Appendix A: Introduction to Probabilistic Simulation

Our knowledge of the way things work, in society or in nature, comes trailing clouds of vagueness. Vast ills have followed a belief in certainty.

Kenneth Arrow, *I Know a Hawk from a Handsaw*

Appendix Overview

This appendix provides a very brief introduction to probabilistic simulation (the quantification and propagation of uncertainty). Because detailed discussion of this topic is well beyond the scope of this appendix, readers who are unfamiliar with this field are strongly encouraged to consult additional literature. A good introduction to the representation of uncertainty is provided by Finkel (1990) and a more detailed treatment is provided by Morgan and Henrion (1990). The basic elements of probability theory are discussed in Harr (1987) and more detailed discussions can be found in Benjamin and Cornell (1970) and Ang and Tang (1984).

In this Appendix

This appendix discusses the following:

- Types of Uncertainty
- Quantifying Uncertainty
- Propagating Uncertainty
- A Comparison of Probabilistic and Deterministic Analyses
- Appendix A References

Types of Uncertainty

Many of the features, events and processes which control the behavior of a complex system will not be known or understood with certainty. Although there are a variety of ways to categorize the sources of this uncertainty, for the purpose of this discussion it is convenient to consider the following four types:

- Value (parameter) uncertainty: The uncertainty in the value of a particular parameter (e.g., a physical property, or the development cost of a new product);
- Uncertainty regarding future events: The uncertainty in the ability to predict future perturbations of the system (e.g., a strike, an accident, or an earthquake).
- Conceptual model uncertainty: The uncertainty regarding the detailed understanding and representation of the processes controlling a particular system (e.g., the complex interactions controlling the water flow through the subsurface); and
- Numerical model uncertainty: The uncertainty introduced by approximations in the computational tool used to evaluate the system.

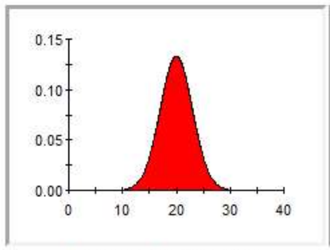
Incorporating these uncertainties into the predictions of system behavior is called ***probabilistic simulation*** or in some applications, ***probabilistic performance assessment***. Probabilistic simulation consists of explicitly representing the uncertainty in the parameters, processes and events controlling the system and propagating this uncertainty through the system such that the uncertainty in the results (i.e., predicted future performance) can be quantified.

Quantifying Uncertainty

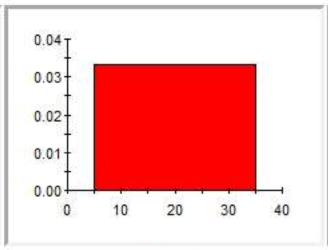
Understanding Probability Distributions

When uncertainty is quantified, it is expressed in terms of ***probability distributions***. A probability distribution is a mathematical representation of the relative likelihood of an uncertain variable having certain specific values.

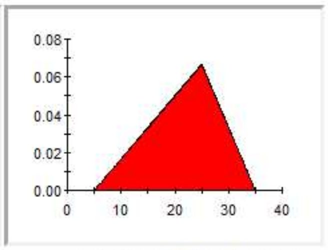
There are many types of probability distributions. Common distributions include the normal, uniform and triangular distributions, illustrated below:



Normal Distribution



Uniform Distribution

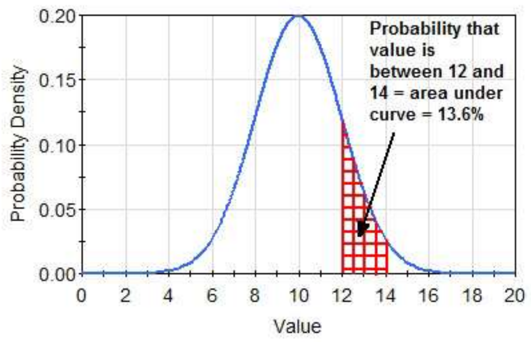


Triangular Distribution

All distribution types use a set of arguments to specify the relative likelihood for each possible value. For example, the normal distribution uses a mean and a standard deviation as its arguments. The mean defines the value around which the bell curve will be centered, and the standard deviation defines the spread of values around the mean. The arguments for a uniform distribution are a minimum and a maximum value. The arguments for a triangular distribution are a minimum value, a most likely value, and a maximum value.

The nature of an uncertain parameter, and hence the form of the associated probability distribution, can be either ***discrete*** or ***continuous***. Discrete distributions have a limited (discrete) number of possible values (e.g., 0 or 1; yes or no; 10, 20, or 30). Continuous distributions have an infinite number of possible values (e.g., the normal, uniform and triangular distributions shown above are continuous). Good overviews of commonly applied probability distributions are provided by Morgan and Henrion (1990) and Stephens et al. (1993).

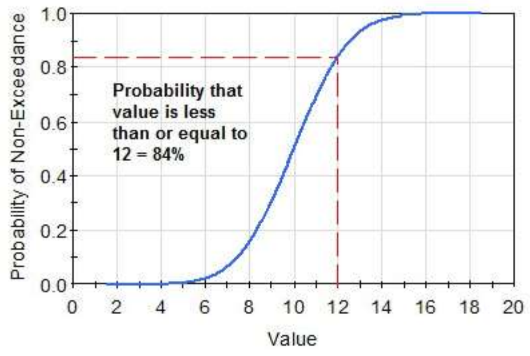
There are a number of ways in which probability distributions can be graphically displayed. The simplest way is to express the distribution in terms of a ***probability density function*** (PDF), which is how the three distributions shown above are displayed. In simple terms, this plots the relative likelihood of the various possible values, and is illustrated schematically below:



Note that the "height" of the PDF for any given value is not a direct measurement of the probability. Rather, it represents the probability density, such that integrating under the PDF between any two points results in the probability of the actual value being between those two points.

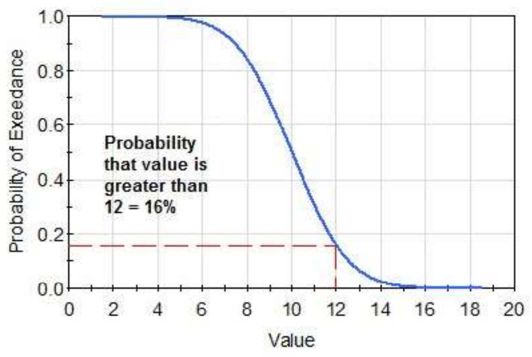
Note: Discrete distributions are described mathematically using ***probability mass functions*** (PMF), rather than probability density functions. Probability mass functions specify actual probabilities for given values, rather than probability densities.

An alternative manner of representing the same information contained in a PDF is the ***cumulative distribution function*** (CDF). This is formed by integrating over the PDF (such that the slope of the CDF at any point equals the height of the PDF at that point). For any value on the horizontal axis, the CDF shows the cumulative probability that the variable will be less than or equal to that value. That is, as shown below, a particular point, say [12, 0.84], on the CDF is interpreted as follows: the probability that the value is less than or equal to 12 is equal to 0.84 (84%):



By definition, the total area under the PDF must integrate to 1.0, and the CDF therefore ranges from 0.0 to 1.0.

A third manner of presenting this information is the complementary cumulative distribution function (CCDF). The CCDF is illustrated schematically below:



A particular point, say [12, 0.16], on the CCDF is interpreted as follows: the probability that the value is greater than 12 is 0.16 (16%). Note that the CCDF is simply the complement of the CDF; that is, in this example 0.84 is equal to 1 – 0.16.

Probability distributions are often described using ***quantiles*** or ***percentiles*** of the CDF. Percentiles of a distribution divide the total frequency of occurrence into hundredths. For example, the 90th percentile is that value of the parameter below which 90% of the distribution lies. The 50th percentile is referred to as the ***median***.

Characterizing Distributions

Probability distributions can be characterized by their ***moments***. The first moment is referred to as the ***mean*** or ***expected value***, and is typically denoted as μ. For a continuous distribution, it is computed as follows:

$$\mu = \int \mathbf{x}\, \mathbf{f(x)}\, \mathbf{dx}$$

where f(x) is the probability density function (PDF) of the variable. For a discrete distribution, it is computed as:

$$\mu = \sum_{i=1}^{N} \mathbf{x_i p(x_i)}$$

where $p(x_i)$ is the probability of x_i, and N is the total number of discrete values in the distribution.

Additional moments of a distribution can also be computed. The n^{th} moment of a continuous distribution is computed as follows:

$$\mu_n = \int (\mathbf{x} - \mu)^n\, \mathbf{f(x)}\, \mathbf{dx}$$

For a discrete distribution, the n^{th} moment is computed as:

$$\mu_n = \sum_{i=1}^{N} (\mathbf{x_i} - \mu)^n\, \mathbf{p(x_i)}$$

The second moment is referred to as the ***variance***, and is typically denoted as σ^2. The square root of the variance, σ, is referred to as the ***standard deviation***. The variance and the standard deviation reflect the amount of spread or dispersion in the distribution. The ratio of the standard deviation to the mean provides a dimensionless measure of the spread, and is referred to as the ***coefficient of variation***.

The ***skewness*** is a dimensionless number computed based on the third moment:

$$\textbf{skewness} = \frac{\mu_3}{\sigma^3}$$

The skewness indicates the symmetry of the distribution. A normal distribution (which is perfectly symmetric) has a skewness of zero. A positive skewness indicates a shift to the right (and example is the log-normal distribution). A negative skewness indicates a shift to the left.

The ***kurtosis*** is a dimensionless number computed based on the fourth moment:

$$\textbf{kurtosis} = \frac{\mu_4}{\sigma^4}$$

The kurtosis is a measure of how "fat" a distribution is, measured relative to a normal distribution with the same standard deviation. A normal distribution has a kurtosis of zero. A positive kurtosis indicates that the distribution is more "peaky" than a normal distribution. A negative kurtosis indicates that the distribution is "flatter" than a normal distribution.

Specifying Probability Distributions

Given the fact that probability distributions represent the means by which uncertainty can be quantified, the task of quantifying uncertainty then becomes a matter of assigning the appropriate distributional forms and arguments to the uncertain aspects of the system. Occasionally, probability distributions can be defined by fitting distributions to data collected from experiments or other data collection efforts. For example, if one could determine that the uncertainty in a particular parameter was due primarily to random measurement errors, one might simply attempt to fit an appropriate distribution to the available data.

Most frequently, however, such an approach is not possible, and probability distributions must be based on subjective assessments (Bonano et al., 1989; Roberds, 1990). Subjective assessments are opinions and judgments about probabilities, based on experience and/or knowledge in a specific area, which are consistent with available information. The process of developing these assessments is sometimes referred to as expert elicitation. Subjectively derived probability distributions can represent the opinions of individuals or of groups. There are a variety of methods for developing subjective probability assessments, ranging from simple informal techniques to complex and time-consuming formal methods. It is beyond the scope of this document to discuss these methods. Roberds (1990), however, provides an overview, and includes a list of references. Morgan and Henrion (1990) also provide a good discussion on the topic.

A key part of all of the various approaches for developing subjective probability assessments is a methodology for developing (and justifying) an appropriate probability distribution for a parameter in a manner that is logically and mathematically consistent with the level of available information. Discussions on the applicability of various distribution types are provided by Harr (1987, Section 2.5), Stephens et al. (1993), and Seiler and Alvarez (1996). Note that methodologies (Bayesian updating) also exist for updating an existing probability distribution when new information becomes available (e.g., Dakins, et al., 1996).

Correlated Distributions

Frequently, parameters describing a system will be correlated (inter-dependent) to some extent. For example, if one were to plot frequency distributions of the height and the weight of the people in an office, there would likely be some degree of positive correlation between the two: taller people would generally also be heavier (although this correlation would not be perfect).

