Making the Case for Using Analytica® for System Dynamics Modeling: A Reference Guide and Comparison with Classical Platforms

Cory Welch [a]

[a] Navigant Consulting, Inc., cory.welch@navigant.com, welch.cory@gmail.com

Abstract

This paper serves as a reference guide for practitioners or theorists who wish to develop System Dynamics (SD) models that are multi-dimensional, integrate System Dynamics with optimization (linear or nonlinear) methods, or require a sophisticated treatment of uncertainty. The paper presents the Analytica®[1] software platform because its flexibility, supported by its array abstraction, combined with its powerful optimization and uncertainty analysis capability, make it more convenient for these applications than classical SD software platforms. I discuss the similarities, differences, advantages, and disadvantages of using Analytica and classical SD software for System Dynamics modeling. I describe how, in some applications, Analytica provides capabilities not currently present in classical SD software; in other applications, complex models can be developed with less effort and greater flexibility. Finally, I offer a generalized derivation of how to initialize an aging chain under conditions of historic growth, including an example of its implementation using array abstraction in Analytica.

Keywords: system dynamics, optimization, uncertainty, multi-method modeling, Analytica, delay function, aging chain, stock initialization.

Introduction

Though many software platforms have been used over the years to develop System Dynamics (SD) models, Vensim[2] and Stella[3] stand out as what I refer to in this paper as the classical SD platforms. Their ease of use, visual modeling interface, declarative language, and prominence in education make them the tools of choice for many practitioners and theorists. Other tools (e.g., AnyLogic, R, Python, Matlab) offer some advantages relative to classic SD platforms, but tend to be less user-friendly and come with additional overhead of required programming skills.

Ten years ago while working at the National Renewable Energy Laboratory (NREL), I was introduced to the Analytica modeling platform when it was selected by NREL, after an extensive software search, as the application most suited to development of a new stochastic, dynamic, energy-economy model (NREL 2016). After realizing that Analytica's modeling environment shared many attributes with classical SD platforms (e.g., visual icon-based modeling interface, declarative language, ease of use), but also offered many advantages when dealing with multi-dimensional and stochastic analyses, I began using it for all my modeling. In the ten years since, I have used Analytica in over thirty modeling projects, about half of which have applied System Dynamics concepts. Clients have included large electric and gas utilities, utility regulatory agencies, non-profit conservation agencies, the U.S Department of Energy, product manufacturers, and national laboratories.

[1] www.lumina.com
[2] www.vensim.com
[3] www.iseesystems.com

After a decade of modeling in Analytica, I hope to introduce others to its advantages and relevance for System Dynamics. In some applications, it permits more sophisticated analyses than classical SD platforms are currently capable of, yet with a similar level of user-friendliness and knowledge required to formulate models. In other applications, it enables building complex models with substantially less effort and with greater flexibility than in classical SD platforms, as I will demonstrate in this paper. I suggest that Analytica can complement rather than replace classical SD software tools, which are well-suited for many but not all applications.

Constructing a System Dynamics Model in Analytica

The primary difference between Analytica and classical SD software, when developing a dynamic model, lies with how a stock is constructed. In classical SD software, the underlying software architecture automatically integrates all inflows and outflows for each stock variable. Stock variables have a different input structure and are signified by a distinct icon, typically a box. In contrast in Analytica, stock and flow variables do not have distinct structures or icons, but rather require the modeler to distinguish between the two in the equation itself. The modeler can change the color of stock and flow "nodes" to facilitate ease of recognition, as shown below (where white boxes depict stocks and green boxes depict flows). Straight black unidirectional arrows between nodes signify a dependency relationship (similar to curved arrows in Vensim), and gray arrows (as exist between Births and Population below) signify the presence of a dynamic dependency between two variables.

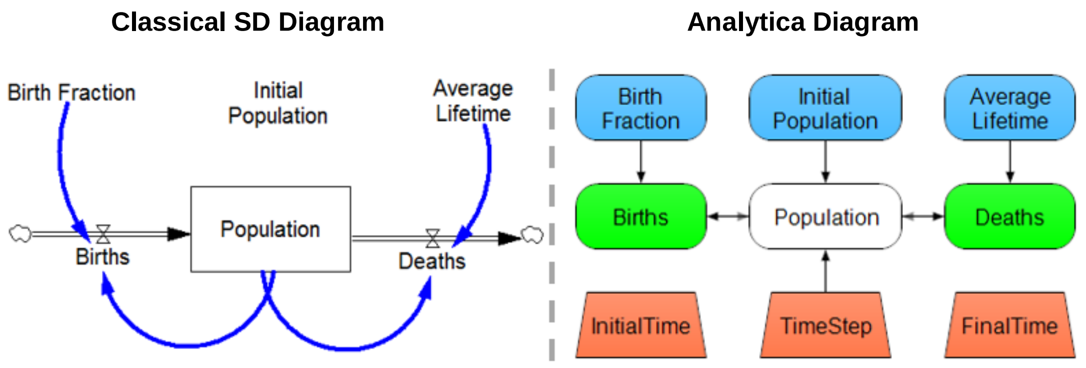


Figure 1. Comparison of Model Diagrams – Classical SD Software (e.g., Vensim) and Analytica

All integration of stocks in classical SD software is controlled under the hood, meaning that the modeler need not think about the mechanics of numerical integration. Inflows and outflows connected to the stock are assumed to be continuous differential equations (e.g., $dStock / dt = X$) and are also signified by a distinct icon (typically resembling a valve, which controls the flow to and from the stock). The stock is calculated by numerically integrating those rate equations over time. Classical SD software provides several options for numerical integration (e.g., Euler, Runge-Kutta), permitting the modeler to use more sophisticated methods if numerical integration error is a concern in the system being modeled. Typically in System Dynamics modeling, especially of social or human systems, Euler integration is sufficiently accurate (Sterman, 2000, p. 911).

Developing a dynamic model in Analytica requires that the user explicitly construct the numerical integration within the equation of any stock variable. The simplest method of doing so entails Euler integration in conjunction with the Dynamic[4] function in Analytica, as illustrated in the first equation

[4] http://wiki.analytica.com/index.php?title=Dynamic_function

below.[5] Those familiar with Euler integration will recognize that the value of the stock *and* flows in the *previous* time step are always used in calculating the value of the stock in the *current* time step. While the Dynamic function in Analytica is quite flexible, and is not necessarily constrained to the structure given below, I recommend consistent use of the formulation in Table 1 when constructing System Dynamics models (which often have numerous feedbacks) to avoid potential pitfalls with cyclic dependencies among variables and to maximize the similarity between classical SD models and those built in Analytica.

For illustration, I provide below how equations in Analytica compare with those in Vensim, though the construct is similar in other classical SD platforms.

Variable	**Vensim**	**Analytica**	**Units**
Population	INTEG[6](Births – Deaths, Initial Population)	Dynamic(InitialPopulation, Self[Time – 1][7] + TimeStep[8] * (Births[Time – 1] – Deaths[Time – 1]))	People
Births	Birth Fraction * Population	Same	People/Year
Deaths	Population / Average Lifetime	Same	People/Year
Initial Population	1000	Same	People
Initial Time	0	Same	Years
Final Time	100	Same	Years
Time Step	0.25	Same	Years
Time[9,10]	In Model Settings	Sequence(InitialTime, FinalTime, TimeStep)	Years

Table 1. Comparison of Equations in Vensim and Analytica for a Simple Dynamic Model

Notice that *the only equation that differs* between the Analytica model above and the model constructed in Vensim is the stock equation. Rate equations, intermediate variables, and constants are identical, though the time sequence is constructed in a different manner, as shown in Table 1.

Using the Analytica stock equation specified in Table 1, one can develop an SD model of any degree of dynamic complexity, incorporating feedbacks and delays just as with classical SD software. As should be apparent, there is very little marginal effort required to create a stock variable using explicit Euler integration in Analytica.

[5] The author has not attempted to employ other numerical integration methods, such as Runge-Kutta, in Analytica.
[6] The INTEG notation is inherent to the stock variable and signifies that the expression in the variable will be integrated over time. The modeler need not input the INTEG function explicitly, as every stock variable has the INTEG function already embedded in it.
[7] In the equation for Population, the [Time – 1] notation signifies the value of those variables *in the previous time step* (as opposed to the value of the variable one year prior, since the time step could be less than or greater than one year). The Self notation is reserved for variables using the Dynamic function and is simply a method permitting self-referral (e.g., at a previous time). You can use the actual identifier, or variable name, of the stock in place of Self.
[8] It is not technically required to explicitly use a time step in a stock equation in Analytica. However, including it in stock equations and in the definition of the Time variable makes it clear that flow variables are differential rate equations in continuous time, as traditional in SD. It also lets you easily vary the time step, to which SD modelers are accustomed.
[9] Initial Time, Final Time, and Time Step are input via the Models/Settings menu in Vensim. The time sequence requires that the time step be the same throughout the simulation.
[10] The time sequence in Analytica is explicitly coded with the Sequence function. The InitialTime and FinalTime variables are specified through creation of separate “constant” nodes. You may also specify Time in other ways, for instance with a time step that varies over the simulation.

Similarities between Analytica and Classical SD Software

Though Analytica and classical SD software products were originally developed for different purposes, Analytica shares many attributes with classical SD platforms that facilitate good modeling practice as espoused by the System Dynamics community. These are the most salient similarities.

Similarity #1: Visual, Icon-Based Modeling Environment

As is evident above in Figure 1, both Analytica and classical SD software employ an icon-based visual modeling environment, which permits modelers and clients to define and visualize the interrelationships among model variables. This feature facilitates transparency. Models are much easier to navigate and understand than in typical programing languages, which often require wading through hundreds or thousands of lines of code to understand their logic. Model diagrams (which Analytica terms “influence diagrams”) are easy to create using drag-and-drop methods. In both platforms, you can access the underlying equations by clicking each variable directly in the diagram.

