Modeling Market and Regulatory Mechanisms for Pollution Abatement with Sharp and Random Variables

Thomas Robert Fielden

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Let us know how access to this document benefits you.

Recommended Citation

https://doi.org/10.15760/etd.282

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.
Modeling Market and Regulatory Mechanisms for Pollution Abatement with Sharp and Random Variables

by

Thomas Robert Fielden

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematical Sciences

Dissertation Committee:
Steven A. Bleiler, Chair
Dacian N. Daescu
Jong-Sung Kim
Randall Bluffstone
Thomas S. Ferguson
John Settle

Portland State University
©2011
ABSTRACT

This dissertation is motivated by the problem of uncertainty and sensitivity in business-class models such as the carbon emission abatement policy model featured in this work. Uncertain model inputs are represented by numerical random variables and a computational methodology is developed to numerically compute business-class models as if sharp inputs were given. A new description for correlation of random variables is presented that arises spontaneously within a numerical model. Methods of numerically computing correlated random variables are implemented in software and represented.

The major contribution of this work is a methodology for the numerical computation of models under uncertainty that expresses no preference for unlikelihood of model input combinations. The methodology presented here serves a sharp contrast to traditional Monte Carlo methods that implicitly equate likelihood of model input values with importance of results. The new methodology herein shifts the computational burden from likelihood of inputs to resolution of input space.
# TABLE OF CONTENTS

ABSTRACT ................................................................. i  
LIST OF TABLES ...................................................... v  
LIST OF FIGURES ..................................................... vii  
CHAPTER  
1. Statement of the Problem ............................................. 1  
   1.1 The California Question ........................................... 1  
   1.2 Background ........................................................... 2  
      1.2.1 Cap-and-Trade ............................................... 2  
      1.2.2 Modeling Techniques of Environmental Regulation ....... 3  
      1.2.3 Requirements for the CARB Model ....................... 6  
      1.2.4 Modeling Background for the CARB Model .............. 6  
2. Elementary Modeling with Sharp Variables ......................... 9  
   2.1 A Modeling Approach .............................................. 9  
   2.2 Overview of the Elementary Model ............................... 10  
      2.2.1 Processing Raw Data in the Elementary Model .......... 13  
      2.2.2 Applying Policy Features to the Elementary Model .... 14  
      2.2.3 Simulating a Cost Effective Market .................... 15  
      2.2.4 Introducing a Class Constraint to the Elementary Model 20  
      2.2.5 Introducing a Foreign Market to the Elementary Model . 23  
      2.2.6 Introducing Mandatory Requirements to the Elementary Model . 31  
   2.3 Implementation Considerations .................................. 33  
      2.3.1 An Example Demonstrating Design Issues ................. 35  
      2.3.2 A Sample of Useful Core Application Data Types ...... 37  
      2.3.3 Two Ways to Access Variants ............................. 40
2.3.4 Example for Running an LP Engine ............... 42
2.4 Insights from the Elementary Model ............... 44

3. Sensitivity, Uncertainty and Risk Analysis I .............. 47

3.1 Introduction ............................................. 47
3.2 Introduction to Random Variable Arithmetic .......... 52
  3.2.1 Functions of a Random Variable ................. 52
  3.2.2 Functions of a Random Vector ................. 54
  3.2.3 Extending the Theory ............................. 56
3.3 Introduction to Numerical Random Variable Arithmetic .... 62
  3.3.1 Representing a Random Variable Numerically .... 63
  3.3.2 Functions of One Random Variable ............. 68
  3.3.3 The Sunrise Algorithm ............................. 73
  3.3.4 One Random Variable Exponential Functions .... 77
  3.3.5 Functions of Two Independent Random Variables .... 78
  3.3.6 Exponential Function With Two Independent Random Variables ........................................... 93
  3.3.7 Sample Results ....................................... 93
  3.3.8 Computing Net Present Value, An Example .... 97
  3.3.9 A Comparison to an Alternate Numerical Approach . 99
  3.3.10 Arithmetic Correlation of Random Variables .... 103

4. Revisiting the Model ..................................... 106

4.1 Revisitations’ Goal ................................. 106
  4.1.1 Model Inputs ....................................... 107
  4.1.2 Capacity ............................................. 108
  4.1.3 Levelized Costs ..................................... 110
  4.1.4 Forecast Costs and Capacity ................... 113
4.2 The Simplex Method .................................... 114
  4.2.1 A Working Example ................................. 114
  4.2.2 The Tableau ........................................ 117
  4.2.3 A Graphical Example ............................... 120
  4.2.4 The First Round .................................... 121
  4.2.5 Another Round, Faster ............................. 126
  4.2.6 The Completion Round ............................. 129
4.3 The Effect on Random Variables ......................... 131

5. Sensitivity, Uncertainty and Risk Analysis II .............. 133

5.1 A Standard Approach .................................. 133
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Initial Simplex Tableau</td>
<td>120</td>
</tr>
<tr>
<td>4.2</td>
<td>First Step Simplex Tableau</td>
<td>127</td>
</tr>
<tr>
<td>4.3</td>
<td>Second Step Simplex Tableau</td>
<td>128</td>
</tr>
<tr>
<td>4.4</td>
<td>Third Step Simplex Tableau</td>
<td>130</td>
</tr>
<tr>
<td>5.1</td>
<td>Synchronised Partitions Table</td>
<td>143</td>
</tr>
<tr>
<td>6.1</td>
<td>Tables and Chairs Simplex Tableau for State 0011</td>
<td>219</td>
</tr>
<tr>
<td>6.2</td>
<td>Tables and Chairs Simplex Tableau for State 0101</td>
<td>221</td>
</tr>
<tr>
<td>6.3</td>
<td>Tables and Chairs Simplex Tableau for State 1100</td>
<td>222</td>
</tr>
<tr>
<td>6.4</td>
<td>Tables and Chairs Simplex Tableau for State 0011 with Unknown Prices</td>
<td>226</td>
</tr>
<tr>
<td>6.5</td>
<td>Tables and Chairs Simplex Tableau for State 1010 with Unknown Prices</td>
<td>227</td>
</tr>
<tr>
<td>6.6</td>
<td>Tables and Chairs Simplex Tableau for State 0101 with Unknown Prices</td>
<td>229</td>
</tr>
<tr>
<td>6.7</td>
<td>Tables and Chairs Simplex Tableau for State 1100 with Unknown Prices</td>
<td>230</td>
</tr>
<tr>
<td>6.8</td>
<td>Tables and Chairs Simplex Tableau for Unknown Prices and Resources</td>
<td>254</td>
</tr>
<tr>
<td>6.9</td>
<td>Tableau for Unknown Prices and Resources, State 1001</td>
<td>254</td>
</tr>
<tr>
<td>6.10</td>
<td>Tableau for Unknown Prices and Resources, State 1010</td>
<td>254</td>
</tr>
</tbody>
</table>
6.11 Tableau for Unknown Prices and Resources, State 0101 . . . . . . . . . . 255
6.12 Tableau for Unknown Prices and Resources, State 0110 . . . . . . . . . . 255
6.13 Tableau for Unknown Prices and Resources, State 1100 . . . . . . . . . . 256
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Portfolio Compliance Constraint Space</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Example Market Pricing of Credits</td>
<td>24</td>
</tr>
<tr>
<td>2.3</td>
<td>Example Minimum Total Compliance Costs</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Example Market Pricing of Compliance Costs</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>Example Approximation of Market Pricing of Compliance Costs</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Probability Distribution of Random Variables $X = \text{Tri}(1,2,4)$ and $Y = \text{Tri}(1,2,3)$</td>
<td>50</td>
</tr>
<tr>
<td>3.2</td>
<td>Probability Distribution of Product of $X = \text{Tri}(1,2,4)$ and $Y = \text{Tri}(1,2,3)$</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Doubly-Linked Node</td>
<td>65</td>
</tr>
<tr>
<td>3.4</td>
<td>Discrete Numerical Random Variable</td>
<td>65</td>
</tr>
<tr>
<td>3.5</td>
<td>Continuous Numerical Random Variable</td>
<td>66</td>
</tr>
<tr>
<td>3.6</td>
<td>Function of a Random Variable</td>
<td>70</td>
</tr>
<tr>
<td>3.7</td>
<td>Piecewise Linear Function of a Random Variable</td>
<td>70</td>
</tr>
<tr>
<td>3.8</td>
<td>Clipped Piecewise Linear Function of a Random Variable</td>
<td>72</td>
</tr>
<tr>
<td>3.9</td>
<td>Sunrise Algorithm Part I</td>
<td>74</td>
</tr>
<tr>
<td>3.10</td>
<td>Sunrise Algorithm Part II</td>
<td>75</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.11</td>
<td>Sunrise Algorithm Part III</td>
<td>76</td>
</tr>
<tr>
<td>3.12</td>
<td>Joint Probability Distribution of Two Independent Random Variables</td>
<td>79</td>
</tr>
<tr>
<td>3.13</td>
<td>Joint Probability Distribution With Level Curves</td>
<td>80</td>
</tr>
<tr>
<td>3.14</td>
<td>Joint Uniform Probability Distribution With Many Level Curves</td>
<td>82</td>
</tr>
<tr>
<td>3.15</td>
<td>Probability Distribution of the Sum of Two Independent Random Variables</td>
<td>83</td>
</tr>
<tr>
<td>3.16</td>
<td>Addition of Two Independent Random Variables, Partitioned</td>
<td>84</td>
</tr>
<tr>
<td>3.17</td>
<td>Subtraction of Two Independent Random Variables</td>
<td>87</td>
</tr>
<tr>
<td>3.18</td>
<td>Product of Two Independent Random Variables</td>
<td>88</td>
</tr>
<tr>
<td>3.19</td>
<td>Quotient of Two Independent Random Variables</td>
<td>89</td>
</tr>
<tr>
<td>3.20</td>
<td>Product of Independent A and Reciprocal of B</td>
<td>92</td>
</tr>
<tr>
<td>3.21</td>
<td>Two Partitions of Reciprocal of Random Variable</td>
<td>93</td>
</tr>
<tr>
<td>3.22</td>
<td>Product of Two Independent Normal Random Variables</td>
<td>94</td>
</tr>
<tr>
<td>3.23</td>
<td>Quotient of Two Independent Normal Random Variables</td>
<td>94</td>
</tr>
<tr>
<td>3.24</td>
<td>Sum of Independent Normal and Bernoulli Random Variables</td>
<td>95</td>
</tr>
<tr>
<td>3.25</td>
<td>Constant to a Random Exponent</td>
<td>95</td>
</tr>
<tr>
<td>3.26</td>
<td>Random Variable to a Constant Exponent</td>
<td>96</td>
</tr>
<tr>
<td>3.27</td>
<td>Random Variable to a Random Exponent</td>
<td>96</td>
</tr>
<tr>
<td>3.28</td>
<td>PHoX Calculation of NPV Example</td>
<td>98</td>
</tr>
<tr>
<td>3.29</td>
<td>Net Present Value</td>
<td>98</td>
</tr>
<tr>
<td>3.30</td>
<td>F(5,2) probability distribution</td>
<td>101</td>
</tr>
<tr>
<td>3.31</td>
<td>F(5,2) probability distribution (zoomed in)</td>
<td>101</td>
</tr>
</tbody>
</table>
6.31 Random Variable Table and Chair Prices . . . . . . . . . . . . . . . . . . . 247
6.32 Joint Price Partition with Block Vertex Projections . . . . . . . . . . . . . 248
6.33 Joint Price Partition with Convex Block Projection . . . . . . . . . . . . . 249
6.34 Joint Price Partition with Rectangular Block Projection . . . . . . . . . . 250
6.35 Joint Probability Distribution of Table and Chair Prices . . . . . . . . . . 251
6.36 Joint Probability Distribution of Table and Chair Prices (Top View) . . . . 251
6.37 Directed Graph for Tables and Chairs with Unknown Prices and Resources 255
1.1 The California Question

The research herein is centered around the specific problem of modeling the cap-and-trade-based pollution abatement policy proposed by the California Air Resources Board (CARB) to meet a portion of their mandated goal of reducing California carbon emissions to 1990 levels by the year 2020. The California law creating this mandate is known as Global Warming Solution Act of 2006, Assembly Bill 32 (see [38]).

In this work a mathematical approach is described addressing the real-world issue of model implementation. This research takes a step beyond work-a-day modeling to describe effect of the introduction of uncertainty as represented by random variables interposed with data and parametric inputs to the model. Ultimately a class of models is identified that can be implemented using the techniques presented here so as to possess the properties of transparency and configurability and are expressible in the presence of uncertain inputs to provide risk and sensitivity analysis without re-expression of the underlying model.

We first present the background of cap-and-trade systems. The modeling approach used in this work builds upon and modeling technique called mathematical programming defined and discussed below. A number of models employing mathematical programming are highlighted as well as models of cap-and-trade systems specifically designed for abate-
ment of an environmental pollutant. The technique of mathematical programming is used far beyond the narrow application of the CARB model as described in this work and consequently the techniques developed in this work are applicable to a much wide problem space.

The work in this dissertation lives in the space spanned by the statement of an abstract model and its specific design. The result of this work and the associated implementation is the production of not just a predictive model but also of a computer application useful to both policy planners (such as those at CARB) and to participating stake holders.

1.2 Background

There are two threads of background to be related. The first is that of cap-and-trade as a tool to regulate the emission of some environmental pollutant. The second is the mathematical technique used to model the outcome from the imposition of a particular cap-and-trade system (see Colby [14]). Examples of such systems include the national Environmental Protection Agency (EPA) Acid Rain Program (ARP) and the Regional Clean Air Incentives Market (RECLAIM) specific to California.

1.2.1 Cap-and-Trade

The idea of employing a market-based mechanism to the problem of environmental damage, at least in a capital sense, is often traced to an article by Coase [13]. There is no mention of cap-and-trade per se, but the idea of government action with respect to market forces as opposed to complex taxation strategies is argued. The idea presented by Coase is referred to as the Coase theorem (Hoffman [28]) which states that two parties to an externality will strike a mutually advantageous bargain in the absence of transaction costs as long as there is perfect knowledge, a competitive market and no wealth effects.
Another early work by Weitzman [55] considers the trade-off between control of some externality such as atmospheric pollution by direct taxation or quantity cap. The issue in the Weitzman paper is one of uncertainty in the outcome of each type of instrumentation. The conclusion as to which is best is that it depends on the marginal costs of abatement for the regulated parties. An optimal mixture of taxation and quantity capping is suggested by Mandell [33].

While Coase, Weitzman and Mandell discuss caps in abstract terms our interest is in modeling their effect once specifics are known or proposed. It is instructive to look at the attempts by others at modeling emission abatement (cap-and-trade) markets and different techniques used to implement these models.

1.2.2 Modeling Techniques of Environmental Regulation

There are many relevant modeling techniques. One source for modeling techniques may be found in control theory texts e.g. (Sontag 1998 [49]) or in texts on optimization e.g. (Boyd 2004 [5]). We will favor the latter approach in this work.

The modeling technique specifically employed in this work is generally referred to as mathematical programming (Greenberg 1995 [26]). This approach has its roots in the technique of linear programming (LP) first invented during World War II by George Dantzig (born in Portland, Oregon incidentally) and declassified in 1947 [24]. According to Greenberg there are several standard techniques comprising mathematical programming. The techniques include linear programming, non-linear programming (NLP), stochastic programming (SP), dynamic programming (DP), and stochastic dynamic programming (SDP). More recently the term convex optimization (CO) is being used to cover a range of optimization techniques [5].

In Greenberg, we have a comprehensive review of mathematical programming techniques applied to the area of environmental quality control. The recent thesis of Nelson
[36] employed linear programming to the issue of carbon regulation and energy policy. A technical report by Caldwell [10] on behalf of the Pew Center on Global Climate Change appears to use linear programming specifically to model a possible carbon market in the midwestern United States. This particular work most closely matches the initial technique used in this proposal, but applied to a different market with differing policies.

Another review article, this time by Cooper et al [15], focuses on mathematical programming as it relates specifically to air quality control issues in the United States. Particular attention is paid to sulfur and nitrogen dioxides. In Cooper the term goal programming is described as an extension of linear programming through incorporation of constraint violation to minimize total cost. The goal programming model described is still characterized by Greenberg as a linear programming model. Cooper shows that stochastic programming (in the simplified sense of univariate normal random variables) is a non-linear program and allows for a specific level of uncertainty. To incorporate probabilities of constraint violations, Cooper shows that chance constrained programming (CCP) may be introduced. The stochastic models reviewed in Cooper are developed mostly for direct control strategies.

According to Greenberg there are a number of standard models available for modeling environmental quality control. The MARKAL-MACRO model [34], for example, developed at the Brookhaven National Lab is intended to help assess energy strategy on a regional or national basis. It is a linked energy-economy model employing a linear programming model (MARKAL) and a general equilibrium macro economic model (MACRO). Both models assume perfect foresight with respect to technologies and economic conditions. The MARKAL model uses linear programming and the MACRO model uses non-linear optimization. The linkage between MARKAL and MACRO is through an economy-wide production function. It is written in GAMS (generalized algebraic modeling system).

In his text Nordhaus [37] describes his DICE model which is an acronym for Dynamic Integrated model of Climate and the Economy. The DICE model uses optimal
Ramsey growth [43], a utility maximizing investment strategy according to Arrow [2]. The DICE model operates globally in steps of 10 years expressing its output through the Cobb-Douglas production function. The Cobb-Douglas production function is a non-linear function that relates output of some economic good to the inputs of capital, labor and technology and is discussed in many economic texts such as Gujarati [27].

According to Greenberg, Morrison and Rubin use LP in their model called OMEGA for acid rain control policies through \(SO_2\) emission reduction. There is a U.S. national model for greenhouse gas abatement economic impacts using a computable general equilibrium model known as IGEM (see Jorgenson [29]). An overview of the cap-and-trade economic implications nationally using a game-theoretic model, FEEM-RICE is given by Buchner [8].

One recent paper of Carmona et al [11] employs the technique of mathematical programming involving a stochastic process to model much of the EuETS and the electricity market in Texas. The intention of the Carmona paper is to serve as a mathematical guide for policy makers.

There exist three studies that provide the estimates of the compliance cost of AB32, one by the CAT of the State of California (see CAT [46]), a study by the Center for Clean Air Policy (CCAT [52]), and finally a study by Roland-Holst [45]. According to Stavins et al. [50], all three studies conclude that the resulting offsetting benefits of AB32 will outweigh the direct cost of the regulation, resulting in zero net compliance cost. The authors are critical of this conclusion and point out several known causes that result in underestimation of the costs. Wassmer [53] also provides critical comments on some of the variables employed in the CAT study, such as an unrealistically high gas tax ($2.57 per gallon) and inclusion of benefits realized outside the state of California.

The modeling technique employed by Roland-Holst [44] is that of general equilibrium. According to Greenberg this technique may or may not fall under the rubric of mathemati-
There are commercial organizations that provide modeling services to government and industry in the area of climate change management and related risk management such as ICF International and DVN (Det Norske Veritas). Unsurprisingly any policy models produced by these organizations for their clients are proprietary and not usually available for public review.

1.2.3 Requirements for the CARB Model

A consultation with the California Air Resources Board (CARB), the agency charged with the design of this cap-and-trade system and complementary command-and-control regulations, revealed the need for a real-world on-line, multi-user, modeling application that, despite operating with both imperfect and incomplete information, evaluates the effects of various policy profiles in a manner that is interactive, verifiable and transparent.

1.2.4 Modeling Background for the CARB Model

Real world modeling necessitates the solution of a wide range of interacting problems. According to the efficient market hypothesis of Fama [21] the cost of getting prices to reflect all available information is zero. In essence a financial market is a kind of optimization engine. Modelers must correctly and concisely describe precisely how the market economics interact with the mathematics of optimization coupled with the statistics of tracking uncertainty through the processing of uncertain data. In addition, cutting across these interactions are implementation issues that accompany the design of concordant data structures and their algorithmic transformation both within the model and the implementing software.

The presumption is that the cap-and-trade system proposed by CARB will include an organized market to facilitate trading between parties. Perhaps models developed for financial markets are applicable to cap-and-trade. This cross-pollination of ideas is not so
clearly available because in contrast to a traditional financial market trading in securities, a cap-and-trade market is created by mandate and infused with rules designed to achieve outcomes desirable to planners, regulators and their constituents (see Napolitano [35]). A fundamental difference between a cap-and-trade market and a modern financial market is that of price replication (see Cerny [12]). Pricing a financial security is often done by creating a portfolio of other securities that are designed to perform similarly in all foreseeable circumstances as the given security (see Follmer [23]). If the prices of the security in question and its replicating portfolio differ, an arbitrage opportunity is created and presumably exploited, bringing the two prices back in line according to many standard financial texts including Follmer. Price replication is not possible in a market such as a one commodity cap-and-trade market (see Karoui 1995 [30]). Markets such as this are termed incomplete and the study of such markets is an area of active current research (Platen 2006 [42]). In fact, according to Magill 2002 [32], a single asset market is necessarily incomplete.

Since none of the modeling approaches surveyed fully meets the requirements suggested by CARB especially when we include the added element of modeling in the presence of uncertainty we must start our model building by asking some fundamental questions; What is the right set of data structures and right set of interactions between data structures for a cap-and-trade modeling application? How can modeling data be represented so that the widest possible range of policy parameters can be easily and veritably applied? How can uncertainty be reliably captured and faithfully represented? Furthermore, any application must respect economic principles and make plain any departures. The replies to these questions and considerations must further result in an application design that can be realized on real hardware.

In the following sections a modeling approach and the corresponding preparatory work is described. Here, as in all instances, the foundational design of the system that will ultimately power the model developed for this work benefits greatly from the seemingly
pedestrian material initially developed. It is the author’s opinion that the foundational work of a project only looks mundane until the superstructure takes shape.
2.1 A Modeling Approach

Which approaches will be used? The general equilibrium modeling approach, for example, is both venerable and ponderous [32]. It rolls together 200 years of classical economic theory. It may be described as the convergence of principles toward a real world solution. Behavioral economics suggests [17] that classical economics may leave much to be desired when predicting real outcomes. General equilibrium models used exclusively to study sustainability impacts tend to suffer from simplifications and behavioral assumptions necessary in order to ensure the existence of an equilibrium (see Scrieciu [47]).

Modeling through systems of differential equations will also not be employed. This is largely a matter of taste. According to Greenberg [26] the mathematical programming model is defined by four ingredients. The first is a finite set $X$ of decision variables. There must be a constraint function $g$ that maps $X$ into $\mathbb{R}^m$. The constraint function may be bounded, as in $L \leq (x, g(x)) \leq U$. Logical bounding includes a non-negativity requirement, a capacity limit and a demand requirement. Lastly there must be an objective function $f$ that maps $X$ into $\mathbb{R}$.

Along with the choice of modeling approach is the choice of application development approach. While the actual software involved in running the policy model described here
is beyond the scope of the work presented here, the software’s design is pertinent. The approach here is to design and independently construct a series of models from elementary to full featured. The completion of each stage in this concerted approach highlights areas of strength and weakness to be addressed in design and construction of the next stage. The first stage incorporates the processing of raw data into a common format, its combination with simple policy parameters is then fed into a market simulator yielding a portfolio result. This is the foundation stage of the model.

The discussion herein is specifically about the design of a model for evaluating proposed regulatory schemes assuming an ideal pollution abatement (cap-and-trade) market. The work herein does not include the actual construction details of the associated software application, but acknowledges and derives advantage from the parallel construction of this application. The goal is to recognize and bridge the mathematical divide between an abstract model and its real world implementation.

While there is a software application being developed in parallel with the work presented here which we will refer to as the reference implementation there should be no confusion between this and a commercialized software product.

Below described is the design of the elementary model and some extensions. This description will highlight where the mathematically interesting improvements that will be made.

2.2 Overview of the Elementary Model

To begin, we create an elementary model reflecting a simplification of the real world situation. The simplified situation is the following. A governing body, in this case CARB, has determined that there is too much of a particular class of pollutants, in this case atmospheric greenhouse gases (GHG). Rather than require emissions reductions by some
percentage, say 10%, of each emitter CARB estimates the amount of future emissions and creates *allowances* (right to emit certificates) covering 90% of the expected emissions. Then CARB auctions off or otherwise allocates these allowances at the beginning of a *compliance period* and then audits the emitters at the end of that period. The emitters must present an allowance certificate for every unit (metric ton) of GHG they emitted or pay a significant penalty.

It is assumed that CARB set up a market where allowances granted to certain *capped* industries may be traded like a commodity. Ideally the market will function is such a way that allowances will be available to emitters who need them at minimal cost.

The degree to which real financial markets depart from their ideal function is not addressed in the foundation stage. Competition, gamesmanship and uncertainties of all flavors are likewise left for later stages. All such issues are suppressed by assuming that there is only one super-emitter under regulation. In terms of game theory this represents a one player game. The goal of this single super-emitter player is to minimize the cost to themselves in the face of new emission control regulation.

Modeling this elementary stage may appear as straight forward. The hidden feature is that the details of the design bias all that follows.

The cap portion of cap-and-trade is the number of allowances auctioned or allocated by the regulating body. The difference between the number allowances available and the business-as-usual (BAU) emissions drives the whole cap-and-trade scheme.

The trade portion of cap-and-trade is the trading of allowances between GHG emitters. We assume emitters behave as a single super-emitter entity for the purposes of the elementary model, but it’s instructive to realize that individual emitters may choose to reduce their emissions without trading any allowances or may choose to trade allowance directly with other emitters without using the market mechanism. The existence of a market mechanism implies that low (ideally zero) cost trading of allowances is an available
option to emitters. Individual emitters with more allowances than actual emissions may sell their surplus allowances.

The units of GHG are denoted $tCO_2e$ and one allowance is equivalent to one $tCO_2e$. The *equivalent* concept says that if you emit one tonne of a greenhouse gas like methane which is 23 times as potent in terms of its greenhouse effect as carbon dioxide that this is equivalent to emitting 23 tonnes of carbon dioxide (IPCC [40]). Incidentally, some gases are many thousands of times as potent as carbon dioxide (IPCC [40]).

Emitters may reduce their emissions through reduced economic activity, efficiency measures applied to existing emission generation processes or through a technological change in the emitting process itself.

Emitters may also reduce their apparent emissions by engaging in emission capture and storage. Some entities may specialize in capture and storage or conversion of GHG’s into more benign forms such as the combustion of captured methane into carbon dioxide, a net improvement in terms of greenhouse effect.

A mechanism exists to allow entities that specialize is capture and/or conversion of GHG emissions to more benign forms to create then equivalent of a regulator issued allowance called an *offset*. In part because offsets have the potential to play the role of counterfeited allowances they are closely monitored and a significant portion of the regulation rules applied directly to offsets.

Each emission reduction measure described above is represented in the elementary model by a cost per tonne and quantity per year of GHG reduction. These measures are referred to collectively as the *supply* of emission reduction opportunities. Notice that there is no accompanying *demand* in the traditional sense. Our one player super-emitter acquires all the allowances and engages in some of the available emission reduction activities and possibly purchases offsets as appropriate. The difference between business-as-usual emissions and issued allowances is herein called *shortfall*. 


The situation is more complicated than first blush would imply since costs and quantities of emission reduction opportunities change over the years as do emissions and available allowances. The system may be scoped over some number of years with the question of what happens after that left unanswered. In the case of AB32 the years are 2012 through 2020 and allowances auctioned and allocated in any given year will likely decrease giving time for emitters to adjust to tightening conditions as the overall goal is approached.

2.2.1 Processing Raw Data in the Elementary Model

Two basic activities must occur while processing the raw emission abatement data. The first is the calculation of cost per $tCO_2e$ and the second is the quantity of reduction possible at that cost. Collectively the data form a supply curve often called a marginal abatement cost or MAC curve. Individual project classes must be chosen to be fine grained enough for realistic results, but not so small that the problem becomes computationally intractable.

The costing of $tCO_2e$ may be done in any suitable manner. Traditionally \[39\] such costs are computed by dividing the net present value of all expenses by the similarly discounted output, $tCO_2e$ in this case. This process is called levelizing and is used in our elementary model.

Finding the capacity of $tCO_2e$ is often straightforward unless there are contingencies. As with cost computing, this is found on a per-project basis.

To incorporate variation of cost or capacity over time many methods exist. A simple method is to find both a current estimate for raw data and a corresponding future estimate. Each estimate is processed to find cost and quantity values for each project. A straight line interpolation may be computed through each pair of values for each project. From this fitted line data cost and quantity information may be stated for any year.

Any number of policy parameters may be incorporated into the cost and quantity calculations parametrically. Alternative costing algorithms may be triggered, different discount
rates employed, and project lifetimes may be overridden.

Further policy parameters that affect the background assumptions of fuel prices or availability of funds for technical improvements beyond abatement investments may be incorporated. Such parameters can create different economic climates to stress the model and thereby shed more light on the performance of the overall policy in question.

2.2.2 Applying Policy Features to the Elementary Model

A significant policy parameter is the choice of which industrial sectors to subject to an emissions cap. The raw data must be available for all candidate sectors. A filtering process must be in place to limit the data to only sectors that are subject to the cap. The model simply classifies the data as capped or not. There are further classifications such as offsets which will be discussed below.

What can be said of the data or done to it is determined by what can be known about it. In this case the data consist of a set of abatement projects or classes of projects with all their original properties intact. These properties include the location of the project, such as California, its sector such as transportation, and possibly a sub-sector or two. Any number of property tags may be associated with each abatement project. In this manner subsets of the cost and quantity information may be identified and altered.

An example of the kind of policy feature that may be naturally applied is a price floor. The physical mechanism by which prices are altered by policy must be known, but in their simplest form a subset of items, such as all items within a sector may be altered so that none is lower than a specific value. Similarly quantities of a particular subset of projects may be constrained in some manner.

Put succinctly, there is an opportunity to apply a transform on a set that may be expressed as,
Any policy feature expressible as such a transform is applicable.

2.2.3 Simulating a Cost Effective Market

Once all costs and quantities for all items at each time within the model are known then an cost minimization algorithm may be applied. The result of this cost minimization is a portfolio detailing the quantities of items purchased at specific times.

In the foundation stage all items represent projects that generate a constant supply of credits each year of ownership and cost the same per credit each year. In subsequent stages credits will be allowed to be purchased directly (possibly from other markets) and credit generation will be allowed to be non-constant. The constancy of cost per credit may also be relaxed. There is less call for relaxation of constant cost per credit since financial arrangements, like a mortgage on a home, tend to even out costs. Note however that the size of the initial investment may form a barrier to entry.

The portfolio optimization in the foundation stage is expressible as a linear programming [24] exercise,

\[
\min_x \left\{ c'x \mid Ax \leq b \right\}; \quad c, x \in \mathbb{R}^{T \times N}, \quad A \in \mathbb{R}^{k \times T \times N}, \quad b \in \mathbb{R}^k.
\]

where \( x \) is the desired portfolio, \( c \) is total cost of ownership for each item at each time, \( A \) and \( b \) are respectively the linear constraint matrix and associated constraint vector described below. Also, \( T \) is the total number of model years, \( N \) is the number of items.
being modeled and \( k \) is the number of discrete modeling constraints.

A simple example will make the situation plain. Consider two project items \((N = 2)\) available in any of three years \((T = 3)\). Suppose further that their cost-per-ton \(P\) and quantities \(Q\) are,

\[
P = \begin{pmatrix} 10 & 7 \\ 11 & 8 \\ 12 & 9 \end{pmatrix} \quad Q = \begin{pmatrix} 5 & 2 \\ 6 & 3 \\ 7 & 4 \end{pmatrix} \quad d = \begin{pmatrix} 3 \\ 0 \\ 22 \end{pmatrix}
\]

If one unit of the first item is purchased in the first year it will generate 5 credits in each year for a total of 15 and cost \(3 \times 10 = 30\) dollars in total. As a result the cost vector \(c\) is then a column-stacked version of \(P\) which has been updated to reflect lifetime costs rather than annual costs as follows,

\[
c = \text{column stack of} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} 10 & 7 \\ 11 & 8 \\ 12 & 9 \end{pmatrix} = \begin{pmatrix} 30 \\ 22 \\ 12 \\ 21 \\ 16 \\ 9 \end{pmatrix}
\]

The \(A\) matrix and \(b\) vector are constructed as a vertical stacking of block components corresponding to required model features. In this foundation stage there are three such features and as a result we will have,
The first constraint is that the model cannot sell any items. The portfolio vector $x$ reflects the end result of all market activity in all years. Put another way, project items cannot absorb credits as they are in the business of selling them. This is the non-negativity constraint which is expressed as,

$$A = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad A \in \mathbb{R}^{TN \times TN}, b \in \mathbb{R}^{TN}$$

The second constraint requires that the model not purchase any items in excess of their given capacity, the capacity constraint. This constraint is represented by the block components $A_2$ and $b_2$. Recall that the portfolio $x$ represents the purchase of generators or credit versus direct purchase of credits. A sequence of partial sums must occur to form the proper inequality,
\[
A_2 = \begin{pmatrix}
1 \\
1
\end{pmatrix} \otimes \begin{pmatrix}
1 & 1 \\
1 & 1 & 1
\end{pmatrix}
\quad b_2 = \begin{pmatrix}
5 \\
6 \\
7 \\
2 \\
3 \\
4
\end{pmatrix}
\]

where \( A_2 \) is a diagonal stacking of the unit lower triangular matrix which will create the appropriate partial sums for each item and \( b_2 \) is the column-stacked version of the capacity matrix \( Q \).

The third constraint requires that the given number of credits be held in each given year. If we compute a double accumulation for each portfolio item it may be properly compared to the cumulative shortfall values as in,

\[
A_3 = -\left(1 \ldots 1\right) \otimes \begin{pmatrix}
1 & 1 \\
1 & 1 & 1
\end{pmatrix}^2 
\quad b_3 = -\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
1
\end{pmatrix} \begin{pmatrix}
3 \\
0 \\
22
\end{pmatrix}
\]

where the \( \begin{pmatrix}1 & \ldots & 1\end{pmatrix} \) matrix is \( \mathbb{R}^{1 \times N} \) and assumes that all items contribute to satisfying the overall shortfall. This assumption will be relaxed in later stages as other considerations are included in the model.

There is a subtlety in this formulation. If the \( A_3 \) matrix only contained a single accumulation then the associated vector \( b_3 \) would be equivalent to the shortfall and not the cumulative shortfall. The formulation would be,
The practical difference is that any excess credits generated in previous years would not be applicable in later years. In policy terms the $A_3/b_3$ case is said to presume banking of credits while the $A_{3}^{alt}/b_{3}^{alt}$ case disallows banking. It is necessary to be possible to switch between these two cases as a policy parameter or even mix them, allowing certain classes of credit generators to contribute bank-able credits and while others are not banked. Such a partitioning may be between capped sector credit generators versus offset credit generators.

With all the elements in place, the information may be fed into a linear programming engine and the following minimal cost portfolio vector $x$ (and corresponding matrix $X$) is observed,

$$A_{3}^{alt} = - \begin{pmatrix} 1 & \ldots & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad b_{3}^{alt} = - \begin{pmatrix} 3 \\ 0 \\ 22 \end{pmatrix}$$

$$x = \begin{pmatrix} 5 \\ 1/2 \\ 0 \\ 2 \\ 1 \\ 1 \end{pmatrix} \quad X = \begin{pmatrix} 5 & 2 \\ 1/2 & 1 \\ 0 & 1 \end{pmatrix}$$

The overall minimal cost is then computed to be,
\[
\text{cost} = c^T x = \begin{pmatrix} 30 \\ 22 \\ 12 \\ 21 \\ 16 \\ 9 \end{pmatrix}^T \begin{pmatrix} 5 \\ \frac{1}{2} \\ 0 \\ 2 \\ 1 \\ 1 \end{pmatrix} = \$228
\]

The cumulative number of credits generated in each of the \( T = 3 \) years is then found as,

\[
\text{cumulative credits} = -A_3 \times x = \begin{pmatrix} 7 \\ 15.5 \\ 25 \end{pmatrix}
\]

where we notice for this example that in the final year exactly the right number of cumulative credits are generated as required. As new modeling stages are added this property may no longer hold.

### 2.2.4 Introducing a Class Constraint to the Elementary Model

In our system, abatement activities come from capped sectors and offset sectors. Let \( z_{\text{cap}} \) denote the total number of abatement activities generated within a capped sector, and \( z_{\text{off}} \) those generated from offset sources. Often, there is a limit imposed on the use of offsets, so suppose that it is capped at 3% of emissions. Let \( e \) denote the emissions level and \( d \) denote the shortfall in a given period. Since the total supply of abatement activities gen-
erated must meet the shortfall, $z_{cap} + z_{off} \geq d$. In other words, the compliance constraint is given by,

$$z_{cap} \geq d - z_{off} = d - 0.03e.$$

The region $F$ in figure 2.1 shows the feasible combinations of $z_{cap}$ and $z_{off}$.

In addition, in order to allow for offset sources to be turned off, applying the technique employed for the mortality constraint, define

$$m_i = \begin{cases} 
\infty & \text{if type } i \text{ is an immortal credit generator}, \\
0 & \text{if type } i \text{ is an offset (turns it off)}, \\
1 & \text{if type } i \text{ is a rental item}.
\end{cases}$$

The allowed offset as a percentage of emissions level may be different for different
years. so let $\rho_{t,i}$ denote the percentage for sector $i$ in year $t$. To ease the notation, let the diagonal function $diag$ be such that:

$$\rho = \begin{pmatrix} 0.03 \\ 0.02 \\ 0.02 \end{pmatrix} \quad \text{then} \quad diag(\rho) = \begin{pmatrix} 0.03 \\ 0.02 \\ 0.02 \end{pmatrix}.$$

Then, the compliance constraint $A_{(4)}x \leq b_{(4)}$ for $T = 3$ can be defined with,

$$A_{(4)} = -\bigodot_{i=1}^{N} L_T L_{m_i} \quad b_{(4)} = -(d - diag(\rho) e).$$

where we use the following convenient construct for banded lower unit triangular matrices,

$$\mathcal{L} = \begin{pmatrix} 1 \\ 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \mathcal{L}_{(2)} = \begin{pmatrix} 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

and introduce the block matrix horizontal stacking operator, $\Theta$, defined by,
2.2.5 Introducing a Foreign Market to the Elementary Model

Capped-sector companies can purchase excess allowances directly from other capped or offset sectors through the allowance market at a market price. The sector-specific data represents the excess abatement capacity of the sector that will generate allowances for sale. Such abatement activities are regarded as being "rented". In any given year, those who are subject to the emissions cap enter the allowance market with a certain shortfall. The greater the shortfall is, the higher the price will be. The modeling challenge comes from this fact that the market price of allowance varies, depending on the overall shortfall.

As configured, the linear programming engine cannot handle variably priced items. Described below is an approximation whereby prices are fixed, but the overall cost closely matches the case if prices were allowed to vary.

Consider an example containing two types of abatement activities in a single year. The Marginal Abatement Cost (MAC) of the first type is 10 dollars per unit, with a capacity of 3 units of allowances per year, and the second type has the MAC of 15 dollars per unit and capacity of 5. The figure 2.2 summarizes these values.