The degree of correlation can be measured using a correlation coefficient, which varies between 1 and -1. A correlation coefficient of 1 or -1 indicates perfect positive or negative correlation, respectively. A positive correlation indicates that the parameters increase or decrease together. A negative correlation indicates that increasing one parameter decreases the other. A correlation coefficient of 0 indicates no correlation (the parameters are apparently independent of each other). Correlation coefficients can be computed based on the actual values of the parameters (which measures linear relationships) or the rank-order of the values of the parameters (which can be used to measure non-linear relationships).

One way to express correlations in a system is to directly specify the correlation coefficients between various model parameters. In practice, however, assessing and quantifying correlations in this manner is difficult. Oftentimes, a more practical way of representing correlations is to explicitly model the cause of the dependency. That is, the analyst adds detail to the model such that the underlying functional relationship causing the correlation is directly represented.

For example, one might be uncertain regarding the solubility of two contaminants in water, while knowing that the solubilities tend to be correlated. If the main source of this uncertainty was actually uncertainty in pH conditions, and the solubility of each contaminant was expressed as a function of pH, the distributions of the two solubilities would then be explicitly correlated. If both solubilities increased or decreased with increasing pH, the correlation would be positive. If one decreased while one increased, the correlation would be negative.

Ignoring correlations, particularly if they are very strong (i.e., the absolute value of the correlation coefficient is close to 1) can lead to physically unrealistic simulations. In the above example, if the solubilities of the two contaminants were positively correlated (e.g., due to a pH dependence), it would be physically inconsistent for one contaminant's solubility to be selected from the high end of its possible range while the other's was selected from the low end of its possible range. Hence, when defining probability distributions, it is critical that the analyst determine whether correlations need to be represented.

Variability and Ignorance

When quantifying the uncertainty in a system, there are two fundamental causes of uncertainty which are important to distinguish: 1) that due to inherent variability; and 2) that due to ignorance or lack of knowledge. IAEA (1989) refers to the former as "Type A uncertainty" and the latter as "Type B uncertainty". These are also sometimes referred to as ***aleatory*** and ***epistemic*** uncertainty, respectively.

Aleatory uncertainty results from the fact that many parameters are inherently variable (random or noisy) over time such that their behavior can only be described statistically. Examples include the flow rate in a river, the price of a stock or the temperature at a particular location.

Variability in a parameter can be expressed using ***frequency distributions***. A frequency distribution displays the relative frequency of a particular value versus the value. For example, one could sample the flow rate of a river once an hour for a week, and plot a frequency distribution of the hourly flow rate (the x-axis being the flow rate, and the y-axis being the frequency of the observation over the week).

Other parameters are not inherently variable over time, but cannot be specified precisely due to epistemic uncertainty: we lack sufficient information or knowledge to specify their value with certainty. Examples include the strength of a particular material, the mass of a planet, or the efficacy of a new drug.

A fundamental difference between these two types of uncertainty is that epistemic uncertainty (i.e., resulting from lack of knowledge) can theoretically be reduced by studying the parameter or system. That is, since the variability is due to a lack of knowledge, theoretically that knowledge could be improved by carrying out experiments, collecting data or doing research. Aleatory uncertainty, on the other hand, is inherently irreducible. If the parameter itself is inherently variable, studying the parameter further will certainly not do anything to change that variability. This is important because one of the key purposes of probabilistic simulation modeling is not just to make predictions, but to identify those parameters that are contributing the most to the uncertainty in results. If the uncertainty in the results is due primarily to epistemic parameters, we know that we could (at least theoretically) reduce our uncertainty in our results by gaining more information about those parameters.

It should be noted that parameters which have both kinds of uncertainty are not uncommon in simulation models. For example, in considering the flow rate in a river, we know that it will be temporally variable (inherently random in time so it can only be described statistically), but in the absence of adequate data, we will have uncertainty about the statistical measures (e.g., mean, standard deviation) describing that variability. By taking measurements, we can reduce our uncertainty in these statistical measures (i.e., what is the mean flow rate?), but we will not be able to reduce the inherent variability in the flow.

Note that some quantities are variable not over time, but over space or within a collection of items or instances. An example is the age of population. If you had a group of 1000 individuals, you could obtain the age of each individual and create a frequency distribution of the age of the group. This kind of distribution is similar to the example of the flow rate in a river discussed above in that both are described using frequency distributions (one showing a frequency in time, and one showing a frequency of occurrence within a group). The age example, however, is fundamentally different from an inherently random parameter. Whereas a distribution representing an inherently random parameter truly is describing uncertainty (we cannot predict the value at any given time), a distibution representing the age distribution is not describing uncertainty at all. It is simply describing a variability within the group that we could actually measure and define very precisely.

It is critical not to combine variability like this with uncertainty and represent both using a single distribution. For example, suppose that you needed to represent the efficacy of a new drug. The efficacy is different for different age groups. Moreover, for each age group, there is scientific uncertainty regarding its efficacy. A common mistake would be to define a single probability distribution that represents both the variability due to age and the uncertainty due to lack of knowledge. Not only would it be difficult to define the shape of such a distribution in the first place, this would produce simulation results that would be difficult, if not impossible, to interpret in a meaningful way. The correct way to handle such a situation would be to disaggregate the problem (by explicitly modeling each age group separately) and then define different probability distributions for each age group (with each distribution representing only the scientific uncertainty in the efficacy for that age group).

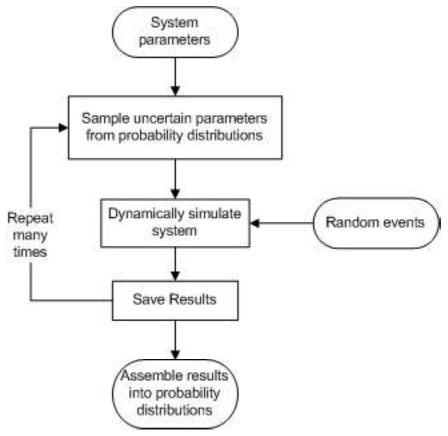
Propagating Uncertainty

If the inputs describing a system are uncertain, the prediction of the future performance of the system is necessarily uncertain. That is, the result of any analysis based on inputs represented by probability distributions is itself a probability distribution.

In order to compute the probability distribution of predicted performance, it is necessary to propagate (translate) the input uncertainties into uncertainties in the results. A variety of methods exist for propagating uncertainty. Morgan and Henrion (1990) provide a relatively detailed discussion on the various methods.

One common technique for propagating the uncertainty in the various aspects of a system to the predicted performance (and the one used by GoldSim) is ***Monte Carlo simulation***. In Monte Carlo simulation, the entire system is simulated a large number (e.g., 1000) of times. Each simulation is equally likely, and is referred to as a ***realization*** of the system. For each realization, all of the uncertain parameters are sampled (i.e., a single random value is selected from the specified distribution describing each parameter). The system is then simulated through time (given the particular set of input parameters) such that the performance of the system can be computed.

This results in a large number of separate and independent results, each representing a possible “future” for the system (i.e., one possible path the system may follow through time). The results of the independent system realizations are assembled into probability distributions of possible outcomes. A schematic of the Monte Carlo method is shown below:


System parameters
Sample uncertain parameters from probability distributions
Dynamically simulate system
Random events
Repeat many times
Save Results
Assemble results into probability distributions

A Comparison of Probabilistic and Deterministic Simulation Approaches

Having described the basics of probabilistic analysis, it is worthwhile to conclude this appendix with a comparison of probabilistic and ***deterministic*** approaches to simulation, and a discussion of why GoldSim was designed to specifically facilitate both of these approaches.

In the deterministic simulation approach, the analyst, although they may implicitly recognize the uncertainty in the various input parameters, selects single values for each parameter. Typically, these are selected to be "best estimates" or sometimes "worst case estimates". These inputs are evaluated using a simulation model, which then outputs a single result, which presumably represents a "best estimate" or "worst case estimate".

On the other hand, in a probabilistic simulation approach, the analyst explicitly represents the input parameters as probability distributions, and propagates the uncertainty through to the result (e.g., using the Monte Carlo method), such that the result itself is also a probability distribution.

One advantage of deterministic analyses is that they can typically incorporate more detailed components than probabilistic analyses due to computational considerations (since complex probabilistic analyses generally require time-consuming simulation of multiple realizations of the system).

Deterministic analyses, however, have a number of disadvantages:

- "Worst case" deterministic simulations can be extremely misleading. Worst case simulations of a system may be grossly conservative and therefore completely unrealistic (i.e., they typically have an extremely low probability of actually representing the future behavior of the system). Moreover, it is not possible in a deterministic simulation to quantify how conservative a "worst case" simulation actually is. Using a highly improbable simulation to guide policy making (e.g., "is the design safe?") is likely to result in poor decisions.
- "Best estimate" deterministic simulations are often difficult to defend. Because of the inherent uncertainty in most input parameters, defending "best estimate" parameters is often very difficult. In a confrontational environment, "best estimate" analyses will typically evolve into "worst case" analyses.
- Deterministic analyses do not lend themselves directly to detailed uncertainty and sensitivity studies. In order to carry out uncertainty and sensitivity analysis of deterministic simulations, it is usually necessary to carry out a series of separate simulations in which various parameters are varied. This is time-consuming and typically results only in a limited analysis of sensitivity and uncertainty.

These disadvantages do not exist for probabilistic analyses. Rather than facing the difficulties of defining worst case or best estimate inputs, probabilistic analyses attempt to explicitly represent the full range of possible values. The probabilistic approach embodied within GoldSim acknowledges the fact that for many complex systems, predictions are inherently uncertain and should always be presented as such. Probabilistic analysis provides a means to present this uncertainty in a quantitative manner.

Moreover, the output of probabilistic analyses can be used to directly determine parameter sensitivity. Because the output of probabilistic simulations consists of multiple sets of input parameters and corresponding results, the sensitivity of results to various input parameters can be directly determined. The fact that probabilistic analyses lend themselves directly to evaluation of parameter sensitivity is one of the most powerful aspects of this approach, allowing such tools to be used to aid decision-making.

There are, however, some potential disadvantages to probabilistic analyses that should also be noted:

- Probabilistic analyses may be perceived as unnecessarily complex and difficult to understand. Although this sentiment is gradually becoming less prevalent as probabilistic analyses become more common, it cannot be ignored. It is therefore important to develop and present probabilistic analyses in a manner that is straightforward and transparent. In fact, GoldSim was specifically intended to minimize this concern.
- The process of developing input for a probabilistic analysis can sometimes degenerate into futile debates about the "true" probability distributions. This concern can typically be addressed by simply repeating the probabilistic analysis using alternative distributions. If the results are similar, then there is not necessity to pursue the "true" distributions further.
- The public (courts, media, etc.) typically does not fully understand probabilistic analyses and may be suspicious of it. This may improve as such analyses become more prevalent and the public is educated, but is always likely to be a problem. As a result, complementary deterministic simulations will sometimes be required in order to illustrate the performance of the system under a specific set of conditions (e.g., "expected" or "most likely" conditions).

As this last point illustrates, it is important to understand that use of a probabilistic analysis does not preclude the use of deterministic analysis. In fact, deterministic analyses of various system components are often essential in order to provide input to probabilistic analyses. The key point is that for many systems, deterministic analyses alone can have significant disadvantages and in these cases, they should be complemented by probabilistic analyses.

Appendix A References

The references cited in this appendix are listed below.

Ang, A. H-S. and W.H. Tang, 1984, Probability Concepts in Engineering Planning and Design, Volume II: Decision, Risk, and Reliability, John Wiley & Sons, New York.

Bonano, E.J., S.C. Hora, R.L. Keaney and C. von Winterfeldt, 1989, Elicitation and Use of Expert Judgment in Performance Assessment for High-Level Radioactive Waste Repositories, Sandia Report SAND89-1821, Sandia National Laboratories.