Similarity #2: Declarative Language

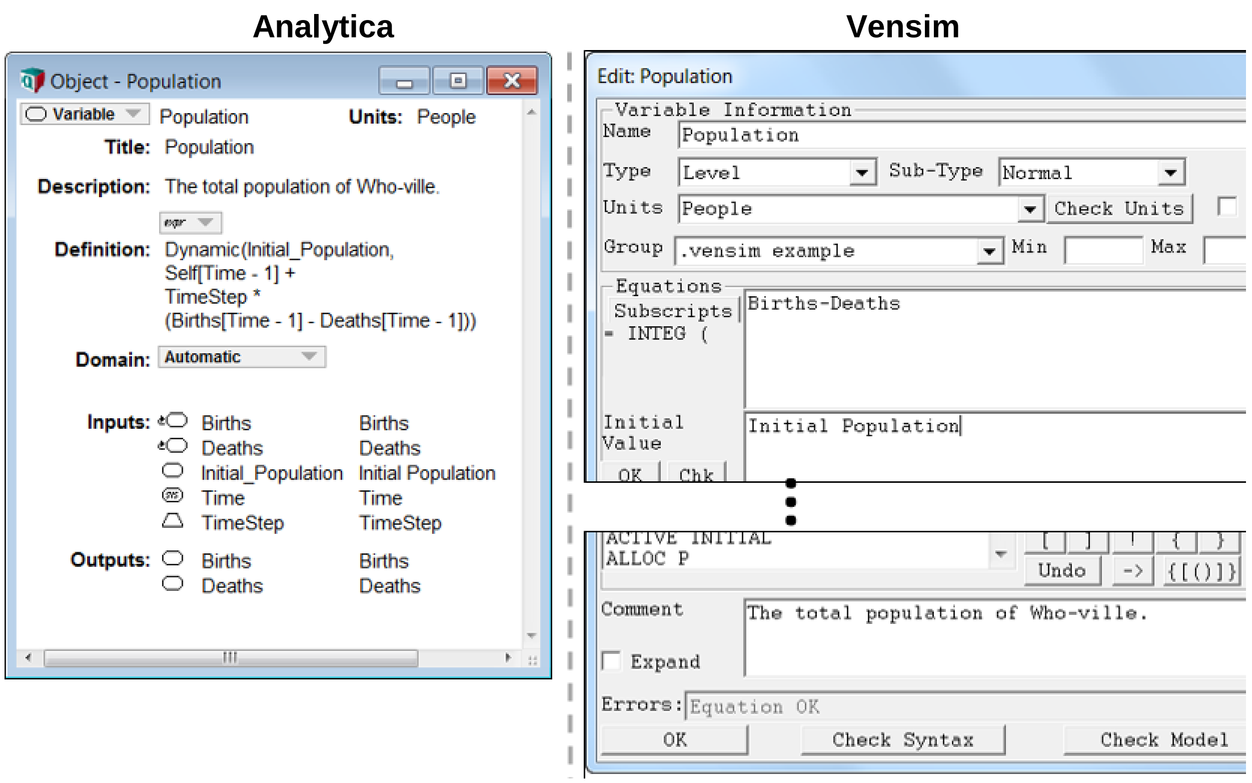
Both platforms use a purely declarative[11] language, meaning that each variable is defined by an equation or expression in terms of other variables. The modeler does not have to specify the flow of control. The platform automatically manages the sequence in which to calculate variables based on consistency and efficiency. Thus, they require no programming in the traditional sense. This also makes it much easier to write, understand, and debug models than with traditional imperative or procedural languages.

Similarity #3: Internal Documentation

Another key feature of both platforms is the ease of internally documenting every equation with a comment or description, and specifying the units of each variable.[12] Good System Dynamics modeling practice dictates that every variable in a model have a description and that units be clearly specified (Sterman, 2000, p. 852), to facilitate transparency and understanding. Figure 2 compares the object windows for a variable in Analytica and Vensim. They both contain fields for Description/Comment, Units, and Definition/Equations, and Domain (Min and Max in Vensim).

[11] http://vensim.com/the-next-great-thing/

[12] Note that some classical SD platforms have the capability to check units of each variable for internal consistency, a convenient feature that prevents unit inconsistency errors. Analytica does not have this feature.



Object - Population

Variable: Population Units: People
Title: Population
Description: The total population of Who-ville.
Definition: Dynamic(Initial_Population,
Self[Time - 1] +
TimeStep *
(Births[Time - 1] - Deaths[Time - 1]))
Domain: Automatic

Inputs:	Births	Births
	Deaths	Deaths
	Initial_Population	Initial Population
	Time	Time
	TimeStep	TimeStep
Outputs:	Births	Births
	Deaths	Deaths

Edit: Population

Variable Information
Name: Population
Type: Level Sub-Type: Normal
Units: People Check Units
Group: .vensim example Min Max
Equations: Births-Deaths
Subscripts
= INTEG (
Initial Value: Initial Population
OK Chk
ACTIVE INITIAL
ALLOC P
Undo
Comment: The total population of Who-ville.
Expand
Errors: Equation OK
OK Check Syntax Check Model

Figure 2. Variable Object Window in Analytica (left) and Vensim (right)

Other Similarities between Analytica and Classical SD Software

Several other similarities between Analytica and classical SD platforms should be reassuring to System Dynamics modelers. Both enable users to easily:

- create graphical user interfaces, permitting ready access to model inputs and outputs,
- trace model logic and dependencies among variables, and
- share models through an online interface without substantial model modification or programing required.

Advantages of Analytica

Analytica was originally developed by Dr. Max Henrion in the mid-1990s to efficiently handle multi-dimensional analyses (a limitation of spreadsheets) and to better address uncertainty and risk.[13] It has since been refined to incorporate advanced optimization and other capabilities. Here I describe several features I have found to be especially convenient in the development of System Dynamics models.

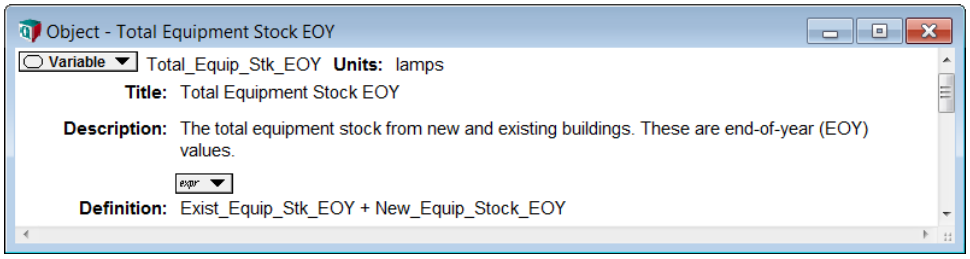
Analytica Advantage #1: Array Abstraction

Arguably the most powerful and differentiating feature of Analytica is array abstraction: You can change any variable from a scalar to a vector, or to a multi-dimensional array, and the rest of the model *automatically* adapts accordingly. The ease with which you can do this in Analytica is unparalleled, owing to its proprietary Intelligent Arrays™ algorithm. This algorithm automatically propagates the dimensionality of any data input or intermediate variable to all downstream variables, requiring no

[13] https://en.wikipedia.org/wiki/Analytica_(software)#cite_note-83

additional changes to variables carrying the dimensions. This capability differs dramatically from constructing multi-dimensional analyses in classical SD software using subscripts (in Vensim, for instance), which requires the modeler to ensure every equation carrying the desired dimensionality contains all appropriate subscripts. While it is *feasible* to construct multi-dimensional models in classical SD software platforms, it is *considerably more difficult and less transparent* than in Analytica, as illustrated in these examples.

The equation for the value calculated in Figure 3, for instance, is quite simple – it just adds two variables. However, the summed variables are both arrays with multiple dimensions, with *indexes*[14] including Technologies, Applications, Lumen Bins, Housing Types, and Scenarios. So, the output is also dimensioned by these indexes, as illustrated in Figure 4, even though the equation does not mention those indexes.

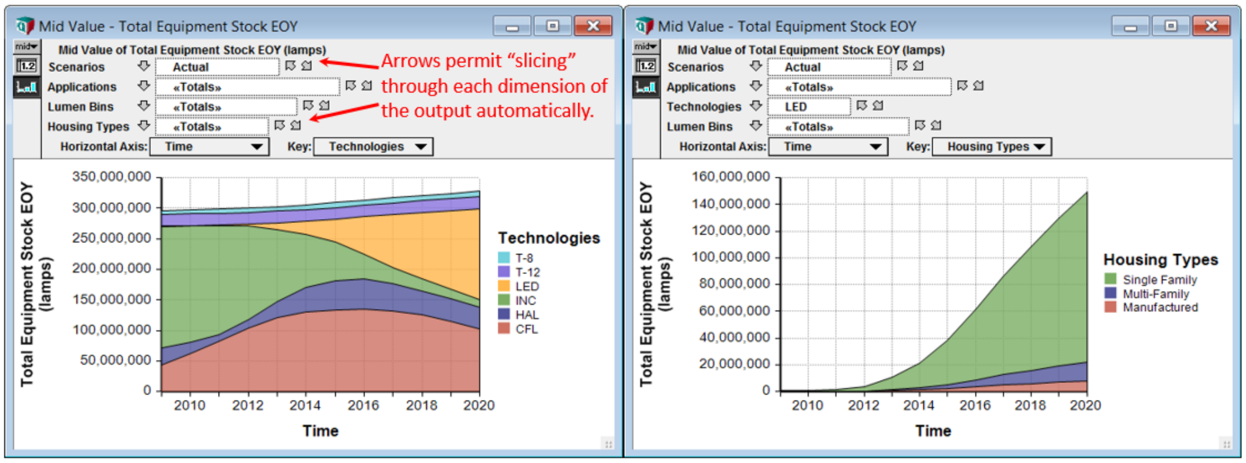


Total_Equip_Stk_EOY **Units:** lamps
Title: Total Equipment Stock EOY
Description: The total equipment stock from new and existing buildings. These are end-of-year (EOY) values.
Definition: Exist_Equip_Stk_EOY + New_Equip_Stock_EOY

Figure 3. Illustration of a Simple Equation in Analytica, though Output is Multi-Dimensional

If you expand or reduce the dimensions of the input nodes, *the equation for Total Equipment Stock EOY remains unchanged due to array abstraction*. I provide an illustration of the output of this node in graphical form in Figure 4. You can select any two-dimensional slice in the result graph or table using the arrows to the right of each index, making visual inspection of every dimension of the model a trivial matter. You can rearrange the axes and key to pivot, stack or sort the output along different dimensions with just a button click. For instance, the figure below shows the same output node viewed in two ways (one showing all Lumen Bins, stacked by Technologies, and the other showing only the LED Technology, stacked by Housing Types). The ability to rapidly visually inspect multi-dimensional output facilitates model transparency in granular models and accelerates learning.

[14] In Analytica, the various dimensions along which variables are calculated are referred to as indexes, which are analogous to, but more flexible than, subscripts in classical SD software, such as Vensim.