The least expensive activities will be purchased first in the form of excess allowances. Let $x$ represent the total number of abatement capacity to be purchased. The figure 2.3 shows the minimum total compliance cost $TC_{min}$ as a function of $x$.

On the other hand, figure 2.4 shows an alternative total compliance cost curve based on the market price of allowance ($TC_{mkt}$). In deriving $TC_{min}$, we assume that the allowance market is perfectly competitive and that the price ($p$) equals the MAC of the last unit of
Figure 2.2: Example Market Pricing of Credits

Figure 2.3: Example Minimum Total Compliance Costs
Therefore, in this example, \( p = MAC_{x=8} = 15 \).

In both cases the slope of each line segment represents the price of the underlying abatement activity. The difference is that, in the "market pricing" case, each line segment, when extended, intersects the origin rather than forming a piecewise continuous curve.

For linear programming, a continuous piecewise linear convex curve is required where each segment represents an underlying abatement activity. The convexity requirement ensures that the order in which the allowances are purchased remains fixed. On the other hand, market pricing requires that certain levels of cost are reached as if the cost of the last unit of allowance purchased represents the cost of all allowances purchased in a given period (one year). We generate an approximate total compliance cost curve (TC) by re-pricing all the allowances in the market for a given year and repeating the process for each model year. Here is the process in more detail.

The approximation curve is derived through an optimization problem with a set of linear inequality constraints. The chosen objective here is the sum of the absolute difference in the area under each segment of each curve: the approximation curve and the original market price curve. To demonstrate this process, we extend the current example to introduce additional abatement activities. Suppose that there are 6 types of activities, with
different marginal abatement costs in figure 2.5. A possible approximation curve (thin line) is overlayed on the computed market cost curves (thick lines).

The fitted approximation curve is continuous piecewise linear with breakpoints at \((x_i, y_i)\). Because \(x_i\) denotes the cumulative capacity, we can also define \(x_i\) as,

\[
x_i = q_1 + \cdots + q_i
\]
where \( q_i \) denote the capacity of abatement alternative \( i \), as shown in figure 2.5. To ensure the proper convexity the following condition is introduced,

\[
\frac{y_1}{q_1} \leq \frac{y_2 - y_1}{q_2} \leq \cdots \leq \frac{y_n - y_{n-1}}{q_n}
\]

Let us define:

\[
y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad q = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}, \quad p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}
\]

\[
x_i = \mathcal{L}q
\]

The convexity condition may now be restated as a set of linear inequality constraints for the linear programming about to be solved. Notice first that,

\[
\frac{y_2 - y_1}{q_2} \leq \frac{y_3 - y_2}{q_3},
\]

therefore,

\[
q_3(y_2 - y_1) \leq q_2(y_3 - y_2)
\]
hence

\[ 0 \leq q_3 y_1 - (q_2 + q_3) y_2 + q_2 y_3 \]

The convexity constraint can then be expressed as,

\[ 0 \leq V\mathbf{y} \]

where,

\[
V = \begin{pmatrix}
-(q_1 + q_2) & q_1 \\
q_3 & -(q_2 + q_3) & q_2 \\
q_4 & -(q_3 + q_4) & q_3 \\
& & & & & & \\
& & & \ddots & & & \\
& & & & \ddots & & \\
& & & & & \ddots & \\
& & & & & & \ddots \\
\end{pmatrix}
\]

The area under each segment of the market cost curve may be expressed in vector form as,

\[
\mathbf{r}_{mkt} = \begin{pmatrix}
\mathbf{r}_{mkt\ 1} \\
\cdots \\
\mathbf{r}_{mkt\ N}
\end{pmatrix}
\]

\[ r_{mkt\ i} = p_i \frac{x_i + x_{i-1}}{2} q_i \quad x_0 = 0 \]

Similarly the area under the \( i^{th} \) segment of the continuous approximation curve may be expressed as,
\[ \mathbf{r}_{\text{min}} = \begin{pmatrix} r_{\text{min} 1} \\ \vdots \\ r_{\text{min} n} \end{pmatrix} \]

where

\[ r_{\text{min} i} = \frac{y_i + y_{i-1}}{2} q_i \]

and \( \mathbf{r}_{\text{min}} \) may be decomposed into the product of a matrix \( \mathcal{L}(2) \) and a vector \( \mathbf{y} \) as,

\[ \mathbf{r}_{\text{min}} = \mathcal{L}(2) \mathbf{y} \]

where

\[ \mathcal{L}(2) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ \vdots \\ 1 & 1 \end{pmatrix} \]

The objective is then a function of \( \mathbf{y} \). It represents the sum of absolute differences of the area of each trapezoidal region under the two cost curves expressed as,

\[ ||\mathcal{L}(2) \mathbf{y} - \mathbf{r}_{\text{mkt}}||_1 \]

where, for example,

\[ \left| \begin{pmatrix} 5 \\ -2 \\ -3 \end{pmatrix} \right|_1 = |5| + |-2| + |-3| = 10 \]

The full statement of the optimization problem to find the approximating market cost
curve is the,

\[ \min_y \left\{ \|L(2)y - r_{mkt}\|_1 \mid 0 \leq Vy \right\} \]

This is not in a form suitable for implementation. As with the main model optimization
the objective must be stated as a linear product such as \( \mathbf{c}'y \). By restating the 1-norm (\( || \circ ||_1 \))
the optimization statement becomes,

\[ \min_y \left\{ \|\mathbf{y}' \mid -\tau \leq L(2)y - r_{mkt} \leq \tau, 0 \leq Vy \right\} \]

Note that here a new variable \( \tau \) is introduced, yet the statement is not quite finished
since the objective no longer involves the main variable \( y \). What must be done is to stack
the two variables \( \tau \) and \( y \) and then make commensurate changes to the condition matrices
as follows,

\[ \min_y \left\{ h'\tilde{y} \mid \tilde{L}(2)\tilde{y} \leq \tilde{r}_{mkt} \right\} \]

where, for example,

\[
\begin{align*}
\mathbf{h} &= \begin{pmatrix} \mathbb{O}_n \\ \mathbb{I}_n \end{pmatrix} \\
\tilde{y} &= \begin{pmatrix} y \\ \tau \end{pmatrix} \\
\tilde{L}(2) &= \begin{pmatrix} L(2) & -\mathbb{I}_n \\ -L(2) & -\mathbb{I}_n \\ -V & \mathbb{O}_n \end{pmatrix} \\
\tilde{r}_{mkt} &= \begin{pmatrix} r_{mkt} \\ -r_{mkt} \\ \mathbb{O}_n \end{pmatrix} \\
y, \tau &\in \mathbb{R}^n
\end{align*}
\]
By solving this optimization problem for each year and for each linked market the re-pricing of the credit generating items in each case is complete. These new prices are used within the model to approximate the effect of market pricing.

2.2.6 Introducing Mandatory Requirements to the Elementary Model

Mandatory requirements are stated in terms of cumulative capacity matrix $E$. In the language of our model, a mandatory requirement sets a lower limit on the number of certain types of abatement activities that must be generated in certain sectors. An example of such a class is the Renewable Portfolio Standard (RPS) in a particular region.

An instructive example: consider wind energy in a given region as a type of mandatory constraint within a larger renewable portfolio standard (RPS) for that region. Let $G_j$ represent the vector of cumulative capacity values that must be met for wind energy among its constituent items.

For $T = 3$, suppose that the cumulative capacity constraint is given by $G_{1,j} = 100$, $G_{2,j} = 150$, and $G_{3,j} = 180$. Then vector $G_j$ is given by,

$$ G_j = \begin{pmatrix} 100 \\ 150 \\ 180 \end{pmatrix} $$
Suppose further that $N = 3$ and that the first two items are wind energy-based abatement activities whereas the third item is some other type of abatement activity. Then the mandatory constraint type $j$ is given by,

\[
\begin{pmatrix}
1 & 1 & 0
\end{pmatrix} \otimes \mathcal{L}_T \mathbf{x} \geq G_j
\]

or more explicitly by,

\[
\begin{pmatrix}
1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
X_{1,1} \\
X_{2,1} \\
X_{3,1} \\
X_{1,2} \\
X_{2,2} \\
X_{3,2} \\
X_{1,3} \\
X_{2,3} \\
X_{3,3}
\end{pmatrix} \geq
\begin{pmatrix}
100 \\
150 \\
180
\end{pmatrix}
\]

There are $R$ such mandatory constraints. An indicator matrix $I$ is defined to assign any of the $N$ item-types into one of the constraints. There is one column for each item-type and one row for each mandatory constraint. In any location a one represents membership in the constrained class for the current mandatory constraint (row). Therefore, for

\[
I \in \mathbb{R}^{R \times N} \quad \text{and} \quad G \in \mathbb{R}^{T \times R} \iff g \in \mathbb{R}^{T \times R}
\]
the mandatory constraint is given by $A_{(5)}x \leq b_{(5)}$ where

$$A_{(5)} = -I \otimes L_T$$

and

$$b_{(5)} = -g.$$

## 2.3 Implementation Considerations

According to the CARB request the model must be accessible to multiple users through a web based application and, to further the policy discussion among users, all modeling assumptions must be readily discoverable.

There are several choices available for implementing a secure, multi-user web based application and the discussion of these choices is beyond the scope of this work. What will be discussed below are the mathematical considerations need to realize such an application.

Even for an experienced software developer hitting on the right development path for a novel application is a challenge. In developing the reference implementation for the sharp variable mode a number of different approaches were investigated, all focused on the idea of maximizing configurability. The idea that yielded the best response and admitted a clear development path to meet all the design requirements came through establishing two design principles. The design principles set the stage for discussing the middle ground between abstract model description and application implementation.

The first design principle became that the core application must have no knowledge about its specific purpose. This implies that all application-specific information and program code must live outside the application core. In the current independent reference implementation this dichotomy is manifest in the code being written in the Java programming language and all configuration code being written in the Python programming language. It needs to be emphasized that this principle is not a matter of vacuous semantics. The
consequence of this design principle is the development of a stand-alone core which in fact represents a modeling development environment. A significant reason a new development environment needed to be developed rather than choosing any one of a number of quality off-the-shelf development environments stems from the second design principle, following.

The second design principle is that the configuration portion, the cerebrum of the overall application, must be written so that a non-programming user may readily recognize their model in the code. In part this implies that a certain amount of syntactic sugar must be supported by the core application. On a basic level this means that such common operations as addition, multiplication and exponentiation must be sensible to the operands. In the application, for example, a vector can be represented by any variable name such as $a$ and another as $b$ then vector multiplication of $a \times b$ will result in the expected value. If, instead, $a$ and $b$ are similar multi-dimensional arrays then their product must result in a similar multi-dimensional array. The rules for combining through basic operations of all combinations of variable types must be carefully designed and implemented in the core application. In the C++ programming language, as well as other typed languages this feature is referred to variously as operator overloading and polymorphism. This issue is addressed by the National Institute of Standards and Technology (NIST [22]) as a significant limitation for the Java programming language in its use for numerical computation.

For the purposes here, Java is very well supported for web based application development and its operator overloading deficiency is overcome through its support for an embeddable version of the Python language, a language with excellent support for operator overloading. This combination is used in the reference implementation.

To give the reader a better sense of the issues involved in navigating the middle ground between an abstract model and its implementation, two detailed examples of their usage from the reference implementation will be presented as well as a selection of supported data types and some brief examples of their usage.
2.3.1 An Example Demonstrating Design Issues

The following is an illustrative example from the reference implementation. Though the reference implementation specifics are outside the scope of this work, its contents provide a rich source of explanatory material. In this example the \textit{TABLE} variable contains a programmatic \textit{dictionary}. It represents a table of data with labeled columns each represented by an array of values (numbers or strings). When \textit{TABLE} is presented with a valid label it, and all other such \textit{STRING\_MAPs} will return the eponymous column (array). Every variable in the following Python listing represents an array except for discount rate \( R \).

```python
from Constants import *
from Functions import *

R = Lookup("UserConfig", "Discount Rate")
fin_year = TABLE["FinancialYear"]
lifetime = TABLE["ProjectLifetime"]
periods = TABLE["Period1", "Period2", "Period3"]
accums = TABLE["Accumulation1",
               "Accumulation2",
               "Accumulation3"]
cap_costs = money(TABLE["Capital Costs"], fin_year)
op_costs = money(TABLE["Operating Costs"], fin_year)
c__ha = PLDCF(R, lifetime, periods, accums)
co2 = CO2__C * c__ha
cost = cap_costs + op_costs * NPV(R, lifetime)
```

35
The core application defines various generically useful functions such as,

\[ \text{money}, \text{PLDCF}, \text{NPV} \]

The \textit{money()} function will access pre-configured consumer price index values, a pre-configured \textit{current year} and given an array of nominal dollar values and a similar array of nominal years convert the given dollars to real dollars. The \textit{PLDCF()} function if given a financial discount rate and three arrays collectively representing an array of piecewise linear functions, returns the net present value of each discounted cash flow. This is a standard business calculation implemented in the core application and so does not violate the first design principle of the core being application agnostic. Finally, the above listing uses a simple net present value function \textit{NPV()}, but implemented to accept single values or arrays of values as employed in this case.

The last variable defined in the above listing is cost per tonne of CO\textsubscript{2}e, \textit{cpt}. There is a constant variable defined in a separate module referred to as \textit{Constants} and made available to the module listed above through a Python-specific \textit{import} statement. This constant, CO\textsubscript{2}e\_C is the factor by mass of carbon dioxide to mass of carbon. Since this value is application-specific it, and all other constants are encoded into a code module outside the core application.

This elementary example demonstrates the ease with which data processing can be stated, given a set of generic functions and operator overloading. There is nothing beyond rudimentary mathematics shown here, but it sets the stage for the much more sophisticated operations that must be accommodated to support an abatement market model.
2.3.2 A Sample of Useful Core Application Data Types

A key insight found through developing the reference implementation in parallel with this work is the following judicious collection of data types that comprise a fruitful and powerful way to build a modeling application. This appear to stem largely from the constructed ability to overload operators as well as functions. Operator and function overloading can inspire the creation of a plethora of variable types. The downside to having too many variable types is that they quickly becomes unwieldy, confusing and ultimately counter productive. The problem of which data types to create is comparable to the evolutionary problem of creating the right number of species to populate an ecosystem.

Some types such as \texttt{DOUBLE} and \texttt{STRING} are expected since they represent respectively double precision floating point numbers and strings of characters (words or sequences of words). Interestingly, it is economical to have both an \texttt{ARRAY} type representing an array of objects and the more specific \texttt{INTEGERS} representing an array of integers.

It is programmatically advantageous to have every model related data type share a common type. In the reference implementation this common type is called \texttt{Variant}. The Java programming language has the concept of each \texttt{object} (a variable, a function, a class etc) sharing a common type, unsurprisingly called \texttt{Object}. Linguistically one may say any variable, function, class, etc. in the Java programming language is an \texttt{Object}. In this sense it is useful to say each model object in the core application is a \texttt{Variant}. This is true logically and programmatically.

Each model object type is of type \texttt{Variant} in general and of some other type like \texttt{DOUBLE} specifically. In this manner the definition of the \texttt{Variant} of specific type \texttt{ARRAY} is an array of objects of general type \texttt{Variant}, but individually of unspecified specific type.

From a mathematical point of view it’s interesting to note that it is useful to have a \texttt{set} data type, but not allow it to contain generic \texttt{Variants} as the \texttt{ARRAY} type does, but
to create two separate data types `STRING_SET` and `INT_SET`. The former represents a set of Variants of sub-type `STRING` and the latter representing a set of Variants of sub-type `INT` representing integers. The arithmetic operations of addition and subtraction between operands each being either `STRING_SET` or `INT_SET` will have their usual set theoretic meaning.

As shown in the first code example there was a variable named `TABLE` referred to as a dictionary. Specifically, it is a Variant of sub type `STRING_MAP`.

A number of programming languages including Java have an efficient built-in way to take a string and return as associated object of any type. The interesting feature to note here is that a certain amount of structure is prudent. In this case `STRING_MAP` accepts an object convertible to a string and returns the associated (generic) Variant and not the language-wide generic `Object`. This first code example shows the elementary usage of a `STRING_MAP`, that is, its use to look up an object associated with a string as in,

```
fin_year = TABLE["FinancialYear"]
```

Consider the following usage of the `STRING_MAP` shown below,

```
periods = TABLE["Period1", "Period2", "Period3"]
```

In this case an array of three strings is presented to the `STRING_MAP` named `TABLE`. The behavior embodied here is abstracted to the twin concepts of structural versus target access. In this case, ("Period1", "Period2", "Period3") is automatically converted through the embedding process from a Python `List` to a `Variant` of sub type `ARRAY` of `STRING Variants`. The structural access is as follows, an `ARRAY` is access presented so an `ARRAY` result is expected, that is, the structure of the result is described. The target access is the fact that each element of the presented `ARRAY` is of type `STRING` and is expected to perform the elementary lookup operation within the `STRING_MAP`. The result in this case with then be an array of columns representing the three periods in the given data `TABLE`. 

38
Given the *structural* versus *target* dichotomy it is easy to understand what will result in the following line,

\[
\text{periods} = \text{TABLE}\{"\text{Period1"}, "\text{Period2"}, "\text{Period3"}\}
\]

There is the subtle difference where the parentheses in,

\[
("\text{Period1"},"\text{Period2"},"\text{Period3"})
\]

are replaced by braces in,

\[
\{"\text{Period1"},"\text{Period2"},"\text{Period3"}\}
\]

It is understood that an array of strings surrounded by braces represents a *STRING_SET*. With respect to lookup within a *STRING_MAP* such as \( \text{TABLE} \) above the expected result is then structurally dictated to be another *STRING_MAP*. In this case *periods* will be a *STRING_MAP* that is a subset of the *STRING_MAP* \( \text{TABLE} \).

Many *Variant* sub types, such as *STRING_MAP* are designed to contain other *Variants*, there is no logical limit to their recursion. This makes it possible to build configuration modules containing a single top-level *Variant* representing all external information relevant to that module. For example, this allows a detailed description of fuel prices and fuel price projections across application-specific regions. Note that the existence of such large scale structures necessitates the existence of convenient access idioms.
2.3.3 Two Ways to Access Variants

The existence of Variant sub types allows for a rich set of data access methods. An interesting idea that came from attempting to implement the reference application is the necessary existence of two closely related but distinct ways to access data from within a hierarchical Variant construction. In order to illustrate this idea properly, it is necessary to give a toy example of such a hierarchical structure and discuss some variants of the usual method of accessing sub structure data.

Consider,

\[
m = \text{Variant} (\{ 'a': \{ 'A': 1 , 'B': 2 \}, 'b': \{ 'A': 3 , 'B': 4 \} \})
\]

Usually within the application modules there’s no need to explicitly call Variant as a function as this call is implicit from context. In this case there is no other context and the new variable \( m \) is a hierarchical structure as follows. We see at the top level that \( m \) is a STRING_MAP where the string/character \( a \) is mapped to the STRING_MAP representing a map from string \( A \) to number 1 and from string \( B \) to number 2. Similarly the top level STRING_MAP maps \( b \) to the map of \( A \) to 3 and \( B \) to 4. Internally, each of the structures are one of STRING_MAP, INT or STRING, all sub types of Variant. Note that the quotation marks surrounding the characters distinguished them as strings and not names of variables.

The usual way to access this toy structure is through a get() function symbolized with square brackets. The first line is the command issued and the second line is the result obtained,

\[
m[ 'a' ]
\]

\[
\{ A: 1 , B: 2 \}
\]

Note that quotation marks need not surround strings in the response line. The response in this case is the Variant associated with the key string \( a \) which happens to be itself a STRING_MAP.
Using the idiom of *structural access* we may pass in to the `STRING_MAP m` an `ARRAY` and expect an `ARRAY` response as in,

\[
m[( 'a' , 'a' )] \to \{A: 1, B: 2\}, \{A: 1, B: 2\}
\]

In this case the same key was issued twice. In practice any array with key strings compatible with the `STRING_MAP m` will work thus creating an array translator.

Since `m` is a hierarchical structure of more than one level it is convenient to develop a second type of access methodology. Rather than the familiar `get()` or simply `[]` there is the function `lookup()`. The `lookup()` function is designed as a *deep* or *sequential* `get()`. An elementary example distinguished it superficially from the `get()` function,

\[
m. lookup ([ 'a' , 'A' ])
\]

1

This example is equivalent to the `get()` version as in,

\[
m[ 'a' ][ 'A' ]
\]

1

appearing as a sequence of `get()`s into a hierarchical structure. If that were solely the capability of the `lookup()` function it would be a mere convenience. The more practical use of this function is seen in the following example,

\[
m. lookup ([( 'a' , 'b' , 'a' ) , ( 'A' , 'A' , 'B' )])
\]

(1, 3, 2)

This last example is nearly equivalent to,

\[
(m[ 'a' ][ 'A' ] , m[ 'b' ][ 'A' ] , m[ 'a' ][ 'B' ]) \\
(1, 3, 2)
\]
a minor difference being that the result in the first example is a Variant and in the second it’s a Python tuple which may require an explicit conversion to type Variant. The major difference is that arrays of keys such as \((a', b', a')\) and \((A', A', B')\) arise naturally while processing arrays of information.

It is convenient at this point to introduce another idiom; when an array is required and a non-array is presented, an array of the appropriate length is implied. This allows the following access statement,

\[
m.\text{lookup}((\text{'a'}, \text{'b'}, \text{'a'}), \text{'A'})
\]

\[(1, 3, 1)\]

which is equivalent to,

\[
(m[\text{'a'}][\text{'A'}], m[\text{'b'}][\text{'A'}], m[\text{'a'}][\text{'A'}])
\]

\[(1, 3, 1)\]

The singleton to array idiom is useful in many areas. As a result it is possible to state a configuration value as an array or a single value, and apply functions using this configuration value safely without knowing its type.

There are a number of other ways to access Variants depending on their type including indexing, masking and slicing all obeying the structural/target idiom.

2.3.4 Example for Running an LP Engine

The numerical market example given earlier was run with the current reference implementation. Using the core application and its cerebral embedded Python interpreter the following lines of code are all that is needed to run the full example in the reference implementation (including line numbers for reference),

\[
(\text{line})
\]

\[(1)\]

\[
P = \text{Variant}(((10, 11, 12), (7, 8, 9)))
\]
\[
\begin{align*}
Q &= \text{Variant}(((5, 6, 7), (2, 3, 4))) \\
d &= \text{createVector}((3, 0, 22)) \\
N &= \text{P.size}() \\
T &= \text{P[0].size}() \\
A1 &= -\text{createSparseMatrix}(N \times T, N \times T, 0) \\
b1 &= \text{createVector}(N \times T) \\
LT &= \text{createSparseMatrix}(T, T, T) \\
A2 &= \text{createSparseMatrixDiagonal}(LT \ast \text{ones}(N)) \\
b2 &= Q\text{.toVector}() \\
A3 &= -\text{createSparseMatrixHorizontal}(
\text{LT}^{\ast\ast}2 \ast \text{ones}(N)) \\
b3 &= -d\text{.cumsum}() \\
C &= \text{createSparseMatrixDiagonal}(
\text{createSparseDiagonal}(
\text{arange}(T)[:: -1] + 1) \ast \text{ones}(N)) \\
cost &= C \ast \text{P.toVector}() \\
A &= \text{createSparseMatrixVertical}([A1, A2, A3]) \\
b &= \text{createVector}([b1, b2, b3]) \\
LP &= \text{minimizeLP}(\text{cost, A, b}) \\
x &= \text{createVector}(LP\text{.getX}()) \\
X &= x\text{.reshape}((N, T)) \\
\end{align*}
\]

The result is the portfolio matrix \(X\) corresponding to the \(N = 2\) abatement items over the \(T = 3\) model years,

\[
X = ((5, 0.5, 0), (2, 1, 1))
\]
Creating a user guide for the reference implementation is beyond the scope of this proposal and thesis, but there are some features in the above listing to note.

The first three lines create the example data for the price and quantity matrices $P$ and $Q$ and the shortfall vector $d$. Lines 6-12 correspond to previous definitions for the components of the linear programming constraint matrix $A$ and vector $b$ which are assembled in lines 15 and 16. The cost vector $C$ is constructed to fit the statement (in the actual code module) on a single line. The statement $[:: -1]$ is an example of a slice mentioned above simply reverses the order of an array and the `arange()` function accepts an integer $n$ and returns the array $(0, 1, 2, ..., n - 1)$. The critical line in this listing is 17. In line 17 we call a core application function that provides access to a linear programming solver available from a third party. In the reference implementation the LP engine is provided by the Gnu Linear Programming Kit GLPK, available as an open-source application. The remaining two lines of code pull the result portfolio from the LP engine first as a vector, then as an array of the appropriate shape.

In summary, note that the material in this section demonstrates that the numerical modeling and simulation can be accomplished with readily understandable code without sacrificing overall application performance or restriction to a deployment restrictive commercial solution.

2.4 Insights from the Elementary Model

Our initial intuition regarding the introduction of mandatory requirements was to carefully trace all the decision points and report any that had changed. The effort involved to trace each point illuminated the fact that there is no limit to the scope of changes to a policy result after mandatory requirements are introduced. The solution of running the model once for each case and comparing the results is a robust direct approach.
There is a software implementation design challenge posed by requirement that two dissimilar sets of policy results be comparable. A regulatory policies with and without mandatory requirements are two distinct policies and the PHoX modeling system reference implementation of the AB32 model must stand ready to compare any two pieces of data and configuration of results. We liken this design requirement to the foundation footprint of a building.

The insight gained by augmenting the AB32 reference implementation to allow comparison between regulatory policies with and the otherwise same policy without a set of mandatory requirements is the identification of implicit regulation. Requiring a minimal investment schedule in photovoltaic solar capacity may result in a reduction (or increase) in cement output, for example. This cause-and-effect connection is identifiable within the PHoX modeling system.

A further insight, and perhaps future direction, is that the policy feature we call price triggers defined above do not seem to be accommodated within a linear programming optimization structure. The fundamental issue seems to be that an otherwise static policy is altered as the result of prices escaping stated price limits. The fact that prices are not observable within the AB32 reference model, as implemented, means that the model must be re-run to accommodate the altered state indicated by the triggered price threshold. However, in this new state the observed price may not escape the price limits.

If, for example, given a certain policy (policy #1) the AB32 model finds that the price for a carbon abatement credit is $12 and there is a price trigger set at $10 we must re-run the model. The consequence of triggering a price trigger is spelled out by the trigger requirements and augments the policy accordingly resulting in policy #2. If we run the AB32 model using policy #2 we may find that the credit price is $9 and no longer justifies the use of policy #2 over the preferred policy #1. Policy triggers, as they appear in the AB32 reference model, react to the credit price in one year by requiring policy changes.
in following years. As we learned from our experience with mandatory requirements, the model reacts to policy across time just a human game players anticipate future situations.

The model implementation challenge posed by the inclusion of policy altering price triggers is not resolved in the AB32 reference implementation. A future direction is indicated. In our inquiries into this matter we find that the price trigger problem has the flavor of a dynamic programming situation as described by Bellman [3]. Bellman describes a hallmark dynamic programming problem through the example of two gold mines sequentially mined by a single piece of equipment. The gold mining activity terminates when the mining machine ceases to function or a predetermined wealth in gold is achieved. According to Bellman, the great insight of dynamic programming is to solve the problem backwards.

The manner in which we believe the problem of supporting price triggers within the AB32 reference implementation is through the dynamic programming activity of supposing a final state for the model and re-running the model with different price trigger policies starting with the last year of the model as recommended in Bellman [3].
3.1 Introduction

According to Saltelli (Saltelli 2004[19]) an important phase in model analysis is sensitivity and uncertainty analysis. We may add risk analysis to that list. Our focus model is the carbon abatement market which impacts business finances and uncertainty implies risk of loss of business profits.

The standard approach, according to Saltelli ([19]), is to use Monte Carlo analysis. A description of a family of Monte Carlo methods of analysis may be found in Tanner (Tanner 1996 [51]). As may be found in Saltelli or Tanner, the basic idea behind the Monte Carlo method is to assign random variables to model inputs to represent their uncertainty. A value from each input is chosen and the model is run. The result of the model run is stored and the process is repeated.

The techniques involved with model uncertainty and sensitivity analysis using Monte Carlo include, according to Tanner ([51]) and more recently Weare (Weare 2007 [54]), incorporation of Markov Chains. It may be that the technique we develop in this work for analyzing sensitivity, uncertainty and risk provides no computational advantage over modern Monte Carlo methods. It is the opinion of the author of this work that by taking a glass-box approach where we take the model implementation into account as opposed
to the *black-box* approach where only model outputs are captured, contrasting results of model analysis will be obtained.

We begin our uncertainty, sensitivity and risk analysis in the standard, according to Saltelli ([19]), approach of assigning a probability distribution to each model input to represent its uncertainty. We then regard model inputs as random variables and notice that within our reference model implementation that these inputs are combined arithmetically. We understand from elementary texts such as Bickel [4] that functions of random variables, including their arithmetic combinations, result in still more random variables. We then ask ourselves what if we re-constructed our reference model to allow for random variables in place of numerical values and ran the entire model with this substitution. We wonder, would we then have undertaken an uncertainty, sensitivity and risk analysis by other means? This is the central question for the remainder of this work.

In the opinion of the author, an attractive feature that Monte Carlo methods seem to provide is that the model under analysis need not be re-implemented. We wish to preserve this feature of non-re-implementation in our work. As we have seen previously, programmatic objects in the PHoX modeling system are all derived from a common object called Variant. Our goal will be to describe an upgrade of Variant that includes random variables as a possible representation along side real values, lists, maps, matrices, etc. that we have already identified.

To accomplish our goal of substituting random variables for real-valued model inputs we find it necessary to extend the established theory of functions of random variables. Our theory extension will go far enough to include all the functions we find not only in the reference model, but in general *business class* models as far as we can determine. A business class model, as defined herein, uses functions and concepts available in standard business textbooks such as Brealey [6].

In following sections we describe an implementation that numerically computes all the
functions of random variables we require. We will find that merely being able to program-
matically represent objects such as $f(A)$, $A + B$, $A \div B$, etc. for independent random
variables $A$ and $B$ is not sufficient for our purposes. We return to Saltelli ([19]) to find we
must extend the theory of random variables and our implementation thereof to account for
unavoidable correlations between random variables.

In keeping with our glass box approach we notice that it is no longer feasible to pre-
pare our reference model for optimization and then pass this programmatic preparation to
an external optimization engine expecting results. We must implement the optimization
step within PHoX so that it will allow random variable inputs. The reference model im-
plemented within PHoX uses a linear programming engine which we will implement using
the Simplex Method as described in Gass (Gass 1975 [24]). We will describe the Simplex
Method in detail to determine if any techniques for handling random variables are needed
beyond those already described.

A final chapter is devoted to special considerations for random variable processing par-
ticular to the Simplex Method. The structure of the model results following the Simplex
Method will be presented.

Our literature search suggests that the approach described herein is novel.

In a monograph by Drew et al. [18] a Maple module called A Probability Programming
Language (APPL) is presented. The goal of Drew et al. is to develop data structures and
algorithms with which to derive existing and new results in probability and statistics.

The data structures in APPL allow the representation of continuous random variables
through piecewise symbolic functions. Using the hosting Maple’s computer algebra system
algorithms, new random variables are created through high-level statements. Citing an
example from the Drew text [18], let
Figure 3.1: Probability Distribution of Random Variables $X = \text{Tri}(1,2,4)$ and $Y = \text{Tri}(1,2,3)$

For convenience the triangular distributions are shown in figure 3.1.

The resulting probability distribution for the example $V = X \times Y$ above is given in the Drew text [18] by the probability density function $f_V(v)$,
\[
f_V(v) = \begin{cases} 
\frac{-4}{3}v + \frac{2}{3} \log v + \frac{2v}{3} \log v + \frac{4}{3} & 1 < v \leq 2 \\
-8 + \frac{14}{3} \log 2 + \frac{7v}{3} \log 2 + \frac{10}{3}v - 4 \log v - \frac{5v}{3} \log v & 2 < v \leq 3 \\
-4 + \frac{14}{3} \log 2 + \frac{7v}{3} \log 2 + 2v & \\
-2 \log v - v \log v - 2 \log 3 - \frac{2v}{3} \log 3 & 3 < v \leq 4 \\
\frac{44}{3} - 14 \log 2 - \frac{7v}{3} \log 2 - \frac{8v}{3} - 2 \log 3 \\
+\frac{22}{3} \log v - \frac{2v}{3} \log 3 + \frac{4v}{3} \log v & 4 < v \leq 6 \\
\frac{8}{3} - 8 \log 2 - \frac{4v}{3} \log 2 - \frac{2}{3}v \\
+\frac{4}{3} \log v + \frac{v}{3} \log v + 4 \log 3 + \frac{v}{3} \log 3 & 6 < v \leq 8 \\
-8 + 8 \log 2 + \frac{2v}{3} \log 2 + \frac{2}{3}v \\
+4 \log 3 - 4 \log v + \frac{v}{3} \log 3 - \frac{v}{3} \log v & 8 < v < 12 
\end{cases}
\]

As we’ll discuss below, the PHoX modeling system is capable of numerically multiplying piecewise uniform representations of random variables. An overlay of the plot of the function \( f_V(v) \) above with the PHoX result of multiplying the triangular \( X \) and \( Y \) is shown in figure 3.2. We note that the two curves are similar enough as to be indistinguishable at the full scale shown.

A software package called RandVar available for the statistics computing environment R that involves computing with random variables is created and maintained by Kohl [31]. The package RandVar is a modest effort to allow functions and arithmetic combinations of random variables of various types. The 24 page manual and 10 page user guide suggest that correlated random variables are beyond the scope of the RandVar work. The focus of the RandVar work, judging by the test results available on the web, are on finding integer powers, encoding random variables into matrices and allowing for their multiplication.
3.2 Introduction to Random Variable Arithmetic

A concise explanation of how to combine random variables is provided in a paper by Gray [25]. In this section we review the salient features of that paper in order to provide a theoretical underpinnings to the numerical methods we develop later in this work.

3.2.1 Functions of a Random Variable

The familiar change-of-variable formula for finding the probability distribution of a random variable $Y$ expressible as the function of a known random variable $X$ as in $Y = f(X)$ as may be found in Bickel [4] requires special consideration when $f$ is not a bijection. The Gray [25] paper describes a technique to build the change-of-variable formula using the Fourier transform (FT) of the probability distribution function [PDF] of $X$. We use this (FT) technique to extend the examples in the Gray [25] paper to include exponentiation
of random variables. We do this because the class of models we have chosen to support includes this operation.

We begin the development of the arithmetic of random variables by recalling the following formulas as found in Gray [25],

\[ M_Y(\theta) = \int e^{i\theta f(x)} P_X(x) dx \]  
(3.1)

\[ P_Y(y) = \frac{1}{2\pi} \int e^{-i\theta y} M_Y(\theta) d\theta \]  
(3.2)

\[ = \frac{1}{2\pi} \int \int e^{i\theta(f(x)-y)} d\theta P_X(x) dx \]  
(3.3)

\[ = \int \delta(f(x) - y) P_X(x) dx \]  
(3.4)

since the Fourier transform of 1 is \(2\pi \delta(\cdot)\). Notice by change-of-variables that,

\[ \delta(g(x)) = \sum_{x_i|g(x_i)=0} \frac{\delta(x - x_i)}{|g'(x_i)|} \]  
(3.5)

then the PDF of \(Y = f(X)\) may be expressed in terms of \(P_X(x)\) as,

\[ P_Y(y) = \sum_{x_i|f(x_i)=y} \frac{P_X(x_i)}{|f'(x_i)|} \]  
(3.6)

The following one dimensional transformations will be of particular interest below: \(Y = \ln(X)\) and \(Y = e^X\).
3.2.1.1 Example: \(Y = \ln(X)\)

If \(f(x) = \ln(x)\) then \(f'(x) = 1/x\) and \(f(x) = y \implies x = e^y\) uniquely so that,

\[
P_Y(y) = e^y P_x(e^y) \quad (3.7)
\]

3.2.1.2 Example: \(Y = e^X\)

If \(f(x) = e^x\) then \(f'(x) = e^x\) and \(f(x) = y \implies x = \ln(y)\) uniquely so that,

\[
P_Y(y) = \begin{cases} 
\frac{1}{y} P_x(ln(y)) & \text{if } y > 0 \\
0 & \text{if } y \leq 0
\end{cases} \quad (3.8)
\]

3.2.2 Functions of a Random Vector

A collection of random variables \(X_1, X_2, \ldots, X_n\) may be expressed as a vector \(X = (X_1, X_2, \ldots, X_n)\). If \(f\) is a real-valued function of \(X\) then the PDF of \(Y = f(X)\) may be expressed as

\[
P_Y(y) = \int \delta(f(x) - y) P_X(x) dx \quad (3.9)
\]

3.2.2.1 Example: \(Y = X_1 + X_2\)

Assume random variables \(X_1\) and \(X_2\) are independent. In this case of \(Y = f(X)\) we have \(f(x_1, x_2) = x_1 + x_2\) so that,
\[
P_Y(y) = \int \int \delta(x_1 + x_2 - y)P_X(x_1, x_2)dx_1dx_2 \tag{3.10}
\]

\[
= \int P_X(x_1, y - x_1)dx_1 \tag{3.11}
\]

\[
= \int P_{X_1}(x_1)P_{X_2}(y - x_1)dx_1 \quad \text{if } X_1, X_2 \text{ independent} \tag{3.12}
\]

where the last expression is the familiar convolution expression for the sum of two independent random variables.

3.2.2.2 Example: \(Y = X_1X_2\)

In this case of \(Y = f(X)\) we have \(f(x_1, x_2) = x_1x_2\) so that,

\[
P_Y(y) = \int \int \delta(x_1x_2 - y)P_X(x_1, x_2)dx_1dx_2 \tag{3.13}
\]

Choosing \(x_2\) to be the focus variable let,

\[
g(x_2) = x_1x_2 - y \tag{3.14}
\]

\[
g'(x_2) = x_1 \tag{3.15}
\]

then solving for \(g(x_2) = 0\) and finding \(x_2 = y/x_1\) we have the identity,

\[
\delta(x_1x_2 - y) = \frac{\delta(x_2 - y/x_1)}{|x_1|} \tag{3.16}
\]
so the PDF of $Y$ may be expressed as,

\[
P_Y(y) = \int \int \frac{\delta(x_2 - y/x_1)}{|x_1|} P_X(x_1, x_2) dx_2 dx_1 \\
= \int \frac{P_X(x_1, y/x_1)}{|x_1|} dx_1
\]

(3.17)  

(3.18)

where we have assumed that the integrals are square integrable and the order of integration may be changed without consequence. The result is a well-known expression for the PDF of a product of two random variables that are not necessarily independent.

