
Floor Area	$A_{Floor}$	10,000	$ft^2$
Zone Set Point Temperature	$T_{ZA,SP}$	70	$^\circ F$
Supply Air Temperature	$T_{SA}$	55	$^\circ F$
Air Density Times Specific Heat	$\rho c_p$	1.08	$\frac{Btu}{h \cdot ^\circ F \cdot CFM}$
Time Step Period	$\delta$	1.0	$h$
Min Supply Air Flow Normalized to Floor Area	$\bar{V}_{SA,Min}$	0.3	$\frac{CFM}{ft^2}$
Max Supply Air Flow Normalized to Floor Area	$\bar{V}_{SA,Max}$	1.0	$\frac{CFM}{ft^2}$
Zone Load Normalized to Floor Area	$\bar{Q}_S$	10.0	$\frac{Btu}{h \cdot ft^2}$

$$R_{Wall} = \frac{l_{Wall}}{k_{Wall}} \quad 4.6$$

$$C_{Wall} = \rho_{Wall} c_{p,Wall} l_{Wall} \quad 4.7$$

$$\tau_{Wall} = R_{Wall} C_{Wall} \quad 4.8$$

$$\bar{U}A_{Wall} = \frac{A_{Wall}}{R_{Wall}A_Z} = \frac{k_{Wall}A_{Wall}}{l_{Wall}A_Z} \quad 4.9$$

**Table 6 – Calculated Parameters for the Example Hybrid Model**

Parameter	Symbol	Value	Unit
Wall Thermal Resistance	$R_{Wall}$	0.86	$\frac{h \cdot ft \cdot ^\circ F}{Btu}$
Wall Thermal Capacitance	$C_{Wall}$	8.6	$\frac{Btu}{ft^2 \cdot ^\circ F}$
Wall Time Constant	$\tau_{Wall}$	7.4	$h$
Wall UA Value Normalized to Floor Area	$\overline{UA}_{Wall}$	0.65	$\frac{Btu}{h \cdot ft^2 \cdot ^\circ F}$

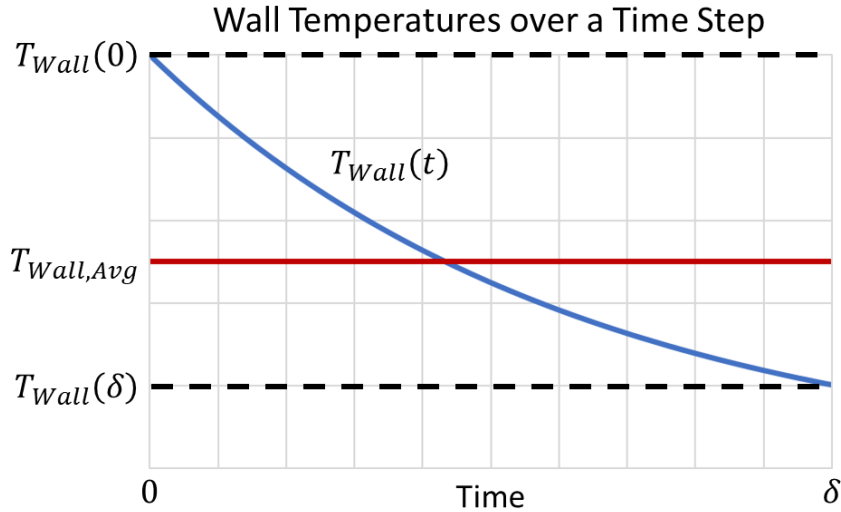
Figure 31 shows a mathematical formulation of the zone/wall model of Figure 30 using constrained optimization. Equations 4.10–4.12 cover the steady state zone model. Equation 4.10 describes the control objective of maintaining the zone temperature set point, equation 4.11 describes the zone temperature balance, and equation 4.12 describes the allowable range of supply air flow rates. Equations 4.13 and 4.14 cover the dynamic wall model. The differential equation of equation 4.13 describes the wall temperature change over time. Equation 4.14 is the formula for calculating the average wall temperature over a time step period  $\delta$ . This couples the steady state and dynamic parts of the model by ensuring that the energy transferred from the wall over a time step equals the load in the zone temperature balance equation 4.11.

minimize	$ T_{ZA} - T_{ZA,SP} $	4.10
subject to	$\rho c_p \bar{V}_{SA}(T_{SA} - T_{ZA}) + 2\overline{UA}_{Wall}(T_{Wall,Avg} - T_{ZA}) + \bar{Q}_S = 0$	4.11
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA} \leq \bar{V}_{SA,Max}$	4.12
	$\tau_{Wall} \dot{T}_{Wall}(t) = -4T_{Wall}(t) + 2T_{OA} + 2T_{ZA}$	4.13
	$T_{Wall,Avg} = \frac{1}{\delta} \int_0^\delta T_{Wall}(t) dt$	4.14

**Figure 31 – Mathematical Formulation of the Zone Model**

Figure 32 shows how the continuous wall temperature  $T_{Wall}(t)$  in the hybrid model changes over a time step and its average value  $\bar{T}_{Wall}$ . At the beginning of a time

step the wall temperature is known to be  $T_{Wall}(0)$ . Also, equation 4.13 controls how the  $T_{Wall}(t)$  changes over time. Equation 4.13 is a linear differential equation over each time step period because of the assumption that its steady state variables are constant over a time step. This causes  $T_{Wall}(t)$  to exponentially converge to the wall temperature at the end of the time step  $T_{Wall}(\delta)$ .



**Figure 32 – Average and Continuous Wall Temperatures over a Time Step**

Equations 4.15–4.27 show the calculation of average wall temperatures from the differential equation in equation 4.13 and the definition of the average wall temperature over a time step in equation 4.14. First, equation 4.13 is solved for wall temperature  $T_{Wall}(t)$  over time, as shown in equations 4.15–4.20.

$$\dot{T}_{Wall}(t) = -\frac{4}{\tau_{Wall}}T_{Wall}(t) + \frac{2}{\tau_{Wall}}T_{OA} + \frac{2}{\tau_{Wall}}T_{ZA} \quad 4.15$$

$$T_{Wall}(t) = e^{-\frac{4}{\tau_{Wall}}t}T_{Wall}(0) + \int_0^t e^{-\frac{4}{\tau_{Wall}}(t-\tau)} \frac{2}{\tau_{Wall}}(T_{OA} + T_{ZA})d\tau \quad 4.16$$

$$T_{Wall}(t) = e^{-\frac{4}{\tau_{Wall}}t}T_{Wall}(0) + \int_0^t e^{-\frac{4}{\tau_{Wall}}t} e^{\frac{4}{\tau_{Wall}}\tau} \frac{2}{\tau_{Wall}}(T_{OA} + T_{ZA})d\tau \quad 4.17$$

$$T_{Wall}(t) = e^{-\frac{4}{\tau_{Wall}}t} T_{Wall}(0) + \frac{2}{\tau_{Wall}} (T_{OA} + T_{ZA}) e^{-\frac{4}{\tau_{Wall}}t} \int_0^t e^{\frac{4}{\tau_{Wall}}\tau} d\tau \quad 4.18$$

$$T_{Wall}(t) = e^{-\frac{4}{\tau_{Wall}}t} T_{Wall}(0) + \frac{T_{OA} + T_{ZA}}{2} \left( 1 - e^{-\frac{4}{\tau_{Wall}}t} \right) \quad 4.19$$

$$T_{Wall}(t) = \frac{T_{OA} + T_{ZA}}{2} + \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) e^{-\frac{4}{\tau_{Wall}}t} \quad 4.20$$

Next, the solution for  $T_{Wall}(t)$  is substituted into equation 4.14 to calculate the average wall temperature  $T_{Wall,Avg}$ , as shown in equations 4.21 and 4.22. Equations 4.23–4.27 show the evaluation of the integral and the simplification process. Equation 4.27 expresses the average wall temperature in terms of the initial condition and steady state variables. It can be coupled with the steady state part of the problem to construct the quasi-steady state model of Figure 33.

$$T_{Wall,Avg} = \frac{1}{\delta} \int_0^\delta T_{Wall}(\tau) d\tau \quad 4.21$$

$$T_{Wall,Avg} = \frac{1}{\delta} \int_0^\delta \frac{T_{OA} + T_{ZA}}{2} + \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) e^{-\frac{4}{\tau_{Wall}}\tau} d\tau \quad 4.22$$

$$T_{Wall,Avg} = \frac{1}{\delta} \int_0^\delta \frac{T_{OA} + T_{ZA}}{2} d\tau + \frac{1}{\delta} \int_0^\delta \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) e^{-\frac{4}{\tau_{Wall}}\tau} d\tau \quad 4.23$$

$$T_{Wall,Avg} = \frac{1}{\delta} \frac{T_{OA} + T_{ZA}}{2} \int_0^\delta d\tau + \frac{1}{\delta} \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) \int_0^\delta e^{-\frac{4}{\tau_{Wall}}\tau} d\tau \quad 4.24$$

$$T_{Wall,Avg} = \frac{T_{OA} + T_{ZA}}{2} + \frac{1}{\delta} \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) \int_0^\delta e^{-\frac{4}{\tau_{Wall}}\tau} d\tau \quad 4.25$$

$$T_{Wall,Avg} = \frac{T_{OA} + T_{ZA}}{2} + \frac{1}{\delta} \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) \frac{\tau_{Wall}}{4} \left( 1 - e^{-\frac{4\delta}{\tau_{Wall}}} \right) \quad 4.26$$

$$T_{Wall,Avg} = \frac{T_{OA} + T_{ZA}}{2} + \frac{\tau_{Wall}}{4\delta} \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) \left( 1 - e^{-\frac{4\delta}{\tau_{Wall}}} \right) \quad 4.27$$

In the formulation of Figure 33 equations 4.13 and 4.14 of Figure 31 are replaced by equation 4.27 to solve for the average wall temperature as it varies over time on a time step. The initial wall temperature  $T_{Wall}(0)$  is given at the beginning of a time step, and this value is updated for the next time step. To do this, equation 4.20 is evaluated at the end of the time step, resulting in equation 4.32. eliminating all differential equations and calculus from the model. The model of Figure 33 is solvable using nonlinear programming since it no longer contains any differential equations or calculus.

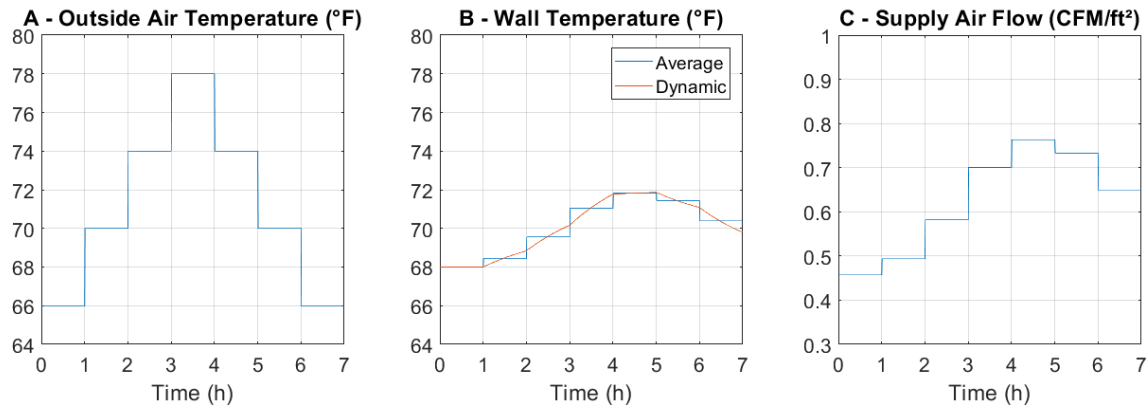
minimize:	$ T_{ZA} - T_{ZA,SP} $	4.28
subject to:	$\rho c_p \bar{V}_{SA}(T_{SA} - T_{ZA}) + 2\bar{U}\bar{A}_{Wall}(T_{Wall,Avg} - T_{ZA}) + \bar{Q}_S = 0$	4.29
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA} \leq \bar{V}_{SA,Max}$	4.30
	$T_{Wall,Avg} = \frac{T_{OA} + T_{ZA}}{2} + \frac{\tau_{Wall}}{4\delta} \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) \left( 1 - e^{-\frac{4\delta}{\tau_{Wall}}} \right)$	4.31
	$T_{Wall}(\delta) = \frac{T_{OA} + T_{ZA}}{2} + \left( T_{Wall}(0) - \frac{T_{OA} + T_{ZA}}{2} \right) e^{-\frac{4\delta}{\tau_{Wall}}}$	4.32

**Figure 33 – Solvable Hybrid Model of a Zone with Thermal Mass**

The SZVAV model was simulated with the outside air temperature data shown in Figure 34A. This outside air temperature data symmetrically rises and lowers over a seven-hour period, staying constant over each hour. A single quasi-steady state simulation is performed for each hour, with dynamics embedded within the model. The initial wall temperature node is set to the average of the zone temperature set point and the initial outside air temperature. Figure 34B shows the average wall temperatures from the steady state simulation and how the wall temperature changes continuously over time.

At the beginning of the simulation the wall has a temperature of  $68^\circ F$ , as seen in Figure 34B. No change in wall temperature occurs during the first hour since this is balanced between the outside air temperature of  $66^\circ F$  and the zone temperature of  $70^\circ F$ . Later in the simulation the wall temperature rise from the outside air is delayed by the thermal mass. Figure 34C shows steady state supply air flows for each hour of the

simulation. In a purely steady state model supply air flows would only depend on the outside air temperature, but the thermal mass in this model causes a delay in when supply air is required.



**Figure 34 – Example Hybrid Model Simulation  
Integrating Linear Differential Equations to Steady State Models**

Dynamic states in hybrid models change over time during a time step while averaged states are assumed to remain constant. Systems of linear differential equations within a hybrid model that are coupled to the steady state part of the model can be written in the form of equation 4.33. In this equation the  $\mathbf{x}$  vector represents dynamic states that vary continuously, the  $\mathbf{y}$  vector represents averaged states that vary once on each time step, and the  $\mathbf{A}$  matrix is square.

$$\dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{y})\mathbf{x}(t) + \mathbf{b}(\mathbf{y}) \quad 4.33$$

Elements of  $\mathbf{A}(\mathbf{y})$ , and  $\mathbf{b}(\mathbf{y})$  can be constants or functions of the averaged states. Since averaged states stay constant on a time step, even if elements of  $\mathbf{A}(\mathbf{y})$ , and  $\mathbf{b}(\mathbf{y})$  depend nonlinearly on  $\mathbf{y}$  the dynamic part of the model is always a set of linear differential equations. The calculations below show how to integrate systems of equations in the form of equation 4.33 into a constrained optimization model. This formulation contains some steady state variables and some quasi-steady state variables.



Linear differential equations in the form of equation 4.33 are applicable for modeling one dimensional heat transfer [70]. In RC thermal network calculations  $\mathbf{A}(\mathbf{y})$  represents thermal resistances and thermal mass and is usually constant, while  $\mathbf{b}(\mathbf{y})$  represents temperatures such as outside air temperature that drive heat flow. Appendix B shows a derivation of RC thermal networks with equally spaced nodes.

In the following equations the time variable  $t$  represents the time over an individual time step, where  $t = 0$  at the beginning of a time step. Equation 4.44 gives the overall solution to equation 4.33 over a time step. Evaluating the integral results in equation 4.35. Equation 4.36 gives the time average of a function over a time step. Substituting equation 4.35 into equation 4.36 and evaluating results in equation 4.37, which is a linear function of the initial conditions of the dynamic states at the beginning of a time step and the  $\mathbf{b}(\mathbf{y})$  vector. Appendix A shows the derivation of equations 4.35 and 4.37 over a time step.

$$\mathbf{x}(t) = e^{\mathbf{A}(\mathbf{y})t} \mathbf{x}(0) + \int_{t_i}^t e^{\mathbf{A}(\mathbf{y})(t-\tau)} \mathbf{b}(\mathbf{y}) d\tau \quad 4.34$$

$$\mathbf{x}(t) = e^{\mathbf{A}(\mathbf{y})t} \mathbf{x}(0) + (e^{\mathbf{A}(\mathbf{y})t} - \mathbf{I}) \mathbf{A}(\mathbf{y})^{-1} \mathbf{b}(\mathbf{y}) \quad 4.35$$

$$\bar{\mathbf{x}} = \frac{1}{\delta} \int_0^{\delta} \mathbf{x}(\tau) d\tau \quad 4.36$$

$$\bar{\mathbf{x}} = \frac{1}{\delta} (e^{\mathbf{A}(\mathbf{y})\delta} - \mathbf{I}) \mathbf{A}(\mathbf{y})^{-1} \mathbf{x}(0) - \left( \frac{1}{\delta} (e^{\mathbf{A}(\mathbf{y})\delta} - \mathbf{I}) \mathbf{A}(\mathbf{y})^{-1} - \mathbf{I} \right) \mathbf{A}(\mathbf{y})^{-1} \mathbf{b}(\mathbf{y}) \quad 4.37$$

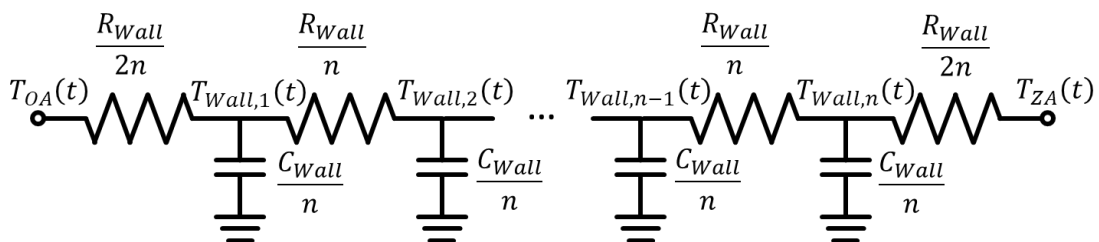
Calculations for the averaged dynamic variables  $\bar{\mathbf{x}}$  include the initial conditions for the dynamic states,  $\mathbf{x}(0)$ . For calculations after the first time step  $\mathbf{x}(0)$  is equal to  $\mathbf{x}(\delta)$  at the end of the previous time step. Equation 4.38 gives the formula for  $\mathbf{x}(\delta)$ , which is equation 4.35 evaluated at  $t = \delta$ .

$$\mathbf{x}(\delta) = e^{\mathbf{A}(\mathbf{y})\delta} \mathbf{x}(0) + (e^{\mathbf{A}(\mathbf{y})\delta} - \mathbf{I}) \mathbf{A}(\mathbf{y})^{-1} \mathbf{b}(\mathbf{y}) \quad 4.38$$

Whenever the  $\mathbf{A}$  matrix is independent of the averaged states  $\mathbf{y}$  all equations from this section reduce to linear functions that don't change over a simulation. This allows thermal mass model variables to be presolved out of the actual model that's solved by a nonlinear programming solver. Since solving the remaining linear equations with known variables is trivial compared to constrained optimization, this allows large thermal mass models with hundreds of variables to be solved in almost the same amount of time as a resistance only model with no thermal mass.

### Hybrid Wall Models with Multiple Nodes

Thermal mass of a wall can also be modeled by dividing a wall into multiple layers and using an RC circuit for each layer. This more accurately models one dimensional heat transfer through a wall since it tracks temperatures at different levels instead of assuming that the wall's thermal mass is at a single temperature. Figure 35 shows a network for a wall made of a single substance divided into  $n$  equal sections that are assumed to each be at a uniform temperature. Each section has a capacitance of  $\frac{C_{Wall}}{n}$ , where  $C_{Wall}$  is the total thermal capacitance of the wall. Resistances at the wall surfaces have a resistance of  $\frac{R_{Wall}}{2n}$ , where  $R_{Wall}$  is the total thermal resistance of the wall. Interior resistances have a value of  $\frac{R_{Wall}}{n}$  since they're made up of two  $\frac{R_{Wall}}{2n}$  resistances joined together.



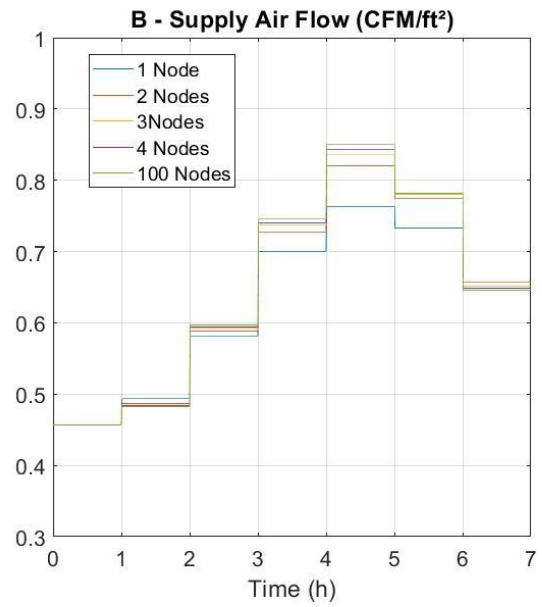
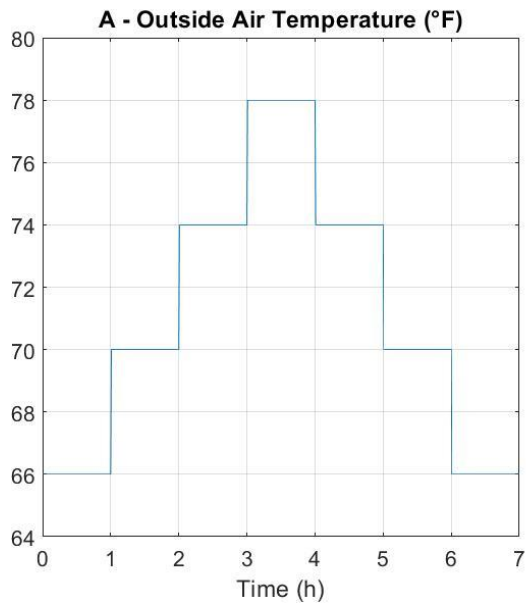
**Figure 35 – An RC Network for Modeling Thermal Mass of a Wall**

The model of Figure 36 shows the wall/zone model of Figure 31 with a dynamic wall model that can use any number of nodes. Appendix B shows the derivation of this equation. This linear system can be solved using the equations of the previous section.

minimize:	$ T_{ZA} - T_{ZA,SP} $	4.39
subject to:	$\rho c_p \bar{V}_{SA}(T_{SA} - T_{ZA}) + 2n\bar{U}A_{Wall}(\bar{T}_{Wall,n} - T_{ZA}) + \bar{Q}_S = 0$	4.40
	$\bar{V}_{SA,Min} \leq \bar{V}_{SA} \leq \bar{V}_{SA,Max}$	4.41
	$\frac{\tau_{Wall}}{n^2} \begin{bmatrix} \dot{T}_{Wall,1} \\ \dot{T}_{Wall,2} \\ \dot{T}_{Wall,3} \\ \vdots \\ \dot{T}_{Wall,n-2} \\ \dot{T}_{Wall,n-1} \\ \dot{T}_{Wall,n} \end{bmatrix} = \begin{bmatrix} -3 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -2 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -3 \end{bmatrix} \begin{bmatrix} T_{Wall,1} \\ T_{Wall,2} \\ T_{Wall,3} \\ \vdots \\ T_{Wall,n-2} \\ T_{Wall,n-1} \\ T_{Wall,n} \end{bmatrix}$ $+ \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} T_{OA} \\ T_{ZA} \end{bmatrix}$	4.42
	$\bar{T}_{Wall,i} = \frac{1}{\delta} \int_0^\delta T_{Wall,i}(\tau) d\tau$ for each wall node $i$	4.43

**Figure 36 – Hybrid Model with the Wall Divided into Multiple Layers**

Figure 37 shows outside air temperatures and resulting supply air flows for the models of Figure 31 and Figure 36 simulated with different numbers of wall temperature nodes. Initial wall temperature nodes were set to their steady state value with the zone at its setpoint temperature and the outside air temperature at the first hour's value. As the number of nodes increase the magnitude of the heating load passed increases, and this results in higher supply air flows. Higher loads with more nodes are due to the smaller discrete thermal masses. More nodes allow heat to flow deeper inside instead of having to heat a larger thermal mass first. This effect is limited, however, which can be seen by comparing the 100 node simulation with the other models.



**Figure 37 – Air Flow Rates using Different Numbers of Wall Temperature Nodes**

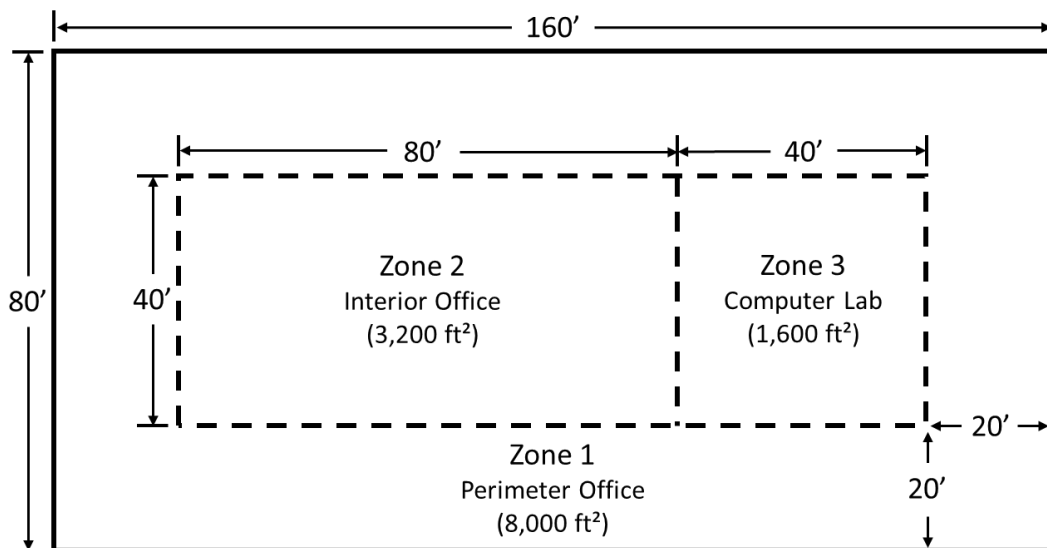
CHAPTER V  
EXAMPLE HVAC MODELS FOR COMMON AIR HANDLER TYPES

Commercial buildings and their HVAC system designs vary by design requirements, region, when they were built, and the personal preferences of the architects and engineers involved in a project. This variety makes it challenging to develop flexible building energy use modeling methods than can correctly represent a variety of systems in one tool. This chapter gives examples of constrained optimization models for a variety of HVAC systems and components for an example building. Modifications are made to these models to highlight how different system variations can be represented without a major reworking of source code.

**Building Description**

*Dimensions and Layout*

Figure 38 shows the layout of the simple, hypothetical office space that's modeled in this chapter. For modeling purposes this building is divided into a perimeter zone of office space, an interior zone of office space, and an interior zone for a computer room. Table 7 gives the input parameters that define the building and zone dimensions.



**Figure 38 – Example Building Floor Layout**

**Table 7 – Example Building Dimensions**

Parameter	Symbol	Value	Unit
Building Length	$l_{Building}$	160	<i>ft</i>
Building Width	$w_{Building}$	80	<i>ft</i>
Building Height	$h_{Building}$	15	<i>ft</i>
Zone 2 Length	$l_{Z2}$	80	<i>ft</i>
Zone 2 Width	$w_{Z2}$	40	<i>ft</i>
Zone 3 Length	$l_{Z3}$	40	<i>ft</i>
Zone 3 Width	$w_{Z3}$	40	<i>ft</i>

Equations 5.1 and 5.2 give the interior zone areas in terms of their inputted widths and heights. Equation 5.3 gives the perimeter zone area as the overall building area minus the interior zone areas. Since the example building is a single floor the roof areas of each zone equal their zone areas, as given in equations 5.4–5.6. Table 8 lists these calculated values.

$$A_{Z2} = l_{Z2}w_{Z2} \quad 5.1$$

$$A_{Z3} = l_{Z3}w_{Z3} \quad 5.2$$

$$A_{Z1} = l_{building}w_{building} - A_{Z2} - A_{Z3} \quad 5.3$$

$$A_{Roof1} = A_{Z1} \quad 5.4$$

$$A_{Roof2} = A_{Z2} \quad 5.5$$

$$A_{Roof3} = A_{Z3} \quad 5.6$$

**Table 8 – Building Zone Parameters**

Parameter	Symbol (Zone i)	Zone 1 (Perimeter Office)	Zone 2 (Interior Office)	Zone 3 (Computer Lab)	Unit
Floor Area	$A_{Zi}$	8,000	3,200	1,600	<i>ft</i> <sup>2</sup>
Roof Area	$A_{Roofi}$	8,000	3,200	1,600	<i>ft</i> <sup>2</sup>

Equation 5.7 gives the total zone area as the sum of the three individual zone areas. Equations 5.8–5.10 give the total window and wall areas for the perimeter zone. Table 9 lists their calculated values.

$$A_Z = A_{Z1} + A_{Z2} + A_{Z3} \quad 5.7$$

$$A_{Wall,Total} = 2(l_{Building} + w_{Building})h_{Building} \quad 5.8$$

$$A_{Window} = 0.01A_{Wall,Total}Pct_{Window} \quad 5.9$$

$$A_{Wall} = A_{Wall,Total} - A_{Window} \quad 5.10$$

**Table 9 – Calculated Building Areas**

Parameter	Symbol	Value	Unit
Total Floor Area	$A_Z$	12,800	$ft^2$
Total Wall Area (Windows and Walls)	$A_{Wall,Total}$	7,200	$ft^2$
Window Area	$A_{Window}$	2,880	$ft^2$
Wall Area	$A_{Wall}$	4,320	$ft^2$

### *Thermal Properties*

Table 10 gives thermal property inputs for the building. The perimeter walls are made of concrete with a thickness of 0.66 *ft*, with windows covering 40% of the perimeter surface. The wall's density and specific heat are only used in the DDVAV model with dynamics. All other models treat exterior heat conduction as steady state.

**Table 10 – Building Thermal Property Inputs**

Parameter	Symbol	Value	Unit
Wall Thermal Conductivity	$k_{Wall}$	0.42	$\frac{Btu}{h \cdot ft \cdot ^\circ F}$
Wall Thickness	$l_{Wall}$	0.66	$ft$
Wall Density	$\rho_{Wall}$	120	$\frac{lbm}{ft^3}$
Wall Specific Heat	$c_{p,Wall}$	0.20	$\frac{Btu}{lbm \cdot ^\circ F}$
Window R-Value	$R_{Window}$	2.0	$\frac{h \cdot ft^2 \cdot ^\circ F}{Btu}$
Roof R-Value	$R_{Roof}$	20	$\frac{h \cdot ft^2 \cdot ^\circ F}{Btu}$
Window Percentage	$Pct_{Window}$	40	%

Equations 5.11 and 5.12 give the overall wall resistance and thermal capacitance. Multiplying these two parameters together gives the overall wall time constant, shown in equation 5.13. Equations 5.14 and 5.15 give the wall and window/roof UA values for the perimeter zone normalized to the perimeter zone area. Table 11 shows the calculated values of these parameters.

$$R_{Wall} = \frac{l_{Wall}}{k_{Wall}} \quad 5.11$$

$$C_{Wall} = \rho_{Wall} c_{p,Wall} l_{Wall} \quad 5.12$$

$$\tau_{Wall} = R_{Wall} C_{Wall} \quad 5.13$$

$$\overline{UA}_{Z1,Wall} = \frac{A_{Wall}}{R_{Wall} A_{Z1}} \quad 5.14$$

$$\overline{UA}_{Z1,Window/Roof} = \frac{A_{Window}}{R_{Window} A_{Z1}} + \frac{A_{Roof1}}{R_{Roof} A_{Z1}} \quad 5.15$$



**Table 11 – Calculated Wall Parameters**

Parameter	Symbol	Value	Unit
Wall R-Value	$R_{Wall}$	1.57	$\frac{h \cdot ft^2 \cdot ^\circ F}{Btu}$
Wall Thermal Capacitance	$C_{Wall}$	15.8	$\frac{Btu}{ft^2 \cdot ^\circ F}$
Wall Time Constant	$\tau_{Wall}$	24.9	$h$
Perimeter Zone Wall UA Value	$\overline{UA}_{Z1,Wall}$	0.34	$\frac{Btu}{h \cdot ft^2 \cdot ^\circ F}$
Perimeter Zone Window and Roof UA Value	$\overline{UA}_{Z1,Window/Roof}$	0.23	$\frac{Btu}{h \cdot ft^2 \cdot ^\circ F}$

Equation 5.16 gives the overall floor area normalized UA value for the perimeter zone. The two interior zones only have roof area, so their floor area normalized UA values can be calculated from the roof component alone. This is shown in equations 5.17 and 5.18. Equation 5.19 gives the overall air volume of each zone as the zone floor area times the building height. Table 12 shows these calculated zone parameters.

$$\overline{UA}_{Z1} = \overline{UA}_{Z1,Wall} + \overline{UA}_{Z1,Window/Roof} \quad 5.16$$

$$\overline{UA}_{Z2} = \frac{A_{Roof2}}{R_{Roof}A_{Z2}} \quad 5.17$$

$$\overline{UA}_{Z3} = \frac{A_{Roof3}}{R_{Roof}A_{Z3}} \quad 5.18$$

$$V_{Zi} = A_{Zi}h_{Building} \quad 5.19$$

**Table 12 – Calculated Zone Parameters**

Parameter	Symbol (Zone i)	Zone 1 (Perimeter Office)	Zone 2 (Interior Office)	Zone 3 (Computer Lab)	Unit
Total Zone UA Value per $ft^2$ of Floor Area	$\overline{UA}_{Zi}$	0.57	0.05	0.05	$\frac{Btu}{h \cdot ft^2 \cdot ^\circ F}$
Zone Air Volume	$V_{Zi}$	120,000	48,000	24,000	$ft^3$

*Zone Loads*

Table 13 gives the sensible and latent loads per person used in the models of this chapter as well as the symbols and units used for the lighting and plug load ratios. Table 14 gives the peak lighting and plug usages and peak occupancies of each zone.

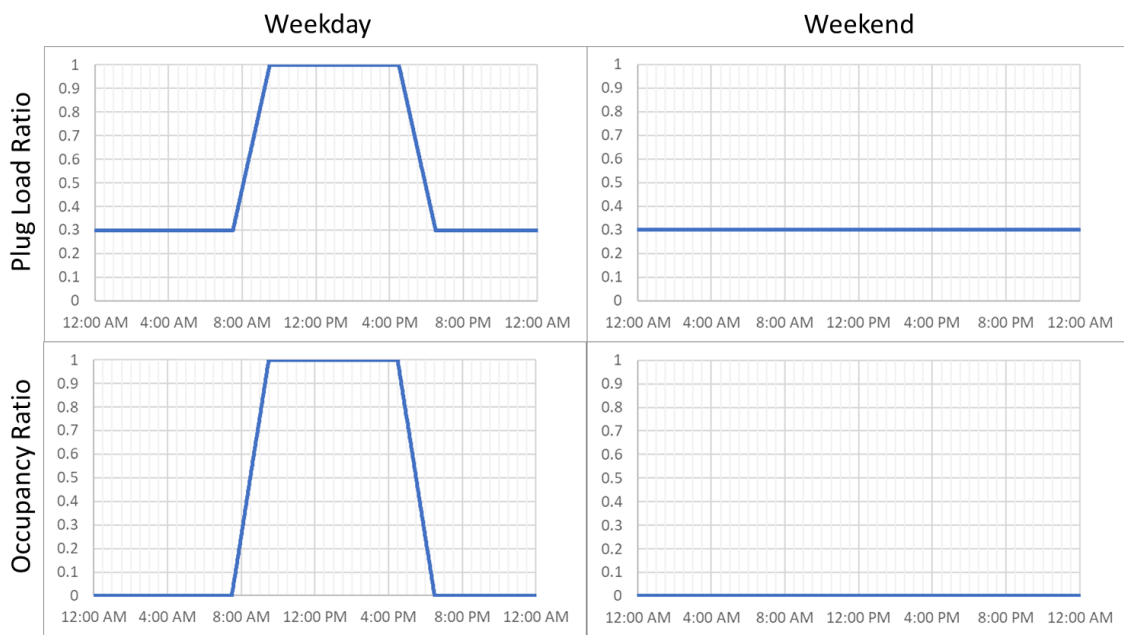
**Table 13 – Lighting, Plug and People Load Inputs**

Parameter	Symbol	Value	Unit
Sensible Load per Person	$\tilde{Q}_{S,PerPerson}$	300	$\frac{Btu}{h \cdot Person}$
Latent Load per Person	$\tilde{Q}_{L,PerPerson}$	250	$\frac{Btu}{h \cdot Person}$
Lighting and Plug Load Ratio	$X_{L\&P}$	Varies	0 – 1
People Load Ratio	$X_{People}$	Varies	0 – 1

**Table 14 – Zone Load Inputs**

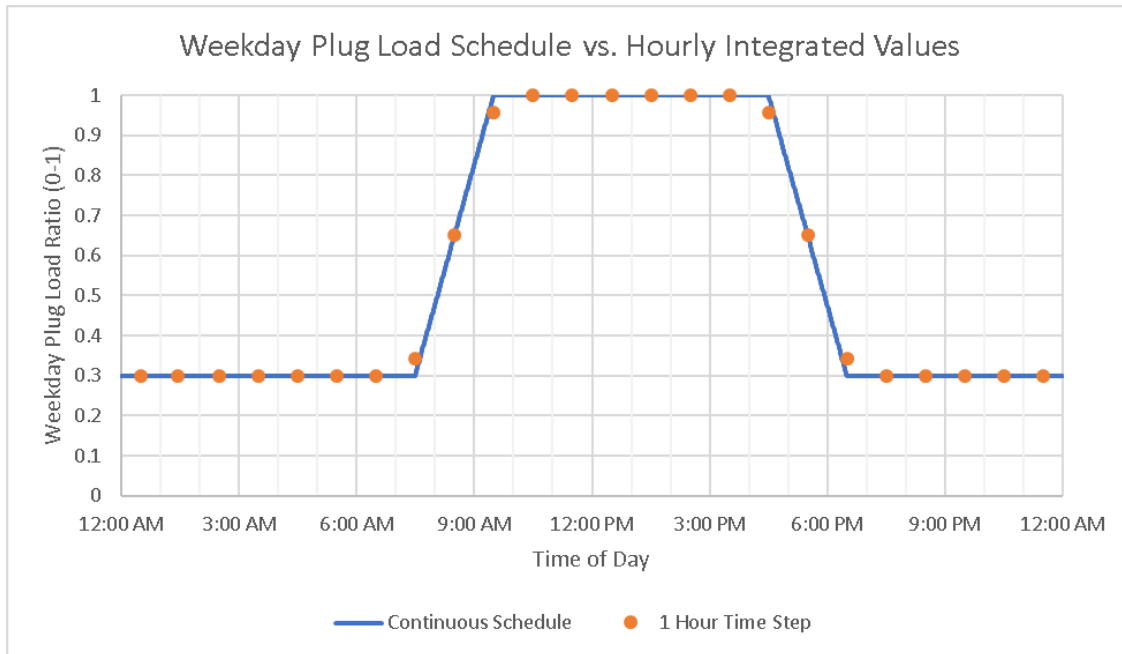
Parameter	Symbol (Zone i)	Zone 1 (Perimeter Office)	Zone 2 (Interior Office)	Zone 3 (Computer Lab)	Unit
Peak Lighting and Plug Load per Square Foot	$\bar{E}_{L\&Pi,Max}$	1.3	1.3	2.0	$\frac{W}{ft^2}$
Peak Occupancy per Person	$\tilde{OCC}_{i,Max}$	150	150	80	$\frac{ft^2}{Person}$

Plug and occupancy loads are assumed to change depending on the time of day and whether the day is a weekday or weekend. Figure 39 shows the schedules for plug loads and occupancy as a fraction of their maximum value for weekdays and weekends. When the building is unoccupied at night or on a weekend plug loads are 30% of the peak value. The workday begins at 7:30 AM, reaches peak occupancy at 9:30 AM, continues until 4:30 PM, and ramps down until it reaches its final unoccupied state at 6:30 PM.



**Figure 39 – Plug and Occupancy Load Ratios for the Example Building**

Steady state building energy models assume that loads don't change over time. Averaging the continuous plug and occupancy load ratios over a time step give the fixed values that are used in the steady state model. Figure 40 shows the weekday plug ratio values used for a time step period of one hour. Hourly schedule values don't all fall on the continuous schedule since ramp up and ramp down times start on the 30 minute mark while time steps occur on the hour itself.



**Figure 40 – Hourly Weekday Plug Load Schedule Values**

*Energy Sources and Prices*

All cooling and heating is supplied using chilled and hot water. Electric usages consist of fans, lights, and plug loads. Table 15 gives the default energy prices used in models.

**Table 15 – Energy Price Inputs**

Parameter	Symbol	Value	Unit
Electricity Price	$Price_{Electric}$	0.12	$\frac{\$}{kWh}$
Chilled Water Price	$Price_{ChW}$	15	$\frac{\$}{MMBtu}$
Hot Water Price	$Price_{HW}$	12	$\frac{\$}{MMBtu}$

Energy and cost variables in models are normalized to zone square footages and time for readability and to make variable scaling for the robustness of the numerical

methods easier. For instance, typical fan energy usages in terms of in  $\frac{W}{ft^2}$  fall within a known range, where fan energy usages in  $W$  varies depending on the size of the building. Actual total energy usages or costs can be obtained by multiplying the usage by the normalizing floor area  $A_{Floor}$  and the time step period  $\delta$ , as shown in equations 5.20–5.22.

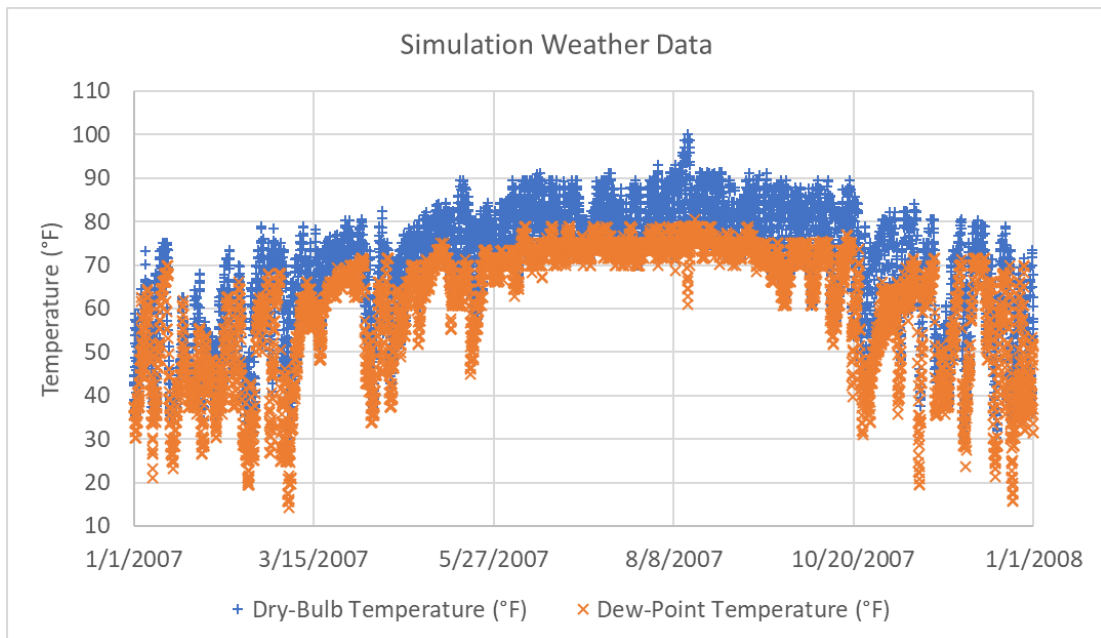
$$E_x = \bar{E}_x A_{Floor} \delta \quad 5.20$$

$$Q_x = \bar{Q}_x A_{Floor} \delta \quad 5.21$$

$$Cost_x = \overline{Cost}_x A_{Floor} \delta \quad 5.22$$

### *Weather Data for Simulations*

Simulations in this chapter are performed using weather data for Houston TX in 2007. Figure 41 shows the hourly dry bulb and dew point temperatures used. The hourly data was created by linearly interpolating raw NOAA weather station data [71].



**Figure 41 – Weather Data Used in Simulations**

### *Constants and Conversion Factors*

Table 16 lists physical constants and conversion factors used in this modeling. Properties of air are assumed to be constant for all temperatures and pressures, and the atmospheric pressure is assumed to be a typical sea level pressure. Model equations use variables for conversion factors instead of writing out numbers for readability and consistency.