Benjamin, J.R. and C.A. Cornell, 1970, Probability, Statistics, and Decision for Civil Engineers, McGraw-Hill, New York.

Dakins, M.E., J.E. Toll, M.J. Small and K.P. Brand, 1996, *Risk-Based Environmental Remediation: Bayesian Monte Carlo Analysis and the Expected Value of Sample Information*, Risk Analysis, Vol. 16, No. 1, pp. 67-79.

Finkel, A., 1990, Confronting Uncertainty in Risk Management: A Guide for Decision-Makers, Center for Risk Management, Resources for the Future, Washington, D.C.

Harr, M.E., 1987, Reliability-Based Design in Civil Engineering, McGraw-Hill, New York.

IAEA, 1989, Evaluating the Reliability of Predictions Made Using Environmental Transfer Models, IAEA Safety Series No. 100, International Atomic Energy Agency, Vienna.

Morgan, M.G. and M. Henrion, 1990, Uncertainty, Cambridge University Press, New York.

Roberds, W.J., 1990, *Methods for Developing Defensible Subjective Probability Assessments*, Transportation Research Record, No. 1288, Transportation Research Board, National Research Council, Washington, D.C., January 1990.

Seiler, F.A and J.L. Alvarez, 1996, *On the Selection of Distributions for Stochastic Variables*, Risk Analysis, Vol. 16, No. 1, pp. 5-18.

Stephens, M.E., B.W. Goodwin and T.H. Andres, 1993, *Deriving Parameter Probability Density Functions*, Reliability Engineering and System Safety, Vol. 42, pp. 271-291.

Appendix B: Probabilistic Simulation Details

Clever liars give details, but the cleverest don't.

Anonymous

Appendix Overview

This appendix provides the mathematical details of how GoldSim represents and propagates uncertainty, and the manner in which it constructs and displays probability distributions of computed results. While someone who is not familiar with the mathematics of probabilistic simulation should find this appendix informative and occasionally useful, most users need not be concerned with these details. Hence, this appendix is primarily intended for the serious analyst who is quite familiar with the mathematics of probabilistic simulation and wishes to understand the specific algorithms employed by GoldSim.

In this Appendix

This appendix discusses the following:

- Mathematical Representation of Probability Distributions
- Correlation Algorithms
- Sampling Techniques
- Representing Random (Poisson) Events
- Computing and Displaying Result Distributions
- Computing Sensitivity Analysis Measures
- Appendix B References

Mathematical Representation of Probability Distributions

Distributional Forms

The arguments, probability density (or mass) function (PDF or PMF), cumulative distribution function (CDF), and the mean and variance for each of the probability distributions available within GoldSim are presented below.

Beta Distribution

The beta distribution for a parameter is specified by a minimum value (a), a maximum value (b), and two shape parameters denoted S and T. The beta distribution represents the distribution of the underlying probability of success for a binomial sample, where S represents the observed number of successes in a binomial trial of T total draws.

Alternative formulations of the beta distribution use parameters α and β, or α_1 and α_2, where S = α =α_1 and (T-S) = β = α_2.

Frequently the beta distribution is also defined in terms of a minimum, maximum, mean, and standard deviation. The shape parameters are then computed from these statistics.

The beta distribution has many variations controlled by the shape parameters. It is always limited to the interval (a,b). Within (a,b), however, a variety of distribution forms are possible (e.g., the distribution can be configured to behave exponentially, positively or negatively skewed, and symmetrically). The distribution form obtained by different S and T values is predictable for a skilled user.

PDF

$$f(x) = \frac{1}{B(b-a)^{T-1}}(x-a)^{S-1}(b-x)^{T-S-1}$$

where

$$B = \frac{\Gamma(S)\Gamma(T-S)}{\Gamma(T)}$$

and

$$\Gamma(k) = \int_0^\infty e^{-u}u^{k-1}du$$

CDF

$$F(x) = I_x(\alpha, \beta)$$

where $I_x(\alpha,\beta)$ is the regularized incomplete beta function.

Mean

$$\mu = a + \frac{S}{T}(b-a)$$

Variance

$$\sigma^2 = (b-a)^2 \frac{S(T-S)}{T^2(T+1)}$$

Notes

Within GoldSim, there are three ways to define a Beta distribution (three different distribution types from the drop-list defining the distribution):

- Beta Distribution
- Generalized Beta Distribution
- BetaPERT Distribution

You can choose to specify S and T (Beta Distribution).

Alternatively, you can specify a mean, standard deviation, minimum and maximum (Generalized Beta Distribution). In this case, GoldSim limits the standard deviations that can be specified as follows:

$$\sigma^* <= 0.6\sqrt{\mu^*(1-\mu^*)}$$

where

$$\mu^* = \frac{\mu - a}{b-a}$$

and

$$\sigma^* = \frac{\sigma}{b-a}$$

This constraint ensures that the distribution has a single peak and that it does not have a discrete probability mass at either end of its range.

Finally, you can specify a minimum (a), maximum (b) and most likely value (c) (BetaPERT distribution). In this case, GoldSim assumes shape parameters are as follows:

$$\alpha = 1 + 4c_n$$

$$\beta = 5 - 4c_n$$

where

$$c_n = \frac{c-a}{b-a}$$

Note that in this case, the most likely value specified by the user is not mathematically the most likely value (but is a very close approximation to it).

Note that if the BetaPERT is defined using the 10th and 90th percentile (instead of a minimum and a maximum) the minimum and maximum are estimated through iteration.

Binomial Distribution

The binomial distribution is a discrete distribution specified by a batch size (n) and a probability of occurrence (p). This distribution can be used to model the number of parts that failed from a given set of parts, where n is the number of parts and p is the probability of the part failing.

PMF

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x} \qquad x = 0, 1, 2, 3\ldots$$

where

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}$$

CDF

$$F(x) = \sum_{i=0}^{x} \binom{n}{i} p^i (1-p)^{n-i}$$

Mean

$$\mu = np$$

Variance

$$\sigma^2 = np(1-p)$$

Boolean Distribution

The Boolean (or logical) distribution is a discrete distribution that requires a single input: the probability of being true, p. The distribution takes on one of two values: False (0) or True (1).

PMF

$$P(x) = \begin{cases} 1-p & x = 0 \\ p & x = 1 \end{cases}$$

CDF

$$F(x) = \begin{cases} 1-p & x = 0 \\ 1 & x = 1 \end{cases}$$

Mean

$\mu = p$

Variance

$\sigma^2 = p(1\text{-}p)$

Cumulative Distribution

The cumulative distribution enables the user to input a piece-wise linear cumulative distribution function by simply specifying value (xi) and cumulative probability (pi) pairs.

GoldSim allows input of an unlimited number of pairs, x_i, p_i. In order to conform to a cumulative distribution function, it is a requirement that the first probability equal 0 and the last equal 1. The associated values, denoted x_0 and x_n, respectively, define the minimum value and maximum value of the distribution.

PDF

$$f(x) = \begin{cases} 0 & x \le x_0 \text{ or } x \ge x_n \\ \dfrac{p_{i+1} - p_i}{x_{i+1} - x_i} & x_i \le x \le x_{i+1} \end{cases}$$

CDF

$$F(x) = \begin{cases} 0 & x \le x_0 \\ p_i + (p_{i+1} - p_i)\frac{x-x_i}{x_{i+1}-x_i} & x_i \le x \le x_{i+1} \\ 1 & x \ge x_n \end{cases}$$

Mean

$$\mu \approx \sum_{i=1}^{n-1} \frac{(x_{i+1} + x_i)(p_{i+1} - p_i)}{2}$$

Variance2

$$\sigma^2 \approx \sum_{i=1}^{n-1} \frac{(x_{i+1}^3 - x_i^3)}{3} f(x_i) - \mu^2$$

Log-Cumulative Distribution

The log-cumulative distribution enables the user to input a piece-wise logarithmic cumulative distribution function by simply specifying value (xi) and cumulative probability (pi) pairs. Whereas in a cumulative distribution, the density between values is constant (i.e., the distribution between values is uniform), in a log-cumulative, the density of the log of the value is constant (i.e., the distribution between values is log-uniform).

The log-cumulative distribution enables the user to input a piece-wise logarithmic cumulative distribution function by simply specifying value (x_i) and cumulative probability (p_i) pairs. Whereas in a cumulative distribution, the density between values is constant (i.e., the distribution between values is uniform), in a log-cumulative, the density of the log of the value is constant (i.e., the distribution between values is log-uniform).

GoldSim allows input of an unlimited number of pairs, xi, pi. In order to conform to a cumulative distribution function, it is a requirement that the first probability equal 0 and the last equal 1. The associated values, denoted x_0 and x_n, respectively, define the minimum value and maximum value of the distribution. Also, all values must be positive.

PDF

$$f(x) = \begin{cases} 0 & x \le x_0 \text{ or } x \ge x_n \\ \dfrac{p_{i+1} - p_i}{x \ln(x_{i+1}/x_i)} & x_i \le x \le x_{i+1} \end{cases}$$

CDF

$$F(x) = \begin{cases} 0 & x \le x_0 \\ p_i \dfrac{\ln(x_{i+1}/x)}{\ln(x_{i+1}/x_i)} + p_{i+1} \dfrac{\ln(x/x_i)}{\ln(x_{i+1}/x_i)} & x_i \le x \le x_{i+1} \\ 1 & x \ge x_n \end{cases}$$

Mean

$$\mu \approx \sum_{i=1}^{n-1} \frac{(p_{i+1} - p_i)(x_{i+1} - x_i)}{\ln(x_{i+1}/x_i)}$$

Variance

$$\sigma^2 \approx \sum_{i=1}^{n-1} \frac{(p_{i+1} - p_i)(x_{i+1}^2 - x_i^2)}{2\ln(x_{i+1}/x_i)} - \mu^2$$

Discrete Distribution

The discrete distribution enables the user to directly input a probability mass function for a discrete parameter. Each of the n discrete values, x_i, that may be assigned to the parameter, has an associated probability, p_i, indicating its likelihood to occur. To conform to the requirements of a probability mass function, the sum of the probabilities, p_i, must equal 1. The discrete distribution is commonly used for situations with a relatively small number of possible outcomes.

PMF

$$P(x_i) = p_i$$

CDF

$$F(x_i) = \sum_{j=1}^{i} p_j$$

Mean

$$\mu = \sum_{i=1}^{n} x_i p_i$$

Variance

$$\sigma^2 = \sum_{i=1}^{n} x_i^2 p_i - \mu^2$$

Exponential Distribution

The Exponential distribution is a continuous distribution specified by a mean value (μ) which must be positive. This distribution is typically used to model the time required to complete a task or achieve a milestone.

PDF

$$f(x) = \begin{cases} \frac{1}{\mu} e^{-x/\mu} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

CDF

$$F(x) = \begin{cases} 1 - e^{-x/\mu} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Mean

μ

Variance

μ^2

Extreme Probability Distribution

The Extreme Probability distribution provides the expected extreme probability level of a uniform distribution given a specific number of samples (n). Because it represents a probability, by definition it has a minimum (a) of 0 and maximum (b) of 1. The distribution is specified as either a Minimum or a Maximum Extreme Probability Distribution. These are equivalent to the Beta distribution with the following parameters:

Parameter	Maximum	Minimum
S	n	1
(T-S)	1	n

PDF

$$f(x) = \frac{1}{B(b-a)^{T-1}} (x-a)^{S-1} (b-x)^{T-S-1}$$

where

$$B = \frac{\Gamma(S)\,\Gamma(T-S)}{\Gamma(T)}$$

and

$$\Gamma(k) = \int_0^\infty e^{-u} u^{k-1} du$$

CDF

No closed form

Mean

$$\mu = a + \frac{S}{T}(b - a)$$

Variance

$$\sigma^2 = (b - a)^2 \frac{S(T - S)}{T^2(T + 1)}$$

Extreme Value Distribution

The Extreme Value distribution (also known as the Gumbel distribution) is used to represent the maximum or minimum expected value of a variable. It is specified with the Location (m) and a Scale parameter (s). The Scale parameter must be positive. The distribution is specified as either a Minimum or a Maximum Extreme Value Distribution.