3.2.3 Extending the Theory

An extensive search of the literature has yet to reveal consideration of the particular function of two random variables. We then establish the following theorems,

**Theorem 3.1.** Given two independent random variables $X_1$ and $X_2$, the probability density function of $Y = X_1^{X_2}$

is given by,

\[
P_Y(y) = \begin{cases} 
\frac{1}{y} \int P_X(x_1, \ln(y)/\ln(x_1)) dx_1 & \text{if } y > 0, \\
0 & \text{if } y \leq 0
\end{cases}
\]

(3.20)

**Proof.** Given two independent random variables $X_1$ and $X_2$ let,
By equation 3.4, the probability density function of $Y$ is,

$$ P_Y(y) = \int \int \delta(x_1^{x_2} - y) P_x(x_1, x_2) dx_1 dx_2 $$  \hspace{1cm} (3.22)

Choosing $x_2$ as the focus variable define,

$$ g(x_2) = x_1^{x_2} - y $$  \hspace{1cm} (3.23)
$$ g'(x_2) = \ln(x_1)x_1^{x_2} $$  \hspace{1cm} (3.24)

Assume $y > 0$. Solving $g(x_2) = 0$ and finding $x_2 = \ln(y)/\ln(x_1)$ uniquely we have,

$$ P_Y(y) = \int \int \delta(x_2 - \ln(y)/\ln(x_1)) |\ln(x_1)x_1^{x_2}| P_X(x_1, x_2) dx_1 dx_2 $$  \hspace{1cm} (3.25)

$$ = \int P_X(x_1, \ln(y)/\ln(x_1)) \left| \ln(x_1)x_1^{\ln(y)/\ln(x_1)} \right| dx_1 $$  \hspace{1cm} (3.26)
$$ = \frac{1}{|y|} \int \frac{P_X(x_1, \ln(y)/\ln(x_1))}{|\ln(x_1)|} dx_1 $$  \hspace{1cm} (3.27)

By similar calculation, if we chose $x_1$ as the focus variable in the above derivation instead of $x_2$ the PDF of $Y$ is the same up to change-of-variable.

Looking ahead to the issue of software implementation of numerical methods to com-
pute random variables we note the following identities for real values \( a \) and \( b \),

\[
ab = \exp(\ln(a) + \ln(b)) \quad \text{if } a, b > 0 \tag{3.28}
\]

\[
ab^b = \exp(\exp(\ln(a) + \ln(\ln(b)))) \quad \text{if } a > 0 \text{ and } b > 1 \tag{3.29}
\]

We will next show that identity 3.28 and similarly that identity 3.29 hold when real values \( a \) and \( b \) are replaced by independent random variables \( X_1 \) and \( X_2 \), respectively.

Identity 3.28 offers the an implementation choice, one of which may be programmatical-ly advantageous over the other. Rather than numerically computing the product of two random variables directly we may transform each by application of the logarithm function and compute their sum and transform back by the exponential function. Following the form of identity 3.28 we find \( Y = X_1 X_2 \) as,

\[
W_1 = \ln(X_1) \tag{3.30}
\]

\[
W_2 = \ln(X_2) \tag{3.31}
\]

\[
Z = W_1 + W_2 \tag{3.32}
\]

\[
Y = e^Z \tag{3.33}
\]

We show that identity 3.28 holds for independent random variables formally using the Fourier Transform method developed above in the following theorem.

**Theorem 3.2.** Given independent random variables \( X_1 \) and \( X_2 \) such that \( P(X_1 < 0) = 0 \) and \( P(X_2 < 0) = 0 \) the following identity holds,
\[ X_1 X_2 = e^{\ln(X_1) + \ln(X_2)} \]  

(3.34)

**Proof.** Given two non-negative independent random variables \( X_1 \) and \( X_2 \) let,

\[ A = \ln(X_1) \]  

(3.35)

\[ B = \ln(X_2) \]  

(3.36)

\[ C = A + B \]  

(3.37)

\[ Y = e^C \]  

(3.38)

then we have,

\[ P_A(a) = e^a P_{X_1}(e^a) \]  

by eq. 3.7  

(3.39)

\[ P_B(b) = e^b P_{X_2}(e^b) \]  

by eq. 3.7  

(3.40)

\[ P_C(c) = \int P_A(a) P_B(c-a) da \]  

by eq. 3.12  

(3.41)

\[ = \int e^a P_{X_1}(e^a) e^{c-a} P_{X_2}(e^{c-a}) da \]  

(3.42)

\[ = e^c \int P_{X_1}(e^a) P_{X_2}(e^c/e^a) da \]  

(3.43)

Let \( x_1 = e^a \implies \frac{1}{|x_1|} dx_1 = da \)  

(3.44)

\[ = e^c \int P_{X_1}(x_1) P_{X_2}(e^c/x_1) \frac{1}{|x_1|} dx_1 \]  

(3.45)

\[ P_Y(y) = \frac{1}{y} P_C(ln(y)) \]  

by eq. 3.8  

(3.46)

\[ = \int P_{X_1}(x_1) P_{X_2}(y/x_1) \frac{1}{|x_1|} dx_1 \]  

(3.47)
To extend identity 3.34 to any pair of independent random variables \(X_1, X_2\) we must break the problem into four cases. For any random variable \(X\) let,

\[
X^+ = \max(X, 0) \tag{3.48}
\]

\[
X^- = \max(-X, 0) \tag{3.49}
\]

Notice that \(P(X^+ < 0) = 0 = P(X^- < 0)\) and \(X = X^+ - X^-\). Then we may write,

\[
Y = X_1 X_2 \tag{3.50}
\]

\[
= (X_1^+ - X_1^-)(X_2^+ - X_2^-) \tag{3.51}
\]

\[
= X_1^+ X_2^+ + (-X_1^+ X_2^-) + (-X_2^+ X_1^-) + X_1^- X_2^- \tag{3.52}
\]

where each product involves only pairs of non-negative random variables.

Moving on to the exponentiation of random variables we have the following theorem,

**Theorem 3.3.** Given independent random variables \(X_1\) and \(X_2\) such that \(P(X_1 < 0) = 0\) the following identity holds,

\[
X_1^{X_2} = e^{X_2 \ln(X_1)} \tag{3.53}
\]

*Proof.* Given \(X_1\) and \(X_2\) be two independent random variables such that \(P(X_1 < 0) = 0\). We want to show that the probability density of \(X_1^{X_2}\) in equation 3.20 is the same as for \(e^{X_2 \ln(X_1)}\). To this end we let,
\[ A = \ln(X_1) \] (3.54)
\[ B = AX_2 \] (3.55)
\[ Y = e^B \] (3.56)

then we have,

\[ P_A(a) = e^a P_{X_1}(e^a) \]

by eq. 3.7 (3.57)

\[ P_B(b) = \int \frac{P_A(a)P_{X_2}(b/a)}{|a|} da \]

by eq. 3.18 (3.58)

\[ = \int \frac{e^a P_{X_1}(e^a)P_{X_2}(b/a)}{|a|} da \]

(3.59)

\[ P_Y(y) = \frac{1}{y} P_B(ln(c)), \ y > 0 \]

by eq. 3.8 (3.60)

\[ = \frac{1}{y} \int \frac{e^a P_{X_1}(e^a)P_{X_2}(b/a)}{|a|} da \]

(3.61)

\[ = \frac{1}{y} \int \frac{P_{X_1}(x_1)P_{X_2}(ln(y)/ln(x_1))}{|ln(x_1)|} dx_1 \]

(3.62)

We have shown that we may write the exponential expression \( X_1^{X_2} \) as the exponential of the product \( X_2 \ln(X_1) \) which can in turn be written as the exponential of the sum \( \ln(X_2) + \ln(\ln(X_1)) \). This means, for implementation purposes we need only split random variables, such as \( X \), into positive (\( X^+ \)) and negative (\( X^- \)) parts, perform logarithmic and exponential transformations as appropriate and ultimately only have to compute the sum of two independent random variables. That is, we have established that,
\[ X_1^{X_2} = e^{\ln(\ln(X_1)) + \ln(X_2)} \quad \text{if } X_1 > 1, X_2 > 0 \] (3.63)

In order to handle any two independent random variables \(X_1\) and \(X_2\) we need to break into cases. For any random variable \(X\) let,

\[ X^{1+} = \max(X, 1) \] (3.64)
\[ X^{01} = \frac{1}{\min(X^+, 1)} \] (3.65)

Notice that \(X^+ = X^{1+} + 1 / X^{01}\) and that \(X^{1+}, X^{01} \geq 1\).

3.3 Introduction to Numerical Random Variable Arithmetic

The theory for arithmetic combinations of random variables presented and developed in this work thus far assume the operand random variables are independent. A closer examination of our reference model reveals, as we will detail later, that a particular form of random variable correlation is unavoidable. Accordingly, our implementation will be updated and extended to accommodate the correlation found in the model. In this section we describe the software implementation of random variables, how to programmatically compute functions of random variables and arithmetic combinations of independent random variables. In later sections will describe the extensions and updates necessary when we relax the assumption of random variable independence.
3.3.1 Representing a Random Variable Numerically

The language of implementation is the language of software. We begin with a cascade of definitions familiar to software programmers.

**Definition 3.4** (Programmatic Object). A Programmatic Object, or simply *object* is characterized by the ability to refer to it within a software program with a single value called a *handle*. A structured collection of programmatic objects. The handle may refer to a location in computer memory or some abstract address that can be resolved in a systematic manner to a structured collection of objects.

**Definition 3.5** (Node). A Node is a programmatic object that contains other objects. The term *node* is used in conjunction with *lists*.

**Definition 3.6** (List). A List is a sequence of programmatic objects. In the context of a list, programmatic objects are called *nodes*.

**Definition 3.7** (Linked List). A Linked List is a list of nodes wherein each node contains the handle of another node within the linked list. Within the context of a linked list this handle is referred to as a *pointer*. Mathematically, a linked list implements a finite chain. The first node of the linked list is referred to as the *head* and the last is referred to as the *tail*. The pointer of the tail node may refer back to the head forming a programmatic *ring*.

**Definition 3.8** (Doubly-Linked List). A Doubly Linked List is a list in which each member node contains two pointers. The first pointer refers to the next node in the list and the second pointer refers to the previous node. The pointers of the head and tail nodes may refer to their counterpart.

**Definition 3.9** (Discrete Random Variable). A Discrete Random Variable is a doubly-linked list of nodes such that,
1. Each node contains a real-valued position.

2. Each node contains a real-valued probability value.

3. The probability of a node is concentrated at the position of that node.

4. The head node has position value $-\infty$.

5. The tail node has position value $+\infty$.

6. Position values are strictly ascending in the list.

**Definition 3.10** (Continuous Random Variable). A Continuous Random Variable is a doubly-linked list of nodes such that, where each node contains a real-valued position and a real-valued probability quantity between the current node and the next node in the list. There are several requirements,

1. Each node contains a real-valued position.

2. Each node contains a real-valued probability value.

3. The head node contains position value $-\infty$.

4. The tail node contains position value $+\infty$.

5. Position values are strictly ascending in the list.

6. The probability value of nodes with finite position value is assume to be distributed between the current node and the next node.

7. The probability value of the head node represents the probability in the neighborhood of $-\infty$.

8. The probability value of the tail node represents the probability in the neighborhood of $+\infty$. 

64
9. There must be at least one node with finite position value.

10. The probability value of the node previous to the tail node is zero.

**Definition 3.11** (Random Variable). A Random Variable is a two-element set containing a continuous random variable and a discrete random variable.

The figure 3.3 shows a doubly-linked node containing two values, a position value $x$ and a probability value $p$. This node also contains two pointer values, named next and prev (meaning previous) intended to refer to other nodes. If the node is an element of a discrete or continuous random variable then its position value is smaller than the position value of the node referred to by its next pointer and larger than the position value of the node referred to by its prev pointer.

The figures 3.4 and 3.5 depict our numerical representations of discrete and continuous random variables, respectively. The discrete random variable in figure 3.4 depicts a 4 node random variable where the positions of the head and tail nodes are shown as $\pm\infty$ as required. The continuous random variable in figure 3.5 depicts a 5 node random variable
where the head and tail nodes are $\pm \infty$ as required. The probability associated with the node previous to the head node of the continuous random variable is assumed to be zero, that is $p_4 = 0$, so it is not shown. In both figures 3.4 and 3.5 only the next-node pointer is shown and the next-node pointer of the tail node is shown referring to the head node, but this is not required by the definition.

We symbolically represent a numerical discrete random variable $D$ as,

$$D = \{(d_1, ..., d_n), (p_1, ..., p_n)\} \quad (3.66)$$

where,

$$-\infty = d_1 < d_2 < ... < d_n = +\infty \quad (3.67)$$

and,

$$P(D = d_i) = p_i \quad \forall \ i = 1 \ldots n \quad (3.68)$$

We symbolically represent a numerical continuous random variable $C$ as,
\[ C = \{(c_1, ..., c_{n+1}), (q_1, ..., q_n)\} \quad (3.69) \]

where,

\[-\infty = c_1 < c_2 < ... < c_{n+1} = +\infty \quad (3.70)\]

and,

\[ P(c_i < C < c_{i+1}) = q_i \quad \forall \ i = 1 \ldots n \quad (3.71) \]
\[ P(C = c_i) = 0 \quad (3.72) \]

We refer to the list of position values of a numerical continuous random variable as its \textit{partition} because it partitions the presumed domain of the Reals.

When we discuss an implementation of the PHoX modeling system that allows numerical random variables we will drop the term \textit{numerical} for simplicity.

Note that there are functions and arithmetic combinations of continuous random variables that result in a sum of discrete and continuous random variables. As an example suppose \( X \) is a continuous random variable such that \( P(X \leq 0) = p \) and \( Y = \Re(\sqrt{X}) \) then \( Y \) is both discrete since \( P(Y = 0) = p \) and continuous since \( Y^+ = \sqrt{X^+} \) and the square root function is a bijection over the non-negative Reals.
3.3.2 Functions of One Random Variable

Given a random variable $A$ such that,

$$A = \{(a_1, \ldots, a_{n+1}), (p_1, \ldots, p_n)\}$$

We assume that $A$ is in proper form if,

$$a_1 < a_2 < \ldots < a_{n+1}$$

Suppose that $A_1 = f(A)$, that is, that random variable $A_1$ is a function of our original random variable $A$. Note that, by convention used throughout this work, a subscript indicates that a random variable is a function of another random variable. To accommodate functions of random variables numerically we distinguish the proper form of a random variable from synchronous form for derived random variables. We may then express $A_1$ as,

$$A_1 = \{(f(a_1), \ldots, f(a_{n+1})), (p_1, \ldots, p_n)\}$$

and note that $A_1$ is in synchronous form with respect to $A$ since each partition endpoint is related by $f$ to its counterpart in $A$ and the probability of each partition interval is the same as $A$. Random Variables that are not synchronous with any other random variable are referred to as first class random variables. In practice we don’t replicate the list of probability values from $A$ to derived random variables such as $A_1$ since they are understood.
The use of the proper form of a random variable are used in at least two instances. One instance is when we wish to present the random variable to a user in some form such as a graph. Another instance is when the PHoX modeling system chooses to treat \( A_1 \) as a first class random variable. This latter case will be explained in a later section.

For example, suppose \( A_1 = A^2 \) and,

\[
A = \{(-2, -1, 0, 1, 2, 3), (p_1, p_2, p_3, p_4, p_5)\}
\]
\[
A_1 = \{(4, 1, 0, 1, 4, 9), (p_1, p_2, p_3, p_4, p_5)\}
\]

To find the proper form of \( A_1 \) we need to sort the partition points and ensure that the probabilities of each partition interval are consistent. The result in this case is found by inspection to be,

\[
A_1 = \{(0, 1, 4, 9), (p_2 + p_3, p_1 + p_4, p_5)\}
\]

We graph the example in figure 3.6. The shaded region emphasizes how the partition interval \((2, 3)\) of \( A \) maps the probability \( p_5 \) onto the partition interval \((4, 9)\) of \( A_1 \).

Notice that since we assume the each partition interval has uniformly distributed probability and that the resulting partition intervals of \( A_1 \) are also uniformly distributed then we are implicitly assuming a piecewise linear approximation of the function \( f \) that create \( A_1 \) from \( A \). Thus the functional relationship between \( A \) and \( A_1 \) in our example is more properly represented in figure 3.7.
Figure 3.6: Function of a Random Variable

Figure 3.7: Piecewise Linear Function of a Random Variable
In the case of the above example, we have chosen a particularly favorable partition for $A$. In particular the interval includes the origin which is an extreme point of the graph of $f$ and the mapped partition intervals such as $(-2, -1)$ and $(1, 2)$ of $A$ perfectly overlap to partition interval $(1, 4)$ of $A_1$.

If we alter our example to not include the origin, but keeping all else the same, we have,

$$A = \{(-2, -1, 1, 2, 3), (p_1, p_2, p_3, p_4)\}$$

$$A_1 = \{(4, 1, 1, 4, 9), (p_1, p_2, p_3, p_4)\}$$

In figure 3.8 we see the shaded region mapping the partition intervals $(-2, -1)$ and $(1, 2)$ of $A$ to the partition interval $(1, 4)$ of $A_1$ so that $P(1 < A_1 < 4) = p_1 + p_3$. A notable concern is that the partition interval $(-1, 1)$ of $A$ maps the probability value $p_2$ to the discrete value 1 of $A_1$, that is $P(A_1 = 1) = p_2$. This discrete probability is an artifact of the two endpoints of the partition interval $(-1, 1)$ of $A$ being mapped to the same value, 1, of $A_1$.

If we offset one endpoint of the partition interval $(-1, 1)$ of $A$ by some small value, $\epsilon$ so that the interval becomes $(-1, 1 + \epsilon)$ then the corresponding partition interval of $A_1$ becomes $(1, (1 + \epsilon)^2)$. This interval is assigned the the probability $p_2$, but though the probability is assumed uniform and therefore continuous, the probability density is arbitrarily large. We refer to this as the clipping problem.

Since we assume that once a partition for a random variable is chosen it does not change as long as it is referenced by other random variables there seems no clear remedy to the clipping problem. We address the clipping problem in two stages. The first stage is detection and endpoint flagging where we attempt to determine if a partition interval exhibits the
clipping problem and the second stage is *endpoint inclusion*.

To detect the clipping problem we perform a *midpoint test* of each partition interval of $A_1 = f(A)$ as follows,

$$x_i = f\left( \frac{a_i + a_{i+1}}{2} \right) \quad i \in 1 \ldots n$$

If $x_i > \max(f(a_i), f(a_{i+1}))$ or $x < \min(f(a_i), f(a_{i+1}))$ then we have a clipping problem. Notice that the midpoint test may fail to detect the midpoint problem, but as partition interval size is reduced the midpoint test accuracy will increase. If the midpoint problem is detected then we may use an extreme point detection algorithm such as a binary search as found in Burden [9] to find an intermediate point $y_i$ in the partition interval $(a_i, a_{i+1})$ exhibiting the clipping problem so that,

$$f(y_i) \geq f(x) \text{ or } f(y_i) \leq f(x) \text{ for all } x \in (a_i, a_{i+1})$$

The value $y_i \in (a_i, a_{i+1})$ is then flagged as a desirable endpoint for $A$ in any future
iterations of the model. That is, if the model is re-run we expect to see $y_i$ as a partition endpoint. In our example for $A_1 = f(A)$ we expect to find $y_2 = 0$ so that,

$$A = \{(-2, -1, y_2 = 0, 1, 2, 3), (p_1, p_2^a, p_2^b, p_3, p_4)\}$$

where $p_2^a + p_2^b = p_2$, ceteris paribus.

In the PHoX modeling system we implement a number of functions directly, such as $f(A) = A^2$, in which as the fact that zero is a desirable partition endpoint is flagged without necessitating the use of a detection algorithm.

To address the issue of the that mapped partition intervals from $A$ may not overlap ideally we must devise an algorithm that can re-assign probability values. We refer to this algorithm as the *Sunrise* algorithm.

### 3.3.3 The Sunrise Algorithm

The *Sunrise* algorithm (so called because of the location where the idea of its development occurred) accepts an arbitrary finite list of values forming the endpoints of connected intervals each identified with a probability value. Let,

$$A_1 = \{(19, 3, 1, 9, 33), (p_1, p_2, p_3, p_4)\}$$

We recognize $A_1$ as a discrete function so that $A_1[1] = 19, \ldots A_1[5] = 33$. We introduce the idiom $A_1[(1, 2, 3, 4, 5)] = (19, 3, 1, 9, 33)$ then we have,
Figure 3.9: Sunrise Algorithm Part I

\[ I = (3, 2, 4, 1, 5) \]
\[ A_1[I] = (1, 3, 9, 19, 33) \]
\[ J = (4, 2, 1, 3, 5) \]
\[ A_1[I][J] = (1, 3, 9, 19, 33)[J] = (19, 3, 1, 9, 33) = \text{x-list of } A_1 \]

Where \( I \) is the index sort result from sorting the x-list of \( A_1 \) and \( J \) is the index sort result from sorting \( I \). As shown above \( J \) may be said to be the inverse sort of \( I \). The figure 3.9 shows each \( A_1 \) x-value with its associated \( I \)-index denoted \( i \) and associated \( J \)-index denoted \( j \).

The purpose of \( J \) is to relate probability values to their new intervals,

\[ J = (3, 1, 0, 2, 4) \sim (p_1, p_2, p_3, p_4) \]
that is, \( p_1 \) is the amount of probability uniformly distributed between the 3\(^{rd} \) and 1\(^{st} \) indices into the sorted x-values of \( A_1 \). The Sunrise algorithm may be summed up except of a special case explained in figure 3.10.

Let \( A_1^I \) be the proper form of \( A_1 \) so that,

\[
A_1^I = \{(1, 3, 9, 19, 33), (q_1, q_2, q_3, q_4)\}
\]

where,

\[
q_1 = q_2 + \frac{3 - 1}{9 - 1} p_3
\]
\[
q_2 = \frac{9 - 3}{19 - 3} p_1 + \frac{9 - 3}{9 - 1} p_3
\]
\[
q_3 = \frac{19 - 9}{19 - 3} p_1 + \frac{19 - 9}{33 - 9} p_4
\]
\[
q_4 = \frac{33 - 19}{33 - 9} p_4
\]

If a value is repeated, the duplicate may be removed. There are two considerations in this case that the following example will address,
Figure 3.11: Sunrise Algorithm Part III

\[ B = \{(−2, −1, 1, 2), (p_1, p_2, p_3)\} \]
\[ B_1 = B^2 = \{(4, 1, 1, 4), (p_1, p_2, p_3)\} \]
\[ I = (2, 3, 1, 4) \]
\[ J = (3, 1, 2, 4) \]
\[ B_1[I] = (1, 1, 4, 4) \]
\[ B_1[I][J] = (4, 1, 1, 4) \]

where again, \( J \) is a list of indices into the \( B_1[I] \) list of x-values. Figure 3.11 represents the situation.

where the \( j \)-values are the \( B_1^I \) indices and \( J \) is \( j \)-index the sequence of values joined by the original probability values \( p_1, p_2 \) and \( p_3 \). The Sunrise algorithm may be run by inspection to find,

\[ B_1^I = \{(1, 1, 4, 4), (p_2, p_1 + p_3, 0)\} \]
where we see the two cases for repeated values. In the case of the repeated 1 the probability $p_2$ is concentrated at 1 and will be considered discrete and in the case of the repeated 4, there is no probability captured between the two 4’s so the second one may be removed without further notation. Therefore the proper form of $B_1$ is then,

$$B^I_1 = \{(1, 4), (p_1 + p_3)\}, \{[1], [p_2]\}$$

where $\{[1], [p_2]\}$ indicates a list of discrete values with their associated probability. We may now observe that $P(B_1 = 1) = p_2$ and $P(1 < B_1 < 4) = p_1 + p_3$ where the $I$ superscript is dropped.

3.3.4 One Random Variable Exponential Functions

Consider the following two functions of one random variable. The first is $A^d$ and the second $d^A$ where $d \in \mathbb{R}$.

The PHoX modeling system does not represent complex values so raising a random variable to a real exponent requires finding the real part of complex expressions. Recall that,

$$\text{Re}\{(-.95)^{1.2}\} = .95^{1.2}\cos(1.2\pi)$$

The special case exemplified by $A^{1.2}$, for example, and $P(A < 0) \neq 0$ requires splitting $A$ into positive and negative components $A^+$ and $A^-$ respectively. Recall that $A^- > 0$ by definition. Then we re-express $A^{1.2}$ as,
\[ A^{1.2} = (A^+)^{1.2} \oplus (A^-)^{1.2} \cos(1.2\pi) \]

where \( \oplus \) means to sum the probability distributions of the two operands.

The special case exemplified by \((-0.95)^A\) requires special partitioning concerns. Notice this situation is related to a damped (or amped) simply harmonic oscillator. In that case the random variable represents time-uncertainty of position observation. This is deemed by the author to be unlikely to arise in the course of business or public policy modeling. For this reason discussion of the special techniques involved in numerically computing this case are beyond the scope of this work.

3.3.5 Functions of Two Independent Random Variables

We suppose that \( A \) and \( B \) are two independent random variables. We assume that \( A \) and \( B \) are expressed in proper form, that is, a partition is chosen for each \( A \) and \( B \) and there are probability values associated with each partition interval. We make no further assumptions about the range of any probability value, that is, we may have negative probability values or probability values larger than unity and the sum of the probability values for any given random variable implementation need not sum to unity.

For example let,

\[
A = \{(1, 2, 3), (10, 20)\}
\]

\[
B = \{(4, 5, 6), (40, 50)\}
\]

where \( A \) is partitioned into two intervals \((1, 2)\) and \((2, 3)\) and the probability \( P(1 < \)
A < 2) is 10. We can add up the constituent probabilities of $A$ as $(10 + 20 = 30)$ and, if we wish to enforce an overall unit probability, rescale the constituent probabilities by 30. By allowing random variables in the PHoX system to have non-unit total probability we can more easily test system software components and allow random variable to lose probability under certain well-defined conditions.

Independent random variables have a rectangular joint probability distribution that, for piecewise uniform probability distributions, is the cross product of the partition intervals and product of the associated probability values (see Bickel [4]). For our example $A$ and $B$ the joint distribution is shown in figure 3.12 where the probabilities of each interval and joint region are enclosed in parentheses.

Suppose we wish to numerically compute the result of an expressions such as,

$$X = f(A, B)$$

The $X$ in this case is a random variable. To represent $X$ numerically we must assign a partition of the real line with associated probability values. To describe how to find the
partition of $X$ we first suppose one of the partition intervals is $(x_i, x_{i+1})$ for some $x_i < x_{i+1}$.

If the probability value associated with this interval is $p_i$ then we have the relation,

$$P(x_i \leq X < x_{i+1}) = p_i$$

We then notice that for a fixed value of $X$, say $x$, we have the equation,

$$x = f(A, B)$$

which describes a level curve in the joint probability distribution space of $A$ and $B$.

Continuing with our example $A$ and $B$ we suppose some function $f$ and depict the possible level curves for various values of $x$ in figure 3.13.

If we assume,
then we conclude from figure 3.13 that,

\[ x_1 < f(a, b) < x_6 \quad \forall a \in (1, 3), \ b \in (4, 6) \]

and notice that,

\[ P(x_6 < X < x_7) = 0 \]

We realize that we must choose a partition for \( X \) before we calculate the associated probability values. We do not propose a particular partition choice. We recognize a good partition choice as one that treats all joint distribution partition elements equally and results in a computationally manageable number of partition elements.

We refer to a partition choice for \( X \) as \textit{natural} if it consists of the \( x \)-values of the vertices in the joint partition of \( A \) and \( B \). If \( X = A + B \) the natural partition for \( X \) would then be,

\[
X = \{(1 \pm 4, 2 \pm 4, 3 \pm 4, 1 \pm 5, 2 \pm 5, 3 \pm 5, 1 \pm 6, 2 \pm 6, 3 \pm 6), \,(p_1, \ldots, p_n)\}
\]

\[
= \{(5, 6, 7, 6, 7, 8, 7, 8, 9), \,(p_1, \ldots, p_n)\}
\]

\[
= \{(5, 6, 7, 8, 9), \,(p_1, p_2, p_3, p_4)\}
\]
Figure 3.14: Joint Uniform Probability Distribution With Many Level Curves

where the partition \((5, 6, 7, 8, 9)\) is in proper form.

The natural partition for \(X\) is not necessarily best. Suppose that,

\[
C \sim U(c_1, c_2) \\
D \sim U(d_1, d_2)
\]

that is, uniformly distributed. The distribution of \(X = C + D\) is a triangle distribution. In figure 3.14 we depict the \(C + D\) example with a number of level curves for the addition case which we will cover in more detail below. Since the joint rectangle \((c_1, c_2) \times (d_1, d_2)\) is uniformly distributed we can calculate the result of \(X\) for this example by equating the probability between each level curve with the area enclosed.

The result is shown in figure 3.15 where we assume that \(c_1 + d_2 = c_2 + d_1\) for simplicity. Notice that the probability density is not the same as probability. The probability associated with each partition element in figure 3.15 is the height of each step times its width. In this case the partition element widths are all equal so the distinction is not readily apparent.

In the following sections we will detail the computation of addition, subtraction, multi-
Figure 3.15: Probability Distribution of the Sum of Two Independent Random Variables

3.3.5.1 Addition of Two Independent Random Variables

Next we must choose the partition for the result $X = A + B$. We do not assume that $A$ and $B$ have evenly spaced or regular partitioning so there must be some consideration put into the choice for $X$. We notice that the smallest value that $X$ may take on is $1 + 4 = 5$ and the largest is $3 + 6 = 9$, so for simplicity of this example we form a regular partition of the interval $(5, 9)$ using 3 intervals. Thus we already have $X = \{(5, 6, 7, 8, 9), (p_1, p_2, p_3, p_4)\}$ here the $p_i$ are yet to be determined. In practice we use 500 to 1000 intervals to represent a random variable.

We indicate in figure 3.16 the level curves for each partition point 5 through 9.

The amount of probability within each partition interval is the sum of the probability from each uniform cell enclosed within the partition interval. Thus,
Figure 3.16: Addition of Two Independent Random Variables, Partitioned

\[ P(5 < X < 6) = 200 \]
\[ P(6 < X < 7) = 250 + 200 + 400 = 850 \]
\[ P(7 < X < 8) = 250 + 500 + 400 = 1150 \]
\[ P(8 < X < 9) = 500 \]

We can check that probability is preserved; the sum of the joint probabilities is \( 400 + 800 + 500 + 1000 = 2700 \) and the sum of the \( X \) probabilities is \( 200 + 850 + 1150 + 500 = 2700 \). That is \( P(X) = 2700 \) and \( P(A) \times P(B) = (10 + 20) \times (40 + 50) = 2700 \). Had we scaled the \( A \) probabilities by 30 and the \( B \) probabilities by 90 all the total probabilities would be 1 as expected.

The final result of our example is,
There is more than one way to proceed to compute $A - B, A \times B, A \div B$. A direct method is to proceed similarly to the addition case by choosing a partition for the resulting random variable then to describe the level curves representing the endpoints of the partition and to compute the probabilities enclosed within each partition interval, that is, between the level curves. Another way is to use logarithms to convert the multiplication and division cases to the addition and subtraction cases as in,

$$X = A \times B = \exp(\log(A \times B)) = \exp(\log(A) + \log(B))$$

$$Y = A \div B = \exp(\log(A \div B)) = \exp(\log(A) - \log(B))$$

We notice that the real $\log()$ function is not defined for negative values so we must split $A$ into $A^+$ and $A^-$ (where $A^+, A^- \geq 0$) and similarly for $B$. For multiplication we then have,
where “⊕” represents the addition of distribution functions described above. Division is handled similarly. Notice that \( \log(A^-) \) is a monotonically increasing function as is \( \exp() \) over the Reals so the whole computation reduces to four versions of \( A_1 + B_1 \).

The main problem with this method is that it assumes the transformed random variables are still piecewise uniform. This is not true in general, but for a fine partition the error may be made acceptable. An illustrative example is \( C, D \sim U(1, 2) \) and if we choose partition \((1, 2, 4)\) then \( C \ast D = \{(1, 2, 4), (.3862, .61371)\} \) and \( \exp(\log(C) + \log(D)) = \{(1, 2, 4), (.5, .5)\} \). This is because \( \log(C) \) is not uniformly distributed if \( C \) is uniformly distributed.

3.3.5.2 Subtraction of Two Independent Random Variables

Another reason to prefer direct methods is that as we explore correlated cases below we find that the indirect approach does not extend well and that the direct approach will be our workhorse. The direct approach for subtraction is similar to addition. If \( X = A - B \) then the \( B \)-axis is reversed which, graphically, mirrors the level curves about the \( A \)-axis. Figure 3.17 depicts this for our running example.

We had made a similar partition choice for \( X \) as in the addition case so that our example result is,
Figure 3.17: Subtraction of Two Independent Random Variables

\[ X = A - B \]

\[ = \{(1, 2, 3), (10, 20)\} - \{(4, 5, 6), (40, 50)\} \]

\[ = \{(-5, -4, -3, -2, -1), (250, 950, 1100, 400)\} \]

The total probability of \( X \) comes out as expected as \( 250 + 950 + 1100 + 400 = 2700 \).

3.3.5.3 Multiplication of Two Independent Random Variables

To compute \( X = A \times B \) directly we proceed in a similar manner to the addition and subtraction cases. We find an expression for the level curves, decide the partition of \( X \) and compute the interval probabilities. We find the level curve for multiplication by supposing that \( X \) takes on a fixed value, \( x \) and solve for \( B \) in terms of \( A \) which find to be \( B_x(A) = x/A \). Therefore the level curves for multiplication are hyperbolic. For our example \( A \) and \( B \) and a regular partition this is shown in figure 3.18.

The result is computed in the same manner as before by summing probability density times area values within each partition interval. For our example the result is,
Figure 3.18: Product of Two Independent Random Variables
Figure 3.19: Quotient of Two Independent Random Variables

\[ X = A \times B = \{(4, 8, 12, 18), (543.35, 1086.69, 1069.96)\} \]

The total probability of \( X \) is again as expected, \( 543.35 + 1086.69 + 1069.96 = 2700. \)

3.3.5.4 Division of Two Independent Random Variables

Computing the division \( X = A/B \) is once again similar to the previous cases. The only difference is the level curves found by fixing \( X \) to a value \( x \) and solving for \( B_x(A) = A/x \). These “curves” are radial lines through the origin. For our example and a convenient partition choice we refer to figure 3.19.

The result for our example \( X \) is then,
\[ X = A \div B \]
\[ = \{(\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{3}{4}), \]
\[ (500 \times \frac{5}{6} + 400 \times \frac{1}{2}), \]
\[ 500 \times \frac{1}{6} + 400 \times \frac{1}{2} + 1000 \times \frac{3}{4} + 800 \times \frac{1}{4}, \]
\[ 1000 \times \frac{1}{4} + 800 \times \frac{3}{4})\} \]
\[ = \{(\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{3}{4}), (616\frac{2}{3}, 1233\frac{1}{3}, 850)\} \]

Finally, the total probability of \( X \) is \( 616\frac{2}{3} + 1233\frac{1}{3} + 850 = 2700 \) as expected. This will always be the case since we are partitioning the joint probability region, but when dealing with numerical algorithms it’s nice to check for the expected results.

3.3.5.5 Division Is Not Quite Inverse Multiplication

Upon inspection of the addition and subtraction operations we conclude they are mirror images of each other and that the subtraction operation may be replaced by negation followed by addition. That is, \( A - B = A + (-B) \). This is because negation is a rigid transform and in particular preserves probability distribution in the following sense; if \( B \sim U(b_1, b_2) \) then \( -B \sim U(-b_2, -b_1) \).

Notice that if \( B \sim U(b_1, b_2) \) then \( 1/B \) is not uniformly distributed. The probability within the inverted interval \((1/b_2, 1/b_1)\) is shifted to one side by the reciprocal operation. The consequence of this distortion is a discrepancy between the expression \( X = A/B \) and \( Y = A \times (1/B) \).

To demonstrate this distortion we return to our working example,
\[ A = \{(1, 2, 3), (10, 20)\} \]
\[ B = \{(4, 5, 6), (40, 50)\} \]
\[ X = A \div B \]
\[ = \{\left(\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{3}{4}\right), (616 \frac{2}{3}, 1233 \frac{1}{3}, 850)\} \]

If we perform the reciprocal operation of \( B \) to form \( 1/B \) without taking care to alter our partition choice we have,

\[ 1/B = \{\left(\frac{1}{6}, \frac{1}{5}, \frac{1}{4}\right), (50, 40)\} \]

where the implicit assumption is that \( 1/B \sim U(\frac{1}{6}, \frac{1}{3}) + U(\frac{1}{5}, \frac{1}{4}) \). If we form \( Y = A \times (1/B) \) with the usual partition scheme the result is shown in figure 3.20.

and,

\[ Y = A \times (1/B) \]
\[ = \{\left(\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{3}{4}\right), (606.66, 1213.31, 880.03)\} \]

We still have \( P(Y) = 2700 \) as expected, but comparing \( X = A/B \) and \( Y = A \times (1/B) \) we find a modest discrepancy.

The consequence for what follows of retaining division as an operation separate from multiplication is significant. This issue is remedied when we address how the reciprocal
operation is performed. If we treat the reciprocal operation with as much care as we put into the other operations we achieve acceptable results. In particular we would refine the partition of $1/B$ so that the assumption of piecewise uniformity of distribution is retained. For example, refining the partition of $B$ and $1/B$ modestly increases fidelity considerably,

$$B = \{(4, 5, 6), (40, 50)\} \implies \{(4, \frac{9}{2}, 5, \frac{11}{2}, 6), (20, 20, 25, 25)\}$$
$$1/B = \{\left(\frac{1}{6}, \frac{2}{11}, \frac{1}{5}, \frac{2}{9}, \frac{1}{4}\right), (25, 25, 20, 20)\}$$

A visual comparison of the two partition schemes of $1/B$ in figure 3.21 is illuminating. and the issue is revealed to be one of resolution of the chosen partition of $1/B$. This choice is within our control so we conclude that we can indeed replace expressions such as $A \div B$ with $A \times (1/B)$. More broadly we assert that we can subsume the division operator by the multiplication operator and reduce our supported operations from $(+, -, \times, \div)$ to $(+, \times)$ with the continued understanding that the choice of partition in any computation
will influence its fidelity.

3.3.6 Exponential Function With Two Independent Random Variables

Similarly to the cases $d^A$ and $A^d$ where $A$ is a random variable and $d$ is a real value we numerically compute the case,

$A^B$

where $A$ and $B$ are random variables. Special considerations are required if $\mathbb{P}(A < 0) \neq 0$. As mentioned above this special situation is implemented in the PHoX modeling system, but beyond the scope of this written work.

3.3.7 Sample Results

We adopt the notation that $N(5, 3)$ refers to a continuous random variable with a Gaussian (Normal) probability distribution parametrized by a mean of 5 and a standard deviation of 3. The figure 3.22 shows the numerically calculated product of $N(5, 3)$ and $N(1, 1)$. The figure 3.23 shows the numerically calculated quotient $N(5, 3) \div N(1, 1)$. 

93
Figure 3.22: The Product of Two Independent Normal Random Variables. One with mean=5 and standard deviation=3, the other with mean=1 and standard deviation=1

Figure 3.23: The Quotient of Two Independent Normal Random Variables. One with mean=5 and standard deviation=3, the other with mean=1 and standard deviation=1
Since a random variable may be continuous as well as well as discrete in figure 3.24 we add $N(0, .25)$ and a Bernoulli $B$ random variable such that $P(B = 0) = 0.3$ and $P(B = 1) = 0.7$.