**Table 16 – Physical Constants and Conversion Factors Used in Modeling**

Parameter	Symbol	Value	Unit
Density of Air times Specific Heat of Air	$\rho c_p$	1.08	$\frac{Btu}{h \cdot ^\circ F \cdot CFM}$
Density of Water times the Latent Heat of Vaporization of Water	$\rho h_{fg}$	4840.0	$\frac{Btu}{h \cdot CFM}$
The ratio of the molecular masses of water and dry air	$M_{ratio}$	0.621945	$\frac{kg/Mol_{water}}{kg/Mol_{dryAir}}$
Outside Air Pressure	$P_{OA}$	14.696	<i>psi</i>
Watts to kW Conversion Factor	$\alpha_{WattsToKw}$	0.001	$\frac{kW}{W}$
Btu to MMBtu Conversion Factor	$\alpha_{BtuToMMBtu}$	0.000001	$\frac{MMBtu}{Btu}$
Horsepower to Watt Conversion Factor	$\alpha_{HpToWatts}$	1341.02	$\frac{kW}{Hp}$
kW to Btu/h Conversion Factor	$\alpha_{KwToBtuPerH}$	3412.142	$\frac{Btu}{h \cdot kW}$
kW to Btu/h Conversion Factor	$\alpha_{HpToBtuPerH}$	4575.69	$\frac{Btu}{h \cdot Hp}$
CFM to 1000 CFM Conversion Factor	$\alpha_{CfmToKCfm}$	0.001	$\frac{kCFM}{CFM}$

### *Hourly Zone Loads*

Equation 5.23 gives the heating load for each zone in terms of the lighting and plug electric usage. Models assume that all electric energy used in a zone results in a

load. Equation 5.24 and 5.25 give peak sensible and latent people loads per square foot of zone floor area. Table 17 gives the calculated values of these parameters for each zone.

$$\bar{Q}_{L\&Pi,Max} = \bar{E}_{L\&Pi,Max} \alpha_{KwToBtuPerHour} \quad 5.23$$

$$\bar{Q}_{Si,People,Max} = \frac{\bar{Q}_{S,PerPerson}}{\bar{OCC}_{i,Max}} \quad 5.24$$

$$\bar{Q}_{Li,People,Max} = \frac{\bar{Q}_{L,PerPerson}}{\bar{OCC}_{i,Max}} \quad 5.25$$

**Table 17 – Calculated Zone Peak Loads**

Parameter	Symbol (Zone i)	Zone 1 (Perimeter Office)	Zone 2 (Interior Office)	Zone 3 (Computer Lab)	Unit
Peak Lighting and Plug Load	$\bar{Q}_{L\&Pi,Max}$	4.44	4.44	6.82	$\frac{Btu}{h \cdot ft^2}$
Peak Sensible Occupancy Load	$\bar{Q}_{Si,People,Max}$	2.0	2.0	3.75	$\frac{Btu}{h \cdot ft^2}$
Peak Latent Occupancy Load	$\bar{Q}_{Li,People,Max}$	1.67	1.67	3.125	$\frac{Btu}{h \cdot ft^2}$

Zone electric usages and loads on a particular time step of a simulation equal the peak load scaled by the current lighting and plug load or people load ratio. Equations 5.26 and 5.27 gives the effective lighting and plug electric usage for each zone and the overall building in  $\frac{W}{ft^2}$ . Equations 5.28 and 5.29 give the total effective sensible and latent loads of a zone on in  $\frac{Btu}{h \cdot ft^2}$ .

$$\bar{E}_{L\&Pi} = X_{L\&P} \bar{E}_{L\&Pi,Max} \quad 5.26$$

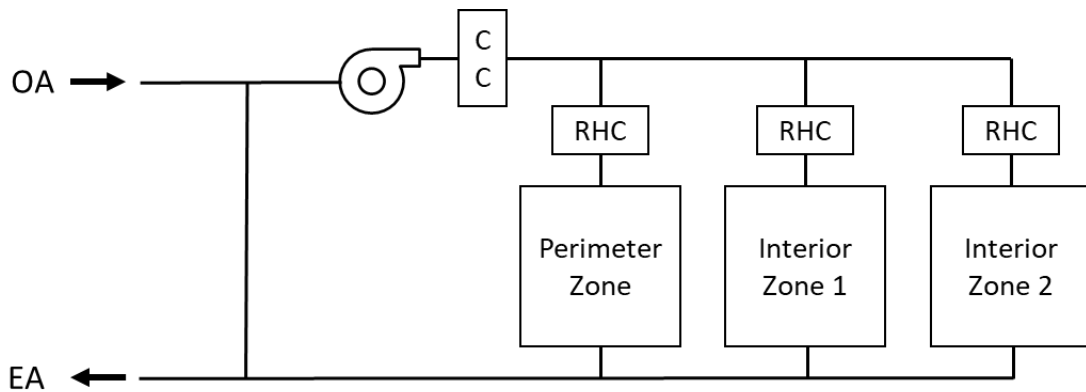
$$\bar{E}_{L\&P} = \frac{\bar{E}_{L\&P1} A_{Z1} + \bar{E}_{L\&P2} A_{Z2} + \bar{E}_{L\&P3} A_{Z3}}{A_Z} \quad 5.27$$

$$\bar{Q}_{Si} = X_{L\&P} \bar{Q}_{L\&P,Max} + X_{People} \bar{Q}_{Si,People,Max} \quad 5.28$$

$$\bar{Q}_{Li} = X_{People} \bar{Q}_{Li,People,Max} \quad 5.29$$

### Single Duct Constant Air Volume (SDCAV) Systems

Single duct constant air volume systems control zone temperatures by reheating a constant flow of conditioned supply air [72]. Figure 42 shows the schematic of a single duct constant air volume air handler with terminal reheat for the three zoned building.



**Figure 42 – SDCAV System Layout**

#### *SDCAV Input Parameters*

Table 18 and

Table 19 give values of system wide and zone parameters that are specific to the SDCAV system. The system has a constant outside air flow rate of  $0.3 \frac{CFM}{ft^2}$  and a constant fan usage of  $0.8 \frac{Hp}{KCFM}$ . The tilde over the fan usage parameter indicates that it's normalized but not to floor area. The cooling coil set point is kept at a constant  $55^{\circ}F$  over the entire year. The zones themselves have different control parameters, with the computer lab having both a lower set point and a higher flow rate.



**Table 18 – SDCAV System Parameters**

Parameter	Symbol	Value	Unit
Outside Air Flow (Normalized)	$\bar{V}_{OA}$	0.3	$\frac{CFM}{ft^2}$
Fan Power per 1000 CFM	$\widetilde{Hp}_{fan}$	0.8	$\frac{Hp}{kCFM}$
Cooling Coil Set Point Temperature	$T_{CC,SP}$	55	$^{\circ}F$

**Table 19 – SDCAV Zone Parameters**

Parameter	Symbol (Zone i)	Zone 1 (Perimeter Office)	Zone 2 (Interior Office)	Zone 3 (Computer Lab)	Unit
Zone Temperature Set Point	$T_{ZAi,SP}$	70	70	68	$^{\circ}F$
Supply Air Flow (Normalized)	$\bar{V}_{SAi}$	1.0	1.0	2.0	$\frac{CFM}{ft^2}$

*Precalculated SDCAV Parameters*

Equations 5.30–5.32 give the equations for the total supply air flow rate, the fan horsepower per square foot of floor area, and the temperature rise across the supply fan. These values can be precalculated before simulating the constrained optimization part of the SDCAV model. Table 20 gives the values of these parameters.

$$\bar{V}_{SA} = \frac{\bar{V}_{SA1}A_{Z1} + \bar{V}_{SA2}A_{Z2} + \bar{V}_{SA3}A_{Z3}}{A_Z} \quad 5.30$$

$$\overline{Hp}_{fan} = \widetilde{Hp}_{fan} \bar{V}_{SA} \alpha_{cfmToKcfm} \quad 5.31$$

$$\Delta T_{SF} = \frac{\overline{Hp}_{Fan} \alpha_{HpToBtuPerHour}}{\rho c_p \bar{V}_{SA}} \quad 5.32$$

**Table 20 – Precalculated SDCAV Parameters**

Parameter	Symbol	Value	Unit
Supply Air Flow (Normalized)	$\bar{V}_{SA}$	1.125	$\frac{CFM}{ft^2}$
Fan Power Normalized to Floor Area	$\overline{Hp}_{fan}$	0.0009	$\frac{Hp}{ft^2}$
Air Temperature Rise across the Supply Fan	$\Delta T_{SF}$	3.39	$^{\circ}F$

*The SDCAV Model*

Figure gives a constrained optimization model for the SDCAV system. SDCAV systems operate to control zone temperatures to a fixed set point, so the objective function is set to the sum of zone temperature errors. The constraints in this model describe how equipment in the system operates, define the zone temperature errors, and give definitions for overall energy usages over a time step period.

minimize:	$E_{TZA}$	5.33
subject to:	<b>AHU Temperature Balance</b>	
	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{RA}$	5.34
	$T_{SF} = T_{MA} + \Delta T_{SF}$	5.35
	$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP})$	5.36
	$\bar{V}_{SA}A_ZT_{RA} = \bar{V}_{SA1}A_{Z1}T_{ZA1} + \bar{V}_{SA2}A_{Z2}T_{ZA2} + \bar{V}_{SA3}A_{Z3}T_{ZA3}$	5.37
	<b>AHU Humidity Balance</b>	
	$\bar{V}_{SA}W_{MA} = \bar{V}_{OA}W_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})W_{RA}$	5.38
	$W_{CC,sat}P_{OA} = (M_{ratio} + W_{CC,sat})P_{ws}(T_{CC})$	5.39
	$W_{CC} = \text{Min}(W_{MA}, W_{CC,sat})$	5.40
	$\bar{V}_{SA}A_ZW_{RA} = \bar{V}_{SA1}A_{Z1}W_{ZA1} + \bar{V}_{SA2}A_{Z2}W_{ZA2} + \bar{V}_{SA3}A_{Z3}W_{ZA3}$	5.41
	<b>Zone 1</b>	
	$T_{SA1} \geq T_{CC}$	5.42
	$\rho c_p \bar{V}_{SA1}(T_{SA1} - T_{ZA1}) + \overline{UA}_{Z1}(T_{OA} - T_{ZA1}) + \bar{Q}_{S1} = 0$	5.43
	$\rho h_{fg} \bar{V}_{SA1}(W_{CC} - W_{ZA1}) + \bar{Q}_{L1} = 0$	5.44

**Figure 43 – Model of a Single Duct Constant Air Volume System**

subject to:	<b>Zone 1</b>	
	$T_{SA1} \geq T_{CC}$	5.42
	$\rho c_p \bar{V}_{SA1}(T_{SA1} - T_{ZA1}) + \bar{U}A_{Z1}(T_{OA} - T_{ZA1}) + \bar{Q}_{S1} = 0$	5.43
	$\rho h_{fg} \bar{V}_{SA1}(W_{CC} - W_{ZA1}) + \bar{Q}_{L1} = 0$	5.44
	<b>Zone 2</b>	
	$T_{SA2} \geq T_{CC}$	5.45
	$\rho c_p \bar{V}_{SA2}(T_{SA2} - T_{ZA2}) + \bar{U}A_{Z2}(T_{OA} - T_{ZA2}) + \bar{Q}_{S2} = 0$	5.46
	$\rho h_{fg} \bar{V}_{SA2}(W_{CC} - W_{ZA2}) + \bar{Q}_{L2} = 0$	5.47
	<b>Zone 3</b>	
	$T_{SA3} \geq T_{CC}$	5.48
	$\rho c_p \bar{V}_{SA3}(T_{SA3} - T_{ZA3}) + \bar{U}A_{Z3}(T_{OA} - T_{ZA3}) + \bar{Q}_{S3} = 0$	5.49
	$\rho h_{fg} \bar{V}_{SA3}(W_{CC} - W_{ZA3}) + \bar{Q}_{L3} = 0$	5.50
	<b>Zone Temperature Errors</b>	
	$E_{TZA1} \geq T_{ZA1,SP} - T_{ZA1}$	5.51
	$E_{TZA1} \geq T_{ZA1} - T_{ZA1,SP}$	5.52
	$E_{TZA2} \geq T_{ZA2,SP} - T_{ZA2}$	5.53
	$E_{TZA2} \geq T_{ZA2} - T_{ZA2,SP}$	5.54
	$E_{TZA3} \geq T_{ZA3,SP} - T_{ZA3}$	5.55
	$E_{TZA3} \geq T_{ZA3} - T_{ZA3,SP}$	5.56
	$E_{TZA,Sum} = E_{TZA1} + E_{TZA2} + E_{TZA3}$	5.57
	<b>Energy Usage</b>	
	$\bar{E}_{Fan} = \bar{H}p_{fan} \alpha_{HpToWatts}$	5.58
	$\bar{E}_{Total} = \bar{E}_{L\&P} + \bar{E}_{Fan}$	5.59
	$\bar{Q}_{CC,S} = \rho c_p \bar{V}_{SA}(T_{CC} - T_{SF})$	5.60
	$\bar{Q}_{CC,L} = \rho h_{fg} \bar{V}_{SA}(W_{CC} - W_{MA})$	5.61
	$\bar{Q}_{CC,Total} = \bar{Q}_{CC,S} + \bar{Q}_{CC,L}$	5.62
$\bar{Q}_{RH1} = \rho c_p \bar{V}_{SA1}(T_{SA1} - T_{CC})$	5.63	
$\bar{Q}_{RH2} = \rho c_p \bar{V}_{SA2}(T_{SA2} - T_{CC})$	5.64	
$\bar{Q}_{RH3} = \rho c_p \bar{V}_{SA3}(T_{SA3} - T_{CC})$	5.65	
$\bar{Q}_{RH,Total} = \bar{Q}_{RH1} + \bar{Q}_{RH2} + \bar{Q}_{RH3}$	5.66	

**Figure 43 Continued**

This model was formulated to represent normal operating conditions in addition to scenarios where all zone temperature setpoints can't be achieved. Factors that limit

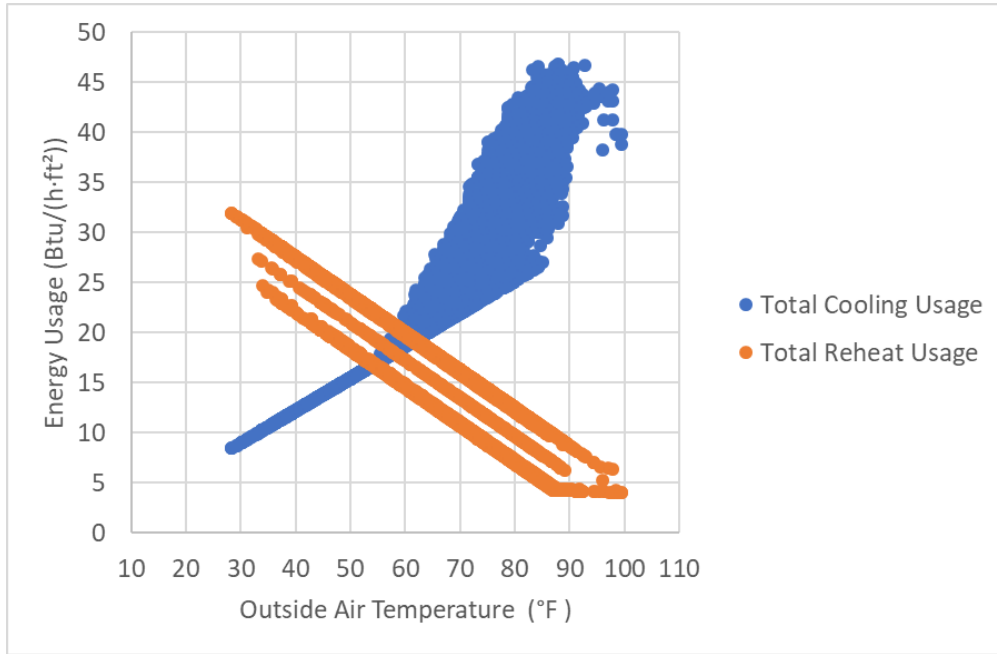
the model's validity are all assumptions that can be modified if desired. For instance, functions for air and water properties can be used instead of constant values. Also, water saturation in places other than the cooling coil can be modeled by limiting humidity ratios throughout a system to the local saturation humidity ratio.

### *SDCAV Simulation Results*

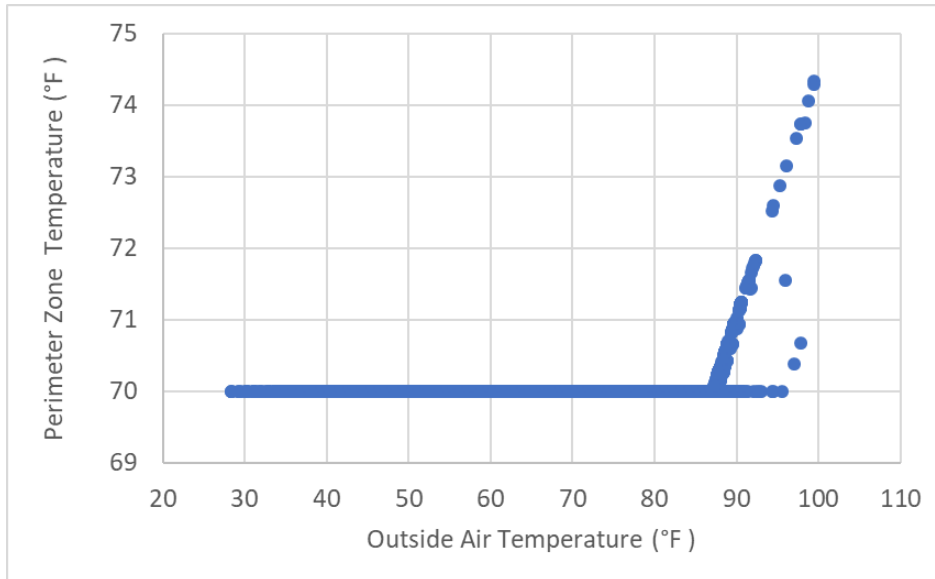
Figure 44 shows the total cooling and reheat usages for the SDCAV system with a time step period of one hour. The system performs cooling for the entire simulation year, with less cooling required during a lower outside air temperature. In cool temperatures with no latent cooling the total cooling usage trend forms a straight line. However, at higher temperatures the total cooling trend forms a cloud due to the different levels of humidity in the air that must be removed by the cooling coil.

Total reheat usage from the building's 3 zones decreases with the outside air temperature until the perimeter zone uses no reheat. With a constant chilled air flow and no reheat the system loses the ability to keep the perimeter zone's temperature at its set point at high outside air temperatures. Figure 45 shows the actual perimeter zone temperature versus the outside air temperature. Perimeter zone temperatures above  $70^{\circ}F$  are time steps where the zone set point cannot be met.

Bands that appear in the perimeter zone temperature and total reheat usage plots are due to the discretization of load schedules. Discretizing the load schedules on one hour increments results in five possible values. Figure 40 shows the hourly values used for the weekday lighting and plug load schedule. Using a smaller time step periods would result in more bands since a larger range of loads would be simulated.



**Figure 44 – SDCAV Heating and Cooling Usages**

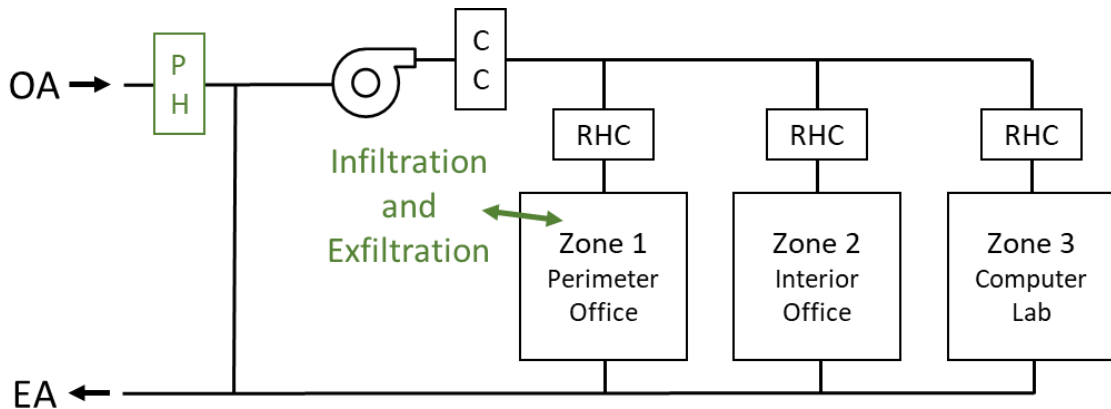


**Figure 45 – Loss of Perimeter Zone Temperature Control in the SDCAV Simulation**

### *Modifying the Air Handler Equipment*

Modifying source code for building energy simulation tools to handle new equipment or equipment configurations can be difficult. With a model based on constrained optimization modifying the objectives and constraints can account for different system behaviors.

Figure 46 shows the system of Figure 42 modified to include outside air preheat and the effect of infiltration and exfiltration in the perimeter zone. Table 21 gives the two additional parameters used in this model: a preheat coil set point of  $45^{\circ}F$  and a maximum perimeter zone outside air exchange rate of  $1.2 \frac{\text{Air Changes}}{h}$ . The energy source of the preheat coil is assumed to be hot water, the same as the reheat coils. Infiltration and exfiltration rates are assumed to be equal, due to badly placed supply and return ducts.



**Figure 46 – The SDCAV Model of Figure 42 with Outside Air Preheat and Perimeter Zone Infiltration**

**Table 21 – Added Parameters for SDCAV Model Changes**

Parameter	Symbol	Value	Unit
Preheat Coil Set Point	$T_{PH,SP}$	45	$^{\circ}F$
Outside Air Changes per Hour for Zone 1	$ACH_{Z1,Max}$	1.2	$\frac{\text{Air Changes}}{h}$

Equation 5.67 gives the maximum perimeter zone outside air exchange volume flow rate in terms of the air changes per hour, zone volume, and zone floor area. Table 22 gives the calculated maximum value for this parameter. The exchange rate experienced at a given moment is assumed to scale with the occupancy of the building. Equation 5.68 gives the actual air exchange rate to use on a time step of a simulation.

$$\bar{V}_{Inf/Exf,Z1,Max} = \frac{ACH_{Z1,Max} V_{Z1}}{60A_{Z1}} \quad 5.67$$

**Table 22 – Calculated Parameters for SDCAV Model Changes**

Parameter	Symbol	Value	Unit
Infiltration/Exfiltration Flow Rate of Zone 1	$\bar{V}_{Inf/Exf,Z1,Max}$	0.3	$\frac{CFM}{ft^2}$

$$\bar{V}_{Inf/Exf,Z1} = X_{People} \bar{V}_{Inf/Exf,Z1,Max} \quad 5.68$$

Modifying the SDCAV model to include infiltration and exfiltration involves adding terms to the perimeter zone's sensible and latent heat balance equations. For balancing air flows it's assumed that the perimeter zone has equal infiltration and exfiltration rates. Equation 5.43 of the original SDCAV model is modified into equation 5.69 and equation 5.44 of the SDCAV model is modified into equation 5.70.

$$\rho c_p \bar{V}_{SA1} (T_{SA1} - T_{ZA1}) + \rho c_p \bar{V}_{Inf/Exf,Z1} (T_{OA} - T_{ZA1}) + \bar{U} \bar{A}_{Z1} (T_{OA} - T_{ZA1}) + \bar{Q}_{S1} = 0 \quad 5.69$$

$$\rho h_{fg} \bar{V}_{SA1} (W_{CC} - W_{ZA1}) + \rho h_{fg} \bar{V}_{Inf/Exf,Z1} (W_{OA} - W_{ZA1}) + \bar{Q}_{L1} = 0 \quad 5.70$$

Modifying the SDCAV model to reflect the preheat equipment change involves adding two equations and modifying a third. Equation 5.71 is added gives the preheat coil leaving air temperature, taking the preheat coil set point into account, and equation 5.72 gives the preheat coil hot water usage. Equation 5.34 of the original SDCAV

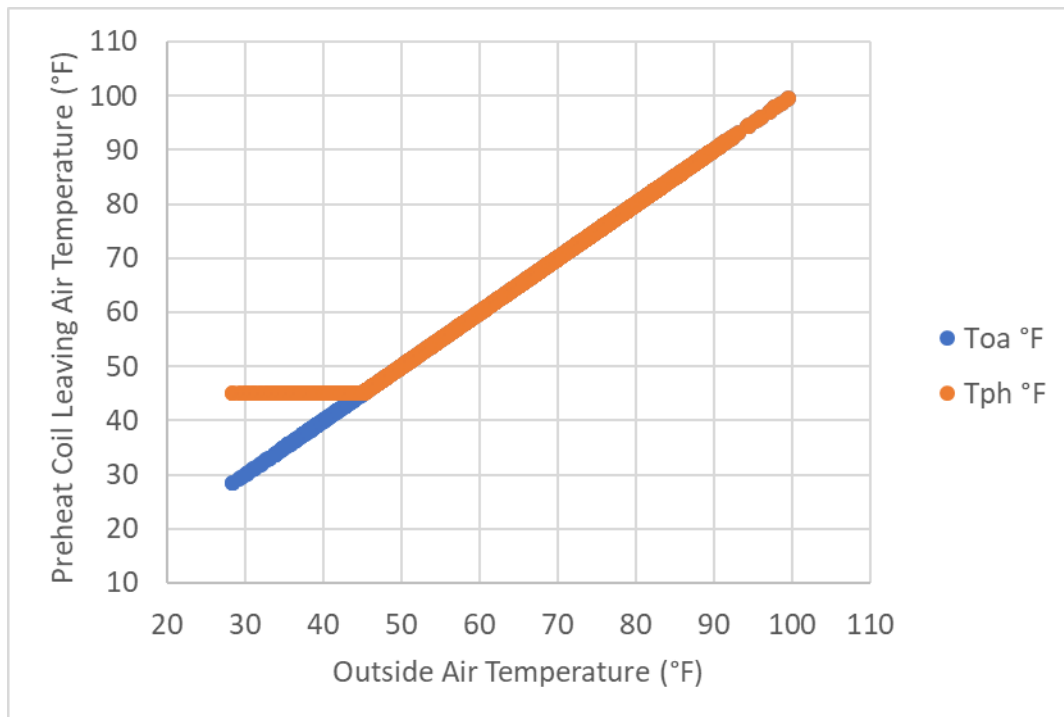
model is modified into equation 5.73 to account for outside air passing through a preheat coil before being replaced with return air.

$$T_{PH} = \text{Max}(T_{OA}, T_{PH,SP}) \quad 5.71$$

$$\bar{Q}_{PH} = \rho c_p \bar{V}_{OA} (T_{PH} - T_{OA}) \quad 5.72$$

$$\bar{V}_{SA} T_{MA} = \bar{V}_{OA} T_{PH} + (\bar{V}_{SA} - \bar{V}_{OA}) T_{RA} \quad 5.73$$

Figure 47 shows preheat coil leaving air temperatures versus outside air temperature for the modified and unmodified SDVAV models. Below the preheat coil setpoint of 45°F preheat is used to raise outside air to that temperature.

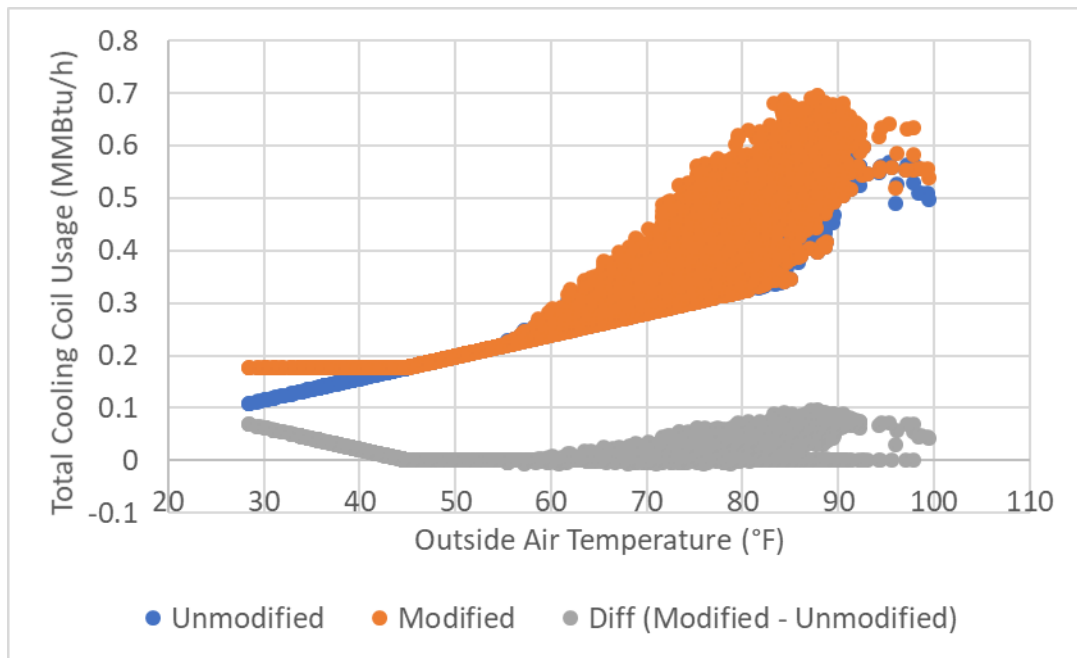


**Figure 47 – Preheat Coil Leaving Air Temperature for the Modified SDCAV Model**

Figure 48 shows cooling coil chilled water usages for the modified and unmodified SDVAV models. Cooling occurs year round in both models, so the cooling coil leaving air temperature stays constant at 55°F. At low outside air temperatures

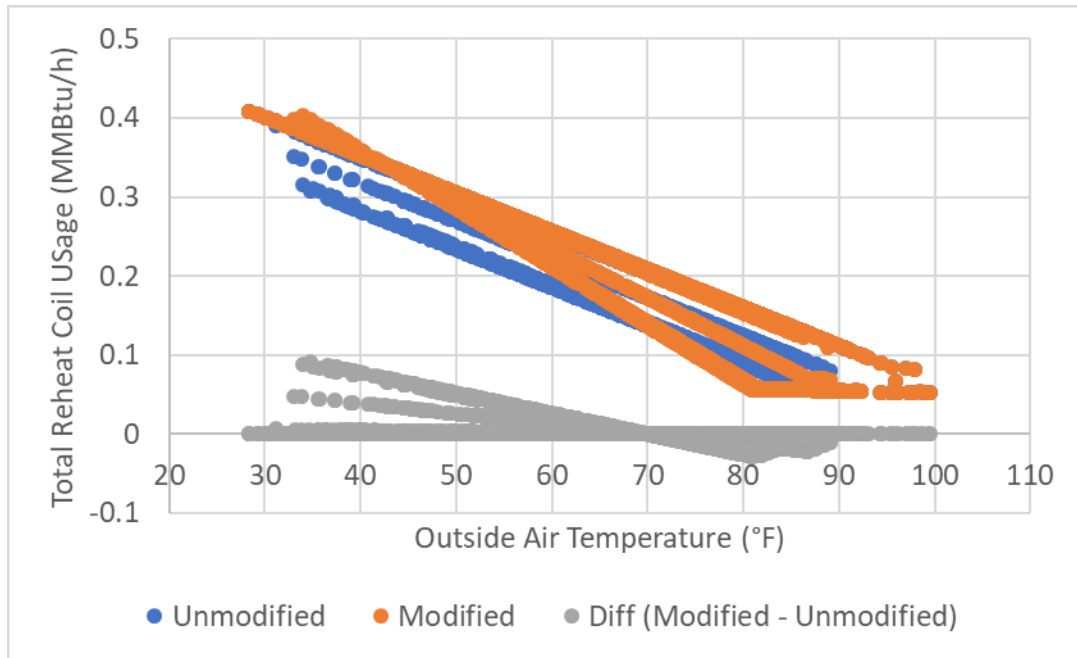


preheat usage causes the cooling coil to perform additional cooling. At higher temperatures extra cooling is necessary due to infiltration and exfiltration.



**Figure 48 – Cooling Coil Chilled Water Usages for the Modified and Unmodified Systems**

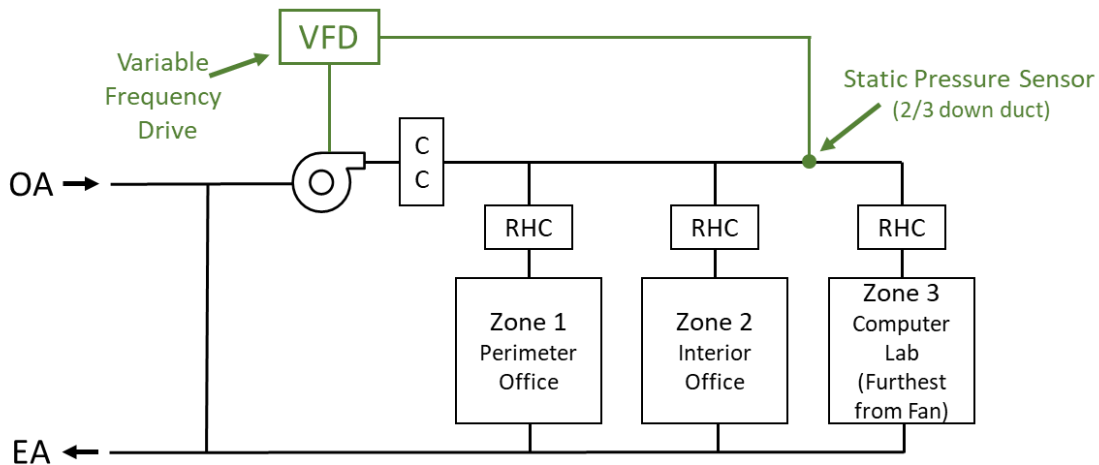
Figure 49 shows reheat usages for the modified and unmodified SDVAV models. At low temperatures less reheat is required due to infiltration and exfiltration. At high temperatures more reheat is required. Bands in reheat usages are due to the discretization of the people load profile.



**Figure 49 – Reheat Coil Hot Water Usages for the Modified and Unmodified Systems**

### Single Duct Variable Air Volume (SDVAV) Systems

The Single Duct Variable Air Volume system uses the same equipment layout as the Single Duct Constant Air Volume system of Figure 42. However, a variable speed drive powers the fan instead of running the fan at a constant speed, as shown in Figure 50. This variable speed drive maintains a static pressure setpoint approximately 2/3 of the way down the duct. Zone terminal boxes independently control dampers to provide supply air flow for cooling. Reheat is used if a zone needs heating, which only occurs when the terminal box is providing the minimum possible amount of flow.



**Figure 50 – A Single Duct Variable Air Volume System with Static Pressure Control**

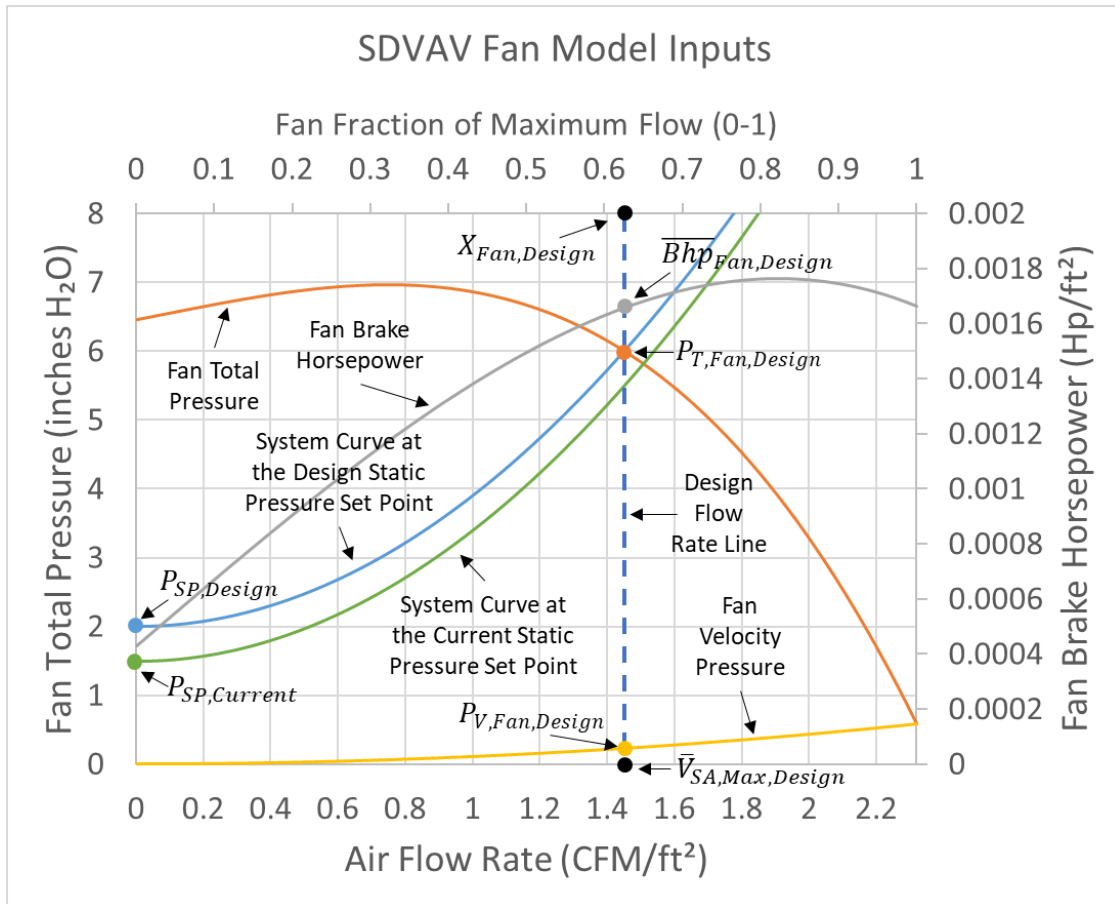
#### *SDVAV Parameters*

Table 23 gives the values of system wide parameters that are specific to the SDVAV model. The system uses a constant outside air percentage of 20% of the current supply air flow, which translates to an outside air fraction of 0.2. The cooling coil cools entering air to a fixed setpoint of 55°F.

The remaining parameters specify the operation of the fan. The motor that drives the fan has a constant efficiency of 88%. With all zone terminal box dampers fully open the system has a maximum supply air flow of  $1.45 \frac{CFM}{ft^2}$ . At a design operating fraction of 0.625 the system has a velocity pressure of  $0.25 \text{ inches } H_2O$ , a total pressure of  $6.0 \text{ inches } H_2O$ , and a total fan horsepower of  $1.3 \frac{HP}{kCFM}$ . At design conditions the system has a static pressure set point of  $2.0 \text{ inches } H_2O$ , but is currently operating with a static pressure set point of  $1.5 \text{ inches } H_2O$ . Figure 51 shows these parameters on the fan and system curves that they define.

**Table 23 – SDVAV System Parameters**

<b>Parameter</b>	<b>Symbol</b>	<b>Value</b>	<b>Unit</b>
Outside Air Fraction	$X_{OA}$	0.2	0 – 1
Cooling Coil Set Point Temperature	$T_{CC,SP}$	55	°F
Fan Motor Efficiency	$\eta_{Motor}$	0.88	0 – 1
Maximum Supply Air Flow at Design Conditions	$\bar{V}_{SA,Max,Design}$	1.45	$\frac{CFM}{ft^2}$
Fan Design Fraction of Maximum Flow	$X_{Fan,Design}$	0.625	0 – 1
Fan Design Velocity Pressure	$P_{V,Fan,Design}$	0.25	<i>inches H<sub>2</sub>O</i>
Fan Design Total Pressure	$P_{T,Fan,Design}$	6.0	<i>inches H<sub>2</sub>O</i>
Fan Design Total Horsepower	$\bar{H}p_{Fan,Design}$	1.3	$\frac{HP}{kCFM}$
Design Static Pressure Set Point	$P_{SP,Design}$	2.0	<i>inches H<sub>2</sub>O</i>
Current Static Pressure Set Point	$P_{SP,Curr}$	1.5	<i>inches H<sub>2</sub>O</i>



**Figure 51 – Fan Curve Inputs for the SDVAV Model**

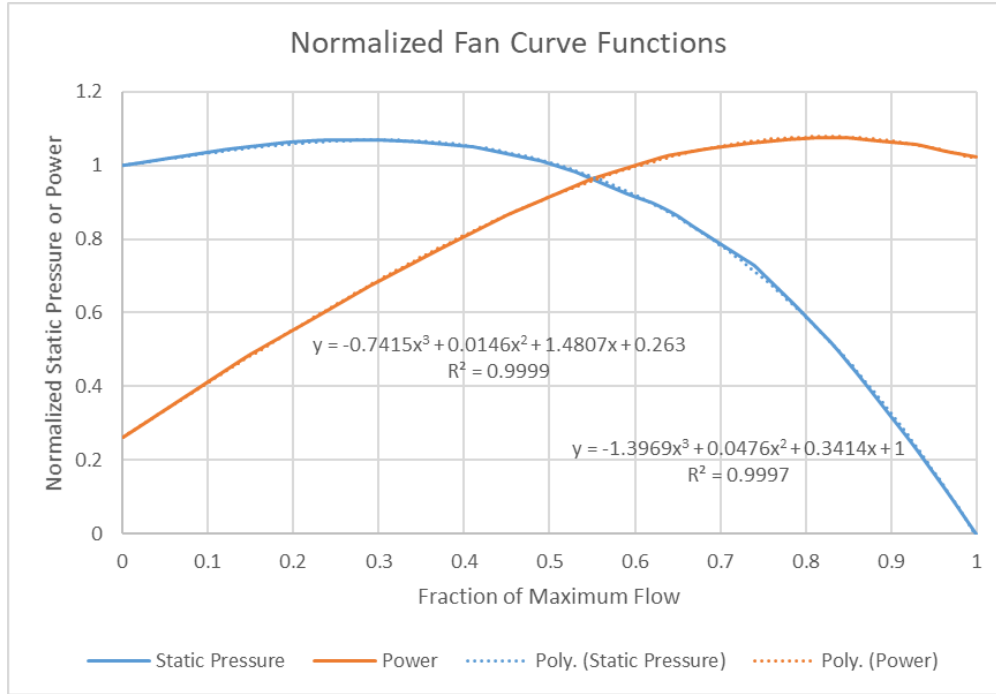
Table 24 gives zone parameters for the SDVAV system. Heating and cooling setpoints and minimum and maximum supply air flow settings vary by zone. These minimum and maximum supply air flow settings control how much air terminal boxes will try to provide. The fan system is modeled separately. The fan and variable speed drive’s physical characteristics and the static pressure set point ultimately determine how much air flow the system can provide. This model assumes that the fan can at least provide enough air flow to meet the minimum terminal box flow rates.

**Table 24 – SDVAV Zone Parameters**

Parameter	Symbol (Zone i)	Zone 1 (Perimeter Office)	Zone 2 (Interior Office)	Zone 3 (Computer Lab)	Unit
Zone Heating Set Point Temperature	$T_{ZAi,HeatingSP}$	68	68	67	$^{\circ}F$
Zone Cooling Set Point Temperature	$T_{ZAi,CoolingSP}$	73	73	71	$^{\circ}F$
Minimum Supply Air Flow (Normalized)	$\bar{V}_{SAi,Min}$	0.3	0.3	0.3	$\frac{CFM}{ft^2}$
Maximum Supply Air Flow (Normalized)	$\bar{V}_{SAi,Max}$	1.5	1.0	2.0	$\frac{CFM}{ft^2}$

*Fan Curve Functions*

Figure 52 shows the shape of typical static pressure and brake horsepower curves for a backward curved centrifugal fan [73]. This figure also gives cubic polynomials fit to this data that can be used in a constrained optimization based SDVAV model. The curves and equations shown in this figure and equations 5.74 and 5.75 are unitless. They represent the relative magnitude of the fan static pressure and brake horsepower curves as the fraction of a fan's maximum open flow increases and must be scaled by an appropriate factor in an actual model.