Mid Value - Total Equipment Stock EOY

Mid Value of Total Equipment Stock EOY (lamps)

Scenarios: Actual
Applications: «Totals»
Lumen Bins: «Totals»
Housing Types: «Totals»
Horizontal Axis: Time
Key: Technologies

Arrows permit "slicing" through each dimension of the output automatically.

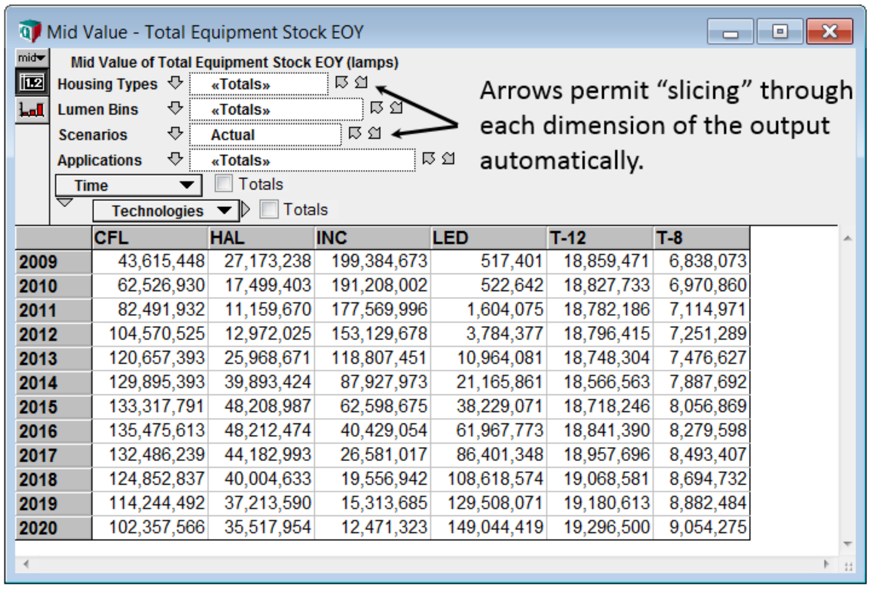
Mid Value - Total Equipment Stock EOY

Mid Value of Total Equipment Stock EOY (lamps)

Scenarios: Actual
Applications: «Totals»
Technologies: LED
Lumen Bins: «Totals»
Horizontal Axis: Time
Key: Housing Types

Figure 4. Automated Multi-Dimensional Output in Analytica: Two Different Views of the Same Variable

This output is also automatically viewable as a multi-dimensional data table (as is every variable in the model), each dimension of which is also easily accessible, as illustrated below.


Mid Value - Total Equipment Stock EOY

Mid Value of Total Equipment Stock EOY (lamps)

Housing Types: «Totals»
Lumen Bins: «Totals»
Scenarios: Actual
Applications: «Totals»
Time ☐ Totals
Technologies ☐ Totals

Arrows permit "slicing" through each dimension of the output automatically.

	CFL	HAL	INC	LED	T-12	T-8
2009	43,615,448	27,173,238	199,384,673	517,401	18,859,471	6,838,073
2010	62,526,930	17,499,403	191,208,002	522,642	18,827,733	6,970,860
2011	82,491,932	11,159,670	177,569,996	1,604,075	18,782,186	7,114,971
2012	104,570,525	12,972,025	153,129,678	3,784,377	18,796,415	7,251,289
2013	120,657,393	25,968,671	118,807,451	10,964,081	18,748,304	7,476,627
2014	129,895,393	39,893,424	87,927,973	21,165,861	18,566,563	7,887,692
2015	133,317,791	48,208,987	62,598,675	38,229,071	18,718,246	8,056,869
2016	135,475,613	48,212,474	40,429,054	61,967,773	18,841,390	8,279,598
2017	132,486,239	44,182,993	26,581,017	86,401,348	18,957,696	8,493,407
2018	124,852,837	40,004,633	19,556,942	108,618,574	19,068,581	8,694,732
2019	114,244,492	37,213,590	15,313,685	129,508,071	19,180,613	8,882,484
2020	102,357,566	35,517,954	12,471,323	149,044,419	19,296,500	9,054,275

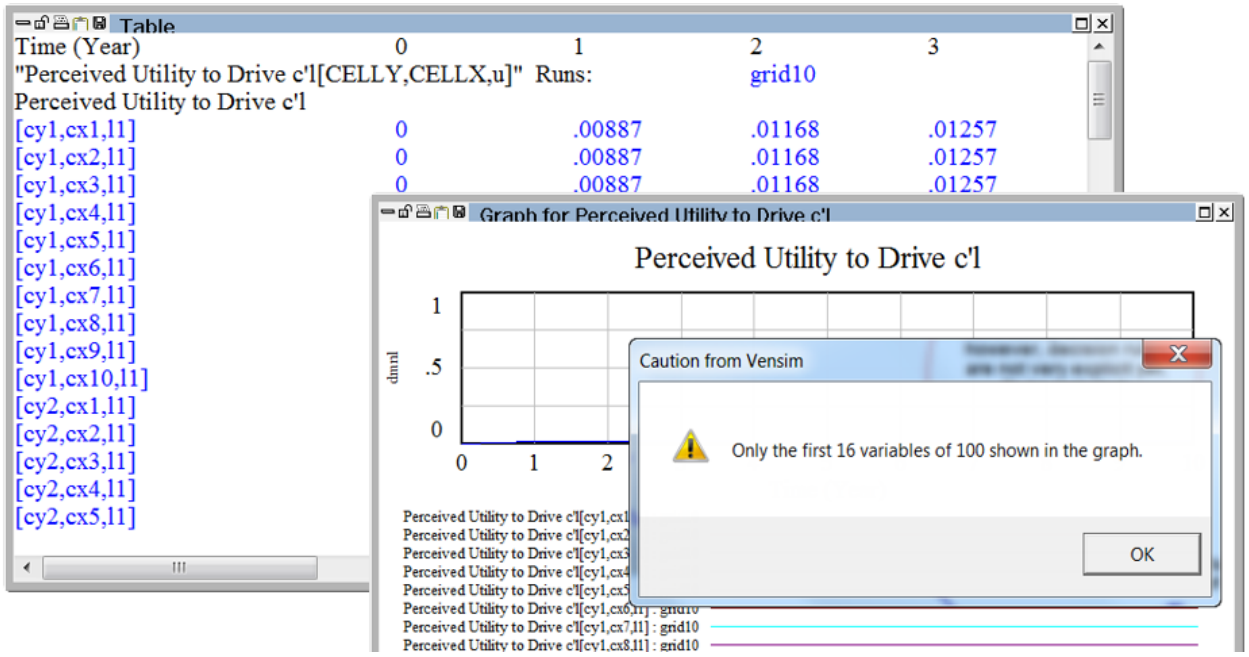
Figure 5. Illustration of a Multi-Dimensional Output Table of a Variable In Analytica

In contrast, a similar equation in Vensim requires you to specify every subscript for each variable, for example:

> Total Equipment Stock EOY = Exist_Equip_Stock_EOY[Technologies, Applications, Lumen Bins, Scenario, Housing Types] + New_Equip_Stock_EOY[Technologies, Applications, Lumen Bins, Scenario, Housing Types]

More significantly, *every downstream variable carrying those dimensions would also have to specify every subscript*, meaning that adding or removing a dimension requires you to modify many equations in the model (possibly dozens or hundreds). Creating equations that manipulate or transform multi-dimensional variables is much easier with Analytica's array abstraction than manipulating multi-dimensional variables in classical SD software (e.g., using the VECTOR ELM MAP and other functions in Vensim).

Additionally, it is difficult in classical SD software to view the many dimensions of the output, which makes understanding the model at that level of granularity challenging. For illustration, I have excerpted data output from a Vensim model that forecasts spatial adoption of hydrogen fuel cell vehicles (Struben 2006). The multi-dimensional data are only viewable in flat format in Vensim, as illustrated in Figure 6. Graphically, it is impossible to view more than a subset of the data, and automatically slicing the data along the different dimensions is not possible as it is in Analytica. Rather, to visualize and understand these output, I had to export the output into Microsoft Excel, rearrange it to be two-dimensional, and then manually build a graph to display it (Welch 2006). Such effort to view and understand the output of a multi-dimensional model slows the development and learning processes. Other classical SD software platforms suffer a similar affliction.


```
Table
Time (Year)                                0          1          2          3
"Perceived Utility to Drive c'l[CELLY,CELLX,u]"  Runs:           grid10
Perceived Utility to Drive c'l
[cy1,cx1,l1]                               0          .00887     .01168     .01257
[cy1,cx2,l1]                               0          .00887     .01168     .01257
[cy1,cx3,l1]                               0          .00887     .01168     .01257
[cy1,cx4,l1]
[cy1,cx5,l1]
[cy1,cx6,l1]
[cy1,cx7,l1]
[cy1,cx8,l1]
[cy1,cx9,l1]
[cy1,cx10,l1]
[cy2,cx1,l1]
[cy2,cx2,l1]
[cy2,cx3,l1]
[cy2,cx4,l1]
[cy2,cx5,l1]
```

```
Graph for Perceived Utility to Drive c'l
Perceived Utility to Drive c'l
Perceived Utility to Drive c'l[cy1,cx1
Perceived Utility to Drive c'l[cy1,cx2
Perceived Utility to Drive c'l[cy1,cx3
Perceived Utility to Drive c'l[cy1,cx4
Perceived Utility to Drive c'l[cy1,cx5
Perceived Utility to Drive c'l[cy1,cx6,l1] : grid10
Perceived Utility to Drive c'l[cy1,cx7,l1] : grid10
Perceived Utility to Drive c'l[cy1,cx8,l1] : grid10
```

Caution from Vensim

Only the first 16 variables of 100 shown in the graph.