The figure 3.25 shows the Real part of $(-0.95)^{N(0,5)}$. To calculate the result special consideration is made for the clipping problem as mentioned above. A physical interpretation of the result is that it represents the probability distribution of an damped harmonic oscillator observed at a random time in with probability distribution $N(0, 5)$, that is, around $t = 0$.

In figure 3.26 we have three probability distributions. One is the familiar standard Normal, $N(0, 1)$. The other two are the standard Normal raised to 0.95 and 1.1. Notice that the latter two probability distributions are not symmetric. The reason is that the negative
portion of the Normal becomes complex when used as the exponent of a constant. Since 
$Re[(-1)^{.95}] = \cos(.95\pi)$, for example, the negative side is scaled asymmetrically toward 
the origin.

In standard statistics texts such as Bickel [4] and financial texts such as Brealey [6] it 
is common to study the moments of random variables, such as mean, variance, kurtosis, 
etc. In figure 3.27 we show standard Normal random variable raised to a the power of a 
discrete random variable. The result is a mixed discrete-continuous random variables for 
which moment calculations would seem to be misleading.
3.3.8 Computing Net Present Value, An Example

A common business calculation, as may be found in Brigham [7], is that of finding the net present value, or \( NPV \), of a series of cash flows.

There are many sources of uncertainty. There is the uncertainty of future cash flows themselves, the uncertainty of the appropriate discount rate and the uncertainty of when the cash flows will occur in time. As an example we suppose that all these sources of uncertainty are present and that the uncertainty in the magnitude of future cash flows grows with their distance into the future.

Suppose we invest 100 thousand dollars today and expect to receive 25 thousand dollars each year for the next 6 years. The time-value-of-money discount rate is about 10\%. In a typical NPV problem with sharp data the resulting sharp value would be compared to zero, that is, if the sharp NPV is positive then the project requiring this investment and returning the stated cash flows is considered worthwhile. Since we are quantifiably uncertain about the future cash flows and discount rate we encode the problem in PHoX as shown in code listing 3.28. If there were no uncertainty the NPV result for this example would be,

\[
NPV = -100,000 + \sum_{t=1}^{6} \frac{25,000}{(1 + 0.10)^t}
\]

\[
= $8,881.52
\]

In the code listing 3.28 we specify each Normal random variables be sampled with 1000 points, that \( \text{range}(6) = (0, 1, 2, 3, 4, 5) \) and that we’re using a python-based programming language to access the RandomVariable library which is written in Java. The second-last line performs the discount function and is written in a way that seems natural for a programming language. The resulting NPV is shown in figure 3.29.
import RandomVariable

P = -100000
R = 25000
r = .1
NPV = P

for t in range(6):
    N1 = RandomVariable.Normal(R, 2000+300*(t+1), 1000)
    N2 = RandomVariable.Normal(r + .1*t , 0.2+0.1*t , 1000)
    N3 = RandomVariable.Normal(1, .01, 1000)
    NPV += N1/(1+N2)**((t+1)*N3)

NPV.plot("Y")

Figure 3.28: PHoX Calculation of NPV Example

Figure 3.29: Net Present Value
The NPV result using uncertain cash flows and discount rate shown in figure 3.29 has a median value of $8,881.52, the same value as for the sharp version of this example. Taking into account the uncertainty of our input data we may calculate that $P(NPV < 0) = 0.4669365$. This means there is nearly a 47% chance that the proposed project will fail to return positive results. The proposed project is exposed to be a risky investment and that risk is quantified in the PHoX modeling system.

3.3.9 A Comparison to an Alternate Numerical Approach

As mentioned in statistics texts such as Bickel [4], if we want to compute the sum of two independent random variables we can find the resulting probability density by convolving the probability density functions of the two operands. According to signal processing texts such as Oppenheim [41] the computation complexity of this calculation computationally is $O(N^2)$, that is, if we sample the probability density of each operand random variable with $N$ values each. If, again according to Shannon [48], we select a single sample spacing and take $N$ evenly spaced samples from each operand, we may perform a Fast Fourier Transform (FFT) on each operand sample, find the product of the transformed operand samples and compute an inverse FFT on the result, we may compute the convolution operation with $O(N\log(N))$ computational complexity.

Using signal processing to compute convolutions of finite samples to effectively compute the sum of two random variables is computationally attractive. In particular, if the sample frequency is above the Nyquist frequency, according to Shannon [48], the result is arbitrarily accurate, even for finite values of $N$.

As we have explained above, we could use logarithms and exponential functions to turn expressions such as $A \times B$, $A \div B$ and $A^B$ into problems of addition and it would seem that the FFT-based approach suits our needs. It does not for the reasons detailed below.
3.3.9.1 Evenly Spaced Partitions Fail to Cover All Random Variables

The FFT-based approach requires that we represent the probability distributions of random variables with evenly-spaced partitions. For a Normal random variable we could choose a partition such as,

\[ A = \{(\mu - 8\sigma, \ldots, \mu, \ldots, \mu + 8\sigma), (p_1, \ldots, p_n)\} \sim N(\mu, \sigma)\]

That is, our finite partition extends eight standard deviations on either side of the mean. The probability not covered by this partition is \( P(A < \mu - 8\sigma) + P(\mu + 8\sigma < A) \approx 0. \)

For quotient-derived distributions like the \( F \)-distribution or the Cauchy an evenly space partition provides unsatisfactory results. In the case of the \( F \)-distribution (actually a parametric family of distributions) we may start the partition at zero since the \( F \)-distribution is non-negative. Choosing an upper-bound for the partition, which implies a specific spacing for fixed partition size, is problematic.

As specific exemplary example is to choose \( N = 1000 \) of an \( X = F(5, 2) \) random variable, that is and \( F \)-distribution with 5 primary and 2 secondary degrees of freedom. If we choose the upper bound of \( x = 1000 \) then the partition spacing is one. In figure 3.30 we show the probability distribution of \( X \). We have,

\[ Z = \{(0, 1, 2, \ldots, 999, 1000), (0.43, .20, \ldots, 1E - 6)\}\]

\[ P(Z > 1000) = .001 \]

This means that the first two partition intervals contain 63% of the overall probability and we have failed to represent .001 of the total probability since it is beyond our partition
upper bound. If we choose a smaller upper bound so that we capture more samples where the \( F(5, 2) \) distribution varies the most then we only increase the amount of probability above the upper bound and therefore not represented.

In the PHoX modeling system we choose a partition that balances the need to capture more points where the distribution varies most and yet having a large upper bound. The figure 3.30 was created with a PHoX partition of 1000 points. The upper bound of the PHoX partition is \( 4.13E13 \) and there is no numerically detectable probability unrepresented. In figure 3.31 we show the same \( X = F(5, 2) \) distribution, but only for \( 0 < X < 0.5 \) to demonstrate the fidelity that non-even sample spacing can achieve.

Since the PHoX modeling system must represent a wide and undefined set of random variables we cannot restrict partitions to even spacing. This is one reason that the FFT-
based approach is non-viable.

3.3.9.2 Evenly Spaced Partitions Fail to Extend Beyond Addition

The PHoX modeling system must support operations on random variables beyond addition and subtraction. The FFT-based approach requires that we represent random variables with an evenly spaced partition. If we wish to multiply two independent random variables \( X \) and \( Y \) we need to first transform each using a logarithm. That is,

\[
X \times Y = \exp(\log(X) + \log(Y))
\]

where we perform the addition operation on \( \log(X) \) and \( \log(Y) \). Suppose, for example, that the we represent \( X \) and \( Y \) as follows,

\[
X = \{(1, 2, 3), (p_1, p_2)\}
\]

\[
Y = \{(4, 5, 6), (q_1, q_2)\}
\]

then we have,

\[
\log(X) = \{(0, 0.693, 1.099), (p_1, p_2)\}
\]

\[
\log(Y) = \{(1.386, 1.609, 1.792), (q_1, q_2)\}
\]

but \( 0.693 \neq (1.099 - 0.693 = .405) \) and \( (1.609 - 1.386 = .223) \neq (1.792 - 1.609 = .182) \). The partitions for \( \log(X) \) and \( \log(Y) \) are therefore not evenly spaced. If we try to
impose an evenly spaced partition on $\log(X)$ and $\log(Y)$ by re-assigning the probability values $p_1, p_2$ and $q_1, q_2$ then we have partitions that are no longer synchronous with their parent random variables $X$ and $Y$.

It is the opinion of the author that the FFT-based approach is not appropriate for use in the PHoX modeling system.

3.3.10 Arithmetic Correlation of Random Variables

Since we assume a priori that all random model inputs are pairwise independent there seems no compelling reason to consider correlation of random variables. Consider, however, the two random variables,

$$Y = X + X$$

$$Z = 2X$$

If we let $X \sim N(1, 1)$ and use PHoX to compute $Y$ and $Z$ we find the two random variables differ significantly in figure 3.32.

The reason that $X + X \neq 2X$ according to the implementation of PHoX we have described thus far is that PHoX has falsely assumed that the two random variables in the
expression $X + X$ are independent. Symbolically we see they are not only dependent, but identical.

Since PHoX is equipped with a compiler (see: Aho [1]) it is able to convert the expression $X + X$ into $2X$ when so instructed. While some expressions may be converted into equivalent forms that PHoX can compute correctly while still assuming independence of operand random variables in arithmetic expressions, not all expressions can be addressed in this manner.

For example we consider the expression,

$$X = A + \frac{A^2}{B}$$

where $A$ and $B$ are independent random variables. We could factor out the $A$ term to form,

$$X = A \times \left(1 + \frac{A}{B}\right)$$

but we still have to problem that $A$ is being multiplied by a random variable expressed as $(1 + A/B)$ that is not independent of $A$ so none of what we have presented thus far informs us how to perform the multiplication to find the $X$ result. We say that $A$ and $(1 + A/B)$ are \textit{arithmetically correlated} because they have at least one common random variable in their description and we refer to expressions whose sub-expressions are arithmetically correlated as \textit{arithmetically correlated expressions}.

In a later section we ask what other expressions similar to $X$ must PHoX support. We then consider what changes we must make to the implementation of PHoX so that
it supports the arithmetic correlations we anticipate the expected PHoX modeling system user-base to request.

We will find that arithmetically correlated expressions are denumerable, but that their set is infinite. We will also find that a specific expansion of the PHoX modeling system allows it to handle an acceptably wide range of arithmetically correlated expressions.
4.1 Revisitations’ Goal

Our goal is to replace all model inputs that demonstrate uncertainty with their random variable representatives and run the model as if all the model inputs were sharp. Since we will allow the PHoX modeling system to treat the model as a glass box and examine its contents before running the model itself, we must have access to the software description of the model. If the software description of the model is in a form recognizable by the PHoX modeling system, then it will recompile that model and allow for random variable inputs. See Aho [1] for a description of the process of compiling software from descriptive to operative forms.

We refer to the process of replacing model inputs with random variables and running the upgraded model as Random Variable Model Analysis.

To understand what types of operations involving random variables the PHoX modeling system must support, we use our reference implementation of the AB32 model as our guide. Recall that the reference implementation of the AB32 model, according to the standard set by Greenberg [26], represents a wide class of models from many disciplines.
4.1.1 Model Inputs

The reference implementation of the AB32 model draws its input form a database of carefully gathered information. In general we may regard this input database as a set of tables of values (text and numbers). To represent a given policy, the AB32 model also gathers input from a configuration data set which details such instructions as which economic sectors will participate in a cap-and-trade scheme, how many reductions will be required at the end of each period, etc. In general we have a set of numerical and textual data with high-level instructions on how to process that data.

Beyond input data and configuration information are detailed processing instructions. The PHoX model partitions the work of processing the input data into batches that roughly mirror the economic sectors seen in the data. The collection of these code batches constitute the pre-processing phase of the PHoX model. The pre-processing code is what loads in the raw data, reads a relevant portion of the configuration data set and produces a structured body of data which we may describe as standardized. Specifically, PHoX reduces all carbon abatement projects to unit abatement projects with a (levelized) cost per unit of reduction per year (referred to as cost-per-ton or cpt) and capacity up to the given period.

Once PHoX has standardized all participating carbon abatement projects it combines them with the policy-dictated production rules and then finds the optimal buying strategy for the market under the assumptions of a cost-effective market with perfect cooperation. Whether the AB32 model results reflect a real market is not our concern at this stage. What we want to know is if we can replace the data inputs with random variables and then run the upgraded model. To find out we will detail a typical path that a set of input data may take through the PHoX model or through a more general PHoX-like model.
4.1.2 Capacity

In the AB32 model the capacity for a unit abatement project is computed differently for each economic sector. For agriculture it’s a simple table look-up as in,

\[ \text{capacity} = \text{TABLE} ["\text{Total tCO2 Potential}"] \]

where \( \text{TABLE} \) is a PHoX function that accesses a data table based on the context of which sector is being processed and the parameter “Total tCO2 Potential” is the column heading representing the capacity for a number of agriculture projects being processed. The result is that \( \text{capacity} \) is a variable representing a list of data values.

This agriculture capacity case is the simplest possible. Replacing the \( \text{capacity} \) values with random variables can be done in many ways. One obvious way is to use multiple columns of data instead of one and encode parameters for a random variable. For example, instead of encoding \( \text{capacity} = (5, 7, 3) \) we encode \( \text{capacity} = ((5, 1, \text{normal}), (7, 1.2, \text{lognormal}), (3, 0.2, \text{normal}) \) where \( (5, 1, \text{normal}) \) represents a Normal (Gaussian) random variable with a mean of 5 and a standard deviation of 1. The specific random variable upgrade mechanism is not important; what matters is that the upgraded version of PHoX (called RticPHoX) reads random variables from the data tables instead of just numbers. The result in this case is that the variable \( \text{capacity} \) is now a list of random variables.

Looking at the forestry sector in the AB32 model we find that \( \text{capacity} \) is computed rather than simply referenced form the database,

\[
\begin{align*}
\text{co2\_ha} &= \text{TABLE} ["\text{Sequestration Per Hectare}"] \\
\text{h\_pot} &= \text{TABLE} ["\text{Hectare Potential}"] \\
\text{capacity} &= \text{co2\_ha} \times \text{h\_pot}
\end{align*}
\]

If we assume now that RticPHoX will use the \( \text{TABLE} \) function to create a list of random variables from the database based on a title such as “Sequestration per Hectare” within the context of forestry-sector pre-processing then we see our first combination of random
variables, multiplication. Since list multiplication is assumed to be performed element-by-
element we have no difficulties here.

Reviewing the other pre-processing sectors in the AB32 model we find more compli-
cated examples for computing capacity such as from the electricity sector,

\[
\begin{align*}
\text{mw\_potential} & = \text{TABLE["MegaWatt Potential"]} \\
\text{mw\_capacity} & = \text{TABLE["Capacity"]} \\
\text{capacity\_factor} & = \text{TABLE["Capacity Factor"]} \\
\text{project\_kwh} & = \text{capacity\_factor} \times \text{HRS\_YEAR} \\
\text{fossil\_fuel\_eff} & = \text{fossil\_fuel\_ef} \times \text{btu\_kwh} \\
\text{b\_fossil\_fuel\_eff} & = \text{b\_fossil\_fuel\_ef} \times \text{btu\_kwh} \\
\text{b\_co2\_mwh} & = \text{b\_fossil\_fuel\_ef} \times \text{b\_btu\_kwh} \\
\text{a\_co2\_mmmbtu} & = \text{fossil\_fuel\_eff} \times \text{factor} \\
\text{co2\_red\_mwh} & = \text{b\_co2\_mwh} - \text{a\_co2\_mmmbtu} \\
\text{capacity} & = \text{co2\_red\_mwh} \times \text{mw\_potential} \times \\
& \quad \text{project\_kwh} \times \text{MWH\_KWH}
\end{align*}
\]

We have not shown the equation for every variable above, but selected a representative
subset. All the operations involved are either arithmetic combinations of lists of random
variables or simple database look-ups. The variable names are cryptic, but meaningful
to the carbon abatement practitioner. The electricity sector calculations are complicated
largely because they explore projects involving fuel switching and must account for usage
efficiency, thus the suffix’s \_ef and \_eff.
Lastly, some of the variables in the above code are certainly not random variables. In particular capitalized variable names imply by convention represent physical or fixed constants. For example, \( KG\_TONNE \) represents the number of kilograms in a metric tonne and is clearly not random.

In each example above, \( capacity \) represents a list of random variables that will be used by the optimization engine.

An important point we have not yet explored is that of random variable independence. We will assume throughout this dissertation that any random variables that replace input data are mutually independent. In the case of \( capacity \) each value in the list represents the period capacity of an individual project. Since every operation used to compute \( capacity \) is element-wise we must have that the random variables in the \( capacity \) list are mutually independent. This is not to say that mutual independence within the random variables comprising the \( capacity \) list is guaranteed or required for all PHoX-like models, but simply that we have not yet encountered in this model revisitaiton a case implying random variable correlation. We will see below that this happy situation will not persist.

4.1.3 Levelized Costs

Each pre-processing sector in the AB32 model computes two variables, \( cpt \) and \( capacity \). Each variable represents a list of values, one per possible carbon abatement project. Upon upgrading to random variable input data then each variable represents a list of random variables. In this section we consider the production of the \( cpt \) variable. We will find that, in the AB32 model at least, its production is significantly more complex. One of the simplest expressions of \( cpt \) in the AB32 model is found in the forestry pre-processing sector,

\[
R = \text{Lookup(”UserConfig”, ”Discount Rate”)}
\]
\text{lifetime} = \text{TABLE["ProjectLifetime"]}

\text{cap\_costs} = \text{money(\text{TABLE["Capital Costs"]})}
\text{o\_m} = \text{money(\text{TABLE["O\_M"]})}
\text{cost} = \text{cap\_costs} + \text{o\_m} \times \text{NPV(R, lifetime)}

\text{co2\_ha} = \text{TABLE["Sequestration Per Hectare"]}
\text{co2} = \text{co2\_ha} \times \text{NPV(R, lifetime)}

\text{cpt} = \frac{\text{cost}}{\text{co2}}

where \text{TABLE()} is a PHoX function for looking up lists of data from the database tables, \text{money()} converts nominal values of money to real values given two years found from context. The \text{NPV()} function accepts a discount rate \text{R} and a either a sequence of cash flows or a number of years such as the project \text{lifetime} if cash flows are assumed identical and return the net present value of those cash flows. An expression such as \text{om\_cost} \times \text{NPV(R, lifetime)} accepts a list of values representing the operating and maintenance costs of each project and found the net present value of these costs assuming they are incurred each year throughout each project lifetime. Here, \text{lifetime} is a list of values representing the lifetime of each project under consideration.

We’ve emphasized the list-processing nature of each arithmetic operation, but we set it aside to more clearly understand what is at issue.

The net present value of a project is the some of a time-value-of-money discounted sequence of cash flows (See Brigham [7]). In the code above we see capital or initial costs \text{cap\_costs} followed by a sequence of identical operating and maintenance (\text{o\_m}) expenses for each period over the projected \text{lifetime} of each project. Similarly, the net present value
of the sequence of anticipated $\text{co}_2$ reductions for each project are discounted. These calculations happen to be stated per hectare (indicated by the _ha suffix).

So far there is no indication that we need concern ourselves with any random variables being dependent on any others when we replace the input data with random variables. The final variable $cpt$ will be a list of random variables which depend on all their constituent random variables, but are independent of each other. At no point in the production do we see any evidence of two correlated random variables being combined.

As we consider the more general case for PHoX-like models we ask if we can guarantee that arithmetic combinations and comparisons of two random variables will only involve independent random variables. A second look at the production of $cpt$ suggests we may have violated this hope-for guarantee.

Consider the two variables $\text{cap\_costs}$ and $\text{co2\_ha}$ that figure prominently in the production of $cpt$, the levelized cost of each prospective project. It seems obvious that $\text{cap\_costs}$ and $\text{co2\_ha}$ are correlated. The former is the initial investment required for a given project and the latter is the amount of $\text{co}_2$ abated by the project. The bigger the project the more $\text{co}_2$ abated and the more expensive the initial investment. On the other hand, investing more in a project does not guarantee a larger return (in the form of greater $\text{co}_2$ abatement). We conclude that the two variables ($\text{cap\_costs}$ and $\text{co2\_ha}$) are most likely correlated, but not necessarily perfectly correlated (i.e. related by a bijective function).

Since our target variable, $cpt$, is presumed to be produced by the arithmetic (quotient) combination of two correlated random variables we must make an effort to investigate the implications of this situation. We will address this issue of arithmetic combinations of correlated random variables below. In the meantime we continue of revisiting of the AB32 model model to uncover other issues we need to address before we can replace input values with random variables and run the upgraded PHoX model which will be known as RticPHoX.
4.1.4 Forecast Costs and Capacity

Notice that the production of the \textit{cpt} and \textit{capacity} variables as described above apply to a single period, yet the AB32 model is a multi-period model. In the case of the AB32 model there are nine years/periods represented. This means that we need to produce nine copies of the \textit{cpt} and of the \textit{capacity} variables.

The AB32 model is designed to study a specific period of time, 2012-2020. This implies that any values computed for \textit{cpt} or \textit{capacity}, are necessarily uncertain because they all represent forecasts.

The AB32 model takes a very simple approach to forecasting; it leaves it to others. Two estimates of each piece of input data are made, one near-term or current and the other longer-term. Given these two points for each piece of input data, a value for any date may be interpolated from the line through the two given points. For example, if a particular project is estimated to have a capacity of 100 tonnes of \textit{co}$_2$ abatement in the near-term and 150 tonnes in 10 years then the AB32 model would assume it has 125 tonnes of capacity in 5 years and 200 tonnes in 20 years, in other words, a first-order approximation.

Next we consider what methods PHoX-like models might employ to model forecast data. A likely source of input is from financial engineering (see Brealey/Myers [6]) where each piece of input data is assigned a stochastic process. This approach may choose to assume that amounts of abatement capacity or project price components evolve in time independently, which seems unlikely. Another approach may be to accept external expert forecasts, but not treat them as perfect (or sharp), but introduce a measure of uncertainty in the local model.

Regardless of the forecasting methodology chosen for PHoX-like models we can make some reasonable suppositions. Suppose that one of our model inputs represents the hectares of available land for afforestation (tree planting) \textit{co}$_2$ abatement projects. This number may
be subject to government action, that is, there may be legislation pending or proposed that will increase or reduce the amount of land available for afforestation projects. The amount of afforestation-ready land may be known precisely today and the value available in some future period is partially subject to a (from our point of view) random process. We have two random variables, the former trivially random or sharp and the latter correlated to the former. The argument again is in favor of understanding the arithmetic combination and binary comparison of correlated random variables since they appear to be ubiquitous.

4.2 The Simplex Method

The PHoX model employs linear programming as its main optimization technique. There are also some other areas where it’s used within the model. There are two types of optimization algorithms for solving a linear program, the simplex methods and interior point methods. This article will explore the details of the original simplex method invented by George Danzig during WWII.

Our goal is not to explain the simplex method, but to understand the effect that random variable inputs will experience when applied to the simplex method. The overall goal is to understand what happens with random variables are used as model inputs instead of fixed numbers and in the PHoX model this includes passing though an linear programming optimization engine. As described above the specific algorithmic details matter right down to the basic arithmetic operations.

4.2.1 A Working Example

With hundreds or even thousands of variables corresponding to as many dimensions finding a realistic example that can be illustrated is challenging. Limiting the example to two variables and using PHoX as our guide means either optimizing the choice between
two project options over a single period or a single option over two periods. This latter choice seems more interesting. Beginning with a generic linear program statement,

\[
\min \{ c^T x \mid Ax \leq b \}
\]

Let \( x = x_1, x_2 \) represent the amount of investment in a single project per period in each of two subsequent periods where 1 unit represents the sum of money expected to be required to produce 1 ton of \( CO_2 \) emissions reduction. This exactly matches the pattern of the PHoX model.

Let \( c = c_1, c_2 \) be the $-per-ton levelized cost for the the emission reduction project in each of the two periods. Recall that levelized cost include assumptions such as project lifetime and discount rate of emissions reductions and of maintenance costs as well as rolling in the initial project investment.

Before defining \( A \) and \( b \) we introduce two kinds of constraints. The first is the capacity constraint,

\[
x_1 \leq q_1
\]
\[
x_1 + x_2 \leq q_1 + q_2
\]

where \( q = (q_1, q_2) \) is the maximum tons of reductions the project can produce in each period. If \( q_2 > 0 \) then the project has extra capacity available in the second period. The second constraint is the shortfall constraint,
\[ x_1 \geq d_1 \]
\[ 2x_1 + x_2 \geq d_2 \]

where \( d = (d_1, d_2) \) is the tons of emissions reductions above BAU in each period. Since \( x_1 \) is invested in the first period it is assumed to be invested again in the second period since all investments are assumed recurring unless stated otherwise. The PHoX model does allow for short-run investments, not represented here.

The simple method assumes implicitly that \( x_1, x_2 \geq 0 \) so we need not include that as a constraint. Finally we may compose our constrains,

\[ x_1 \leq q_1 \]
\[ x_1 + x_2 \leq q_1 + q_2 \]
\[ -x_1 \leq -d_1 \]
\[ -2x_1 - x_2 \leq -d_2 \]

into \( A \) and \( b \) form,

\[
A = \begin{pmatrix}
1 & 0 \\
1 & 1 \\
-1 & 0 \\
-2 & -1 \\
\end{pmatrix}
\]

\[
b = \begin{pmatrix}
q_1 \\
q_1 + q_2 \\
-d_1 \\
-d_2 \\
\end{pmatrix}
\]

116
4.2.2 The Tableau

The simplex method operates on an array of vectors forming a tableau. There are two steps to create the initial tableau and some recurrent steps to run the algorithm detailed below.

The first step is to introduce slack variables to turn our constraint inequalities into equalities. One slack variable for each equality. There our four inequalities in the working example so we introduce four slack variables $s_1, s_2, s_3, s_4$. Our four specific constraints become,

\[
\begin{align*}
    x_1 + s_1 &= q_1 \\
    x_1 + x_2 + s_2 &= q_1 + q_2 \\
    -x_1 + s_3 &= -d_1 \\
    -2x_1 - x_2 + s_4 &= -d_2
\end{align*}
\]

Given the new variables our $A$ matrix and updated $x, c$ and $b$ vectors becomes,

\[
\tilde{A} = \begin{pmatrix}
    1 & 0 & 1 & 0 & 0 & 0 \\
    1 & 1 & 0 & 1 & 0 & 0 \\
    -1 & 0 & 0 & 0 & 1 & 0 \\
    -2 & -1 & 0 & 0 & 0 & 1
\end{pmatrix}, \quad \tilde{x} = \begin{pmatrix}
    x_1 \\
    x_2 \\
    s_1 \\
    s_2 \\
    s_3 \\
    s_4
\end{pmatrix}, \quad \tilde{c} = \begin{pmatrix}
    c_1 \\
    c_2 \\
    0 \\
    0 \\
    0 \\
    0
\end{pmatrix}, \quad \tilde{b} = \begin{pmatrix}
    q_1 \\
    q_1 + q_2 \\
    -d_1 \\
    -d_2
\end{pmatrix}
\]

The simplex method works by finding starting feasible point which is also a vertex of
the constraint polytope (intersection of half-spaces represented by our constraint inequalities). The constraint equations are rewritten by applying the inverse of a full-rank square matrix $B^{-1}$ to each side of our set of equations represented by $\tilde{A}\tilde{x} = b$ as in,

$$B^{-1}\tilde{A}\tilde{x} = B^{-1}b$$

The simplex method applies a sequence of $B^{-1}$ matrices to each side of our constraint equations. Each such $B$ may be subscripted so that at stage 3, for example, we have,

$$B_3^{-1}B_2^{-1}B_1^{-1}\tilde{A}\tilde{x} = B_3^{-1}B_2^{-1}B_1^{-1}b$$

Let $b_i = B_i^{-1}\ldots B_1^{-1}b$. It will be made plain below why the requirement of feasibility is exactly the requirement that $b_i \geq 0$. Taking this for granted for the moment motivates us to notice that the $b$ in our example violates feasibility because $-d_1, -d_2 < 0$. To remedy this the constraint equations are rewritten again and two new artificial variables are introduced, $f_1, f_2$,

$$x_1 + s_1 = q_1$$
$$x_1 + x_2 + s_2 = q_1 + q_2$$
$$x_1 - s_3 + f_1 = d_1$$
$$2x_1 + x_2 - s_4 + f_2 = d_2$$

The constraints are encoded as,
\[ A'x' = b' \]

where,

\[
A' = \begin{pmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0 & 0 & -1 & 0 & 1
\end{pmatrix}
\]

and,

\[
x' = \begin{pmatrix}
x_1 \\
x_2 \\
s_1 \\
s_2 \\
s_3 \\
s_4 \\
f_1 \\
f_2
\end{pmatrix}, \quad
c' = \begin{pmatrix}
c_1 \\
c_2 \\
0 \\
0 \\
0 \\
0 \\
w \\
w
\end{pmatrix}, \quad
b' = \begin{pmatrix}
q_1 + q_2 \\
q_1 \\
d_1 \\
d_2
\end{pmatrix} = b
\]

Notice one detail that the cost (in the \(c\) vector) of each of the two artificial variables \(f_1, f_2\) is \(w\). The goal is to minimize \(c'^T x'\) and the artificial variable must be zero at that point so they are assigned infinite cost. We can’t actually put \(\infty\) into the computer program since because we want \(w * f_1 = 0\) and we can’t guarantee that \(\infty * 0 = 0\).
We will need to handle $w$ specially. We’ll find that in practice the two variables $f_1, f_2$ are optimized away almost right away since they represent such huge cost and once they’re set to zero we can remove them from the analysis.

Our initial tableau shown in figure 4.1 is formed by creating a row of variables corresponding to $x'$, an array corresponding to $A'$ with an additional column corresponding to $b'$. There is one extra column described in table 4.1.

The first column is called the *basis vector*. Notice that each each equation there is one variable that appears nowhere else and has a unit coefficient.

The simplex method makes the assumption that all variables not in the *basis* are zero. This means in this initial tableau that $s_1 = q_1, s_2 = q_1 + q_2, f_1 = d_1, f_2 = d_2$ and the other variables including $x_1$ and $x_2$ are zero. This is solution to our constraint equations and therefore has a cost $c^T x' = 0 + 0 + w + w = 2w$. Since $w$ is our virtual infinity this solution is unlikely to be optimal. The point is that we can arrange for other combinations of four variables to form our *basis* which may yield lower costs and in a linear program only these extreme points (vertices of the constraint polytope) need be considered.

### 4.2.3 A Graphical Example

The working example was chosen to be simple enough to graph. A few assumptions must be made to get an image. If we assume $d_1 < q_1 < q_2 < d_2 < 2q_1$ then we get the graph in figure 4.1.

The feasible polytope is the shaded region. Depending on the relationship between

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$q_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$q_1 + q_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$d_1$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

Table 4.1: Initial Simplex Tableau
\( c_1, c_2 \) and \( d_1, d_2 \) we may find either circled points 1 or 2 as optimal. Let’s leave a little to mystery and press on with the optimization.

4.2.4 The First Round

Returning to our initial tableau where \((s_1, s_2, f_1, f_2) = b_0\) where \(b_i\) is defined above as the current \(b\) vector after \(i\) transformations as yet to be described properly. Nonetheless, since all the other variables are zero and \(b_0 \geq 0\) all the constraints are satisfied so the result our \(x_i\) (following the same subscripting convention) must be feasible. In this case we have,

\[
 x_0 = \begin{pmatrix} 0 & 0 & q_1 & q_1 + q_2 & 0 & 0 & d_1 & d_2 \end{pmatrix}^T 
\]

Notice that the vertices of our feasible polytope happen at the intersections of the con-
Graphically, the equations are similarly labeled in figure 4.2.

The situation in the initial tableau is not as easy to picture because of the artificial variables. Suppose instead we consider the situation were we are at the intersection of E2 and E3 (labeled as point 1). This means these two constraints are active implying we are at their limit. The other constraints are satisfied, but otherwise inactive. An equation is active
if there is no slack available. In this case would have $s_3, s_4, f_1, f_2 = 0$ and $x_1, x_2, s_1, s_2 \geq 0$. We would then have $x_1 = d_1$ and $x_2 = d_2 - 2d_1$ in order to satisfy the updated equations,

\[
\begin{align*}
    x_1 + s_1 &= q_1 & E1 \\
    x_1 + x_2 + s_2 &= q_1 + q_2 & E2 \\
    x_1 &= d_1 & E3 \\
    2x_1 + x_2 &= d_2 & E4
\end{align*}
\]

Graphically it’s easy to see that the other two equations (E1, E2) are also satisfied.

The key observation is that equations E3 and E4 are active precisely because variables $s_1$ and $s_2$ are zero and this is because they are not in the basis.

Now we see that active constraints imply position (at a vertex) and that basis membership implies inactive constraints so if we want to move from one vertex to another we need to exchange one basis variable for one non-basis variable. This involves three basic steps.

Step 1. Select a non-basis variable to enter the basis. This is called the entering variable.

Step 2. Select a basis variable to exit the bases and become a non-basis variable. This is called the exiting variable. Step 3. Create as basis transforming matrix $B_i$ and apply its inverse to the constraint equations.

4.2.4.1 Step 1

To find the non-basis variable we consider the impact of our choice. Any variable we choose is allowed to increase from its assumed value of zero. If we choose a variable like $x_1$ or $x_2$ then the change in cost will be $c_1$ or $c_2$ respectively. If we choose one of the slack variables there’s no direct cost impact. Finally note that we will always begin with the artificial variables like $f_1$ and $f_2$ in the basis and because of the virtually infinite cost
impact never choose them once removed from the basis.

On the other hand we must consider the indirect cost impact of our choice of non-basis variable. Consider choosing $x_1$ in our example. Here’s the equations in their current form with non-basis variables set to zero except for $x_1$,

\begin{align*}
x_1 + s_1 &= q_1 & \text{E1} \\
x_1 + s_2 &= q_1 + q_2 & \text{E2} \\
x_1 + f_1 &= d_1 & \text{E3} \\
2x_1 + f_2 &= d_2 & \text{E4}
\end{align*}

If $x_1$ is allowed to increase then E1 tells us that $s_1$ will decrease at the same rate. Similarly, E2 says $s_2$ will decrease at the same rate, E3 says $f_1$ will decrease at the same rate and finally E4 tells us that $f_2$ will decrease at twice the rate that $x_1$ increases.

Increasing or decreasing slack variables has no cost impact, but decreasing artificial variables has a virtually infinite impact. Specifically, the cost impact of choosing $x_1$ and thereby allowing it to increase is,

$$Z(x_1) = c_1 - (1 \times 0 + 1 \times 0 + 1 \times w + 2 \times w) = -3w$$

using this implied cost impact function $Z(.)$ for the other non-basis variables,
\[
Z(x_2) = c_2 - (0 \cdot 0 + 1 \cdot 0 + 0 \cdot w + 1 \cdot w) = -w
\]
\[
Z(s_3) = 0 - (0 \cdot 0 + 0 \cdot 0 + (-1) \cdot w + 0 \cdot w) = w
\]
\[
Z(s_4) = 0 - (0 \cdot 0 + 0 \cdot 0 + 0 \cdot w + (-1) \cdot w) = w
\]

The greatest decrease in cost is from choosing \(x_1\) as basis vector since \(-3w < -w < w\).

4.2.4.2 Step 2

Next we must choose an exiting variable. Now that we know our entering variable is \(x_1\) we ask how much \(x_1\) has to increase to force one of the basis variables to zero. Returning to the current set of equations,

\[
x_1 + s_1 = q_1 \quad E1
\]
\[
x_1 + s_2 = q_1 + q_2 \quad E2
\]
\[
x_1 + f_1 = d_1 \quad E3
\]
\[
2x_1 + f_2 = d_2 \quad E4
\]

Equation E1 tells us that if \(x_1\) increase from zero to \(q_1\) this will force \(s_1\) to be zero. Equation E2 tells us that if \(x_1\) increases to \(q_1 + q_2\) the \(s_2\) will be forces to zero, but since we assume \(q_1 < q_1 + q_2\) then \(x_1\) will send \(s_1\) to zero before \(s_2\). Equation E3 tells us that if \(x_1\) increases to \(d_1\) then \(f_1\) will be forced to zero and E4 tells us that if \(x_1\) increases to \(d_2/2\) then \(f_2\) will be forced to zero. By our assumptions in this example we see that \(d_1 < d_2/2 < q_1 < q_1 + q_2\). We conclude that \(f_1\) will be the first sent to zero as \(x_1\) increases.
and therefore we choose \( f_1 \) as our exiting variable.

In general we divide the current \( b \) by the coefficients of the entering variable and choose the variable corresponding to the minimum of these quotients as the exiting variable.

4.2.4.3 Step 3

We form our basis transforming matrix \( B_1 \) by inserting the coefficients of our entering variable in place of those of the exiting variable. Since \( x_1 \) is entering and \( f_1 \) is exiting we have \( B_1 \) and its inverse,

\[
B_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix}, \quad B_1^{-1} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{pmatrix}
\]

We use \( B_1^{-1} \) to rewrite our equations as in,

\[
B_1^{-1} A' x' = B_1^{-1} b'
\]

and our tableau is shown in figure 4.2 where each column vector has been multiplied on the left by \( B_1^{-1} \) and the basis is updated to \((s_1, s_2, x_1, f_2)\) and we’ve dropped \( f_1 \) since it will never be re-chosen.

4.2.5 Another Round, Faster

From the tableau we take our three step more quickly to see how the algorithm works in practice.
Step 1: find the least cost (entering) variable,

\[
\begin{align*}
Z(x_1) &= c_1 -(0 \cdot 0 + 0 \cdot 0 + 1 \cdot c_1 + 0 \cdot 0) = 0 \\
Z(x_2) &= c_2 -(0 \cdot 0 + 1 \cdot 0 + 0 \cdot c_1 + 1 \cdot w) = -w \\
Z(s_1) &= 0 -(1 \cdot 0 + 0 \cdot 0 + 0 \cdot c_1 + 0 \cdot w) = 0 \\
Z(s_2) &= 0 -(0 \cdot 0 + 1 \cdot 0 + 0 \cdot c_1 + 0 \cdot w) = 0 \\
Z(s_3) &= 0 -(1 \cdot 0 + 1 \cdot 0 - 1 \cdot c_1 + 2 \cdot w) = -2w \\
Z(s_4) &= 0 -(0 \cdot 0 + 0 \cdot 0 + 0 \cdot c_1 - 1 \cdot w) = w \\
Z(f_2) &= 0 -(0 \cdot 0 + 1 \cdot 0 + 0 \cdot c_1 - 1 \cdot w) = -w
\end{align*}
\]

Since \(-2w < -w < 0 < w\) we choose \(s_3\) as the entering variable. Notice that we need never consider basis variables in this cost calculation for two reasons; (1) they are already in the basis and therefore cannot be chosen and (2) their cost impact is always zero and we would terminate the processes if we could find no negative cost impact variables.

Step 2: Find the least positive ratio (exiting) variable.