**Figure 52 – Typical Static Pressure and Brake Horsepower Power Curves for a Backward Curved Centrifugal Fan**

$$\hat{P}_{S,Fan}(X_{Fan}) = -1.3969X_{Fan}^3 + 0.0476X_{Fan}^2 + 0.3414X_{Fan} + 1 \quad 5.74$$

$$\widehat{Bhp}_{Fan}(X_{Fan}) = -0.7415X_{Fan}^3 + 0.0146X_{Fan}^2 + 1.4807X_{Fan} + 0.263 \quad 5.75$$

The typical fan curve shapes used came in the form of static pressure and brake horsepower curves in terms of a fan's flow with zero resistance. However, the SDVAV model uses fan total pressure and brake horsepower in terms of actual supply air flow rates. Creating these functions involves scaling and converting equations in the form of equations 5.67 and 5.77 into the form shown in equations 5.78 and 5.79 with units of *psi* and  $\frac{Hp}{ft^2}$ .

$$\hat{P}_{S,Fan}(X_{Fan}) = \hat{a}_3X_{Fan}^3 + \hat{a}_2X_{Fan}^2 + \hat{a}_1X_{Fan} + \hat{a}_0 \quad 5.76$$

$$\widehat{Bhp}_{Fan}(X_{Fan}) = \hat{b}_3X_{Fan}^3 + \hat{b}_2X_{Fan}^2 + \hat{b}_1X_{Fan} + \hat{b}_0 \quad 5.77$$

$$P_{T,Fan}(\bar{V}_{SA}) = a_3\bar{V}_{SA}^3 + a_2\bar{V}_{SA}^2 + a_1\bar{V}_{SA} + a_0 \quad 5.78$$

$$\overline{Bhp}_{Fan}(\bar{V}_{SA}) = b_3 \bar{V}_{SA}^3 + b_2 \bar{V}_{SA}^2 + b_1 \bar{V}_{SA} + b_0 \quad 5.79$$

A fan's total pressure is the sum of the fan's static and velocity pressures. Equation 5.80 shows this balance at design conditions. Replacing the values in equation 5.80 for known input variables results in equation 5.81. In equation 5.81 only  $\alpha_{Ps}$  is known, so it can be solved for to give equation 5.82.

$$P_{T,Fan}(X_{Fan,Design}) = P_{S,Fan}(X_{Fan,Design}) + P_{V,Fan}(X_{Fan,Design}) \quad 5.80$$

$$P_{T,Fan,Design} = \alpha_{Ps} \hat{P}_{S,Fan}(X_{Fan,Design}) + P_{V,Fan,Design} \quad 5.81$$

$$\alpha_{Ps} = \frac{P_{T,Fan,Design} - P_{V,Fan,Design}}{\hat{P}_{S,Fan}(X_{Fan,Design})} \quad 5.82$$

Equation 5.83 gives the total pressure drop across a fan in terms of a general a general  $X_{Fan}$ . Substituting equation 5.74 for the normalized static pressure drop and adding the equation for fan velocity pressure gives equation 5.84. Rearranging this in terms of polynomial coefficients of  $X_{Fan}$  gives equation 5.85.

$$P_{T,Fan}(X_{Fan}) = \alpha_{Ps} \hat{P}_{S,Fan}(X_{Fan}) + P_{V,Fan}(X_{Fan}) \quad 5.83$$

$$P_{T,Fan}(X_{Fan}) = \alpha_{Ps} (\hat{a}_3 X_{Fan}^3 + \hat{a}_2 X_{Fan}^2 + \hat{a}_1 X_{Fan} + \hat{a}_0) + \frac{P_{V,Fan,Design}}{X_{Fan,Design}^2} X_{Fan}^2 \quad 5.84$$

$$P_{T,Fan}(X_{Fan}) = \alpha_{Ps} \hat{a}_3 X_{Fan}^3 + \left( \alpha_{Ps} \hat{a}_2 + \frac{P_{V,Fan,Design}}{X_{Fan,Design}^2} \right) X_{Fan}^2 + \alpha_{Ps} \hat{a}_1 X_{Fan} + \alpha_{Ps} \hat{a}_0 \quad 5.85$$

Equation 5.86 gives the relationship between a fan's air flow and its fraction of maximum flow at design conditions. The variable  $\zeta$  is defined in equation 5.86 to reduce the length of other equations. Substituting  $X_{Fan}$  in equation 5.85 in terms of  $\bar{V}_{SA}$  results in equation 5.89, the fan total pressure curve used in the SDVAV models.



$$\frac{\bar{V}_{SA,Max,Design}}{\bar{V}_{SA}} = \frac{X_{Fan,Design}}{X_{Fan}} \quad 5.86$$

$$X_{Fan} = \frac{X_{Fan,Design}}{\bar{V}_{SA,Max,Design}} \bar{V}_{SA} = \zeta \bar{V}_{SA} \quad 5.87$$

$$\zeta = \frac{X_{Design}}{\bar{V}_{SA,Max,Design}} \quad 5.88$$

$$P_{T,Fan}(\bar{V}_{SA}) = \alpha_{Ps} \hat{a}_3 \zeta^3 \bar{V}_{SA}^3 + \left( \alpha_{Ps} \hat{a}_2 + \frac{P_{V,Fan,Design}}{X_{Fan,Design}^2} \right) \zeta^2 \bar{V}_{SA}^2 + \alpha_{Ps} \hat{a}_1 \zeta \bar{V}_{SA} + \alpha_{Ps} \hat{a}_0 \quad 5.89$$

Equation 5.90 gives the fan design horsepower normalized to floor area instead of  $\frac{HP}{kCFM}$ . Taking motor efficiency into account gives the brake horsepower in equation 5.91. Equation 5.92 shows how the typical brake horsepower curve of equation 5.75 is scaled. 5.93 gives the formula for  $\alpha_{Bhp}$  using equation 5.92 and known parameters at design conditions. Substituting the typical brake horsepower curve of equation 5.75 into equation 5.92 gives equation 5.94. Substituting  $X_{Fan}$  in equation 5.94 in terms of  $\bar{V}_{SA}$  results in equation 5.95, the fan brake horsepower curve used in the SDVAV models. Table 25 gives the values of the fan curve coefficients used in the final model, and equations 5.96 and 5.97 give equations 5.94 and 5.95 with all values substituted.

$$\bar{H}p_{Fan,Design} = \widetilde{H}p_{Fan,Design} \bar{V}_{SA,Max,Design} \alpha_{CfmToKcfm} \quad 5.90$$

$$\bar{B}hp_{Fan,Design} = \bar{H}p_{Fan,Design} \eta_{Motor} \quad 5.91$$

$$\bar{B}hp_{Fan}(X_{Fan}) = \alpha_{Bhp} \widehat{B}hp_{Fan}(X_{Fan}) \quad 5.92$$

$$\alpha_{Bhp} = \frac{\bar{B}hp_{Fan,Design}}{\widehat{B}hp_{Fan}(X_{Fan,Design})} \quad 5.93$$

$$\bar{B}hp_{fan}(X_{Fan}) = \alpha_{Bhp} \hat{b}_3 X_{Fan}^3 + \alpha_{Bhp} \hat{b}_2 X_{Fan}^2 + \alpha_{Bhp} \hat{b}_1 X_{Fan} + \alpha_{Bhp} \hat{b}_0 \quad 5.94$$

$$\bar{B}hp_{fan}(\bar{V}_{SA}) = \alpha_{Bhp} \hat{b}_3 \zeta^3 \bar{V}_{SA}^3 + \alpha_{Bhp} \hat{b}_2 \zeta^2 \bar{V}_{SA}^2 + \alpha_{Bhp} \hat{b}_1 \zeta \bar{V}_{SA} + \alpha_{Bhp} \hat{b}_0 \quad 5.95$$

**Table 25 – Denormalized Fan Curve Coefficients for the SDVAV Model**

Coefficient	Formula	Value
$a_3$	$\alpha_{P_S} \hat{a}_3 \zeta^3$	-0.722
$a_2$	$\left( \alpha_{P_S} \hat{a}_2 + \frac{P_{V,Fan,Design}}{X_{Fan,Design}^2} \right) \zeta^2$	0.176
$a_1$	$\alpha_{P_S} \hat{a}_1 \zeta$	0.95
$a_0$	$\alpha_{P_S} \hat{a}_0$	6.45
$b_3$	$\alpha_{Bhp} \hat{b}_3 \zeta^3$	-9.7E-5
$b_2$	$\alpha_{Bhp} \hat{b}_2 \zeta^2$	4.44E-6
$b_1$	$\alpha_{Bhp} \hat{b}_1 \zeta$	1.05E-3
$b_0$	$\alpha_{Bhp} \hat{b}_0$	4.31E-4

$$P_{T,Fan}(\bar{V}_{SA}) = -0.722\bar{V}_{SA}^3 + 0.176\bar{V}_{SA}^2 + 0.95\bar{V}_{SA} + 6.45 \quad 5.96$$

$$\overline{Bhp}_{Fan}(\bar{V}_{SA}) = -9.7E^{-5}\bar{V}_{SA}^3 + 4.44E^{-6}\bar{V}_{SA}^2 + 1.05E^{-3}\bar{V}_{SA} + 4.31E^{-4} \quad 5.97$$

### *The SDVAV Model*

Figure gives a model for the SDVAV system. Like the SDCAV model of Figure , the constraints describe how equipment in the system operates, define temperature errors, and give definitions for overall energy usages over a time step period. However, this model uses lexicographic objectives to describe how the system seeks to control zone temperatures while minimizing reheat and air flow. Minimizing zone temperature error, the sum of deviations from the valid set point range serves as the highest objective. Minimizing air flow as the second objective ensures that the lowest amounts of reheat and air flow are used while zone temperatures are controlled as close to the heating/cooling set point range as possible.

minimize: (in order)	$E_{TZA}$	5.98
	$\bar{V}_{SA}$	5.99
	$\bar{Q}_{RH,Total}$	5.100
subject to:	<b>AHU Temperature Balance</b>	
	$T_{MA} = X_{OA}T_{OA} + (1 - X_{OA})T_{RA}$	5.101
	$\rho c_p \bar{V}_{SA} \Delta T_{SF} = \bar{E}_{Fan,Curr} \alpha_{KwToBtuPerHour}$	5.102
	$T_{SF} = T_{MA} + \Delta T_{SF}$	5.103
	$T_{CC} = \text{Min}(T_{SF}, T_{CC,SP})$	5.104
	$\bar{V}_{SA} A_Z T_{RA} = \bar{V}_{SA1} A_{Z1} T_{ZA1} + \bar{V}_{SA2} A_{Z2} T_{ZA2} + \bar{V}_{SA3} A_{Z3} T_{ZA3}$	5.105
	<b>AHU Humidity Balance</b>	
	$W_{MA} = X_{OA}W_{OA} + (1 - X_{OA})W_{RA}$	5.106
	$W_{CC,Sat} P_{OA} = (M_{ratio} + W_{CC,Sat}) P_{ws}(T_{CC})$	5.107
	$W_{CC} = \text{Min}(W_{MA}, W_{CC,Sat})$	5.108
	$\bar{V}_{SA} A_Z W_{RA} = \bar{V}_{SA1} A_{Z1} W_{ZA1} + \bar{V}_{SA2} A_{Z2} W_{ZA2} + \bar{V}_{SA3} A_{Z3} W_{ZA3}$	5.109
	<b>Fan Pressure and Power Calculations</b>	
	$P_{T,Fan}(\bar{V}_{SA,Max,Design}) = P_{SP,Design} + \alpha_{Sys} \bar{V}_{SA,Max,Design}^2$	5.110
	$P_{T,Fan}(\bar{V}_{SA,Max,Curr}) = P_{SP,Curr} + \alpha_{Sys} \bar{V}_{SA,Max,Curr}^2$	5.111
	$P_{T,Fan,Curr} = P_{SP,Curr} + \alpha_{Sys} \bar{V}_{SA}^2$	5.112
	$\frac{P_{T,Fan,Curr}}{P_{T,Fan,Intersect}} = \frac{\bar{V}_{SA}^2}{\bar{V}_{SA,Intersect}^2}$	5.113
	$\overline{Bhp}_{Fan,Intersect} = \overline{Bhp}_{Fan}(\bar{V}_{SA,Intersect})$	5.114
	$\frac{\overline{Bhp}_{Fan,Curr}}{\overline{Bhp}_{Fan,Intersect}} = \frac{\bar{V}_{SA}^3}{\bar{V}_{SA,Intersect}^3}$	5.115
	<b>AHU Flow Balance</b>	
	$\bar{V}_{SA} A_Z = \bar{V}_{SA1} A_{Z1} + \bar{V}_{SA2} A_{Z2} + \bar{V}_{SA3} A_{Z3}$	5.116
	$\bar{V}_{SA} \leq \bar{V}_{SA,max,curr}$	5.117
<b>Zone 1</b>		
$T_{SA1} \geq T_{CC}$	5.118	
$\rho c_p \bar{V}_{SA1} (T_{SA1} - T_{ZA1}) + \bar{U} \bar{A}_{Z1} (T_{OA} - T_{ZA1}) + \bar{Q}_{S1} = 0$	5.119	
$\rho h_{fg} \bar{V}_{SA1} (W_{CC} - W_{ZA1}) + \bar{Q}_{L1} = 0$	5.120	
$\bar{V}_{SA1,Min} \leq \bar{V}_{SA1} \leq \bar{V}_{SA1,Max}$	5.121	

Figure 53 – Model of a Single Duct Variable Air Volume System

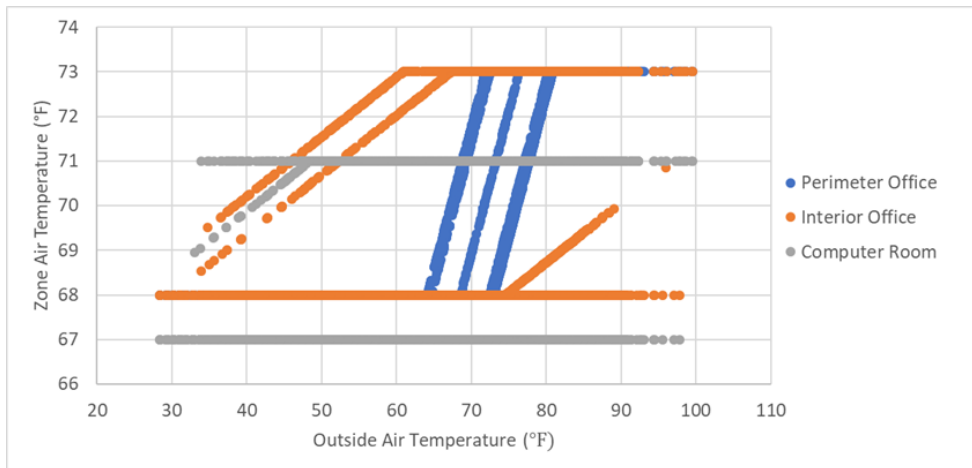
subject to:	<b>Zone 2</b>	
	$T_{SA2} \geq T_{CC}$	5.122
	$\rho c_p \bar{V}_{SA2}(T_{SA2} - T_{ZA2}) + \bar{U}A_{Z2}(T_{OA} - T_{ZA2}) + \bar{Q}_{S2} = 0$	5.123
	$\rho h_{fg} \bar{V}_{SA2}(W_{CC} - W_{ZA2}) + \bar{Q}_{L2} = 0$	5.124
	$\bar{V}_{SA2,Min} \leq \bar{V}_{SA2} \leq \bar{V}_{SA2,Max}$	5.125
	<b>Zone 3</b>	
	$T_{SA3} \geq T_{CC}$	5.126
	$\rho c_p \bar{V}_{SA3}(T_{SA3} - T_{ZA3}) + \bar{U}A_{Z3}(T_{OA} - T_{ZA3}) + \bar{Q}_{S3} = 0$	5.127
	$\rho h_{fg} \bar{V}_{SA3}(W_{CC} - W_{ZA3}) + \bar{Q}_{L3} = 0$	5.128
	$\bar{V}_{SA3,Min} \leq \bar{V}_{SA3} \leq \bar{V}_{SA3,Max}$	5.129
	<b>Zone Temperature Errors</b>	
	$E_{TZA1} \geq 0$	5.130
	$E_{TZA1} \geq T_{ZA1,HeatingSP} - T_{ZA1}$	5.131
	$E_{TZA1} \geq T_{ZA1} - T_{ZA1,CoolingSP}$	5.132
	$E_{TZA2} \geq 0$	5.133
	$E_{TZA2} \geq T_{ZA2,HeatingSP} - T_{ZA2}$	5.134
	$E_{TZA2} \geq T_{ZA2} - T_{ZA2,CoolingSP}$	5.135
	$E_{TZA3} \geq 0$	5.136
	$E_{TZA3} \geq T_{ZA3,HeatingSP} - T_{ZA3}$	5.137
	$E_{TZA3} \geq T_{ZA3} - T_{ZA3,CoolingSP}$	5.138
	$E_{TZA} = E_{TZA1} + E_{TZA2} + E_{TZA3}$	5.139
	<b>Energy Usage</b>	
	$\bar{E}_{Fan} = \frac{\bar{B}h\bar{p}_{fan,curr}\alpha_{HpToWatts}}{\eta_{Motor}}$	5.140
	$\bar{E}_{Total} = \bar{E}_{L\&P} + \bar{E}_{Fan}$	5.141
	$\bar{Q}_{CC,S} = \rho c_p \bar{V}_{SA}(T_{SF} - T_{CC})$	5.142
	$\bar{Q}_{CC,L} = \rho h_{fg} \bar{V}_{SA}(W_{MA} - W_{CC})$	5.143
	$\bar{Q}_{CC,Total} = \bar{Q}_{CC,S} + \bar{Q}_{CC,L}$	5.144
	$\bar{Q}_{RH1} = \rho c_p \bar{V}_{SA1}(T_{SA1} - T_{CC})$	5.145
	$\bar{Q}_{RH2} = \rho c_p \bar{V}_{SA2}(T_{SA2} - T_{CC})$	5.146
	$\bar{Q}_{RH3} = \rho c_p \bar{V}_{SA3}(T_{SA3} - T_{CC})$	5.147
$\bar{Q}_{RH,Total} = \bar{Q}_{RH1} + \bar{Q}_{RH2} + \bar{Q}_{RH3}$	5.148	

**Figure 53 Continued**

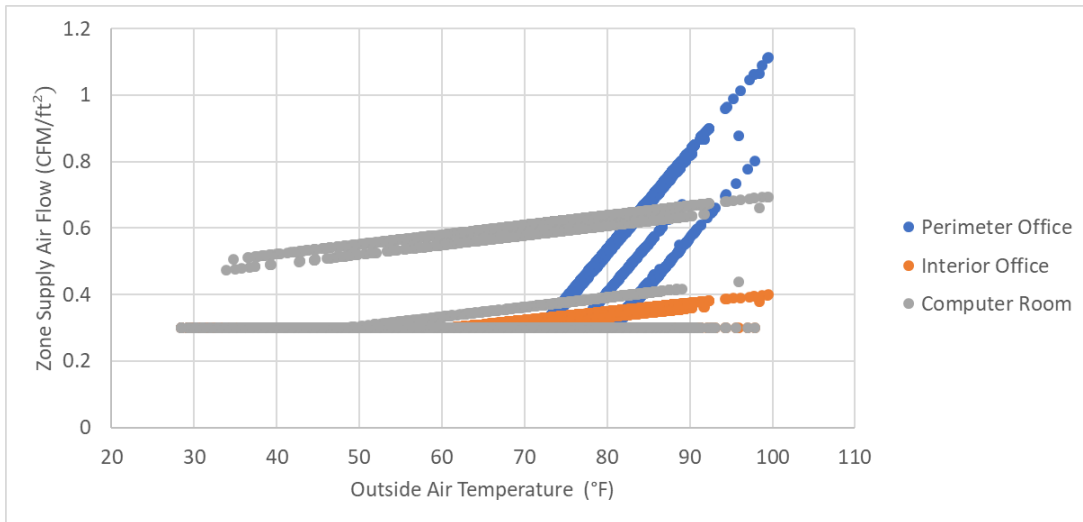
subject to:	<b>Costs</b>	
	$\overline{Cost}_{Electric} = \overline{E}_{Fan,Curr} Price_{Electric}$	5.149
	$\overline{Cost}_{ChW} = \alpha_{MMBtuPerBtu} \overline{Q}_{CC,Total} Price_{ChW}$	5.150
	$\overline{Cost}_{HW} = \alpha_{MMBtuPerBtu} \overline{Q}_{RH,Total} Price_{HW}$	5.151
	$\overline{Cost}_{Total} = \overline{Cost}_{Electric} + \overline{Cost}_{ChW} + \overline{Cost}_{HW}$	5.152

**Figure 53 Continued**

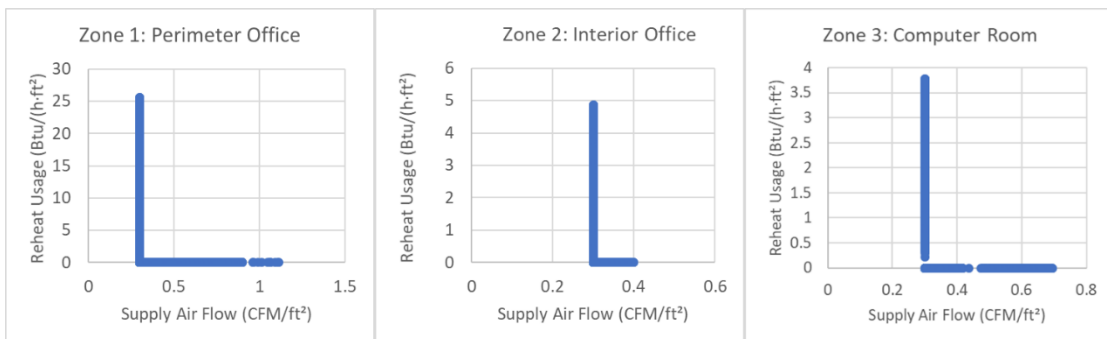
Figure 54 shows zone temperatures for the SDVAV model of Figure . Bands in this figure are due to the discretization of load schedules. As outside air temperatures and loads increase each zone transitions from heating mode to cooling mode. Figure 55 shows zone supply air flows for the model. Supply air flows also increase as outside air temperatures and loads for each zone rise. Figure 56 plots supply air flow versus reheat usage for each zone, which verifies that reheat is only used when a zone is using its minimum required air flow.



**Figure 54 – SDVAV Zone Temperatures**



**Figure 55 – SDVAV Supply Air and Zone Temperatures**



**Figure 56 – SDVAV Supply Air Flow versus Reheat Usage**

### *Optimizing Energy Cost*

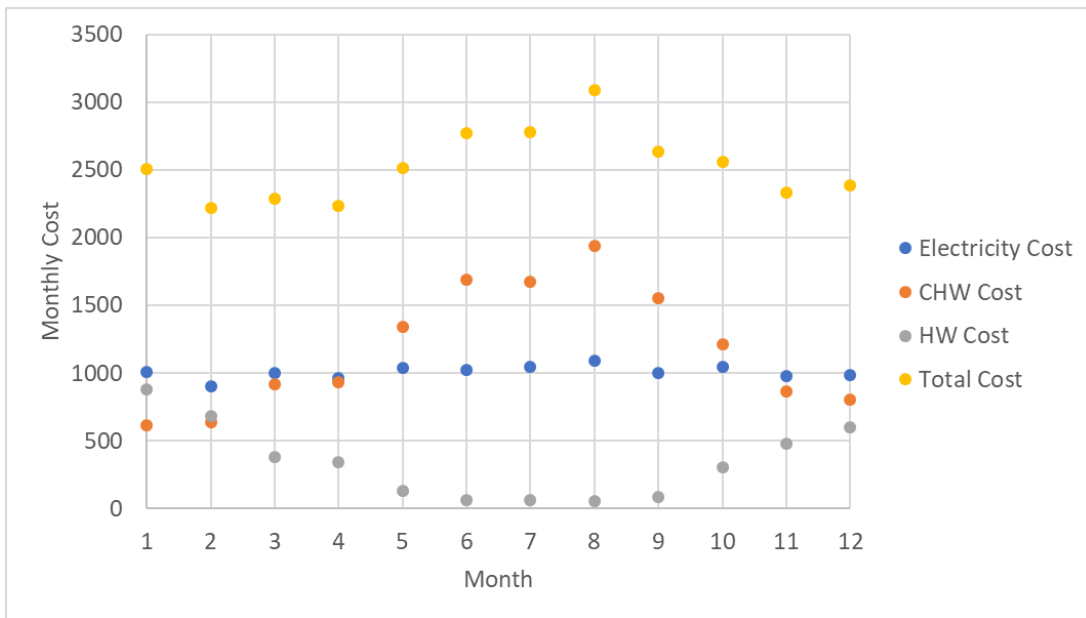
The original SDVAV model adjusts flow rates so that fan power and reheat are minimized while achieving zone temperature control. This optimization mimics the behavior of a variable air volume control system with a fan controlled to maintain a static pressure setpoint. Other control system behaviors can be simulated by modifying the model’s objective functions.

A control behavior that optimizes energy cost can be simulated by replacing the last two objective functions of the original SDVAV model with the total energy cost  $\overline{Cost}_{Total}$ . Figure 57 shows the resulting lexicographic objectives.

minimize:	$E_{TZA}$	5.153
(in order)	$Cost_{Total}$	5.154

**Figure 57 – Lexicographic Objective Functions to Optimize Energy Cost for the SDVAV Model**

Figure 58 shows the monthly energy costs for the cost optimized SDVAV model. These costs turn out to be the same as the energy costs as the original SDVAV model. This shows that the original SDVAV control strategy is an optimal control strategy for the free variables of this model under the simulated conditions.



**Figure 58 – Monthly Energy Costs for the Cost Optimized SDVAV Model**

The original and cost optimization SDVAV models both use the same lexicographic nonlinear programming structure. This allows both problems to be solved in a similar amount of time. It also allows any variable that varies on each time step be optimized with nonlinear programming methods that take advantage of a problem's gradients and second derivatives.

### *Cooling Coil Setpoint Optimization*

The cooling coil set point of the SDVAV model can be optimized on each time step by replacing its constant value with minimum and maximum bounds. This involves adding equation 5.155 to the original SDVAV model. During a simulation time step the cooling coil set point varies freely between the two bounds while the objective functions force it to a value that optimizes cost while achieving zone temperature control. Table 26 shows the values of the bounds used in simulations.

$$T_{CC,SP,Min} \leq T_{CC,SP} \leq T_{CC,SP,Max} \quad 5.155$$

**Table 26 – SDVAV Cooling Coil Setpoint Optimization Additional Parameters**

Parameter	Symbol	Value	Unit
Minimum Allowable Cooling Coil Set Point Temperature	$T_{CC,SP,Min}$	53	$^{\circ}F$
Maximum Allowable Cooling Coil Set Point Temperature	$T_{CC,SP,Max}$	65	$^{\circ}F$

In general, constant variables that vary from time step to time step can be optimized by removing their constant values. Minimum and maximum bounds can then be added to constraint parameters within a specified range. This allows optimal parameter settings to be calculated without having to iterate on multiple simulations.

High cooling coil setpoints reduce the dehumidification abilities of SDVAV systems. Taking humidity control into account avoids settings that result in high zone relative humidities. To implement this, first equations 5.156–5.158 are added to the SDVAV cost optimization model for calculating the relative humidity of each zone. Next, equations 5.159– 5.165 are added to define the relative humidity control error. The relative humidity error of each zone is defined as the amount its relative humidity goes above a maximum allowed value. Finally, the total relative humidity error from



equation 5.165 is added. This total error is defined as the sum of the individual zone relative humidity errors. No minimum relative humidity limits are given, and the equations 5.156–5.165 allow for relative humidities over 100% since saturation isn't modeled.

$$P_{ws}(T_{ZA1})(W_{ZA1} + M_{ratio})Rh_{ZA1} = 100P_{OA}W_{ZA1} \quad 5.156$$

$$P_{ws}(T_{ZA2})(W_{ZA2} + M_{ratio})Rh_{ZA2} = 100P_{OA}W_{ZA2} \quad 5.157$$

$$P_{ws}(T_{ZA3})(W_{ZA3} + M_{ratio})Rh_{ZA3} = 100P_{OA}W_{ZA3} \quad 5.158$$

$$E_{RhZA1} \geq 0 \quad 5.159$$

$$E_{RhZA1} \geq Rh_{ZA1} - Rh_{Max,Z1} \quad 5.160$$

$$E_{RhZA2} \geq 0 \quad 5.161$$

$$E_{RhZA2} \geq Rh_{ZA2} - Rh_{Max,Z2} \quad 5.162$$

$$E_{RhZA3} \geq 0 \quad 5.163$$

$$E_{RhZA3} \geq Rh_{ZA3} - Rh_{Max,Z3} \quad 5.164$$

$$E_{RhZA} = E_{RhZA1} + E_{RhZA2} + E_{RhZA3} \quad 5.165$$

The objective functions of the original cooling coil setpoint optimization model ensure that energy costs are minimized while keeping zone temperatures as close as possible to their set points. Adding relative humidity control so that zones avoid high humidity levels involves adding an additional objective function to minimize the overall zone relative humidity error  $E_{RhZA}$ . This objective function is placed between the zone temperature error minimization and cost minimization as shown in Figure 59. This ordering prioritizes temperature control over humidity control and humidity control is prioritized over energy cost.

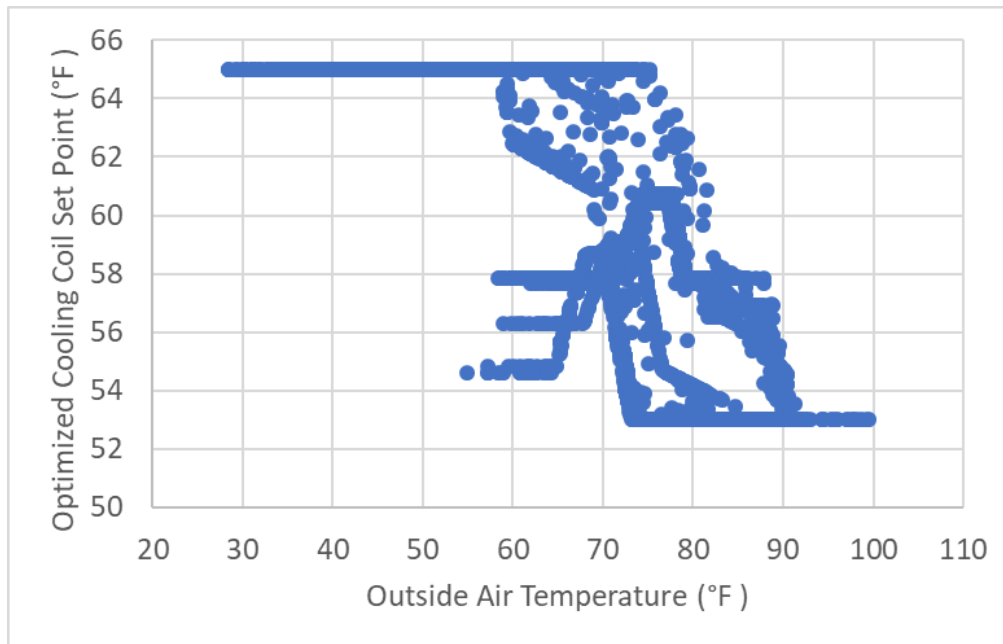
Turning the cooling coil set point from a constant to a variable creates an additional degree of freedom in the simulation state. With zone temperature setpoint ranges it's possible that multiple cooling coil set points will achieve the proper control. To resolve these ambiguities a fourth objective function is added to use the highest cooling coil set point that achieves zone temperature and humidity control. Figure 59

shows this fourth objective of equation 5.169 placed after the first three objectives of equations 5.166–5.168. This causes a nonlinear programming algorithm to choose the highest cooling coil set point when multiple set points achieve the same cost optimization.

minimize: (in order)	$E_{TZA}$	5.166
	$E_{RHZA}$	5.167
	$\overline{Cost}_{Total}$	5.168
	$-T_{CC,SP}$	5.169

**Figure 59 – New Objective Functions for Achieving Relative Humidity Control while Optimizing Cooling Coil Setpoints**

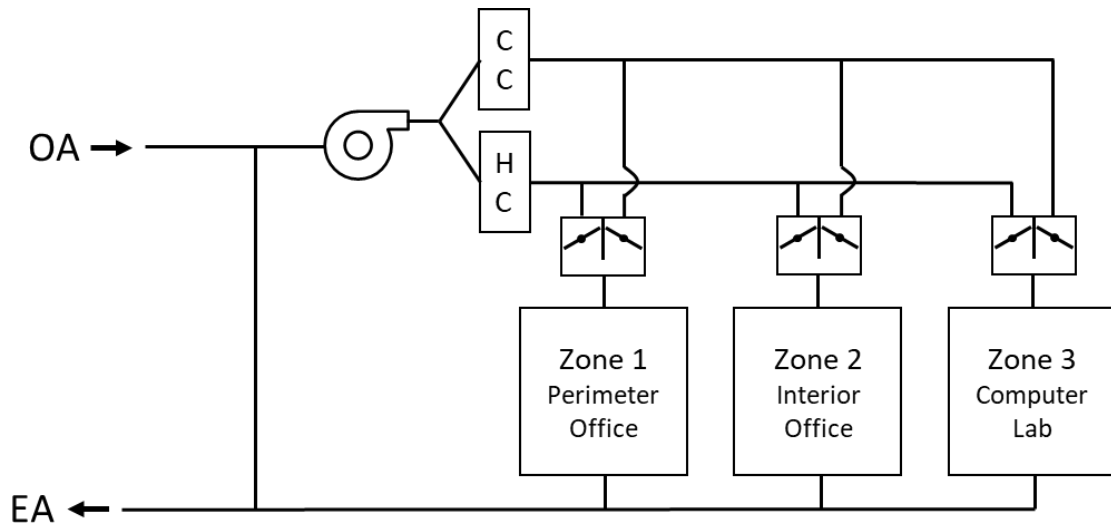
Figure 60 shows the cooling coil set points that optimize energy costs while maintaining zone temperature and relative humidity control. At high temperatures a low cooling coil set point is required, while at lower temperatures a higher value can be used. Yearly energy costs for the building amount to \$30,326 when a cooling coil set point of  $55^{\circ}F$  is used, and \$26,528 when the cooling coil set point is optimized. This yearly energy savings of 12.5% can serve as an upper bound on the energy savings potential of using a cooling coil schedule instead of a constant value.



**Figure 60 – Optimal Cooling Coil Set Points for the SDVAV Cost Optimization Model**

### **Dual Duct Variable Air Volume (DDVAV) Systems**

Dual Duct Variable Air Volume systems supply cooled and heated air to zones using separate ducts. One duct supplies cool air, the other supplies heated air, and mixing boxes mixes controlled amounts of air from each duct before supplying the mixed air to zones. The schematic of Figure 61 shows a version of this system that uses one fan to pressurize both ducts.



**Figure 61 – DDVAV System Layout**

The following DDVAV model uses the same zone cooling and heating setpoints and minimum and maximum air flow rates as the SDVAV system. These parameters are given in Table 24. Outside air for DDVAV simulations is kept at a constant rate of 20% of the supply air flow. The cold deck setpoint is a constant  $55^{\circ}F$  and the hot deck setpoint is a constant  $120^{\circ}F$ . These parameters along with the fan temperature rise and power at maximum flow are given in Table 27. Equation 5.170 gives the fan power at maximum flow translated into  $\frac{W}{ft^2}$ , which is given in Table 28.

**Table 27 – DDVAV System Parameters**

Parameter	Symbol	Value	Unit
Outside Air Fraction	$X_{OA}$	0.2	0 – 1
Cold Deck Set Point Temperature	$T_{CD,SP}$	55	$^{\circ}F$
Hot Deck Set Point Temperature	$T_{HD,SP}$	120	$^{\circ}F$
Maximum Fan Delta-T	$\Delta T_{SF,Max}$	2.0	$^{\circ}F$
Fan Maximum Total Horsepower	$\tilde{H}p_{Fan,Design}$	0.8	$\frac{HP}{kCFM}$

$$\bar{E}_{Fan} = \bar{H} \bar{p}_{Fan,Design} \bar{V}_{SA,Max,Design} \alpha_{CfmToKCfm} \alpha_{HpToWatts} \quad 5.170$$

**Table 28 – Calculated DDVAV System Parameters**

Parameter	Symbol	Value	Unit
Fan Maximum Energy Usage	$\bar{E}_{Fan,Max}$	1.54	$\frac{W}{ft^2}$

A constrained optimization based DDVAV model is given in Figure . Airflow balance, temperature balance, and humidity balance equations are modified from the SDCAV and SDVAV models to account for the new duct layout. Also, the supply fan temperature rise and energy usage are assumed to be quadratic functions of the current fraction of supply air flow.

For system control objectives the zone temperature error is minimized first so that supply air flow rates can take whatever values necessary to achieve zone temperature control. Next, supply air flow is minimized since DDVAV control systems are designed to use as little flow as possible. Minimizing this objective using heating and cooling set points at a zone’s minimum flow can result in situations where a variety of cold and hot deck flows achieve a valid zone temperature. To give a single solution, the total hot deck flow is minimized as the third objective so that air flow through the cold deck is preferred.

minimize: (in order)	$E_{TZA,Sum}$	5.171
	$\bar{V}_{SA}$	5.172
	$\bar{V}_{HD}$	5.173
subject to:	<b>AHU Air Flow Balance</b>	
	$\bar{V}_{SA}A_Z = \bar{V}_{SA1}A_{Z1} + \bar{V}_{SA2}A_{Z2} + \bar{V}_{SA3}A_{Z3}$	5.174
	$\bar{V}_{SA} = \bar{V}_{CD} + \bar{V}_{HD}$	5.175
	$\bar{V}_{CD}A_Z = \bar{V}_{CD1}A_{Z1} + \bar{V}_{CD2}A_{Z2} + \bar{V}_{CD3}A_{Z3}$	5.176
	$\bar{V}_{HD}A_Z = \bar{V}_{HD1}A_{Z1} + \bar{V}_{HD2}A_{Z2} + \bar{V}_{HD3}A_{Z3}$	5.177

**Figure 62 – Model of a Dual Duct Variable Air Volume System (Continued Below)**

subject to:	<b>AHU Temperature Balance</b>	
	$\bar{V}_{SA}T_{MA} = \bar{V}_{OA}T_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})T_{RA}$	5.178
	$\Delta T_{SF} = \Delta T_{SF,Max} \left( \frac{\bar{V}_{SA}}{\bar{V}_{SA,Max}} \right)^2$	5.179
	$T_{SF} = T_{MA} + \Delta T_{SF}$	5.180
	$T_{CD} = \text{Min}(T_{SF}, T_{CD,SP})$	5.181
	$T_{HD} = \text{Max}(T_{SF}, T_{HD,SP})$	5.182
	$\bar{V}_{SA}A_ZT_{RA} = \bar{V}_{SA1}A_{Z1}T_{ZA1} + \bar{V}_{SA2}A_{Z2}T_{ZA2} + \bar{V}_{SA1}A_{Z3}T_{ZA3}$	5.183
	<b>AHU Humidity Balance</b>	
	$\bar{V}_{SA}W_{MA} = \bar{V}_{OA}W_{OA} + (\bar{V}_{SA} - \bar{V}_{OA})W_{RA}$	5.184
	$W_{CD,sat}P_{OA} = (M_{ratio} + W_{CD,sat})P_{ws}(T_{CD})$	5.185
	$W_{CD} = \text{Min}(W_{MA}, W_{CD,sat})$	5.186
	$\bar{V}_{SA}A_ZW_{RA} = \bar{V}_{SA1}A_{Z1}W_{ZA1} + \bar{V}_{SA2}A_{Z2}W_{ZA2} + \bar{V}_{SA3}A_{Z3}W_{ZA3}$	5.187
	<b>Zone 1</b>	
	$\bar{V}_{SA1}T_{SA1} = \bar{V}_{CD1}T_{CD} + \bar{V}_{HD1}T_{HD}$	5.188
	$\rho c_p \bar{V}_{SA1}(T_{SA1} - T_{ZA1}) + \bar{U}A_{Z1}(T_{OA} - T_{ZA1}) + \bar{Q}_{S1} = 0$	5.189
	$\bar{V}_{SA1}W_{SA1} = \bar{V}_{CD1}W_{CD} + \bar{V}_{HD1}W_{MA}$	5.190
	$\rho h_{fg} \bar{V}_{SA1}(W_{SA1} - W_{ZA1}) + \bar{Q}_{L1} = 0$	5.191
	$\bar{V}_{SA1} = \bar{V}_{CD1} + \bar{V}_{HD1}$	5.192
	$\bar{V}_{SA1,Min} \leq \bar{V}_{SA1} \leq \bar{V}_{SA1,Max}$	5.193
	$\bar{V}_{CD1} \geq 0$	5.194
	$\bar{V}_{HD1} \geq 0$	5.195
	<b>Zone 2</b>	
	$\bar{V}_{SA2}T_{SA2} = \bar{V}_{CD2}T_{CD} + \bar{V}_{HD2}T_{HD}$	5.196
	$\rho c_p \bar{V}_{SA2}(T_{SA2} - T_{ZA2}) + \bar{U}A_{Z2}(T_{OA} - T_{ZA2}) + \bar{Q}_{S2} = 0$	5.197
	$\bar{V}_{SA2}W_{SA2} = \bar{V}_{CD2}W_{CD} + \bar{V}_{HD2}W_{MA}$	5.198
	$\rho h_{fg} \bar{V}_{SA2}(W_{SA2} - W_{ZA2}) + \bar{Q}_{L2} = 0$	5.199
	$\bar{V}_{SA2} = \bar{V}_{CD2} + \bar{V}_{HD2}$	5.200
	$\bar{V}_{SA2,Min} \leq \bar{V}_{SA2} \leq \bar{V}_{SA2,Max}$	5.201
	$\bar{V}_{CD2} \geq 0$	5.202
$\bar{V}_{HD2} \geq 0$	5.203	

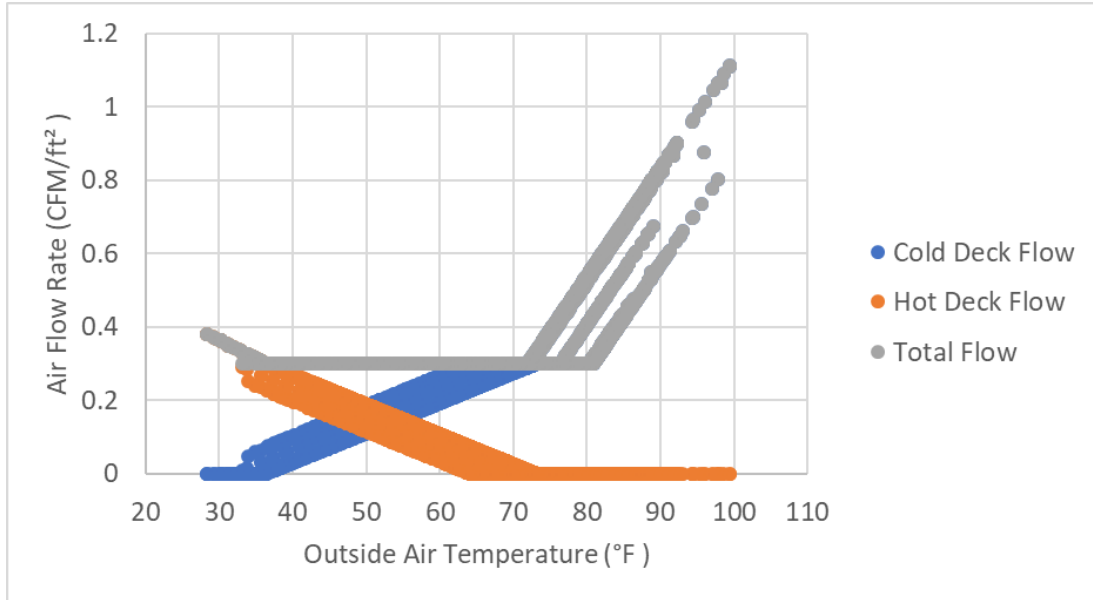
**Figure 62 Continued**

subject to:	<b>Zone 3</b>	
	$\bar{V}_{SA3}T_{SA3} = \bar{V}_{CD3}T_{CD} + \bar{V}_{HD3}T_{HD}$	5.204
	$\rho c_p \bar{V}_{SA3}(T_{SA3} - T_{ZA3}) + \bar{U}A_{Z3}(T_{OA} - T_{ZA3}) + \bar{Q}_{S3} = 0$	5.205
	$\bar{V}_{SA3}W_{SA3} = \bar{V}_{CD3}W_{CD} + \bar{V}_{HD3}W_{MA}$	5.206
	$\rho h_{fg} \bar{V}_{SA3}(W_{SA3} - W_{ZA3}) + \bar{Q}_{L3} = 0$	5.207
	$\bar{V}_{SA3} = \bar{V}_{CD3} + \bar{V}_{HD3}$	5.208
	$\bar{V}_{SA3,Min} \leq \bar{V}_{SA3} \leq \bar{V}_{SA3,Max}$	5.209
	$\bar{V}_{CD3} \geq 0$	5.210
	$\bar{V}_{HD3} \geq 0$	5.211
	<b>Zone Temperature Errors</b>	
	$E_{TZA1} \geq 0$	5.212
	$E_{TZA1} \geq T_{ZA1,HeatingSP} - T_{ZA1}$	5.213
	$E_{TZA1} \geq T_{ZA1} - T_{ZA1,CoolingSP}$	5.214
	$E_{TZA2} \geq 0$	5.215
	$E_{TZA2} \geq T_{ZA2,HeatingSP} - T_{ZA2}$	5.216
	$E_{TZA2} \geq T_{ZA2} - T_{ZA2,CoolingSP}$	5.217
	$E_{TZA3} \geq 0$	5.218
	$E_{TZA3} \geq T_{ZA3,HeatingSP} - T_{ZA3}$	5.219
	$E_{TZA3} \geq T_{ZA3} - T_{ZA3,CoolingSP}$	5.220
	$E_{TZA,Sum} = E_{TZA1} + E_{TZA2} + E_{TZA3}$	5.221
	<b>Energy Usage</b>	
	$\bar{E}_{Fan} = \bar{E}_{Fan,Max} \left( \frac{\bar{V}_{SA}}{\bar{V}_{SA,Max}} \right)^3$	5.222
	$\bar{E}_{Total} = \bar{E}_{L\&P} + \bar{E}_{Fan}$	
	$\bar{Q}_{CC,S} = \rho c_p \bar{V}_{SA}(T_{SF} - T_{CD})$	5.223
	$\bar{Q}_{CC,L} = \rho h_{fg} \bar{V}_{SA}(W_{MA} - W_{CD})$	5.224
	$\bar{Q}_{CC,Total} = \bar{Q}_{CC,S} + \bar{Q}_{CC,L}$	5.225
	$\bar{Q}_{HC} = \rho c_p \bar{V}_{SA}(T_{HD} - T_{SF})$	5.226
<b>Costs</b>		
$\overline{Cost}_{Electric} = \bar{E}_{Fan} \alpha_{WattsToKw} Price_{Electric}$	5.227	
$\overline{Cost}_{ChW} = \bar{Q}_{CC,Total} \alpha_{BtuToMMBtu} Price_{ChW}$	5.228	
$\overline{Cost}_{HW} = \bar{Q}_{RH,Total} \alpha_{BtuToMMBtu} Price_{HW}$	5.229	
$\overline{Cost}_{Total} = \overline{Cost}_{Electric} + \overline{Cost}_{ChW} + \overline{Cost}_{HW}$	5.230	

**Figure 62 Continued**

During a yearly simulation of the DDVAV system the cold deck and hot deck temperatures stayed at their set points. Figure 63 shows cold deck and hot deck air flow

rates for the perimeter zone. At low outside temperatures the system only performed heating, at high temperatures the system only performed cooling, and at intermediate temperatures the system performed heating and cooling to maintain the minimum zone flow rate of  $0.3 \frac{CFM}{ft^2}$ .