PDF (Maximum)

$$f(x) = \frac{z}{s}e^{-z}$$

where

$$z = e^{\frac{-(x-m)}{s}}$$

PDF (Minimum)

$$f(x) = \frac{z}{s}e^{-z}$$

where

$$z = e^{\frac{(x-m)}{s}}$$

CDF (Maximum)

$$F(x) = e^{-z}$$

CDF (Minimum)

$$F(x) = 1 - e^{-z}$$

Mean (Maximum)

$$\mu = m + 0.57722\,s$$

Mean (Minimum)

$$\mu = m - 0.57722\,s$$

Variance (Maximum and Minimum)

$$\sigma^2 = \frac{(s\pi)^2}{6}$$

Gamma Distribution

The gamma distribution is most commonly used to model the time to the kth event, when such an event is modeled by a Poisson process with rate parameter λ. Whereas the Poisson distribution is typically used to model the number of events in a period of given length, the gamma distribution models the time to the kth event (or alternatively, the time separating the kth and kth+1 events).

The gamma distribution is specified by the Poisson rate variable, λ, and the event number, k. The random variable, denoted as x, is the time period to the kth event. Within GoldSim, the gamma distribution is specified by the mean (μ) and the standard deviation (σ). λ and k can be expressed in terms of these as follows:

$$k = \frac{\mu^2}{\sigma^2}$$

$$\lambda = \frac{\mu}{\sigma^2}$$

PDF

$$f(x) = \frac{\lambda(\lambda x)^{k-1} e^{-\lambda x}}{\Gamma(k)}$$

where

$$\Gamma(k) = \int_0^\infty e^{-u} u^{k-1} du \quad \text{(gamma function)}$$

CDF

$$F(x) = \frac{\Gamma(k, \lambda x)}{\Gamma(k)}$$

where

$$\Gamma(k, x) = \int_0^x e^{-u} u^{k-1} du \quad \text{(incomplete gamma function)}$$

Mean

μ (specified directly)

Variance

σ^2 (σ specified directly)

Notes

If k is near zero, the distribution is highly skewed. For k=1, the gamma distribution reduces to an exponential distribution with mean of 1/λ. If k = n/2 and λ= ½, the distribution is known as a chi-squared distribution with n degrees of freedom.

Note: If the mean value (in terms of SI units) is less than 1E-13 or the ratio of the standard deviation to the mean is less than 0.1, the gamma is approximated as a normal distribution (truncated such that it is never less than 0).

Negative Binomial Distribution

The negative binomial distribution is a discrete distribution specified by a number of successes (n), which can be fractional, and a probability of success (p). This distribution can be used to model the number of failures that occur when trying to achieve a given number of successes, and is used frequently in actuarial models.

PMF

$$P(x) = \binom{x+n-1}{x} p^n (1-p)^x \qquad x = 0, 1, 2, 3\ldots$$

where

$$\binom{x+n-1}{x} = \frac{(x+n-1)!}{x!(n-1)!}$$

CDF

$$F(x) = p^n \sum_{i=0}^{x} \binom{i+n-1}{i} (1-p)^i$$

Mean

$$\frac{n(1-p)}{p}$$

Variance

$$\frac{n(1-p)}{p^2}$$

Normal Distribution

The normal distribution is specified by a mean (μ) and a standard deviation (σ). The linear normal distribution is a bell shaped curve centered about the mean value with a half-width of about four standard deviations. Error or uncertainty that can be higher or lower than the mean with equal probability may be satisfactorily represented with a normal distribution. The uncertainty of average values, such as a mean value, is often well represented by a normal distribution, and this relation is further supported by the Central Limit Theorem for large sample sizes.

PDF

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

CDF

No closed form solution

Mean

μ (specified directly)

Variance

σ^2 (σ specified directly)

Log-Normal Distribution

The log-normal distribution is used when the logarithm of the random variable is described by a normal distribution. The log-normal distribution is often used to describe environmental variables that must be positive and are positively skewed.

In GoldSim, the log-normal distribution may be based on either the true (arithmetic) mean (μ) and standard deviation (σ), or on the geometric mean (identical to the median) and the geometric standard deviation. Thus, if the variable x is distributed log-normally, the mean and standard deviation of log x may be used to characterize the log-normal distribution. (Note that either base 10 or base e logarithms may be used).

PDF

$$f(x) = \frac{1}{\zeta x\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\ln(x)-\lambda}{\zeta}\right)^2}$$

where

ζ is referred to as the shape factor.

$$\zeta^2 = \ln\left[1 + \left(\frac{\sigma}{\mu}\right)^2\right] \quad \text{(variance of ln x)}$$

$$\lambda = \ln(\mu) - \frac{1}{2}\zeta^2 \quad \text{(expected value of ln x)}$$

CDF

No closed form solution

Mean (Arithmetic)

$$\mu = \exp\left[\lambda + \frac{1}{2}\zeta^2\right]$$

The mean computed by the above formula is the expected value of the log-normally distributed variable x and is a function of the mean and standard deviation of ln(x). The mean value can be estimated by the arithmetic mean of a sample data set.

Variance (Arithmetic)

$$\sigma^2 = \mu^2 \left[\exp(\zeta^2) - 1\right]$$

The variance computed by the above formula is the variance of the log-normally distributed variable x. It is a function of the mean of x and the standard deviation of ln(x). The variance of x can be estimated by the sample variance computed arithmetically.

Notes

Other useful formulas:

$$\text{Geometric mean} = e^{\lambda}$$

$$\text{Geometric standard deviation} = e^{\zeta}$$

A commonly used descriptor for a log-normal distribution is its Error Factor (EF), where the EF is defined as (geometric standard deviation) ^ 1.645.

90% of the distribution lies between Median/EF and Median*EF.

Pareto Distribution

The Pareto distribution is a continuous, long-tailed distribution specified by a shape factor (a) and a scale (b). The scale is both the minimum value and the mode. The shape parameter and the scale parameter must be greater than zero. This distribution can be used to model things like network traffic in a telecommunications system or income levels in a particular country.

PDF

$$f(x) = \begin{cases} \dfrac{ab^a}{x^{a+1}} & x \geq b \\ 0 & x < b \end{cases}$$

CDF

$$F(x) = \begin{cases} 1 - \left(\dfrac{b}{x}\right)^a & x \geq b \\ 0 & x < b \end{cases}$$

Mean

$$\mu = \frac{ab}{a-1}$$

Variance

$$\sigma^2 = \frac{ab^2}{(a-1)^2(a-2)}$$

Pearson Type III Distribution

Often used in financial and environmental modeling, the Pearson Type III distribution is a continuous distribution specified by location (α), scale (β) and shape (p) parameters. Both the scale and shape parameters must be positive.

Note that the Pearson Type III distribution is equivalent to a gamma distribution if the location parameter is set to zero.

PDF

$$f(x) = \begin{cases} \frac{1}{\beta\Gamma(p)}\left(\frac{x-\alpha}{\beta}\right)^{p-1} e^{-\left(\frac{x-\alpha}{\beta}\right)} & x \geq \alpha \\ 0 & x < \alpha \end{cases}$$

where

$$\Gamma(k) = \int_0^\infty e^{-u}u^{k-1}du \quad \text{(gamma function)}$$

CDF

$$F(x) = \frac{\Gamma\left(p, \frac{x-\alpha}{\beta}\right)}{\Gamma(p)}$$

where

$$\Gamma(k,x) = \int_0^x e^{-u}u^{k-1}du \quad \text{(incomplete gamma function)}$$

Mean

$$\mu = \alpha + p\beta$$

Variance

$$\sigma^2 = p\beta^2$$

Poisson Distribution

The Poisson distribution is a discrete distribution specified by a mean value, μ. The Poisson distribution is most often used to determine the probability for one or more events occurring in a given period of time. In this type of application, the mean is equal to the product of a rate parameter, λ, and a period of time, ω. For example, the Poisson distribution could be used to estimate probabilities for numbers of earthquakes occurring in a 100 year period. A rate parameter characterizing the number of earthquakes per year would be needed for input to the distribution. The time period would simply be equal to 100 years.

PMF

$$P(x) = \frac{e^{-\mu}\mu^x}{x!} \qquad x = 0, 1, 2, 3...$$

CDF

$$F(x) = e^{-\mu} \sum_{i=0}^{x} \frac{\mu^i}{i!}$$

Mean

$$\mu = \lambda\omega$$

Variance

$$\sigma^2 = \mu$$

Student's t Distribution

The Student’s t distribution requires a single input: the number of degrees of freedom, ν, which equals the number of samples minus one. It is it symmetric around zero and bell-shaped. It is similar to the standard Normal distribution (mean 0, SD 1), but with heavier tales. As $\nu \to \infty$ it becomes the standard Normal.

PDF

$$f(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma(\nu/2)} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

where

$$\Gamma(k) = \int_0^{\infty} e^{-u} u^{k-1} du \quad \text{(gamma function)}$$

CDF

$$F(x) = \frac{1}{2} + x\Gamma\left(\frac{\nu+1}{2}\right) \frac{{}_2F_1\left(\frac{1}{2}, \frac{\nu+1}{2}; \frac{3}{2}; -\frac{x^2}{\nu}\right)}{\sqrt{\nu\pi}\,\Gamma(\nu/2)}$$

where

${}_2F_1$ is the hypergeometric function.

Mean

$$\mu = 0$$

Variance

$$\sigma^2 = \frac{\nu}{\nu - 2}$$

Sampled Result Distribution

The sampled result distribution allows you to construct a distribution using observed results. GoldSim generates a CDF by sorting the observations and assuming that a cumulative probability of 1/(Number of Observations) exists between each data point. If there are multiple data points at the same value, a discrete probability equal to (N)/(Number of Observations) is applied at the value, where N is equal to the number of identical observations.

If the Extrapolation option is cleared, a discrete probability of 0.5/(Number of observations) is assigned to the minimum and maximum values. When the extrapolation option is selected, GoldSim extends the generated CDF to cumulative probability levels of 0 and 1 using the slope between the two smallest and two largest unique observations.

Triangular Distribution

The triangular distribution is specified by a minimum value (a), a most likely value (b), and a maximum value (c). Note that if the triangular is defined using the 10th and 90th percentile (instead of a minimum and a maximum) the minimum and maximum are estimated through iteration.

PDF

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(b-a)(c-a)} & a \le x \le b \\ \dfrac{2(c-x)}{(c-b)(c-a)} & b < x \le c \\ 0 & x < a \text{ or } x > c \end{cases}$$

CDF

$$F(x) = \begin{cases} 0 & x < a \\ \dfrac{(x-a)^2}{(b-a)(c-a)} & a \le x \le b \\ 1 - \dfrac{(c-x)^2}{(c-b)(c-a)} & b < x \le c \\ 1 & x > c \end{cases}$$

Mean

$$\mu = \frac{a+b+c}{3}$$

Variance

$$\sigma^2 = \frac{a^2+b^2+c^2-ab-ac-bc}{18}$$

Log-Triangular Distribution

The log-triangular distribution is used when the logarithm of the random variable is described by a triangular distribution. The minimum (a), most likely (b), and maximum (c) values are specified in linear space. Note that if the log-triangular is defined using the 10th and 90th percentile (instead of a minimum and a maximum) the minimum and maximum are estimated through iteration.