We divide the \(b\) column element-wise by the \(s_3\) column and find which row is the least positive value,
Table 4.3: Second Step Simplex Tableau

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>0</td>
<td>-1/2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>$q_1 - d_2/2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1/2</td>
<td>$q_1 + q_2 - d_2/2$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1/2</td>
<td>$d_2/2$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1/2</td>
<td>$d_2/2 - d_1$</td>
</tr>
</tbody>
</table>

exiting variable = \( \text{argmin}(\frac{q_1 - d_1}{1}, \frac{q_1 + q_2 - d_1}{1}, \frac{d_1}{-1}, \frac{d_2 - 2d_1}{2} | \text{positive}) \)

Since \( d_1/(-1) < 0 \) the corresponding variable \( x_1 \) is discarded as an option. Since \( (q_1 - d_1) < (q_1 + q_2 - d_1) \) the corresponding variable \( s_2 \) is discarded. We are left to discover the relationship of \( (q_1 - d_1) \) and \( (d_2/2 - d_1) \). Since \( d_2/2 < q_1 \) we choose the corresponding variable \( f_2 \) as the exiting variable.

Step 3: Form basis change matrix and apply.

Again, we remove the \( f_2 \) variable in favor of the \( s_3 \) variable and will not mention \( f_2 \) any further.

\[
B_2 = \begin{pmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & -1 \\
0 & 0 & 0 & 2
\end{pmatrix}
\]

\[
B_2^{-1} = \begin{pmatrix}
1 & 0 & 0 & -1/2 \\
0 & 1 & 0 & -1/2 \\
0 & 0 & 1 & 1/2 \\
0 & 0 & 0 & 1/2
\end{pmatrix}
\]

Applying \( B_2^{-1} \) to both sides of our constraint equation as before we update the vectors in our tableau shown in table 4.3.

Recalling that non-basis variables are assume zero we can read the equations from the tableau and see that \((s_1, s_2, x_1, s_3) = b\). In particular notice that \( x_1 = d_2/2 \) and \( x_2 = 0 \).
In our original constraint figure this corresponds to point 2. We’ve reached a critical point where we may stop or continue depending on the relationship between cost and shortfall. Let’s have one more round...

4.2.6 The Completion Round

Step 1: Try to find an exiting variable,

\[ Z(x_2) = c_2 \quad -(-1/2 \times 0 + 1/2 \times 0 + 1/2 \times c_1 + 1/2 \times 0) = c_2 - c_1/2 \]
\[ Z(s_4) = 0 \quad -(1/2 \times 0 + 1/2 \times 0 + -1/2 \times c_1 - 1/2 \times 0) = c_1/2 \]

Since \( c_1/2 > 0 \) we would not choose the corresponding \( s_4 \) as an entering variable. What we have not decided is whether \( c_2 < c_1/2 \) or not.

If \( c_2 \geq c_1/2 \) then we are done and the optimal solution is \((x_1, x_2) = (d_2/2, 0)\) which we may call the *invest early* strategy. The cost of this strategy is \( c_1 \times d_2/2 + c_2 \times 0 \).

If \( c_2 < c_1/2 \) then we are in a situation of rapidly falling prices and it is better to buy only what you need right way. In terms of the simplex method this means we choose \( x_2 \) as our entering variable. Let’s make this choice and finish up,

Step 2: Choose the exiting variable given \( x_2 \) is entering,

\[
\text{exiting variable} = \text{argmin}(d_2 - 2q_1, 2q_1 + 2q_2 - d_2, d_2, d_2 - 2d_1 \mid \text{positive})
\]

Since \( d_2 - 2q_1 < 0 \) it’s ignored. Since \( 0 < d_2 - 2d_1 < d_2 \) we eliminate \( x_1 \) from consideration. We need then to compare \( 2q_1 + 2q_2 - d_2 \) and \( d_2 - 2d_1 \). Since \( d_2 - d_1 < q_1 + q_2 \)
we choose \( s_3 \) as the exiting variable. We know from previous arguments that this means \( s_3 \) and \( s_4 \) will be non-basis variables (and therefore zero) so that E3 and E4 will be active which corresponds to point 1 in the constraint figure. Since we’re so close let’s take the third step and find the answer.

Step 3: Form the basis change matrix given \( x_2 \) replacing \( s_3 \),

\[
B_3 = \begin{pmatrix}
1 & 0 & 0 & -1/2 \\
0 & 1 & 0 & 1/2 \\
0 & 0 & 1 & 1/2 \\
0 & 0 & 0 & 1/2 \\
\end{pmatrix} \quad \quad B_3^{-1} = \begin{pmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & -1 \\
0 & 0 & 1 & -1 \\
0 & 0 & 0 & 2 \\
\end{pmatrix}
\]

The result of applying this to our tableau is shown in table 4.4.

The optimal answer given our assumption of rapidly decreasing cost is \( (x_1, x_2) = (d_1, d_2 - 2d_1) \). This is point 1 in the constraint figure. We can check our answer. We know that having \( s_3 \) and \( s_4 \) as non-basis variable that equations E3 and E4 are active. The original inequalities turned into equations are,

\[
x_1 = d_1 \quad \quad E3 \\
2x_1 + x_2 = d_2 \quad \quad E4
\]
and their solution is exactly \((x_1, x_2) = (d_1, d_2 - 2d_1)\) as we found with the simplex method.

One remaining question is whether the simplex method will terminate at this point. Let’s take the last step (1) and find out,

\[
\begin{align*}
Z(s_3) &= 0 \\
&= -(1 \times 0 - 1 \times 0 - 1 \times c_1 + 2 \times c_2) = c_1 - 2 \times c_2 \\
Z(s_4) &= 0 \\
&= -(0 \times 0 + 1 \times 0 + 0 \times c_1 - 1 \times c_2) = c_2
\end{align*}
\]

We assume that \(c_2 < c_1/2\) so that choosing \(s_3\) does not offer a cost improvement and \(c_2 > 0\) implies that choosing \(s_4\) is also not a cost improvement. The simplex method therefore terminates.

4.3 The Effect on Random Variables

Now we consider the impact this analysis has on our goal of using random variables in place of fixed values when those values are uncertain. The most striking feature is that the matrix \(A\) does not contain any random variables. It forms the structure of the model being constructed based on supposed policy rules. We assume these rules to be fixed for the purposes of the model. Furthermore the shortfall variables denoted by the vector \(d\) are not expected to be random. They are given as policy variables. The only random inputs seem to be the cost \((c)\) vector and the quantity \((q)\) vector.

The cost and quantity vector are made up of a set of correlated random variables. The term *correlation* is being used in its general sense that allows for trivial correlations commonly referred to 100% correlated and 0% correlated or independent random variables.
The simplex method forms linear combinations of the $b$ vector random variables and compares them to find the *minimal* positive one. It also forms linear combinations of the cost ($c$) variables and seeks the *minimal* negative one. As we’ve seen, these are the only operations required.

The two operations required of our random variables are effectively identical and involve two steps. The first step is to eliminate the positive or negative random variables and the second step is to the the minimal one of those remaining random variables.
5.1 A Standard Approach

In Saltelli et al. (Saltelli [19]) a pair of simple examples are subjected to an exhaustive treatment of sensitivity, uncertainty and risk analysis. The first example of the pair may be represented by,

\[ Y = c_1 \times X_1 + c_2 \times X_2 \]

where \( c_1 \) and \( c_2 \) are constants and \( X_1 \) and \( X_2 \) are input factors that have uncertain value. The risk represented in the Saltelli example is of the return of the linear combination of two hedged portfolios whose returns are represented by \( X_1 \) and \( X_2 \).

In the second Saltelli example the \( c_1 \) and \( c_2 \) are admitted as input factors and therefore uncertain. We represent this second example as,

\[ Y = C_1 \times X_1 + C_2 \times X_2 \]

where capital case implies uncertain values.
According to Saltelli ([19]) uncertainty analysis assigns probability distributions to each input factor and seeks the probability distribution of each output factor. In the case of the Saltelli example pair there is only one output factor, namely $Y$.

In the PHoX system the main difference from Saltelli regarding the considered pair of examples is that to each input factor we associate a random variable and proceed to arithmetically combine random variables to produce output factors, in the Saltelli sense. Saltelli suggests that the probability distributions of each input factor be sampled some $N$ number of times and the output factor $Y$ be calculated $N$ times. This is the essence of the Monte Carlo method.

To perform a sensitivity analysis we ask, according to Saltelli, which input factor is most influential upon the output factor(s). Saltelli makes the point that merely calculating partial derivatives of $Y$ with respect to each input factor $X_i$ is misleading. In the first example where the coefficients $c_1$ and $c_2$ are constants, they are merely returned as in,

$$\frac{\partial Y}{\partial X_1} = c_1, \quad \frac{\partial Y}{\partial X_2} = c_2$$

If we suppose $c_1 > c_2$, but that the volatility of $X_1$ is less than that of $X_2$ we draw the inappropriate conclusion that the volatility of $Y$ (and therefore the riskiness of holding $Y$ as a portfolio) is most influenced by $X_1$, the low volatility/risk portfolio as opposed to $X_2$, the high volatility/risk portfolio. Certainly this would be true if $c_2 = 0$, or even if $c_2 << c_1$ depending on the relative volatility of $X_2$.

The question of which input factor is most influential on a given output factor bears some closer inquiry. Saltelli (Saltelli et al. [20]) offers a possible improvement,
where $\sigma_{X_i}$ is the standard deviation of $X_i$ and similarly $\sigma_Y$ is the standard deviation of output factor $Y$. This normalized derivative is, according to Saltelli, recommended for sensitivity analysis by a guideline of the Intergovernmental Panel for Climate Change (IPCC)(1999, 2000).

5.2 Algebraic Correlation

A model within the PHoX modeling system is composed of several kinds of software objects possibly including arithmetic expressions such as,

$$X = A + \frac{A^2}{B}$$

where $A$ and $B$ are random variables. Established theory dictates that if $A$ and $B$ are independent random variable then we form a two dimensional joint probability distribution for them and perform line integrals along level curves so that,

$$P(X = x) = \int_{x = a + \frac{A^2}{B}} dP(A = a) \ dP(B = b)$$

The problem is that the PHoX modeling system provides a modeling language in which a user can implement a model, it does not know what algebraic expressions will arise within that user-defined model. The PHoX system must handle arbitrary arithmetic expressions
involving random variables or at least a well-defined subset.

Since the PHoX system is designed for business-class models rather than scientific modeling some simplifying assumptions will be made. The first such assumption is that arithmetic expressions that arise within a PHoX model are relatively rudimentary, of the kind encountered in a business textbook (Brigham [7]) or introductory finance textbook (Brealey/Myers [6]). To make these assumptions more concrete we refer to the existing Carbon Emission Reduction Market (AB32) model implemented within the PHoX system.

The most complex arithmetic expression in the Carbon Emissions Reduction Market model described elsewhere in this paper is the levelized cost of a carbon emission reduction project. A levelized cost is typically the quotient of two net present value calculation of anticipated cash flows, in the case of the numerator, and the net present value of units of reduction achieved over time, in the case of the denominator. Each net present value calculation is made with some model-defined discount factor.

To introduce uncertainty through random variables into the calculation of a levelized cost we recognize that there is a random component that is common to costs and reductions achieved and a random component that is separate for each. If $A$, $B$ and $C$ are independent random variables and $f$, $g$, $h$ and $k$ are functions we may represent a levelized cost as,

$$
\text{Cost} = \frac{f(A) + h(B)}{g(A) + k(C)}
$$

Our goal is to represent arithmetic combinations of random variables that are, in some sense, at least as complicated as for the case of levelized costs in a computationally tractable manner. To reach this goal we will enumerate a range of cases we need the PHoX system to handle and describe issues involved in implementing appropriate computational algorithms.
5.2.1 Notation and Definitions

Let $A$, $B$, etc. are independent random variables and that ‘*’ represents any of the binary arithmetic operations ($+, -, \times , \div$). When two random variables are related by a subscript as in $A_1$ and $A_2$ we assume that each is related to the same parent random variable $A$ by a rational polynomial function. For example we may have $A_1 = f(A)$ and $A_2 = g(A)$ where $f$ and $g$ are rational polynomial functions of $A$. We will explain this choice of function class below. We refer to variables related by subscript as directly related. We refer to random variables directly related to exactly one first class random variable as basic.

5.2.2 Directly Related Random Variables

Claim: The set of rational polynomial functions $\mathbb{Q}$ of a single variables, $A$, is closed with respect to the binary arithmetic operations ($+,-,\times,\div$).

Proof of Claim: Let $f,g \in \mathbb{Q}$ where,

$$f(A) = \frac{p(A)}{q(A)}$$
$$g(A) = \frac{r(A)}{s(A)}$$

and $p(A), q(A), r(A), s(A)$ polynomials in $A$. Allowing that $c$ is an arbitrary constant and that juxtaposition implies multiplication, we need to check each operation,
Since $p(A) \pm q(A)$ is a polynomial in $A$ and $p(A) \times q(A)$ is also a polynomial in $A$ we reach our desired conclusion.

The reason this restriction of only allowing basic arithmetic operations on an input variable, $A$, in our model is valid is because the kind of models we that interest us tend to have physical dimensional inputs. That is, the input data to the models we consider represent dimensions such as meters or number of items per year, etc. The software implementation of models we have considered results in a dichotomy of foreground and background or behind-the-schemes coding. In background coding any operation or function is allowed in order to perform some specific task such as an optimization or other specialized function. In foreground coding we find that more formal steps must be taken. This tends to disallows the application of functions such as $\exp()$ or $\log()$ or $\sin()$, etc. that require non-dimensional inputs.

We will see that the PHoX modeling system is capable of handling expressions that do not meet our definition of business-class, such as,
\[ A + (B \div (A + C)) \]

There is no presumption that random variables in the PHoX modeling system have associated physical unit. We seek to balance computational tractability, *representational fidelity* and range of acceptable models. We will next discuss the numerical representation of random variables in the PHoX modeling system. As with any numerical representation or numerical method there remains the question of whether the result accurately represents the theoretical object. This is what we mean by representational fidelity.

5.2.3 Categorizing Algebraically Correlated Cases

To investigate the algebraic expressions we may encounter in a PHoX-based model we adopt the following notation,

\[
\text{Let } \mathbb{A} = \{(n_1, n_2, ..., n_m)\}, \forall m, n_1 \leq n_2 \leq ... \leq n_m \in \mathbb{N}
\]

where \( \mathbb{N} = \{1, 2, ...\} \) and \( \mathbb{A} \) represents the set of expressions containing \( n_1 \) versions of a random variable \( A \), \( n_2 \) versions of another random variable \( B \), etc. where \( A, B, \) etc. are mutually independent. The \( n_1 + n_2 + ... + n_m = N \) random variables may be combined using the basic arithmetic operator (+, −, ×, ÷) in any order. By convention we assume (without loss of generality) that \( n_1 \leq n_2 \leq ... \leq n_m \).

We will systematically enumerate and address some elements of \( \mathbb{A} \). For example, notice that \( \mathbb{A}_{2,1} \in \mathbb{A} \) and that the expression \( A_1 + A_2 \times B_1 \subset \mathbb{A}_{2,1} \). Since we may have that \( A_1 = 2A, A_2 = A^3 \) and \( B_1 = 1/B \) that the expression \( (3A + A^3/B) \in (A_1 + A_2 \times B_1) \)
so that $3A + A^3/B \in A_{2,1}$. Since the cases we will address will not surpass single digit indices we will, in an abuse of notation, equate,

$$A_{2,1} = A_{21}$$

The address the issue of computability of elements of $A$. We shall see below that an essential feature of computability is describing the joint distribution of the random variables in any given expression.

We begin with the case of $A_1$. Since this represents a single random variable there is nothing to compute. Employing the random variable indices as a way to differentiate two versions of a random variable as in $A_1 = f_1(A)$ and $A_2 = f_2(A)$ where $f_1$ and $f_2$ are any two rational polynomials of $A$ we see that $A_1$ contains only the generic case of $A_1$. The case of $A_n$ represents the case $A_1 \ast A_2 \ast \ldots \ast A_n$ which may be denoted by $A_{n+1}$ because it is directly related to $A$ since $f_1(A) \ast f_2(A) \ast \ldots \ast f_n(A)$ is a rational polynomial in $A$ for any combination of operators $(+, -, \times, \div)$.

The $A_{11}$ set is enumerated as \{ $A_1 + B_1, A_1 - B_1, A_1 \times B_1, A_1 \div B_1$ \}. Since $A$ and $B$ are assumed to be mutually independent, $A_1$ and $B_1$ are also mutually independent. Before moving on we will describe how each case is handled for our piecewise uniform random variables.

5.2.3.1 Example: $X = A^2 \div (A - 2)$

It is illuminating to consider how to compute the random variable formed by the quotient of two directly correlated random variables. Let,
Figure 5.1: Quotient of Two Directly Related Random Variables

\[ X = \frac{A^2}{A - 2} \]

which could be rewritten as,

\[ A_3 = \frac{A_1}{A_2} \]

where \( A_1 = A^2 \), \( A_2 = A - 2 \) and \( A_3 = X \). The expression \( A_1 \div A_2 \) seems to be an element of \( A_2 \), but collapses to \( A_1 \). Nonetheless, the manner in which we compute \( X \) will highlight an issue that is central to computing correlated random variable expressions, namely, joint probability distributions embedded into larger spaces. In this case the joint probability distribution of \( A_1 \) and \( A_2 \) is a one dimensional embedding into the \( (A_1, A_2) \)-space. The figure 5.1 depicts the situation,

where,
\[A = \{(-2, -1, 0, 1, 2, 4), (p_1, p_2, p_3, p_4, p_5)\}\]

If we assume that \(A\) is partitioned as \((-2, -1, 0, 1, 2, 4)\) the corresponding points in \((A_1, A_2)\) are \(((4, -4, (1, -3), (0, -2), (1, -1), (4, 0), (16, 2))\) with corresponding \(x\)-values \((-1, -1/3, 0, -1, \pm \infty, 8)\). The interval endpoints are marked with points in figure 5.1. We refer to these representations of \(A_1\) and \(A_2\) as being synchronized with \(A\).

Now suppose we choose the \(X\)-partition as,

\[X = \{(-\infty, -1, -1/3, 0, 8, \infty), (q_1, q_2, q_3, q_4, q_5)\}\]

Proceeding by inspection of figure 5.1 we estimate the probability values of \(X\) to be,

\[P(-\infty < X < -1) = p_4\]
\[P(-1 < X < -1/3) = p_1 + \frac{1}{2}p_3\]
\[P(-1/3 < X < 0) = p_2 + \frac{1}{2}p_3\]
\[P(0 < X < 8) = 0\]
\[p(8 < X < \infty) = p_5\]

The significance of this example is that it informs us how to proceed in what follows. We notice that the partition of \(A\) is synchronized with the partitions of \(A_1\) and \(A_2\) which in turn is synchronized to their quotient as shown in table 5.1.
Table 5.1: Synchronized Partitions Table

<table>
<thead>
<tr>
<th>Expression</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$A^2$</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$A-2$</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$X$</td>
<td>$A_1/A_2$</td>
<td>-1</td>
<td>-1/3</td>
<td>0</td>
<td>-1</td>
<td>$\pm\infty$</td>
</tr>
</tbody>
</table>

Figure 5.2: Folding of Quotient of Two Directly Related Random Variables

To find the probability values $(q_1, q_2, q_3, q_4)$ for $X$ we take each interval of $X$ for which we have probability values $(p_1, ..., p_5)$ and allocated them to the chosen partition for $X$. In figure 5.2 the $X_3$ interval, $(-1, 0)$ with associated probability $p_3$ is most interesting since it must be split over two two $X$ partition intervals, $(-1, -1/3)$ and $(-1/3, 0)$.

The $X_1$ and $X_2$ intervals happen to line up precisely with respective $X$-partition intervals. The $X_4$ and $X_5$ intervals have infinite endpoints. There is an ambiguity of which polarity of infinity should be used for these two intervals. We have chosen $(-\infty, -1)$ for $X_4$ and $(8, +\infty)$ for $X_5$ because it is consistent with the graph, but a specific algorithm will have to wait until we have established the specific programmatic structures to represent and perform these calculations numerically.

Assuming uniform probability distributions within intervals we conclude for interval $X_3 = \{(-1, 0), (p_3)\}$ in our example that $X$ partitions $(-1, -1/3)$ and $(-1/3, 0)$ should receive $2/3p_3$ and $1/3p_3$ respectively. Using this numerical, rather than inspection, method we
find that our resultant $X$ is,

$$X = \{(-\infty, -1, -\frac{1}{3}, 0, 8, +\infty), (p_4, p_1 + \frac{2}{3}p_3, p_2 + \frac{1}{3}p_3, 0, p_5)\}$$

When a random variable is presented with a first list of distinct values in ascending order and a second list of probability values presumed interstitial to the first list we say that random variable is in *proper form*. This as opposed to the synchronized form we used above.

We have solved a one dimensional example in a manner consistent with our approach for two dimensions and higher. Rather than partitioning a one dimensional interval such as $X_3$ is our example using a given $X$-partition we will, in future examples, partition two dimensional and higher objects.

5.2.4 Enumerating Algebraically Correlated Cases

Introducing more random variables and versions of random variables into an expression requires some rules and key observations in order to avoid a combinatorial explosion of cases.

Consider the expression $(A_1 + B_1)/A_2$. Since we have demonstrated that $1/A_2$ may be replaced by its reciprocal, renamed $A_2$ the expression is computationally equivalent to $(A_1 + B_1) \times A_2$. Since multiplication is commutative, the original expression is equivalent to $A_2 \times (A_1 + B_1)$. We note, in contrast, that the expression $A_2 \div (A_1 + B_1)$ is not subject to the same analysis. We do not claim that $1/(A_1 + B_1)$ is computationally equivalent to $(A_1 + B_1)$.

As a further contrasting example note that $A_2 \div (A_1 \times B_1)$ is the same as $A_2 \times ((1/A_1) \times (1/B_1))$ and since we may replace the reciprocal of a basic random variable with a basic
random variable re-using the same name the expression is computationally equivalent to
\( A_2 \times (A_1 \times B_1) \). By associativity, this expression is equal to \((A_2 \times A_1) \times B_1\) and by our combination rule we may replace \((A_2 \times A_1)\) with a single directly related random variable and are allowed to recycle one of the names to \( A_1 \times B_1 \). This shows that an apparent \( A_{21} \) case may collapse to a \( A_{11} \) case.

5.2.4.1 Computational Equivalence Rules

Expanding the above discussion into a set of rules for computational equivalence,

1. \(-A_1 \sim A_1\)

2. \(1/A_1 \sim A_1\)

3. \((A_1 \ast A_2) \sim A_1\) for any \( \ast = (+, -, \times, \div) \)

4. \((A_1 \div B_1) \sim (A_1 \times B_1)\)

5. \((B_1 \ast A_1) \sim (A_1 \ast B_1)\) for any \( \ast = (+, -, \times, \div) \)

6. \((A_1 \ast B_1) \ast A_2 \sim A_2 \ast (A_1 \ast B_1)\) unless \( A_2 \div (A_1 + B_1) \)

7. \((A_1 + B_1) + A_2 \sim A_1 + B_1\)

8. \((A_1 \times B_1) \times A_2 \sim A_1 \times B_1\)

9. \((A_1 + B_1) \times A_2 \sim A_1 \times B_1 + A_2\)

The last three rules are redundant since they recognize associativity and distributivity, but are enumerated for convenience.

Notice that since expressions with instances of the generic operator “\(\ast\)” represent sets of expressions we have,
\[(A_1 \ast B_1) \ast A_2 \subset A_2 \ast (A_1 \ast B_1)\]

because \((A_1 + B_1) \div A_2 \sim (A_1 + B_1) \times A_2\) while \(A_2 \div (A_1 + B_1)\) is a separate expression.

5.2.4.2 Case Enumeration Procedure

To address cases beyond \(A_{11}\) we will use the following procedural steps,

1. Present the variables names involved. E.g. \(A_{21} = A_1 \ast A_2 \ast B_1\)

2. Show all distinct expressions by permutation of variable names.

3. Show all distinct expressions through association of pairs of variables.

4. Show all distinct expressions where the generic operation “∗” is replaced with supported arithmetic operations.

Notice that in the cases of \(A_{22}\) there are two version of \(A\) and two version of \(B\) present namely \(A_1 \ast A_2 \ast B_1 \ast B_2\) so the permutation \(B_2 \ast B_1 \ast A_1 \ast A_2\) is the same as \(A_1 \ast A_2 \ast B_1 \ast B_2\) because the names \(A\) and \(B\) may be swapped since they appear in equal number and the subscripts can be swapped, that is, \(A_2 \ast A_1 \sim A_1 \ast A_2\).

5.2.5 Enumerating the \(A_{21}\) Case

Following our case enumeration steps,

1. \(A_{21} = A_1 \ast A_2 \ast B_1\)

2. There are \(3! / 2! = 3\) permutations of \(A_{21}\),

\[
A_1 \ast A_2 \ast B_1 \quad A_1 \ast B_1 \ast A_2 \quad B_1 \ast A_1 \ast A_2
\]
3. Applying associations our expressions become,

\[ A_1 \ast (A_2 \ast B_1) \quad (A_1 \ast B_1) \ast A_2 \quad A_1 \ast (B_1 \ast A_2) \quad (B_1 \ast A_1) \ast A_2 \]

Applying the equivalence and renaming rules we see that,

\[ A_1 \ast (A_2 \ast B_1) \sim A_2 \ast (A_1 \ast B_1) \]
\[ (A_1 \ast B_1) \ast A_2 \subset A_2 \ast (A_1 \ast B_1) \]
\[ A_1 \ast (B_1 \ast A_2) \sim A_2 \ast (A_1 \ast B_1) \]
\[ (B_1 \ast A_1) \ast A_2 \subset A_2 \ast (A_1 \ast B_1) \]

Leaving only the generic case: \( A_2 \ast (A_1 \ast B_1) \).

4. Introducing specific arithmetic operations we realize by enumeration rule 5 that there are \( 3 \times 2 = 6 \) cases which we represent symbolically as \((+, \times, \div) \otimes (+, \times)\). In the following enumeration we indicate which cases we will retain as distinct and particular to \( A_{21} \) and which are relegate to case \( A_{11} \) and so have already been addressed,

\[ A_2 + (A_1 + B_1) \quad \in A_{11} \text{ by rule 7} \]
\[ A_2 \times (A_1 + B_1) \sim A_1 + (A_2 \times B_1) \in A_{21} \text{ by rule 9} \]
\[ A_2 \div (A_1 + B_1) \sim A_1 \div (A_2 + B_1) \in A_{21} \]
\[ A_2 + (A_1 \times B_1) \sim A_1 + (A_1 \times B_1) \]
\[ A_2 \times (A_1 \times B_1) \quad \in A_{11} \text{ by rule 8} \]
\[ A_2 \div (A_1 \times B_1) \quad \in A_{11} \text{ by rules 5 and 8} \]

Thus we find two irreducible cases particular to \( A_{21} \) and write,
\[ A_{21} = \{ A_1 + (A_2 \times B_1), A_2 \div (A_1 + B_1) \} \]

These two cases represent something new. They are each result in a random variable that is arithmetically correlated to two independent random variables. We will describe below how to numerically compute these resultant random variables.

5.2.5.1 An Example: \( X = A_1 + (A_2 \times B_1) \)

In the computation of correlated random variables section we will present a software implementation for making such calculations in a manner that balances computational performance and result fidelity. Our goal with this example is to expose some of the issues that our eventual software implementation must address.

We first recognize that the expression \( A_1 + (A_2 \times B_1) \) represents a class of algebraic expressions depending on the particular choice for \( A_1, A_2 \) and \( B_1 \). Some such choices can be handled by techniques we have presented so far. If, for example, we choose \( Y = A + A \times B \) we see that we can factor out the \( A \) and form \( Y = A \times (1 + B) \) which may be rewritten as \( Y = A \times B_1 \) and we see the expression reduces so a previous case.

We now present an example in the class of \( A_1 + (A_2 \times B_1) \) which have not yet described how to handle,
\[ X = (A^2 + A) + ((A + 1) \times (1/B + 1)) \]

Let \( A_1 = A^2 + A \)

Let \( A_2 = A + 1 \)

Let \( B_1 = 1/B + 1 \)

\[ X = A_1 + A_2 \times B_1 \]

We recognize this expression for \( X \) as a proper element of \( A_{21} \).

To approach this example problem we recall that \( A \) and \( B \) are each partitioned into intervals with associated probability values since each is a piecewise uniform distribution. We presume that we have established a partition for \( X \) and that our task in computing \( X \) is to assign probability values to each of its partition intervals. This task is broken down into finding the probability contribution each partition interval of \( A \) and \( B \) makes to the partition intervals of \( X \). In figure 5.3 we show the rectangular joint probability distribution jointly partitioned by \( A \) and \( B \). The \( A_1, A_2 \) and \( B_1 \) axes are referred to as ancillary since the share partition endpoints with the \( A \) and \( B \) axes, but are not assumed monotonic. For example, a value such as zero may appear more than once on an ancillary axis.

We refer to the rectangle \((a_i, a_{i+1}) \times (b_j, b_{j+1})\) as the \( i,j \)-rectangle and if \( P((a_i, a_{i+1})) = p_i \) and \( P((b_j, b_{j+1})) = q_j \) then \( P(i,j - \text{rectangle}) = p_i \times q_j \), the joint probability.

To find the probabilities allocated to each partition element of the resultant \( X \) we use the partition of \( X \) to partition each \( i,j \)-rectangle and allocate the resulting fractions of the joint probability associated with the \( i,j \)-rectangle to the respective partition elements of \( X \).

For clarity of this example we suppose that \( A \) and \( B \) are uniformly distributed and that,
Figure 5.3: Partition of Joint Rectangle

$$A \sim U(1, 2)$$
$$B \sim U(\frac{1}{2}, 1)$$

Since $A_1$ and $A_2$ are functions of $A$ and $B_1$ is a function of $B$ their synchronous forms are,

$$A_1 = (2, 6)$$
$$A_2 = (2, 3)$$
$$B_1 = (3, 2)$$

Just as in the previous example we see that $A_1$ and $A_2$ form a parametric curve with respect to $A$ shown in figure 5.4.

To find the joint probability density we find the direct product of the $(A_1, A_2)$ graph
Figure 5.4: Joint Distribution of Two Directly Related Random Variables

and \( B_1 \) with result shown in figure 5.5.

A given partition of \( X \) implies a partition of the \( (A_1, A_2, B_1) \)-space. The joint probability in this example is distributed over the section of cylindrical parabola shown in figure 5.5. The partitions of \( X \) will appear as non-intersecting level surfaces.

We have shown above that our assumption of uniform probability distribution over each partition interval of each first class random variable, \( A \) and \( B \) in this case, does not necessarily result in a uniform distribution of probability over the joint surface shown above. This frees us to make the assumption that transforming the \( X \)-partition surfaces and joint probability surface to \( (A, B) \)-space will result in similar partitioning of the joint probability into the \( X \)-partition elements.

In figure 5.6 we show the joint probability area of \( A_1, A_2 \) and \( B_1 \) with respect to the first class \( A \) and \( B \). This area is always rectangular because \( A \) and \( B \) are disjoint. We include the ancillary axes \( A_1, A_2 \) and \( B_1 \) as before. We further indicate the \( x \)-values for the vertices and the \( X \) level curves that pass through the vertices. If we choose \((6, 8, 12, 15)\) as the partition for \( X \) is shown in figure 5.6.

To find the \( x \) level curves we solve \( X = A_1 + A_2 \times B_1 \) for \( B(A|x) \), that is, \( B \) as a function of \( A \) for a fixed value of \( x \). In this example we find,
Figure 5.5: Joint Distribution of an $A_{21}$ Case

Figure 5.6: Partition of Joint Distribution of an $A_{21}$ Case
\[ x = A_1 + A_2 \times B_1 \]
\[ = A^2 + A + (A + 1) \times \left( \frac{1}{B} + 1 \right) \]
\[ B(A|x) = \frac{A + 1}{x - (A^2 + A) - (A + 1)} \]
\[ = \frac{A + 1}{x - (A + 1)^2} \]

The fraction of the total area of the rectangle between any two adjacent \(x\)-curves is proportional to the amount of probability allocated to the corresponding \(X\)-partition element.

To recap, we are assuming that the joint probability of \(A\) and \(B\) in this example is distributed uniformly over this rectangle. Since \(A\) and \(B\) happen to be uniform random variables this is the only joint uniformly distributed rectangle in the example and carries a probability of one. Visually we estimate \(X\) to be,

\[ X = \{(6, 8, 12, 15), (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})\} \]

To compute the probability values of \(X\) we either must integrate the area between \(x\) curves (within \(X\) partition intervals) or make further assumptions to reduce the problem to computations of triangles and trapezoids. We defer that discussion to the computation section. It turns out we will find it advantageous to make a different choice.

We realize from this example that some expressions of the form \(A_1 + A_2 \times B_1\) are reducible to the previously discussed independent case and some, like the example addressed here, are not reducible. By focusing on the \((A, B)\)-space partition for \(A_{nm}\) cases we will find a unified computation approach.

153
5.2.5.2 An Example: \( X = A_1 + (A_2 \times B_1) \), Again

To better understand how level curves affect the computation of correlated random variables we revisit the previous example. Let,

\[
X = A_1 + (A_2 \times B_1)
\]

Rather than choosing specific functions \( f \) and \( g \) so that \( A_1 = f(A) \) and \( A_2 = g(A) \) for the common random variable \( A \), we instead assume \( A_2 = m_i \times A_1 + b_i \), where \( m_i \) and \( b_i \) are constants that depend on the partition interval of \( A_1 \). This allows us to better understand the form \( A_1 + (A_2 + B_1) \) in general terms.

We choose the \( i^{th} \) partition of \( A_1 \) and \( j^{th} \) partition of \( B_1 \) and suppose as before,

\[
A_1 = (2, 6) \\
A_2 = (2, 3) \\
B_1 = (3, 2)
\]

Assuming the affine relation between \( A_1 \) and \( A_2 \) for this partition interval choice we find,

\[
A_2 = \frac{3 - 2}{6 - 2} \times (A_1 - 2) + 2 = \frac{1}{4} A_1 + \frac{3}{2}
\]
To find the $x$-level curves we find $X$ to be $x$ and solve for $B_1(A_1|x)$ in,

$$x = A_1 + (A_2 \times B_1)$$

$$B_1 = \frac{x - A_1}{A_2}$$

$$B_1(A_1|x) = \frac{x - A_1}{\frac{1}{4}A_1 + \frac{3}{2}}$$

We again suppose the partition for $X$ to be $(6, 8, 12, 15)$ and plot the level curves with the $i, j$-rectangle as shown in figure 5.7.

where we have $A_1$ and $B_1$ as axes and $A_2$ as the sole ancillary axis. We have kept the orientation of $B_1$ the same as the previous example for ease of comparison. We find that the $x$-level curves in this example compare favorably to the previous example even though this example made no assumptions about the specific relationship between $A_1$ and $A$ or between $A_2$ and $A$. All that was assumed here is that $A_1$ and $A_2$ are directly related and supposed the relationship to be affine.
This technique of assuming a piecewise affine relationship between versions of a random variable in an expression will be adopted for the remainder of this section.

5.2.5.3 Computation Technique for $A_1 \div (A_2 + B_1)$

With so much groundwork laid in the previous section we may proceed more directly. We again form the expression,

\[
x = A_1 \div (A_2 + B_1)
\]

where \(x\) is a constant representing the value of a level curve. We assume that working copies of \(A\) and \(B\) are appropriately partitioned and that we are describing how to compute a specific pair, \((i, j)\) of partition intervals, one from \(A\) and the other from \(B\). That is, \(A^i = [a_1, a_2]\) and \(B^j = [b_1, b_2]\). For this choice of partition interval pair we further assume an affine relationship between \(A_1\) and \(A_2\) as,

\[
A_2(A_1) = m * (A_1 - a_{11}) + a_{21}
\]

where we have dropped the superscripts from \(A_1^i\) and \(A_2^i\) for clarity and assumed as before the relevant intervals,

\[
A_1^i = U(a_{11}, a_{12}) \quad A_2^i = U(a_{21}, a_{22}) \quad B_1^j = U(b_{11}, b_{12})
\]

and that \(m = \frac{a_{22} - a_{21}}{a_{12} - a_{11}}\). Solving for \(B_1\) in terms of \(A_1\) and \(x\),

156
\[ B_1(A_1|x) = \frac{A_1}{x} - A_2 \]

\[ = \frac{A_1}{x} - m(A_1 - a_{11}) - a_{21} \]

\[ = (\frac{1}{x} - m)A_1 + (ma_{11} - a_{21}) \]

Notice that the \( B_1 \)-axis intercept occurs at the same distance from the origin as the \( A_1 | A_2 = 0 \) point on the \( A_1 \)-axis. Thus the effect of introducing an extra version of \( A \), namely \( A_2 \) into the calculation of \( A_1 \div B_1 \) is an offsetting from the origin of the focal point of the level curves.

Using the values from the previous example our new result is shown in figure 5.8.

where we have again chosen a partition of \( X = A_1 \div (A_2 + B_1) \) that happens to pass through the vertices of the \((i, j)\) partition cell for illustration purposes. Unlike the previous example where the level curves are actually curved it is very computationally straightforward to calculate the areas enclosed by the \( x \)-partition intervals for each \((i, j)\) partition cell. This will be explored further after the \( A_{211} \) cases is described.

5.2.5.4 Further Considerations of the \( A_{21} \) Case

The practical issues involved in computing \((2, 1)\) cases of correlated random variables include maintaining a construction history for each computed variable. This means that if an expression for some variable, call it \( X \), is formed during model creation the supporting system must remember which input random variables are involved in forming \( X \). If \( X \) depends solely on input random variable \( A \) then we may internally refer to it as \( A_1 \) regardless of the name given it by the model developer. This amounts to maintaining a parse tree which is a standard object in computer science.
Figure 5.8: Example of an $A_{21}$ Case
To form a new *model variable* (any variable created within the model) the PHoX system goes through an exercise similar to the one we used to reduce our \((2, 1)\) cases to one of four special cases after first determining that we are indeed facing a \((2, 1)\) case.

Running a model within the PHoX system involves several *passes*. The first pass in a *compiler* pass where all the model variables are formed and analyzed as to their particular case and which compute strategy will be employed. This includes identifying poles and peaks for each rational polynomial. Identifying the poles and peaks can be performed internally to the PHoX system or off-loaded to a symbolic solver. This compiler pass only needs to be performed when the model code changes, not for changes to input data. Compiling is an exercise in structural analysis of model code. This has favorable implications for model performance.