**Figure 63 – Perimeter Zone DDVAV Air Flow Rates**

### *Modeling Thermal Mass*

A single node wall thermal mass model can be implemented in the DDVAV model by adding equations 5.232 and 5.233 and replacing equation 5.189 of the model in Figure with equation 5.234.

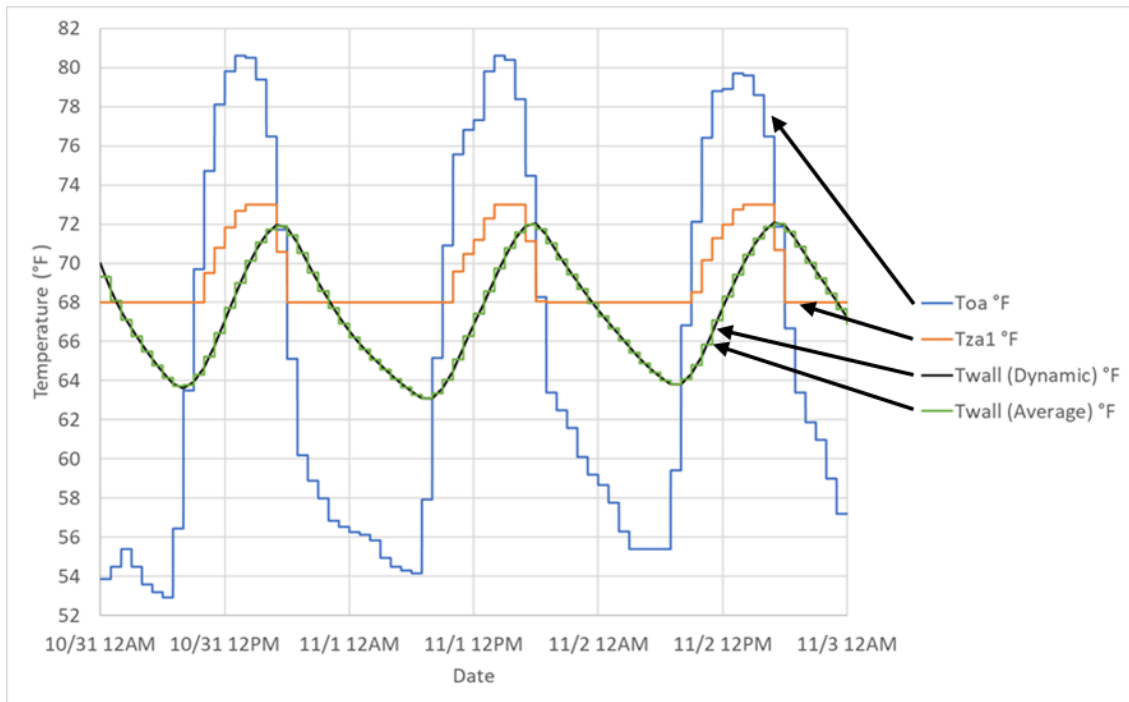
$$\tau_{Wall} \dot{T}_{Wall}(t) = -4T_{Wall}(t) + 2T_{OA} + 2T_{ZA1} \quad 5.232$$

$$T_{Wall,Avg} = \frac{1}{\delta} \int_0^{\delta} T_{Wall}(t) dt \quad 5.233$$

$$\rho c_p \bar{V}_{SA1} (T_{SA1} - T_{ZA1}) + \bar{U} \bar{A}_{Z1,Window/Roof} (T_{OA} - T_{ZA1}) + 2\bar{U} \bar{A}_{Z1,Wall} (T_{Wall,Avg} - T_{ZA1}) + \bar{Q}_{S1} = 0 \quad 5.234$$

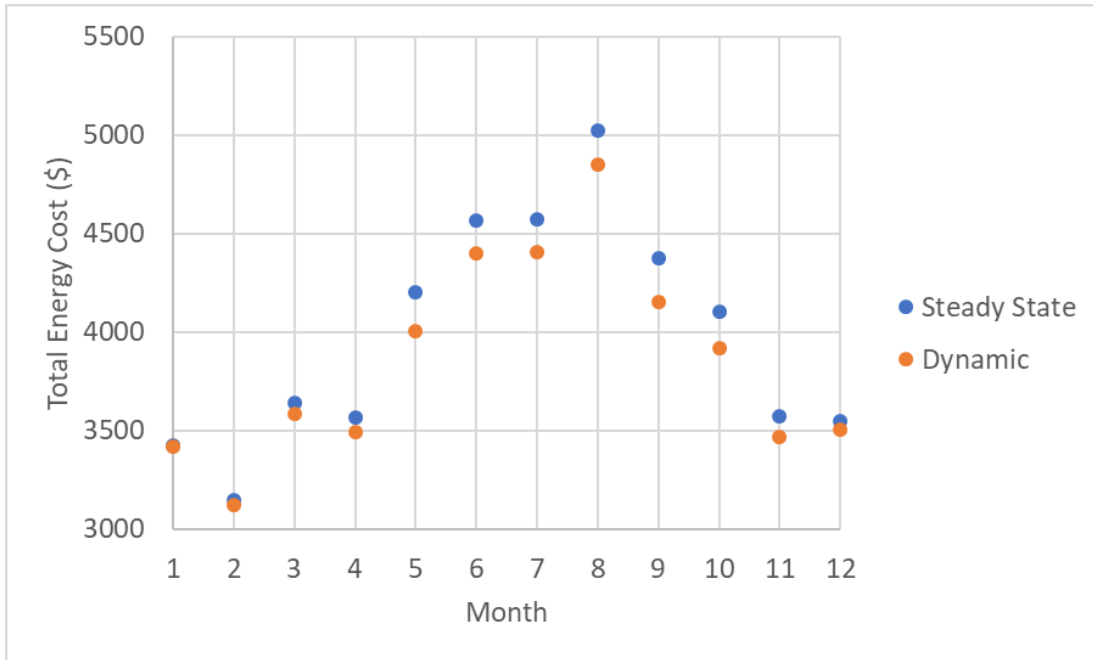


Figure 64 shows three days of simulation results for the dynamic DDVAV model. During this period outside air temperatures peak at approximately 1:00 PM and the wall temperature peaks at approximately 5:00 PM, which delays the effect of wall heat conduction on the zone control response. This delay can be seen in the zone air temperature plot. Dynamic wall temperatures are also plotted along with the average wall temperature on each time step.



**Figure 64 – Dynamic Wall Temperature for the DDVAV Model**

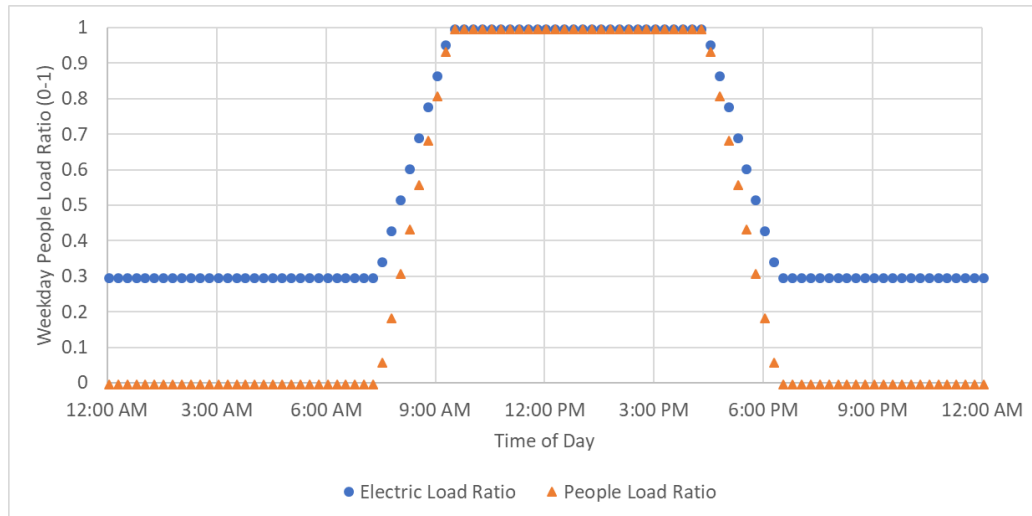
Figure 65 shows monthly total energy costs for the DDVAV model with and without single node wall dynamics using a time step of one hour. Incorporating wall dynamics reduced the simulated total yearly energy costs by 3%, with hotter months having the largest difference. A wall model with six nodes was also run, and its yearly energy cost difference with the one node model was just 0.08%.



**Figure 65 – Monthly DDVAV Energy Costs with and without Dynamics**

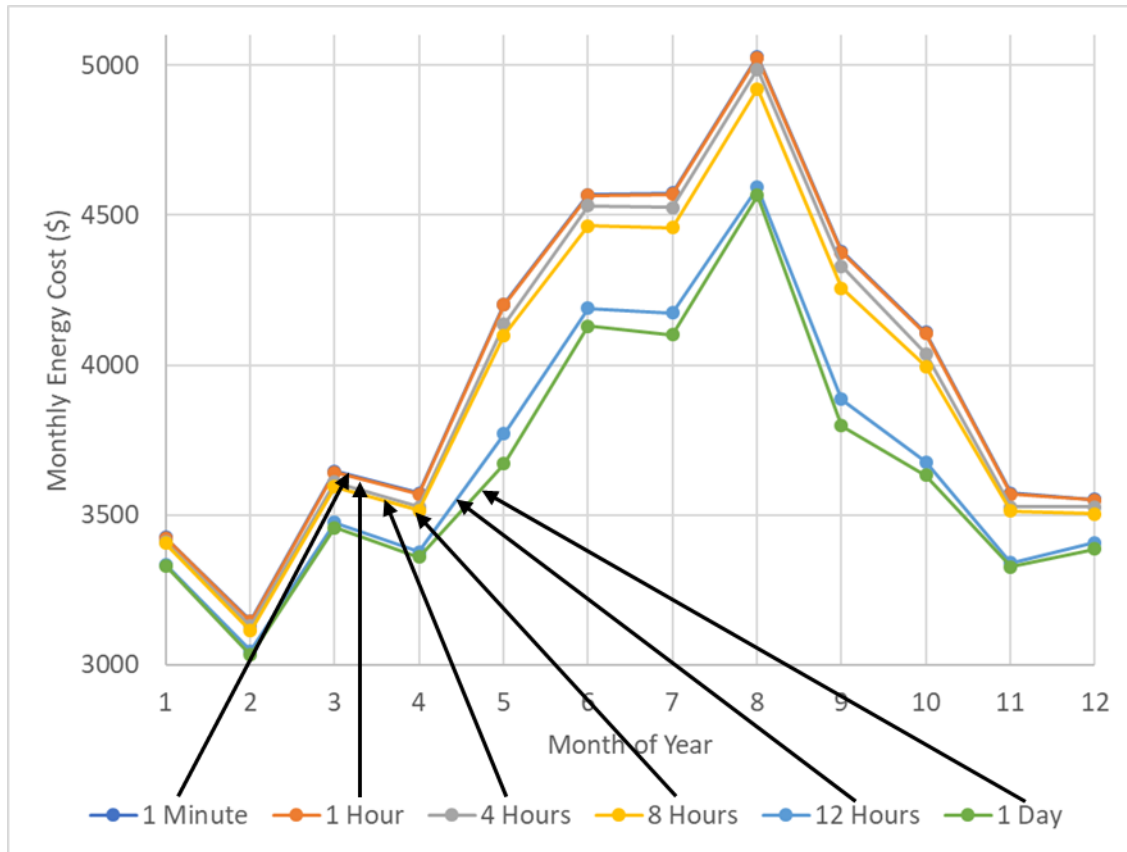
*Effect of Time Step Period Length*

A building energy use model’s time step period length controls the resolution of the part load conditions that it considers. For instance, the 15 minute time schedule of Figure 66 considers 10 separate electric and people load levels while the 1 hour schedule from Figure 40 considers 5. This improves the simulation’s accuracy while increasing the simulation time in proportion to the number of time steps used.



**Figure 66 – Weekday Electric and People Load Schedules for a Time Step of 15 Minutes**

Figure 67 shows monthly total energy costs for the non-dynamic DDVAV model with time steps of 1 minute, 1 hour, 4 hours, 8 hours, 12 hours, and 1 day. The 1 hour time step results was almost identical to the 1 minute results, with 0.08% of a difference in their yearly totals. As the time step grew the difference between 1 minute results grew, with 1% for the 4 hour time step, 2% for the 8 hour time step, 7% for the 12 hour time step, and 8% for the 1 day time step.

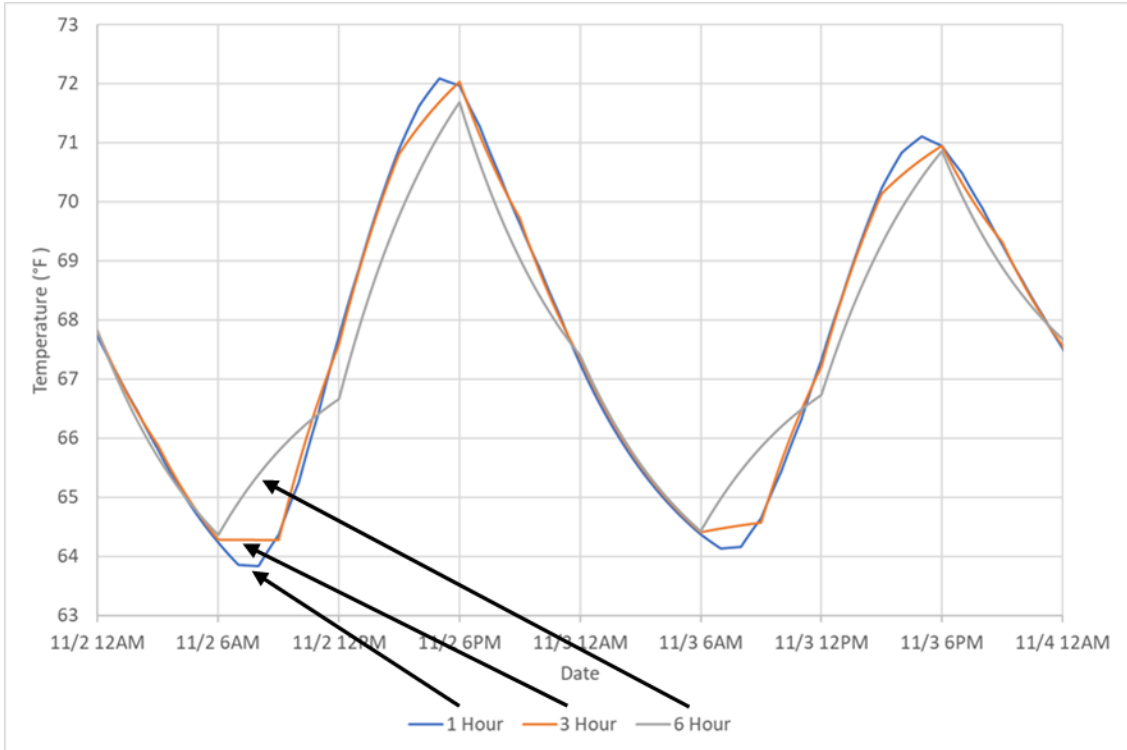


**Figure 67 – Monthly Energy Costs for Different Time Step Sizes for the Steady State DDVAV Model**

On each time step the outside air temperature and perimeter zone air temperature are assumed to be constant while the wall temperature is allowed to vary over time. An energy balance assures that energy transfer from or to the wall equals the energy lost or gained in the zone. This means that dynamic wall temperatures on a time step follow an exponential curve, as shown in Figure 32.

Figure 68 shows the dynamic wall temperature for the DDVAV system with a single wall node for three different time step periods over two days. Shorter time steps result in smoother, more accurate curves than longer time steps. The exponentially decaying nature of heat transfer can especially be seen with a time step of 6 hours. However, despite these differences the total yearly energy cost with a 6 hour time step

was only 1.4% different than using a 1 hour time step, and the total yearly cost with a 1 hour time step was only 0.06% different than using a 1 minute time step.



**Figure 68 – Dynamic DDVAV Wall Temperatures for Different Time Step Periods**

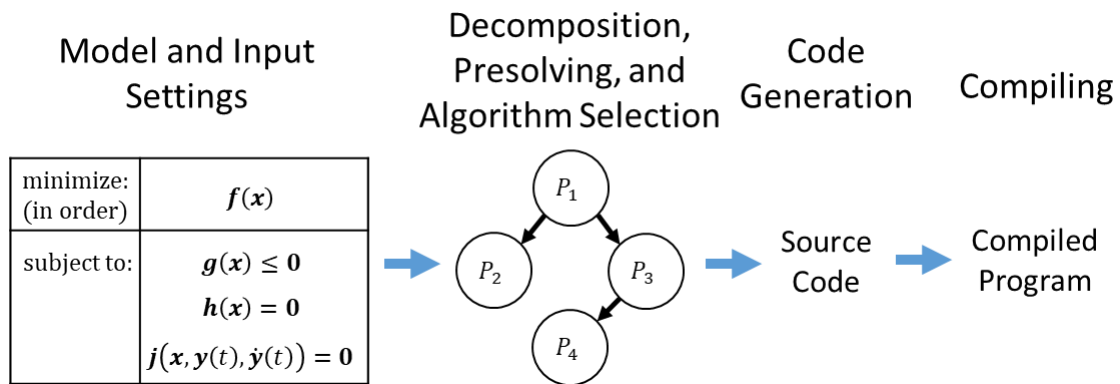
## CHAPTER VI

### AN AUTOMATED SOLUTION ALGORITHM

Computers use numerical algorithms to solve systems of algebraic equations, systems of differential equations, and constrained optimizations. Solving these problems in a robust and fast way involves manipulating the mathematics and choosing optimal algorithms and algorithm settings. Handling this manually complicates HVAC modeling processes by requiring that users be familiar with how many different numerical algorithms work internally. An automated solution algorithm for converting models to executable programs simplifies this process.

#### **Overall Algorithm**

The solution algorithm presented in this chapter performs a simulation by manipulating the mathematics and compiling an executable program that's tailored to solve each individual problem. Differential-algebraic equation solvers such as JModelica [74] can use the same approach. However, this algorithm is tailored for steady state HVAC simulations with embedded dynamic models. Figure 69 shows the solution algorithm's breakdown. It first uses a preprocessing algorithm create a set of solution steps for the model. After preprocessing the solution algorithm generates source code for the overall solution algorithm, before finally compiling a digital linked library, or DLL file for execution. Executing this program calculates the model outputs and writes them to an Excel file.



**Figure 69 – Automated Compilation Process**

### Input Models

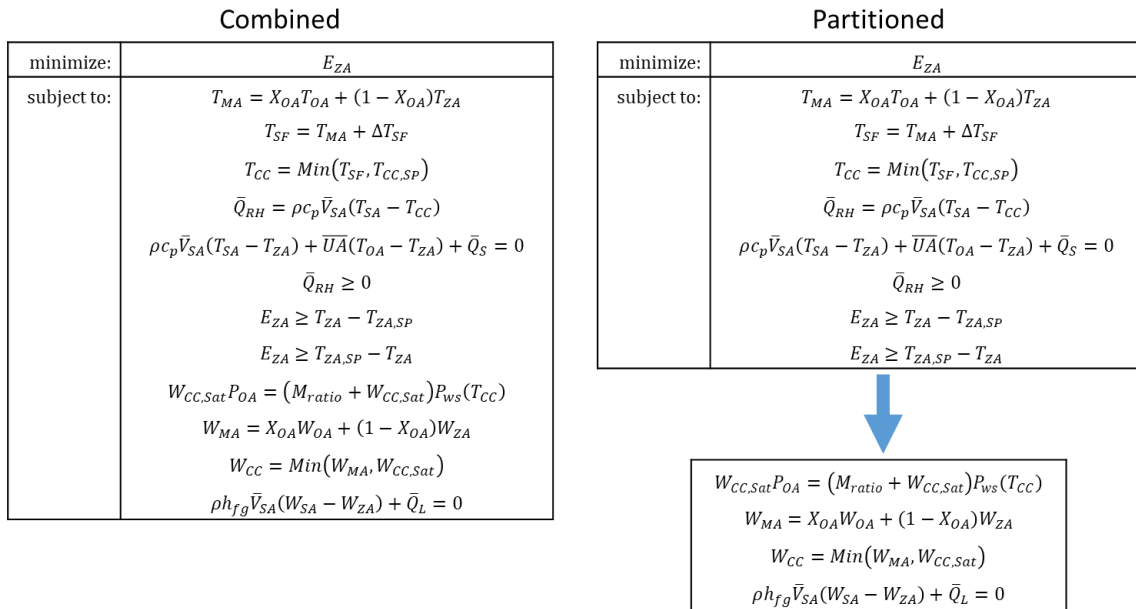
An equation-based building energy model can be formulated as a set of nonlinear lexicographic optimization problems with dynamics as in Figure 10. This model paired with simulation start and stop dates, a time step period, time varying input data such as weather conditions and loads, optional information on how to override default preprocessing and numerical algorithm settings, and desired output variables gives enough information to process a model and perform a simulation. Table 29 lists these components.

**Table 29 – Input Model Components**

Component	Description
Mathematical Model	An optimization problem in the form of Figure 10
Simulation Start Date	The inclusive date/time that a simulation starts
Simulation End Date	The exclusive date/time that a simulation ends
Time Step Period	A time step length that evenly divides the simulation period
Input Data	Varying parameters on each time step, like weather conditions and loads
User Settings	User preferences for how models are processed and how algorithms are run
Output Variables	The desired outputs of a simulation

The preprocessing algorithm described in this chapter cannot always decompose a system when it's beneficial to do so. For instance, humidity calculations can be decoupled from temperature calculations in systems that control temperatures only, but no algorithm that can detect that situation was discovered. Solving for temperatures and flows first and then solving for humidity variables allows smaller subproblems to be solved. This decomposition can increase the speed of calculations by allowing smaller matrices to be used, and calculations can be more robust since there's fewer nonlinear equations being solved simultaneously.

Manually created partitions in mathematical models allow for decompositions that can't be identified automatically. Figure 70 shows the SDCAV system of Figure 15 with humidity calculations. The figure on the left shows the combined model while the figure on the right shows the humidity calculations partitioned out.



**Figure 70 – Partitioning a SDCAV Model into Smaller Subproblems**

### Model Preprocessing

Preprocessing a model simplifies its mathematics, eliminates calculable variables, decomposes it into multiple subproblems, selects algorithms to use on each



subproblem, and modifies subproblems so that they run robustly and quickly for the selected algorithm. These operations make simulations run faster and more robustly for the numerical algorithms used.

Figure 71 shows pseudo-code for the preprocessing algorithm. First, the user-partitioned model is translated into subproblems. Next, a series of preprocessing operations are applied to subproblems to convert a model to its final set of calculation steps. Each preprocessing operation performs a specific task such as splitting independent subproblems or solving linear equations. The preprocessing algorithm terminates when no more operations can be applied.

```
Starting with the User-Entered Model
Start:
For Each Preprocessing Operation
  For Each Subproblem
    If Operation Applies to the Subproblem
    {
      Apply Operation to the Subproblem
      Goto Start
    }
```

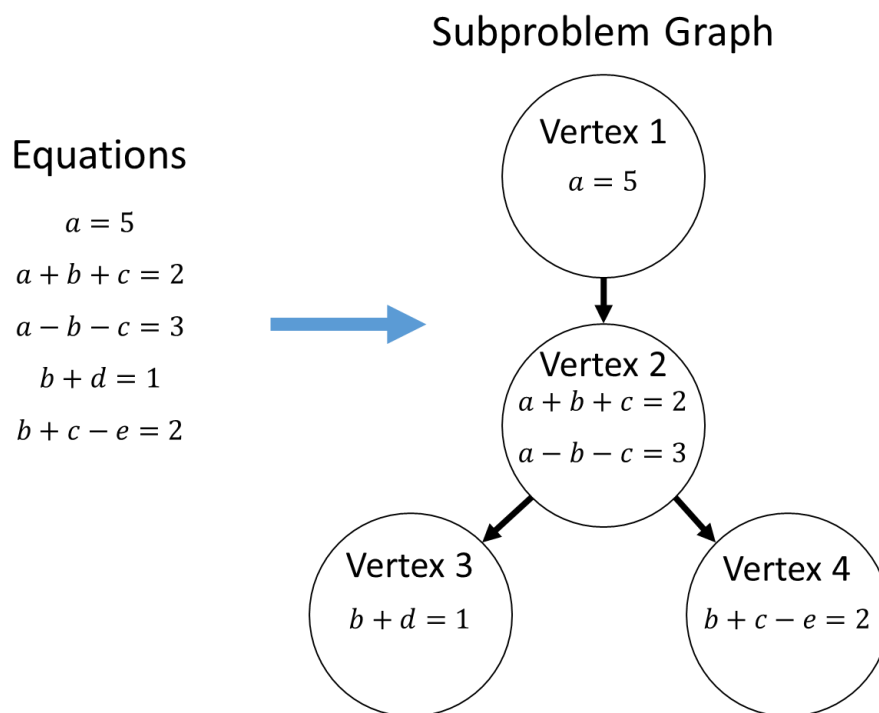
**Figure 71 – Preprocessing Algorithm Pseudocode**

### *Subproblem Graphs*

A *subproblem graph* represents all calculations to perform in a single time step of a steady state building energy simulation, as shown in Figure 72. Each vertex of a subproblem graph represents a subproblem to solve with a numerical algorithm. Edges between vertices define the order that subproblems must be solved in. After a vertex is calculated the variables it fixes can be used as inputs on child vertices. This gives subproblem graphs directed and acyclic structure. The preprocessing algorithm decomposes and simplifies problems by modifying subproblem graphs and the subproblems stored in their vertices. When these modifications change the inputs or

outputs of a vertex a topological sorting algorithm such as a depth-first search can be used to order vertices so that subproblems are calculated in the right order.

The subproblem graph of Figure 72 shows five algebraic equations decomposed into an optimal solution order. Variables in each subproblem are fixed after being calculated on each time step, and these fixed variables serve as inputs to child vertex subproblems. Solving the subproblems of Vertex 1, then Vertex 2, then Vertices 3 and 4 in either order gives the same results as solving all 5 equations simultaneously.



**Figure 72 – A Subproblem Graph for Algebraic Equations**

### *Preprocessing Operations*

Preprocessing operations are atomic operations that modify a subproblem graph based on a detected scenario. For instance, the preprocessing operation of deleting a vertex looks for vertices with subproblems that have no constraints or optimization goals. It will only delete the vertex if it meets that criteria. Operations within the preprocessing algorithm are ordered to look for trivial simplifications and

decompositions first to make the final subproblems as small as possible. Later operations process subproblems so that they're solved efficiently on the particular numerical solver used on them.

Table summarizes the operations performed by the preprocessing algorithm. Each operation is described on a single row. The left-hand column describes the type of subproblem or mathematical function that an operation applies to. When an operation applies to an entire subproblem the equations and constraints appear in brackets. The right-hand column describes the operations performed on a subproblem or function. Operations are applied in the order given in Table in the preprocessing algorithm, and the preprocessing algorithm terminates when an error is detected or no more operations apply.

**Table 30 – Preprocessing Operations**

<b>Applicable Subproblem or Function</b>	<b>Operation Performed</b>
$\left\{ \begin{array}{l} f(x, y) \\ g(x, y) \leq 0 \\ h(x, y) = 0 \\ a(y) \leq x \leq b(y) \end{array} \right\}$ <ul style="list-style-type: none"> <li>• No solved variables are outputs or used as inputs later in the simulation</li> <li>• Can be an empty subproblem</li> </ul>	<p style="text-align: center;"><u>Unused Vertex Deletion</u></p> <ul style="list-style-type: none"> <li>• Delete the unused vertex from the subproblem graph</li> </ul>
$f(c) = 0 \text{ or } f(c) \leq 0$ <ul style="list-style-type: none"> <li>• A trivial equality or inequality with only known constants</li> <li>• For example, <math>e^{\ln(1)} - 1 = 0</math></li> </ul>	<p style="text-align: center;"><u>Trivial Constraint Removal</u></p> <ul style="list-style-type: none"> <li>• Assert the validity of the equality or inequality</li> <li>• Add an error if the error is above some epsilon</li> <li>• Remove the equality or inequality from the subproblem</li> </ul>
$f(y) = 0 \text{ or } f(y) \leq 0$ <ul style="list-style-type: none"> <li>• A trivial equality or inequality with input variables that change on each time step</li> </ul>	<p style="text-align: center;"><u>Trivial Constant Processing</u></p> <ul style="list-style-type: none"> <li>• Move the equality or equality to a new vertex</li> <li>• Assert that functions of input variables hold during a simulation</li> </ul>

Table 30 Continued

Applicable Subproblem or Function	Operation Performed
$ax + b = 0$ <ul style="list-style-type: none"> <li>• A linear equation with a single solved variable</li> </ul>	<p style="text-align: center;"><u>Constant Solving</u></p> <ul style="list-style-type: none"> <li>• Delete the equation from the vertex</li> <li>• Set <math>x</math> to a constant value for the entire subproblem graph</li> <li>• Assert any constant bounds of <math>x</math></li> <li>• Remove variable bounds of <math>x</math> to their own new vertex for asserting during a simulation</li> </ul>
$\{P1, P2, \dots, Pn\}$ <ul style="list-style-type: none"> <li>• A subproblem with multiple independent partitions</li> </ul>	<p style="text-align: center;"><u>Independent Problem Partitioning</u></p> <ul style="list-style-type: none"> <li>• Move each independent subproblem to their own vertex</li> </ul>
$f(\mathbf{y}) \leq x \leq f(\mathbf{y})$ <ul style="list-style-type: none"> <li>• A solved variable with the same minimum and maximum bounds</li> <li>• <math>f(\mathbf{y})</math> is a function of output variables</li> </ul>	<p style="text-align: center;"><u>Removing Forced Bounds</u></p> <ul style="list-style-type: none"> <li>• Create a vertex with the equation <math>x = f(\mathbf{y})</math></li> <li>• <math>x</math> becomes an input variable on the original vertex</li> <li>• Delete the bounds for <math>x</math> on the original vertex</li> <li>• Reformulate bounds for <math>x</math> on other vertices</li> </ul>
$f(\mathbf{y})x + g(\mathbf{y}) = 0$ <ul style="list-style-type: none"> <li>• An algebraic linear equation with a single solved variable</li> <li>• Coefficients can be functions of <math>y</math></li> <li>• The vertex contains more than one function</li> </ul>	<p style="text-align: center;"><u>Univariate Linear Equality Solving</u></p> <ul style="list-style-type: none"> <li>• Create a new vertex with the equation <math display="block">x = -\frac{g(\mathbf{y})}{f(\mathbf{y})}</math> </li> <li>• Delete the equation from the original vertex</li> <li>• Reformulate any bounds on <math>x</math></li> <li>• Delete the bounds for <math>x</math> on the original vertex</li> <li>• Reformulate bounds for <math>x</math> on other vertices</li> </ul>

Table 30 Continued

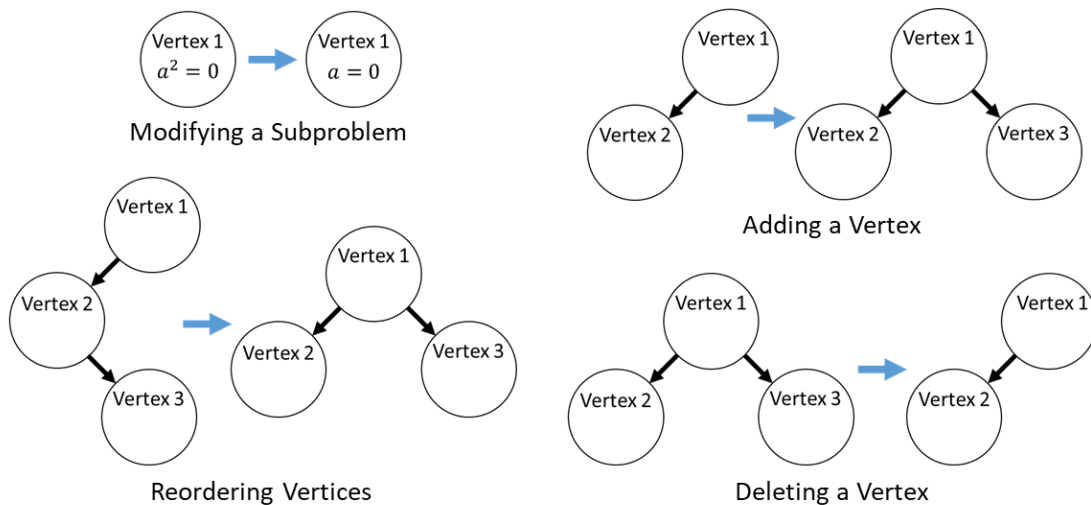
Applicable Subproblem or Function	Operation Performed
$f(\mathbf{y})x_1 + g(\mathbf{y})x_2 + h(\mathbf{y}) = 0$ <ul style="list-style-type: none"> <li>• A bivariate linear equation</li> <li>• Equation is linear in terms of <math>x_1</math> and <math>x_2</math></li> <li>• <math>f(\mathbf{y})</math> is not zero</li> <li>• The <math>x_2</math> coefficient must be constant (nonzero), positive definite, or negative definite if bounds exist (use the variable with constant bounds)</li> </ul>	<p style="text-align: center;"><u>Bivariate Linear Equality Solving</u></p> <ul style="list-style-type: none"> <li>• Create a new vertex with the equation <math display="block">x_1 = -\frac{g(\mathbf{y})}{f(\mathbf{y})}x_2 - \frac{h(\mathbf{y})}{f(\mathbf{y})}</math> </li> <li>• Replace <math>x_1</math> on all vertex equations</li> <li>• Remove <math>x_1</math> from the list of solved variables</li> <li>• Update bounds of <math>x_2</math> taking the bounds of <math>x_1</math> into account</li> </ul>
$f(\mathbf{y})x + g(\mathbf{y}) \leq 0$ <ul style="list-style-type: none"> <li>• An inequality with one solved variable</li> <li>• <math>g(\mathbf{y})</math> is constant or a function of input variables</li> <li>• <math>f(\mathbf{y}) \neq 0</math></li> <li>• <math>f(\mathbf{y})</math> is positive semidefinite or negative semidefinite for all possible input values</li> </ul>	<p style="text-align: center;"><u>Linear Inequality Reformulating</u></p> <ul style="list-style-type: none"> <li>• Reformulate as a bound on <math>x</math> on the vertex</li> </ul>
$f(\mathbf{x}, \mathbf{y}) \leq 0$ <ul style="list-style-type: none"> <li>• A general inequality</li> </ul>	<p style="text-align: center;"><u>General Inequality Reformulating</u></p> <ul style="list-style-type: none"> <li>• Reformulate the inequality with a slack variable and a bound on the vertex <math display="block">f(\mathbf{x}, \mathbf{y}) - z = 0</math> <math display="block">z \leq 0</math> </li> </ul>
$\frac{d\mathbf{x}_{dyn}}{dt} = \mathbf{A}\mathbf{x}_{dyn} + \mathbf{B}\mathbf{x}_{avg} + \mathbf{f}(\mathbf{y})$ <ul style="list-style-type: none"> <li>• A vertex containing a linear system of differential equations</li> <li>• <math>\mathbf{x}_{dyn}</math> are dynamic variables</li> <li>• <math>\mathbf{x}_{avg}</math> are averaged variables</li> <li>• <math>\mathbf{f}(\mathbf{y})</math> is a linear or nonlinear function of output variables</li> </ul>	<p style="text-align: center;"><u>Linear ODE Solving</u></p> <ul style="list-style-type: none"> <li>• Delete the differential equations</li> <li>• Add equations for the averages of the dynamic variables over a time step</li> </ul>
$x = f(\mathbf{x}_{other}, \mathbf{y})$ <ul style="list-style-type: none"> <li>• An algebraic equation</li> <li>• One solved variable is solvable in terms of other solved variables</li> <li>• <math>x</math> does not have any bounds</li> <li>• <math>x</math> is not in any inequalities</li> </ul>	<p style="text-align: center;"><u>General Algebraic Decomposition</u></p> <ul style="list-style-type: none"> <li>• Place the equation to a new vertex</li> <li>• Replace <math>x</math> with the function in the current subproblem</li> <li>• Remove <math>x</math> variable from the vertex</li> </ul>

Table 30 Continued

Applicable Subproblem or Function	Operation Performed
$\begin{cases} f(x, y) \\ g(x, y) \leq 0 \\ h(x, y) = 0 \\ a(y) \leq x \leq b(y) \end{cases}$ <ul style="list-style-type: none"> <li>An unscaled optimization or algebraic problem that must use a numerical solver</li> </ul>	<p><u>Optimization and Algebraic Problem Scaling</u></p> <ul style="list-style-type: none"> <li>Scale the solved variables so they lie between -1 and 1</li> <li>Add new variables for the scaled variables</li> <li>Add vertices to solve for the unscaled variables</li> </ul>
$\begin{cases} f(x, y) \\ g(x, y) \leq 0 \\ h(x, y) = 0 \\ a(y) \leq x \leq b(y) \end{cases}$ <ul style="list-style-type: none"> <li>A constrained optimization with multiple, lexicographic objectives</li> </ul>	<p><u>Lexicographic Decomposition</u></p> <ul style="list-style-type: none"> <li>Perform lexicographic decomposition to separate the subproblem into one stage for each objective.</li> <li>This process is shown in Figure 11.</li> </ul>
$\begin{cases} f(x, y) \\ g(x, y) \leq 0 \\ h(x, y) = 0 \\ a(y) \leq x \leq b(y) \end{cases}$ <ul style="list-style-type: none"> <li>A constrained optimization with a single objective</li> </ul>	<p><u>Optimization Function Scaling</u></p> <ul style="list-style-type: none"> <li>Scale the optimized functions so they typically lie between -1 and 1</li> </ul>
<ul style="list-style-type: none"> <li>Any expression with a <i>Min</i>, <i>Max</i>, or <i>Abs</i> function in it</li> <li>On a vertex of an unscaled optimization or algebraic problem that must use a numerical solver</li> </ul>	<p><u>Equation Smoother</u></p> <ul style="list-style-type: none"> <li>Smooth discontinuous expressions</li> <li>Replace Max, Min, and Absolute Value functions with smooth versions described in Appendix B</li> </ul>
$h(x) = 0$ <ul style="list-style-type: none"> <li>A system of algebraic equations with no input variables</li> </ul>	<p><u>Algebraic Presolving</u></p> <ul style="list-style-type: none"> <li>Solve the system of equations</li> <li>Set the solved variables as constants throughout the subproblem graph</li> <li>Delete the vertex</li> </ul>
$\begin{cases} f(x) \\ g(x) \leq 0 \\ h(x) = 0 \\ a \leq x \leq b \end{cases}$ <ul style="list-style-type: none"> <li>A constrained optimization with a single objective and no input variables</li> </ul>	<p><u>Optimization Presolving</u></p> <ul style="list-style-type: none"> <li>Solve the constrained optimization</li> <li>Set the solved variables as constants throughout the subproblem graph</li> <li>Delete the vertex</li> </ul>

### Basic Subproblem Graph Operations

Preprocessing operations make use of a few basic operations that change a subproblem graph's structure. Figure 73 shows these basic graph operations. Modifying a subproblem involves making changes to the equations, constraints, objective functions, or variables in a vertex's subproblem. Vertices can be added to split up subproblems or deleted if a step isn't required. For instance, a vertex with two equations with independent variables can split into two vertices with one equation each. If one of those equations isn't used that vertex can be deleted. Reordering vertices ensures that edges describe a valid solution order as vertex modifications are made.



**Figure 73 – Basic Subproblem Graph Operations**

### Code Generation, Compilation, and Execution

After preprocessing, a model the automated solution algorithm generates customized source code for the model and compiles it for execution. This allows low level optimizations to be written into the source code, and third-party numerical libraries or programs can be linked as well. Next this source code is compiled into an executable program. Finally, the program is executed, running the simulation and creating outputs for the model.

## CHAPTER VII

### SOLVER PROGRAM IMPLEMENTATION

A prototype solver named Beryl was developed to provide an upper bound on the efficiency and robustness building energy models based on constrained optimization. with lexicographic goals and integrated dynamics. Beryl allows models to be entered and simulated from a text format and returns outputs in an Excel format or displayed as a chart window. This chapter discusses the internal structure of the program, while Appendix E describes its input file format and how to run models.

Beryl was primarily written in C# in Visual Studio 2017. Linear algebra calculations were performed using Intel's MKL library [75] and compiled using Intel Parallel Studio XE. The C# language, Visual Studio development environment, and plugin extensions such as ReSharper were useful in maintaining the code base, which consists of over 2,000 pages of code. However, most algebraic and constrained optimization algorithms are centered around intensive linear algebra calculations.

During testing Beryl successfully simulated the models given in Chapter V, with models taking between 0.5 and 5.0 seconds to perform a yearly calculation with hourly time steps. Using sensitivity analysis to predict the solution of one time step from the previous time step's result allowed the constrained optimization algorithm to converge in as little as 7 iterations per time step. All models successfully executed on the solver's default settings, which solves the Karush-Kuhn-Tucker conditions of models to a relative error of  $10^{-9}$ .

#### **Program Structure**

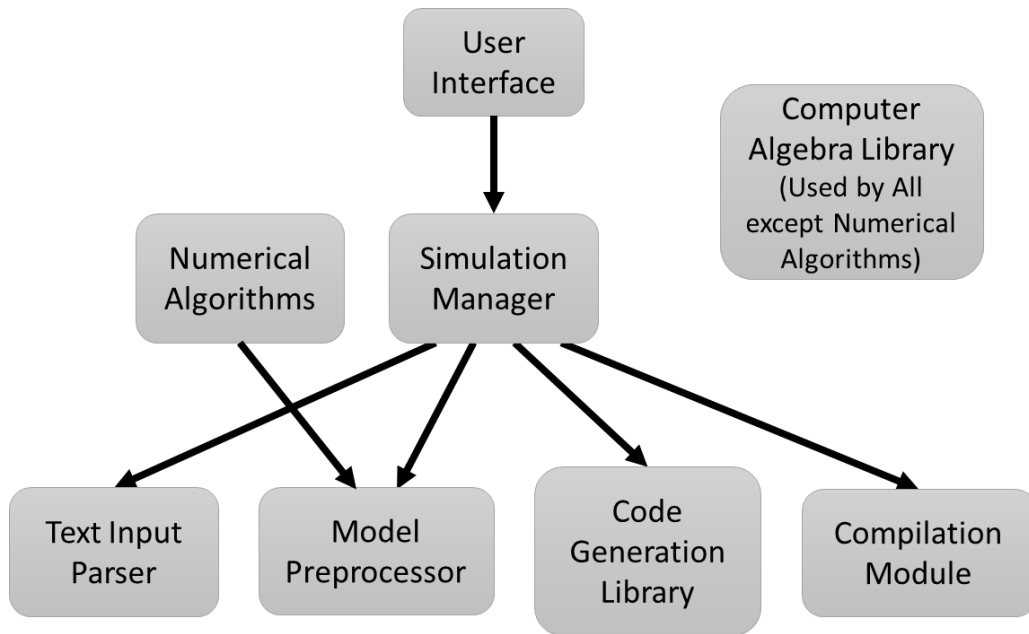
Figure 69 shows the process used by Beryl in performing a simulation. First, models are decomposed and simplified using the automated solution algorithm from Chapter VI. Next, C# source code is generated for the model and linked to external numerical solvers. Finally, the source code is executed and output Excel files are written and charts are displayed.