PDF

$$f(x) = \begin{cases} \dfrac{2\ln(x/a)}{x\ln(b/a)\ln(c/a)} & a \le x \le b \\ \dfrac{2\ln(c/x)}{x\ln(c/a)\ln(c/b)} & b < x \le c \\ 0 & x < a \text{ or } x > c \end{cases}$$

CDF

$$F(x) = \begin{cases} 0 & x < a \\ \dfrac{[\ln(x/a)]^2}{\ln(b/a)\ln(c/a)} & a \le x \le b \\ \dfrac{[\ln(c/x)]^2}{\ln(x/a)\ln(c/b)} & b < x \le c \\ 1 & x > c \end{cases}$$

Mean

$$\mu = \frac{2}{d_1}\{a + b[\ln(b/a) - 1]\} + \frac{2}{d_2}\{c + b[\ln(b/c) - 1]\}$$

Variance

$$\sigma^2 = \frac{2}{d_1}\left\{\frac{a^2}{4} + \frac{b^2}{2}\left[\ln(b/a) - \frac{1}{2}\right]\right\} + \frac{2}{d_2}\left\{\frac{c^2}{4} + \frac{b^2}{2}\left[\ln(b/c) - \frac{1}{2}\right]\right\} - \mu^2$$

where

$$d_1 = \ln(c/a)\ln(b/a)$$

and

$$d_2 = \ln(c/a)\ln(c/b)$$

Uniform Distribution

The uniform distribution is specified by a minimum value (a) and a maximum value (b). Each interval between the endpoints has equal probability of occurrence. This distribution is used when a quantity varies uniformly between two values, or when only the endpoints of a quantity are known.

PDF

$$f(x) = \begin{cases} \dfrac{1}{b - a} & a \le x \le b \\ 0 & x < a \text{ or } x > b \end{cases}$$

CDF

$$F(x) = \begin{cases} 0 & x < a \\ \dfrac{x - a}{b - a} & a \le x \le b \\ 1 & x > b \end{cases}$$

Mean

$$\mu = \frac{b + a}{2}$$

Variance

$$\sigma^2 = \frac{(b - a)^2}{12}$$

Log-Uniform Distribution

The log-uniform distribution is used when the logarithm of the random variable is described by a uniform distribution. Log-uniform is the distribution of choice for many environmental parameters that may range in value over two or more log-cycles and for which only a minimum value and a maximum value can be reasonably estimated. The log-uniform distribution has the effect of assigning equal probability to the occurrence of intervals within each of the log-cycles. In contrast, if a linear uniform distribution were used, only the intervals in the upper log-cycle would be represented uniformly.

PDF

$$f(x) = \begin{cases} \dfrac{1}{x \ln(b/a)} & a \le x \le b \\ 0 & x < a \text{ or } x > b \end{cases}$$

CDF

$$F(x) = \begin{cases} 0 & x < a \\ \dfrac{\ln(x/a)}{\ln(b/a)} & a \le x \le b \\ 1 & x > b \end{cases}$$

Mean

$$\mu = \frac{b - a}{\ln(b/a)}$$

Variance

$$\sigma^2 = \frac{b^2 - a^2}{2 \ln(b/a)} - \left(\frac{b - a}{\ln(b/a)}\right)^2$$

Weibull Distribution

The Weibull distribution is often used to characterize failure times in reliability models. However, it can be used to model many other environmental parameters that must be positive. There are a variety of distribution forms that can be developed using different values of the distribution parameters.

The Weibull distribution is typically specified by a minimum (location) value (ε), a scale (or characteristic life) parameter (β), and a shape(or slope) parameter (α). The random variable must be greater than 0 and also greater than the minimum value, ε.

Within GoldSim, the Weibull is defined by ε, α, and the mean - ε. As shown below, the mean can be readily computed as a function of ε, α, and β.

PDF

$$f(x) = \frac{\alpha}{\beta}\left(\frac{x-\varepsilon}{\beta}\right)^{\alpha-1} e^{-\left(\frac{x-\varepsilon}{\beta}\right)^{\alpha}}$$

CDF

$$F(x) = 1 - e^{-\left(\frac{x-\varepsilon}{\beta}\right)^{\alpha}}$$

Mean

$$\mu = \varepsilon + \beta\Gamma\left(1+\frac{1}{\alpha}\right)$$

Variance

$$\sigma^2 = \beta^2\left\{\Gamma\left(1+\frac{2}{\alpha}\right) - \left[\Gamma\left(1+\frac{1}{\alpha}\right)\right]^2\right\}$$

Representing Truncated Distributions

Several distributions in GoldSim can be truncated at the ends (normal, log-normal, Gamma, and Weibull). That is, by specifying a lower bound and/or an upper bound, you can restrict the sampled values to lie within a portion of the full distribution's range.

The manner in which truncated distributions are sampled is straightforward. Because each point in a full distribution corresponds to a specific cumulative probability level between 0 and 1, it is possible to identify the cumulative probability levels of the truncation points. These then define a scaling function which allows sampled values to be mapped into the truncated range.

In particular, suppose the cumulative probability levels for the lower bound and upper bound were L and U, respectively. Any sampled random number R (representing a cumulative probability level between 0 and 1) would then be scaled as follows:

L + R(U-L)

This resulting "scaled" cumulative probability level would then be used to compute the sampled value for the distribution. The scaling operation ensures that it falls within the truncated range.

Correlation Algorithms

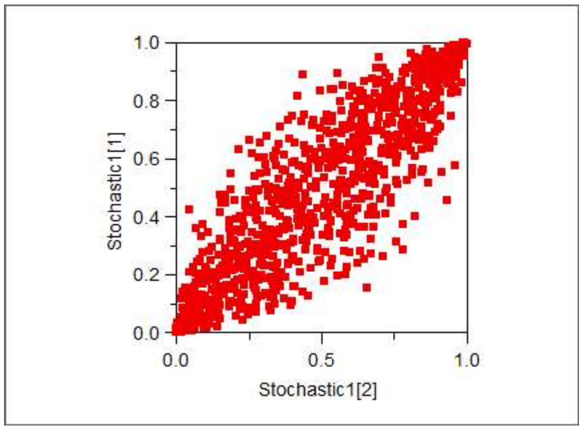
Several GoldSim elements that are used to represent uncertainty or stochastic behavior in models permit the user to define correlations between elements or amongst the items of a vector-type element.

To generate sampled values that reflect the specified correlations GoldSim uses copulas and the Iman and Conover methodology.

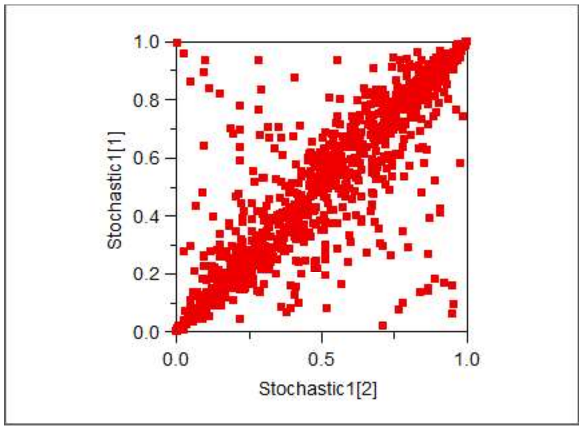
A copula is a function that joins two or more univariate distributions to form a multivariate distribution. As such, it provides a method for specifiying the correlation between two variables. Copulas are described in detail by Embrechts, Lindskog and McNeil (2001).

GoldSim uses three different methods to generate correlated values: the Gaussian copula, the t-distribution copula and the Iman and Conover method. When a Stochastic element is correlated to itself, or to another Stochastic element, GoldSim uses the Gaussian copula to generate the correlated value. A vector-type Stochastic or History Generator can use the Gaussian copula, the t-distribution copula, or the Iman and Conover method to generate correlated values.

The Gaussian copula produces values where the correlation between variables is stronger towards the middle of the distributions than it is at the tails. The plot below shows the values for two variables (uniform distributions between 0 and 1) generated using the Gaussian copula with a correlation coefficient of 0.9:



A t-distribution copula produces a correlation that is stronger at the tails than in the middle. The plot below shows the values for the two variables generated using the t-distribution copula for the same variables with a correlation coefficient of 0.9 and the **Degrees of Freedom** setting in the copula of 1):



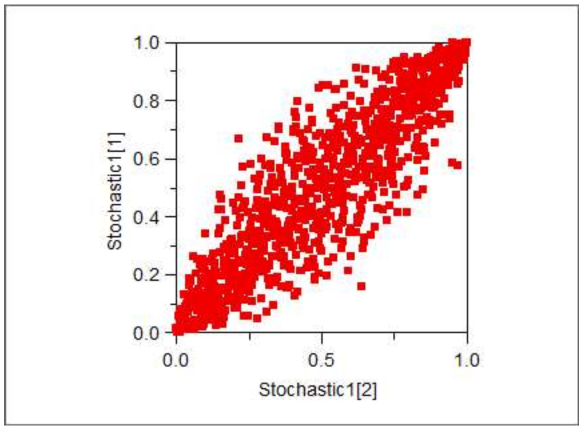
The t-distribution's form is often what is observed in the real world: correlations at the extremes (e.g. representing a rare, but significant event such as a war) tend to be higher than correlations in the middle (representing the higher variability in more common occurrences).

The **Degrees of Freedom** setting controls the tail dependency in the copula. A low value produces stronger dependence in the tails, while higher values produce stronger correlations in the middle of the distributions. This means that a t-distribution copula with a high number of Degrees of Freedom will begin to behave like a Gaussian copula.

One of the weaknesses of the copula approach to generating correlated samples is that it does not respect Latin Hypercube Sampling (with the exception of the first item in a vector-type stochastic where the Gaussian copula is used to generate sampled values).

The Iman and Conover approach is designed to produce a set of correlated items that each respect Latin Hypercube sampling. Complete details on the algorithm's methodology can be found in Iman and Conover (1982).

Its behavior is similar, but not identical, to a Gaussian for the first sample:



However, if the element is resampled during a realization, elements that use the Iman and Conover approach will use the Gaussian copula to generate the second and subsequent sets of sampled data.

Sampling Techniques

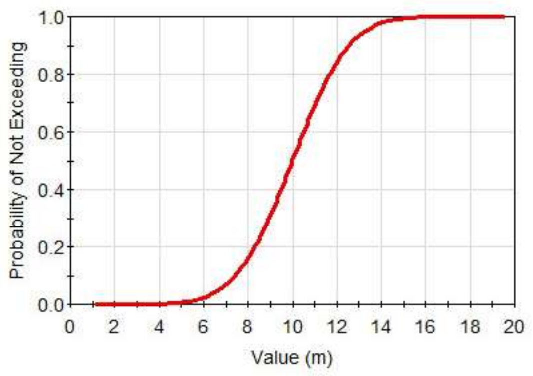
This section discusses the techniques used by GoldSim to sample elements with random behavior. These include the following GoldSim elements:

- Stochastic
- Random Choice
- Timed Event Generator
- Event Delay
- Discrete Change Delay
- History Generator
- Source (in the Radionuclide Transport Module)
- Action element (in the Reliability module)
- Function element (in the Reliability module)

After first discussing how GoldSim generates random numbers in order to sample these elements, two enhanced sampling techniques provided by GoldSim (Latin Hypercube sampling and importance sampling) are discussed.

Generating Random Numbers to Sample Elements

In order to sample an element (we will simplify the discussion here by using a Stochastic element as an example rather than one of the other types of elements), GoldSim starts with the CDF of the distribution that we want to sample. Below is a CDF for a Normal distribution with a mean of 10m and a standard deviation of 2m:



To randomly sample this distribution, we simply need to do the following:

1. Obtain a random number (i.e., a number between 0 and 1).
2. Use the CDF to map that random number to the corresponding sampled value.

So, for example, in the CDF above, a random number of about 0.2 would correspond to a sampled value of about 8.3m.

As can be seen, this sampling process itself is conceptually very simple. The more complicated part involves obtaining the random number needed to sample the element. That is, in order to carry out Monte Carlo simulation, GoldSim (and any Monte Carlo simulator) needs to consistently generate a series of random numbers.