5.2.6 Enumerating the \(A_{31}\) Case

Following our case enumeration steps,

1. \(A_{31} = A_1 \ast A_2 \ast A_3 \ast B_1\)

2. There are \(4!/3! = 4\) permutations of \(A_{31}\),

\[
A_1 \ast A_2 \ast A_3 \ast B_1, \quad A_1 \ast A_2 \ast B_1 \ast A_3, \quad A_1 \ast B_1 \ast A_2 \ast A_3, \quad B_1 \ast A_1 \ast A_2 \ast A_3
\]
3. Applying associations our expressions become,

\[ A_1 \ast (A_2 \ast (A_3 \ast B)), \]

\[ (A_1 \ast (A_2 \ast B)) \ast A_3, \quad A_1 \ast (A_2 \ast (B_1 \ast A_3)), \]

\[ (A_1 \ast (A_2 \ast B_1) \ast A_3, \quad A_1 \ast ((A_2 \ast B_1) \ast A_3), \]

\[ ((A_1 \ast B_1) \ast A_2) \ast A_3, \quad (A_1 \ast (B_1 \ast A_2)) \ast A_3, \quad A_1 \ast ((B_1 \ast A_2) \ast A_3), \]

\[ ((B_1 \ast A_1) \ast A_2) \ast A_3 \]

Applying the equivalence and renaming rules leaves us with a single case,

\[ A_1 \ast (A_2 \ast (A_3 \ast B)) \]

We may write this as,

\[ A_1 \otimes A_{21} \]

where we have already reduced the \((A_2 \ast (A_3 \ast B))\) expression to,

\[ A_3 \times B_1 + A_2 \quad A_2 \div (A_3 + B_1) \]

so that we must expand,

\[ A_1 \ast (A_3 \times B_1 + A_2) \quad A_1 \ast (A_2 \div (A_3 + B_1)) \]
4. Expanding the cases we find the following $3 + 3 = 6$ cases,

\[
\begin{align*}
A_1 + (A_3 \times B_1 + A_2) & \sim A_1 + A_2 \times B_2 \text{ by rule 7} \\
A_1 \times (A_3 \times B_1 + A_2) & \sim A_1 + A_2 \times B_2 \text{ by rule 3} \\
A_1 \div (A_3 \times B_1 + A_2) & \sim A_1 + A_2 \times B_2 \text{ by rule 7} \\
A_1 + (A_2 \div (A_3 + B_1)) & \in A_{31} \\
A_1 \times (A_2 \div (A_3 + B_1)) & \sim A_2 \div (A_3 + B_1) \text{ by rule 3} \\
A_1 \div (A_2 \div (A_3 + B_1)) & \sim A_1 + A_2 \times B_2 \text{ by rule 7}
\end{align*}
\]

We see that we have one irreducible case peculiar to $A_{31}$ and write,

\[
A_{31} = A_1 + (A_2 \div (A_3 + B_1))
\]

5.2.6.1 Computation Technique for $A_1 + (A_2 \div (A_3 + B_1))$

We follow the same pattern as before but assume, for the $(i,j)^{th}$ partition interval of $A$ and $B$ that $A_2$ and $A_3$ are both affine transforms of $A_1$ and write,

\[
\begin{align*}
A_2(A_1) &= m(A_1 - a_{11}) + a_{21} \\
A_3(A_1) &= n(A_1 - a_{11}) + a_{31}
\end{align*}
\]

where,
\[ A_1^i = U(a_{11}, a_{12}) \quad A_2^i = U(a_{21}, a_{22}) \quad A_3^i = U(a_{31}, a_{32}) \]

\[ B_1^j = U(b_{11}, b_{12}) \]

\[ m = \frac{a_{22} - a_{21}}{a_{12} - a_{11}} \quad n = \frac{a_{32} - a_{31}}{a_{12} - a_{11}} \]

Assuming \( X = A_1 + (A_2 \div (A_3 + B_1)) \) and fixing \( X = x \) to find the level curves of \( B_1(A_1|x) \) we have,

\[
x = A_1 + A_2 \div (A_3 + B_1)
\]

\[
(x - A_1) \times (A_3 + B_1) = A_2
\]

\[
B_1(A_1|x) = A_2 \div (x - A_1) - A_3
\]

\[
= \frac{m(A_1 - a_{11}) + a_{21}}{x - A_1} - n(A_1 - a_{11}) - a_{31}
\]

To illustrate with our familiar example where,

\[
A_1 = U(1, 2) \quad A_2 = U(5, 7) \quad A_3 = U(8, 9)
\]

we see level curves asymptotic to their own \( x \) value on the \( A_1 \)-axis shown in figure 5.9.

We notice for future reference that the level curves are almost straight lines in this example chosen. While this situation may not persist for all examples it suggest that a piecewise linear approximation to the level curves may be a computational compromise over time-consuming calculation for each \((i, j)\) interval and each \(X\)-partition interval.
Figure 5.9: Example of an $A_{31}$ Case
5.2.7 Enumerating the $A_{41}$ Case and Beyond

Now that we have detailed the previous cases and come to $A_{41}$ we realize that a full enumeration and association of $A_1 * A_2 * A_3 * A_4 * B_1$ is not necessary. There is only one version of $B$ present. It will be associated with a version of $A$ which we assume without loss of generality to be $A_4$ operating on the left giving $(A_4 * B_1)$. To this expression we associate one of the remaining versions of $A$ which we assume to be $A_3$ again operating on the left since this is the most general case, $(A_3 * (A_4 * B_1))$. Proceeding similarly for the remaining versions of $A$ we find the one generic expression to be,

$$A_1 * (A_2 * (A_3 * (A_4 * B_1)))$$

Furthermore we recognize this expression as,

$$A_1 \otimes A_{31}$$

and since $A_{31}$ has only one case peculiar to it we have,

$$A_1 * (A_2 + A_3 \div (A_4 + B_1))$$

which we enumerate as,
\[ A_1 + (A_2 + A_3 \div (A_4 + B_1)) \sim A_1 + A_2 \div (A_3 + B_1) \in \mathbb{A}_{31} \]

\[ A_1 \times (A_2 + A_3 \div (A_4 + B_1)) \sim A_1 + A_2 \div (A_3 + B_1) \in \mathbb{A}_{31} \]

\[ A_1 \div (A_2 + A_3 \div (A_4 + B_1)) \in \mathbb{A}_{41} \]

Thus the only irreducible case particular to \(\mathbb{A}_{41}\) is,

\[ \mathbb{A}_{41} = \left( \frac{A_1}{A_2 + \frac{A_3}{A_4 + B_1}} \right) \]

If we relabeled the versions of \(A\) and added a new version of \(A\) on the left we would have,

\[ \mathbb{A}_{51} = (A_1 + \frac{A_2}{A_3 + \frac{A_4}{A_5 + B_1}}) \]

Notice that if we multiply on the left by \(A_1\) instead of add, the \(A_1\) would combine with \(A_2\) in the numerator and reduce the case to \(\mathbb{A}_{41}\). If we divide on the left by \(A_1\) instead of add we would have,

\[ \frac{A_1}{A_2 + \frac{A_3}{A_4 + A_5 + B_1}} = \frac{A_1}{A_2} (A_3 + \frac{A_4}{A_5 + B_1}) \]

\[ \sim A_1 + \frac{A_2}{A_3 + B_1} \]

which is an element of \(\mathbb{A}_{31}\) and not particular to \(\mathbb{A}_{51}\). Thus our statement of \(\mathbb{A}_{51}\) having
a single irreducible element particular to it stands.

The emerging pattern suggests that $A_n$ for $n > 2 \in \mathbb{N}$ each consist of a single irreducible expression particular to it in the form of a continued fraction.

5.2.8 Enumerating the $A_{111}$ Case

Before we enumerate the $A_{211}$ cases we must consider the $A_{111}$ case. It is represented by,

$$A_{111} = A_1 \ast B_1 \ast C_1$$

Since we have equal numbers of each version of random variable, that is one each, there is only one permutation. Introducing associations,

$$(A_1 \ast B_1) \ast C_1 \quad A_1 \ast (B_1 \ast C_1)$$

reveals a property that will be important when describing the $A_{211}$ case. We notice that $(A_1 \ast B_1) \in A_{11}$ and that $(A_1 \ast B_1)$ is independent of $C_1$. Letting $AB_{11} := (A_1 \ast B_1)$ for some operation “∗” we see that $AB_{11} \ast C_1 \in A_{11}$. Similarly $A_1 \ast BC_{11} \in A_{11}$ meaning that there are no elements particular to $A_{111}$. In that sense $A_{111}$ is empty.

The point to understand from the $A_{111}$ case is that while it is possible to form a three dimensional joint distribution for $A_1$, $B_1$ and $C_1$, it is unnecessary. We say that expressions of the form $(A_1 \ast B_1) \ast C_1$ or $A_1 \ast (B_1 \ast C_1)$ are sequential, that is, they may be computed sequentially.
5.2.9 Enumerating the $A_{211}$ Case

Following the enumeration steps we developed for two dimensional arithmetically correlated random variables we have,

1. The generic case for $A_{211}$ is $A_1 * A_2 * B_1 * C_1$.

2. There are $4!/2!/2! = 6$ permutations since $B_1$ is interchangeable with $C_1$ because there are the same number of versions (one) for each.

   \[
   A_1 * A_2 * B_1 * C_1, \quad A_1 * B_1 * A_2 * C_1, \quad A_1 * B_1 * C_1 * A_2, \\
   B_1 * A_1 * A_2 * C_1, \quad B_1 * A_1 * C_1 * A_2, \quad B_1 * C_1 * A_1 * A_2
   \]

3. Introducing associations we recognize that associating $(A_1 * A_2)$ would reduce the expression to $A_{111}$ and similarly associating $(B_1 * C_1)$ would reduce the expression to $A_{21}$.

   \[
   (A_1 * (A_2 * B_1)) * C_1, \quad A_1 * ((A_2 * B_1) * C_1) \\
   ((A_1 * B_1) * A_2) * C_1, \quad (A_1 * B_1) * (A_2 * C_1), \quad A_1 * (B_1 * (A_2 * C_1)) \\
   (A_1 * (B_1 * A_2)) * C_1, \quad A_1 * ((B_1 * A_2) * C_1) \\
   ((A_1 * B_1) * C_1) * A_2, \quad A_1 * (B_1 * (C_1 * A_2)), \quad (A_1 * B_1) * (C_1 * A_2) \\
   ((B_1 * A_1) * A_2) * C_1, \quad (B_1 * A_1) * (A_2 * C_1), \quad B_1 * (A_1 * (A_2 * C_1)) \\
   ((B_1 * A_1) * C_1) * A_2, \quad (B_1 * A_1) * (C_1 * A_2), \quad B_1 * (A_1 * (C_1 * A_2)) \\
   (B_1 * (C_1 * A_1)) * A_2, \quad B_1 * ((C_1 * A_1) * A_2)
   \]

We rule the expressions where $B_1$ or $C_1$ appears outside the parentheses as sequential
and remove them leaving,

\[
A_1 * ((A_2 * B_1) * C_1) \\
(A_1 * B_1) * (A_2 * C_1) & A_1 * (B_1 * (A_2 * C_1)) \\
A_1 * ((B_1 * A_2) * C_1) \\
((A_1 * B_1) * C_1) * A_2 & A_1 * (B_1 * (C_1 * A_2)) & (A_1 * B_1) * (C_1 * A_2) \\
(B_1 * A_1) * (A_2 * C_1) \\
((B_1 * A_1) * C_1) * A_2 & (B_1 * A_1) * (C_1 * A_2) \\
(B_1 * (C_1 * A_1)) * A_2
\]

Enumeration Rule 4 allows us to commute basic random variables within an associative pair. If we rewrite the above with the preference of associating \(A_1\) with \(B_1\) and \(A_2\) with \(C_1\) and with \(A_1\) or \(A_2\) appearing first in any pair we create a number of duplicate expressions,

\[
A_1 * ((A_2 * C_1) * B_1) \\
(A_1 * B_1) * (A_2 * C_1) & A_1 * (B_1 * (A_2 * C_1)) \\
A_1 * ((A_2 * C_1) * B_1) \\
((A_1 * B_1) * C_1) * A_2 & A_1 * (B_1 * (A_2 * C_1)) & (A_1 * B_1) * (A_2 * C_1) \\
(A_1 * B_1) * (A_2 * C_1) \\
((A_1 * B_1) * C_1) * A_2 & (A_1 * B_1) * (A_2 * C_1) \\
(C_1 * (A_1 * B_1)) * A_2
\]
Removing the duplicates leaves the following with annotations,

\[
A_1 \ast ((A_2 \ast C_1) \ast B_1) \subset A_1 \ast (B_1 \ast (A_2 \ast C_1))
\]

\[
(A_1 \ast B_1) \ast (A_2 \ast C_1)
\]

\[
A_1 \ast (B_1 \ast (A_2 \ast C_1))
\]

\[
((A_1 \ast B_1) \ast C_1) \ast A_2 \subset A_2 \ast (C_1 \ast (A_1 \ast B_1)) \sim A_1 \ast (B_1 \ast (A_2 \ast C_1))
\]

\[
(C_1 \ast (A_1 \ast B_1)) \ast A_2 \subset A_2 \ast (C_1 \ast (A_1 \ast B_1)) \sim A_1 \ast (B_1 \ast (A_2 \ast C_1))
\]

Thus we have two remaining generic cases,

\[
A_1 \ast (B_1 \ast (A_2 \ast C_1)) \quad (A_1 \ast B_1) \ast (A_2 \ast C_1)
\]

4. Now we introduce specific operations in place of the generic “\ast” operation. Recall that division is computationally similar to multiplication within a basic random variable pair because basic random variables may be replaced by their reciprocal. Recall that division is a separate operation from multiplication is followed by addition of the right-hand operand. That is \(B_1 \div (A_2 + C_1)\) is computationally dissimilar from \(B_1 \times (A_2 + C_1)\). We first expand the inner parentheses of the first case namely
\[(B_1 \ast (A_2 \ast C_1)),\]

\[
B_1 + (A_2 + C_1) = A_2 + (B_1 + C_1) \quad \in A_{11}
\]

\[
B_1 + (A_2 \times C_1)
\]

\[
B_1 \times (A_2 + C_1)
\]

\[
B_1 \times (A_2 \times C_1) = A_2 \times (B_1 \times C_1) \quad \in A_{11}
\]

\[
B_1 \div (A_2 + C_1)
\]

Discarding the reduced \((+, +)\) and \((\times, \times)\) cases we expand the full generic case, \(A_1 \ast (B_1 \ast (A_2 \ast C_1)),\)

\[
A_1 + (B_1 + (A_2 \times C_1)) = B_1 + (A_1 + (A_2 \times C_1)) \quad \text{sequential}
\]

\[
A_1 \times (B_1 + (A_2 \times C_1)) \sim A_1 \times B_1 + A_2 \times C_1
\]

\[
A_1 \div (B_1 + (A_2 \times C_1))
\]

\[
A_1 + (B_1 \times (A_2 + C_1)) = (A_1 + A_2 \times B_1) + B_1 \times C_1 \quad \text{sequential}
\]

\[
A_1 \times (B_1 \times (A_2 + C_1)) \sim (A_1 + (A_2 \times C_1)) \times B_1 \quad \text{sequential}
\]

\[
A_1 + (B_1 \div (A_2 + C_1))
\]

\[
A_1 \times (B_1 \div (A_2 + C_1)) = B_1 \times (A_1 \div (A_2 + C_1)) \quad \text{sequential}
\]

Thus we expand \(A_1 \ast (B_1 \ast (A_2 \ast C_1))\) to the following cases,

\[
(A_1 \times B_1) + (A_2 \times C_2) \quad \frac{A_1}{B_1 + (A_2 \times C_1)} \quad A_1 + \frac{B_1}{A_2 + C_1}
\]

Now we turn to the expansion of the \((A_1 \ast B_1) \ast (A_2 \ast C_1)\) expression. Following all
the familiar rules,

\[(A_1 + B_1) + (A_2 + C_1) \sim (A_1 + B_1) + C_1 \quad \text{sequential}\]

\[(A_1 + B_1) + (A_2 \times C_1) \sim (A_1 + A_2 \times C_1) + B_1 \quad \text{sequential}\]

\[(A_1 + B_1) \times (A_2 + C_1) \]

\[(A_1 + B_1) \times (A_2 \times C_1) \sim (A_1 + (A_2 \times B_1)) + C_1 \quad \text{sequential}\]

\[(A_1 \times B_1) + (A_2 + C_1) \sim (A_1 + (A_2 \times B_1)) + C_1 \quad \text{sequential}\]

\[(A_1 \times B_1) + (A_2 \times C_1) \]

\[(A_1 \times B_1) \times (A_2 + C_1) = (A_1 \times (A_2 + C_1)) \times B_1 \quad \text{sequential}\]

\[(A_1 \times B_1) \times (A_2 \times C_1) = (A_1 \times (A_2 \times C_1)) \times B_1 \quad \text{sequential}\]

\[(A_1 + B_1) \div (A_2 + C_1) \]

\[(A_1 \times B_1) \div (A_2 + C_1) = (A_1 \div (A_2 + C_1)) \times B_1 \quad \text{sequential}\]

There are three new cases, but one is a duplicate so the final set of computationally dissimilar cases peculiar to \(A_{211}\) are,

\[A_{211} = \{(A_1 \times B_1) + (A_2 \times C_2), \ (A_1 + B_1) \times (A_2 + C_1), \]
\[
\frac{A_1}{B_1 + (A_2 \times C_1)}, \ A_1 + \frac{B_1}{A_2 + C_1}, \ A_1 + \frac{A_1 + B_1}{A_2 + C_1}\}\]

5.2.10 Insights Gained

Taking a glass box approach to model analysis as opposed to the traditional black box approach adopted by the Monte Carlo method, as defined in Saltelli [20], leads us directly to the consideration of algebraic correlation. We found that the PHoX modeling system uses a symbolic computation approach to determine the simplest case for processing each mathematical expression involving random variable operands. The PHoX modeling system then
uses robust algorithms that balance numerical accuracy and computational performance. The robustness of the algorithms allows the symbolic processing algorithms to be more coarse-gained in their determination of specific mathematical expression case.

An advantage the PHoX approach of direct computation of random variables has over randomization of inputs such as the Monte Carlo approach is that of coverage. This advantage will become more clear when we describe the consequences of programmatic conditional statements such as if, then, else have when employing random variable operands.

If, instead of the PHoX approach of arithmetically combining random variables, we had repeatedly run our model with sets of random values chosen from the random variables representing our model inputs, then likely outcomes would necessarily occur most often. More to the point, we would spend most of the model runs in an effort to observe the likely outcomes. A more significant issue arises when we consider unlikely outcomes.

Consider a model the predicts patient outcome when given a certain drug. For simplicity suppose that the output of this model is success (patient survives) or failure (patient dies). If the true likelihood of failure is, say, 1 in 1 billion, then we expect to run a randomized-input type model at one billion times to observe a single failure. To be confident the single observation of failure is one in one billion we need to run the model many times one billion runs so that several failures are observed. Using computational effort (time and hardware resources) as a proxy for importance of outcome we see that the randomized-input approach attaches nearly no importance to the most significant outcome, patient death in this case.

In the PHoX-style model there is no importance associated with likelihood and similar computational effort is applied to each quanta of outcome likelihood. In the drug example, PHoX would apply similar computational effort to each of the two outcomes, success and failure, without prejudice for either.
6.1 Numerically Computing Correlated Random Variables

The key to numerically computing expressions with correlated random variables is to establish a partition for all model input random variables with an associated and to track this associated index through any expression in which it appears.

In the previous section where we described how PHoX numerically computes functions and arithmetic combinations of independent random variables we established the concepts of a fixed partition for a random variable and of synchronous random variables. A random variable for which we establish a fixed partition is a *first class* random variable. All model inputs are first class random variables. A random variable that is the function of a first class random variable is representable as a synchronous random variable. Since synchronous random variables may not be in proper form they may also be represented as a first class random variable which is by definition in proper form.

As we introduce other random variables into random variable expressions we will all that such as expression has many forms in which as we say it is *polymorphic*.

We will expand our description of the PHoX modeling system to include *symbolic* representations of random variable expressions. This means that a random variable may be represented in synchronous, proper or symbolic form where it consist of a *parse tree*.
the describes the steps by which the other forms may be obtained. The term parse tree is common to compilers of computer languages as detailed in Aho [1].

Before we begin describing how the PHoX modeling system represents models using parse trees we must first expand our numeric representation of random variable.

6.1.1 Expanded Numeric Representation of Random Variables

In the PHoX modeling system random variables are recognized numerically as two kinds; first class and synchronous. We augment our definitions of a first class random variable, \( A \), as follows,

\[
A = \{ (a_1, ..., a_n), (p_1, ..., p_n) \}
\]

and,

\[
-\infty = a_1 < a_2 < ... < a_n = +\infty \quad n > 2 \quad p_{n-1} = 0
\]

where \( p_1 \) and \( p_n \) represent the probability in the neighborhood of \(-\infty\) and \(+\infty\), respectively. Keeping the partition array and the probability array the same length has implementation advantages when indexing within programmatic loops, but since probability values represent the probability between partition endpoints one of the probability values must be eliminated. This is accomplished with the convention that the penultimate probability value fixed as zero.

Let \( A_1 \) be a synchronous random variable with respect to \( A \) so that,
\[ A_1 = (a_{1,1}, \ldots, a_{1,n}) \]

and we do not require any specific ordering of the \( a_{1,i} \) values or for them to be unique. We understand that \( a_{1,i} \) is some function of \( a_i \) for each \( i = 1 \ldots n \).

We notice that the Sunrise algorithm described above is not the only way to generate a first class random variable from \( A \). In fact, the Sunrise algorithm provides a natural choice of partition upon which to project \( A_1 \).

When giving numeric examples we will alter the representation of first class and synchronous random variables for clarity. When presenting first class random variables we may choose to omit the infinite endpoints. Similarly, when presenting synchronous random variables we may choose to include the interleaving probability values as an adjoining array for when the associated first class random variable is not available. This allows us to present random variables in improper form without concern of their parentage. In this case we may name a random variable \( B \) and not \( A_1 \) or \( A_2 \), etc, if the first class \( A \) is not necessary for the examples point.

For example, if we have the following synchronous random variable (improper form),

\[ B = \{(1, 0, 2), (0.5, 0.5)\} \]

Recall that if we use the Sunrise algorithm to create a proper form random variable from \( B \), called \( C \) the result is,
\[ C = \{ (0, 1, 2), (0.75, 0.25) \} \]

If we desire a different partition for the proper form of \( B \) we use an algorithm such as depicted in figure 5.2. Suppose we wished to used the following partition,

\[ (-5, 0, 5) \]

then by inspection we see that the proper form of \( B \) given this particular partition, called \( D \), is

\[ D = \{ (-5, 0, 5), (0, 1) \} \]

A random vector is a list of random variables as defined in standard statistics texts such as Bickel [4]. Random vectors are often presented with the random variables independent and identically distributed and denoted \( iid \). In the PHoX modeling system random variables may be correlated and a representation that allows that possibility is required.

We first define and discuss 2D random variables then \( nD \) random variables.

A 2D random variable comes in the familiar two flavors; first class and synchronous. A first class 2D random variable \( A \) is defined as,

\[ A = \{ ((a_i^1), (a_j^2)), (p_{i,j}) \} \quad \text{where } i = 1, \ldots, n, \ j = 1, \ldots, m \]
The infinite endpoint and zero penultimate probability constraints are described by the requirement that \( A \) be consistent with the following construction. Suppose \( B \) and \( C \) are two independent first class random variables. We can encode them into a 2D first class random variable as follows,

\[
B = \{(b_1, \ldots, b_n), (p_1, \ldots, p_n)\}
\]
\[
C = \{(c_1, \ldots, c_m), (q_1, \ldots, q_m)\}
\]
\[
A = \{(b_1, \ldots, b_n), (c_1, \ldots, c_m), (p_i \times q_j)_{i=1,\ldots,n, j=1,\ldots,m}\}
\]

We note that such a construction may internally represent the 2D array of probability values of \( A \) as a pair of references the first to the probability array of \( B \) and the second to that of \( C \) as in,

\[
(p_i \times q_j)_{i=1,\ldots,n, j=1,\ldots,m} = (p_1, \ldots, p_n) \times (q_1, \ldots, q_m)
\]

which is true as long as the component random variables are independent. If any programmatic operation causes the assumption of component random variable independence to be violated the PHoX modeling system will switch to a full 2D array representation of probability values as per the definition. We refer to this as virtual implementation since the PHoX modeling system behaves as if a 2D array is present while a pair of 1D arrays may actually be stored in the computer memory. This concept of virtual implementation will be expanded upon later.

We turn now to the representation of synchronous 2D random variables. The definition is consistent with the 1D definition. Assuming the 2D first class random variable \( A \) above
we create a 2D synchronous random variable $A_1$ with respect to $A$ by application of some 2D function, $f$ as follows,

$$A = \{(a^1_i, a^2_j, p_{i,j})\}_{i=1,...,n, j=1,...,m}$$

$$A_1 = (f(a^1_i, a^2_j))_{i=1,...,n, j=1,...,m}$$

that is, $A_1$ is a 2D array of values that are functionally related to the two 1D arrays representing the rectangular partitioning of $\mathbb{R}^2$.

When we wish to de-emphasize the relationship between a first class 2D random variable $A$ and its synchronous offspring $A_1$, as in the 1D case, we pair the 2D value array of $A_1$ with the 2D probability array of $A$. If this alternate representation is called $D$ then we have,

$$D = \{(d_{i,j}, p_{i,j})\}_{i=1,...,n, j=1,...,m}$$

This does not represent an alternate implementation within the PHoX modeling system, only an alternate description for the purposes of presenting the necessary concepts within this written work.

We now introduce a further extension of the definition of a synchronous 2D random variable, the possibility that more than one functional relation exists between itself, $A_2$ and its parent first class 2D random variable $A$. Suppose we have two 2D functions $f$ and $g$ so that $A_2$ is defined as,
\[
A = \{((a_i^1), (a_j^2)), (p_{i,j})\}_{i=1,\ldots,n, j=1,\ldots,m}
\]
\[
A_2 = (f(a_i^1, a_j^2), (g(a_i^1, a_j^2)))_{i=1,\ldots,n, j=1,\ldots,m}
\]

where \(A\) is restated for convenience. The plethora of parentheses notwithstanding, a synchronous 2D random variable is a list of one or, in this case, two 2D arrays functionally related to an associated first class 2D random variable.

We will provide some motivation for the definition of a synchronous 2D random variable below and discuss how to produce proper-form (first class) random variables from synchronous random variables. We note that if a synchronous 2D random variable contains one 2D array then we can produce a first class 1D random variable once a 1D partition is provided. If a synchronous 2D random variable contains two 2D arrays then we can produce either two 1D first class random variables representing a pair of marginal probability distributions or one 2D random variable.

The reader will notice that there various definitions are building toward a representation of the interactions between n-dimensional functions, partial integrations and change-of-variable formulas.

We end this line of discussion with the definition of an nD first class random variable \(A\) and one of its associated nD synchronous random variables as,

\[
A = \{((a_{i_k}^k)_{i_k=1,\ldots,N_k})_{k=1,\ldots,n}, (p_I)_{I \in \times_{k=1,\ldots,n} (1,\ldots,N_k)}\}
\]
\[
A_1 = (f_j((a_{i_k}^k)_{i_k=1,\ldots,N_k})_{k=1,\ldots,n})_{j=1,\ldots,m}
\]

where \(A\) is composed of a list of \(n\) partitions of the real line and an n-dimensional
rectangular array of corresponding probability values. Similar to the 2D case, $A_1$ is list of $m \leq n$ n-dimensional arrays of values.

Now that we have our implementation structures are in place we present the modeling conditions that necessitate them. When unspecified, we assume a random variable is of one dimension and not the more general higher dimensional cases. Some preliminary techniques for numerical computation with higher dimensional random variables will be presented. In a later section we build a prototype and present further techniques that provide a good balance of fidelity and computational performance. Other balance points are possible within the framework described herein.

6.1.2 Computing Expressions with One First Class Random Variable

Given a single first class random variable $A$ we consider how to compute aggregate expressions. We represent aggregate algebraic expression with a directed graph. For example, the expression $A^2 + 2A$ may be represented in figure 6.1.

where we have associated an $i$ with each direction arrow. Since we assume $A$ has a fixed partition we will index it by $i$. Similarly we will index $B$ by $j$ and $C$ by $k$ in later discussion. We will also form aggregate indices such as $ij$, $jk$ or $ik$ which will also be
discussed below.

The four random variables $A$, $A^2$, $2A$ and $A^2 + 2A$ shown in the figure above are all synchronous therefore their computation is a pointwise computation. If any such random variable needs to be presented to the user, the Sunrise algorithm is applied to put the random variable into proper form.

The main point to note is that the $A^2 + 2A$ expression has two parents, but they share a common index $i$ which indicates they share a common first class random variable, namely $A$. To compute $A^2 + 2A$ we treat each parent as a function of $i$ and then have,

\[
A = \{(a_1, \ldots, a_{n+1}), (p_1, \ldots, p_n)\}
\]
\[
A^2 = \{(a_1^2, \ldots, a_{n+1}^2), (p_1, \ldots, p_n)\}
\]
\[
2A = \{(2a_1, \ldots, 2a_{n+1}), (p_1, \ldots, p_n)\}
\]
\[
A^2 + 2A = \{(2a_1 + a_1^2, \ldots, 2a_{n+1} + a_{n+1}^2), (p_1, \ldots, p_n)\}
\]

We next consider the case with two first class random variables, the 2D case. The techniques we use for the 3D, 4D, etc. cases differ in order to maintain the balance between computation performance and numerical accuracy. In addition there are mixed aggregate cases we will address.

6.1.3 Computation Technique for the 2D Case

Consider the directed graph in figure 6.2.

We have already considered how to compute $3B$ and $1/C$ since these are one dimensional functions of $B$ and $C$ respectively. The last random variable, $3B + 1/C$ in a function
of two first class random variables. In general we are computing $B_2 = f(B_1, C_1)$.

We assume that $j = 1..N_B$, $k = 1..N_C$ where $N_B$ and $N_C$ are the number of partition elements of $B$ and $C$ respectively and by synchronicity of $3B$ and $1/C$ respectively. We then form the partitioned rectangle seen in figure 6.3.

The major axes of $B$ and $C$ dictate the extents, partitioning and joint probability of each partition element within the enveloping rectangle. The auxiliary axes are synchronous with their associated axis, $B_1$ is synchronous with $B$ and $C_1$ is synchronous with $C$. The
darkened rectangle is indexes by \((i, j)\) and referred to as the \(i, j\)-rectangle. Assuming,

\[
B = \{(b_1, \ldots, b_{N_B+1}), (q_1, \ldots, q_{N_B})\}
\]

\[
C = \{(c_1, \ldots, c_{N_C+1}), (r_1, \ldots, r_{N_C})\}
\]

For the running example we have \(B_1 = 3B, C_1 = 1/C\) and \(B_2 = B_1 + C_1\) and \(i, j\)-rectangle as shown in figure 6.4.

where \(x_{j,k} = 3b_k + 1/c_k\) and similarly for the other three vertex values. To determine the proper form of the result we need a partition and the associated lists of probability values.

6.1.3.1 Determining the 2D Partition

For the 2D case involving \(B\) and \(C\) first class random variables have have both a \(N_B \times N_C\) lattice of vertices and a corresponding array of \(i, j\)-rectangles. We assign the generic
name \( X \) to a random variable expression. In our example we have \( X = 3B + 1/C \).

The PHoX system follows the idiom of *lazy evaluation* which means, in this case, that no further computation is performed. Just as in the previous cases a proper form of \( X \) may be requested in which case we must first prepare a partition for \( X \).

A direct way to determine the partition for \( X \) is to sort the list of all \( x \)-values from the \( i, j \)-rectangle vertices. If, for example a 1000 point partition is desired for \( X \) and the lattice of \( x \)-values consists of one million points, then we might choose every one thousandth point in the sorted list of \( x \)-values as a partition endpoint.

The drawback with the above exhaustive method is that it requires all \( x \)-values to be computed at this stage and in particular for the entire list to be sorted. An alternative is to compute a random selection of 1000 \( x \)-values, sort this list and use each value as a partition endpoint. The drawback here is that extreme values may not be adequately represented and partition widths are not controlled.

The joint \( B \times C \) probability will be fully partitioned regardless of the choice of partition endpoints since \( \pm \infty \) are the true endpoint of a random variable partition by definition.

Describing a *best* partition identification method is beyond the scope of this paper. Two computationally tractable methods have been presented which meets the standard set for this paper.

6.1.3.2 Apportioning the 2D Joint Probability

Each \( j, k \)-rectangle of the 2D joint distribution carries a portion of the total probability which must be allocated to the appropriate partition element of our resultant random variable, \( X \). Given a partition for \( X \) we process each \( j, k \)-rectangle in turn and allocate the \( q_{jr} r_{jk} \) probability to the appropriate \( X \)-partition elements.

The \( x \)-values for the \( j, k \)-rectangle can be in any order so we must recognize and address each case. Furthermore we must in principle check each \( X \)-partition element to determine
the probability contribution from the $j,k$-rectangle. We can provide a simple short-cut to this process by noticing that if $(a, b, c, d)$ represent the four $x$-values for the vertices of the $j,k$-rectangle and if the $X$-partition is represented by,

$$X = \{(x_1, x_2, ..., x_N); \text{(undetermined probability)}\}$$

then we start the scanning process of $X$ from

$$i_{\text{min}} = \text{argmax}_i(x_i \leq \text{min}(a, b, c, d))$$

$$i_{\text{max}} = \text{argmin}_i(\text{max}(a, b, c, d) \leq x_i)$$

Since we assume uniform probability density for each random variable then we have uniform probability density for the $j,k$-rectangle and our task reduces to finding the area between two $X$-partition endpoints. The endpoints we will use are then,

$$(x_{i_{\text{min}}}, ..., x_{i_{\text{max}}})$$

Our task is simplified when we observe that we can work with area differences. The image for this operation is shown in figure 6.5.

The fractional area of the $j,k$-rectangle is scaled by the probability carried by the $j,k$-rectangle ($q_j r_k$) and apportioned to each $X_i$ partition element in turn.

For clarity assume that the $x$-values are a $j,k$-rectangle are $a$, $b$, $c$ and $d$. The $j,k$-rectangle is then labeled as in figure 6.6.

Using these four vertices we transform the $j,k$-rectangle from $B,C$-space to $t,X$-space.
where \( t \) is a dummy variable corresponding to a normalized \( C \)-axis as figure 6.7 represents, where we have assumed that \((a, 0), (b, 1), (c, 1), (d, 0)\)-vertices of the \( t, X \)-shape may be connected with straight lines to form a trapezoid.

Since we will be computing the area of the intersection of this trapezoid and the \((-\infty, x_{i+1})\) interval we are free to mirror the trapezoid vertically and, as long as we track polarity, free to mirror horizontally as represented by figure 6.8.

This observation about mirroring allows us to assume without loss of generality that we always arrange for the \( a \)-vertex to correspond with the least \( X \)-value. There are then 3! = 6 possible orderings for the vertices,
Recall, we need only compute the area enclosed by the \((−∞, x_{i+1})\) interval for the \(X_i\) partition element. Notice that each ordering has several cases for any intersecting \(X_i\) partition element.

In figure 6.9 we show each of the 6 cases for \(x\)-values of vertices. The individual shapes have \(X\) as their horizontal axis and an artificial vertical axis giving each shape a unit height. The black-and-white shapes on the right show the three or four cases for constant \(x\)-value representing the endpoint of an \(X\)-partition element. The \(x\)-value in each case is indicated by a vertical line and the transition in color from black to white. We break out individual cases to access the complexity of the area-calculating formulas. The black coloring indicates the area of the \(j, k\)-rectangle intersecting the \(X_i\)-partition.

Notice that the blackened area in each of the black-and-white shapes above is, when normalized by the total area of the shape, the same as the normalized area of the corresponding \(j, k\)-rectangle. To see we integrate the fractional distance from the left edge of the shape to the given \(x\)-value as a function of height with respect to height (from zero to
Figure 6.9: i.j-Rectangle Cases
one). The result is the fraction of the area intersecting the $X_i$ partition with $x$ as the upper bound. The fractional distance integrand which is a horizontal line, when transformed to a normalized (unit) $j, k$-rectangle is still a horizontal line and of the same length. The result of the analogous integral of the intersection of the fraction of the $j, k$-rectangle with the $X_i$ partition is the same in either setting. The example depicted in figure 6.10 underscores the argument.

![Figure 6.10: i,j-Rectangle Integration](image)

In the figure above the left two sub-figures represent the $acdb$ case. The top-left sub-figure shows a number of vertical bars representing different values of $x$. The right two sub-figures represent the $j, k$-rectangle with normalized width and height. Each vertical bar ($x$-value) in the left sub-figures corresponds to a level curve in the right sub-figures. If we
assume that the \( X_i \) partition has \( x \) as its upper bound point it will intersect each sub-figure in the shaded region bounded by line or curve indicated by an \( x \). The lower two sub-figures make the fractional-area-equivalency argument. Each sub-figure is one unit tall. We need only compare the relative width of the horizontal line-of-integration indicated with an \( h \) in each sub-figure. In the lower-left sub-figure the length of the \( h \)-line is normalized by the width \( w \), a function of height \( h \). This normalized width is the same as the width of the \( h \)-line in the lower-right sub-figure by definition. Therefore the fraction of the shaded area with respect to the total area is the same in all sub-figures.

Referring to the lower sub-figures in the figure above we have,

\[
L(t) = (b - a)t + a \\
K(t) = (c - d)t + d \\
r(t|x) = x - L(t) \\
s(t|x) = K(t) - L(t) \\
w(t|x) = \frac{r(t|x)}{s(t|x)}
\]

The advantage to the observation that we can find the fraction of total area of the left sub-figures in the figure above instead of the right sub-figures (which represent the normalized \( j, k \)-rectangle) in a computationally efficient manner. We are finding areas of triangles and trapezoids rather than integrating rational polynomials as is required to compute the areas of the right sub-figures.
6.1.3.3 Computing 2D Fractional Area

Based on the labeling \((abcd)\) of the \(j,k\)-rectangle if \(b \leq c\) and \(a \leq d\) then we say that the rectangle is flat, if \(c < b\) then we say the rectangle is twisted.

Notice that in the twisted case there is an \(x\)-value where \(K(t) = L(t)\), that is, the figure crosses itself. We refer to the \(x\)-value of the crossing as \(x_m\) found as,

\[
L(t) = (b - a)t + a \\
K(t) = (c - d)t + d \\
t_m : L(t_m) = K(t_m) \\
\quad = \frac{d - a}{(b - a) - (c - d)} \\
x_m = L(t_m) \\
\quad = \frac{(b - a)(d - a)}{(b - a) - (c - d)} + a
\]

We may now enumerate the partition cases for all the permutations of \(a, b, c, d\) we must handle. The \(a\) is always first (since least) and for the flat case \(b \leq c\) so we have,

\[
\begin{array}{ccc}
  a|bcd & ab|cd & abc|d \\
  ab|dc & abd|c & \\
\end{array}
\]

For the twisted case we have the \(x_m\) value, simply denoted \(m\). The cases are then,
The 5 flat cases and 6 twisted cases are enumerated in figure 6.11.