Beryl’s source code consists of eight major modules, listed in Table 31. Users interact with the *user interface*, where models can be created and run from text files. The *simulation manager* coordinates the overall simulation process, from parsing an input text file to compiling a program for a model. The simulation manager itself uses the *text input parser*, *model preprocessor*, *the code generation library*, and the *compilation module* for this process. All of these modules use a custom *computer algebra library* for representing mathematical expressions. Also, custom *numerical algorithms* are used during model preprocessing and in the executable programs generated to simulate a model. Figure 74 shows the relationship between these different modules.

**Table 31 – Major Modules of Beryl**

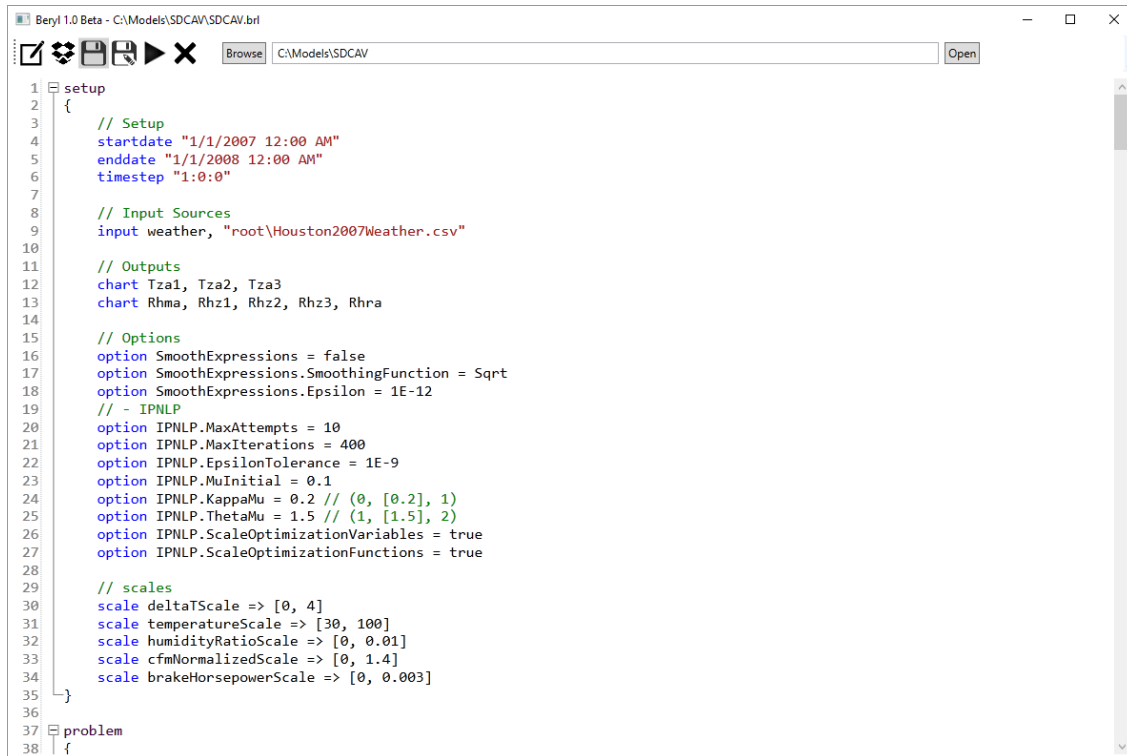
<b>Module</b>	<b>Function</b>
User Interface	Allows users to create and run simulations
Simulation Manager	Coordinates the overall simulation process
Text Input Parser	Parses text input files to input models
Model Preprocessor	Simplifies and reformulates a model for numerical solvers
Code Generation Library	Generating customized source code for a model
Compilation Module	Compiles source code into an executable .dll
Computer Algebra Library	Parsing and modifying mathematical expressions
Numerical Algorithms	Solves model subproblems during a simulation



**Figure 74 – Usage Relationships between Beryl’s Modules**

*User Interface*

Beryl’s user interface allows users to create, modify and run building energy models. Figure 75 shows a screen shot of the user interface. This window contains a code editor and controls to open and close files, select working directories, and run simulations. Details of constructing models and running simulations are discussed in Appendix E.



```
1 setup
2 {
3     // Setup
4     startdate "1/1/2007 12:00 AM"
5     enddate "1/1/2008 12:00 AM"
6     timestep "1:0:0"
7
8     // Input Sources
9     input weather, "root\Houston2007Weather.csv"
10
11    // Outputs
12    chart Tza1, Tza2, Tza3
13    chart Rhma, Rhz1, Rhz2, Rhz3, Rhra
14
15    // Options
16    option SmoothExpressions = false
17    option SmoothExpressions.SmoothingFunction = Sqrt
18    option SmoothExpressions.Epsilon = 1E-12
19    // - IPNLP
20    option IPNLP.MaxAttempts = 10
21    option IPNLP.MaxIterations = 400
22    option IPNLP.EpsilonTolerance = 1E-9
23    option IPNLP.MuInitial = 0.1
24    option IPNLP.KappaMu = 0.2 // (0, [0.2], 1)
25    option IPNLP.ThetaMu = 1.5 // (1, [1.5], 2)
26    option IPNLP.ScaleOptimizationVariables = true
27    option IPNLP.ScaleOptimizationFunctions = true
28
29    // scales
30    scale deltaTScale => [0, 4]
31    scale temperatureScale => [30, 100]
32    scale humidityRatioScale => [0, 0.01]
33    scale cfmNormalizedScale => [0, 1.4]
34    scale brakeHorsepowerScale => [0, 0.003]
35 }
36
37 problem
38 {
```

**Figure 75 – A Screenshot of Beryl’s User Interface**

### *Simulation Manager*

The simulation manager controls the process of parsing, simplifying, and running a simulation. It interfaces the various code modules that perform each of these tasks and returns an error message if calculations fail. Having this module independent allows simulations to be executed from other interfaces besides the current user interface.

### *Text Input Parser*

The text input parser validates text inputs from the UI and parses them into an internal model structure. Errors found in a model are displayed to the user through the user interface, as seen in Figure 76. If a model is valid it can be simulated, and to simulate a model the internal model structure created by the text input parser is sent to the model preprocessor, which implements the decomposition and simplification algorithm from Chapter VI.

```

// Calculated Building Parameters
// - floor, window, and wall areas
double AFloorTotal, ft2
double AWallTotal, ft2
double AWindow, ft2
double AWall, ft2
constraint AFloorTot = AFloor1 + AFloor2 + AFloor3
const Variable not defined al = 2*(WBuilding + LBuilding)*HBuilding
constraint AWindow = 0.01*AWallTotal*PctWindow
constraint AWall = AWallTotal - AWindow

```

**Figure 76 – An Error Message for an Undefined Variable in Beryl**

### *Model Preprocessor*

The model preprocessor module implements the preprocessing algorithm from Chapter VI. This simplifies a model, eliminates unneeded variables, splits the model into a sequence of subproblems, and selects the numerical solver to use on each subproblem. A sequence of steps from this process serves as the algorithm used to simulate one time step of a specific model. After preprocessing this sequence is used to generate an executable program in C# to simulate a model.

The automated solution process assigns a numerical solver for each subproblem in a model after they can no longer be decomposed into simpler calculations. Lexicographic optimizations are decomposed earlier in preprocessing so at the numerical solver selection stage only problems with one or zero objective functions remain. The executable .dll that Beryl generates for a model links to external numerical solvers and generates the required source code for its calculations.

### *Code Generation Library*

A code generation library was created for generating source code that's customized for individual problems in a way that's readable and avoids naming conflicts. The code generation library allows abstract syntax trees [76] that represent the calculations performed in a program to be built up. It does this using C# statements that are structured in the same way as the code they generate for easy debugging. For

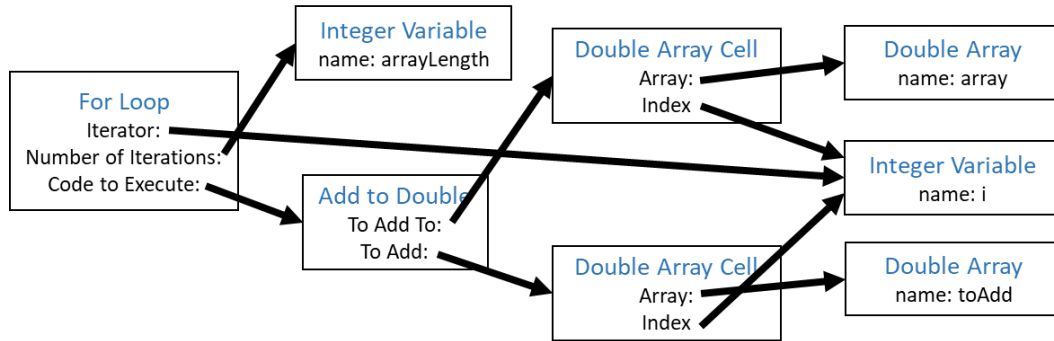
example, the C# code of Figure 77 can be used to generate code that multiplies two matrices stored as two dimensional arrays. The code generation library generates C# source code, but it can be extended other programming languages like C++ and Fortran.

```
For (order, i, new[]
{
    For (order, j, new[]
    {
        x.Declare(0.0),
        For(order, k,
            x.AddTo(array1[i, k]*array2[k, j])),
        result[i, j].SetTo(x)
    })
})
```

**Figure 77 – Code to Generate an Abstract Syntax Tree to Multiply Two 2D Matrices**

The abstract syntax trees used by the code generation library represents C# code elements such as classes, variables, and loops in a tree data structure. For example, the top of Figure 78 shows generation library code for a for loop that adds one array to another element by element. This generates the abstract syntax tree shown in the middle, with nodes of the graph representing parts of the C# language structure. This keeps track of which variables are used where so that any naming conflicts can be resolved by renaming variables. The code at the bottom of Figure 78 shows what the abstract syntax tree generates when no naming conflicts exist.

For(arrayLength, i, array[i].AddTo(toAdd[i]))  
**Source Code to Generate a For Loop**



**Abstract Syntax Tree to Represent the For Loop**

```

for (int i = 0; i < arrayLength; i++)
    array[i] += toAdd[i];
  
```

**Generated Code from the Abstract Syntax Tree**

**Figure 78 – Code Generation Library of a For Loop, It’s Abstract Syntax Tree, and the Generated Code Output**

### *Numerical Algorithms*

Beryl uses custom written numerical solvers to solve four fundamental types of subproblems: solvable variables, systems of algebraic equations, unconstrained optimizations, and constrained optimizations. Table 32 lists information on these problems and the numerical algorithms used to solve them. If a solver fails on a time step or an infeasible constrained optimization is detected variables for the entire time step are set to not a number (NaN). If a dynamic simulation is being performed this ends the simulation since no initial conditions can be calculated for the next time step.

**Table 32 – Numerical Algorithms Implemented in the Beryl**

Subproblem Type	Applicable Subproblems	Numerical Algorithm						
Solvable Variable	$x = f(\mathbf{y})$ <ul style="list-style-type: none"> <li>• A single equation.</li> <li>• <math>x</math> is solved.</li> <li>• <math>\mathbf{y}</math> variables are previously calculated variables or inputs.</li> </ul>	<ul style="list-style-type: none"> <li>• A single line of code that sets <math>x</math> to <math>f(\mathbf{y})</math></li> <li>• No iterations are required.</li> </ul>						
System of Equations	$f(\mathbf{x}, \mathbf{y}) = \mathbf{0}$ <ul style="list-style-type: none"> <li>• One or more linear or nonlinear equations.</li> <li>• <math>\mathbf{x}</math> variables are solved.</li> <li>• <math>\mathbf{y}</math> variables are previously calculated variables or inputs.</li> </ul>	<ul style="list-style-type: none"> <li>• Use Newton’s Method with the Levenberg-Marquardt algorithm</li> <li>• Named ALM (Algebraic Levenberg-Marquardt)</li> </ul>						
Unconstrained Optimization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 2px;">minimize:</td> <td style="padding: 2px;"><math>f(\mathbf{x}, \mathbf{y})</math></td> </tr> </table> <ul style="list-style-type: none"> <li>• Minimizes a single function.</li> <li>• No constraints.</li> <li>• <math>\mathbf{x}</math> variables are solved.</li> <li>• <math>\mathbf{y}</math> variables are previously calculated variables or inputs.</li> </ul>	minimize:	$f(\mathbf{x}, \mathbf{y})$	<ul style="list-style-type: none"> <li>• Use a quasi-Newton’s method based on the BFGS algorithm</li> </ul>				
minimize:	$f(\mathbf{x}, \mathbf{y})$							
Constrained Optimization	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 2px;">minimize:</td> <td style="padding: 2px;"><math>f(\mathbf{x}, \mathbf{y})</math></td> </tr> <tr> <td style="padding: 2px;">subject to:</td> <td style="padding: 2px;"><math>\mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0}</math></td> </tr> <tr> <td></td> <td style="padding: 2px;"><math>\mathbf{a}(\mathbf{y}) \leq \mathbf{x} \leq \mathbf{b}(\mathbf{y})</math></td> </tr> </table> <ul style="list-style-type: none"> <li>• Minimizes a single function subject to constraints.</li> <li>• <math>\mathbf{x}</math> variables are solved.</li> <li>• <math>\mathbf{y}</math> variables are previously calculated variables or inputs.</li> <li>• Minimum and maximum bounds. <math>\mathbf{a}(\mathbf{y})</math> and <math>\mathbf{b}(\mathbf{y})</math> are can be a function of the problem’s inputs.</li> </ul>	minimize:	$f(\mathbf{x}, \mathbf{y})$	subject to:	$\mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0}$		$\mathbf{a}(\mathbf{y}) \leq \mathbf{x} \leq \mathbf{b}(\mathbf{y})$	<ul style="list-style-type: none"> <li>• Use IPNLP, an interior point algorithm based on IPOPT</li> </ul>
minimize:	$f(\mathbf{x}, \mathbf{y})$							
subject to:	$\mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0}$							
	$\mathbf{a}(\mathbf{y}) \leq \mathbf{x} \leq \mathbf{b}(\mathbf{y})$							

## Nonlinear Algebraic Equation Solver

Systems of algebraic equations in Beryl are solved using the Levenberg-Marquardt algorithm [77]. This finds a search direction vector  $\boldsymbol{\delta}$  using 7.1. This equation uses the current function errors  $\mathbf{f}(\mathbf{x}, \mathbf{y})$  and Jacobian matrix  $\mathbf{J}(\mathbf{x}, \mathbf{y})$  from 7.2 at the current estimated output  $\mathbf{x}$  and input parameter vector  $\mathbf{y}$ , as well as a damping parameter  $\lambda$ . The damping parameter is adjusted during a calculation to switch between a Gauss-Newton approach and a gradient descent approach. Cholesky factorizations are used to factor the matrix on the left-hand side of 7.1 when solving for  $\boldsymbol{\delta}$ .

$$(\mathbf{J}^T \mathbf{J} + \lambda \text{diag}(\mathbf{J}^T \mathbf{J})) \boldsymbol{\delta} = -\mathbf{J}^T \mathbf{f} \quad 7.1$$

$$\mathbf{J}(\mathbf{x}, \mathbf{y}) = \left[ \begin{array}{ccc} \frac{df(\mathbf{x}, \mathbf{y})}{dx_1} & \dots & \frac{df(\mathbf{x}, \mathbf{y})}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{df_n(\mathbf{x}, \mathbf{y})}{dx_1} & \dots & \frac{df_n(\mathbf{x}, \mathbf{y})}{dx_n} \end{array} \right] = \begin{bmatrix} \frac{df_1(\mathbf{x}, \mathbf{y})}{dx_1} & \dots & \frac{df_1(\mathbf{x}, \mathbf{y})}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{df_n(\mathbf{x}, \mathbf{y})}{dx_1} & \dots & \frac{df_n(\mathbf{x}, \mathbf{y})}{dx_n} \end{bmatrix} \quad 7.2$$

## Nonlinear Unconstrained Optimization Solver

Unconstrained optimizations minimize the function  $f(\mathbf{x}, \mathbf{y})$ , where the  $\mathbf{x}$  vector contains variables to optimize and the  $\mathbf{y}$  vector contains input parameters that can change from time step to time step. Newton's method solves unconstrained optimizations by finding the solution to the system of equations in equation 7.3 [77]. It uses the first and second derivatives of  $f(\mathbf{x}, \mathbf{y})$  to calculate a search direction  $\boldsymbol{\delta}$  for a time step. The second derivative of  $f(\mathbf{x}, \mathbf{y})$  is known as the Hessian matrix. The Hessian matrix has the symbol  $\mathbf{H}(\mathbf{x}, \mathbf{y})$  and its structure is shown in equation 7.4.

$$\mathbf{H}(\mathbf{x}, \mathbf{y}) \boldsymbol{\delta} = -\nabla f(\mathbf{x}, \mathbf{y}) \quad 7.3$$



$$\mathbf{H}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_2^2} & \dots & \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(\mathbf{x}, \mathbf{y})}{\partial x_n^2} \end{bmatrix} \quad 7.4$$

Beryl solves unconstrained optimizations using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [77]. This algorithm iteratively updates an estimate of the inverse of the Hessian matrix  $\mathbf{B}^{-1}$  to avoid the need for matrix factorizations. The algorithm starts with  $\mathbf{B}^{-1}$  equal to the identity matrix. On each iteration  $k$  a line search algorithm is used to find a step size  $\mathbf{s}_k$ . This step size and the change in gradient  $\mathbf{y}_k = \nabla \mathbf{f}(\mathbf{y}_{k+1}) - \nabla \mathbf{f}(\mathbf{y}_k)$  are used to update the estimate of the inverse Hessian without any need for temporary matrices. Equation 7.5 shows the inverse Hessian update. This solver is entirely generated with no calls made to external libraries to demonstrate how solvers can be customized to specific problems.

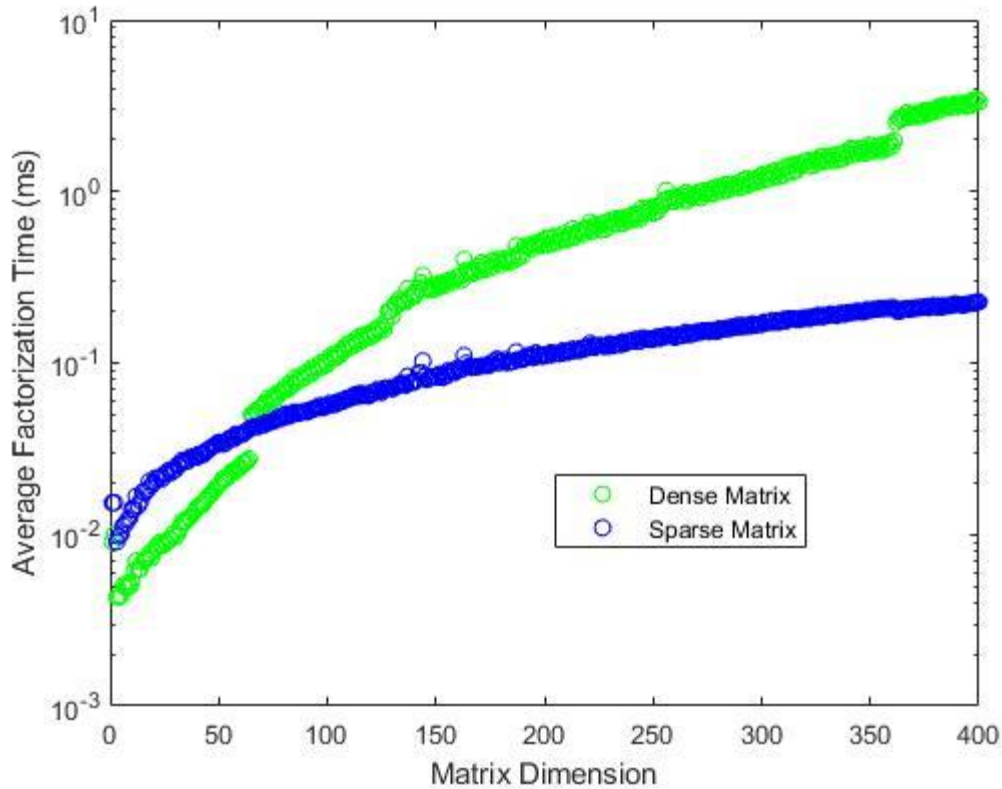
$$\mathbf{B}_{k+1}^{-1} = \mathbf{B}_k^{-1} + \frac{(\mathbf{s}_k^T \mathbf{y}_k + \mathbf{y}_k^T \mathbf{B}_k^{-1} \mathbf{y}_k)(\mathbf{s}_k \mathbf{s}_k^T)}{(\mathbf{s}_k^T \mathbf{y}_k)^2} - \frac{\mathbf{B}_k^{-1} \mathbf{y}_k \mathbf{s}_k^T + \mathbf{s}_k \mathbf{y}_k^T \mathbf{B}_k^{-1}}{\mathbf{s}_k^T \mathbf{y}_k} \quad 7.5$$

### Nonlinear Programming Solver

A constrained optimization solver was written in C# to solve single objective optimization problems with parametric equations and bounds. This solver, named IPNLP (Interior Point Non-Linear Program Solver), uses the same interior point algorithm as IPOPT [41] but is structured for the small, parametric models that occur in simplified building energy simulation models. For instance, it uses dense matrices and allocates arrays once at the beginning of a simulation since the variables that use bounds do not change. It also uses first and second derivative information at the end of a time step and the change in input parameters to initialize the next time step's calculations.

IPNLP uses array calculations based on BLAS (Basic Linear Algebra Subproblems) and dense matrix calculations based on LAPACK (Linear Algebra PACKage). These two libraries provide a standard way for performing array and linear algebra calculations [78]. They were compiled using Intel's Parallel Studio plugin to Microsoft's Visual Studio [75]. Parallel Studio includes an implementation of Intel's MKL Library (Math Kernel Library) which implements and compiles BLAS and LAPACK calls optimized for Intel processors.

Dense matrices can be factored more quickly than sparse matrices for low matrix orders. Figure 79 shows the results of a MATLAB calculation that demonstrates the relative speed of the two approaches. This calculation factored symmetric band matrices with random coefficients using a Block LDL factorization for different matrix orders. This matrix structure is often encountered in HVAC simulations. Below a matrix dimension of 64 dense matrices factored faster than sparse matrices. However, limitations in the precision of MATLAB's timing functions effect accuracy below an order of 28.



**Figure 79 – Sparse vs. Dense Matrix Factorization Speed**

In a building energy simulation, parameters such as outside air conditions and loads vary from time step to time step. Sensitivity analysis [42] uses matrix factorizations on a previous simulation time step to estimate the solution on next time step. Initializing a constrained optimization calculation this way reduces the number of iterations required to solve the problem and can result in more robust solutions.

Appendix D gives the derivation for the sensitivity analysis equations used. These equations were derived to handle parametric upper and lower bounds on variables. The parametric solver sIPOPT cannot take parametric bounds into account directly since it was built using the constraints of the IPOPT solver. Parametric bounds can occur in HVAC simulation problems such as when minimum supply air flow rates are on a time schedule.

## *Compilation*

After generating C# source code for a model, the compilation module compiles it into an executable .dll. This uses Microsoft's Roslyn compiler, which allows C# programs to be compiled within other C# programs. Roslyn is part of the .NET framework, so Beryl doesn't need any external libraries to compile and execute generated C# source code. Since Roslyn is the default C# compiler for Visual Studio 2017, programs compiled with it perform the same as programs compiled in Visual Studio itself.

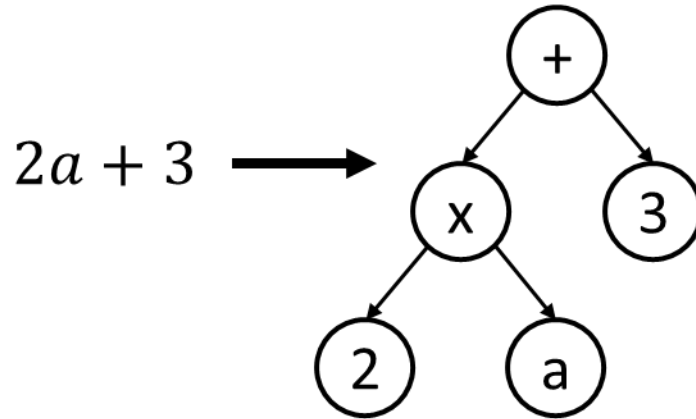
## *Computer Algebra Library*

Computer algebra systems allow symbolic mathematical operations to be performed and on computers. A computer algebra library was implemented in C# to parse, manipulate, differentiate, and simplify mathematical expressions. All modules except for the numerical algorithms make use of this library.

## **Expression Trees**

The computer algebra library represents mathematical expressions using expression tree data structures, as seen in Figure 80. In expression trees operators serve as internal nodes and constants or variables serve as leaf nodes. Evaluating an expression tree involves fixing variables to constant values and performing a traversal that evaluates subtrees from the children up to the root.

## Equation      Expression Tree



**Figure 80 – A Linear Function Represented with an Expression Tree**

### Algebraic Simplification

Automatic simplification of algebraic expressions involves applying simplification rules to remove extraneous symbols from expressions and transform them into a standard form. The computer algebra library implements the algebraic simplification algorithm from [79]. This algorithm recursively applies transformations to an algebraic expression tree to transform it to a more compact form. Automatic simplification is used during preprocessing so that expressions don't grow to an unnecessary length. Also, constants in expression trees are stored as rational numbers with an infinite precision so no precision is lost during the simplification process.

Edge cases exist where the rules of auto-simplification can produce different results for equivalent expressions. For instance, equation 7.6 shows zero raised to the power of one, which simplifies to zero. However, in equation 7.7 zero raised to the same power simplifies to undefined since the simplification algorithm can't simplify the trigonometric part of the expression. Cases such as this are an inherent part of computer algebra systems and can require users to understand computer algebra on a deeper level to ensure that algebraic expressions are processed correctly.

$$0^1 \rightarrow 0 \quad 7.6$$

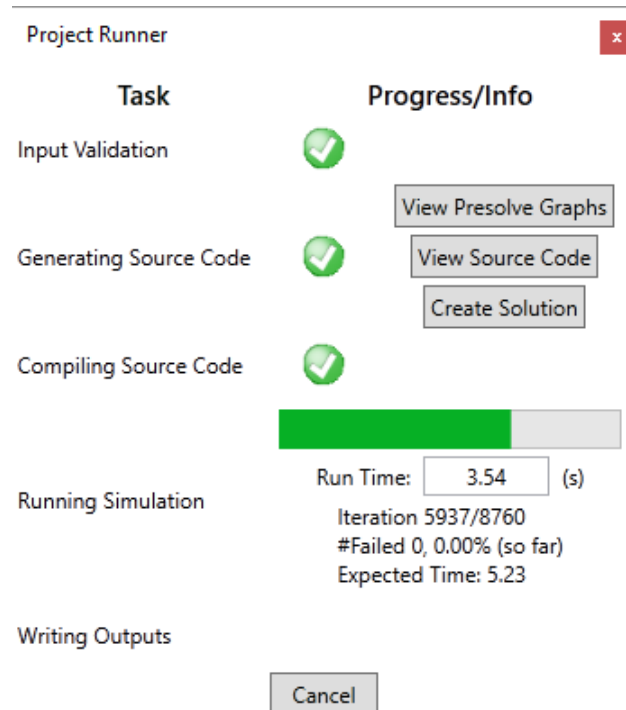
$$0^{2-\sin^2(x)-\cos^2(x)} \rightarrow \text{Undefined} \quad 7.7$$

### **Built-In Functions**

The computer algebra library includes two special functions and their first and second derivatives for use in HVAC modeling and optimization problems. First,  $P_{ws}(T)$  gives water saturation pressures in *psia* are given as a function of temperature in  $^{\circ}F$ . Appendix B gives the formulas used in this equation. Also, for smoothing discontinuous functions the library implements the gamma function from Bertsekas [58, 80]. Appendix A discusses this function in more detail.

### **Running a Simulation**

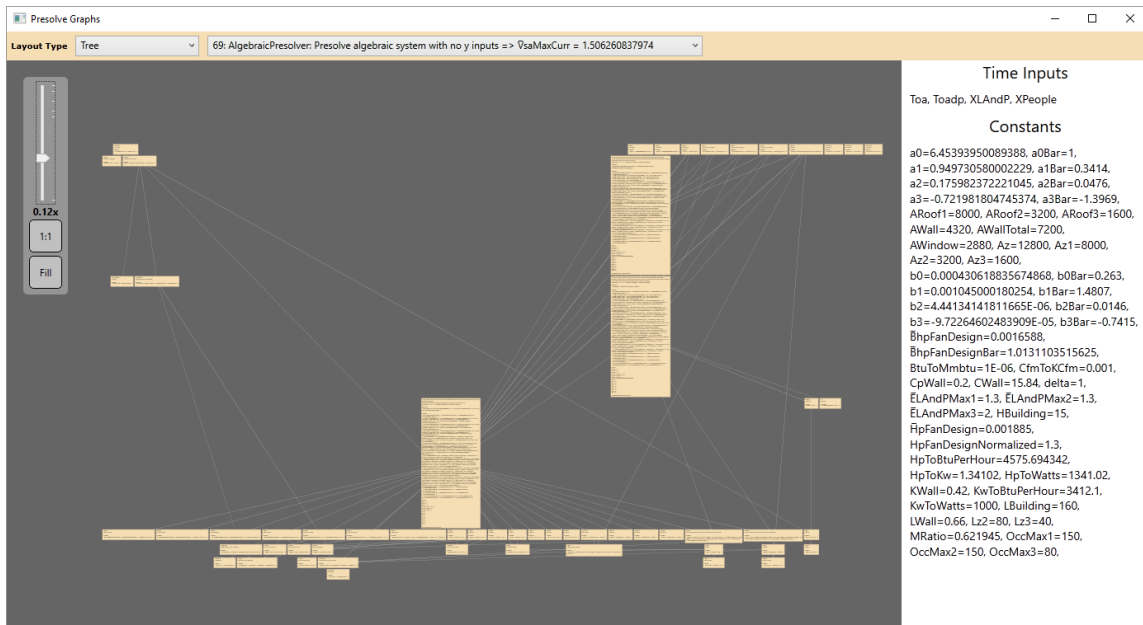
When a user runs a simulation from the user interface the process of parsing an input, preprocessing a model, compiling the model's source code, executing a model, and showing the model's outputs is shown in Beryl's Project Runner Window. Figure 81 shows this window. In this window information is given for each of the five execution stages. When a model executes successfully green check marks show up for first three stages, and the Running Simulation stage shows a simulation's progress, failed iterations, and expected runtime. After a simulation completes output Excel files and charts are written and opened. Iterations that fail due to algorithms not converging appear as blank rows in the output Excel file.



**Figure 81 – The Project Runner Window**

If an execution stage fails information can be displayed on why this happened. If input validation, source code generation, or source code compilation fails a list of errors can be displayed. Uncaught exceptions can also be viewed. This allows Beryl to be more easily debugged.

Clicking “View Presolve Graphs” in the Project Runner Window displays the Presolve Graph Window, shown in Figure 82. This window displays time inputs, constants, and the compilation graph after each action performed by the presolving algorithm from Chapter VI. Connections between vertices give the dependence between displayed subproblems. This allows for the preprocessing process to be examined more closely.

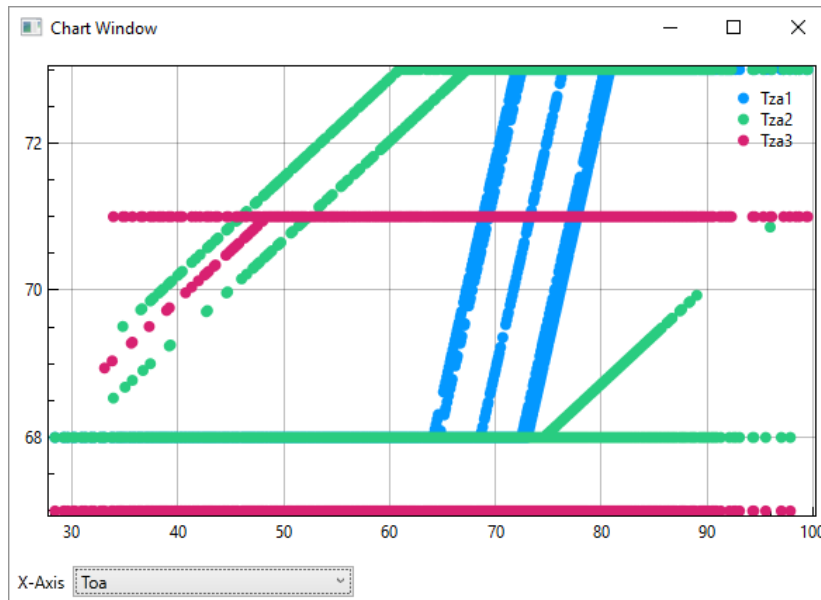


**Figure 82 – Presolve Graph Window**

Clicking “View Source Code” in the Project Runner Window creates and displays a text file of the source code that was generated to simulate a model. If source code was generated with errors the compiler generated error messages are displayed above the line where the error was detected. Clicking “Create Solution” in the Project Runner Window generates a Visual Studio solution with the generated source code and a console app that calls the generated source code with the same inputs as the Beryl model that was ran. This was used to debug generated code by viewing its calculations line by line.

Model outputs from each time step can be displayed in an output Excel file or a chart. Excel outputs can be raw data from each time step or an hourly, daily, monthly, or yearly summation can be performed. For dynamic simulations the instantaneous values of dynamic variables at different points in a time step can be outputted, as seen in the dynamic wall temperature of Figure 64. Figure 83 shows an example of a chart window where three zone air temperatures are plotted against the outside air temperature.





**Figure 83 – An Output Chart Window in Beryl**

### **Validation and Unit Testing**

Unit tests are executable pieces of code written to automatically test the correctness of other code modules [81]. Unit tests allow more in-depth testing of code than manual tests because of their speed. For instance, in a unit test for the computer algebra library 50,000 algebraic expressions were randomly generated and tested to ensure that converting expressions to strings works correctly. This type of testing helps catch edge cases that a programmer might not realize exist when designing manual tests. Unit tests can be performed regularly to flag source code changes that break previously working implementations. This helps uncover bugs that were introduced during refactoring, maintenance, and when making minor changes.

Unit tests make up over 6,300 lines of Beryl’s source code, which is approximately 20% of the code base. Unit tests were created at several levels of resolution to validate functions, classes, projects, and the overall program execution. Debugging modules from the bottom up was critical to getting the program working by helping to ensure that bugs were uncovered while modules were being created and before integrating them together.

## Performance

Calculation speed and robustness limit the practical applications where models can be used. This is due to limits on computer resources and the limited amount of work time that modelers have. These limitations are made worse by the need for multiple model runs to calibrate or optimize a model. Beryl was able to simulate a single AHU model from Chapter V for an entire year using hourly time steps in approximately 0.5 to 5.0 seconds. The following analysis uses this baseline to make projections for the speed of larger and more complex models, including cases where additional model features can be added at little or no cost.

### *Limiting Factors in Calculation Speed*

Beryl's numerical solvers for algebraic, unconstrained optimization, and constrained optimization problems are all based on linear algebra. The main bottlenecks of these solvers are linear algebra factorization, linear algebra solving, and function and gradient evaluations. In a CPU usage profile of a typical SDVAV simulation Beryl spent approximately 60% of its computational effort in linear algebra and approximately 20% of its effort in function evaluations.

Time wise, building energy use simulations in Beryl primarily use the constrained optimization solver IPNLP. Internally, IPNLP factors and solves square symmetric indefinite matrices known as KKT matrices [82]. KKT matrices have a dimension of  $n + m$ , where  $n$  is the number of variables and  $m$  is the number of equality constraints. Internally they have a  $2 \times 2$  block structure, as shown in 7.8. The upper left submatrix  $H$  is symmetric with a dimension of  $n \times n$ , and the lower right-hand submatrix  $C$  has a diagonal structure with a dimension of  $m \times m$ .

$$\begin{bmatrix} H & A \\ A^T & -C \end{bmatrix} \quad 7.8$$

The number of operations to factor a KKT matrix is proportional to  $(n + m)^3$ , and the number of operations to solve a system of linear equations using a factored KKT

matrix is  $(n + m)^2$  [75]. Factoring algorithms that take advantage of symmetry can use fewer floating point operations than matrix algorithms that don't, but the number of floating point operations required is still proportional to  $(n + m)^3$ . Interior point algorithms tend to solve convex optimizations in the same number of iterations regardless of problem size [83]. This was found to be true for building energy simulations as well, so as the number of variables and equality constraints in a problem grows the inefficiency matrix factorization can eventually be expected to dominate the computational effort. Beryl's other numerical solvers are dominated by matrix factorization in the same way. However, their matrices scale with  $n$  alone instead of  $n + m$ .

Table 33 lists several model additions and their effect on a model's runtime. Adding variables and equations that don't increase the size of the KKT matrix used in the interior point algorithm don't increase the runtime at a polynomial rate. Because of this, adding a variable and equation that can be calculated from outputs of an interior point model or adding dynamics in the form of linear differential equations with a constant  $A$  matrix can be performed at the cost of calculating a single function. Adding an inequality to a problem or a zone model typically increases the KKT matrix size though due to slack variables that are added or equations that aren't decomposable. Adding a lexicographic objective creates an additional optimization that has to be performed, so the runtime is increased proportionally assuming that the complexity of all optimizations are the same. Similarly, adjusting the time step size proportionally scales the time required to perform a simulation. Simulating a building as a whole by coupling multiple air handlers together has the largest effect of these modifications. This is due to how matrix factorization has a cubic dependence, how combining multiple air handlers multiplies the KKT matrix's size, and how problems at this scale can't take advantage of using dense matrices in their calculations.

**Table 33 – Effect of Model Additions on Runtime**

<b>Feature</b>	<b>Typical Effect on Runtime</b>
Adding a linear equation and 1 variable	KKT matrix stays the same
Adding a decomposable nonlinear equation and 1 variable	KKT matrix stays the same
Adding dynamics in the form $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{b}(\mathbf{y})$	KKT matrix stays the same
Adding an inequality	Increases the KKT matrix dimension by up to 2
Adding a zone	Increases KKT matrix size
Adding a lexicographic objective	Runtime scales linearly with the number of objectives
Adjusting time step size	Linearly scales the simulation time
Coupling AHUs	Doubles the KKT matrix size, increasing matrix factorization operations by a factor of 8

### *Measured Performance*

The runtime performance of the models presented in Chapter 5 were analyzed to determine the main factors effecting performance. These results were then extrapolated to determine the limits of where constrained optimization based building energy simulation can be performed. All simulations were performed on a desktop computer Intel Core i7-3770 CPU at 3.40 GHz with 6 GB of RAM. All model variables that change from time step to time step were calculated as outputs over the course of a year using hourly time steps, and calculations took place on a single thread. When running the same model multiple times runtimes were found to vary approximately 1%.

Table 34 gives the runtimes of the models from Chapter V where the output equations of each model was placed in a single initial partition. The decomposition algorithm from Chapter VI reduced the constrained problems down to the dimension shown. Default user options for all algorithms were used, which are given in Appendix E.

The SDCAV system ran the fastest due to its smaller problem size, more linear structure and the way that having one lexicographic objective means that only one constrained optimization needs to be performed on each time step. The SDVAV system ran slowest due to it requiring larger matrix factorizations, having to perform three constrained optimizations per time step, and its more complex nonlinear structure due to static pressure modeling. The DDVAV system ran faster than the SDVAV system even though it requires factoring the largest matrices and uses three constrained optimizations per time step because of its simpler equation structure.

**Table 34 – Full Model Runtimes**

<b>Model</b>	<b>Number of Lexicographic Objectives (<math>q</math>)</b>	<b>Reduced Matrix Sizes for Each Objective (<math>n + m</math>)</b>	<b>Yearly Simulation Runtime (<i>seconds</i>)</b>
SDCAV	1	16 (11 + 5)	0.38
SDVAV	3	39 (24 + 15), 40 (24 + 16), 35 (21 + 14)	4.84
DDVAV	3	51 (30 + 21), 52 (30 + 22), 53 (30 + 23)	4.54

Beryl models can be manually partitioned to take advantage of decompositions that the simplification algorithm in Chapter VI can't detect. To test the effectiveness of this, Temperature and humidity calculations were manually decoupled in the models from Table 34. Table 35 shows the reduced constrained optimization matrix sizes and runtimes after making this simplification. Compared to the models that start in a single partition, the partitioned models ran approximately 30% to 40% faster and used matrices that were approximately 20% to 30% smaller.

**Table 35 – Partitioned Temperature and Humidity Model Runtimes**

<b>Model</b>	<b>Number of Lexicographic Objectives (<math>q</math>)</b>	<b>Reduced Matrix Sizes for Each Objective (<math>n + m</math>)</b>	<b>Yearly Simulation Runtime (<i>seconds</i>)</b>
SDCAV	1	12 (9 + 3)	0.26
SDVAV	3	29 (19 + 10), 30 (19 + 11), 27 (17 + 10)	2.86
DDVAV	3	35 (22 + 13), 36 (22 + 14), 37 (22 + 15)	2.97

Adding dynamics based on linear differential equations with a constant  $A$  matrix to a model has a minor impact on the model’s runtime. This is due to how the decomposition algorithm solves the equations for their average values over a time step and can remove the resulting linear equations. To demonstrate this, the single-partition models for each system type were simulated with a six-node dynamic wall model instead of using a constant R-value. As a result, the SDCAV and SDVAV systems ran approximately 25% slower due to more iterations being required time step, while the DDVAV system ran in approximately the same time.

**Table 36 – Full Model Runtimes with Six Node Wall Dynamic Model**

<b>Model</b>	<b>Number of Lexicographic Objectives (<math>q</math>)</b>	<b>Reduced Matrix Sizes for Each Objective (<math>n + m</math>)</b>	<b>Yearly Simulation Runtime (<i>seconds</i>)</b>
SDCAV	1	16 (11 + 5)	0.49
SDVAV	3	39 (24 + 15), 40 (24 + 16), 35 (21 + 14)	6.06
DDVAV	3	51 (30 + 21), 52 (30 + 22), 53 (30 + 23)	4.51

Sensitivity analysis for constrained optimization plays a large role in reducing the runtime of a model by choosing better starting conditions. This results in fewer iterations being required to solve constrained optimizations. Table 37 shows runtimes of the full model without sensitivity analysis. The amount of slowdown from not using sensitivity analysis was relative to the complexity of the model. The SDCAV model was

slower by a factor of 2.7 while the SDVAV and DDVAV models were slower by factors of 26 and 16.

**Table 37 – Full Model Runtimes without Sensitivity Analysis**

<b>Model</b>	<b>Number of Lexicographic Objectives (<i>q</i>)</b>	<b>Reduced Matrix Sizes for Each Objective (<i>n + m</i>)</b>	<b>Yearly Simulation Runtime (<i>seconds</i>)</b>
SDCAV	1	16 (11 + 5)	1.02
SDVAV	3	39 (24 + 15), 40 (24 + 16), 35 (21 + 14)	124.52
DDVAV	3	51 (30 + 21), 52 (30 + 22), 53 (30 + 23)	71.55

## CHAPTER VIII

### CONCLUSIONS AND FUTURE RESEARCH

Future building energy simulation tools must deal with the increasing complexity of buildings and their systems. Equation-based modeling aids this by separating the domains of modeling and solving and allowing for modular approaches. Extending the math that an equation-based modeling tool can process offers a way to model multiple domains in the same environment. This thesis presented a way to model steady state building energy usage using an equation-based approach, an extended the approach to model wall dynamics, and a solver to investigate the practicality of this approach.

#### **Conclusions**

Constrained optimization allows building energy models to be constructed that are equation-based models and solvable by numerical solvers. These numerical solvers are based on linear algebra calculations, which makes models scalable in the same way as models that use algebraic or differential equations.

A strategy for adding dynamic variables to steady state building energy models is to couple their averages over a time step with the steady state variables. This can be accomplished by embedding the solutions of differential equations over a time step into the steady state part of the model. Due to presolving, simulations with added dynamics perform on the same order of magnitude of time when dynamic variables are described by linear differential equations with constant coefficients.

Simplified building energy use models for three air handler types were presented. These models are on the order of complexity as Knebel [9], with additional features such as additional zones and static pressure setpoint modeling. The techniques used in constructing these models can be extended to larger models and different types of equipment.

An algorithm to decompose and simplify lexicographic constrained optimizations with embedded dynamics was developed to solve simplified building energy models in a practical way. This algorithm decomposes lexicographic optimizations into a series of



optimizations and embeds linear differential equations with constant coefficients for the dynamic variables into the model.

A numerical solver named Beryl was created to test the level of speed and robustness that could be achieved when solving constrained optimization based building energy models. Beryl was optimized for the small, parametric problems of simplified building energy use modeling, since dense matrices calculate faster than sparse matrices.