In GoldSim, the process consists of the following components:

- Several different types of ***random number seeds***. You can simply think of a random number seed as an integer number. It actually consists of a pair of long (32-bit) integers, but that is not important to the discussion that follows.
- ***Random numbers***. A random number, as used here, has a specific definition: it is a real number between 0 and 1.
- A ***seed generator***. This is an algorithm that takes as input one random number seed and randomly generates a new random number seed. A particular value for the input seed always generates the same output seed, but different input seeds generate different output seeds.
- A ***random number generator***. This is another algorithm. It takes as input one random number seed and generates a random number. A particular value for the random number seed always generates the same random number, but different random number seeds generate different random numbers.

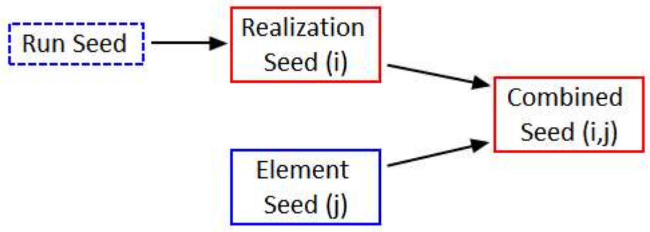
Within GoldSim, there are several types of random number seeds:

- The model itself (as well as any SubModel) has a ***run seed***. If you choose to **Repeat Sampling Sequences** (an option on the **Monte Carlo** tab of the Simulation Settings dialog), this seed is (reproducibly) created based on an integer number that can be edited by the user. If you do not **Repeat Sampling Sequences**, it is randomly created based on the computer's system clock.
- The ***realization seed*** is a mutable seed that is initialized with the run seed and then updated for each realization.
- Each Stochastic element (as well as other elements that behave probabilistically) has its own random number seed. This is referred to as the ***element seed***. This seed is created (in a random manner, based on the system clock) when the element is first created.

Note: No two elements in a GoldSim model can have the same ***element seed***. This means that if an element is copied and pasted into a model where no elements have the same seed value, its seed will be unchanged. However, if it is pasted into the same model, or into a model where another element already has that seed value, one of the elements with the same seed value will be given a new unique seed. (Element seeds can be displayed by selecting the element in the graphics pane and pressing **Ctrl-Alt-Shift-F12**).

- For every element that behaves probabilistically), the realization seed and the element seed are combined together to create a ***combined seed*** for that element. It is this combined seed that is used to generate a random number using a random number generator.

This random number seed structure is illustrated below:



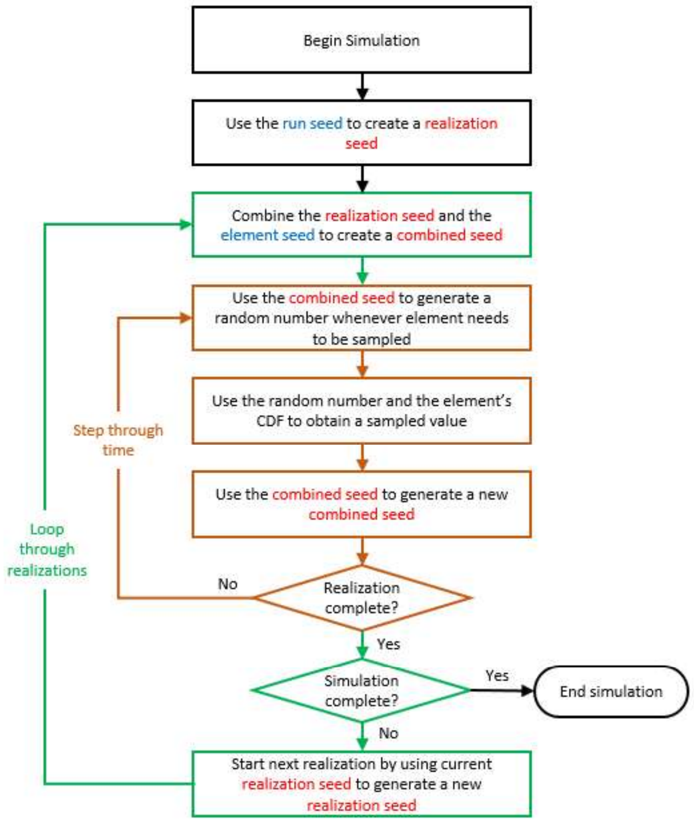
The ***run seed*** and ***element seeds*** are constant during a simulation (they never change). The run seed is marked using a dashed line to indicate that although it is constant during a simulation, it can be changed by the user. The element seed (which is different for each element j) is created when the element is created and cannot be changed. As we shall see below, however, the ***realization seeds*** and ***combined seeds*** are not constant during a simulation but change as the simulation proceeds.

So given all of this information, let's describe how GoldSim carries out a Monte Carlo simulation by considering a very simple model consisting of a single Stochastic element that is resampled every day, with the model being run for multiple realizations:

1. At the beginning of the simulation, GoldSim has a value for the ***run seed***, and a single ***element seed***.
2. At the beginning of the simulation (assuming we are repeating sampling sequences), the ***run seed*** is used to initialize a ***realization seed***.

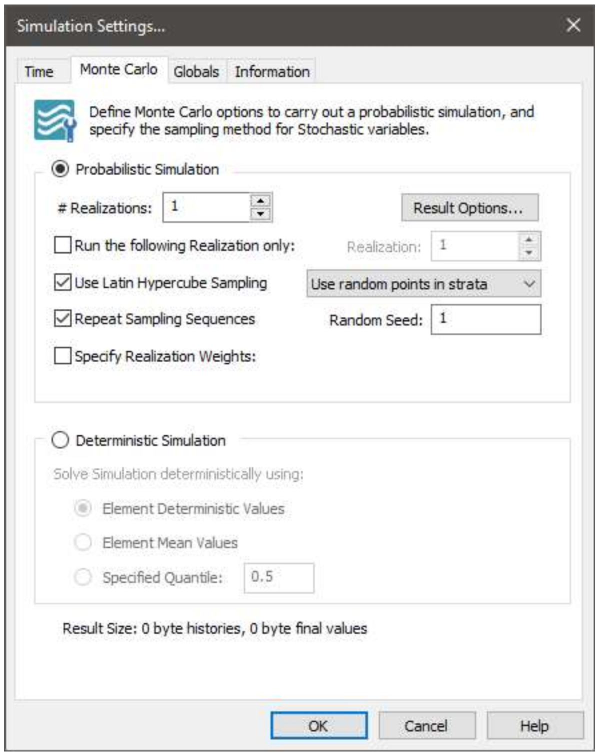
3. At the beginning of each realization, we do the following:
    a. The current ***realization seed*** is input int the seed generator to generate a new ***realization seed*** for this realization.
    b. The ***realization seed*** is combined with the ***element seed*** to create a ***combined seed*** for the element.
    c. Now that we have the ***combined seed***, two things happen:
        i. The ***combined seed*** is input into the random number generator to generate a ***random number*** for the element. This random number is then mapped to the CDF to obtain a ***sampled value*** for the element.
        ii. The ***combined seed*** is input into the seed generator to generate a new ***combined seed*** that we will use the next time we need a random number.
    d. Whenever we need to resample the element during the realization (in this case every day), we need a new random number. To do so, every day we repeat steps i and ii above.

This same logic is shown schematically below:


Begin Simulation
Use the run seed to create a realization seed
Combine the realization seed and the element seed to create a combined seed
Use the combined seed to generate a random number whenever element needs to be sampled
Use the random number and the element's CDF to obtain a sampled value
Use the combined seed to generate a new combined seed
Step through time
Loop through realizations
No
Realization complete?
Yes
Simulation complete?
Yes
End simulation
No
Start next realization by using current realization seed to generate a new realization seed

Note: GoldSim's random number generation process is based on a method presented by L'Ecuyer (1988). As pointed out above, each seed actually consists of two long (32-bit) integers. When random numbers are generated, each of the integers cycles through a standard linear congruential sequence, with different constants used for the two sequences (so that their repeat-cycles are different). The random numbers that are generated are a function of the combination of the two seeds, so that the length of their repeat period is extremely long.

Now let's consider the various options on the **Monte Carlo** tab of the Simulation Settings dialog:


Simulation Settings...

Time | Monte Carlo | Globals | Information

Define Monte Carlo options to carry out a probabilistic simulation, and specify the sampling method for Stochastic variables.

● Probabilistic Simulation

Realizations: 1

Result Options...

☐ Run the following Realization only: Realization: 1

☑ Use Latin Hypercube Sampling — Use random points in strata

☑ Repeat Sampling Sequences — Random Seed: 1

☐ Specify Realization Weights:

○ Deterministic Simulation

Solve Simulation deterministically using:

● Element Deterministic Values

○ Element Mean Values

○ Specified Quantile: 0.5

Result Size: 0 byte histories, 0 byte final values

OK | Cancel | Help

In particular, we will focus on just two fields: **Repeat Sampling Sequences** and **Random Seed**. These two fields impact the ***run seed*** in the following ways:

- If **Repeat Sampling Sequences** in checked, you can specify a **Random Seed**. The **Random Seed** is used to create the ***run seed***.
- If **Repeat Sampling Sequences** is cleared, the ***run seed*** is created "on the fly" using the system clock. As a result, it is different every time the model is run.

These facts, combined with the logic outlined above, can be used to describe exactly how elements will be sampled in various models under any set of circumstances. In particular:

- If **Repeat Sampling Sequences** is checked (the default), as long as you do not modify the model, you will get the same results (i.e., the same random numbers will be used) if you run the model today, and then run it again tomorrow. This is because the ***run seed*** is unchanged.
- Similarly, if **Repeat Sampling Sequences** is checked and you copy a model to someone else (and they do not make any changes), they will get the same results as you.
- If **Repeat Sampling Sequences** is checked, but the **Random Seed** is changed (e.g., from 1 to 2), you will get different results (i.e., different random numbers will be used). This is because the ***run seed*** is different. If you then change the **Random Seed** back to the original value, you will reproduce the original results.
- If **Repeat Sampling Sequences** is cleared, every time you run the model you will get different results (i.e., different random numbers will be used). This is because the ***run seed*** is different
- If two people simultaneously build the same simple model with the exact same inputs (including the same **Random Seed**), the results will still be different (i.e., different random numbers will be used). This is because the ***element seeds*** will be different.
- If the user selects the option to **Run the following Realization only**, such as realization 16, GoldSim simply iterates through the process the necessary number of times prior to starting the specified realization.

Latin Hypercube Sampling

GoldSim provides an option to implement a ***Latin Hypercube sampling*** (LHS) scheme (in fact, it is the default when a new GoldSim file is created). The LHS option results in forced sampling from each "stratum" of each parameter.

The following elements use LHS sampling:

- Stochastic
- Random Choice
- Timed Event Generator
- Time Series (when time shifting using a random starting point)
- Action (in the Reliability Module)
- Function (in the Reliability Module)

Each element's probability distribution (0 to 1) is divided into up to 10000 equally likely strata or slices (actually, the lesser of the number of realizations and 10000). The strata are then "shuffled" into a random sequence, and a random value is then picked from each stratum in turn. This approach ensures that a uniform spanning sampling is achieved.

Note: If possible, GoldSim will attempt to create LHS sequences where subsets are also complete LHS sequences. This means that if the total number of realizations is an even number, the first half and second half of the realizations are complete LHS sequences. If the total number of realizations is divisible by 4 or 8, that fraction of the total number of realizations, run in sequence, will be complete LHS sequences. This property of GoldSim's LHS sequences is sometimes useful for statistical purposes and also permits a user to extract a valid LHS sequence from a partially completed simulation by screening realizations. The details of this approach are discussed below ("Latin Hypercube Subsets").