Recalling that each figure is of unit height, the area the flat shape is that of a trapezoid,

\[ A_{\text{flat}} = \frac{(d-a) + (c-b)}{2} \]

The area of the twisted shape is that of two stacked triangles. The height of the lower triangle is the \( t \)-value of the crossing point already found as \( t_m = \frac{(d-a)}{(b-a) - (c-d)} \). The height of upper rectangle is \( 1 - t_m \). Since we have the \( x \)-value of the crossing point as \( x_m = \frac{((b-a)(d-a))}{((b-a) - (c-d)) + a} \) we can divide each rectangle into two right triangles and find the total area to be,
\[
A_{\text{twisted}} = \frac{1}{2} \left[ \{(x_m - a) + (d - x_m)\} t_m + \{(x_m - c) + (b - x_m)\} (1 - t_m) \right] \\
= \frac{1}{2} \left[ a^2 + b^2 + c^2 + d^2 - 2(ad + bc) \right] \\
= \frac{d - c + b - a}{2}
\]

We now find the areas corresponding to each of the \( x_i \)-cases in the flat cases above,

\[
A(x_1) = \frac{(x - a)^2}{2(b - a)} \\
A(x_2) = \frac{(x - b) + (x - a)}{2} \\
A(x_3) = A_{\text{flat}} - \frac{(d - x)^2}{2(d - c)} \\
A(x_4) = \frac{(x - a)^2}{2(b - a)} - \frac{(x - d)^2}{2(c - d)} \\
A(x_5) = A_{\text{flat}} - \frac{(c - x)^2}{2(c - d)}
\]

For the twisted cases we similarly have,
To find fractional areas for the flat- and twisted-cases need only be divided by $A_{flat}$ or $A_{twisted}$ respectively.

6.1.3.4 A Twisted-Case Example

The twisted cases arises readily in computations involving two random variables such as $B$ and $C$. Consider the product expression $X = B \times C$. The transformation of the joint $B,C$-rectangle from $B,C$-space to $X,C$-space is depicted in figure 6.12, where the circled values are $B_i \times C_j$ or simply $x$-values.

We may either view this as a single $j,k$-rectangle that includes the origin or the full extents of the joint probability distribution of $B,C$ where the partitions are not shown.

Notice that in this case, had the partition of $B$ or that of $C$ included the origin as an endpoint this twisted case would not arise. We form the intuition (without proof) that twisted cases are a rare occurrence. Notice that the existence of a twisted cases is detectable and indicative of a potentially interesting feature of the model.
6.1.3.5 Some Illustrative Examples

In the following example we assume the usual form for first-class random variables $A$ and $B$, namely,

$$A = \{(a_1, ..., a_{N+1}), (p_1, ..., p_N)\}$$

$$B = \{(b_1, ..., b_{M+1}), (q_1, ..., q_M)\}$$

And we let $X = f(A, B)$ and $Y = g(A, X)$ for real valued functions $f$ and $g$. The result is shown in figure 6.13.
To compute the proper form of $X$ we apply the 2D method described above. In particular we form the 2D lattice,

$$
\begin{pmatrix}
    f(a_1, b_{M+1}) & \cdots & \cdots & f(a_{N+1}, b_{M+1}) \\
    \cdots & f(a_i, b_{j+1}) & f(a_{i+1}, b_{j+1}) & \cdots \\
    \cdots & f(a_i, b_j) & f(a_{i+1}, b_j) & \cdots \\
    f(a_1, b_1) & \cdots & \cdots & f(a_{N+1}, b_1)
\end{pmatrix}
$$

where the $i, j$-rectangle is shown explicitly. The joint probability of the $i, j$-rectangle is $p_i \times q_j$ as usual. Our goal is not to compute $X$, but to show how to compute the proper form of $Y$.

We notice that the 2D lattice we need to compute the proper form of $Y$ is constructed pointwise. We see that $Y$ has two parent expressions, $A$ and $X$. We see further that $A$ is accessed by the single index $i$ whereas $X$ requires two indices $i$ and $j$. Therefore we must form a 2D $i, j$ lattice. We treat $X$ as a doubly indexed function $X(i, j)$ to access its 2D lattice pointwise. We can, in principle, view $A$ as a doubly indexed function $A(i, j)$ and access its virtual 2D lattice. Clearly $A$ is only singly indexed, but it’s illustrative to view it as containing a 2D lattice because then we can proceed pointwise. The 2D $i, j$ lattice for $Y$ is then,

$$
\begin{pmatrix}
    g(a_1, f(a_1, b_{M+1})) & \cdots & g(a_{N+1}, f(a_{N+1}, b_{M+1})) \\
    \cdots & g(a_i, f(a_i, b_j)) & \cdots \\
    g(a_1, f(a_1, b_1)) & \cdots & g(a_{N+1}, f(a_{N+1}, b_1))
\end{pmatrix}
$$

If $f$ and $g$ are both addition then the $i, j$ lattice point is,
In the correlation section we needed to recognize that 
\( a_i + (a_i + b_j) = 2a_i + b_j \) in order 
to observe the expression required only a 2D joint probability distribution. BY working 
pointwise in the fashion described by the 2D method and evidenced in this example, no 
higher-level computations are required. That is, PHoX does not need to provide symbolic 
manipulation of the expression parse trees and may proceed in a direct manner.

The following example is the sum,

\[ W = ((A + B) + (C + D)) + E \]

The parse tree for this example is shown in figure 6.14.

To compute \( W \) directly would require 5 dimensions. If each first class random variable 
\( A, ..., E \) has a 1000-point partition then \( W \) requires a \( 10^{15} \)-point lattice. This is a computa-
tionally intractable approach. Instead we use the dimension reduction method and compute proper forms for $X = A + B$ and $Y = C + D$ using the 2D method. This allows us to use the 2D method to compute $Z = X + Y$ where the proper form of $X$ is indexed by $ij$ (no comma) and $Y$ is indexed by $jk$ (no comma). With $Z$ in proper form it can be indexed by $ijkl$ (no commas) and finally $W$ is computed with the 2D method with 2D lattice indexed by $ijkl,m$.

A question that is beyond the scope of this paper is whether the parentheses in this example affect the result for $W$. This question bears directly on the fidelity of the intermediate random variables $X$, $Y$ and $Z$. Since the computation of the proper form of a random variable is a numerical integration issue we need to provide error bounds in some form. We make no fidelity assertion in this paper save for the empirical observation that 1000 point partitions for random variable throughout the model seem to consistently provide graphically satisfying results as compared to theoretical results where available.

6.1.4 Computation Technique for the 3D Case

Building on our running example, we form the expression that is the quotient of the two previously considered expressions namely,

$$A_1 = A^2 + 2A$$
$$BC_1 = 3B + 1/C$$
$$X = BC_1 / A_1$$
$$= \frac{3B + 1/C}{A^2 + 2A}$$

The full directed graph for this final expression is shown in figure 6.15.
The last expression has two graph edges connecting it to its parent expressions. The leftmost edge has an $i$ index and the rightmost has a $j, k$ index-pair. There are two ways to proceed. The *dimension reduction method* and the *Dimension Preservation Method*.

### 6.1.4.1 Dimension Reduction Method

Simply creating the expression $X = (3B + 1/C) \div (A^2 + 2A)$ in a PHoX-hosted model does not cause any computation other than the formation of the directed graph in the previous figure. Following the principle of lazy evaluation described above, calculation ensues when a proper-form random variable is requested by the user or required elsewhere by the model.

Referring to figure 6.15 where one expression is indexed by $i$ and the other is doubly-indexed by $j$ and $k$ we notice there is no overlap of indices. This means that the parent expressions to $X$ are mutually independent. We will discuss below what happens when this is not true. We now have the option of asking the PHoX modeling system to compute a proper form random variable for the expression $3B + 1/C$. If we choose to do this, the
3B + 1/C is replaced by BC1, a proper-form first-class random variable with aggregate index jk (no comma). We calculate it using the 2D algorithm described in the last section. The symbolically transformed directed graph is shown in figure 6.16.

The association of BC1 with its underlying expression 3B + 1/C is not lost, but merely inactive for the X expression given the reduction method choice.

To compute the proper-form of the last (X) expression we use the 2D method just as for BC1.

6.1.4.2 Dimension Preservation Method

While it is possible to compute the proper form for an expression such the (3B + 1/C) ÷ (A^2 + 2A) used in our running example using a sequence of 2D-method calculations, this is not always an available option. For example,
In this example (see figure 6.17) we see that the method used to compute a proper-form random variable depends on context. If we used the 2D method to compute a proper form of $X$ this could serve as an first-class parent for $Y$. The proper form of $Y$ could then be computed using the dimension reduction method and would have a $j$-indexed parent and an $ik$-indexed parent, but no longer an $i$-indexed parent because of the dimension reduction action.

To compute the proper form of the $Z$ expression we notice that we require that $Y$ retain $i$ as a separate index. This in turn requires that $X$ retain its $i$-index parent. There is no conflict since PHoX random variables allow multiple representations. Rather than computing the
proper form of $X$, we allow $X$ to represent a two dimensional lattice of values indexed by $i, j$ whose values, in this example, represent the sum of partition endpoints of $A$ and $C$. Similarly $Y$ represents a three dimensional lattice indexed by $i, j, k$. To compute the proper form of $Z$ we form the three dimensional lattice by indexing by $i, j, k$ and combining $a_i$ with $y_{i,j,k}$, in this case through addition, where $a_i$ is the $i^{th}$ endpoint of the established partition of $A$ and similarly for $y_{i,j,k}$.

If we suppose that each partition of a first-class random variable in the PHoX system has 1000 endpoints then the 3D lattice for $Z$ would have one billion points and roughly a corresponding one billion 3D blocks. Each $i, j, k$-block has 8 vertices, $z_{i,j,k}, \ldots, z_{i+1,j+1,k+1}$. If we follow a method analogous to the 2D method for computing fractional area and allocating fractional-block probabilities to a $Z$-partition we run into both a computational barrier and significantly more cases for volume fragments than for the 2D area fragments. If we assign a 1000-point partition to $Z$ then there are roughly one million contributing $i, j, k$-blocks compared to the roughly one thousand contributing $i, j$-rectangles in the 2D case, for a similar partition element.

Since there are some many more $i, j, k$-blocks per $Z$-partition element compared to the 2D case we choose to compute a single representative $z_{i,j,k}$-value for each $i, j, k$-block. In practice we may average the $z$-values of the 8 vertices of the $i, j, k$-block or simply to opposing vertices to speed computation.

By having a single $z_{i,j,k}$ value for each $i, j, k$-block we have reduced the piecewise continuous problem to a discrete problem. We will now describe an algorithm for assigning the joint probability to the $Z$ partition.

Each lattice point of $Z$ has a $z_{i,j,k}$ value and a corresponding $p_{i,j,k}$ joint probability value derived from the associated first-class random variables $A$, $B$, and $C$. We iterate over each $i, j, k$-lattice point. The $Z$-partition is implemented, as usual, as a parallel pair of arrays of end-point values and probability values. The probability array is initialized with
zero values. Each lattice point will contribute its associated probability exclusively to a partition element. Finding the correct $Z$-partition element for the first lattice point involves a computationally expensive search of the $Z$-partition endpoint values. The next lattice points will benefit from the assumption that their associated $Z$-partition element is nearby. Exploiting this observation is then balanced with the computational expense of a full $Z$-partition search. In this manner the computational cost per lattice point it lower enough to ensure acceptable computational performance overall.

Referring to the example in this section, we notice that $Y = B \div (A + C)$ could be represented with two indices $i, jk(i)$, rather then three where $jk(i)$ is an aggregate index parametrized by $i$. This suggests an extension to the dimension reduction method.

6.1.4.3 Extended Dimension Reduction Method

Suppose a 3D lattice of values forming the vertices of blocks of uniform probability distribution indexed by $i, j, k$. We may regard the lattice as an indexed set of 2D lattices. Each 2D lattice represents the joint probability distribution indexed by $j, k$. We refer to each of these 2D lattices as sheets.

In the dimension reduction method described above there was one such sheet and from it we computed proper form random variable whose index was labeled $jk$ (no comma). Notice in particular that the range of $jk$ is unrelated to the ranges of $j$ or of $k$, but serves as the index over the chosen partition for the new random variable.

The extension of the dimension reduction method involves computing a proper form random variable for each index $i$. Assuming that $i$ is associated with first-class random variable $A$ with partition size $N$, this extended method is $N$-fold more computationally intense than the original dimension reduction method. Therefore it would not be chosen if the original method where available and if it can be shown to be superior in some way to the dimension preserving method.
In the dimension preserving method we represent each block (for three dimensions and above) with a single pair of values. Assuming we are computing the proper form of a random variable called $Z$ then the pair of values is the $z_{i,j,k,...}$ result value and $p_{i,j,k,...}$ joint probability value, as usual. Instead of a single point, each block is represented by one of its faces. If $i$ is the sheet index then this face is the $j,k$-face of the $i,j,k$-block, that is, the face that includes the $z_{i,j,k}$, $z_{i,j+1,k}$, $z_{i,j,k+1}$ and $z_{i,j+1,k+1}$ vertices.

Referring to the example in the previous section we find the result in figure 6.18, where $Y = \frac{B}{A+C}$.

There are two advantages this extended dimension reduction method has over the dimension preserving method. The first is that each $i,j,k$-block is possibly partitioned according the partition that occurs at one of its faces instead of the all-or-nothing approach.

The figure 6.19 represents an $i,j,k$-block. The front face is the $jk(i)$ face. The embedded plane represents the projection into the block of the $Z$-partition of the front face. The fraction of the volume times the joint probability of the block is partitioned to two adjacent $Z$-partition elements.
The second advantage offered by this method over the dimension preserving method comes if \( jk(i) \) is reusable by other expressions. It is computationally more expensive that a single application of the dimension preserving method, but not necessarily more expensive than two separate dimension preserving calculations. The question of which method to use is left to the executive portion of the PHoX modeling system that will choose the manner in which all random variables are ultimately computed.

6.1.4.4 Computing 4D Cases and Higher

In practice, probability distributions tend to have lower probability ranges. The tails of the Gaussian probability distribution fall off in significance for intervals far from the mean. The joint probability distribution of random variables with many low probability intervals results in extremely low probability rectangles and blocks. Suppose an interval of random variable \( A \), say \((a_i, a_{i+})\) has probability \(1E - 10\) and similarly, \(P(b_j < B < b_{j+1}) = 1E - 10\) and \(P(c_k < C < c_{k+1}) = 1E - 10\). Suppose further that \( X \) is a function of \( A, B \) and \( C \) then the joint probability of the \( i, j, k \)-block is \(1E - 30\). The sum of each one billion such probabilities is still only \(1E - 21\) so it is reasonable to set a probability threshold.
whereby joint probability values below the threshold are considered zero.

If a $j, k$-rectangle in the 2D case or the $i, j, k$-block in the 3D case have probability values below some established threshold then any computation associated with them may be skipped with minimal computational overhead. Considering, for example, the computation of the joint distribution of three independent Gaussian distributions. A large percentage of the $i, j, k$-blocks may fall below the probability threshold and their computation skipped increasing the computational performance of the overall calculation.

Another method that may be employed when the dimension of an expression cannot be reduced by any available method is to perform a random sample. To compute the proper form of an expression $X$ involving $n$ indices, we create an $N$-point sample of the $n$-dimensional lattice. We assume the probability of each lattice point is concentrated at that point as in the discretization used for the 3D method. We then find the fraction $f$ of sample points compared to the number of points in the full lattice and scale up this probability of each sampled lattice point. We can use a smaller sample to form the endpoints of the $X$-partition as we did in the 3D method.

For example if we have a 5-dimensional lattice for $X$ with 1000-points in each dimension then a one million point sample can of the lattice can be used to form a 1000-point $X$-partition and a one billion point sample with rescaled probabilities can be allocated to the 1000-point $X$-lattice. Such a scheme will be computationally equivalent to the 3D equivalent case, but with some degradation in random variable fidelity.

6.1.5 Technique for Incorporating Conditional Statements

The AB32 model has few conditional statements in the data processing modules, but the linear programming modules contains a great number of conditional statements. In this section we describe how the PHoX system handles conditional statements involving correlated random variables.
Conditional statement cause the execution of a model to choose between different execution paths. In principle, a conditional statement creates two or more separate models from a single programmatic description. These models are identical up the point where the conditional statement is encountered, but differ afterward. Consider the conditional statement,

```c
if (X < Y) {
    execute path 1
} else {
    execute path 2
}
```

When $X$ and $Y$ represent real values then the statement is one of either-or, that is, exactly one program path is executed to the exclusion of the other. A key observation is that both program branches are executed. Each branch is assigned a value of probability. In the example above, branch 1 is assigned probability $P(X < Y)$ and branch 2 is assigned probability $1 - P(X < Y)$.

If $X$ and $Y$ are independent then we form the (rectangular) joint probability distribution and find the probability above the diagonal line shown as the shaded region in figure 6.20.

For each conditional statement the PHoX system must create a separate model instance. We will describe this in detail after the simple method section where this issue will become more significant.

Notice that,

$$P(X < Y) = P(0 < Y - X)$$

Since conditional statements assign probability to each execution path we must com-
Figure 6.20: Comparing Independent Random Variables

Compute probabilities such as $P(X < Y)$ which is the same as $P(0 < Y - X)$. If we let $Z = Y - X$ then we have created a new random variable $Z$ which we will test against zero. Since $X$ and $Y$ may be correlated we may use any of the techniques discussed thus far to compute the proper form of the new random variable $Z$. The simplifying feature is that instead of creating a partition for $Z$ containing a large number of points, we only need one at zero. That is, we will find $Z$ so that,

$$Z = \{(-\infty, 0, \infty), (p_{no}, p_{yes})\}$$

where we have included the endpoints at $\infty$ explicitly for emphasis since they are usually assumed. So we have,

$$P(X < Y) = P(0 < Z) = p_{yes}$$

$$P(X \geq Y) = p_{no}$$

208
There is a potentially positive impact on numerical performance of the calculation of the proper form of $Z$ because the partition has one point rather than the usual 1000 or so. Different techniques may be used in such a case that may increase performance dramatically.

Throughout this paper we assume that all random variables are piecewise continuous, but bear in mind that the PHoX system implements mixed random variables, that is, both discrete and continuous. This means it is possible to form non-trivial conditional statements such as,

\[
\text{if } (X = Y) \quad \{
\begin{array}{l}
\text{execute path 1} \\
\end{array}
\} \quad \text{else} \quad \{
\begin{array}{l}
\text{execute path 2} \\
\end{array}
\}
\]

since $P(X = Y)$ may be non-zero for mixed random variables.

6.1.6 The Model Compilation Step

The correlation section showed that expressions such as $A + (B \div (A + C))$ are irreducible and require a 3D joint probability distribution. On the other hand, expressions such as $A + (B + (A + C))$ are reducible. The parse tree for this addition expression is shown in figure 6.21.

where $X = A + C$, $Y = B + X$ and $Z = A + Y = A + (B + (A + C))$. From the model execution point of view we must compute $Z$ with a 3D joint probability. If, on the other hand, we were presented the equivalent expression, $A + (A + (B + C))$ the parse tree is shown in figure 6.22.

where $X = B + C$, $Y = A + X$ and $Z = A + Y = A + (A + (B + C))$ and the
Figure 6.21: Addition Expression

Figure 6.22: Alternate Addition Expression
changed directed edges have solid tips. The computational difference is that we compute \( Z \) with a 2D joint probability distribution because we first compute the proper form of \( Y \) also with a 2D joint probability distribution. This is represented by the directed edge indexing of aggregate \( jk \) as opposed to separate \( j, k \). This means that the proper form of \( Z \) is computed with a sequence of two 2D calculations as opposed to a single 3D computation. If we assume the 1000-point partition standard the computational cost is two one million rectangle computations as opposed to a one billion point computation. This is a significant computation and numerical difference.

To perform the addition parse tree transformation described above, and other similar transformations, we need a *compilation step*. During the compilation step, PHoX builds the parse tree for each expression in the model and makes all parse tree transformations it can recognize like the addition transformation described above.

In principle each variable in a PHoX model requires a separate parse tree, but in practice parse tree fragments are shared between expressions. Just as with the principle of lazy execution PHoX follows the principle of *lazy copy*. If, for example, a PHoX model contains the two expressions,

\[
\begin{align*}
X &= A + 2B \\
Y &= 3X + C \\
Z &= B / (A+X)
\end{align*}
\]

we see that the parse tree for \( Y \) contains the parse tree for \( X \) since \( Y \) depends on \( X \). Similarly the parse tree for \( Z \) contains the parse tree for \( X \). The issue is that \( Y \) may require the parse tree for \( X \) to be transformed, but the parse tree for \( Z \) requires the original form of the parse tree for \( X \). In such a case, the PHoX compilation step must recognize the contradiction and build two separate parse trees for \( X \), one for inclusion into \( Y \) and the other for \( Z \). This is accomplished by initially allowing \( Y \) and \( Z \) to share the parse tree
for \( X \). Once the PHoX system decides that \( Y \) requires a transformed version of the \( X \) parse tree, it makes a copy and transforms the copy leaving the original parse tree for \( X \) associated with \( Z \) as required.

The transformation of parse tree are a part of the compilation step of many computer languages such as the \( C, \ C++, \ Java \), etc. Parse tree transformation is standard compilation technique (Aho 2006 [1]).

A PHoX model must be compiled once each time the model description is updated. A PHoX model may be run many times with different inputs without requiring re-compilation. After a PHoX model is compiled it may then be run. At this point model input data is processed and results are generated. The time during this process is called \textit{run-time}.

In addition to transformation of parse trees to optimize computational and numerical performance, conditional statements are identified. The probability of each program path identified by a conditional statement is not known until run-time, but the existence of the alternate program paths are known to PHoX. Since some program branches will be assigned higher probability that others at run-time, PHoX needs an execution strategy for deciding in what order to follow the separate program paths.

The compilation step involves the \textit{structural analysis} of the given model. The PHoX system can then present the model developer or model user with the results of this structural analysis so that the PHoX model execution strategy may be influenced.

The naive model execution strategy is to follow the program path with the highest assigned probability. This strategy assumes implicitly that the most likely program path is the most important or most interesting since its results will appear before those of other program paths. The PHoX system does not presume the importance of results is directly related to their likelihood. That is, the PHoX system is agnostic concerning the importance of results.
6.1.7 Computation Technique for Higher Dimensions

Our goal in this section is to address the question of how many dimensions must be computed to process a typical model in the PHoX modeling system.

Characterizing a typical model is beyond the scope of this paper, but we can examine the existing California Carbon Reduction Market (AB32) model.

The AB32 model is broken into data processing modules turn input data for economic sectors into cost and quantity values for each time period covered by the model (2012-2020). The cost and quantity values are fed into a linear programming module that meets specified requirements at minimal cost.

As mentioned in the correlation section, the most complex expression in the data processing modules is (levelized) cost. We realize now that we may represented levelized cost for a single time period as,

\[
\text{Cost} = \frac{f(A, B)}{g(A, C)}
\]

where \( f, g \) are real-valued functions and \( A, B \) and \( C \) may not be first-class random variables, but are at least mutually independent.

In the AB32 model the cost and quantity must be determined for each time period. The data models are run once for each of two sets of input data. One input set is a recent estimate of data. One such estimate may be the current number of cement factories in California, for example. The other set of data comes from estimates of future data from some reliable source such as a published or commissioned report. The cost for any time period is then an affine combination of two levelized costs, \( \text{Cost}_1 \) and \( \text{Cost}_2 \) computed by PHoX. Thus we write,
\[ Cost(t) = \alpha(t)Cost_1 + (1 - \alpha(t))Cost_2 \]

\[ = \alpha(t) \frac{f(A,B)}{g(A,C)} + (1 - \alpha(t)) \frac{h(D,E)}{k(D,F)} \]

where \( \alpha(t) \) is some interpolation factor depending on the time period \( t \). In the PHoX AB32 model \( Cost_1 \) and \( Cost_2 \) are independent so a proper form random variable may be computed for each and their affine combination can be accomplished with the 2D method.

The AB32 model requires PHoX to use the simplex method for computing the linear programming cost minimization step. The simplex method requires the comparison of cost values between economic sectors and between time periods. In this way different emission reduction opportunities may be compared and the most opportune time period to seize the opportunity may be determined.

The levelized costs for different economic sectors are independent for the AB32 model. Each levelized cost is a random variable so the comparison of costs between economic sectors is the comparison of independent random variables. Comparing costs between time periods within an economic sector is the comparison of the following,

\[ Cost(t_1) = \alpha(t_1)Cost_1 + (1 - \alpha(t_1))Cost_2 \]
\[ Cost(t_2) = \alpha(t_2)Cost_1 + (1 - \alpha(t_2))Cost_2 \]

We must compute the new random variable \( X \) where,
\[ X = \text{Cost}(t_2) - \text{Cost}(t_1) \]

\[ = \alpha(t_2)\text{Cost}_1 + (1 - \alpha(t_2))\text{Cost}_2 - \alpha(t_1)\text{Cost}_1 - (1 - \alpha(t_1))\text{Cost}_2 \]

\[ = (\alpha(t_2) - \alpha(t_1))\text{Cost}_1 + (\alpha(t_1) - \alpha(t_2))\text{Cost}_2 \]

Since \( \text{Cost}_1 \) and \( \text{Cost}_2 \) are independent the calculation of \( X \) will be handled by the 2D method.

### 6.1.7.1 Beyond the AB32 Model

We now consider some augmentations to the AB32 model in order to explore the computational complexity that might be encountered with other, as yet undetermined, model.

We will begin by considering the model inputs. The AB32 model starts with observations of existing data. Let \( A \) represent such a value. Instead of accepting an outside estimate of a future version of \( A \) and claim independence, other models may choose to express the future version of \( A \) as \( f(A, \{B_1, ..., B_t\}; t) \) where \( t \) represents the time period, \( B_t \) is some stochastic variable representing drift and prediction uncertainty, \( f \) is some real-valued function.

The modification of introducing an independent random variable between each time period instance for every data input value may result in a computationally intractable problem. It’s also possible that each \( B_t \) introduced to represent statistical uncertainty is readily folded into other variables and the dimension of each subsequent expression is manageable. By folding we mean that a \( B_t \) random variable is combined with another random variable and neither is referenced separately so that the introduction of \( B_t \) has no effect of the dimension of subsequent expressions. This is the situation in the 2D reduction method.
6.1.8 Summary and Insights Gained

We have described a method for computing the random variable result for any function of correlated random variables. The fidelity of the result remains in question especially when the number of dimensions involved is large. We have also found that the number of dimensions involved in the calculation of any given expression may be far less than the number of independent random variables involved.

The correlation section detailed specific cases where the dimension of a random variable calculation reduces to a lower order. This section demonstrated a practical method for identifying the a reduced number of dimensions for each random variable expression.

We left open the question of determining the error bound for a random variable computation. There is also want for a statement of bounds on the number of dimensions that may appear in future business class models. Such a statement about computational complexity is beyond the scope of this paper. Our suspicion is that while a theoretic dimension bound may be quite high, the practical situation is likely to see very low dimension expressions and prove to be computationally tractable with acceptable fidelity of results.

As we found in the correlation section, a simply way to form a very high dimensional expression is with a continued fraction of many random variables with many repeats of random variables. Such expressions do not appear in business or business-finance texts we’ve encountered. The intuition developed while building the computational methods described in this section suggest that nearly all expressions in a model are of one or two dimensions and that expressions requiring three or four dimensions for computation represent a culmination of results and are therefore rare. Expressions that require several dimensions to compute seem to represent very interesting features of a calculation and are deserving of close scrutiny.

An insight gained in this section is that an interesting range of random variable correla-
tion is representable and computable in a practical sense. Real-world models such as AB32, enhanced with correlated uncertainty, are fully computable. This will be fully realized once the simplex method is presented in a later section.

6.2 Tables and Chairs, An Exemplary Example

We present here an example that is small enough to describe in full detail and incorporate many of the key features of algebraically correlated random variables described in this work. The example will be presented in two stages, first with sharp inputs replicating an example from Gass [24] and the second with algebraically correlated random variable inputs. We will then describe how the example may be generalized to encompass the much large reference example of the AB32 carbon emission abatement market.

The basis for the tables and chairs example can be found in Gass [24] wherein a decision must be made by a small furniture manufacturer under resource constraints. The choice is whether to manufacture tables or chair or some combination of both. The goal is to maximize the revenue from the sale of the tables and chairs assuming that all will be sold. The specifics are,

1. There is 400 board-feet of wood available.

2. There is 450 man-hours of labor available.

3. It takes 5 board-feet of wood and 10 man-hours of labor to make a chair.

4. It takes 20 board-feet of wood and 15 man-hours of labor to make a table.

5. Chairs sell for $45 each.

6. Tables sell for $80 each.
Stating the problem in standard form according to Boyd [5] and Greenberg [26] we have,

\[
\begin{align*}
\text{maximize} & \quad 45x_c + 80x_t \\
\text{s.t.} & \quad 5x_c + 20x_t \leq 400 \\
& \quad 10x_c + 15x_t \leq 450 \\
& \quad x_c, x_t \geq 0
\end{align*}
\]

where \( x_c \) represents the number of chairs to manufacture and sell and \( x_t \) represents the number of tables to manufacture and sell.

To form a baseline we will first solve this optimization problem using the simplex method as described in the simplex method section. We will then solve the problem again with the prices kept unknown. At that point we will be ready to introduce random pricing into the problem.

6.2.1 Tables and Chairs with All Inputs Sharp and Known

The simplex method requires all constraints to be stated as equalities so we introduce a slack variable into each inequality. The problem is restated as,

\[
\begin{align*}
\text{maximize} & \quad 45x_c + 80x_t \\
\text{s.t.} & \quad 5x_c + 20x_t + s_W = 400 \\
& \quad 10x_c + 15x_t + s_L \leq 450 \\
& \quad x_c, x_t, s_W, s_L \geq 0
\end{align*}
\]
where $s_W$ is the slack variable for the wood resource equation and $s_L$ is the slack variable for the labor resource equation. All variables, $x_c$, $x_t$, $s_W$ and $s_L$ are constrained to be non-negative.

Since each slack variable appears exclusively once in the constraint equations and their coefficients are +1 they collectively form a basis for the simplex tableau in table 6.1.

We collect the problem variables into the list $X = (x_c, x_t, s_W, s_L)$ and using the order of this list denote the variables in the current basis with a 1 and the others with a 0 we describe the current simplex state with the binary value,

$$State_0 = 0011$$

To pivot the table we find the variable to enter the basis and the basis variable to exit the basis. To find the entering variable we compute the cost impact of each,

$$Z_c = 45 - (5 \times 0 + 10 \times 0) = 45$$

$$Z_t = 80 - (20 \times 0 + 15 \times 0) = 80$$

where $Z_c$ is the cost impact of introducing variable $x_c$ into the basis and $Z_t$ is the cost impact for $x_t$. Recall that increasing $x_c$ the assumed zero value for a non-basis variable by one unit (one more chair sold, for example) will increase revenue by the price of one chair,
$45$, and will decrease the slack variables $s_W$ and $s_L$ by 5 and 10 units respectively. Since there is no revenue impact to increasing or decreasing slack variables the revenue impact is zero for each.

The entering variable is selected as,

$$\text{argmax}\{Z_c, Z_t\} \implies x_t$$

We now know that the next simplex state has the form $10??$ because $x_t$ is the entering variable and where the question marks indicate that we do not yet know the exiting variable.

Since $x_t$ is the entering variable we divide $b = (400, 450)$ element-wise by the basis coefficients for $x_t$ namely 20 and 15 and find the minimum non-negative value. In particular we find,

$$\text{argmin}\left\{ \frac{400}{20}, \frac{450}{15} \right\} \implies s_W$$

Since $\frac{400}{20} < \frac{450}{15}$ and the former value is associated with basis variable $s_W$ it is chosen as the exit variable. Recall that the reason is because we are allowing the entering variable $x_t$ to increase from zero it forces the basis variable in each equation toward zero. In the first equation a unit increase in $x_t$ is a 20 unit decrease in $s_W = 400$, but only a 15 unit decrease in $s_L = 450$. Since no variable is allowed to be negative the we find which basis variable is driven to zero first by an increase in the entering variable. We now see the new simplex state to be,
Table 6.2: Tables and Chairs Simplex Tableau for State 0101

\[
\begin{array}{cccc|c}

\text{State} & x_c & x_t & s_W & s_L & b \\

x_t & \frac{1}{3} & 1 & \frac{1}{20} & 0 & 20 \\

s_L & \frac{6}{3} & 0 & \frac{3}{4} & 1 & 150 \\

\text{Revenue} & 45 & 80 & 0 & 0 & \\

\end{array}
\]

\text{State}_1 = 0101

To transform the equations and update the tableau we form the transformation matrix to state 1, \(B_1\) and its inverse as,

\[
B_1 = \begin{pmatrix} 20 & 0 \\ 15 & 1 \end{pmatrix} \quad B_1^{-1} = \frac{1}{20} \begin{pmatrix} 1 & 0 \\ -15 & 20 \end{pmatrix}
\]

recognizing each tableau column as a vector and multiplying on the left by \(B_1^{-1}\) we find the new tableau in table 6.2.

The two non-basis variables are now \(x_c\) and \(s_W\) so we find the cost impact for introducing each into the basis,

\[
Z_c = 45 - \left( \frac{1}{4} \ast 80 + \frac{1}{4} \ast 0 \right) = 25 \\
Z_w = 0 - \left( \frac{1}{20} \ast 80 - \frac{3}{4} \ast 0 \right) = -4
\]

where \(Z_w\) is the cost impact of (re)-introducing \(s_W\) into the basis. Since \(Z_w\) is negative, \(s_W\) it is not eligible to be a basis vector leaving \(x_c\) as the only available choice for entering variable.
Dividing \( b \) by the vector of coefficients associated with \( x_c \) and finding the smallest non-negative value we have,

\[
\arg\min\left\{ \frac{20}{\frac{1}{4}}, \frac{150}{\frac{6}{4}} \right\} = \arg\min\{80, 24\} \implies s_L
\]

demonstrating the \( s_L \) is the exiting variable. The \( B_2 \) basis transformation matrix and its inverse become,

\[
B_2 = \begin{pmatrix}
\frac{1}{4} & 1 \\
6\frac{1}{4} & 0
\end{pmatrix} \quad \quad \quad B_2^{-1} = \begin{pmatrix}
0 & 0.16 \\
1 & -0.04
\end{pmatrix}
\]

The simplex state is then,

\[
State_2 = 1100
\]

and the tableau for state 1100 is shown in table 6.3.

We see that the two slack variables are no longer in the basis so they are both zero. This means that at the current state we are using all available resources to manufacture our tables and chairs. We ask if either of the two slack variables should be re-introduced into the basis by calculating the revenue impact for each,
Figure 6.23: Tables and Chairs Constraints and Optimal Point

\[ Z_W = 0 - (-0.12 \times 45 + 0.08 \times 80) = -1 \]

\[ Z_L = 0 - (0.16 \times 45 - 0.04 \times 80) = -4 \]

Since each cost impact is negative we conclude there is no possible way to improve the revenue of the problem and the algorithm terminates with the results,

\[ x_c = 24 \]
\[ x_t = 14 \]

revenue = \(24 \times 45 + 14 \times 80 = $2,200\)

since \((x_c, x_t) = b\). This means that the optimal choice for the manufacturer is to make 24 chairs and 14 tables which, when sold, will generate a revenue of $2,200.

Figure 6.23 shows the resource constraints (diagonal lines), the feasible region (shaded area) and the optimal point, (24, 14). The simplex method starts at the origin in this case.
and follows the heavy line in figure 6.23 from vertex to vertex of the polytope described by the half-space constraints to the optimal vertex. We notice that in this case there is an alternate vertex-path from the origin to the optimal vertex, namely passing through point \((45, 0)\). We will see that the choice of vertex-path is significant in the next version of this example when we leave the prices unknown.

### 6.2.2 Tables and Chairs with Unknown Prices

As an intermediate step we consider where uncertainty may be injected into the tables and chairs example and decide that leaving the prices sharp, but unknown leads to some revealing results.

A priori there are three places in the tables and chairs example where uncertainty may be injected; the constraint vector \(b\), the price vector \(p\) and the constraint matrix \(A\) where,

\[
A = \begin{pmatrix} 5 & 20 \\ 10 & 15 \end{pmatrix}
\]

\[
b = \begin{pmatrix} 400 \\ 450 \end{pmatrix}
\]

\[
p = \begin{pmatrix} 45 & 80 \end{pmatrix}
\]

We recognize the values in the \(A\) matrix as the amount of resources of each type consumed to manufacture each kind of product, \(b\) is the number of resources of each kind available and \(p\) is the prices charged for each product.

Suppose that instead of the 5 in matrix \(A\) we introduced a random variable. In the context of the tables and chairs example this means that the manufacturer is uncertain about the amount of wood necessary to construct a chair. We assume there is only one kind of
chair else we would likely call out the different kinds as different products and give them separate variables. We make similar statements about each value in the $A$ matrix.

Since we are most interested in reflecting the AB32 reference model into the tables and chairs example we elect not to introduce random variables into the $A$ matrix. The reason is that the corresponding $A$ matrix of the AB32 model represents policy features and physical limitations which are assumed for the given policy under consideration.

To choose to introduce random variables into the $b$ or $p$ vectors we must understand how they are used within the simplex algorithm. The simplex method uses a pivot-table approach whereby a column and related row within the simplex tableau are chosen and a transition is made to a new state within the algorithm.

To choose the simplex tableau column we compare revenue impacts given a choice of one of the non-basis variables. The values involved in computing the revenue impact of each non-basis variable are prices and products of prices and $A$ matrix coefficients from columns corresponding to the non-basis variables. If we assume that the $A$ matrix values are fixed then we are comparing linear combinations of $p$ vector prices to find the non-basis variable corresponding to the non-negative maximum of revenue impact values.

To choose the simplex tableau row given a column choice we compute the quotient of $b$ and the values in $A$ corresponding to that column. Thus we are comparing linear combinations of $b$ vector values to find the basis variable corresponding to the the non-negative minimum of linear combinations of constraint values.

We thus see that $p$ and $b$ values do not interact directly within the simplex method. We will then choose the $p$ vector for introduction of random variables suggesting, in the tables and chairs example, price uncertainty over the $b$ vector values which would suggest resource uncertainty. We will see that no new insight is gained though choosing $b$ over $p$ or through choosing both for random variable introduction. We will comment below on the choice of $A$ for random variable introduction especially if $p$ or $b$ are chosen as well.
For this intermediate tables and chairs example we have unknown, but sharp prices $p_c$ and $p_t$ for chairs and tables respectively. We make one assumption about these unknown prices; they are positive. The problem may then be stated in standard form as,

$$
\text{maximize} \quad p_c x_c + p_t x_t \\
\text{s.t.} \quad 5x_c + 20x_t \leq 450 \\
\quad \quad \quad \quad 10x_c + 15x_t \leq 450 \\
\quad \quad \quad \quad x_c, x_t \geq 0 \\
\quad \quad \quad \quad p_c, p_t > 0
$$

The initial simplex tableau is shown in table 6.4. The only differences from the initial tableau of the first example is the introduction of the state value 0011 into the upper-left corner and the unknown prices $p_c$ and $p_t$.