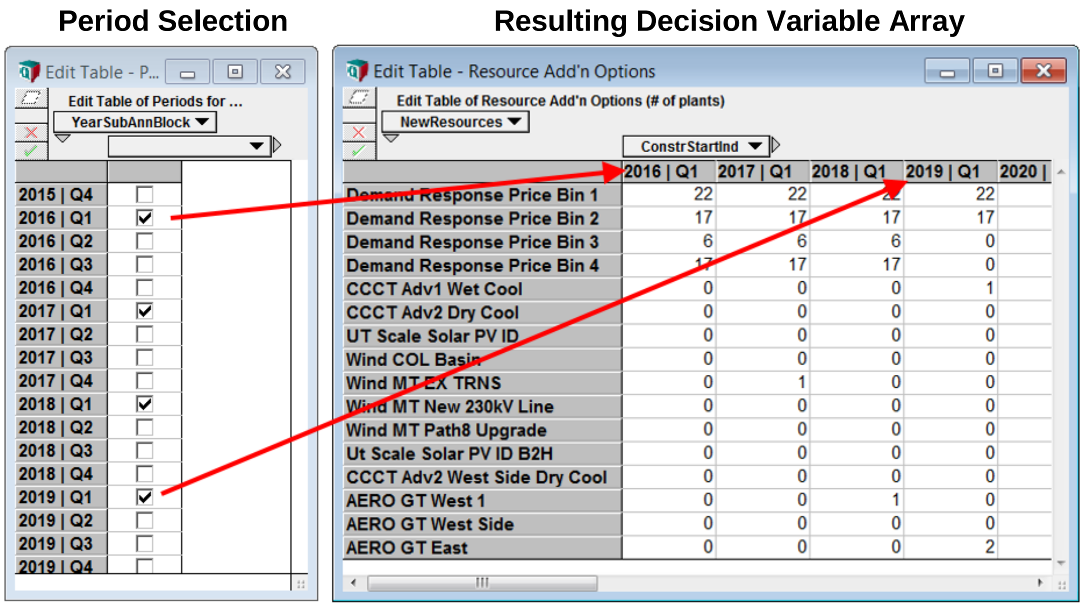
Figure 6. Illustration of Multi-Dimensional Data Viewing Limitations in Vensim. Graphic Source (Struben 2006 Model)

Often a modeler is not sure when starting a model which and how many dimensions will be needed to achieve adequate accuracy. Sometimes client requirements change during a modeling engagement. The ability to modify dimensions so easily with Analytica makes it much easier to adapt the model detail as needs dictate. In one engagement, my team transformed a sector-level (e.g., residential, commercial) analysis of solar photovoltaic adoption using Bass (1969) diffusion (as presented in Sterman, 2000, p. 332) into to a spatial model, simply by disaggregating key inputs such as customer counts and initial installations to the electric utility substation level. This change was easy to implement, only requiring modification of a few key inputs. The rest of the model then automatically expanded by the added input dimension – including every downstream variable, output table, and output graphic. In another case, we originally modeled the efficient lighting market in the Northwest U.S. (using Bass Diffusion overlaid on a stock turnover construct) at the level of building type (e.g., single-family homes, retail, offices). During the engagement, my team determined that data availability and accuracy did not warrant such disaggregation, so the inputs were collapsed so that they were only at the sector level. Such changes are made easily in Analytica due to array abstraction, but often would require reconstructive surgery in classical SD software platforms.

As should be apparent, the ability of Analytica to deal with multi-dimensional analyses affords the modeler and the client a great deal of flexibility in addition to efficiency of model construction (meaning more time can be spent understanding the actual problem).

Analytica Advantage #2: Structured Optimization

Another key advantage of the Analytica software platform is its ability to perform what it terms “structured optimization.” All properties of array abstraction extend to the formulation of optimization problems, which are easy to formulate. This feature facilitates their creation and modification in several ways. First, many optimization problems involve multi-dimensional decision variables and constraints. With structured optimization, you can specify a vector or array of decision variables and constraints that are automatically recognized by the optimization engine. Likewise, you can easily construct multi-dimensional upper and lower bounds on the decision variables. For instance, in a dynamic optimization model I developed for the Northwest Power and Conservation Council (NPCC 2016), one set of decision variables involved the number of electric resources to be added in each of several time periods. This decision variable was indexed by the resource type and by the start time for resource construction. As the user adds or subtracts technologies (or time periods for the resource addition decision), the decision variables and constraints automatically adjust, just as other variables do with array abstraction. Further, the user can easily modify the length of each of these indexes through a user interface. Figure 7 shows how the user can select which time periods to include in the decision variables.


Period Selection
Resulting Decision Variable Array
Edit Table - P...
Edit Table of Periods for ...
YearSubAnnBlock
Edit Table - Resource Add'n Options
Edit Table of Resource Add'n Options (# of plants)
NewResources
ConstrStartInd

2015 | Q4	☐
2016 | Q1	✓
2016 | Q2	☐
2016 | Q3	☐
2016 | Q4	☐
2017 | Q1	✓
2017 | Q2	☐
2017 | Q3	☐
2017 | Q4	☐
2018 | Q1	✓
2018 | Q2	☐
2018 | Q3	☐
2018 | Q4	☐
2019 | Q1	✓
2019 | Q2	☐
2019 | Q3	☐
2019 | Q4	☐

	2016 | Q1	2017 | Q1	2018 | Q1	2019 | Q1	2020 |
Demand Response Price Bin 1	22	22	22	22	
Demand Response Price Bin 2	17	17	17	17	
Demand Response Price Bin 3	6	6	6	0	
Demand Response Price Bin 4	17	17	17	0	
CCCT Adv1 Wet Cool	0	0	0	1	
CCCT Adv2 Dry Cool	0	0	0	0	
UT Scale Solar PV ID	0	0	0	0	
Wind COL Basin	0	0	0	0	
Wind MT EX TRNS	0	1	0	0	
Wind MT New 230kV Line	0	0	0	0	
Wind MT Path8 Upgrade	0	0	0	0	
Ut Scale Solar PV ID B2H	0	0	0	0	
CCCT Adv2 West Side Dry Cool	0	0	0	0	
AERO GT West 1	0	0	1	0	
AERO GT West Side	0	0	0	0	
AERO GT East	0	0	0	2	

Figure 7. Example of Multi-Dimensional Decision Variable in a System Dynamics Model

Structured optimization combined with array abstraction permits the modeler to solve numerous optimization problems within a single dynamic model. This powerful feature provides a degree of modeling flexibility not available in classical SD software tools. In one engagement with a large electric utility, I used this capability to solve two distinct optimizations in every time step of a Bass diffusion model to simulate adoption of rooftop solar photovoltaic (PV) and battery storage systems. The first was a linear program (LP) that calculated the optimal dispatch of a battery storage system under electricity rates that varied by time of day. This LP incorporated a distinct but integrated dynamic sub-model, employing Analytica's Dynamic function along an orthogonal time index (the hour in a representative week of system operation). The sub-model simulated the battery stage-of-charge (the stock) and rates of charging and discharging (the flows). An illustration of the solution to a single LP (for one time step in the larger dynamic model) is provided in Figure 8, which shows an optimal battery charge and discharge strategy under a given customer electric load profile, solar PV generation profile, and electric rate structure.

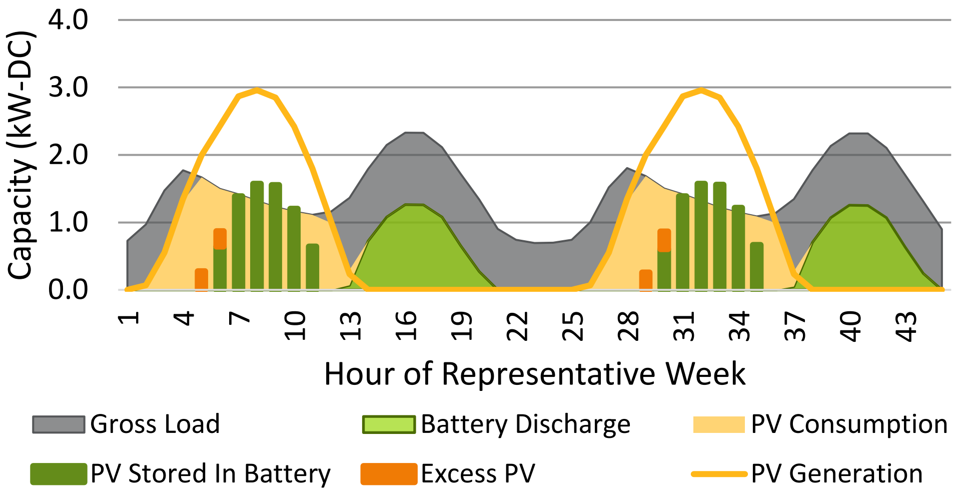
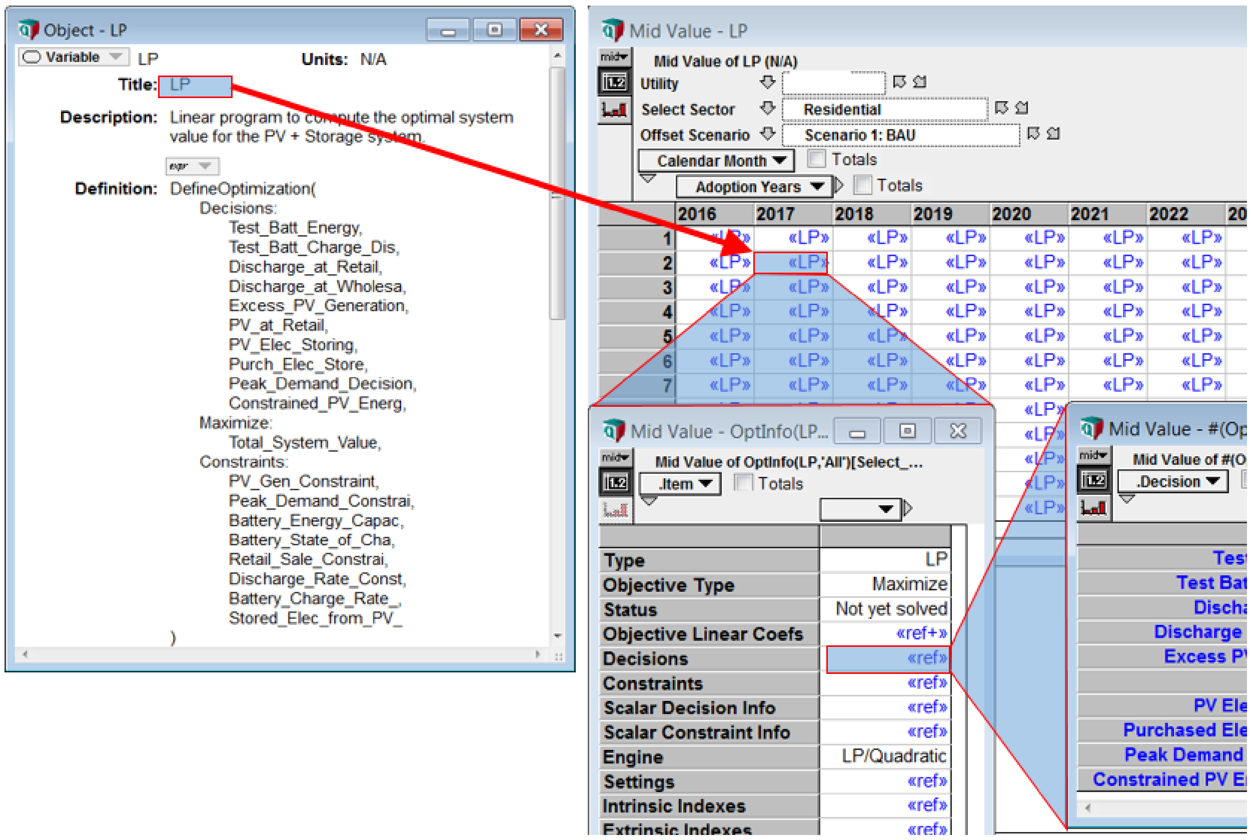