Beryl contains a user interface where models can be created using a domain specific language that was created for simplified building energy modeling. When models are simulated the decomposition algorithm is first executed on it to simplify the calculations and develop a solution strategy. Next, a custom program is written and compiled to run the simulation before displaying outputs in an Excel file or chart format. This program links to external numerical solvers, including a constrained optimization solver based on the IPOPT and sIPOPT solvers.

Performance wise, Beryl was able to perform yearly simulations using hourly time steps for the example AHU models in approximately 0.5 to 5.0 seconds on a single thread. Model runs succeeding on Beryl's default settings meant that models could be simulated and modified without having to worry about lowering convergence errors or tweaking settings to ensure that models calculate. Having a runtime in seconds combined with the runtime scaling properties of matrices mean that this approach can be used for larger, more detailed models as well.

## **Future Research**

### *Modeling Improvements*

Models for variations SDCAV, SDVAV, and DDVAV systems were presented, and the techniques used in these models can be extended to larger models and different air handler types. However, modeling different equipment requires taking the features of nonlinear programming algorithms into account. Formulations of existing equipment models may need to be modified to run successfully or more efficiently in a nonlinear programming solver.

The lexicographic constrained optimization models that were presented use floating point variables. Integer variables can also be used to model features such as AHU on/off schedules or staged chillers where one or more can be on at the same time. In the case of AHU on/off schedules, a binary variable could determine which model to use over a time step. The value of this variable would be known before a simulation. In the case of staged chillers binary variables could be used to signify that a chiller is on or off. Their values would have to be simultaneously calculated with unknown floating point variables, resulting in a mixed-integer optimization.

Building energy models are often used to optimize a control parameter such as a temperature setpoint or schedule. Parameter optimizations using the methods presented optimize parameters on each individual time step but let the optimized value vary across time steps. Time steps can be calculated simultaneously in order to optimize parameters across time steps. In this strategy variables and equations are repeated over multiple time steps and combined into a single problem. However, this creates a very large problem. It would be ideal to exploit the independence or sequential nature of time steps within a problem. Sensitivity analysis provides a way to directly calculate gradients for a model. This could allow optimizations of parameters that stay fixed over the course of a year to be solvable in the time it takes to perform a few dozen yearly simulations.

#### *Solver Improvements*

The overall strength of the numerical solver depends on the algorithms it calls during a simulation. Beryl uses an interior point method for constrained optimization calculations and the Levenberg-Marquardt algorithm for algebraic calculations, but this can be extended to use other solvers that work better for specific problem cases. For instance, a linear algebra solver can be used to solve linear algebraic problems, or sequential quadratic programming can be used in addition to interior point methods. Solvers that provide a global solution or use quad or infinite precision mathematics can be added as well to be able to calculate more accurate solutions. Also, sparse matrix calculations can be added in addition to the dense matrix calculations that are used to more efficiently handle large building simulations.

Time steps of steady state simulations are typically simulated in order from the simulation start date to end date. However, the solutions for the time steps of steady state models don't depend on one another. This allows multithreading to be used, which increases the calculation speed proportionally to the number of threads used. However, the time step solutions for dynamic models depend on each other, so they can't be executed out of order. Multithreading can also be used to speed up linear algebra calculations if a linear algebra library that implements multithreading is used.

The structure of a constrained optimization problem allows any equation-based model to be coupled together. For instance, partial differential equations and dynamic control models can be coupled with steady state building energy models by coupling the equations and embedding dynamic solutions within the steady state model. Optimizations of these models can also be expressed by adding optimization objectives. This allows a single simulation environment to be used for a variety of building modeling types, with the ability to combine different modeling methods together. Such a solver could also make reusing models easier since inputs could be shared by a variety of users, such as architects, building retro-commissioners, and HVAC control designers.

## REFERENCES

- [1] D.B. Crawley, J.W. Hand, M. Kummert, B.T. Griffith, Contrasting the capabilities of building energy performance simulation programs, *Building and environment*, 43 (4) (2008) 661-673.
- [2] M. Wetter, A view on future building system modeling and simulation, in, Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA (US), 2011.
- [3] D.E. Fisher, R.D. Taylor, F. Buhl, R.J. Liesen, R.K. Strand, A modular, loop-based approach to HVAC energy simulation and its implementation in EnergyPlus, in: *Proceedings of Building Simulation*, 1999, pp. 1245-1252.
- [4] M. Wetter, W. Zuo, T.S. Noudui, X. Pang, Modelica buildings library, *Journal of Building Performance Simulation*, 7 (4) (2014) 253-270.
- [5] J. Åkesson, Optimica—an extension of modelica supporting dynamic optimization, in: *Proc. 6th International Modelica Conference*, 2008.
- [6] ASHRAE, Energy Estimating and Modeling Methods, in: *ASHRAE Fundamentals*, 2009.
- [7] B. Abushakra, A. Sreshthaputra, J. Haberl, D. Claridge, Compilation of Diversity Factors and Schedules for Energy and Cooling Load Calculations, ASHRAE Research Project 1093-RP, Final Report, in, 2001.
- [8] R.G. Quayle, H.F. Diaz, Heating degree day data applied to residential heating energy consumption, *Journal of Applied Meteorology*, 19 (3) (1980) 241-246.
- [9] D.E. Knebel, Simplified Energy Analysis Using the Modified Bin Method, American Society of Heating, Refrigerating, and Air-Conditioning Engineers, 1983.
- [10] R. Mazzucchi, K. Devine, Energy Use Analysis for the Federal Energy Management Program, in, Energy Systems Laboratory (<http://esl.tamu.edu>); Texas A&M University (<http://www.tamu.edu>), 1988.
- [11] K. Devine, R. Mazzucchi, Use of Metering for Facility and Whole Building Energy Analysis by the US Department of Energy Federal Energy Management Program, in, PNL-SA—17155, Richland, Wash.: Pacific Northwest Laboratory, 1989.

- [12] S. Oh, Origins of Analysis Methods in Energy Simulation Programs Used for High Performance Commercial Buildings, Texas A&M University, 2013.
- [13] USPS, United States Postal Service Symposium: Computer Program for Analysis of Energy Utilization, in, Washington, D.C., 1971.
- [14] J.M. Ayres, Predicting building energy requirements, Energy and Buildings, 1 (1) (1977) 11-18.
- [15] J. Haberl, S. Cho, Literature Review of Uncertainty of Analysis Methods,(DOE-2 Program), Report to the Texas Commission on Environmental Quality, (2004).
- [16] B. Birdsall, W. Buhl, K. Ellington, A. Erdem, F. Winkelmann, Overview of the DOE-2 building energy analysis program, Report LBL-19735m rev. w, Lawrence Berkeley Laboratory, Berkeley, CA, (1990).
- [17] D.B. Crawley, L.K. Lawrie, F.C. Winkelmann, W.F. Buhl, Y.J. Huang, C.O. Pedersen, R.K. Strand, R.J. Liesen, D.E. Fisher, M.J. Witte, EnergyPlus: creating a new-generation building energy simulation program, Energy and Buildings, 33 (4) (2001) 319-331.
- [18] U.S. Army CERL, BLAST Fact Sheet, in: U.S. Army (Ed.), Champaign, IL, 1999.
- [19] R.K. Strand, C.O. Pedersen, D. Crawley, Modularization and simulation techniques for heat balance based energy and load calculation programs: The experience of the ASHRAE loads toolkit and EnergyPlus, Proceedings of Building Simulation 2001, (2001) 43-50.
- [20] R. Strand, F. Winkelmann, F. Buhl, J. Huang, R. Liesen, C. Pedersen, D. Fisher, R. Taylor, D. Crawley, L. Lawrie, Enhancing and extending the capabilities of the building heat balance simulation technique for use in EnergyPlus, in: Proceedings of Building Simulation'99, 1999, pp. 653-660.
- [21] G.N. Walton, AIRNET: A Computer Program for Building Airflow Network Modeling, National Institute of Standards and Technology Gaithersburg, USA, 1989.
- [22] LBNL, COMIS Fundamentals, in, Lawrence Berkeley National Laboratory, 1990.
- [23] H.E. Feustel, COMIS - An International Multizone Air-Flow and Contaminant Transport Model, in, Lawrence Berkeley National Laboratory, 1998.

- [24] G.N. Walton, W.S. Dols, CONTAM User Guide and Program Documentation, in, National Institute of Standards and Technology, Gaithersburg, MD, 2005.
- [25] G. Silverman, S. Jurovics, D. Low, E. Sowell, Modeling and optimization of HVAC systems using network concepts, ASHRAE Transactions, 87 (2) (1981) 585-597.
- [26] R. Sargent, A review of methods for solving nonlinear algebraic equations, Foundations of Computer-Aided Chemical Process Design, American Institute of Chemical Engineers, New York, (1981).
- [27] E. Sowell, K. Taghavi, H. Levy, D. Low, Generation of building energy system models, ASHRAE transactions, 90 (1B) (1984) 573-586.
- [28] J. Clarke, The energy kernel system, Energy and Buildings, 10 (3) (1988) 259-266.
- [29] W. Buhl, A. Erdem, F. Winkelmann, E. Sowell, Recent improvements in SPARK: strong component decomposition, multivalued objects, and graphical interface, in, Lawrence Berkeley Lab., CA (United States), 1993.
- [30] E.F. Sowell, P. Haves, Numerical performance of the SPARK graph-theoretic simulation program, in: Proceedings of the Building Simulation, Citeseer, 1999.
- [31] E.F. Sowell, P. Haves, Efficient solution strategies for building energy system simulation, Energy and Buildings, 33 (4) (2001) 309-317.
- [32] H. Elmqvist, S.E. Mattsson, M. Otter, Modelica: The new object-oriented modeling language, in: 12th European Simulation Multiconference, Manchester, UK, 1998.
- [33] C. Park, D.R. Clark, G.E. Kelly, An overview of HVACSIM+, a dynamic building/HVAC/control systems simulation program, in: Proceedings of the 1st Annual Building Energy Simulation Conference, Seattle, WA, 1985.
- [34] T.P. McDowell, D.E. Bradley, J.W. Thornton, M. Kummert, Simulation synergy: expanding TRNSYS capabilities and usability, (2004).
- [35] M. Wetter, T.S. Nouidui, D. Lorenzetti, E.A. Lee, A. Roth, Prototyping the next generation energyplus simulation engine, in: Proceedings of the 3rd IBPSA Conference, Jeju island, South Korea, 2015, pp. 27-29.
- [36] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, The functional mockup interface for tool

- independent exchange of simulation models, in: Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany, Linköping University Electronic Press, 2011, pp. 105-114.
- [37] B.P. Zeigler, J.S. Lee, Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment, in: Enabling Technology for Simulation Science II, International Society for Optics and Photonics, 1998, pp. 49-59.
- [38] M. Wetter, GenOpt®- A Generic Optimization Program, in: Seventh international IBPSA conference, 2001, pp. 601-608.
- [39] M. Wetter, GenOpt, A Generic Optimization Program, Version 3.0.0 User Manual 3.0, in, Berkeley, CA, 2009.
- [40] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, H. Tummescheit, Modeling and optimization with Optimica and JModelica. org—Languages and tools for solving large-scale dynamic optimization problems, *Computers & Chemical Engineering*, 34 (11) (2010) 1737-1749.
- [41] A. Wächter, L.T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, *Mathematical Programming*, 106 (1) (2004) 25-57.
- [42] H. Pirnay, R. López-Negrete, L.T. Biegler, Optimal sensitivity based on IPOPT, *Mathematical Programming Computation*, (2012) 1-25.
- [43] S.P. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [44] R. Fourer, A simplex algorithm for piecewise-linear programming I: Derivation and proof, *Mathematical Programming*, 33 (2) (1985) 204-233.
- [45] P. Wolfe, The simplex method for quadratic programming, *Econometrica: Journal of the Econometric Society*, (1959) 382-398.
- [46] P.d. Fermat, *Method for the Study of Maxima and Minima*, 1636.
- [47] J. Lagrange, *Mécanique Analitique*, la Veuve Desaint, in, Paris, 1788.
- [48] W. Karush, *Minima of functions of several variables with inequalities as side conditions*, Master thesis, University of Chicago, (1939).

- [49] H. Kuhn, A. Tucker, Nonlinear programming. In Proceedings of the second Berkeley symposium on mathematical statistics and probability, 5, in, University of California Press, Berkeley, 1951.
- [50] Nobel Media AB, The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 1975, in, [http://www.nobelprize.org/nobel\\_prizes/economic-sciences/laureates/1975/](http://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/1975/), 2014.
- [51] G.B. Dantzig, Linear Programming and Extensions, Princeton University Press, 1965.
- [52] M. Wright, The interior-point revolution in optimization: history, recent developments, and lasting consequences, Bulletin of the American mathematical society, 42 (1) (2005) 39-56.
- [53] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: Proceedings of the sixteenth annual ACM symposium on Theory of computing, ACM, 1984, pp. 302-311.
- [54] R.T. Marler, J.S. Arora, Survey of multi-objective optimization methods for engineering, Structural and multidisciplinary optimization, 26 (6) (2004) 369-395.
- [55] J.P. Ignizio, Goal Programming and Extensions, Lexington Books Lexington, MA, 1976.
- [56] M.J. Rentmeesters, W.K. Tsai, K.-J. Lin, A theory of lexicographic multi-criteria optimization, in: Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on, IEEE, 1996, pp. 76-79.
- [57] S.M. Robinson, R.H. Day, A sufficient condition for continuity of optimal sets in mathematical programming, Journal of Mathematical Analysis and Applications, 45 (2) (1974) 506-511.
- [58] D.P. Bertsekas, Nonlinear programming: Second Edition, Athena Scientific, Belmont, Massachusetts, 2008.
- [59] Y. Ma, G. Anderson, F. Borrelli, A distributed predictive control approach to building temperature regulation, in: American Control Conference (ACC), 2011, IEEE, 2011, pp. 2089-2094.



- [60] P.T. Boggs, J.W. Tolle, Sequential quadratic programming, *Acta Numerica*, 4 (1) (1995) 1-51.
- [61] R.J. Dakin, A tree-search algorithm for mixed integer programming problems, *The Computer Journal*, 8 (3) (1965) 250-255.
- [62] J. Clausen, Branch and bound algorithms-principles and examples, Department of Computer Science, University of Copenhagen, (1999) 1-30.
- [63] G.R. Zheng, M. Zaheer-Uddin, Optimization of thermal processes in a variable air volume HVAC system, *Energy*, 21 (5) (1996) 407-420.
- [64] G.R. Zheng, Dynamic modeling and global optimal operation of multizone variable air volume HVAC systems, Concordia University, 1997.
- [65] Y. Ma, F. Borrelli, B. Hency, A. Packard, S. Bortoff, Model predictive control of thermal energy storage in building cooling systems, in: *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, IEEE, 2009*, pp. 392-397.
- [66] Y. Ma, F. Borrelli, B. Hency, B. Coffey, S. Bengea, P. Haves, Model predictive control for the operation of building cooling systems, *Control Systems Technology, IEEE Transactions on*, 20 (3) (2012) 796-803.
- [67] Y. Ma, S. Vichik, F. Borrelli, Fast stochastic MPC with optimal risk allocation applied to building control systems, in: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on, IEEE, 2012*, pp. 7559-7564.
- [68] A. Kelman, Y. Ma, F. Borrelli, Analysis of local optima in predictive control for energy efficient buildings, in: *50th IEEE Conference on Decision and Control and European Control Conference, Orlando, FL, USA, 2011*, pp. 5125-5130.
- [69] R. Lindberg, M. Korpi, J. Vinha, Factors affecting energy consumption of buildings, in: *Proceedings of the BEST1 conference: Building Enclosure Science and Technology, 2008*.
- [70] A. Robertson, D. Gross, An electrical-analog method for transient heat-flow analysis, *Journal of Research of the National Bureau of Standards*, 61 (2) (1958) 105.

- [71] J.N. Lott, R. Baldwin, The FCC integrated surface hourly database: A new resource of global climate data, in, NOAA, 2001.
- [72] ASHRAE, 2016 ASHRAE Handbook - Systems and Equipment, Inch-Pound Edition ed., American Society of Heating, Refrigeration, and Air-Conditioning Engineers, Inc., Atlanta, GA, 2016.
- [73] SMACNA, HVAC Systems Testing, Adjusting, and Balancing, 3rd Edition ed., Sheet Metal and Air Conditioning Contractors' National Association, Inc., Chantilly, VA, 2002.
- [74] J. Åkesson, T. Ekman, G. Hedin, Development of a Modelica compiler using JastAdd, *Electronic Notes in Theoretical Computer Science*, 203 (2) (2008) 117-131.
- [75] Intel, Intel math kernel library reference manual, in, 2011.
- [76] T.Æ. Mogensen, *Basics of Compiler Design*, Torben Ægidius Mogensen, 2009.
- [77] M.S. Bazaraa, H.D. Sherali, C.M. Shetty, *Nonlinear Programming: Theory and Algorithms*, John Wiley & Sons, 2013.
- [78] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, D. Sorensen, LAPACK: A portable linear algebra library for high-performance computers, in: *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, IEEE Computer Society Press, 1990, pp. 2-11.
- [79] J.S. Cohen, *Computer Algebra and Symbolic Computation: Mathematical methods*, Universities Press, 2003.
- [80] D.P. Bertsekas, *Nondifferentiable optimization via approximation*, *Nondifferentiable optimization*, (1975) 1-25.
- [81] R. Osherove, *The Art of Unit Testing: With Examples in C#*, Manning, 2013.
- [82] M. Hagemann, O. Schenk, A. Wächter, Combinatorial approaches to the solution of saddlepoint problems in large-scale parallel interior-point optimization, in, *Research Report RC 23824*. IBM TJ Watson Research Center, Yorktown, USA, December, 2005.
- [83] J. Gondzio, Interior point methods 25 years later, *European Journal of Operational Research*, 218 (3) (2012) 587-601.
- [84] C.-T. Chen, *Linear System Theory and Design*, Oxford University Press, Inc., 1995.

[85] B. Blackadar, A General Implicit/Inverse Function Theorem, arXiv preprint arXiv:1509.06025, (2015).

APPENDIX A  
TIME AVERAGES OF SOLUTIONS TO LINEAR DIFFERENTIAL EQUATIONS

Variables that vary continuously over time can be coupled to steady state building energy models by embedding equations for their average value over a time step period within the model. Equation A.1 shows the form of a linear differential equation where the  $\mathbf{A}$  and  $\mathbf{b}$  matrices are constant over a time period from its start at  $t = 0$  to its end at  $t = \rho$ . The average value of  $\mathbf{x}(t)$  over a time step is calculated by first calculating the time varying solution of  $\mathbf{x}(t)$  and then averaging it over a time step.

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{b} \quad \text{A.1}$$

**Time Varying Solution**

Equation A.2 gives the overall solution to the system of linear equations in equation A.1 starting at  $t = 0$  [84]. The properties of matrix exponential functions are used to transform equation A.2 into equation A.4 in equations A.2–A.4. The matrix integral is evaluated in equation A.6 using equation A.5. After this, the properties of matrix exponentials are used to transform equation A.6 into the solution's final form in equation A.8 in equations A.6–A.8.

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{b}d\tau \quad \text{A.2}$$

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}t}e^{-\mathbf{A}\tau}\mathbf{b}d\tau \quad \text{A.3}$$

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + e^{\mathbf{A}t} \int_0^t e^{-\mathbf{A}\tau}d\tau \mathbf{b} \quad \text{A.4}$$

$$\int_0^t e^{\mathbf{A}\tau}d\tau = (e^{\mathbf{A}t} - \mathbf{I})\mathbf{A}^{-1} \quad \text{A.5}$$

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) - e^{\mathbf{A}t}(e^{-\mathbf{A}t} - \mathbf{I})\mathbf{A}^{-1}\mathbf{b} \quad \text{A.6}$$

$$\mathbf{x}(t) = e^{At}\mathbf{x}(0) - (I - e^{At})\mathbf{A}^{-1}\mathbf{b} \quad \text{A.7}$$

$$\mathbf{x}(t) = e^{At}\mathbf{x}(0) + (e^{At} - I)\mathbf{A}^{-1}\mathbf{b} \quad \text{A.8}$$

### Average of Time Varying Solution

Equation A.9 is the general equation for the time-average of a function over time step with a period of  $\rho$ . Equation A.10 shows equation A.9 with the time varying solution of equation A.8 substituted into it. Equations A.11–A.15 evaluates and simplifies these integrals. This results in A.15, the average of the dynamic states as linear functions of the initial conditions and the input vector  $\mathbf{b}$ .

$$\bar{\mathbf{x}} = \frac{1}{\rho} \int_0^\rho \mathbf{x}(\tau) d\tau \quad \text{A.9}$$

$$\bar{\mathbf{x}} = \frac{1}{\rho} \int_0^\rho e^{A\tau} \mathbf{x}(0) + (e^{A\tau} - I)\mathbf{A}^{-1}\mathbf{b} d\tau \quad \text{A.10}$$

$$\bar{\mathbf{x}} = \frac{1}{\rho} \int_0^\rho e^{A\tau} \mathbf{x}(0) + e^{A\tau} \mathbf{A}^{-1}\mathbf{b} - \mathbf{A}^{-1}\mathbf{b} d\tau \quad \text{A.11}$$

$$\begin{aligned} \bar{\mathbf{x}} &= \frac{1}{\rho} \int_0^\rho e^{A\tau} \mathbf{x}(0) d\tau + \frac{1}{\rho} \int_0^\rho e^{A\tau} \mathbf{A}^{-1}\mathbf{b} d\tau \\ &\quad - \frac{1}{\rho} \int_0^\rho \mathbf{A}^{-1}\mathbf{b} d\tau \end{aligned} \quad \text{A.12}$$

$$\begin{aligned} \bar{\mathbf{x}} &= \frac{1}{\rho} \int_0^\rho e^{A\tau} d\tau \mathbf{x}(0) + \frac{1}{\rho} \int_0^\rho e^{A\tau} d\tau \mathbf{A}^{-1}\mathbf{b} \\ &\quad - \frac{1}{\rho} \int_0^\rho d\tau \mathbf{A}^{-1}\mathbf{b} \end{aligned} \quad \text{A.13}$$

$$\bar{\mathbf{x}} = \frac{1}{\rho} (e^{A\rho} - I)\mathbf{A}^{-1}\mathbf{x}(0) + \frac{1}{\rho} (e^{A\rho} - I)\mathbf{A}^{-1}\mathbf{A}^{-1}\mathbf{b} - \mathbf{A}^{-1}\mathbf{b} \quad \text{A.14}$$

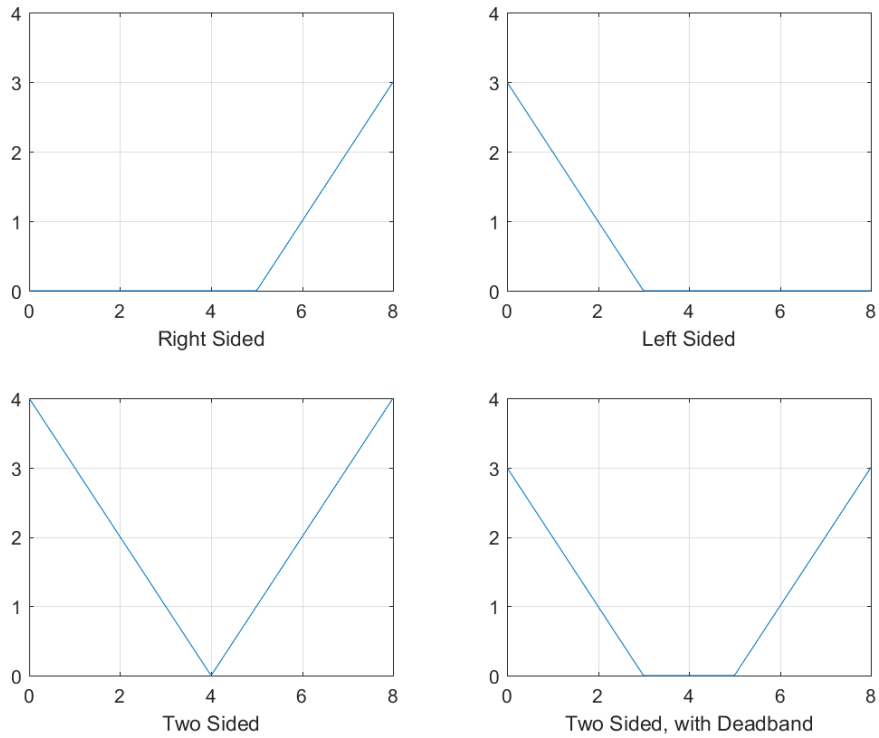
$$\bar{\mathbf{x}} = \frac{1}{\rho} (e^{A\rho} - I)\mathbf{A}^{-1}\mathbf{x}(0) + \left( \frac{1}{\rho} (e^{A\rho} - I)\mathbf{A}^{-1} - I \right) \mathbf{A}^{-1}\mathbf{b} \quad \text{A.15}$$

## APPENDIX B HVAC MODELING

### **Control Error Functions**

An HVAC system's primary purpose is to control the temperature and humidity of the spaces they serve. An error function that equals zero when a control objective is met and increases as the control error increases can be used to describe how much this occurs. Minimizing such an error function subject to the physical constraints of a system gives the steady state system states for that the control that's represented.

Figure 84 shows four basic control error functions: right sided, left sided, two-sided, and two-sided with a deadband. These four functions can be used in modeling different HVAC control scenarios. For example, the control error of a system controlling a zone to a fixed temperature set point can be modeled using a two-sided error function centered around the set point temperature. If a zone has a deadband the control error can be modeled using a two-sided error function with deadband. Zone relative humidity control error can be modeled with a right sided error function, where any relative humidity above the maximum allowable relative humidity has a positive error.



**Figure 84 – Basic Control Error Functions**

Table 38 shows the error functions of Figure 84 in two equivalent mathematical forms. The second column shows each error function as a discontinuous mathematical function. These functions can be used directly as the objective function of a model. However, nonlinear programming algorithms may not be able to calculate discontinuous functions, or they might have to be smoothed which can result in less accurate and reliable results. Another method of implementing a control error function is to integrate it into the constrained optimization itself. The third column shows each control error function as a linear program.

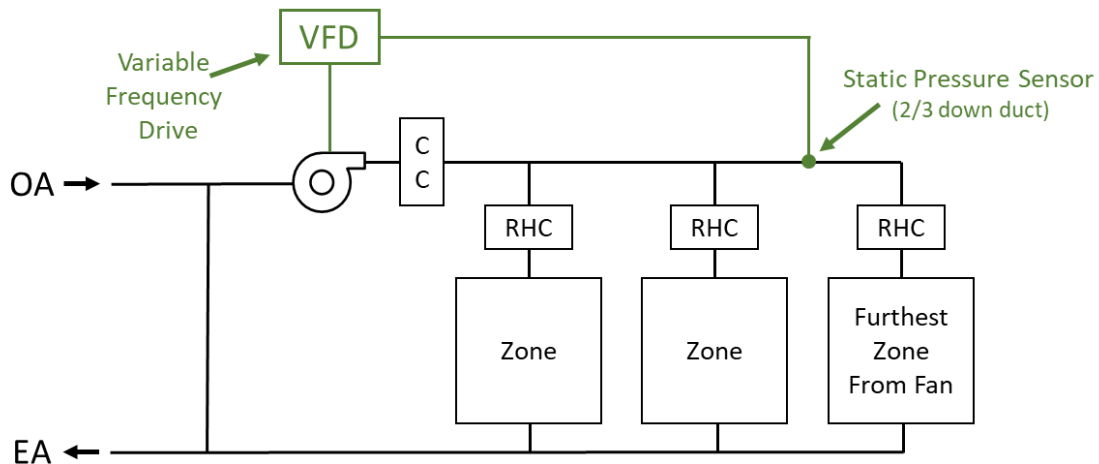
**Table 38 – Basic Error Function Equations**

<b>Error Function</b>	<b>As a Mathematical Function</b>	<b>As a Constrained Optimization</b>				
Right Sided	$y = \text{Max}(0, x - x_{max})$	<table border="1"> <tr> <td>minimize:</td> <td><math>y</math></td> </tr> <tr> <td>subject to:</td> <td><math>y \geq 0</math> <math>y \geq x - x_{max}</math></td> </tr> </table>	minimize:	$y$	subject to:	$y \geq 0$ $y \geq x - x_{max}$
minimize:	$y$					
subject to:	$y \geq 0$ $y \geq x - x_{max}$					
Left Sided	$y = \text{Max}(0, x_{min} - x)$	<table border="1"> <tr> <td>minimize:</td> <td><math>y</math></td> </tr> <tr> <td>subject to:</td> <td><math>y \geq 0</math> <math>y \geq x_{min} - x</math></td> </tr> </table>	minimize:	$y$	subject to:	$y \geq 0$ $y \geq x_{min} - x$
minimize:	$y$					
subject to:	$y \geq 0$ $y \geq x_{min} - x$					
2 Sided	$y = \text{Max}(x_c - x, x - x_c)$ $=  x_c - x $	<table border="1"> <tr> <td>minimize:</td> <td><math>y</math></td> </tr> <tr> <td>subject to:</td> <td><math>y \geq 0</math> <math>y \geq x_c - x</math> <math>y \geq x - x_c</math></td> </tr> </table>	minimize:	$y$	subject to:	$y \geq 0$ $y \geq x_c - x$ $y \geq x - x_c$
minimize:	$y$					
subject to:	$y \geq 0$ $y \geq x_c - x$ $y \geq x - x_c$					
2 Sided, with Deadband	$y = \text{Max}(0, x_{min} - x, x - x_{max})$	<table border="1"> <tr> <td>minimize:</td> <td><math>y</math></td> </tr> <tr> <td>subject to:</td> <td><math>y \geq 0</math> <math>y \geq x_{min} - x</math> <math>y \geq x - x_{max}</math></td> </tr> </table>	minimize:	$y$	subject to:	$y \geq 0$ $y \geq x_{min} - x$ $y \geq x - x_{max}$
minimize:	$y$					
subject to:	$y \geq 0$ $y \geq x_{min} - x$ $y \geq x - x_{max}$					



## Static Pressure Control

Many commercial HVAC systems control fan speeds to maintain a static pressure down a duct. In these systems, terminal boxes adjust dampers to maintain zone loads while the fan controller adjusts the fan's speed to maintain a static pressure set point down the duct. This can be seen in Figure 85. This section presents a simplified fan static pressure model based on the Darcy-Weisbach equation, fan models, and HVAC control schemes. It can be used in simplified building energy analysis models without requiring detailed fan and building pressure modeling such as the methods used in COMIS [22] or CONTAM [24].



**Figure 85 – Fan Static Pressure Control in a VAV System**

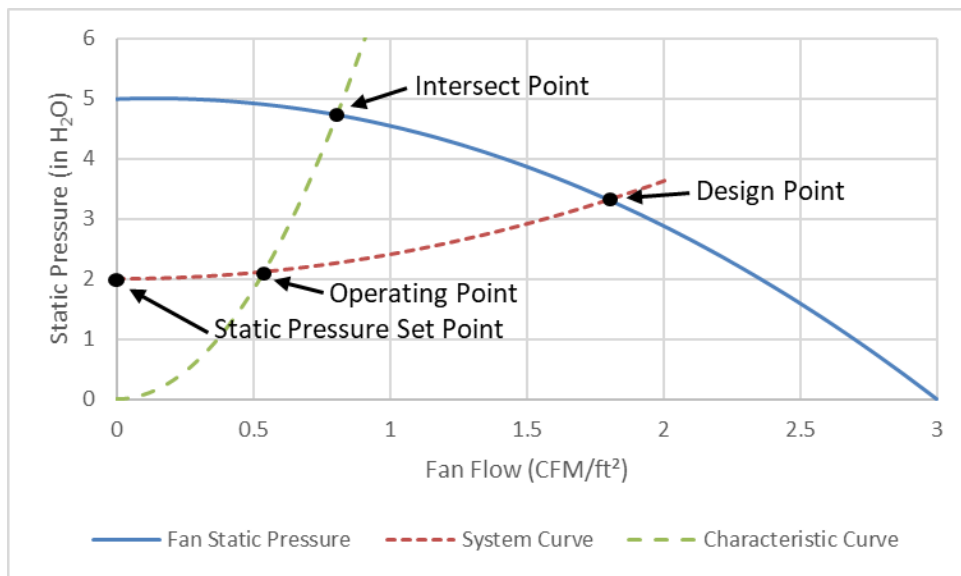
### *Fan Pressure Curves*

The fan pressure model uses three types of curves that relate air flow rate to total pressure: fan total pressure curves, pressure characteristic curves, and system curves. Figure 86 gives examples of these three curves. Fan total pressure curves give fan total pressure as a function of air flow rate at full speed. This function is a property of the fan used in a system. Pressure characteristic curves describe how the fan total pressure curve scales at different fan speeds based on the fan laws [72]. They take the form of equation B.1. Finally, system curves describe how fan total pressure in a system with

static pressure control changes for different air flow rates. System curves take the form of equation B.2 due to features of VAV systems with static pressure control, as explained below.

$$\frac{P_1}{P_2} = \left(\frac{\bar{V}_1}{\bar{V}_2}\right)^2 \quad B.1$$

$$P_{t,system}(V) = P_{sp} + \alpha_{sys}\bar{V}^2 \quad B.2$$

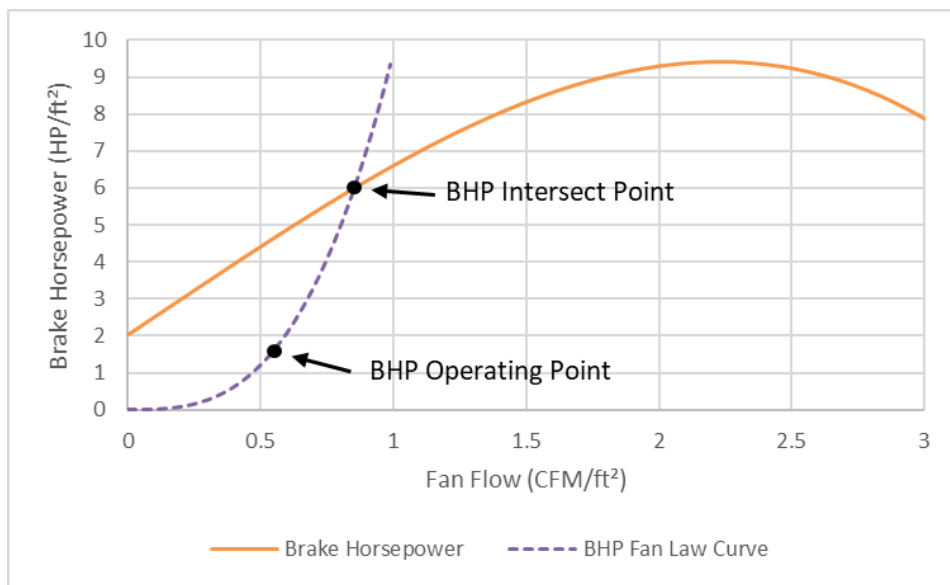


**Figure 86 – Fan Pressure Curves**

Maintaining a static pressure setpoint with zero flow isn't possible in a real system, but it would be if terminal box dampers could completely close and ducts had no leakage. Under these hypothetical conditions a system would still maintain the static pressure set point, so the system curve intersects the y axis at the static pressure set point. At the maximum possible flow all terminal box dampers would be fully open, and the system curve intersects the fan total pressure curve. If all terminal boxes have the same loads the dampers open and close together, which would give a single curve that connects these two points. It's assumed that a quadratic curve connects these two points

due to the way pressure changes with flow across ducts and dampers, which approximates the Darcy-Weisbach equation.

Modeling fan power makes use of two types of curves that relate air flow rate to brake horsepower, as seen in Figure 87. First, fan brake horsepower curves give fan power as a function of air flow rate at full speed. This function is a property of the fan used in a system. Next, fan power characteristic curves describe how the fan total pressure curve scales at different fan speeds. They take the form of equation B.3.



**Figure 87 – Fan Brake Horsepower Curves**

$$\frac{P_1}{P_2} = \left( \frac{Bhp_1}{Bhp_2} \right)^3 \quad B.3$$

*Fixed Static Pressure Set Point Model*

Table 39 gives the required inputs for the fan static pressure model based on the curves of the previous section. These inputs are the fan total pressure and brake horsepower curves at full speed, the maximum possible system flow at the static pressure set point, and the static pressure set point.

**Table 39 – Fixed Static Pressure Set Point Model Inputs**

Parameter	Symbol	Unit
Fan Total Pressure vs. Flow Curve at Full Speed	$P_{T,Fan}(\bar{V})$	<i>inches H<sub>2</sub>O</i>
Fan Brake Horsepower vs. Flow Curve at Full Speed	$\overline{Bhp}_{Fan}(\bar{V})$	$\frac{Hp}{ft^2}$
Maximum Air Flow Rate	$\bar{V}_{Max}$	$\frac{CFM}{ft^2}$
Static Pressure Set Point	$P_{SP}$	<i>inches H<sub>2</sub>O</i>
Current Air Flow Rate	$\bar{V}_{Curr}$	$\frac{CFM}{ft^2}$

**Table 40 – Fixed Static Pressure Set Point Model Outputs**

Parameter	Symbol	Unit
System Curve Quadratic Factor	$\alpha_{sys}$	$\frac{inches\ H_2O}{\left(\frac{CFM}{ft^2}\right)^2}$
Air Flow Rate at the Fan Curve/System Curve Intersect Point	$\bar{V}_{Intersect}$	$\frac{CFM}{ft^2}$
Current Fan Total Pressure	$P_{T,Curr}$	<i>inches H<sub>2</sub>O</i>
Fan Total Pressure the Fan Curve/System Curve Intersect Point	$P_{T,Intersect}$	<i>inches H<sub>2</sub>O</i>
Fan Brake Horsepower at the Fan Curve/System Curve Intersect Point	$\overline{Bhp}_{Intersect}$	$\frac{Hp}{ft^2}$
Current Fan Brake Horsepower	$\overline{Bhp}_{Curr}$	$\frac{Hp}{ft^2}$

Equations B.4–B.8 describe the curves of Figure 86 and Figure 87. Equations B.4–B.6 describe fan pressure calculations. Equation B.4 describes the intersection of the fan total pressure curve and the system curve, equation B.5 describes the intersection of the fan total pressure curve and the current characteristic pressure curve, and equation B.6 describes the current fan characteristic pressure curve. Equations B.7–B.8 describes fan brake horsepower calculations. Equation B.7 describes the intersection of the fan brake horsepower curve and the current fan characteristic brake horsepower curve, and equation B.8 describes the current fan characteristic brake horsepower curve.

$$P_{T,Fan}(\bar{V}_{Max}) = P_{SP} + \alpha_{Sys}\bar{V}_{Max}^2 \quad B.4$$

$$P_{T,Curr} = P_{SP} + \alpha_{Sys}\bar{V}_{Curr}^2 \quad B.5$$

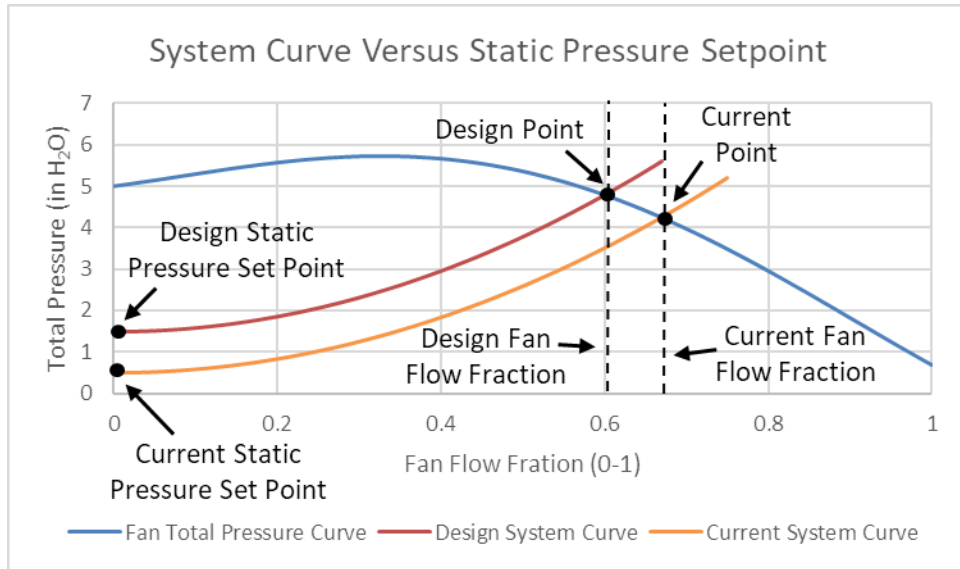
$$\frac{P_{T,Intersect}}{P_{T,Curr}} = \left(\frac{\bar{V}_{Intersect}}{\bar{V}_{Curr}}\right)^2 \quad B.6$$

$$\overline{Bhp}_{Intersect} = \overline{Bhp}_{Fan}(\bar{V}_{Intersect}) \quad B.7$$

$$\frac{\overline{Bhp}_{Intersect}}{\overline{Bhp}_{Curr}} = \left(\frac{\bar{V}_{Intersect}}{\bar{V}_{Curr}}\right)^3 \quad B.8$$

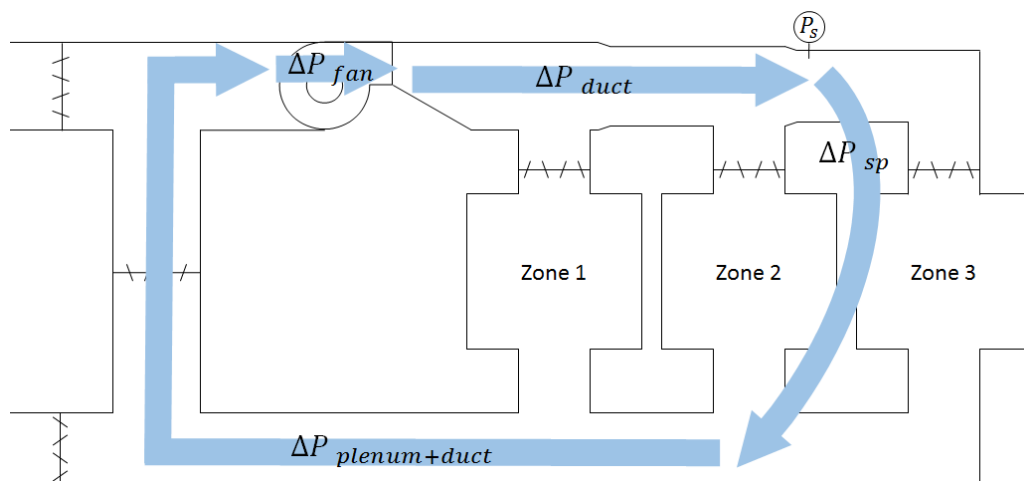
#### *Effect of Changing the Static Pressure Set Point*

In a system with static pressure control, different damper positions are required for different static pressure set points to achieve the same air flow rates. At a lower static pressure set point a higher overall flow is possible with dampers fully open. However, at zero flow the pressure drop on a fully closed system would still equal the static pressure set point. This variation defines how the system curve varies for different static pressure setpoints, as shown in Figure 88.



**Figure 88 – System Curve Changes versus Static Pressure Setpoint Changes**

Pressure changes on a single loop of a Single Duct Variable Air Volume (SDVAV) system are used to model how the maximum possible flow changes with the static pressure set point. Figure 89 shows this loop and breaks it down into 4 components: 1) the pressure rise across the fan, 2) the pressure drop down the ducts, 3) the static pressure drop across the static pressure sensor, and 4) the pressure drop down the plenum. The pressure drop is zero across the entire loop.



**Figure 89 – Pressure Changes in a System with Static Pressure Control**

Equations B.9–B.12 give models for the various pressure drop components. Equation B.9 holds that the total pressure drop around the loop is zero. Total pressure drop across the fan in the loop given by the fan total pressure curve. The pressure drop across ducts and the plenum is assumed to be proportional to the flow squared from the Darcy-Weisbach equation, as shown in equation B.10. During operation the system maintains the static pressure set point, so the drop across the sensor is equal to that, as shown in equation B.11. Equation B.12 gives the overall result that can be solved for the max flow  $V_{Max}$  as a function of the static pressure set point  $P_{SP}$  when the fan curve and a system coefficient  $\alpha_{Sys}$  calculated from nominal operating conditions are known.

$$P_{T,Fan} + P_{Duct} + P_{Sensor} + P_{Plenum} = 0 \quad B.9$$

$$P_{Duct} + P_{Plenum} = -\alpha_{Sys}V^2 \quad B.10$$

$$P_{Sensor} = -P_{SP} \quad B.11$$

$$P_{T,Fan}(V_{Max}) = P_{SP} + \alpha_{Sys}V_{Max}^2 \quad B.12$$

#### *Variable Static Pressure Set Point Model*

Table 41 gives the required inputs for a fan static pressure model that takes varying static pressures into account. This model uses the same fan curves, current volume flow rate, and current static pressure setpoints as the constant static pressure set point model. However, it uses the maximum possible flow and static pressure setpoint under known, nominal conditions as a reference point for calculating system curves under other static pressure set points.