Note that each element has an independent sequence of shuffled strata that are a function of the element's internal random number seed and the number of realizations in the simulation. If the number of realizations exceeds 10,000, then at the 10,001st realization each element makes a random jump to a new position in its strata sequence. A random jump is repeated every 10,000 realizations.

If you select "Use mid-points of strata" in the Simulation Settings dialog in models with less than 10,000 realizations, GoldSim will use the expected value of the strata selected for that realization. (Even if this option is selected, in simulations with greater than 10,000 realizations, the 10,001 and subsequent realizations will use random values from within the strata selected for that realization.) Using mid-points provides a slightly better representation of the distribution, but without the full randomness of the original definition of Latin Hypercube sampling (as described by McKay, Conover and Beckman, 1979).

If you select "Use random points in strata" in the Simulation Settings dialog, GoldSim also picks a random value from each stratum.

Note: LHS is only applied to the first sampled random value in each realization. Subsequent samples will not use LHS.

LHS appears to have a significant benefit only for problems involving a few independent stochastic parameters, and with moderate numbers of realizations. In no case does it perform worse than true random sampling, and accordingly LHS sampling is the default for GoldSim.

Note that the binary subdivision approach (described in more detail below) and the use of mid-stratum values are GoldSim-specific modifications to the original description of Latin Hypercube Sampling, as described in McKay, Conover and Beckman (1979).

Latin Hypercube Subsets

In order to allow users to do convergence tests, GoldSim's LHS sampling automatically organizes the LHS strata for each random variable so that binary subsets of the overall number of realizations each represent an independent LHS sample over the full range of probabilities.

For example, if the user does 1,000 realizations, GoldSim will generate strata such that:

- Realizations 1-125 represent a full LHS sample with 125 strata. Realizations 126-250, 251-375, etc. through 876-1000 also represent full LHS samples with 125 strata each.
- Also, realizations 1-250, 251-500, 501-750, and 751-1000 represent full LHS samples with 250 strata each.
- And, realizations 1-500 and 501-1000 represent full LHS samples with 500 strata each.

The generation of binary subsets is automatic, and is carried out whenever the total number of realizations is an even number. Up to 16 binary subsets will be generated, if the number of realizations can be subdivided evenly four times. For example, if the total number of realizations was 100 then GoldSim would generate 2 subsets of 50 strata each and 4 subsets of 25 strata. If the total number of realizations was 400 then GoldSim would generate 2 subsets of 200 strata, 4 subsets of 100 strata, 8 subsets of 50 strata, and 16 subsets of 25 strata.

The primary purpose of this sampling approach is to use the subsets to carry out statistical tests of convergence. For example, the mean of each of the subsets of results could be evaluated and a t-test used to estimate statistics of the population mean, as described in Iman (1982). Rather than carrying out a set of independent LHS simulations using different random seeds, this approach allows the user to run a single larger simulation, with the benefits of a better overall representation of the system's result distribution, while still being able to test for convergence and to generate confidence bounds on the results. (A secondary benefit of the binary sampling approach is that if a simulation is terminated partway through it should have a slightly greater likelihood of having uniform sampling over the completed realizations than normal Latin Hypercube sampling would.)

The algorithm for assigning strata to the binary subsets is quite simple. For each pair of strata (e.g., 1st and 2nd; 3rd and 4th), the first member of the pair is randomly assigned to one of two "piles" ("left" or "right"), and the second member is assigned to the opposite pile. That is, conceptually, it can be imagined that the full set of strata is sent one at a time to a 'flipper' that randomly chooses 'left' or 'right' on its first, third, fifth etc. activations, and on its second, fourth, sixth etc. activations chooses the opposite of the previous value.

For the case of one binary subdivision of a total of Ns strata, the algorithm goes through the strata sequentially from lowest to highest, and passes them to a flipper that generates two 'piles' of strata. Each pile will therefore randomly contain one of the first two strata, one of the second two, and so on. Thus, each pile will contain one sample from each of the strata that would have been generated if only Ns /2 total samples were to be taken. The full sampling sequence is generated by randomly shuffling each pile and then concatenating the two sequences.

For four binary subdivisions the same approach is extended, with the first flipper passing its 'left' and 'right' outputs to two lower-level flippers. This results in four 'piles' of strata, which again are just randomly shuffled and then concatenated. The same approach is simply extended to generate eight or sixteen strata where possible.

Importance Sampling

For risk analyses, it is frequently necessary to evaluate the low-probability, high-consequence end of the distribution of the performance of the system. Because the models for such systems are often complex (and hence need significant computer time to simulate), and it can be difficult to use the conventional Monte Carlo approach to evaluate these low-probability, high-consequence outcomes, as this may require excessive numbers of realizations.

To facilitate these type of analyses, GoldSim allows you to utilize an ***importance sampling*** algorithm to modify the conventional Monte Carlo approach so that the high-consequence, low-probability outcomes are sampled with an enhanced frequency. During the analysis of the results which are generated, the biasing effects of the importance sampling are reversed. The result is high-resolution development of the high-consequence, low-probability "tails" of the consequences, without paying a high computational price.

The following elements permit importance sampling:

- Stochastic
- Random Choice
- Timed Event Generator
- Action (in the Reliability Module)
- Function (in the Reliability Module)

Note: Importance sampling is only applied to the first sampled random value in each realization for elements with importance sampling enabled. Subsequent samples will use random sampling.

Note: In addition to the importance sampling method described here (in which you can choose to force importance sampling on either the low end or high end of a Stochastic element's range), GoldSim also provides an advanced feature that supports custom importance sampling that can be applied over user-defined regions of the Stochastic element's range.

Read More: *Customized Importance Sampling Using User-Defined Realization Weights* on page 1075

Warning: Importance sampling affects the basic Monte Carlo mechanism, and it should be used with great care and only by expert users. In general, it is recommended that only one or at most a very few parameters should use importance sampling, and these should be selected based on sensitivity analyses using normal Monte Carlo sampling. Importance sampling should only be used for elements whose distribution tails will not be adequately sampled by the selected number of realizations.

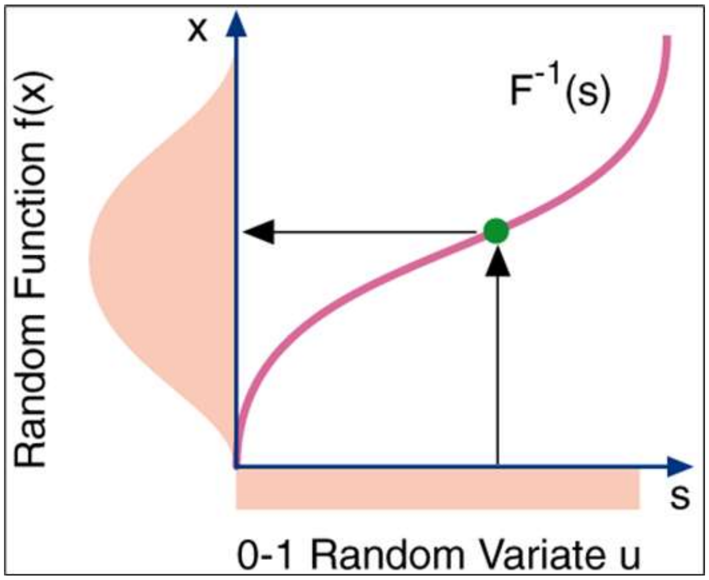
How Importance Sampling Works

Importance sampling is a general approach to selectively enhance sampling of important outcomes for a model. In principle, the approach is simple:

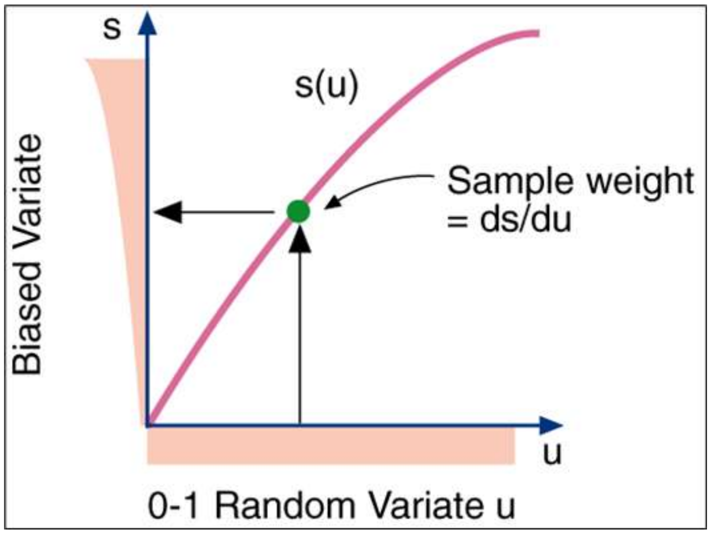
1. Identify an important subset of the sampling space;
2. Sample that subset at an enhanced rate; and
3. When analyzing results, assign each sample a weight inversely proportional to its enhancement-factor.

realization is assumed equally probable. It is straightforward, however, to incorporate a weight associated with each sample in order to represent the relative probability of the sample compared to the others.

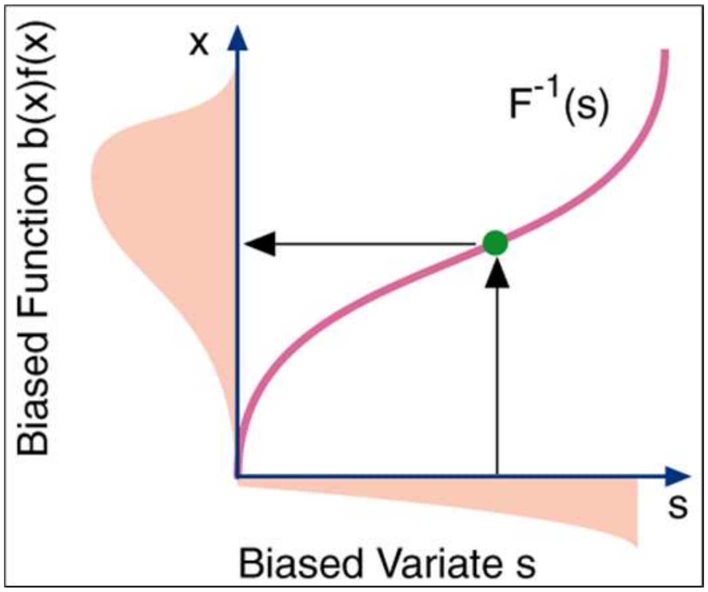
The conventional Monte Carlo approach is as shown below. A uniform 0 – 1 random variable **u** is sampled, and its value is then used as input to the inverse cumulative distribution function of the random variable:



In order to do importance sampling, the original uniformly-distributed random numbers are first mapped onto a non-uniform 'biased' sampling function **s**:



The biased variate **s** is then used to generate the random function. Since the input random numbers are no longer uniformly distributed, the resulting sample set is selectively enriched in high-consequence results:



In general, any continuous monotonic biasing function **s** which spans the range 0-1, and has **s(0)** = 0 and **s(1)** = 1 can be used to generate the set of input random numbers. The weight associated with each sampled realization is **ds/du**, the slope of the biasing function **s** at the sampled point.

Biasing (Enhancement) Functions

GoldSim uses simple functions to selectively enhance either the upper or the lower end of an element's probability distribution.

$$s_{lower}(u) = u - \frac{u}{1+au} + \frac{u^2}{1+a}$$

where a is a function of the number of elements that are using importance sampling. This is equal to zero if only one element uses importance sampling, and is equal to ten times the number of importance sampled elements in all other cases. The effect of increasing a is to restrict the importance sampling to a smaller subset of the full range of the random value, which reduces the negative impacts of importance sampling numerous variables in the same model.

The sample weight is given by:

$$w_{lower} = \frac{ds}{du} = 1 - \frac{1}{(1+au)^2} + \frac{2u}{1+a}$$

Note: This is the *relative* weight (not the actual weight). In a Monte Carlo simulation without biasing, the relative weight of each sample is 1. The actual weight is the relative weight/the number of realizations.