Figure 8. Illustration of One Optimization Output (in One Time Step) in a System Dynamics Model Created in Analytica with Multiple, Distinct Optimizations

The second optimization was a nonlinear program that solved in each time step for the forecast lease price at which a solar-plus-storage system could likely be offered by a provider. The calculation used a discounted cash flow optimization model, which was a function of several variables that changed over time (e.g., installation costs and tax credits). Finally, the same model employed a third optimization (also nonlinear) to calibrate coefficients of the Bass diffusion model to ten years of historical adoption data. The integration of three separate optimization problems, each solved along a different time dimension, in a single dynamic model, is not practicable in classical SD software. This capability was critical to developing a model that could reflect the real-world economics of this rapidly changing technology, a prerequisite to forecasting adoption. Figure 9 illustrates how literally hundreds of optimization problems were solved within a single dynamic model. Each **«LP»** symbol in the table below represents a separate linear program in the dynamic model, each of which has a wealth of information generated by an LP definition that is automatically array abstracted over adoption year, calendar month, electricity rate structure scenario, and customer sector. What is powerful and convenient is that all the information below was automatically generated by array abstraction through a single definition node for the LP. An example of the definition that generated the multi-dimensional optimization structure is provided for illustration in Figure 9. Notice the simplicity of the definition, which only contains the variable identifiers for the objective function, decision variables, and constraints, each of which is multi-dimensional.


Object - LP

Variable LP Units: N/A

Title: LP

Description: Linear program to compute the optimal system value for the PV + Storage system.

Definition:
```
DefineOptimization(
    Decisions:
        Test_Batt_Energy,
        Test_Batt_Charge_Dis,
        Discharge_at_Retail,
        Discharge_at_Wholesa,
        Excess_PV_Generation,
        PV_at_Retail,
        PV_Elec_Storing,
        Purch_Elec_Store,
        Peak_Demand_Decision,
        Constrained_PV_Energ,
    Maximize:
        Total_System_Value,
    Constraints:
        PV_Gen_Constraint,
        Peak_Demand_Constrai,
        Battery_Energy_Capac,
        Battery_State_of_Cha,
        Retail_Sale_Constrai,
        Discharge_Rate_Const,
        Battery_Charge_Rate_,
        Stored_Elec_from_PV_
)
```

Mid Value - LP

Mid Value of LP (N/A)

Utility:

Select Sector: Residential

Offset Scenario: Scenario 1: BAU

Calendar Month ☐ Totals

Adoption Years ☐ Totals

	2016	2017	2018	2019	2020	2021	2022
1	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»
2	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»
3	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»
4	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»
5	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»
6	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»
7	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»	«LP»

Mid Value - OptInfo(LP...

Mid Value of OptInfo(LP,'All')[Select_...

.Item ☐ Totals

Type	LP
Objective Type	Maximize
Status	Not yet solved
Objective Linear Coefs	«ref+»
Decisions	«ref»
Constraints	«ref»
Scalar Decision Info	«ref»
Scalar Constraint Info	«ref»
Engine	LP/Quadratic
Settings	«ref»
Intrinsic Indexes	«ref»
Extrinsic Indexes	«ref»

Mid Value - #(Op

Mid Value of #(O

.Decision

Tes
Test Bat
Discha
Discharge
Excess P
PV Ele
Purchased Ele
Peak Demand
Constrained PV E

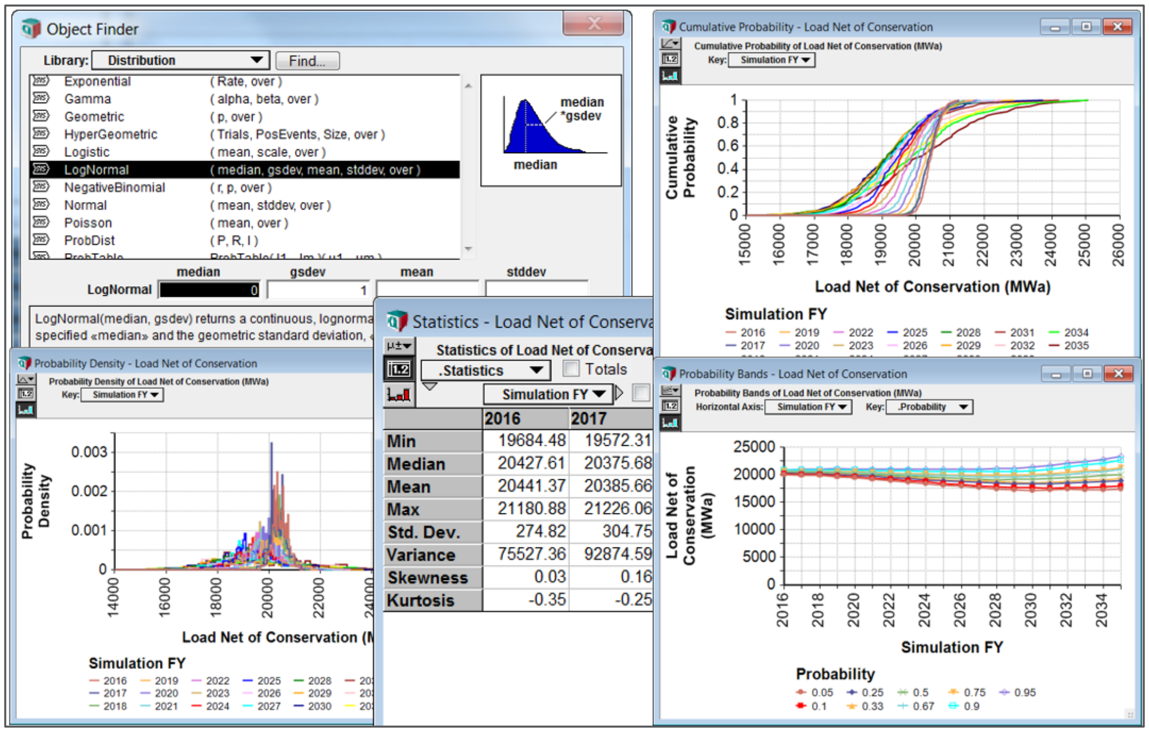
Figure 9. Illustration of Structured Optimization in Analytica. Hundreds of Optimizations Were Solved in a Single System Dynamics Model.

Analytica Advantage #3: Enhanced Uncertainty Analysis Capability

Classical SD software has some capability to incorporate uncertainty analysis (e.g., Monte Carlo sensitivity). However, I find that Analytica offers several advantages to modelers who are especially concerned with accounting for uncertainty and risk, as Analytica was built with a distinct focus on these issues. For instance, Analytica offers:

- a larger selection of standard and custom probability distribution functions,
- a wider range of graphical or tabular ways to display uncertain (in every time step of the simulation), such as:
  - statistics (mean, median, percentiles, etc.),
  - probability density and cumulative distribution functions,
  - data points from the Monte Carlo simulations,
- ability to create correlations among random variables,
- ability to extract specific statistics from any random output for use in downstream equations (e.g., mean values or percentiles of output distributions for objective functions of optimizations),
- ability to calculate risk metrics, such as value at risk, conditional value at risk, etc.,
- ability to slice and view uncertain output using all array abstraction viewing benefits previously discussed,
- ability to control the dimensions over which a random variable is uncertain (e.g., along specified indices).

While I do not offer in this paper a detailed comparison of the uncertainty analysis capabilities of Analytica with those of classical SD software platforms, I provide a few graphics below to give the reader a sense of the capabilities and user-friendliness of Analytica when creating stochastic models.



Object Finder
Library: Distribution
Find...
median
*gsdev
median
LogNormal
median
gsdev
mean
stddev
0
1
Probability Density - Load Net of Conservation
Probability Density of Load Net of Conservation (MWa)
Key: Simulation FY
Probability Density
Load Net of Conservation
Simulation FY
Statistics - Load Net of Conserva
Statistics of Load Net of Conserva
.Statistics
Totals
Simulation FY
Cumulative Probability - Load Net of Conservation
Cumulative Probability of Load Net of Conservation (MWa)
Key: Simulation FY
Cumulative Probability
Load Net of Conservation (MWa)
Simulation FY
Probability Bands - Load Net of Conservation
Probability Bands of Load Net of Conservation (MWa)
Horizontal Axis: Simulation FY
Key: .Probability
Load Net of Conservation (MWa)
Simulation FY
Probability

Exponential	(Rate, over)
Gamma	(alpha, beta, over)
Geometric	(p, over)
HyperGeometric	(Trials, PosEvents, Size, over)
Logistic	(mean, scale, over)
LogNormal	(median, gsdev, mean, stddev, over)
NegativeBinomial	(r, p, over)
Normal	(mean, stddev, over)
Poisson	(mean, over)
ProbDist	(P, R, I)

LogNormal(median, gsdev) returns a continuous, lognorma
specified «median» and the geometric standard deviation,

	2016	2017
Min	19684.48	19572.31
Median	20427.61	20375.68
Mean	20441.37	20385.66
Max	21180.88	21226.06
Std. Dev.	274.82	304.75
Variance	75527.36	92874.59
Skewness	0.03	0.16
Kurtosis	-0.35	-0.25

Figure 10. Various Uncertainty Inputs and Outputs in Analytica

Other Advantages of using Analytica for System Dynamics Modeling

Analytica has several other features that make life easier for a System Dynamics modeler compared with classical SD software, such as:

- ease of importing and exporting multi-dimensional data from/to Excel, databases, etc.,
- automated performance profiling (for tracing drivers of run times and memory),
- modular model construction (facilitates navigation, creation of reusable sub-modules, etc.).