Following the steps from the first example we must find the entering non-basis variable by finding,

$$
\arg\max\{p_c - (5 \cdot 0 + 10 \cdot 0), p_t - (20 \cdot 0 + 15 \cdot 0)\} \\
= \arg\max\{p_c, p_t\}
$$
Table 6.5: Tables and Chairs Simplex Tableau for State 1010 with Unknown Prices

Since $p_c, p_t > 0$ neither case may be disqualified so we have some possibilities. Either $p_c < p_t$ or $p_c > p_t$ or $p_c = p_t$. Since we intend, in the next example, to introduce continuous random variables in place of $p_c$ and $p_t$, equality occurs with probability zero so we ignore that case here.

If $p_c > p_t$ then we choose $x_c$ as the entering variable. To find the exiting variable we compute,

$$\arg\min \left\{ \frac{400}{5}, \frac{450}{10} \right\}$$

$$\Rightarrow s_L$$

Since we have chosen $x_c$ as the entering variable and $s_L$ as the exiting variable our new state is 1010 and the transition matrix $B_{1010}$ and its inverse $B_{1010}^{-1}$ is,

$$B_{1010} = \begin{pmatrix} 5 & 1 \\ 10 & 0 \end{pmatrix} \quad B_{1010}^{-1} = \frac{1}{10} \begin{pmatrix} 0 & 1 \\ 10 & -5 \end{pmatrix}$$

The new 1010 tableau is shown in table 6.5.

The non-basis variables are $x_t$ and $s_L$ so we compute the revenue of (re)-introducing each of them, respectively, and find the entering variable.
argmax\{p_t - (1.5 \times p_c + 12.5 \times 0), 0 - (0.1 \times p_c - 0.5 \times 0)\} = \text{argmax}\{p_t - 1.5 \times p_c, -0.1 \times p_c\}

Since \(p_c > 0\) by assumption we have \(-0.1 \times p_c < 0\) so it must be disqualified as an option. We then ask, under what condition is the first option positive? That is,

\[
p_t - 1.5 \times p_c > 0
\]

\[
p_t > 1.5 \times p_c
\]

\[
\frac{2}{3}p_t > p_c
\]

Since we have already assumed upon entering this case that \(p_c > p_t\) it is not possible for \(\frac{2}{3}p_t > p_c\). We therefore terminate the simplex algorithm. Recalling that non-basis variables must be zero we find the following results,

\[
x_c = 45
\]

\[
x_t = 0
\]

\[
\text{revenue} = 45p_c \quad \text{for } p_t < p_c
\]

Returning to our first decision point we now assume \(p_c < p_t\) as was the case in the first example. The tableau under the assumption that we transition from state 0011 to state 0101 is shown in table 6.6 which we notice is similar to table 6.2 except for the unknown prices and the state being recorded in the upper left corner of the tableau.

From state 0101 the non-basis variables are \(x_c\) and \(s_W\) so we compute the revenue
maximizing variables by the usual methods,

\[
\text{argmax}\{p_c - \left(\frac{1}{4}p_t + 6\frac{1}{4} * 0\right), 0 - \left(\frac{1}{20}p_t - \frac{3}{4} * 0\right)\}
\]

\[= \text{argmax}\{p_c - \frac{1}{4}p_t, -\frac{1}{20}p_t\}\]

Since \(p_t > 0\) the second option of \(-\frac{1}{20}p_t < 0\) and is disqualified. From the first option we conclude that if \(p_c < \frac{1}{4} p_t\) that the simplex algorithm terminates. The results of this termination are,

\[x_c = 0\]
\[x_t = 20\]
\[\text{revenue} = 20p_t \quad \text{for } p_c < \frac{1}{4} p_t\]

For the simplex algorithm to not terminate at this part requires that \(p_c > \frac{1}{4}p_t\). Recall that we are already operating under the assumption that \(p_c < p_t\). Since these two assumptions are compatible (i.e. not impossible) we continue the simplex algorithm. From the calculation for entering variable we conclude that \(x_c\) is the entering variable and we must find the exiting variable. Since we have seen this exact situation in the first example we simply recall the result that \(s_L\) is the exiting variable and that the resulting tableau is shown in table 6.7. Because we are in the same state as before this new figure is nearly identical

<table>
<thead>
<tr>
<th>0101</th>
<th>(x_c)</th>
<th>(x_t)</th>
<th>(s_W)</th>
<th>(s_L)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_L)</td>
<td>(\frac{1}{4})</td>
<td>1</td>
<td>(\frac{1}{20})</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>(x_t)</td>
<td>(6\frac{1}{2})</td>
<td>0</td>
<td>(\frac{3}{4})</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td>Revenue</td>
<td>(p_c)</td>
<td>(p_t)</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6: Tables and Chairs Simplex Tableau for State 0101 with Unknown Prices
Table 6.7: Tables and Chairs Simplex Tableau for State 1100 with Unknown Prices

to the previous table 6.3.

In the first example, once we reached this state (1100) the simplex algorithm terminated. As before we attempt to find the entering variable with the calculation,

\[
\text{argmax}\{0 - (0.12p_c + 0.08p_t), 0 - (0.16p_c - 0.04p_t)\}
\]

\[
= \text{argmax}\{0.12p_c - 0.08p_t, 0.04p_t - 0.16p_c\}
\]

\[
= \text{argmax}\{3p_c - 2p_t, p_t - 4p_c\}
\]

For the second option to be positive and therefore available for consideration requires that \(p_t > 4p_c\) which is to say \(p_c < \frac{1}{4}p_t\). However, to reach this state we assumed \(p_c > \frac{1}{4}p_t\) so the second option is not available.

If \(p_c < \frac{2}{3}p_t\) then the simplex algorithm terminates just as it did in the first example since \(45 < \frac{2}{3} \times 80 = 53.33\ldots\). The result in this case is,

\[
x_c = 14
\]

\[
x_t = 24
\]

\[
\text{revenue} = 14p_c + 24p_t \quad \text{for} \quad \frac{1}{4}p_t < p_c < \frac{2}{3}p_t
\]

If \(\frac{2}{3}p_t < p_c\) then the first option for the entering variable is available and the entering variable is found to be \(s_W\).
The exiting variable in this case is then found as,

\[
\text{argmin}\{24 \div -0.12, 14 \div 0.08\}
\]

Since the first option is negative it is disqualified leaving the second option and therefore the second of the two basis variables \(x_t\) as the exiting variable.

Pivoting on the \(s_W\) column and the \(x_t\) row we have the transition matrix \(B_{1010}\) and its inverse \(B_{1010}^{-1}\) as,

\[
B_{1010} = \begin{pmatrix}
1 & -0.12 \\
0 & 0.08
\end{pmatrix} \quad B_{1010}^{-1} = \begin{pmatrix}
1 & 1.5 \\
0 & 125.5
\end{pmatrix}
\]

Applying our transition matrix \(B_{1010}^{-1}\) to the 1100 state tableau return us to the 1010 state tableau shown in figure 6.5. Since we have determined that state 1010 terminates we have the following results,

\[
x_c = 45 \\
x_t = 0
\]

\[\text{revenue} = 45p_c \quad \text{for } \frac{2}{3}p_t < p_c < p_t\]

Because we followed a different path between states to arrive at state 1010, the conditions for reaching this state are different. We notice that the conditions for reaching this state directly from the initial 0011 state are \(p_t < p_c\) do not intersect the conditions for reaching this state from state 1010 as we have just completed. We combine the two conditions
to see that the revenue outcome of $45p_c$ is reached if $\frac{2}{3}p_t < p_c$.

The result of our investigation is that there are three possible cases for any pair of positive prices given that all other values in the tables and chairs example remain the same, that is,

$$\text{Revenue} = \begin{cases} 
45p_c & \text{if } \frac{2}{3}p_t < p_c \\
24p_c + 14p_t & \text{if } \frac{1}{4}p_t < p_c < \frac{2}{3}p_t \\
20p_t & \text{if } p_c < \frac{1}{4}p_t
\end{cases}$$

The results of this example are summarized by the directed graph in figure 6.24. The nodes of the graph are the states of the simplex algorithm allied to this example, the edges are marked with the conditions under which the simplex algorithm will follow the edge and the three output cases are labeled $A$, $B$ and $C$ accompanied by the resulting revenue.

Looking ahead, suppose we were given random variables $P_t$ and $P_c$ representing the price of tables and chairs respectively. Even if $P_t$ and $P_c$ are correlated in some manner they have a joint distribution which we represent as the shaded region in figure 6.25. The figure (6.25) has three levels of shading with each region labeled with its outcome ($A$, $B$ or $C$) corresponding to the directed graph in figure 6.24. We note that if the two price random variables are independent then the shaded region representing the joint probability distribution of $P_t$ and $P_c$ would be rectangular.

### 6.2.3 Tables and Chairs with Correlated Random Prices

In this final stage of the tables and chairs example we introduce correlated random prices. We follow economic practice by developing a small story around the problem to tie the elements together.
Figure 6.24: Tables and Chairs Directed Graph

Figure 6.25: Tables and Chairs Partitioned Price Probability Space
Suppose that a small furniture manufacturer in Portland, Oregon wants to forecast weekly revenue. The manufacturer makes tables and chairs in a small show with a small crew. Using a forecast for demand for tables, chairs and dinette sets the manufacturer derives the likely market prices for tables and chairs. A dinette set is composed of one table and two chairs.

Figure 6.26 shows the independent random variables corresponding to forecast demand for dinette sets (the exponential curve), tables (the tall Gaussian curve) and chairs (the wide Chi-Squared curve). The vertical axis represents probability density and the horizontal axis represents demand for units (in thousands) in the Portland market.

The manufacturer believes that market price and demand for tables are related by the inverse function,
\[ P_t = \frac{14 \times 80}{D_t + D_d} \]

where \( D_t \) is the demand for tables alone and \( D_d \) is the demand for dinette sets. Thus \( D_t + D_d \) is the total demand for tables. Similarly, the price of chairs is related to the demand for chairs by the inverse function,

\[ P_c = \frac{24 \times 45}{D_c + 2D_d} \]

where \( D_c \) is the demand for chairs alone and again \( D_d \) is the demand for dinette sets. The sale of one dinette set implies the sale of two chairs. The actual functions are immaterial and have been contrived so that the results of this version of the tables and chairs example are comparable to previous versions.

We recognize that \( P_t \) and \( P_c \) are correlated, but do not need to materialize their joint probability distribution in order to compute revenue results.

The example is data-intensive so we create some prototype software to produce numerical results. Rather than presenting the prototype, written in Python using the Numpy library, we describe the data structures and sequence of operations.

Let our input random variables be,

\[ Dt = \{ DXt, DPt \} \]
\[ Dc = \{ DXc, DPc \} \]
\[ Dd = \{ DXd, DPd \} \]
where,

\[ DX_t = (DX_{t1}, \ldots, DX_{tN_t}) \]
\[ DP_t = (DP_{t1}, \ldots, DP_{tN_t}) \]
\[ DX_c = (DX_{c1}, \ldots, DX_{cN_c}) \]
\[ DP_c = (DP_{c1}, \ldots, DP_{cN_c}) \]
\[ DX_d = (DX_{d1}, \ldots, DX_{dN_d}) \]
\[ DP_d = (DP_{d1}, \ldots, DP_{dN_d}) \]

and we assume that \( DP_{tN_t} = DP_{cN_c} = DP_{dN_d} = 0 \) as usual for our numeric random variables since probability values are between partition values. We have \( N_t, N_c \) and \( N_d \) as the number of partition endpoints for each input random variable; tables, chairs and dinette sets respectively.

We form the demand joint probability distribution \( DP \) for the input random variables by Cartesian product,

\[ DP = DP_t \times DP_c \times DP_d \]

We separately form parallel 3D arrays for each input demand,
Figure 6.27: One Layer of Demand Probability Array

\[ DT = DXt \times \text{ones}(Nc) \times \text{ones}(Nd) \]

\[ DC = \text{ones}(Nt) \times DXc \times \text{ones}(Nd) \]

\[ DD = \text{ones}(Nt) \times \text{ones}(Nc) \times DXd \]

where, defining by example,

\[ \text{ones}(5) = (1, 1, 1, 1, 1) \]

Throughout this example presentation we will tend to use 2D diagrams to represent 3D objects for clarity. In figure 6.27 we represent the demand joint probability \( DP \) for a fixed value of \( DXd \).

In an abuse of notation, when no confusion arises, we use \( t, c \) and \( d \) as both identifiers and indices. We then have the indices for tables, chairs and dinette set random variables,
\[ t = 1 \ldots N_t \]
\[ c = 1 \ldots N_c \]
\[ d = 1 \ldots N_d \]

Now we can refer to a specific point within the demand joint probability array as \( DP_{t,c,d} \) or simply \( DP_{tcd} \) where the commas are dropped for clarity. The shaded rectangle in figure 6.27 is then the demand-space rectangular block of uniform probability distribution with value \( DP_{tcd} \).

The four 3D arrays \( DT, DC, DD \) and \( DP \) are all parallel. We will now create other arrays parallel to these. The reason for this parallelism is to ensure that the probability within each block is correctly tracked through each step of the computation process.

The 3D arrays for the prices of tables and chairs are then written,

\[
PT = \frac{14 \times 80}{DT + DD} \\
PC = \frac{24 \times 45}{DC + 2DD}
\]

where the sums, \( DT + DD \) and \( DC + 2DD \), are computed element-wise as well as the reciprocal functions. The result is that \( PT \) and \( PC \) are 3D arrays of size \( N_t \times N_c \times N_d \) and are parallel to the demand and demand probability arrays. Note in particular that the price arrays \( PT \) and \( PC \) are not random variables. We will describe below how they may be converted to random variable form. This is possible because we have values of prices as vertices of a block of probability distribution and we have the value of the probability uniformly distributed within that block in the 3D array \( DP \).
Now that we have our input prices the manufacturer may apply their optimization and
decide what combination of tables and chairs to produce. In this example we have already
computed the optimization exhaustively and can therefore partition our demand space into
the three output cases A, B and C.

Consider that each point in the 3D demand array represents a particular choice of input
values that has associated with it two particular prices, one for tables and the other for
chairs. We have already determined the rule for choosing each output case. We know,
for example, that if \( \frac{2}{3} Pt < Pc \) then output A will be selected and similar rules apply for
outputs B and C. This means for each point in our 3D demand we can assign a Boolean
value 1 or 0 where 1 means that point has an associated price for chairs that is larger then
two-thirds that of tables. We can thus create a parallel 3D array of Boolean values called a
mask based on the optimized output selection rules. Let,

\[
MA = \frac{2}{3} PT < PC
\]
\[
MB = \frac{1}{4} PT < PC < \frac{2}{3} PT
\]
\[
MC = PC < \frac{1}{4} PT
\]

Each output is associated with some revenue. Using the price arrays we can form
parallel revenue arrays. Let,

\[
RA = 45 PC
\]
\[
RB = 24 PC + 14 PT
\]
\[
RC = 20 PT
\]
To convert the revenue arrays, $RA$, $RB$ and $RC$ into random variables we must first find a partition. We notice that while the revenue arrays are parallel, using the masks we see that any given point in the array space indexed by $(t, c, d)$ is intended to be present in exactly one revenue array. This is because the output $A$, $B$ and $C$ are mutually exclusive so, for example, the probability of producing $1000$ and $2000$ of revenue using output $A$ can be added to the probability of producing this same range of revenue for output $B$ and for $C$ to arrive at a probability of producing that range of revenue regardless of output choice.

We would like the partition we use for the revenue random variable we are about to produce to span the range of possible revenue values, be fine where there is more revenue information and course where there is less and be so fine overall that numerical artifacts overwhelm the result. We choose for this example to use every $23^rd$ point from each 175-point input demand random variable and rerun the problem on the partition values alone, not the probability values. In the Python code this amounts to a single function call since all the code is in place for the main computation. The result is are smaller versions over the same revenue arrays representing collectively a sample of the possible revenue values this example model produces using the given demand inputs. The steps are as follows,

1. Form one dimensional arrays of valid revenue values for each output.

2. Run the same process as above to generate revenue arrays and output masks. Prepend an $s$ to the name indicating they are small versions due to the reduced partition size.

3. Concatenate the three 1D arrays into a single array called $temp$.

4. Sort the temporary array and remove any duplicates.

5. append the value $-\infty$ to the start of the array and $\infty$ to the end. Call the result $Rx$.

In this case the Python code from the prototype sums up the process concisely,
\[ temp = \text{concatenate}(\{sRA[sMA], sRB[sMB], sRC[sMC]\}) \]

\[ Rx = \text{concatenate}(\{[-\infty], \text{unique}(temp), [+\infty]\}) \]

where \( sRA[sMA] \) returns a one dimensional array from an arbitrary array only for points where the corresponding point in the \( sMA \) small output mask array is a 1 and \text{unique()} sorts and removes duplicates from an array.

For each (big) output array \( RA, RB \) and \( RC \) with associated masks \( MA, MB \) and \( MC \) we create a one dimensional array for the probability distribution that is parallel to the one dimensional partition array \( Rx \).

\[ Rap = \text{zeros}(Rx) \]
\[ Rbp = \text{zeros}(Rx) \]
\[ Rcp = \text{zeros}(Rx) \]

where, defining by example,

\[ \text{zeros}(5) = (0, 0, 0, 0, 0) \]

The probability arrays, once filled in, will complete the formation of the output revenue random variables,
\[ Ra = \{ Rx, Rap \} \]
\[ Rb = \{ Rx, Rbp \} \]
\[ Rc = \{ Rx, Rcp \} \]

The three output random variables \( Ra, Rb \) and \( Rc \) are mutually exclusive and since they share a common partition we can add their probability values to find the final output revenue random variable \( R \),

\[ R = \{ Rx, Rap + Rbp + Rcp \} \]

It remains to describe how to fill in the probability arrays \( Rap, Rbp \) and \( Rcp \). We will describe the process for \( Rap \) since it is the same for the others.

Given the (big) output revenue 3D array \( RA \), its associated mask \( MA \), the associated 3D probability array \( DP \) and the 1D revenue partition \( Rx \) we proceed as follows to fill in the zero-valued 1D probability array \( Rap \).

The output revenue 3D array \( RA \) together with the associated probability array \( DP \) describes a partition of the joint demand (input) space into blocks. Recall that we index the blocks with indices \( t, c \) and \( d \) so that the \((t, c, d)\) block has uniform probability \( DP_{tcd} \) and eight vertices with the following revenues,
for some block such that $1 \leq t < N_t$, $1 \leq c < N_c$ and $1 \leq d < N_d$. If all the vertices are valid, that is, the associated mask value is 1 for each vertex then figure 6.28 symbolically represents one possible scenario.

The limits of the 3D block projection are the minimum and maximum revenue vertex values. That is,

$$\text{min}_{tcd} = \text{Min}(RA_{t,c,d}, \ldots, RA_{t+1,c+1,d+1})$$

$$\text{max}_{tcd} = \text{Max}(RA_{t,c,d}, \ldots, RA_{t+1,c+1,d+1})$$

For the software prototype version of this example we make the assumption that the 3D block probability $DP_{t,c,d}$ is distributed uniformly over the revenue line segment $(\text{min}, \text{max})$ so that the density is $h_{tcd}$.
Figure 6.29: Partition Allocation of Probability Line

\[ h_{ted} = \frac{DP_{ted}}{max_{ted} - min_{ted}} \]

where we have assumed that \( min_{ted} < max_{ted} \). We will address special cases such as when \( min_{ted} \) and \( max_{ted} \) are equal below. Continuing with the general case we now allocate the uniform probability density \( h_{ted} \) to the revenue probability array \( Rap \) recalling that \( Rap \) is delimited by the partition array \( Rx \). Referring to figure 6.29 we have,

\[
Rap_i = Rap_i + (Rx_{i+1} - min_{ted}) h_{ted}
\]

\[
\ldots
\]

\[
Rap_{i+3} = Rap_{i+3} + (Rx_{i+4} - Rx_{i+3}) h_{ted}
\]

\[
Rap_{i+4} = Rap_{i+4} + (max_{ted} - Rx_{i+4}) h_{ted}
\]

where we have indicated how to compute end cases as well as cases where \( Rx \) partitions are spanned by the \( (min_{ted}, max_{ted}) \) interval.

If the mask \( MA \) indicates that some of the vertices of the \( (t, c, d) \) block are not valid then we must reduce the amount of block probability \( DP_{ted} \) available for allocation. For
example, if 3 of 8 vertices are valid for the \((t, c, d)\) block then the block probability is correspondingly reduced to \(\frac{3}{8}DP_{tcd}\) so that the \((min_{tcd}, max_{tcd})\) interval probability density is,

\[
h_{tcd} = \frac{3}{8} \frac{DP_{tcd}}{max_{tcd} - max_{tcd}}
\]

If it happened that \(min_{tcd} = max_{tcd}\) either because all the valid vertices have the same revenue value or there is only one valid vertex for the \((t, c, d)\) block then the corresponding partition element is located for the \(Rap\) array and its value is incremented with the available probability for that block. If it happens that \(min_{tcd} = max_{tcd}\) equals an \(Rx\) partition endpoint then the available block probability is halved and allocated to the adjacent partition intervals.

We perform the above operations for each output case and combine them into the full revenue random variable and show the result in figure 6.30. The horizontal axis is dollars of revenue and vertical axis is probability density as usual for random variable graphs. We notice that the median value is roughly $2200 because of careful choice of demand inputs and the demand-to-price functional relationship.

Notable features of the optimized revenue in figure 6.30 is that no matter what happens with the projected demand there is a non-zero minimum revenue (about $300), a strongly likelihood of earning about $2200 and significant possibility of earning considerably more than the median $2200.

We use the machinery developed above to convert the 3D price arrays to random variables. Since there is no optimization involved in computing prices there is no need to generate masks. Since we have some information about the range of prices to expect we choose price partitions directly. In this case each price is partitioned into regularly spaced
intervals from 0 to $150. The 3D probability arrays are projected onto the price partitions and the result for each price random variable is shown in figure 6.31. Again we notice that, by design, the price of chairs is has a median price of about $45 and that of tables is about $80 which corresponds with the sharp version of the tables and chairs example.

Notice that the price random variables are marginal probability distributions for a joint probability distribution we have not computed. Since the price for tables and chairs are non-trivially correlated the joint distribution cannot be recovered from the marginal distributions alone as described in a standard statistics textbook such as [4].

6.2.4 Finding the Joint Price Distribution from the Demand Inputs

The reader will notice that we developed the tables and chairs example with unknown prices in preparation for the introduction of random inputs resulting in correlated prices we described how to proceed with a joint distribution for the two prices, tables and chairs. Then
we solved the problem without using, or even finding, the joint price distribution. Instead we used the 3D array created to represent the three demand inputs. For this problem this technique provides directed and satisfactory results.

In this section we revisit the tables and chairs example with the same three demand input, but this time produce the joint price probability distribution. Since so much of this work is devoted to the study of correlated random variables we would be remiss not to include at least one example of same.

We restart the problem with our three 3D demand arrays, $DT$, $DC$ and $DD$. We also have the associated 3D probability array $DP$. Using the same formulas as before for finding the (correlated) prices of tables and chairs we produce the two 3D price arrays $PT$ and $PC$ respectively.

This brings us to the point where we projected each 3D price array onto a price partition and produced random variable representations of the two prices in figure 6.31. We choose

Figure 6.31: Random Variable Table and Chair Prices
the same price partitions as before, evenly space intervals from $0 to $150. This choice allows us to compare the results we are about to obtain with those obtained previously.

Our two price partitions, for $PT$ and $PC$, describe a 2D partition of the $(Pc, Pt)$-space. If the number of points in each price partition is $Np$ then we create a 2D array of size $Np^2$ and initialize it with zero values.

We then realize that our two 3D price arrays $PT$ and $PC$ together describe a 3D lattice of pairs of prices at each vertex surrounding a uniform distribution of probability described by the 3D probability array $DP$. The 8 vertices of each probability block, each containing the two price values, are projected onto the two dimensional $(Pc, Pt)$-space. This the 2D analog of our 1D procedure for finding each marginal price random variable by projecting each price block for $PT$ or $PC$ onto the corresponding one dimensional price line. Figure 6.32 shows an example of price block vertex projection.

Since the cluster of vertex projections in figure 6.32 indicate the projection of the 3D price probability block onto the 2D joint price partition we must allocate the block probability accordingly.

Assuming that the cluster of vertex projections represent the limits of the block projection we can find the convex hull of these points using an algorithm such as the Graham
Scan as described in a textbook of computer algorithms such as Corman [16]. We then assume the block probability is distributed uniformly over the interior region of the convex hull and apportion it accordingly to the partition rectangles of the 2D price distribution, called $JP$. Figure 6.33 shows the convex hull of the projected vertices. The heavy outline of joint price rectangles shows the limits of affected rectangles. Let $p$ be the probability of the projected block and $a$ the area of the convex region, then $h = p/a$ is the probability density. The portion of probability allocated to any given rectangle in the outlined region is $h$ times the area of the rectangle intersecting the convex region.

In our running tables and chairs example we have over 5 million blocks to project so we opt not to engage in a complex computation of multiple rectangle intersections with convex regions associated with each block, at least not for our prototype code. Instead we take a simpler approach as shown in figure 6.34. The heavy outline bounding box represents the $Pt$ and $Pc$ partition limits bounding the block vertex cluster. The shaded inner rectangle represents the rectangular limits of the cluster points. We calculate the probability density of the block probability if distributed uniformly over the inner shaded rectangle and distribute this by area over each intersecting price rectangle.

The results of the calculations of our prototype code for the joint probability distribution
of the two correlated prices is shown in figure 6.35. Be aware that the origin is located in the upper-left corner of the graph. The \( x \) and \( y \) axis are prices of tables and chairs respectively and the vertical axis is probability density.

An top view of the joint probability price distribution is shown figure 6.36. We compare this figure to our original suggestion of the joint probability distribution of prices in figure 6.25.

It is interesting to note that the marginal random variable prices we compute with the prototype code are identical to the random variable price we computed previously and show in figure 6.31.

With the joint distribution of prices in hand we are able to address the question of how to compute the probability of each branch in the simplex graph (see figure 6.24). In particular we are interested in the probabilities of the branch conditional expressions,
Figure 6.35: Joint Probability Distribution of Table and Chair Prices

Figure 6.36: Joint Probability Distribution of Table and Chair Prices (Top View)
\[
\begin{align*}
&pt < pc \\
&\frac{2}{3}pt < pc \\
&\frac{2}{3}pt < pc < pt \\
&\frac{1}{4}pt < pc \\
&\frac{1}{4}pt < pc < \frac{2}{3}pt
\end{align*}
\]

Proceeding as we did from the beginning, but starting with the jointly distributed prices and their partitions \(P_t\) and \(P_c\) we create 2D arrays \(PT_2\) and \(PT_2\) that are parallel to the 2D joint probability distribution array \(JP\). Taking the last inequality above as an example we divide each expression by \(p_t\) to find,

\[
\frac{1}{4} < \frac{pc}{pt} < \frac{2}{3}
\]

In the prototype we form the 2D array expression \(Qtc\) as,

\[
Qtc = \frac{PC_2}{PT_2}
\]

as the element-wise quotient of the two 2D price arrays. We notice that \(Qtc\) together with the \(JP\) form an improper form two-dimensional random variable, a non-standard usage of the expression. If we choose a 1D partition for \(Qtc\) we can project our \(\{Qtc, JP\}\) pair onto this partition and find a proper-form random variable, called \(qtc\). As we have mentioned earlier, we address this case by choosing the special partition,
\[ X_{qtc} = (-\infty, 0, \frac{1}{4}, \frac{2}{3}, 1, \infty) \]

The result is,

\[ P_{qtc} = (0, 0.0001012, 0.7206, 0.2388, 0.03024, 0) \]

Combining these into the random variable \( Q \) for convenience as,

\[ Q = \{ X_{qtc}, P_{qtc} \} \]

These probability values tell us probability of each simplex directed graph branch and therefore the probability of each result. For example, referring to figure 6.24, the probability of taking the first left directed edge under the condition that \( p_t < p_c \) is \( \mathbb{P}(1 < Q) = 0.03024 \). Similarly the probability of reaching result \( B \) is \( \mathbb{P}(\frac{1}{3} < Q < \frac{2}{3}) = 0.7206 \).

6.2.5 Tables and Chairs with Unknowns Prices and Resources

If we allow prices and resources to be described by correlated random variables the impact on the example is to increase the number of branches from each simplex algorithm states and an increase in the number of states. The simplex tableau for unknown prices and resources is shown in table 6.8.

The directed graph for the tables and chairs example with unknown prices and resources is shown in figure 6.37. Notice that there are only \( C(4, 2) = 6 \) possible node states in this example. Notice also that there are 5 possible terminal states; manufacture of only tables
or only chair limited by either wood resource or labor resource and also the mixed case.

While the conditions present when entering a state are significant, the tableau in each state is denumerable as. We refer to tables 6.9, 6.10, 6.11, 6.12 and 6.13.

Starting in state 0011 we follow the simplex two-phase decision and first compare the two prices $p_c$ and $p_t$. We are assuming for clarity, as before, that since we intend to replace $p_c$ and $p_t$ with continuous random variables the probability of equality is zero. In practice we need to check for the possibility of equality. The initial comparisons are,

\[
\arg\max(p_c, p_t)
\]

\[
\arg\min\left(\frac{b_W}{5}, \frac{b_L}{10}\right) > 0 \text{ and } p_c > p_t
\]

\[
\arg\min\left(\frac{b_W}{20}, \frac{b_L}{15}\right) > 0 \text{ and } p_c < p_t
\]

\[
\begin{array}{cccccc}
\text{0011} & x_c & x_t & s_W & s_L & b \\
\hline
s_W & 5 & 20 & 1 & 0 & b_W \\
s_L & 10 & 15 & 0 & 1 & b_L \\
Revenue & p_c & p_t & 0 & 0 & \\
\end{array}
\]

Table 6.8: Tables and Chairs Simplex Tableau for Unknown Prices and Resources

\[
\begin{array}{cccccc}
\text{1001} & x_c & x_t & s_W & s_L & b \\
\hline
s_W & 1 & 4 & \frac{5}{5} & 0 & \frac{b_W}{5} \\
s_L & 0 & -25 & -2 & 1 & b_L - 2b_W \\
Revenue & p_c & p_t & 0 & 0 & \\
\end{array}
\]

Table 6.9: Tableau for Unknown Prices and Resources, State 1001

\[
\begin{array}{cccccc}
\text{1010} & x_c & x_t & s_W & s_L & b \\
\hline
s_W & 1 & \frac{3}{2} & 0 & \frac{1}{10} & \frac{b_W}{10} \\
s_L & 0 & -25 & 1 & -\frac{1}{2} & b_W - \frac{b_L}{2} \\
Revenue & p_c & p_t & 0 & 0 & \\
\end{array}
\]

Table 6.10: Tableau for Unknown Prices and Resources, State 1010

254
Figure 6.37: Directed Graph for Tables and Chairs with Unknown Prices and Resources

<table>
<thead>
<tr>
<th>State</th>
<th>( x_c )</th>
<th>( x_t )</th>
<th>( s_W )</th>
<th>( s_L )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>( \frac{1}{2} )</td>
<td>( 0 )</td>
<td>( \frac{3}{4} )</td>
<td>( 0 )</td>
<td>( \frac{b_W}{25} ) + ( b_L )</td>
</tr>
<tr>
<td>Revenue</td>
<td>( p_c )</td>
<td>( p_t )</td>
<td>0</td>
<td>0</td>
<td>( b_W - \frac{4}{3} b_L )</td>
</tr>
</tbody>
</table>

Table 6.11: Tableau for Unknown Prices and Resources, State 0101

<table>
<thead>
<tr>
<th>State</th>
<th>( x_c )</th>
<th>( x_t )</th>
<th>( s_W )</th>
<th>( s_L )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>( \frac{2}{3} )</td>
<td>( 0 )</td>
<td>( \frac{3}{2} )</td>
<td>( 0 )</td>
<td>( \frac{b_L}{15} )</td>
</tr>
<tr>
<td>Revenue</td>
<td>( p_c )</td>
<td>( p_t )</td>
<td>0</td>
<td>0</td>
<td>( b_W - \frac{4}{3} b_L )</td>
</tr>
</tbody>
</table>

Table 6.12: Tableau for Unknown Prices and Resources, State 0110
where the \( > 0 \) condition refers to the requirement that each operand be positive else it is disqualified from the comparison.

In many cases the first or second phase of the simplex algorithm decision is disqualified since it is either non-positive or contradicts a previous assumption.

Since simplex states may be re-entered a computer algorithm can take advantage of this possibility and cache, rather than recompute, certain elements such as the simplex tableau. At each decision point we see again that we are comparing linear combinations of either price or resource variables with zero in the sense that the expression \( a < b \) can be rewritten as \( 0 < b - a \). We have seen in the previous version of the tables and chairs example that each conditional statement results in a filter on the input space so that the simplex algorithm may be viewed as a filtration process. The task of the PHoX modeling system is to determine what portion of the input space passes through each facet of the simplex filtration process to a terminal node and with what probability.

6.3 Beyond the Tables and Chairs Example

In the tables and chairs example we chose correlated random inputs for prices and argued that we could have chosen correlated random inputs for resources instead. We notice in the simplex algorithm the transition decision from one state to the other involves computing the maximum positive revenue impact in the case of prices and then the minimum positive resource impact. These two choices correspond to the two facets of a table pivot as explained above.

<table>
<thead>
<tr>
<th>1100</th>
<th>( x_c )</th>
<th>( x_t )</th>
<th>( s_W )</th>
<th>( s_L )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_W )</td>
<td>1</td>
<td>0</td>
<td>( -\frac{3}{25} )</td>
<td>( \frac{4}{25} )</td>
<td>( \frac{4b_L - 3b_W}{25} )</td>
</tr>
<tr>
<td>( s_L )</td>
<td>0</td>
<td>1</td>
<td>( \frac{1}{25} )</td>
<td>( -\frac{1}{25} )</td>
<td>( \frac{2b_W - b_L}{25} )</td>
</tr>
<tr>
<td>Revenue</td>
<td>( p_c )</td>
<td>( p_t )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.13: Tableau for Unknown Prices and Resources, State 1100
It remains to be investigated what happens to this example when some or all of the values of $A$ are unknown. In this example the significance of unknown $A$ values is that the manufacturer is unsure how many resources are consumed by each product.

As explained by Bellman [3], the number of solution states, not to mention the number of internal simplex algorithm states, becomes computationally intractable even for modest problems. We see this for ourselves if the problem has 100 variables and 100 constraints then the number of simplex states is at least $C \approx (200, 100) = 9.05 \times 10^{58}$. The reference implementation of the AB32 model has many hundreds of variables and several thousand constraint equations.

A way to proceed is for the PHoX modeling system to partially explore the simplex directed graph. The transition from one state to the next using the simplex algorithm involves finding the maximum of a set of linear combinations of prices, in the context of the tables and chairs example, followed by finding the minimum of a set of linear combinations of resource limits assuming the $A$ values are fixed. As we have seem each choice element is a linear combination of random variables which are themselves random variables. If we assume there are three choice elements denoted $X$, $Y$ and $Z$ then the decision,

$$\text{argmax}\{X, Y, Z\}$$

results in three probability values,

$$\mathbb{P}(X < \{Y, Z\})$$
$$\mathbb{P}(Y < \{X, Z\})$$
$$\mathbb{P}(Z < \{X, Y\})$$
In this case the simplex algorithm state has three initial branches corresponding to the first decision of the pivot element. These probabilities may be computed explicitly and rather than create a directed graph with all choices listed as we did above with the tables and chairs directed graph, we choose only the most highest probability transition. In this manner we reach a terminal node as in the sharp version of the simplex algorithm.

Since we are able to assign probability values to each transition edge in the directed graph we may apply a choice algorithm to explore other paths based on their likelihood of occurrence. As long as the directed graph remains at least partly unexplored, we suspect this is the case in general, then the random variable results will not have full probability, that is, their probability values will sum to less than one. The proximity of the probability sum of a random variable result to unity can be used as a criterion for algorithm termination. That is, if our random variable result is deemed near enough to completion the PHoX algorithm can terminate its exploration of the simplex directed graph.
CHAPTER 7

Conclusion

7.1 Problems Solved

We have shown that for a range of numerical model types we call *business class* models that model input uncertainty expressed as independent random variables requires the numerical computation of correlated random variables. We have shown how it may be possible to incorporate optimization algorithms into a business class model and compute not only random variable results, but numerically partition the input space into regions associated with a given optimization result or set of results.

In this work we found an apparently unexplored avenue for addressing uncertainty and sensitivity of models. It appears to replace the numerical brute force of Monte Carlo methods with manageable computational complexity and the assurance of problem space coverage.

7.2 Problems Identified

We found that models involving random variables and optimization routines rapidly become computationally unwieldy. We presented several algorithms for addressing the computational challenges. A question that arises is that of error bound. Partitioning random variables into separate regions and assuming the probability distribution is uniformly
distributed within each region introduces some amount of error. When we go further and combine partitioned random variables with each other and compute functions of random variables we introduce more error. There are many different techniques for processing numerically defined random variables in this work and each is in want of a certificate of error bound.

Further investigation is needed into the compatibility of the techniques for numerically processing correlated random variables and optimization algorithms beyond the simplex method. According to Boyd [5] there are other algorithms for computing linear optimizations and many other techniques such as interior point methods for computing non-linear optimizations.

Compiler techniques for symbolically identifying algebraically reducible expressions need to be developed. For example, a model compiler for PHoX must notice that an expression such as $A + A$ should be rewritten as $2A$ to avoid treating the two different copies of the random variable $A$ as independent. This issue is detailed above.

7.3 Lessons Learned

This work began with the notion that developing software to express theoretical ideas has synergistic effects. What we learned is that the action of encoding the different facets of the random variable modeling problem into software revealed viewpoints otherwise unimagined. To put it colloquially, we simply wouldn’t have looked at the problem *that* way. Writing the several versions of the PHoX system in Java and the simplex prototype example in Python provides a workspace to investigate the problem and a framework upon which to build a deeper understanding about the problem.

When we wrote the tables and chairs example in prototype form we realized that the whole problem could be centered around a 3D array. Not a particular 3D array, but the
idea of a common lattice of points surrounding blocks of probability distribution. By being able to not only envision the problem in this concrete manner led to the realization that correlated random variables may be expressible more cleanly in higher dimension than their natural joint probability space. An added bonus is to realize that the 3D array used in the tables and chair prototype need never be expressed in the software itself, but referred to as if it were expressed. This leads to the further realization that by starting with a course 3D partition and only refining conditionally that input space boundaries and regions can be identified with optimization output choices. This in turn leads to the idea of software assisting the user while investigating important output cases.

7.4 Future Directions

The vision of the author for this work is to upgrade the existing PHoX modeling system to include model compilation and correlated random variables. Further compiler work will be needed to allow software models written in many software languages to be retrofitted into the PHoX modeling system for analysis. The PHoX modeling system would then provide the model operator (a person) with a control panel and viewing port into their software model. The model operator then allows a subset of the model input to be uncertain, fixing the others to nominal values. The PHoX modeling system would then allow the operator to view any intermediate variable within the system as a random variable, if appropriate, or any pair of random variables (correlated or not). Furthermore, model output cases identified by the model operator can be traced by the PHoX modeling system to subsets of the input space for more careful study.
REFERENCES


[47] S. Serban Scrieciu. The inherent dangers of using computable general equilibrium models as a single integrated modelling framework for sustainability impact assess-