**Table 41 – Variable Static Pressure Set Point Model Inputs**

<b>Parameter</b>	<b>Symbol</b>	<b>Unit</b>
Fan Total Pressure vs. Flow at Full Speed Curve	$P_{T,Fan}(\bar{V})$	<i>inches H<sub>2</sub>O</i>
Fan Brake Horsepower vs. Flow at Full Speed Curve	$\overline{Bhp}_{Fan}(\bar{V})$	$\frac{Hp}{ft^2}$
Maximum Air Flow Rate at the Design Static Pressure Set Point	$\bar{V}_{Max,Design}$	$\frac{CFM}{ft^2}$
Design Static Pressure Set Point	$P_{SP,Design}$	<i>inches H<sub>2</sub>O</i>
Current Static Pressure Set Point	$P_{SP,Curr}$	<i>inches H<sub>2</sub>O</i>
Current Air Flow Rate	$\bar{V}_{Curr}$	$\frac{CFM}{ft^2}$

**Table 42 – Variable Static Pressure Set Point Model Outputs**

<b>Parameter</b>	<b>Symbol</b>	<b>Unit</b>
System Curve Quadratic Factor	$\alpha_{sys}$	$\frac{inches\ H_2O}{\left(\frac{CFM}{ft^2}\right)^2}$
Maximum Flow at the Current Static Pressure Set Point	$\bar{V}_{Max,Curr}$	$\frac{CFM}{ft^2}$
Air Flow Rate at the Fan Curve/System Curve Intersect Point	$\bar{V}_{Intersect}$	$\frac{CFM}{ft^2}$
Current Fan Total Pressure	$P_{T,Curr}$	<i>inches H<sub>2</sub>O</i>
Fan Total Pressure the Fan Curve/System Curve Intersect Point	$P_{T,Intersect}$	<i>inches H<sub>2</sub>O</i>
Fan Brake Horsepower at the Fan Curve/System Curve Intersect Point	$\overline{Bhp}_{Intersect}$	$\frac{Hp}{ft^2}$
Current Fan Brake Horsepower	$\overline{Bhp}_{Curr}$	$\frac{Hp}{ft^2}$



Equations B.13–B.17 describe the fan pressure curves of Figure 88. Equation B.13 describes the intersection of the fan total pressure curve and the system curve at a reference static pressure set point. Equation B.14 describes intersection of the of the fan total pressure curve and the system curve at the current static pressure set point. Equation B.15 describes the intersection of the current system curve and the characteristic curve at the current flow rate and fan total pressure. Equation B.16 describes the intersection of the fan total pressure curve and the characteristic curve. Finally, equation B.17 describes the characteristic curve. Brake horsepower calculations of equations B.18 and B.19 are the same as in the constant static pressure set point equations.

$$P_{T,Fan}(\bar{V}_{Max,Nom}) = P_{SP,Nom} + \alpha_{Sys}\bar{V}_{Max,Nom}^2 \quad B.13$$

$$P_{T,Fan}(\bar{V}_{Max,Curr}) = P_{SP,Curr} + \alpha_{Sys}\bar{V}_{Max,Curr}^2 \quad B.14$$

$$P_{T,Curr} = P_{SP,Curr} + \alpha_{Sys}\bar{V}_{Curr}^2 \quad B.15$$

$$P_{T,Intersect} = P_{SP,Curr} + \alpha_{Sys}\bar{V}_{Intersect}^2 \quad B.16$$

$$\frac{P_{T,Intersect}}{P_{T,Curr}} = \left(\frac{\bar{V}_{Intersect}}{\bar{V}_{Curr}}\right)^2 \quad B.17$$

$$\overline{Bhp}_{Intersect} = \overline{Bhp}_{Fan}(\bar{V}_{Intersect}) \quad B.18$$

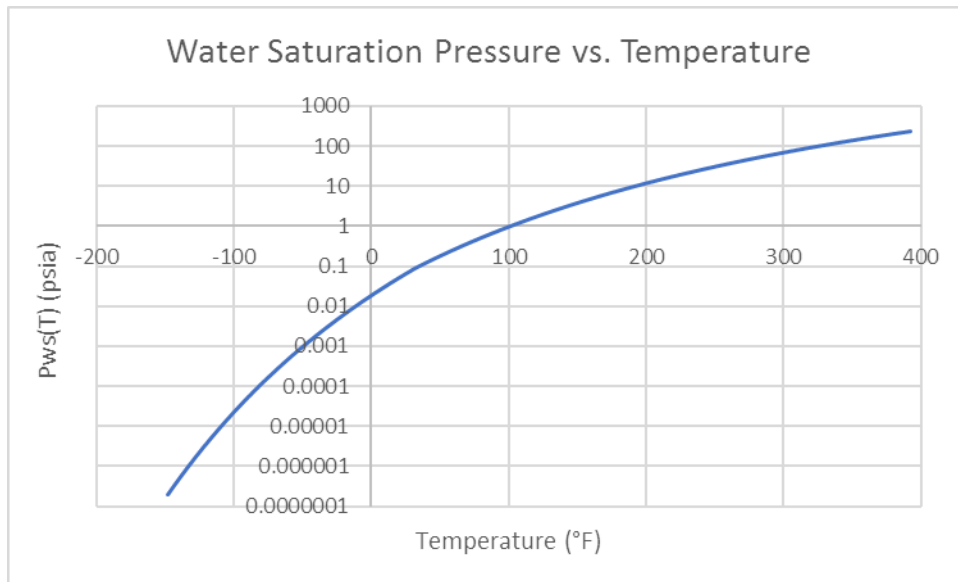
$$\frac{\overline{Bhp}_{Intersect}}{\overline{Bhp}_{Curr}} = \left(\frac{\bar{V}_{Intersect}}{\bar{V}_{Curr}}\right)^3 \quad B.19$$

## Psychrometrics

$$M_{ratio} = \frac{M_w}{M_{da}} = \frac{18.015268}{28.966} = 0.621945 \quad B.20$$

### Water Saturation Pressure

Figure 90 shows a plot of water saturation pressure in *psia* versus temperature in  $^{\circ}F$ . This equation is based off equations B.21 and B.22 where less than or equal to  $32^{\circ}F$  the function uses equation B.21 and above  $32^{\circ}F$  the function uses equation B.22. The variable  $T_R$  in these equations is the absolute temperature in Rankine. Table 43 shows the values of the coefficients used in these equations [72].



**Figure 90 – Water Saturation Pressure Equation from ASHRAE**

$$\ln(P_{ws}(T_R)) = \frac{C_1}{T_R} + C_2 + C_3 T_R + C_4 T_R^2 + C_5 T_R^3 + C_6 T_R^4 + C_7 \ln(T_R) \quad B.21$$

$$\ln(P_{ws}(T_R)) = \frac{C_8}{T_R} + C_9 + C_{10} T_R + C_{11} T_R^2 + C_{12} T_R^3 + C_{13} \ln(T_R) \quad B.22$$

**Table 43 – Coefficients for Saturation Water Pressure Calculations**

Coefficient	Value
$C_1$	-1.0214165E4
$C_2$	-4.8932428
$C_3$	-5.3765794E-3
$C_4$	1.9202377E-7
$C_5$	3.5575832E-10
$C_6$	-9.0344688E-14
$C_7$	4.1635019
$C_8$	-1.0440397E4
$C_9$	-1.1294650E1
$C_{10}$	-2.7022355E-2
$C_{11}$	1.2890360E-5
$C_{12}$	-2.4780681E-9
$C_{13}$	6.5459673

*Saturation Humidity Ratio*

$$W_{Sat} = \frac{M_{ratio}P_{ws}(T)}{P - P_{ws}(T)} \quad B.23$$

$$W_{Sat}P = (M_{ratio} + W_{Sat})P_{ws}(T) \quad B.24$$

*Relative Humidity*

$$Rh = \frac{100PW}{P_{ws}(T)(W + M_{ratio})} \quad B.25$$

**RC Networks for Wall Thermal Mass**

RC networks can be used to approximate one dimensional heat transfer [70]. Figure 92 shows an RC network in a “Tee” configuration, where each node of the network is evenly spaced. Table 44 lists parameters that are commonly used in modeling wall heat transfer. Equations B.26 and B.27 define R-Values and U-Values based on more fundamental parameters, equation B.28 defines thermal capacitance, equation B.29 defines a wall’s time constant, and equation B.30 defines a wall’s UA-Value divided by the floor area connected to the wall. Normalized UA-Values can be used in building energy calculations to help keep variables within known ranges.

**Table 44 – Parameters Used in Wall Thermal Modeling**

Parameter	Symbol	Unit
Wall Density	$\rho_{Wall}$	$\frac{lbm}{ft^3}$
Wall Thermal Conductivity	$k_{Wall}$	$\frac{Btu}{h \cdot ft \cdot ^\circ F}$
Wall Specific Heat	$c_{p,Wall}$	$\frac{Btu}{lbm \cdot ^\circ F}$
Wall Thickness	$l_{Wall}$	$ft$
Wall Surface Area	$A_{Wall}$	$ft^2$
Wall Volume	$V_{Wall}$	$ft^3$
Zone Area Connected to the Wall	$A_z$	$ft^2$
Wall R-Value	$R_{Wall}$	$\frac{h \cdot ft^2 \cdot ^\circ F}{Btu}$
Wall U-Value	$U_{Wall}$	$\frac{Btu}{h \cdot ft^2 \cdot ^\circ F}$
Wall Thermal Capacitance	$C_{Wall}$	$\frac{Btu}{ft^2 \cdot ^\circ F}$
Wall Time Constant	$\tau_{Wall}$	$h$
Wall UA-Value Normalized to a Zone Area	$\overline{UA}_{Wall}$	$\frac{Btu}{h \cdot ft^2 \cdot ^\circ F}$

$$R_{Wall} = \frac{l_{Wall}}{k_{Wall}} \quad B.26$$

$$U_{Wall} = \frac{1}{R_{Wall}} = \frac{k_{Wall}}{l_{Wall}} \quad B.27$$

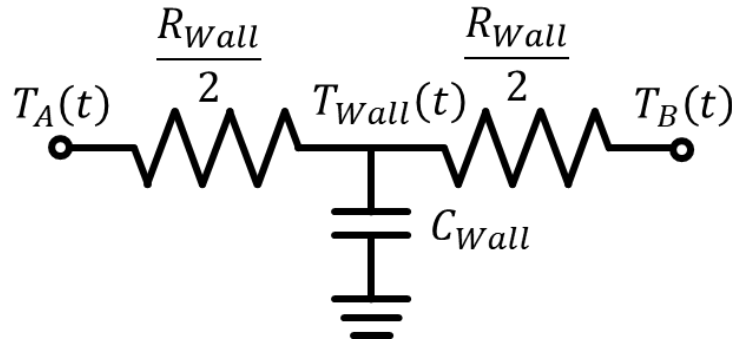
$$C_{Wall} = \rho_{Wall} c_{p,Wall} l_{Wall} \quad B.28$$

$$\tau_{Wall} = R_{Wall} C_{Wall} = \frac{\rho_{Wall} c_{p,Wall} l_{Wall}^2}{k_{Wall}} \quad B.29$$

$$\overline{UA}_{Wall} = \frac{A_{Wall}}{R_{Wall} A_z} = \frac{k_{Wall} A_{Wall}}{l_{Wall} A_z} \quad B.30$$

*Modeled with a Single Node*

Figure 91 shows an RC network modeled with a single thermal mass node  $T_{Wall}(t)$ . Equations B.31–B.33 show the derivation and simplification of the differential equation of equation B.33 that describes this system. Equation B.34 gives the heat transfer into side B from the wall node in  $\frac{Btu}{h}$ , and equation B.35 gives the heat transfer into side A from the wall node in  $\frac{Btu}{h}$ .



**Figure 91 – RC Network with a Single Node**

$$C_{Wall}\dot{T}_{Wall}(t) + \frac{2(T_{Wall}(t) - T_A(t))}{R_{Wall}} + \frac{2(T_{Wall}(t) - T_B(t))}{R_{Wall}} = 0 \quad B.31$$

$$R_{Wall}C_{Wall}\dot{T}_{Wall}(t) = -2T_{Wall}(t) + 2T_A(t) - 2T_{Wall} + 2T_B(t) \quad B.32$$

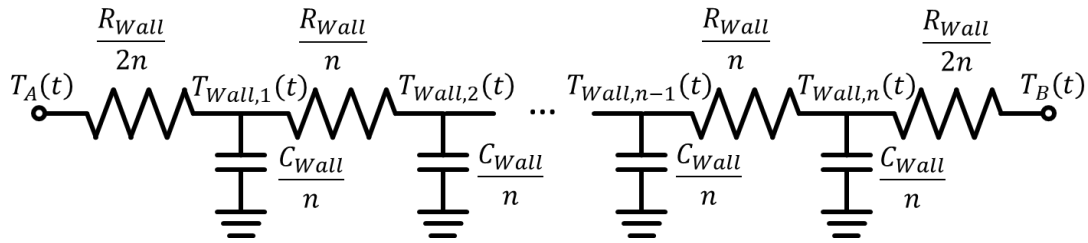
$$\tau_{Wall}\dot{T}_{Wall}(t) = -4T_{Wall}(t) + 2T_A(t) + 2T_B(t) \quad B.33$$

$$Q_{Wall \rightarrow B}(t) = \frac{2A_{Wall}}{R_{Wall}}(T_{Wall}(t) - T_B(t)) \quad B.34$$

$$Q_{Wall \rightarrow A}(t) = \frac{2}{R_{Wall}}(T_{Wall}(t) - T_A(t)) \quad B.35$$

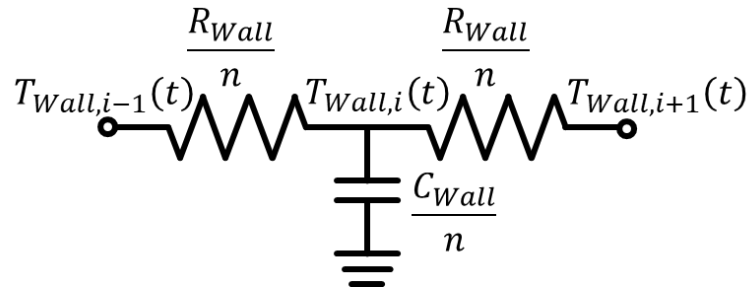
*Modeled with Multiple Nodes*

More detailed wall thermal mass models than the single node model can be constructed by dividing a wall into  $n$  sections. Each section represents a single node and is represented by an equally divided “Tee” network. Models with multiple nodes can be more accurate by modeling the temperature distribution within a wall’s depth.



**Figure 92 – An Equally Divided RC Network with Multiple Nodes**

Figure 93 shows an RC network for an interior node of the model in Figure 92. Equations B.36–B.38 show the derivation and simplification of the differential equation that describes the change in the temperature of  $T_{Wall,i}$ . Equation B.39 gives the heat transfer from wall node  $i$  to wall node  $i + 1$  in  $\frac{Btu}{h}$ .



**Figure 93 – RC Network for an Interior Node**

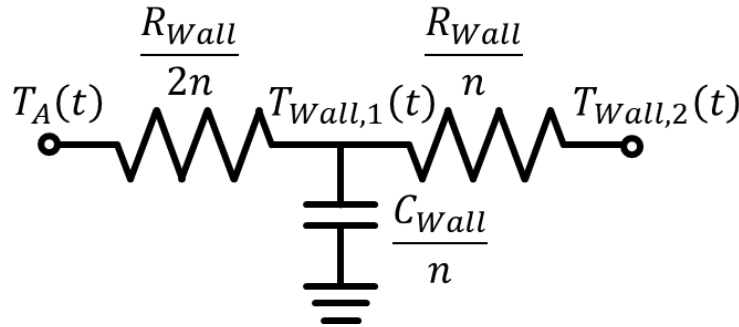
$$\frac{C_{Wall}}{n} \dot{T}_{Wall,i}(t) + \frac{n(T_{Wall,i}(t) - T_{Wall,i-1}(t))}{R_{Wall}} + \frac{n(T_{Wall,i}(t) - T_{Wall,i+1}(t))}{R_{Wall}} = 0 \quad B.36$$

$$\frac{R_{Wall} C_{Wall}}{n^2} \dot{T}_{Wall,i}(t) = -T_{Wall,i}(t) + T_{Wall,i-1}(t) - T_{Wall,i}(t) + T_{Wall,i+1}(t) \quad B.37$$

$$\frac{\tau_{Wall}}{n^2} \dot{T}_{Wall,i}(t) = -2T_{Wall,i}(t) + T_{Wall,i-1}(t) + T_{Wall,i+1}(t) \quad B.38$$

$$Q_{Wall,i \rightarrow Wall,i+1}(t) = \frac{nA_{Wall}}{R_{Wall}} (T_{Wall,i}(t) - T_{Wall,i+1}(t)) \quad B.39$$

Figure 94 shows an RC network for an exterior node of the model in Figure 92. Equations B.40–B.42 show the derivation and simplification of the differential equation that describes the change in the temperature of  $T_{Wall,1}$ . The equation for  $T_{Wall,n}$  can be derived in a similar way. Equation B.43 gives the heat transfer from wall node 1 to the exterior node A in  $\frac{Btu}{h}$ , and equation B.44 gives the heat transfer from wall node 1 to wall node 2, which has twice the resistance, in  $\frac{Btu}{h}$ . A matrix form of a “Tee” RC network in general with  $n$  nodes is given in equation B.45.



**Figure 94 – RC Network for an Exterior Node**

$$\frac{C_{Wall}}{n} \dot{T}_{Wall,1}(t) + \frac{2n(T_{Wall,1}(t) - T_A(t))}{R_{Wall}} + \frac{n(T_{Wall,1}(t) - T_{Wall,2}(t))}{R_{Wall}} \quad B.40$$

$$\frac{R_{Wall}C_{Wall}}{n^2} \dot{T}_{Wall,1}(t) = -2T_{Wall,1}(t) + 2T_A(t) - T_{Wall,1}(t) + T_{Wall,2}(t) \quad B.41$$

$$\frac{\tau_{Wall}}{n^2} \dot{T}_{Wall,1}(t) = -3T_{Wall,1}(t) + 2T_A(t) + T_{Wall,2}(t) \quad B.42$$

$$Q_{Wall,i \rightarrow A}(t) = \frac{2nA_{Wall}}{R_{Wall}} (T_{Wall,1}(t) - T_A(t)) \quad B.43$$

$$Q_{Wall,1 \rightarrow 2}(t) = \frac{nA_{Wall}}{R_{Wall}} (T_{Wall,1}(t) - T_{Wall,2}(t)) \quad B.44$$

$$\frac{\tau_{Wall}}{n^2} \begin{bmatrix} \dot{T}_{Wall,1}(t) \\ \dot{T}_{Wall,2}(t) \\ \dot{T}_{Wall,3}(t) \\ \vdots \\ \dot{T}_{Wall,n-2}(t) \\ \dot{T}_{Wall,n-1}(t) \\ \dot{T}_{Wall,n}(t) \end{bmatrix} = \begin{bmatrix} -3 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & -2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -2 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & -3 \end{bmatrix} \begin{bmatrix} T_{Wall,1}(t) \\ T_{Wall,2}(t) \\ T_{Wall,3}(t) \\ \vdots \\ T_{Wall,n-2}(t) \\ T_{Wall,n-1}(t) \\ T_{Wall,n}(t) \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} T_A(t) \\ T_B(t) \end{bmatrix} \quad B.45$$



### Infiltration and Exfiltration Calculations

Equation B.46 shows how air changes per hour can be converted into a flow rate for zones where infiltration and exfiltration are the same. Table 45 lists the variables used in equation B.46.

**Table 45 – Infiltration/Exfiltration Variables**

Parameter	Symbol	Unit
Zone Air Changes per Hour	$ACH$	$\frac{\text{Air Changes}}{h}$
Zone Air Volume	$V_{Zone}$	$ft^3$
Zone Floor Area	$A_{Zone}$	$ft^2$
Infiltration/Exfiltration Flow Rate	$\bar{V}_{Infiltration}$	$\frac{CFM}{ft^2}$

$$\bar{V}_{Infiltration} = \frac{ACH \cdot V_{Zone}}{60 \cdot A_{Zone}} \quad B.46$$

APPENDIX C  
CONSTRAINED OPTIMIZATION REFERENCE

**Nonlinear Programming Reformulations**

Nonlinear programming solvers may not be able to handle certain types of functions, such as functions with discontinuities. Reformulating problems in these cases might result in a solvable model. Table 46 gives reformulations that may be useful for modeling HVAC systems with absolute values.

**Table 46 – Nonlinear Programming Reformulations**

Situation	Original Form	Reformulated Form
Absolute value in constraints	$ f(x)  \leq z$	$f(x) \leq z$ $-f(x) \leq z$
Absolute value in the objective	$minimize  f(x)  + g(y)$	Introduce a new variable $z$ , substitute it for $x$ in the objective. $minimize  f(z)  + g(y)$ Add 2 new constraints $x \leq z$ $-x \leq z$

**Discontinuous Function Approximations**

Nonlinear programming algorithms such as the one used in IPOPT can fail when objective functions or equality constraints are not twice continuously differentiable [41]. However, discontinuous functions such as min, max, and absolute values can appear in HVAC modeling in situations such as cooling coils, heating coils, and optimization objectives. Smoothing these functions using approximations around discontinuous points can help make models run more robustly.

Approximations of the functions  $|x|$  or  $Max(x, 0)$  can be used to smooth discontinuous functions. Equations C.1–C.4 can be used to approximate  $Min(x, y)$ ,  $Max(x, y)$ , and  $|x|$  from either of these two approximations. The prototype solver Beryl contains user options to automatically smooth discontinuous functions based on these two functions.

$$Max(x, y) = y + Max(x - y, 0) \quad C.1$$

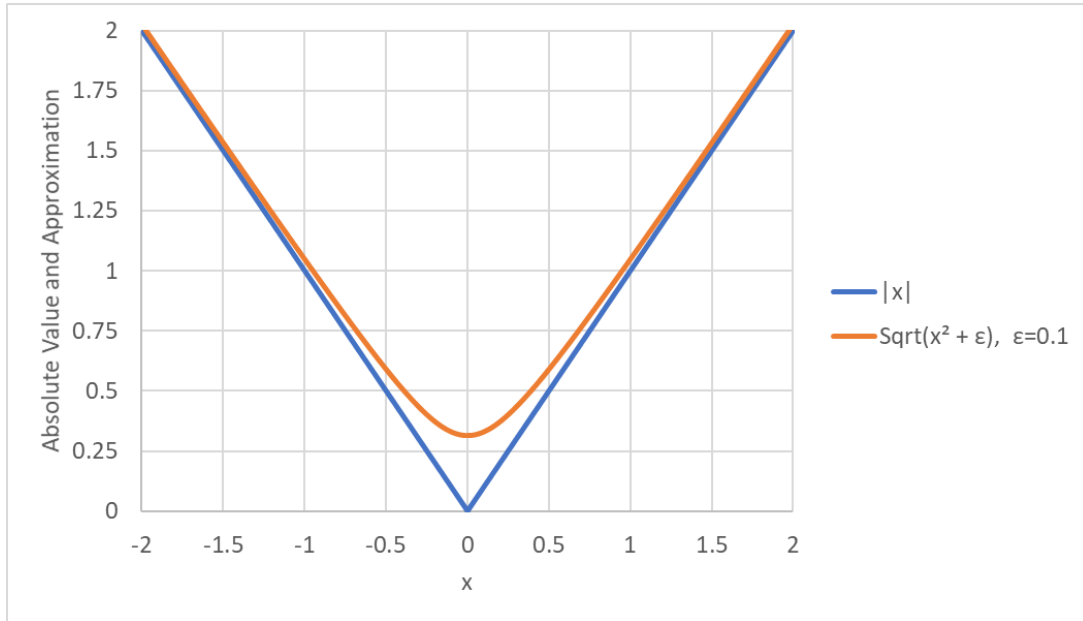
$$Min(x, y) = x - Max(x - y, 0) \quad C.2$$

$$Max(x, 0) = \frac{1}{2}(x + |x|) \quad C.3$$

$$|x| = 2 \cdot Max(x, 0) - x \quad C.4$$

Equation C.5 shows a smooth approximation for an absolute value that uses a square root and a positive parameter  $\varepsilon$  to dampen the discontinuity. Figure 95 compares this approximation with an absolute value and shows that its maximum error occurs at  $x = 0$  where the error equals  $\sqrt{\varepsilon}$ .

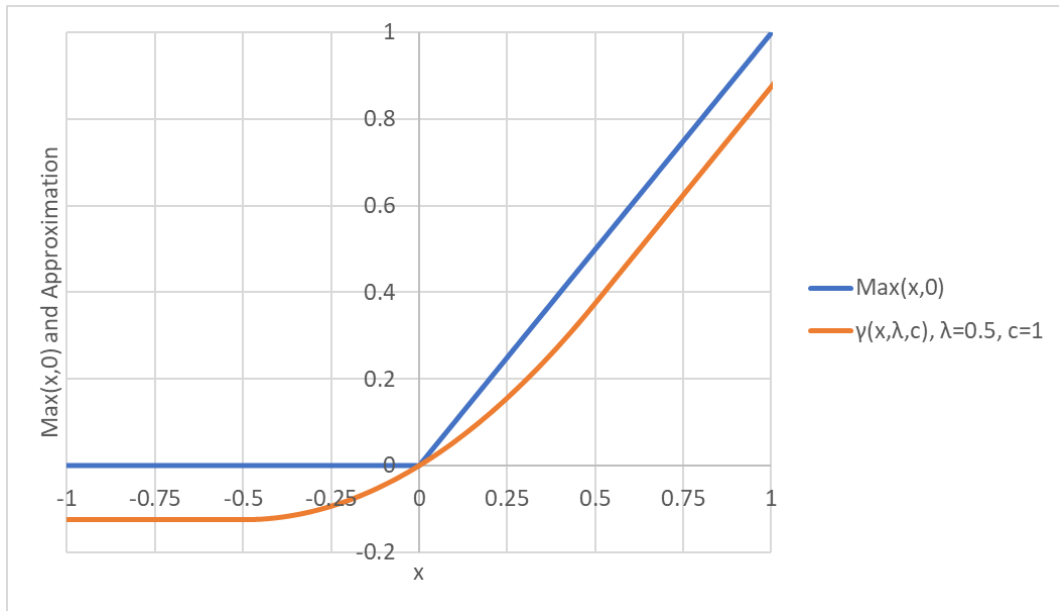
$$|x| \approx \sqrt{x^2 + \varepsilon} \quad C.5$$



**Figure 95 – Continuous Approximation of an Absolute Value**

Equation C.6 shows a piecewise continuous approximation for  $Max(x, 0)$  developed by Bertsekas for optimizing nondifferentiable problems [80]. This function depends on two scaling parameters,  $\lambda$  and  $c$ . The first parameter  $\lambda \in [0, 1]$  controls which side the function is tight on, while the second parameter  $c \in (0, \infty)$  controls the tightness of the fit. Figure 96 shows  $Max(x, 0)$  and this approximation when  $\lambda = 0.5$  to distribute errors evenly on both sides and  $c = 1$ .

$$Max(x, 0) \approx \gamma(x, \lambda, c) = \begin{cases} x - \frac{(1 - \lambda)^2}{2c} & \text{if } \frac{1 - \lambda}{c} \leq x \\ \lambda x + \frac{c}{2} x^2 & \text{if } -\frac{\lambda}{c} \leq x \leq \frac{1 - \lambda}{c} \\ -\frac{\lambda^2}{2c} & \text{if } x \leq -\frac{\lambda}{c} \end{cases} \quad C.6$$



**Figure 96 – Continuous Approximation of  $\text{Max}(x, 0)$  by Bertsekas**

## APPENDIX D

### NONLINEAR PROGRAMMING SENSITIVITY ANALYSIS DERIVATION

Building energy simulations consist of several time steps, where parameters such as outside air temperature and loads vary over time. To solve these problems using constrained optimization a sequence of problems must be solved. This is known as a parametric nonlinear program. Figure 97 shows the box-constrained form of a parametric nonlinear programming problem used by IPNLP, the constrained optimization solver developed for Beryl.

minimize:	$f(\mathbf{x}, \mathbf{y})$
subject to:	$\mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0}$ $\mathbf{a}(\mathbf{y}) \leq \mathbf{x} \leq \mathbf{b}(\mathbf{y})$

**Figure 97 – A Parametric Nonlinear Programming Problem**

Sensitivity analysis uses the KKT matrix factorizations performed in a previous simulation to estimate the solution for new parameters. Table 47 gives descriptions and dimensions of some variables used in the derivation of the equations needed to perform sensitivity analysis calculations.

**Table 47 – Parameters Used in Solving Parametric Nonlinear Programming Problem**

Variable	Description
$\mathbf{x}$	Primal variables
$\boldsymbol{\lambda}$	Equality constraint dual variables
$\mathbf{z}_L$	Lower bound dual variables
$\mathbf{z}_U$	Upper bound dual variables
$\mathbf{y}$	Parameters that change on each time step

The matrices  $\mathbf{E}_L$  and  $\mathbf{E}_U$  are used to add or remove rows in vectors or matrices to change their number of rows between the number of lower or upper bounds and the full length of the  $\mathbf{x}$  vector, as seen in equations D.4–D.5. They’re made up of column vectors for each upper or lower bound with a value of 1 on the bound index. For instance, if  $\mathbf{x}$  had a length of 4 and lower bounds existed on indices 1 and 4,  $\mathbf{E}_L$ ,  $\mathbf{z}_L$ , and  $\mathbf{x}$  would have the values and structure seen in equations D.1–D.3. Multiplying  $\mathbf{E}_L$  by  $\mathbf{z}_L$  results in a vector with the length of  $\mathbf{x}$  with  $\mathbf{z}_L$ ’s indices placed in the indices of the lower bounds, as seen in equation D.4. Multiplying the transpose of  $\mathbf{E}_L$  by  $\mathbf{x}$  results in a vector with the  $\mathbf{x}$  values of the used lower bounds, as seen in equation D.5.  $\mathbf{z}_L$  and  $\mathbf{z}_U$  have the same structure as  $\mathbf{E}_L$  and  $\mathbf{E}_U$  except they use the dual vector parameters instead of 1’s, as seen in equation D.6.

$$\mathbf{E}_L = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad D.1$$

$$\mathbf{z}_L = \begin{bmatrix} z_{L1} \\ z_{L4} \end{bmatrix} \quad D.2$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad D.3$$

$$\mathbf{E}_L \mathbf{z}_L = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{L1} \\ z_{L4} \end{bmatrix} = \begin{bmatrix} z_{L1} \\ 0 \\ 0 \\ z_{L4} \end{bmatrix} \quad D.4$$

$$\mathbf{E}_L^T \mathbf{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_4 \end{bmatrix} \quad D.5$$

$$\mathbf{z}_L = \begin{bmatrix} z_{L1} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & z_{L4} \end{bmatrix} \quad D.6$$

Functions for the distance between  $\mathbf{x}$  and its used lower and upper bounds are given in equations D.7 and D.8. Vectors given in capital letters are diagonalized versions of functions, as in equations D.9 and D.10.

$$\Delta \mathbf{x}_L(\mathbf{y}) = \mathbf{E}_L^T \mathbf{x} - \mathbf{a}(\mathbf{y}) \quad D.7$$

$$\Delta \mathbf{x}_U(\mathbf{y}) = \mathbf{b}(\mathbf{y}) - \mathbf{E}_U^T \mathbf{x} \quad D.8$$

$$\Delta \mathbf{X}_L(\mathbf{y}) = \text{diag}(\Delta \mathbf{x}_L(\mathbf{y})) \quad D.9$$

$$\Delta \mathbf{X}_U(\mathbf{y}) = \text{diag}(\Delta \mathbf{x}_U(\mathbf{y})) \quad D.10$$

The Lagrangian for the constrained optimization problem of Figure 97 is given in equation D.11, and its gradient in terms of  $\mathbf{x}$ , Hessian in terms of  $\mathbf{x}$ , and gradient in terms of  $\mathbf{x}$  and  $\mathbf{y}$  are given in equations D.12–D.14 for later use.

$$L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}_L, \mathbf{z}_U, \mathbf{y}) = \mathbf{f}(\mathbf{x}, \mathbf{y}) + \mathbf{h}(\mathbf{x}, \mathbf{y})\boldsymbol{\lambda} - \Delta \mathbf{x}_L(\mathbf{y})^T \mathbf{z}_L - \Delta \mathbf{x}_U(\mathbf{y})^T \mathbf{z}_U \quad D.11$$

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}_L, \mathbf{z}_U, \mathbf{y}) = \nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x}, \mathbf{y})\boldsymbol{\lambda} - \mathbf{E}_L \mathbf{z}_L + \mathbf{E}_U \mathbf{z}_U \quad D.12$$

$$\nabla_{\mathbf{x}}^2 L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}_L, \mathbf{z}_U, \mathbf{y}) = \nabla_{\mathbf{x}}^2 \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_{\mathbf{x}}^2 \mathbf{h}(\mathbf{x}, \mathbf{y})\boldsymbol{\lambda} \quad D.13$$

$$\nabla_{\mathbf{x}\mathbf{y}} L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}_L, \mathbf{z}_U, \mathbf{y}) = \nabla_{\mathbf{x}\mathbf{y}} \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_{\mathbf{x}\mathbf{y}} \mathbf{h}(\mathbf{x}, \mathbf{y})\boldsymbol{\lambda} \quad D.14$$

The Karush-Kuhn-Tucker conditions of the problem are given in equations D.13–D.18. These conditions for the solution of Figure 97 can be translated into equation D.19 using the implicit function theorem [85]. This expresses the change in  $\mathbf{x}$ ,  $\boldsymbol{\lambda}$ ,  $\mathbf{z}_L$ , and  $\mathbf{z}_U$  required to maintain the solution to the KKT equations after a small change in  $\mathbf{y}$  parameters. Matrix row operations are then used to translate equation D.19 into equation D.22. The block structure of equation D.22 can then be decomposed into equations D.23–D.25. The matrix on the left-hand side of D.23 is the solution used in IPNLP's algorithm. Its factorization can be reused to estimate new values for  $\mathbf{x}$ ,  $\boldsymbol{\lambda}$ ,  $\mathbf{z}_L$ , and  $\mathbf{z}_U$  after a parameter change.



$$\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}_L, \mathbf{z}_U, \mathbf{y}) = \nabla_x \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} - \mathbf{E}_L \mathbf{z}_L + \mathbf{E}_U \mathbf{z}_U = \mathbf{0} \quad D.15$$

$$\mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0} \quad D.16$$

$$\Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L = \mathbf{0} \quad D.17$$

$$\Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U = \mathbf{0} \quad D.18$$

$$\begin{aligned} & \begin{bmatrix} \nabla_x^2 \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_x^2 \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} & \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & -\mathbf{E}_L & \mathbf{E}_U \\ \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{z}_L^T & \mathbf{0} & \Delta \mathbf{X}_L(\mathbf{y}) & \mathbf{0} \\ -\mathbf{z}_U^T & \mathbf{0} & \mathbf{0} & \Delta \mathbf{X}_U(\mathbf{y}) \end{bmatrix} \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \boldsymbol{\lambda} \\ \nabla_y \mathbf{z}_L \\ \nabla_y \mathbf{z}_U \end{bmatrix} \\ &= - \begin{bmatrix} \nabla_{xy} \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_{xy} \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} \\ \nabla_y \mathbf{h}(\mathbf{x}, \mathbf{y}) \\ \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L \\ \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U \end{bmatrix} \quad D.19 \end{aligned}$$

$$\begin{aligned} & \begin{bmatrix} \nabla_x^2 \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_x^2 \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} & \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & -\mathbf{E}_L & \mathbf{E}_U \\ \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \Delta \mathbf{X}_L(\mathbf{y})^{-1} \mathbf{z}_L^T & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ -\Delta \mathbf{X}_U(\mathbf{y})^{-1} \mathbf{z}_U^T & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \boldsymbol{\lambda} \\ \nabla_y \mathbf{z}_L \\ \nabla_y \mathbf{z}_U \end{bmatrix} \\ &= - \begin{bmatrix} \nabla_{xy} \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_{xy} \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} \\ \nabla_y \mathbf{h}(\mathbf{x}, \mathbf{y}) \\ \Delta \mathbf{X}_L(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L \\ \Delta \mathbf{X}_U(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U \end{bmatrix} \quad D.20 \end{aligned}$$

$$\begin{aligned} & \begin{bmatrix} \nabla_x^2 \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_x^2 \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} + \mathbf{E}_L \Delta \mathbf{X}_L(\mathbf{y})^{-1} \mathbf{z}_L^T & \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & \mathbf{0} & \mathbf{E}_U \\ \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \Delta \mathbf{X}_L(\mathbf{y})^{-1} \mathbf{z}_L^T & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ -\Delta \mathbf{X}_U(\mathbf{y})^{-1} \mathbf{z}_U^T & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \boldsymbol{\lambda} \\ \nabla_y \mathbf{z}_L \\ \nabla_y \mathbf{z}_U \end{bmatrix} \\ &= - \begin{bmatrix} \nabla_{xy} \mathbf{f}(\mathbf{x}, \mathbf{y}) + \nabla_{xy} \mathbf{h}(\mathbf{x}, \mathbf{y}) \boldsymbol{\lambda} + \mathbf{E}_L \Delta \mathbf{X}_L(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L \\ \nabla_y \mathbf{h}(\mathbf{x}, \mathbf{y}) \\ \Delta \mathbf{X}_L(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L \\ \Delta \mathbf{X}_U(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U \end{bmatrix} \quad D.21 \end{aligned}$$

$$\begin{aligned}
& \begin{bmatrix} \left( \begin{array}{c} \nabla_x^2 f(\mathbf{x}, \mathbf{y}) + \nabla_x^2 \mathbf{h}(\mathbf{x}, \mathbf{y}) \lambda \\ + \mathbf{E}_L \Delta \mathbf{X}_L(\mathbf{y})^{-1} \mathbf{Z}_L^T + \mathbf{E}_U \Delta \mathbf{X}_U(\mathbf{y})^{-1} \mathbf{Z}_U^T \end{array} \right) & \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & 0 & 0 \\ \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & 0 & 0 & 0 \\ \Delta \mathbf{X}_L(\mathbf{y})^{-1} \mathbf{Z}_L^T & 0 & \mathbf{I} & 0 \\ -\Delta \mathbf{X}_U(\mathbf{y})^{-1} \mathbf{Z}_U^T & 0 & 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \lambda \\ \nabla_y \mathbf{z}_L \\ \nabla_y \mathbf{z}_U \end{bmatrix} \\
& = - \begin{bmatrix} \left( \begin{array}{c} \nabla_{xy} f(\mathbf{x}, \mathbf{y}) + \nabla_{xy} \mathbf{h}(\mathbf{x}, \mathbf{y}) \lambda \\ + \mathbf{E}_L \Delta \mathbf{X}_L(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L - \mathbf{E}_U \Delta \mathbf{X}_U(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U \end{array} \right) \\ \nabla_y \mathbf{h}(\mathbf{x}, \mathbf{y}) \\ \Delta \mathbf{X}_L(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L \\ \Delta \mathbf{X}_U(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U \end{bmatrix} \quad D.22
\end{aligned}$$

Equations D.23–D.25 are used to calculate the change in  $\mathbf{x}$ ,  $\lambda$ ,  $\mathbf{z}_L$ , and  $\mathbf{z}_U$  required to maintain the solutions to the KKT conditions with an infinitesimal change in  $\mathbf{y}$  parameters from a successfully completed simulation. Equation D.23 is calculated first using the factored matrix of the previous time step. Next, the solution to  $\nabla_y \mathbf{x}$  is used in equations D.24 and D.25. The amount to adjust  $\mathbf{x}$ ,  $\lambda$ ,  $\mathbf{z}_L$ , and  $\mathbf{z}_U$  for a new solution can be calculated from equation D.26, where  $\mathbf{y}_{next}$  is the parameter value vector for the next time step. Making this adjustment provides the new solution estimate after an input parameter change.

$$\begin{aligned}
& \begin{bmatrix} \nabla_x^2 f(\mathbf{x}, \mathbf{y}) + \nabla_x^2 \mathbf{h}(\mathbf{x}, \mathbf{y}) \lambda + \mathbf{E}_L \Delta \mathbf{X}_L(\mathbf{y})^{-1} \mathbf{Z}_L^T + \mathbf{E}_U \Delta \mathbf{X}_U(\mathbf{y})^{-1} \mathbf{Z}_U^T & \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) \\ \nabla_x \mathbf{h}(\mathbf{x}, \mathbf{y}) & 0 \end{bmatrix} \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \lambda \end{bmatrix} \\
& = - \begin{bmatrix} \left( \begin{array}{c} \nabla_{xy} f(\mathbf{x}, \mathbf{y}) + \nabla_{xy} \mathbf{h}(\mathbf{x}, \mathbf{y}) \lambda \\ + \mathbf{E}_L \Delta \mathbf{X}_L(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L - \mathbf{E}_U \Delta \mathbf{X}_U(\mathbf{y})^{-1} \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U \end{array} \right) \\ \nabla_y \mathbf{h}(\mathbf{x}, \mathbf{y}) \end{bmatrix} \quad D.23
\end{aligned}$$

$$\nabla_y \mathbf{z}_L = -\Delta \mathbf{X}_L(\mathbf{y})^{-1} (\mathbf{Z}_L^T \nabla_y \mathbf{x} + \nabla_y \Delta \mathbf{X}_L(\mathbf{y}) \mathbf{z}_L) \quad D.24$$

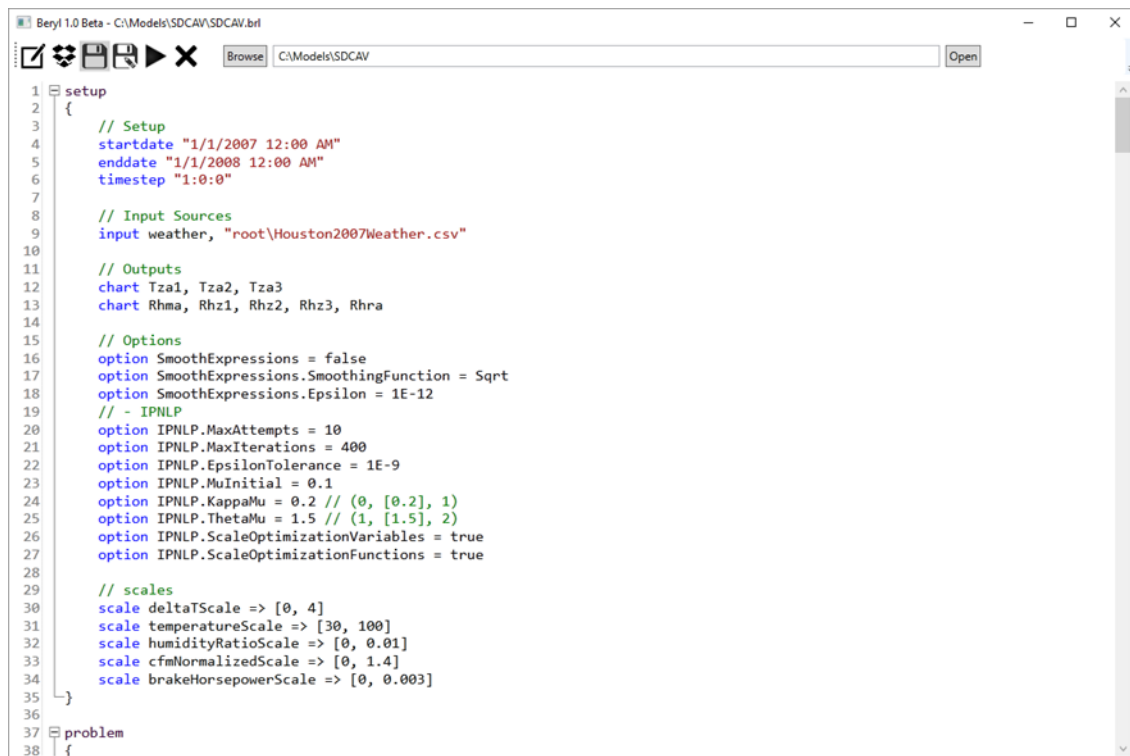
$$\nabla_y \mathbf{z}_U = \Delta \mathbf{X}_U(\mathbf{y})^{-1} (\mathbf{Z}_U^T \nabla_y \mathbf{x} - \nabla_y \Delta \mathbf{X}_U(\mathbf{y}) \mathbf{z}_U) \quad D.25$$

$$\begin{bmatrix} \Delta \mathbf{x} \\ \Delta \lambda \\ \Delta \mathbf{z}_L \\ \Delta \mathbf{z}_U \end{bmatrix} = \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \lambda \\ \nabla_y \mathbf{z}_L \\ \nabla_y \mathbf{z}_U \end{bmatrix} \Delta \mathbf{y} = \begin{bmatrix} \nabla_y \mathbf{x} \\ \nabla_y \lambda \\ \nabla_y \mathbf{z}_L \\ \nabla_y \mathbf{z}_U \end{bmatrix} (\mathbf{y}_{next} - \mathbf{y}) \quad D.26$$

## APPENDIX E BERYL USERS GUIDE

### User Interface

Beryl's main user interface is shown in Figure 98. Icons in the upper left allow users to create new models, open an existing model, save models, and run a simulation. The browse button allows a root directory to be selected for where weather files come from. The selected root directory appears in the text box by the browse button, and the open button opens the root directory.