Advantages of Classical SD Software, with Example Work-arounds in Analytica

Given that classical SD software is purpose-built for building and analyzing dynamic models, it is not surprising that those platforms have some advantages relative to Analytica in certain dynamic modeling applications. If the model is not multi-dimensional, does not have a need to incorporate more than a single optimization, or has a high degree of dynamic complexity and low detail complexity, classical SD software may be preferred. I describe below the main advantages of classical SD software platforms over Analytica for SD applications; I also suggest ways to work around these limitations.

Classical SD software Advantage #1: Visualization of Feedback Loops & Stock/Flow Structure

System Dynamics modelers often represent positive and negative feedback through causal loop diagrams (Sterman 2000, p. 137), using a series of curved arrows to illustrate the feedback loops. Likewise, they are accustomed to viewing the stock/flow structure in the form of boxes and valves, with any feedback from a stock to its flow also conveyed by a curved arrow. In Analytica, the straight arrow structure and lack of distinct stock and flow icons make this depiction more challenging. In one case, I worked around this limitation by copying and pasting stock/flow images from Vensim onto the Analytica diagram to portray the structure in a manner more aligned with System Dynamics convention, as illustrated in Figure 11.

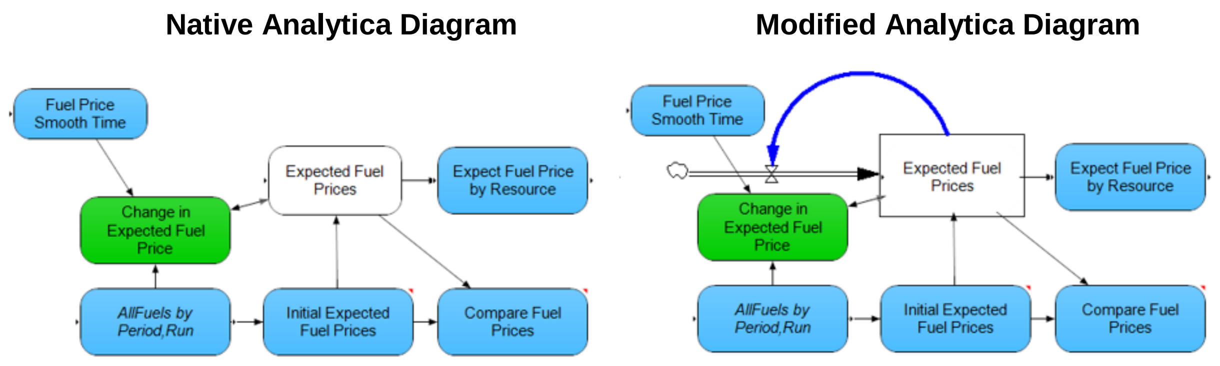
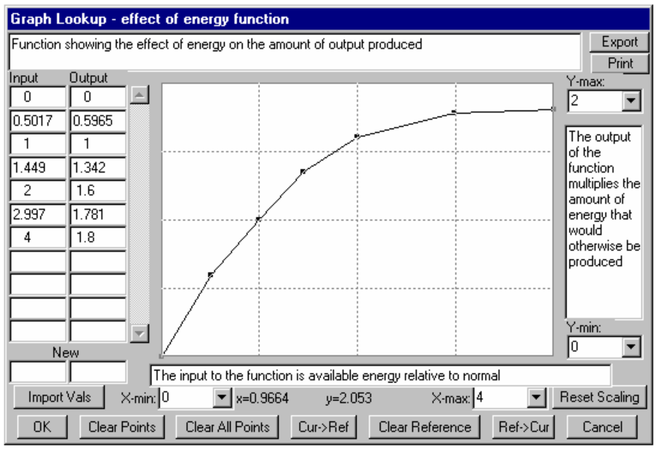

Native Analytica Diagram
Modified Analytica Diagram
Fuel Price Smooth Time
Change in Expected Fuel Price
Expected Fuel Prices
Expect Fuel Price by Resource
AllFuels by Period,Run
Initial Expected Fuel Prices
Compare Fuel Prices
Fuel Price Smooth Time
Change in Expected Fuel Price
Expected Fuel Prices
Expect Fuel Price by Resource
AllFuels by Period,Run
Initial Expected Fuel Prices
Compare Fuel Prices

Figure 11. Stock/Flow Influence Diagram in Analytica (Native and Modified)

Classical SD software Advantage #2: Graphical Creation of Look-up Functions

System Dynamics modeling sometimes employs the use of nonlinear relationships among variables that are created using a "Lookup" function (also referred to as a table function), which in classical SD software can be created graphically through a user interface, as illustrated below. This is a convenient method of creating a user-defined function with an arbitrary relationship between the x and y variables. Downstream variables can use this nonlinear relationship as a function with an input value for x, and the output will return the interpolated value for y. For instance, a downstream variable defined in Vensim as: Production Multiplier = Effect of Energy Function(Input Energy Value), where Input Energy Value = 1.25, would return a value of Production Multiplier = 1.19 with a lookup table defined as shown in Figure 12.



Graph Lookup - effect of energy function

Function showing the effect of energy on the amount of output produced

Input	Output
0	0
0.5017	0.5965
1	1
1.449	1.342
2	1.6
2.997	1.781
4	1.8

Y-max: 2

The output of the function multiplies the amount of energy that would otherwise be produced

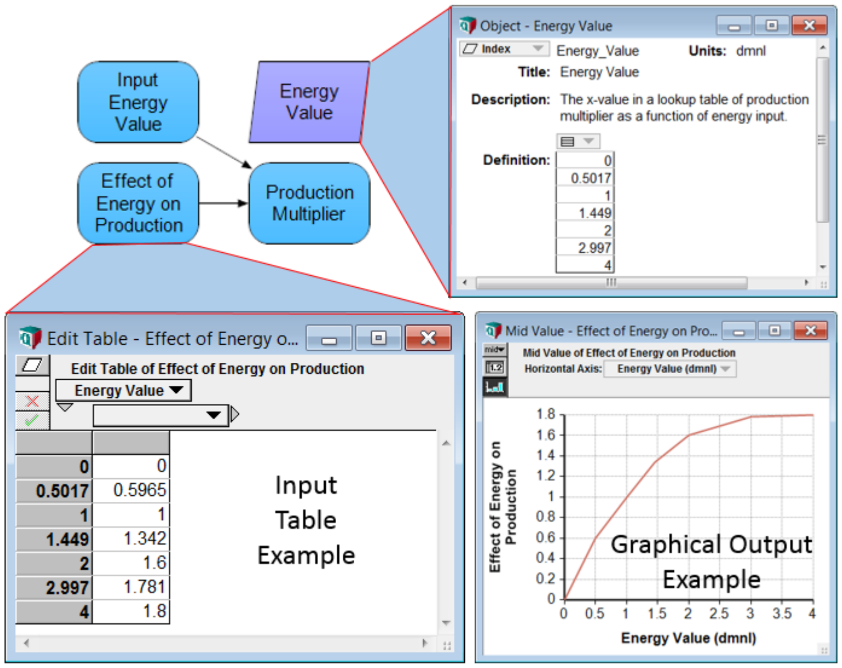
Y-min: 0

The input to the function is available energy relative to normal

X-min: 0 x=0.9664 y=2.053 X-max: 4

Figure 12. Illustration of a Graphical Lookup Table in Vensim. Source: Vensim Help Manual

In Analytica, you must create the table manually by defining an index for the x-values and an edit table that provides the y-values corresponding with each x-value. You use an interpolation function to extract the appropriate value from the table, as in this example:



Index: Energy_Value **Units:** dmnl
Title: Energy Value
Description: The x-value in a lookup table of production multiplier as a function of energy input.
Definition:

0
0.5017
1
1.449
2
2.997
4

Edit Table of Effect of Energy on Production
Energy Value

Energy Value	
0	0
0.5017	0.5965
1	1
1.449	1.342
2	1.6
2.997	1.781
4	1.8

Mid Value of Effect of Energy on Production
Horizontal Axis: Energy Value (dmnl)

Figure 13. Creation of a Lookup Table in Analytica

```
Energy_Value := [0, 0.5017, 1, 1.449,2, 2.997, 4]

Effect_of_Energy_on_Prod := Table(Energy_Value)(0,0.5965,1,1.342,1.6,1.781,1.8)

Production_Multiplier := LinearInterp(Energy_Value, Effect_of_Energy_on_,
          Input_Energy_Value, Energy_Value)
```

This nonlinear lookup function in Analytica works fine, but it is more convenient using classical SD software platforms, which offer graphical creation of Lookup functions.

Classical SD software Advantage #3: Smooths, Delays, and Trending

System Dynamics models by their nature incorporate delays (Sterman 2000, p. 409), including physical material delays (e.g., the average failure time of a piece of equipment) and information delays (e.g., the average time for a decision maker to change an opinion based on new data). A number of constructs are used to model these delays, such as first (and higher) order exponential smoothing, and more complex expectation trending algorithms (Sterman 1987). Classical SD software contains several compiled functions that facilitate modeling delays without having to physically construct each stock and flow in the delay structure. For instance, Vensim contains functions such as SMOOTH, DELAY, and TREND, among others (e.g., higher order and information SMOOTH and DELAY functions). Analytica does not have comparable built-in functions. However, it is possible to model any of these delays through explicit construction of the stocks and flows associated with the delay type desired. In some cases, this approach offers additional flexibility and accuracy, as I will describe, though at the expense of a nominal additional formulation effort.

I provide below an example of how one can use array abstraction in Analytica to simulate an Nth-order material delay (Sterman, 2000, p. 417) (an explicit construction of a material delay represented by the DELAY3 function in Vensim if N were equal to three), as shown in Figure 14.

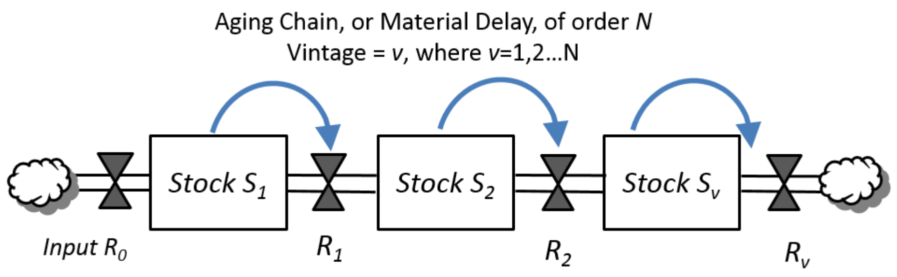

Aging Chain, or Material Delay, of order *N*
Vintage = *v*, where *v*=1,2...N
Stock S_1
Stock S_2
Stock S_v
Input R_0
R_1
R_2
R_v

Figure 14. Illustration of Nth-order Aging Chain, or Material Delay

To facilitate reader understanding and ease of reconstruction, I provide the influence diagram in Figure 15 and the equations in Analytica in Table 2.

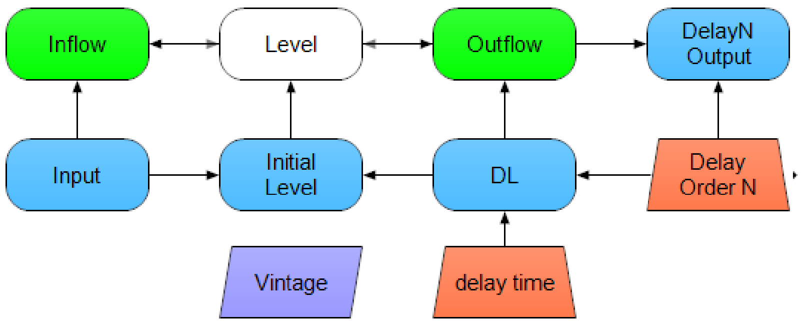


Figure 15. Influence Diagram in Analytica for an Nth-Order Material Delay

Variable	Analytica Equation	Units
Inflow	If Vintage = 1 THEN Input ELSE Outflow[Vintage = Vintage – 1]	Units/Year
Outflow	Level / DL	Units/Year
Level	Dynamic(Initial_Level, Self[Time – 1] + TimeStep * (Inflow[Time – 1] – Outflow[Time – 1]))	Units
Initial_Level	Input[@Time = 1] * DL	Units
Vintage	Sequence(1, Delay_Order_N, 1)	Dmnl[15]
Delay_Order_N	3	Dmnl
Delay_Time	10	Years
DL	Delay_Time / Delay_Order_N	Years
Delay_N_Output	Outflow[Vintage = Delay_Order_N]	Years
Input	100 + 100 * (Time >=5)	Units/Year
Time	Sequence(0, FinalTime, Timestep)	Years
FinalTime	40	Years

Table 2. Equations in Analytica for an Nth-Order Material Delay

Figure 16 compares the DelayN output in Analytica with an input starting at 100 units/year, stepping up to 200 units/year at Time = 5 (where Delay_Time = 10 years and Delay_Order_N = 3). The Time sequence uses the method shown above with Timestep = 0.25 years.

[15] Dimensionless

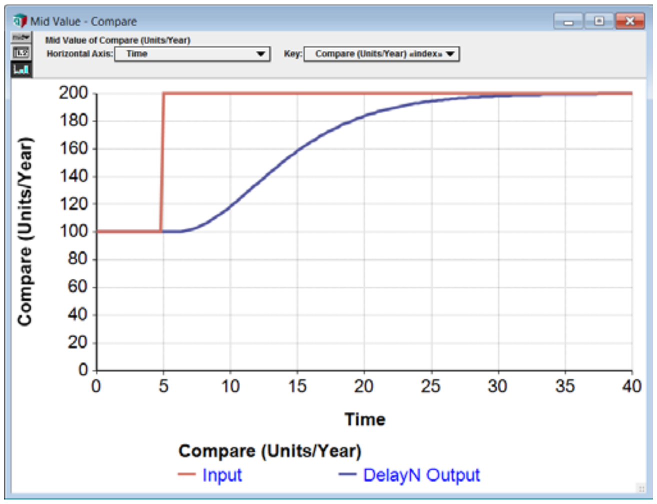


Figure 16. Comparison of Input with DelayN Output. N=3, delay time = 10 years.

This formulation, combined with the previous discussion of how to construct a stock, enables you to use Analytica to recreate any of the typical delay or smoothing structures in classical SD software functions.

Notice that this formulation assumes that stocks and flows in the delay chain begin in dynamic equilibrium (Sterman 2000, p. 232), consistent with the DELAY3 function in Vensim[16]. The initial level of each of the N stocks is the same, and is the product of the initial input and the delay length (DL) of each stock, consistent with Little's Law (Little 1961). This construct, while easy to formulate, can be limiting when simulating a stock that has historically been growing, which is often the case in the real world. In such a situation, the aging chain would be front-loaded, where upstream stocks are larger than downstream stocks. In one analysis conducted by Welch and Rogers (2010) of the failure distribution of residential appliances, we accounted for this by calculating the distribution of initial values of the total stock into each of the vintages, *v*, in an aging chain of order N (the order was optimized to fit historical failure data). Explicitly modeling the aging chain construct in Analytica (as opposed to relying on a DELAY function) facilitated allocating an estimated total stock quantity to each vintage in the chain, which avoided undesirable disequilibrium dynamics at the onset of the simulation and improved the optimization.

If you want to modify the classical SD DELAY formulation to account for situations where stocks were front-loaded due to historic growth, you would only have to replace the equation for the Initial Level provided in Table 2 with that provided in Table 3 (the difference relative to the classical DELAY formulation is highlighted in bold).[17] Refer to Appendix 1 for a derivation of this formula (see (1.15)), and to Appendix 2 for a more generalized derivation of an aging chain that permits having a unique lifetime for each stock in the aging chain and that also permits an additional outflow from each stock in the aging chain (e.g., a hazard rate or other similar outflow mechanism).

Variable	Analytica Equation	Units
Initial_Level	Input[@Time = 1] * DL * **(1 + Historic_Growth_Rate * DL)^(-Vintage)**	Units

Table 3. Modified Initialization of Stocks in a Material Delay with Historic Growth (See Derivation in Appendix 1)

[16] https://www.vensim.com/documentation/index.html?fn_delay_material.htm

[17] Notice the equation simplifies to the classical SD DELAY formulation when the historic_growth_rate = 0.

Other Advantages of Classical SD Software

Classical SD software platforms offer several other advantages over Analytica for dynamic modeling, such as:

- causal loop tracing (to facilitate understanding feedbacks, loop dominance, etc.),
- partial dynamic simulation (i.e., excluding certain loops),
- immediate simulation sensitivity viewing (e.g., through SyntheSim in Vensim), and
- automated units checking.

Conclusion

Though modeling projects applying System Dynamics often focus on dynamic complexity (i.e., incorporation of all relevant feedback and delays) and attempt to avoid detail complexity, practical applications of it often require granularity and methods that can be challenging or impossible to implement using classical SD software tools. Analytica is a powerful, flexible, multi-method tool that maintains the user-friendliness of classical SD software while enabling the modeler to easily disaggregate the analysis, integrate complex optimizations along different time scales and dimensions, and address uncertainty with an ease and flexibility not currently available in other platforms. I recommend that System Dynamics modelers explore what is possible in Analytica and add it to their arsenal of tools used for solving real-world problems. I expect that modelers will be surprised with how easily one can build a complex System Dynamics model in the software with the information provided in this paper, coupled with a short tutorial of the basics of Analytica. I acknowledge that classical SD software may be preferred in some applications due to its focus on analyzing complex dynamics, and consider that the appropriate tool for each model depends on the system being modeled and the problem to be solved. As such, I suggest that Analytica can complement classical SD modeling software, rather than replace it, and hope that this paper will facilitate expanding the capabilities of the System Dynamics community.

References

Bass, Frank M. 1969. 'A New Product Growth Model For Consumer Durables', *Management Science*, 15: 215-27.

Little, John D. C. . 1961. 'A Proof for the Queuing Formula: L= λW', *Operations Research*, 9: 383-87.

NPCC. 2016. Northwest Power and Conservation Council, Accessed December 26, 2016. http://www.nwcouncil.org/energy/rpm/rpmonline.

NREL. 2016. 'Welcome to SEDS.', Accessed December 26, 2016. https://seds.nrel.gov/.

Sterman, John D. 1987. 'Expectation Formation in Behavioral Simulation Models', *Behavioral Science*, 32: 190-211.

———. 2000. *Business Dynamics : Systems Thinking and Modeling for a Complex World* (Irwin/McGraw-Hill: Boston).

Struben, Jeroen. 2006. "Identifying challenges for sustained adoption of alternative fuel vehicles and infrastructure." In *Proceedings of the 24th International Conference of the System Dynamics Society*, 119. Nijmegen, The Netherlands: The System Dynamics Society.

Welch, Cory. 2006. "Lesson Learned from Alternative Transportation Fuels: Modeling Transition Dynamics." In. Golden, CO: National Renewable Energy Laboratory.

Welch, Cory and Rogers, Brad. 2010. "Estimating the Remaining Useful Life of Residential Appliances." In *ACEEE Summer Study on Energy Efficiency in Buildings*, 2 316-27. Monterey, CA.

Appendix 1:

Derivation of Initialization of an Aging Chain (or Material Delay) Undergoing Steady Growth

Consider and Nth-order aging chain, or material delay, as illustrated in Figure A-1.

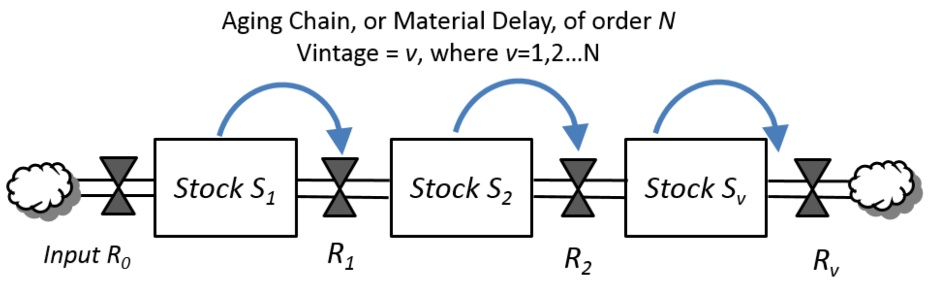

Aging Chain, or Material Delay, of order N
Vintage = v, where v=1,2...N
Stock S1
Stock S2
Stock Sv
Input R0
R1
R2
Rv

Figure A-1. Illustration of an Nth-order Aging Chain, or Material Delay

I define a state of steady-state-growth (SSG) of an aging chain as the point where all inflows, outflows, and stocks are growing at a constant fractional growth rate per year, *g*.