The screenshot shows a window titled "Beryl 1.0 Beta - C:\Models\SDCAV\SDCAV.brl". The window has a toolbar with icons for file operations and a "Browse" button next to a text box containing "C:\Models\SDCAV". Below the toolbar is a code editor displaying the following text:

```
1 setup
2 {
3     // Setup
4     startdate "1/1/2007 12:00 AM"
5     enddate "1/1/2008 12:00 AM"
6     timestep "1:0:0"
7
8     // Input Sources
9     input weather, "root\Houston2007Weather.csv"
10
11    // Outputs
12    chart Tza1, Tza2, Tza3
13    chart Rhma, Rhz1, Rhz2, Rhz3, Rhra
14
15    // Options
16    option SmoothExpressions = false
17    option SmoothExpressions.SmoothingFunction = Sqrt
18    option SmoothExpressions.Epsilon = 1E-12
19    // - IPNLP
20    option IPNLP.MaxAttempts = 10
21    option IPNLP.MaxIterations = 400
22    option IPNLP.EpsilonTolerance = 1E-9
23    option IPNLP.MuInitial = 0.1
24    option IPNLP.KappaMu = 0.2 // (0, [0.2], 1)
25    option IPNLP.ThetaMu = 1.5 // (1, [1.5], 2)
26    option IPNLP.ScaleOptimizationVariables = true
27    option IPNLP.ScaleOptimizationFunctions = true
28
29    // scales
30    scale deltaTScale => [0, 4]
31    scale temperatureScale => [30, 100]
32    scale humidityRatioScale => [0, 0.01]
33    scale cfmNormalizedScale => [0, 1.4]
34    scale brakeHorsepowerScale => [0, 0.003]
35 }
36
37 problem
38 {
```

Figure 98 – Beryl's Main User Interface

### Input Model Format

Input models for Beryl are stored as plain text files with an extension of .brl. Beryl adds color formatting itself, but .brl files can be viewed and edited in any text file

viewer. Two statements cannot appear on the same line, and keywords are case sensitive. Optional comments can be added to the end of any line in a file by adding two backslashes “//” and the comment text.

Model input files have an overall structure that consists of a setup section followed by one or more non-nested problem sections, as shown in Figure 99. The setup section contains overall information for a simulation such as the start date, end date, time step period, input file locations, outputs, and special compilation options for models. Problem sections contain mathematical models, where each problem section defines an initial partition for the preprocessing algorithm. Each section’s keyword (`setup` or `problem`), brackets, and information must appear on a new line as in Figure 99.

```
setup // Setup Section
{
  ...
}

problem // First Problem
{
  ...
}

problem // Second Problem
{
  ...
}

:

problem // Last Problem
{
  ...
}
```

**Figure 99 – Model Input Structure of Beryl**

### The Setup Section

The setup section of a model contains the overall parameters for a simulation such as the duration, time step period, input file locations, outputs, scales definitions, and compilation options. As shown in Figure 99, it consists of a line with the keyword “setup”, a line with an open bracket, lines that define its parameters, and a line with a closed bracket to end the section.

#### Setup Section Line Types

Table 48 gives the line types used in the setup section and their formatting. These line types, blank lines, and comments lines are the only lines allowed in problem sections. In strings to show a line’s general format items in brackets “[ ]” are optional.

**Table 48 – Setup Section Line Types**

Line Type	Format																		
Simulation Start Date	<p><code>startdate "1/1/2007 12:00 AM"</code></p> <ul style="list-style-type: none"> <li>The start date of a simulation.</li> <li>A single start date is required.</li> </ul> <p><code>startdate "MM:dd:YYYY hh:mm[:ss] AM/PM"</code></p> <table border="1"> <thead> <tr> <th>Item</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>startdate</code></td> <td>The keyword for the start date definition</td> </tr> <tr> <td><code>MM</code></td> <td>Month</td> </tr> <tr> <td><code>dd</code></td> <td>Day</td> </tr> <tr> <td><code>YYYY</code></td> <td>Year</td> </tr> <tr> <td><code>hh</code></td> <td>Hour (1-12)</td> </tr> <tr> <td><code>mm</code></td> <td>Minutes (0-59)</td> </tr> <tr> <td><code>ss</code></td> <td>Seconds (0-59) (optional)</td> </tr> <tr> <td><code>AM/PM</code></td> <td>AM or PM</td> </tr> </tbody> </table>	Item	Description	<code>startdate</code>	The keyword for the start date definition	<code>MM</code>	Month	<code>dd</code>	Day	<code>YYYY</code>	Year	<code>hh</code>	Hour (1-12)	<code>mm</code>	Minutes (0-59)	<code>ss</code>	Seconds (0-59) (optional)	<code>AM/PM</code>	AM or PM
Item	Description																		
<code>startdate</code>	The keyword for the start date definition																		
<code>MM</code>	Month																		
<code>dd</code>	Day																		
<code>YYYY</code>	Year																		
<code>hh</code>	Hour (1-12)																		
<code>mm</code>	Minutes (0-59)																		
<code>ss</code>	Seconds (0-59) (optional)																		
<code>AM/PM</code>	AM or PM																		
Simulation End Date	<p><code>enddate "1/1/2008 12:00 AM"</code></p> <ul style="list-style-type: none"> <li>The end date of a simulation.</li> <li>A single end date is required.</li> <li>The keyword <code>enddate</code> followed by a date in quotes.</li> <li>Same format as start dates.</li> </ul>																		

**Table 48 Continued**

<b>Line Type</b>	<b>Format</b>														
Simulation Time Step Period	<p align="center"><code>timestep "1:0:0"</code></p> <ul style="list-style-type: none"> <li>• The time step period of a simulation.</li> <li>• A single time step period is required.</li> <li>• The keyword <code>timestep</code> followed by the time step period in quotes.</li> </ul> <p align="center"><code>timestep "[d.]hh:mm:ss[.ffffff]"</code></p> <table border="1" data-bbox="505 527 1333 789"> <thead> <tr> <th data-bbox="505 527 683 562"><b>Item</b></th> <th data-bbox="683 527 1333 562"><b>Description</b></th> </tr> </thead> <tbody> <tr> <td data-bbox="505 562 683 598"><code>timestep</code></td> <td data-bbox="683 562 1333 598">The keyword for the time step period definition</td> </tr> <tr> <td data-bbox="505 598 683 634"><code>d</code></td> <td data-bbox="683 598 1333 634">Days</td> </tr> <tr> <td data-bbox="505 634 683 669"><code>hh</code></td> <td data-bbox="683 634 1333 669">Hours (0-23)</td> </tr> <tr> <td data-bbox="505 669 683 705"><code>mm</code></td> <td data-bbox="683 669 1333 705">Minutes (0-59)</td> </tr> <tr> <td data-bbox="505 705 683 741"><code>ss</code></td> <td data-bbox="683 705 1333 741">Seconds (0-59)</td> </tr> <tr> <td data-bbox="505 741 683 789"><code>ffffff</code></td> <td data-bbox="683 741 1333 789">Fractional seconds (0-9999999) (optional)</td> </tr> </tbody> </table>	<b>Item</b>	<b>Description</b>	<code>timestep</code>	The keyword for the time step period definition	<code>d</code>	Days	<code>hh</code>	Hours (0-23)	<code>mm</code>	Minutes (0-59)	<code>ss</code>	Seconds (0-59)	<code>ffffff</code>	Fractional seconds (0-9999999) (optional)
<b>Item</b>	<b>Description</b>														
<code>timestep</code>	The keyword for the time step period definition														
<code>d</code>	Days														
<code>hh</code>	Hours (0-23)														
<code>mm</code>	Minutes (0-59)														
<code>ss</code>	Seconds (0-59)														
<code>ffffff</code>	Fractional seconds (0-9999999) (optional)														
Dynamic Outputs per Time Step	<p align="center"><code>substeps 100</code></p> <ul style="list-style-type: none"> <li>• Gives the number of instantaneous dynamic readings that are taken on each time step of a simulation.</li> <li>• The keyword <code>substeps</code> followed by a positive integer.</li> <li>• Only required when dynamic variables are being outputted.</li> </ul>														
Input Source Definition	<p align="center"><code>input weather, "root\Houston2007Weather.csv"</code></p> <ul style="list-style-type: none"> <li>• Defines an input data location and a name for it.</li> <li>• The keyword <code>input</code> followed by a name for the input source, followed by a comma, followed by a file name.</li> <li>• For the file name and path:                             <ul style="list-style-type: none"> <li>○ “root” in the file name above refers to the root directory selected in the user interface.</li> <li>○ A full path can also be entered.</li> </ul> </li> <li>• Must be a CSV (comma separated variable) file.</li> </ul>														

Table 48 Continued

Line Type	Format																								
<p>Excel Output Worksheet Definition</p>	<p><code>excel</code> Toa, Tma, Tccla, Tsa, Tza, Eza, "<code>root\output.xlsx:Temps:Yearly</code>"</p> <ul style="list-style-type: none"> <li>• Defines outputs for a single worksheet of an Excel output spreadsheet.</li> <li>• Multiple Excel Output Worksheet Definition lines can be defined to write multiple worksheets to the same Excel file.</li> </ul> <p><code>excel</code> Inputs, "<code>OutputLocation:OutputWorksheet[:CollectionType]</code>"</p> <table border="1" data-bbox="448 562 1385 1350"> <thead> <tr> <th data-bbox="448 562 699 604">Parameter</th> <th data-bbox="699 562 1385 604">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="448 604 699 636"><code>excel</code></td> <td data-bbox="699 604 1385 636">Keyword for an Excel output line.</td> </tr> <tr> <td data-bbox="448 636 699 674"><code>Inputs</code></td> <td data-bbox="699 636 1385 674">Comma separated list of variables to output.</td> </tr> <tr> <td data-bbox="448 674 699 711"><code>OutputLocation</code></td> <td data-bbox="699 674 1385 711">The name of the new Excel file to write to.</td> </tr> <tr> <td data-bbox="448 711 699 749"><code>OutputWorksheet</code></td> <td data-bbox="699 711 1385 749">The name of the worksheet to write to.</td> </tr> <tr> <td data-bbox="448 749 699 1350"><code>CollectionType</code></td> <td data-bbox="699 749 1385 1350"> <ul style="list-style-type: none"> <li>• How to sum or average output data over time before writing it.</li> <li>• Allowable values: <table border="1" data-bbox="732 865 1357 1129"> <thead> <tr> <th data-bbox="732 865 914 940">Collection Type</th> <th data-bbox="914 865 1357 940">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="732 940 914 978"><code>blank</code></td> <td data-bbox="914 940 1357 978">Write data from each time step.</td> </tr> <tr> <td data-bbox="732 978 914 1016"><code>Hourly</code></td> <td data-bbox="914 978 1357 1016">Sum or average data by hour.</td> </tr> <tr> <td data-bbox="732 1016 914 1054"><code>Daily</code></td> <td data-bbox="914 1016 1357 1054">Sum or average data by day.</td> </tr> <tr> <td data-bbox="732 1054 914 1092"><code>Monthly</code></td> <td data-bbox="914 1054 1357 1092">Sum or average data by month.</td> </tr> <tr> <td data-bbox="732 1092 914 1129"><code>Yearly</code></td> <td data-bbox="914 1092 1357 1129">Sum or average data by year.</td> </tr> </tbody> </table> </li> <li>• Outputs are only written if full and valid data is available to sum.</li> <li>• Inputs such as dollars or energy use are summed, while temperatures, flow rates, humidity ratios, and energy usages normalized to time are averaged.</li> </ul> </td> </tr> </tbody> </table>	Parameter	Description	<code>excel</code>	Keyword for an Excel output line.	<code>Inputs</code>	Comma separated list of variables to output.	<code>OutputLocation</code>	The name of the new Excel file to write to.	<code>OutputWorksheet</code>	The name of the worksheet to write to.	<code>CollectionType</code>	<ul style="list-style-type: none"> <li>• How to sum or average output data over time before writing it.</li> <li>• Allowable values: <table border="1" data-bbox="732 865 1357 1129"> <thead> <tr> <th data-bbox="732 865 914 940">Collection Type</th> <th data-bbox="914 865 1357 940">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="732 940 914 978"><code>blank</code></td> <td data-bbox="914 940 1357 978">Write data from each time step.</td> </tr> <tr> <td data-bbox="732 978 914 1016"><code>Hourly</code></td> <td data-bbox="914 978 1357 1016">Sum or average data by hour.</td> </tr> <tr> <td data-bbox="732 1016 914 1054"><code>Daily</code></td> <td data-bbox="914 1016 1357 1054">Sum or average data by day.</td> </tr> <tr> <td data-bbox="732 1054 914 1092"><code>Monthly</code></td> <td data-bbox="914 1054 1357 1092">Sum or average data by month.</td> </tr> <tr> <td data-bbox="732 1092 914 1129"><code>Yearly</code></td> <td data-bbox="914 1092 1357 1129">Sum or average data by year.</td> </tr> </tbody> </table> </li> <li>• Outputs are only written if full and valid data is available to sum.</li> <li>• Inputs such as dollars or energy use are summed, while temperatures, flow rates, humidity ratios, and energy usages normalized to time are averaged.</li> </ul>	Collection Type	Description	<code>blank</code>	Write data from each time step.	<code>Hourly</code>	Sum or average data by hour.	<code>Daily</code>	Sum or average data by day.	<code>Monthly</code>	Sum or average data by month.	<code>Yearly</code>	Sum or average data by year.
Parameter	Description																								
<code>excel</code>	Keyword for an Excel output line.																								
<code>Inputs</code>	Comma separated list of variables to output.																								
<code>OutputLocation</code>	The name of the new Excel file to write to.																								
<code>OutputWorksheet</code>	The name of the worksheet to write to.																								
<code>CollectionType</code>	<ul style="list-style-type: none"> <li>• How to sum or average output data over time before writing it.</li> <li>• Allowable values: <table border="1" data-bbox="732 865 1357 1129"> <thead> <tr> <th data-bbox="732 865 914 940">Collection Type</th> <th data-bbox="914 865 1357 940">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="732 940 914 978"><code>blank</code></td> <td data-bbox="914 940 1357 978">Write data from each time step.</td> </tr> <tr> <td data-bbox="732 978 914 1016"><code>Hourly</code></td> <td data-bbox="914 978 1357 1016">Sum or average data by hour.</td> </tr> <tr> <td data-bbox="732 1016 914 1054"><code>Daily</code></td> <td data-bbox="914 1016 1357 1054">Sum or average data by day.</td> </tr> <tr> <td data-bbox="732 1054 914 1092"><code>Monthly</code></td> <td data-bbox="914 1054 1357 1092">Sum or average data by month.</td> </tr> <tr> <td data-bbox="732 1092 914 1129"><code>Yearly</code></td> <td data-bbox="914 1092 1357 1129">Sum or average data by year.</td> </tr> </tbody> </table> </li> <li>• Outputs are only written if full and valid data is available to sum.</li> <li>• Inputs such as dollars or energy use are summed, while temperatures, flow rates, humidity ratios, and energy usages normalized to time are averaged.</li> </ul>	Collection Type	Description	<code>blank</code>	Write data from each time step.	<code>Hourly</code>	Sum or average data by hour.	<code>Daily</code>	Sum or average data by day.	<code>Monthly</code>	Sum or average data by month.	<code>Yearly</code>	Sum or average data by year.												
Collection Type	Description																								
<code>blank</code>	Write data from each time step.																								
<code>Hourly</code>	Sum or average data by hour.																								
<code>Daily</code>	Sum or average data by day.																								
<code>Monthly</code>	Sum or average data by month.																								
<code>Yearly</code>	Sum or average data by year.																								
<p>Chart Output Definition</p>	<p><code>chart</code> Toa, Tma, Tccla, Tsa, Tza, Eza</p> <ul style="list-style-type: none"> <li>• The keyword <code>chart</code> followed by a comma delimited list of output variables to display.</li> <li>• Charts are initially shown with all variables on the y-axis and the date as the x-axis.</li> </ul>																								
<p>Scale Definition</p>	<p><code>scale</code> temperatureScale =&gt; [30, 100]</p> <ul style="list-style-type: none"> <li>• Defines a scale so it can be reused on multiple variables in a model.</li> <li>• The keyword <code>scale</code> followed by a name for the scale, followed by a “=&gt;” followed by the min and max values in bracket.</li> </ul>																								

**Table 48 Continued**

Line Type	Format
Compilation Option Definition	<p style="text-align: center;"><code>option SmoothExpressions = true</code></p> <ul style="list-style-type: none"> <li>• The keyword <code>option</code> followed by the option name, an equals sign, and the option's set value.</li> <li>• Descriptions of the different user options allowed are given in Table 49.</li> </ul>

### Beryl User Options

Compilation options in the setup section allow the compilation process to be modified. This is useful for experimenting with different preprocessing steps and settings for numerical methods to improve the overall solving process. Table 49 lists the available user options, their allowed values, and their default values. Default values are used when the user option isn't defined.

**Table 49 – Beryl User Options**

Option and Default Value
<p style="text-align: center;">MaxPresolvingOperations = 1000</p> <ul style="list-style-type: none"> <li>• The maximum number of presolving operations to perform before terminating the algorithm.</li> <li>• Must be greater than zero.</li> </ul>
<p style="text-align: center;">IgnoreAssertions = true</p> <ul style="list-style-type: none"> <li>• Whether to ignore vertices that only contain previously calculated states</li> <li>• Allowed values: true/false</li> </ul>
<p style="text-align: center;">DecomposeUnivariateLinearEqualities = true</p> <ul style="list-style-type: none"> <li>• Whether to perform univariate linear equality solving from Ch VI</li> <li>• Allowed values: true/false</li> </ul>
<p style="text-align: center;">DecomposeBivariateLinearEqualities = true</p> <ul style="list-style-type: none"> <li>• Whether to perform bivariate linear equality solving from Ch VI</li> <li>• Allowed values: true/false</li> </ul>
<p style="text-align: center;">DecomposeGeneralLinearEqualities = false</p> <ul style="list-style-type: none"> <li>• Whether to perform general algebraic decomposition from Ch VI on linear functions.</li> <li>• Allowed values: true/false</li> </ul>



**Table 49 Continued**

<b>Option and Default Value</b>	
SmoothExpressions = false	
<ul style="list-style-type: none"> <li>• Whether to smooth expressions with discontinuous first derivatives.</li> <li>• Allowed values: true/false</li> </ul>	
SmoothExpressions.SmoothingFunction = Sqrt	
<ul style="list-style-type: none"> <li>• The function to use to smooth expressions with discontinuous first derivatives.</li> <li>• Allowable values:</li> </ul>	
<b>Value</b>	<b>Description</b>
Sqrt	Smooths discontinuities based on a square root approximation of absolute values $ x  \approx \sqrt{x^2 + \varepsilon}$ .
BGamma	Smooths discontinuities using Bertsekas's gamma function $Max(x, 0) \approx \gamma(x, \lambda, c)$ from Appendix C
SmoothExpressions.Epsilon = 1E-12	
<ul style="list-style-type: none"> <li>• The <math>\varepsilon</math> to use in the square root smoothing function</li> <li>• <math>\varepsilon \geq 0</math></li> </ul>	
SmoothExpressions.Lambda = 0.5	
<ul style="list-style-type: none"> <li>• The <math>\lambda</math> to use in Bertsekas's gamma function</li> <li>• <math>\lambda \in [0, 1]</math></li> </ul>	
SmoothExpressions.C = 1,000	
<ul style="list-style-type: none"> <li>• The value of <math>c</math> to use in the Bertsekas's gamma function.</li> <li>• <math>c \geq 0</math></li> </ul>	
IPNLP.MaxAttempts = 4	
<ul style="list-style-type: none"> <li>• The maximum number of attempts to successfully use IPNLP algorithm from a unique starting position before giving up.</li> <li>• Must be greater than zero.</li> </ul>	
IPNLP.MaxIterations = 200	
<ul style="list-style-type: none"> <li>• The maximum number of iterations for a single attempt of the IPNLP algorithm.</li> <li>• Must be greater than zero.</li> <li>• Corresponds to the "max_iter" option in IPOPT.</li> </ul>	
IPNLP.EpsilonTolerance = 1E-9	
<ul style="list-style-type: none"> <li>• The convergence tolerance for the IPNLP algorithm.</li> <li>• IPNLP terminates successfully when this error value is reached</li> <li>• Must be greater than or equal to zero.</li> <li>• Corresponds to the "tol" option in IPOPT.</li> </ul>	
IPNLP.EpsilonToleranceAcceptable = 1E-8	
<ul style="list-style-type: none"> <li>• Failed calculations in the IPNLP algorithm are accepted if their error is below this.</li> <li>• Must be greater than or equal to zero.</li> </ul>	

**Table 49 Continued**

<b>Option and Default Value</b>
<p style="text-align: center;">IPNLP.MuInitial = 0.1</p> <ul style="list-style-type: none"> <li>• The initial value of the barrier parameter in the IPNLP algorithm.</li> <li>• Must be greater than zero.</li> <li>• Corresponds to the “mu_init” option in IPOPT</li> </ul>
<p style="text-align: center;">IPNLP.MuInitialWithSensitivityAnalysis = 1E-4</p> <ul style="list-style-type: none"> <li>• The initial value of the barrier parameter in the IPNLP algorithm when the calculation was initialized using a previous solution.</li> <li>• Must be greater than zero.</li> </ul>
<p style="text-align: center;">IPNLP.KappaMu = 0.2</p> <ul style="list-style-type: none"> <li>• Controls the linear decrease rate of the barrier parameter in the Fiacco-McCormick barrier update procedure.</li> <li>• Must be between 0 and 1.</li> <li>• Corresponds to the “mu_linear_decrease_factor” option in IPOPT.</li> </ul>
<p style="text-align: center;">IPNLP.ThetaMu = 1.5</p> <ul style="list-style-type: none"> <li>• Controls the superlinear decrease rate of the barrier parameter in the Fiacco-McCormick barrier update procedure.</li> <li>• Must be between 1 and 2.</li> <li>• Corresponds to the “mu_superlinear_decrease_power” option in IPOPT.</li> </ul>
<p style="text-align: center;">IPNLP.ScaleOptimizationVariables = true</p> <ul style="list-style-type: none"> <li>• Whether to scale optimized variables when using the IPNLP algorithm to constrained optimizations.</li> <li>• Allowed values: true/false</li> </ul>
<p style="text-align: center;">IPNLP.ScaleOptimizationFunctions = true</p> <ul style="list-style-type: none"> <li>• Whether to scale optimization functions when using the IPNLP algorithm to constrained optimizations.</li> <li>• Allowed values: true/false</li> </ul>
<p style="text-align: center;">IPNLP.MainLineSearchReductionFactor = 0.5</p> <ul style="list-style-type: none"> <li>• The factor to reduce the main line search step size by when searching for a better solution in the main phase of the IPNLP algorithm.</li> <li>• Must be between 0 and 1.</li> <li>• Corresponds to the “alpha_red_factor” option in IPOPT.</li> </ul>
<p style="text-align: center;">IPNLP.RestorationLineSearchReductionFactor = 0.5</p> <ul style="list-style-type: none"> <li>• The factor to reduce the main line search step size by when searching for a better solution in the restoration phase of the IPNLP algorithm.</li> <li>• Must be between 0 and 1.</li> </ul>

**Table 49 Continued**

<b>Option and Default Value</b>
IPNLP.InitializeWithSensitivityAnalysis = true
<ul style="list-style-type: none"> <li>• Whether to try to initialize the problem with sensitivity analysis on a first attempt after a successful previous run.</li> <li>• Allowed values: true/false</li> </ul>
IPNLP.KappaEpsilon = 10
<ul style="list-style-type: none"> <li>• Mu is decreased when the barrier function error is less than this times mu.</li> <li>• Must be greater than zero</li> <li>• Corresponds to the “barrier_tol_factor” option in IPOPT</li> </ul>
IPNLP.EnablePreconditioning = true
<ul style="list-style-type: none"> <li>• Whether to allow preconditioning in linear algebra calculations.</li> <li>• Allowed values: true/false</li> </ul>
IPNLP.EnableSolutionRefinement = false
<ul style="list-style-type: none"> <li>• Whether to refine search directions using iterative refinement in linear algebra calculations.</li> <li>• Allowed values: true/false</li> </ul>

### *Problem Sections*

Problem sections in a file contain all mathematical modeling. As shown in Figure 99, it consists of a line with the keyword “problem”, a line with an open bracket, lines that define its parameters, and a line with a closed bracket to end the section.

### **Problem Section Line Types**

Table 50 gives the line types used in the problem section and their formatting. These line types, blank lines, and comments lines are the only lines allowed in problem sections.

**Table 50 – Problem Section Line Types**

Line Type	Format																
<p>Constant Floating Point Variable</p>	<p><code>double</code> <math>\bar{V}_{sa} = 1.4 \text{ CFM/ft}^2</math></p> <ul style="list-style-type: none"> <li>• Defines a constant floating point variable.</li> <li>• The keyword <code>double</code>, followed by the constant’s variable name, followed by an equals sign, followed by the constant value, followed by the unit.</li> <li>• Allowed units are described below.</li> </ul>																
<p>Averaged Floating Point Variable</p>	<p><code>double</code> <math>T_{sa}, ^\circ\text{F}</math>  <code>double</code> <math>T_{sa}, ^\circ\text{F} =&gt; [40, 100]</math>  <code>double</code> <math>T_{sa}, ^\circ\text{F} =&gt; \text{temperatureScale}</math></p> <ul style="list-style-type: none"> <li>• Defines a floating point variable that represents an average value over a time step.</li> <li>• The keyword <code>double</code>, followed by the variable name, followed by a comma, followed by the unit, followed by an optional “=&gt;” and scale.</li> <li>• Allowed units are described below.</li> <li>• The optional scale can be in the “[min, max]” format or can be a reference to a pre-defined scale from the setup section.</li> </ul>																
<p>Input Variable Definition</p>	<p><code>double[]</code> <math>Toa = \text{weather}["A", "B", "2"], ^\circ\text{F}</math></p> <ul style="list-style-type: none"> <li>• Defines a variable and how it’s derived from an input source.</li> </ul> <p><code>double[]</code> Variable = InputSource["DateColumn", "DataColumn", "StartRow"], Unit</p> <table border="1" data-bbox="521 1192 1362 1717"> <thead> <tr> <th data-bbox="521 1192 712 1234">Item</th> <th data-bbox="712 1192 1362 1234">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="521 1234 712 1272"><code>double[]</code></td> <td data-bbox="712 1234 1362 1272">The keyword for an input variable definition.</td> </tr> <tr> <td data-bbox="521 1272 712 1346">Variable</td> <td data-bbox="712 1272 1362 1346">The name of the loaded variable for use in calculations.</td> </tr> <tr> <td data-bbox="521 1346 712 1457">InputSource</td> <td data-bbox="712 1346 1362 1457">The name of the input source to load data from. This comes from an <code>input</code> line in the setup section.</td> </tr> <tr> <td data-bbox="521 1457 712 1530">DateColumn</td> <td data-bbox="712 1457 1362 1530">The column for dates in the input’s CSV file, in Excel’s letter column format.</td> </tr> <tr> <td data-bbox="521 1530 712 1604">DataColumn</td> <td data-bbox="712 1530 1362 1604">The column for data in the input’s CSV file, in Excel’s letter column format.</td> </tr> <tr> <td data-bbox="521 1604 712 1677">StartRow</td> <td data-bbox="712 1604 1362 1677">The row to start getting data from, in Excel’s numeric row format.</td> </tr> <tr> <td data-bbox="521 1677 712 1717">Unit</td> <td data-bbox="712 1677 1362 1717">The variable’s unit.</td> </tr> </tbody> </table>	Item	Description	<code>double[]</code>	The keyword for an input variable definition.	Variable	The name of the loaded variable for use in calculations.	InputSource	The name of the input source to load data from. This comes from an <code>input</code> line in the setup section.	DateColumn	The column for dates in the input’s CSV file, in Excel’s letter column format.	DataColumn	The column for data in the input’s CSV file, in Excel’s letter column format.	StartRow	The row to start getting data from, in Excel’s numeric row format.	Unit	The variable’s unit.
Item	Description																
<code>double[]</code>	The keyword for an input variable definition.																
Variable	The name of the loaded variable for use in calculations.																
InputSource	The name of the input source to load data from. This comes from an <code>input</code> line in the setup section.																
DateColumn	The column for dates in the input’s CSV file, in Excel’s letter column format.																
DataColumn	The column for data in the input’s CSV file, in Excel’s letter column format.																
StartRow	The row to start getting data from, in Excel’s numeric row format.																
Unit	The variable’s unit.																

**Table 50 Continued**

Line Type	Format																										
Schedule Definition	<p><code>double</code>&lt;&gt; XPeople = &lt;{1/1/2007-1/1/2008: [Weekdays: (12AM, 0), (7:30AM, 0), (9:30AM, 1), (4:30PM, 1), (6:30PM, 0), (12AM*, 0)], [Weekends: (12AM, 0), (12AM*, 0)]}&gt;, 0-1</p> <ul style="list-style-type: none"> <li>• Defines a variable that changes over time.</li> <li>• The overall schedule is formatted as follows:  <code>double</code>&lt;&gt; Variable = &lt;Schedules &gt;, Unit</li> </ul> <table border="1" data-bbox="521 600 1365 827"> <thead> <tr> <th>Item</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>double</code>&lt;&gt;</td> <td>The keyword for a schedule definition.</td> </tr> <tr> <td>Variable</td> <td>The name of the schedule variable.</td> </tr> <tr> <td>Schedules</td> <td>A comma separated list of schedules, as described below, placed in round brackets.</td> </tr> <tr> <td>Unit</td> <td>The schedule variable's unit.</td> </tr> </tbody> </table> <ul style="list-style-type: none"> <li>• Schedules are defined as nested lists with four different levels:</li> </ul> <table border="1" data-bbox="526 905 1360 1312"> <thead> <tr> <th>List Level</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Time/Value Pair</td> <td>A time of day paired with an instantaneous value at that time.</td> </tr> <tr> <td>Day Schedule</td> <td>A list of time/value pairs that define how a value changes on a particular type of day.</td> </tr> <tr> <td>Time Schedule</td> <td>A list of day schedules that define how a value changes between two dates.</td> </tr> <tr> <td>Overall Schedule</td> <td>A list of time schedules that define a variable's value over a complete simulation.</td> </tr> </tbody> </table> <ul style="list-style-type: none"> <li>• Time/Value Pairs are placed in round brackets:  <code>(TimeOfDay, Value)</code></li> </ul> <table border="1" data-bbox="634 1423 1252 1539"> <thead> <tr> <th>Item</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>TimeOfDay</td> <td>The time of day for the value</td> </tr> <tr> <td>Value</td> <td>The value at that time of day</td> </tr> </tbody> </table> <ul style="list-style-type: none"> <li>○ An asterisk after the time of day means the next day.</li> <li>○ A schedule extending into the next day overrides the normal existing schedule.</li> </ul>	Item	Description	<code>double</code> <>	The keyword for a schedule definition.	Variable	The name of the schedule variable.	Schedules	A comma separated list of schedules, as described below, placed in round brackets.	Unit	The schedule variable's unit.	List Level	Description	Time/Value Pair	A time of day paired with an instantaneous value at that time.	Day Schedule	A list of time/value pairs that define how a value changes on a particular type of day.	Time Schedule	A list of day schedules that define how a value changes between two dates.	Overall Schedule	A list of time schedules that define a variable's value over a complete simulation.	Item	Description	TimeOfDay	The time of day for the value	Value	The value at that time of day
Item	Description																										
<code>double</code> <>	The keyword for a schedule definition.																										
Variable	The name of the schedule variable.																										
Schedules	A comma separated list of schedules, as described below, placed in round brackets.																										
Unit	The schedule variable's unit.																										
List Level	Description																										
Time/Value Pair	A time of day paired with an instantaneous value at that time.																										
Day Schedule	A list of time/value pairs that define how a value changes on a particular type of day.																										
Time Schedule	A list of day schedules that define how a value changes between two dates.																										
Overall Schedule	A list of time schedules that define a variable's value over a complete simulation.																										
Item	Description																										
TimeOfDay	The time of day for the value																										
Value	The value at that time of day																										

**Table 50 Continued**

Line Type	Format														
<p>Schedule Definition (Continued)</p>	<ul style="list-style-type: none"> <li>Day schedules are placed in square brackets:  <div style="border: 1px solid black; padding: 2px; margin: 5px 0;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 30%;">Item</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DayType</td> <td>The type of day that the time schedule is for.</td> </tr> <tr> <td>ScheduleForDay</td> <td>A list of time/value to define the schedule for the day</td> </tr> </tbody> </table> </div> <ul style="list-style-type: none"> <li>DayType choices are All, Weekdays, Weekends, Sundays, Mondays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays.</li> </ul> </li> <li>Time schedules are placed in curly brackets:  <div style="border: 1px solid black; padding: 2px; margin: 5px 0;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 30%;">Item</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>StartDate</td> <td>When the schedule starts applying.</td> </tr> <tr> <td>EndDate</td> <td>When the schedule stops applying.</td> </tr> <tr> <td>DaySchedules</td> <td>A list of day schedules that define a value between the start and end date.</td> </tr> </tbody> </table> </div> </li> </ul>	Item	Description	DayType	The type of day that the time schedule is for.	ScheduleForDay	A list of time/value to define the schedule for the day	Item	Description	StartDate	When the schedule starts applying.	EndDate	When the schedule stops applying.	DaySchedules	A list of day schedules that define a value between the start and end date.
Item	Description														
DayType	The type of day that the time schedule is for.														
ScheduleForDay	A list of time/value to define the schedule for the day														
Item	Description														
StartDate	When the schedule starts applying.														
EndDate	When the schedule stops applying.														
DaySchedules	A list of day schedules that define a value between the start and end date.														
<p>Dynamic Variable Definition</p>	<p align="center"><b>dynamic</b> TWallDyn, TWallAvg, 70.0, °F</p> <ul style="list-style-type: none"> <li>Defines a variable that varies over time and an associated variable to represent its average over a time step.</li> <li>The keyword <b>dynamic</b>, followed by a comma delimited list with the continuous variable name, averaged variable name, initial value, and unit.</li> </ul>														
<p>Minimization Objective</p>	<p align="center"><b>minimize</b> Eza1 + Eza2, 0</p> <ul style="list-style-type: none"> <li>The keyword <b>minimize</b> followed by an algebraic expression to minimize, followed by a comma, followed by the lexicographic priority.</li> <li>0 is the highest lexicographic priority, with higher numbers having a lower priority.</li> </ul>														
<p>Maximization Objective</p>	<p align="center"><b>maximize</b> X1 + X2, 0</p> <ul style="list-style-type: none"> <li>The keyword <b>maximize</b> followed by an algebraic expression to maximize, followed by a comma, followed by the lexicographic priority.</li> <li>0 is the highest lexicographic priority, with higher numbers having a lower priority.</li> </ul>														

**Table 50 Continued**

<b>Line Type</b>	<b>Format</b>
Equality or Inequality Constraint	<code>constraint <math>\bar{V}_{sa} \geq \bar{V}_{saMin}</math></code> <ul style="list-style-type: none"><li>• Defines an equality or inequality constraint.</li><li>• The keyword <code>constraint</code> followed by an equality or inequality.</li><li>• Inequality operators can be less than or equal or greater than or equal only.</li></ul>

### Units

Every variable in a model is required to have a unit associated with it. This helps keep track of unit consistency in equations. A variable's unit can be displayed in the user interface by holding a mouse over it. This displays a tooltip window that shows text from the variable's definition line.

Table 51 displays the units available in Beryl. For faster access in the user interface a list of these units can be displayed by entering pound '#' into the code editor. These units are typically used in building energy modeling, so not all units are implemented. If a desired unit is missing the unit *None* can be used as a placeholder to run a simulation.

**Table 51 – Units Implemented in Beryl**

<b>Type</b>	<b>SI Units</b>	<b>IP Units</b>
No Unit	<i>None, 0 – 1, %</i>	
Length	<i>m</i>	<i>ft</i>
Area	<i>m<sup>2</sup></i>	<i>ft<sup>2</sup></i>
Volume	<i>m<sup>3</sup></i>	<i>ft<sup>3</sup></i>
Velocity	<i>m/s</i>	<i>ft/s</i>
Flow	<i>m<sup>3</sup>/s</i>	<i>CFM</i>
Flow per Unit Area	<i>m/s</i>	<i>CFM/ft<sup>2</sup></i>
Density	<i>kg/m<sup>3</sup></i>	<i>lbm/ft<sup>3</sup></i>
Energy	<i>J</i>	<i>Btu, MMBtu, kWh</i>
Power	<i>W, kW</i>	<i>hp, Btu/h, MMBtu/h</i>
Power Density	<i>W/ft<sup>2</sup>, kW/ft<sup>2</sup></i>	<i>hp/ft<sup>2</sup>, Btu/(h · ft<sup>2</sup>), MMBtu/(h · ft<sup>2</sup>)</i>
R-Value	<i>(m<sup>2</sup> · K)/W</i>	<i>(h · ft<sup>2</sup> · °F)/Btu</i>
U-Value	<i>W/(m<sup>2</sup> · K)</i>	<i>Btu/(h · ft<sup>2</sup> · °F)</i>
UA-Value	<i>W/K</i>	<i>Btu/(h · °F)</i>
Thermal Conductivity	<i>W/(m · K)</i>	<i>Btu/(h · ft · °F)</i>
Heat Capacity	<i>J/K</i>	<i>Btu/°F</i>
Specific Heat Capacity	<i>J/(kg · K)</i>	<i>Btu/(lbm · °F)</i>
Temperature	<i>°C, K</i>	<i>°F, °R</i>
Pressure	<i>Pa</i>	<i>Psi, inH<sub>2</sub>O</i>
Humidity Ratio	<i>kg<sub>w</sub>/kg<sub>da</sub></i>	<i>lb<sub>w</sub>/lb<sub>da</sub></i>
Frequency	<i>Hz</i>	
Cost	<i>\$</i>	
Cost per Unit Area	<i>\$/m<sup>2</sup></i>	<i>\$/ft<sup>2</sup></i>
Cost per Unit Time per Unit Area	<i>\$/ (h · m<sup>2</sup>)</i>	<i>\$/ (h · ft<sup>2</sup>)</i>



## Scales

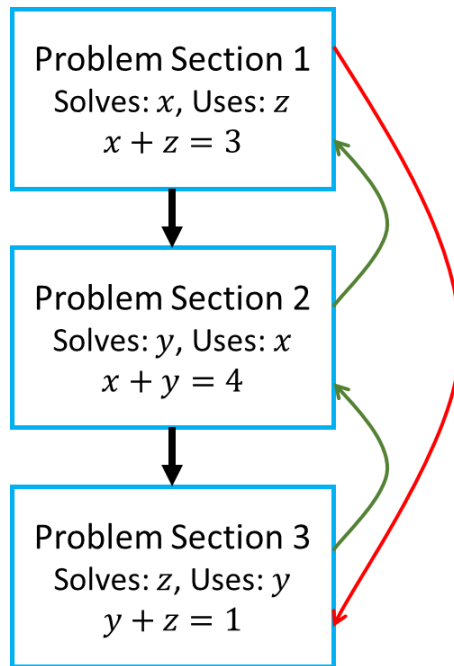
Scaling of variables in constrained optimization problems is enabled by associating variables with a scale. A variable's scale can be defined as a fixed min and max value in square brackets or it can be the identifier of a reused scale that's defined in the setup section. Scales are not required for all variables, but the preprocessing process won't succeed if a constrained optimization that needs to be scaled has unscaled variables.

Scaling the variables and functions of constrained optimization based building energy models is important because of the sensitivity of constrained optimization solvers and the wide variation of variables in these models. For instance, a humidity ratio balance function may involve humidity ratios on the order of  $0.1 \text{ kg}_w/\text{kg}_{da}$ , while an air flow balance in the same model may involve air flows on the order of  $100,000 \text{ CFM}$ . Not scaling these functions to the same magnitude could result in failed calculations or large humidity errors in calculated humidities that cause large errors in a building's calculated energy usage.

## Problem Section Order

Multiple problem sections in an input file serve as the original partitioned subproblems. This allows problems to be partitioned in ways the automatic preprocessing algorithm can't detect. For instance, temperature and humidity calculations can be decoupled when no relative humidity control is present. However, but this must be done manually because no algorithm to decouple the problem as formulated in Chapter V could be found. Multiple problem sections can also serve as containers for model code so that models are more readable.

Problem sections must be entered in a valid solution order, where no problem section uses variables solved later as inputs. This means that problem sections can't have cyclic dependencies between the variables they solve. Figure 100 shows a model that's invalid because of cyclic dependencies. Valid dependencies between problem sections are shown using green arrows, and the invalid dependency is shown using a red arrow. The first problem section uses the variable  $z$  in its equations but  $z$  is calculated in the third problem section. The dependencies between the first and second problem sections and the second and third create a cycle of dependence between all three problem sections.



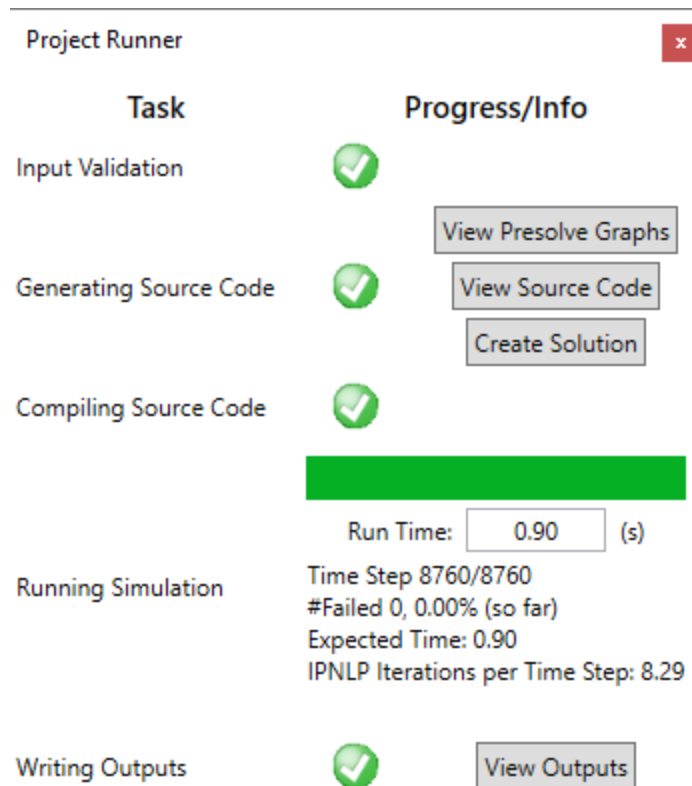
**Figure 100 – Problem Sections with a Cyclic Dependency**

### Running a Model

Models can be run by clicking the “run” button in the upper left-hand corner of the main window. One of two windows is shown depending on whether the model has input errors. If input errors exist the Input File Error Window of Figure 101 is shown. This window lists each error and its position in the file. If no input errors exist the Project Runner Window of Figure 102 is shown. This window gives details on the processing of a model and running of the simulation.



**Figure 101 – Input File Error Window**



**Figure 102 – Project Runner Window**

Processing and running a model takes place in five phases: input validation, source code generation, source code compilation, running the simulation, and writing and showing simulation outputs. The Project Runner Window displays information from each of these phases. If a phase fails a red x is shown and an error message may be viewed. If a phase succeeds it's given a green check or bar.

When running a model, first the input validation phase validates the internal model that was made from the text input file. This validation is independent of the UI validation and was made so that models can be validated when created from another user interface.

Next, the source code generation phase runs the automated solution algorithm from Chapter VI and results in C# source code to compile and execute. The “View Presolve Graphs” button shows this process in detail by opening the window from

Figure 82. The “View Source Code” button displays a text file of the generated source code. Finally, the “Create Solution” button creates and opens a Visual Studio solution containing the source code and inputs from the simulation. This solution executes the same code as a simulation, which allows generated code to be debugged line by line.

Next, in the source code compilation phase the generated source code is compiled. If the compilation fails a button is displayed that displays model errors when clicked. This is used for debugging the code generation process.

If source code is compiled successfully it’s then executed. A progress bar is displayed to show how much of a simulation has been completed, and information on the run time, current time step, failed iterations, expected completion time, and the number of iterations required for each run of the IPNLP algorithm are given. This information is used to fine tune the solution algorithms used.

Finally, if a simulation completes successfully its outputs are written and displayed. Charts and Excel files are displayed automatically and clicking the “View Outputs” button reopens the Excel files. A “View Exception” button appears if the steps of the Project Runner Window fail. Clicking this button gives information on what part of the process has failed.