Thus, by definition, during SSG, the first derivative of Stock S_v is the stock times the growth rate *g*, or:

$$\frac{dS_v}{dt} = g \bullet S_v \text{ , where } v = 1,2,...N \tag{1.1}$$

Additionally, at all times, the first derivative of Stock S_v is equal to the difference between the inflow rate and the outflow rate, or:

$$\frac{dS_v}{dt} = R_{v-1} - R_v \tag{1.2}$$

Setting the right-hand-side (RHS) of (1.1) equal to the RHS of (1.2), and rearranging, we get:

$$\frac{\mathrm{R}_{v-1} - R_v}{g} = S_v \tag{1.3}$$

We also know that each outflow rate is defined as:

$$R_v = \frac{S_v}{(L/N)} \text{ , where } L = \text{mean material lifetime (i.e., average delay time), in years} \tag{1.4}$$

Substituting (1.4) into (1.3), we get:

$$\frac{\dfrac{S_{v-1}}{(L/N)} - \dfrac{S_v}{(L/N)}}{g} = S_v \tag{1.5}$$

Rearranging (1.5) we get the ratio of each stock in the aging chain to its upstream stock:

$$\frac{S_v}{S_{v-1}} = \frac{1}{(1+g \bullet L/N)} \tag{1.6}$$

The ratio of each stock to the first stock in the chain is therefore:

$$\frac{S_v}{S_{v=1}} = \left(\frac{1}{(1+g \bullet L/N)}\right)^{(v-1)} = (1+g \bullet L/N)^{(1-v)} \tag{1.7}$$

If we normalize by arbitrarily setting $S_{v=1}$ = 1, the value for each stock is then:

$$S_v = (1+g \bullet L/N)^{(1-v)}, \text{ when } S_{v=1}=1 \tag{1.8}$$

By definition, the total stock, *TS*, in all *N* vintages in the chain is:

$$TS = \sum_{v=1}^{N} S_v = \sum_{v=1}^{N} (1+g \bullet L/N)^{(1-v)} \tag{1.9}$$

Defining the allocation factor, AF_v, as the ratio of each stock to the total stock, we have:

$$\boxed{AF_v = \frac{S_v}{TS} = \frac{F_v}{\sum_{v=1}^{N} F_v} \text{ , where } F_v = (1+g \bullet L/N)^{(1-v)}} \tag{1.10}$$

To calculate the initial input, R_0, to the aging chain that, in SSG, is consistent with the known total stock, note that the ratio of each stock's outflow to the outflow of the upstream stock is the same as the ratio of each stock to the upstream stock, per (1.6), since each outflow is directly proportional to its stock by a constant factor, per (1.4). Therefore, we know that:

$$\frac{R_0}{R_v} = (1+g \bullet L/N)^v \tag{1.11}$$

Substituting (1.4) into (1.11) and rearranging, we get:

$$R_0 = \frac{S_v}{(L/N)} \bullet (1+g \bullet L/N)^v \tag{1.12}$$

Arbitrarily setting v=1 and substituting $S_{v}=1$ into (1.10), we get:

$$S_{v=1} = \frac{TS}{\sum_{v=1}^{N} F_v} \text{ , where } F_v \text{ is defined per (1.10)} \tag{1.13}$$

One can now substitute (1.13) into (1.12), when $v=1$, and rearrange to get R_0 as a function of the input total stock, *TS*, in SSG.

$$\boxed{R_0 = \frac{TS \bullet \left(\frac{1}{(L/N)} + g \right)}{\sum_{v=1}^{N} F_v}} \text{, where } F_v \text{ is defined per (1.10)} \qquad (1.14)$$

Alternately, one can specify each initial stock vintage, S_v, as a function of a known initial input, R_0 (inflow to the aging chain) by rearranging (1.12):

$$\boxed{S_v = R_0 \bullet (L/N) \bullet (1 + g \bullet L/N)^{-v}} \qquad (1.15)$$

Appendix 2
Derivation of Generalized Initialization of an Aging Chain Undergoing Steady Growth

This appendix derives initial values of an aging chain for a more generalized situation than is provided in Appendix 1. Specifically, it allows for each stock in the aging chain to have a unique lifetime, and it allows for an additional outflow (e.g., a stock-specific hazard rate) to be applied to each stock in the aging chain.

Consider an aging chain with N stocks, each of which can have a unique aging lifetime (L_v) and *other outflow* (O_v), as illustrated in Figure A-1.

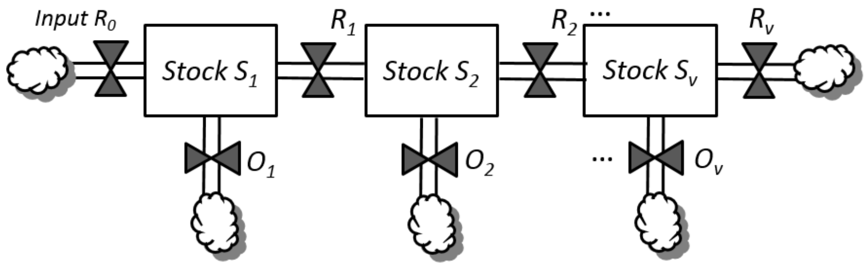


Figure A-1. Illustration of an Aging Chain

I define a state of steady-state-growth (SSG) of an aging chain as the point where all inflows, outflows, and stocks are growing at a constant fractional growth rate per year, *g*.

By definition, during SSG, the first derivative of Stock S_v is the stock times the growth rate *g*, or:

$$\frac{dS_v}{dt} = g \bullet S_v \text{ , where } v = 1, 2, \ldots N \tag{2.1}$$

Additionally, at all times, the first derivative of Stock S_v is equal to the difference between the inflow rate and all outflow rates, or:

$$\frac{dS_v}{dt} = R_{v-1} - R_v - O_v \tag{2.2}$$

Setting the right-hand-side (RHS) of (1.1) equal to the RHS of (1.2), and rearranging, we get:

$$\frac{\mathrm{R}_{v-1} - R_v - O_v}{g} = S_v \tag{2.3}$$

We also know that each aging rate is defined as:

$$R_v = \frac{S_v}{L_v} \text{ ,} \tag{2.4}$$

where L_v = mean aging time of stock *v*.[18]

And, we know by definition that each *other outflow* rate is:

$$O_v = S_v \bullet H_v \ , \tag{2.5}$$

where H_v is the hazard rate (or failure rate), in units of fractional reduction per unit time.[19]

Substituting (2.4) and (2.5) into (1.3), we get:

$$\frac{S_{v-1} / L_{v-1} - S_v / L_v - H_v \bullet S_v}{g} = S_v \tag{2.6}$$

Rearranging (1.5) we get the ratio of each stock in the aging chain to its upstream stock:

$$\frac{S_v}{S_{v-1}} = \frac{1}{L_{v-1} \bullet (g + 1/L_v + H_v)}, \text{ for all } v > 1 \tag{2.7}$$

The ratio of each stock of vintage *v* to the first stock in the chain, which we define to be factor F_v, is therefore:

$$\boxed{F_v \equiv \frac{S_v}{S_1} = \prod_{i=1}^{v} \left(\begin{array}{c} 1, \text{ if } i = 1 \\ \dfrac{1}{L_{v-1} \bullet (g + 1/L_v + H_v)}, \text{ if } i > 1 \end{array} \right)} \tag{2.8}$$

If we normalize by arbitrarily setting $S_1 = 1$, the value for each stock is then:

$$S_v = F_v, \text{ when } S_1 = 1 \tag{2.9}$$

By definition, the total stock, *TS*, in all *N* vintages in the chain is:

$$TS = \sum_{v=1}^{N} S_v = \sum_{x=1}^{N} F_x \ \text{, when } S_1 = 1 \tag{2.10}$$

Defining the allocation factor, AF_v, as the ratio of each stock to the total stock, we have:

$$\boxed{AF_v = \frac{S_v}{TS} = \frac{F_v}{\sum_{x=1}^{N} F_x}} \text{ , where } F \text{ is defined per (2.8)} \tag{2.11}$$

To calculate the initial input, R_0, to the aging chain that, in SSG, is consistent with the known total stock, first substitute (2.5) into (1.3), and then substitute into that result S_v per (2.4) to get:

[18] L_v equals L_T/N for an *N*th-order aging chain of equal delay times per stock vintage, where L_T is the mean total delay time of the entire aging chain.

[19] Note that in some aging chain constructs using a hazard rate, the last aging flow, R_N, is set to zero (meaning the only outflow from the last stock is from the other outflow stream, O_v). This can effectively be accomplished using this set of equations by setting $L_N = \infty$ (or to a sufficiently large number, if your software does not permit specifying infinity as an input).

$$\frac{\mathrm{R}_{v-1} - R_v - H_v \bullet R_v \bullet L_v}{g} = R_v \bullet L_v \tag{2.12}$$

Rearranging, we get the ratio of each inflow to its downstream inflow:

$$\frac{\mathrm{R}_{v-1}}{R_v} = 1 + L_v \bullet (g + H_v) \tag{2.13}$$

Thus, we know the ratio of *R₀* to each *Rᵥ* must also be:

$$\frac{R_0}{R_v} = \prod_{i=1}^{v} (1 + L_v \bullet (g + H_v)) \tag{2.14}$$

Substituting (2.4) into (1.11) and rearranging, we get:

$$R_0 = \frac{S_v}{L_v} \bullet \prod_{i=1}^{v} (1 + L_v \bullet (g + H_v)) \tag{2.15}$$

Arbitrarily setting v=1, substituting S_1 into (1.10), and rearranging, we get:

$$S_1 = \frac{TS}{\sum_{x=1}^{N} F_x} \text{ , where } F \text{ is defined per (2.8)} \tag{2.16}$$

One can now substitute (1.13) into (1.12), when v=1, and isolate R_0.

$$R_0 = \frac{TS}{\sum_{x=1}^{N} F_x} \bullet \frac{1}{L_1} \prod_{i=1}^{1} \left(1 + L_1 \bullet (g + H_1)\right) \tag{2.17}$$

Equation (2.17) simplifies to provide the initial inflow, R_0, as a function of the input total stock, *TS*, in SSG:

$$\boxed{R_0 = \frac{TS}{\sum_{x=1}^{N} F_x} \bullet \left(\frac{1}{L_1} + g + H_1 \right)} \text{ , where } F \text{ is defined per (2.8)} \tag{2.18}$$

Alternately, one can specify each initial stock vintage, S_v, as a function of a known initial inflow, R_0, by rearranging (1.12):

$$\boxed{S_v = R_0 \bullet L_v \bullet \prod_{i=1}^{v} \left(\frac{1}{1 + L_i \bullet (g + H_i)} \right)} \tag{2.19}$$