Note: For the first 10,000 realizations where GoldSim uses the expected values of the LHS strata the weight given to each sample is equal to the integral of s over the stratum divided by the corresponding integral of u over the strata. For the 10,001st and subsequent realizations GoldSim will calculate s and w for the sampled point.

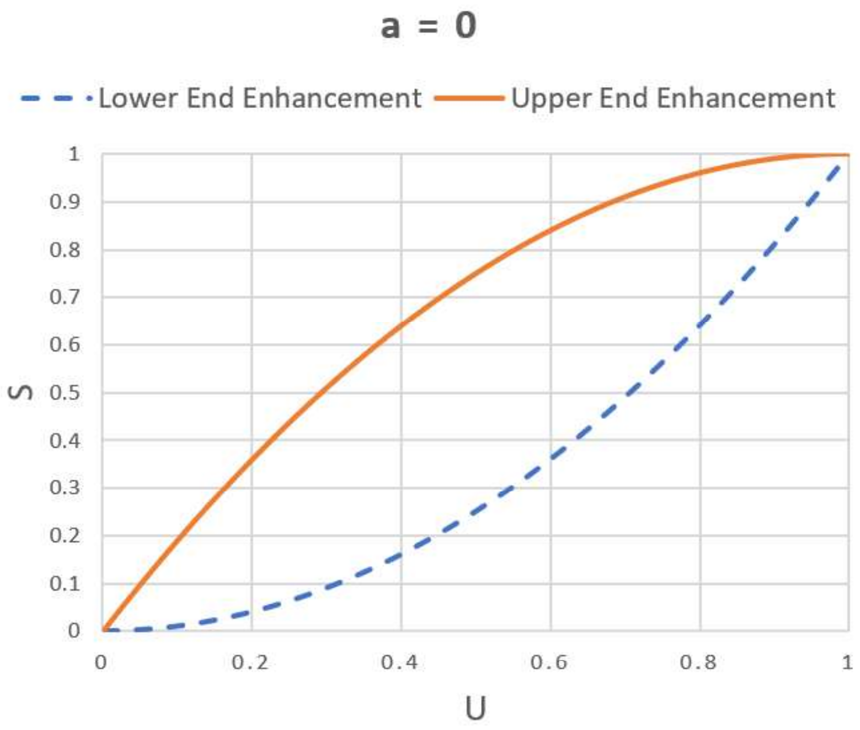
The biasing function for enhancing the upper end is:

$$s_{upper} = 1 - s_{lower}(1-u) = 1 - \left[(1-u) - \frac{(1-u)}{1+a(1-u)} + \frac{(1-u)^2}{1+a}\right]$$

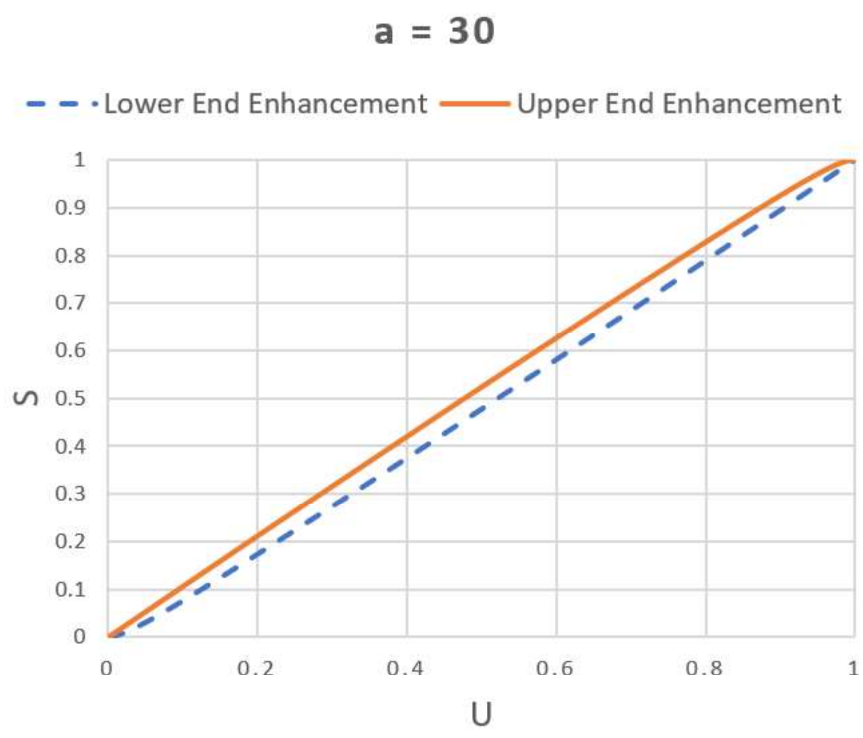
and the corresponding sample weight is given by:

$$w_{upper} = w_{lower}(1-u) = 1 - \frac{1}{[1+a(1-u)]^2} + \frac{2(1-u)}{1+a}$$

The following plot shows the upper and lower biasing function when a single element utilizes importance sampling (a = 0):



The following figure shows the bias function when three elements are importance sampled (a = 30):



Note the less prominent bias as the number of importance sampled elements increases.

Behavior of Elements with Importance Sampling Enabled

The Stochastic element provides the option to choose between upper and lower end enhancement when importance sampling is enabled. However, the other elements that utilize importance sampling (the Timed Event element, the Reliability elements and the Random Choice element) importance sample only one end of the distribution.

Timed Events will use lower end importance sampling on the time to event distribution specified by the user. The Random Choice element has a slightly different behavior. When importance sampling is enabed, the Random Choice element sorts the probability of each outcome from lowest probability to highest probability and assigns them to sections of a uniform distribution. This uniform distribution is then importance sampled at the lower end, so that the least probable outcomes are enhanced.

The Reliability elements will use a combination of these approaches. Time based failure modes behave in a similar manner to the timed event elements. Modes with a “probability of failure” perform importance sampling to enhance the number of failure outcomes.

It is important to note that only the first sampled value utilizes importance sampling. This means that only the first event from a Timed Event, the first Random Choice and the first occurrence of each Failure Mode will use importance sampling.

Representing Random (Poisson) Events

Timed Event elements can be specified to produce discrete event signals regularly or randomly.

Read More: *Timed Event Elements* on page 392

Random events are simulated as occurring according to a Poisson process with a specified ***rate of occurrence***. If an event occurs according to a Poisson process, the probability that N events will occur during a time interval T is described by the following expression (Cox and Miller, 1965):

$$P(N) = \frac{e^{-\lambda T}(\lambda T)^N}{N!}$$

where:

P(N) is the probability of N occurrences of the event within the time interval T;
T is the time interval of interest;
λ is the annual rate of occurrence; and
N is the number of occurrences of the event within the time interval T.

The expected (mean) number of occurrences during the time interval is equal to λT.

The Poisson distribution also has the property that the intervals between events are exponentially distributed (Benjamin and Cornell, 1970):

$$F(t) = 1 - e^{-\lambda t}$$

where F(t) is the probability that the time to the next event will be less than or equal to t.

If you indicate that the event can only occur once, GoldSim simulates the first occurrence according to the above equations, but does not allow the event to occur again.

Note that the rate of occurrence can be specified to be a function of time (i.e., the event process can be non-stationary).

Computing and Displaying Result Distributions

Displaying a CDF

Probabilistic results may be viewed in the form of a CDF (or a CCDF). This section describes how the values realized during the simulations are used to generate the CDF (or CCDF) seen by the user.

Creating the Results

Within GoldSim Monte Carlo results are stored in a particular data structure, referred to here as the results array. As the simulation progresses, each specific Monte Carlo realization result is added to the results array as a pair of values; the value realized and the weight given by GoldSim to the value. The array is filled "on the fly", as each new realization is generated. Theoretically, each separate realization would represent a separate entry in the results array (consisting of a value and a weight). If unbiased sampling were carried out each separate entry would have equal weight.

As implemented in GoldSim, however, the number of data pairs in the results array may be less than the number of realizations: There are two reasons why this may be the case:

- If multiple results have identical values, there is no need to have identical data pairs in the results array: the weight associated with the particular value is simply adjusted (e.g., if the value occurred twice, its weight would be doubled).

- For computational reasons, the results array has a maximum number of unique results which it can store. The maximum number for post-processing GoldSim simulation results is 25,000. If the number of realizations exceeds these limits, results are "merged" in a self-consistent manner. The process of merging results when the number of realizations exceeds 25,000 is discussed below.

To merge a new result with the existing results (in cases where the number of realizations exceeds one of the maxima specified above), GoldSim carries out the following operations:

- GoldSim finds the surrounding pair of existing results, and selects one of them to merge with. GoldSim selects this result based on the ratio of the distance to the result to the weight of the result (i.e., the program preferentially merges with closer, lower weight results).
- After selecting the result to merge with, GoldSim replaces its value with the weighted average of its existing value and the new value; it then replaces its weight with the sum of the existing and new weights.

There is one important exception to the merging algorithm discussed above: If the new result will be an extremum (i.e., a highest or a lowest), GoldSim replaces the existing extremum with the new one, and then merges the existing result instead. This means that GoldSim never merges data with an extremum.

Plotting a CDF

Plotting the CDF from the results array is straightforward. The basic algorithm assumes that the probability distribution between each adjacent pair of result values is uniform, with a total probability equal to half the sum of the weights of the values. One implication of this assumption is that for a continuous distribution the probability of being less than the smallest value is simply equal to half the weight of the lowest value and the probability of being greater than the highest value is half the weight of the highest value.

For example, if we have ten equally weighted results in a continuous distribution, there is a uniform probability, equal to 0.1, of being between any two values. The probability of being below the lowest value or above the highest value would be 0.05. GoldSim extrapolates the value at a probability level of 0 using the slope between the first two observations. Similarly the slope between the last two observations is used to estimate the value at a probability level of 1.

Note: Extrapolation is not carried out for Milestone result distributions, Reliability element failure and repair time distributions, and for Sampled Stochastic distributions if the option to extrapolate is not checked.

In certain circumstances there are several minor variations to the basic algorithm discussed above:

- If a large number of results are identical, GoldSim assumes the entire distribution is discrete (rather than continuous), and lumps the probabilities at the actual values sampled. In particular, if more than 50% of the realization results were identical to an existing result (and there are less than 1000 results), GoldSim presumes the distribution is discrete and plots it accordingly. The user can observe this by sampling from a binomial distribution.
- GoldSim uses a heuristic algorithm to decide if each specific result represents a discrete value: if any result is repeated, GoldSim treats the result as a discrete value and does not ‘smear’ it. For example, suppose the result is 0.0 30% of the time, and normal (mean=10, s.d.=2) the rest of the time. The first result value would be 0.0, with a weight of about 0.3. The second value would be close to 8, with a weight of 1/# realizations. We would not want to smear half of the 0 result over the range from 0 to 8!

When the user selects the confidence bounds options (discussed below), a different algorithm is used to display and plot CDF values. In particular, the displayed value is simply the calculated median (50th percentile) in the probability distribution for the “true” value of the desired quantile.

Displaying a PDF

Displaying PDFs is much more difficult than CDFs, as unless a large number of realizations are run, PDFs tend to be “noisy” even if the corresponding CDF appears smooth (small changes in the slope of the CDF are typically not noticeable to the eye, but these can translate into very noticeable “spikes” in the PDF).

To display a PDF, GoldSim creates a series of equally-spaced bins, with a constant density value within each bin. The density within each bin is the average slope of the CDF over the bin. The number of bins automatically increases with the number of realizations. In particular, the number of bins is equal to the square root of the number of results being considered. The maximum number of bins used is 1000 (and the minimum is 1).

Computing and Displaying Confidence Bounds on the Mean

GoldSim is able to perform a statistical analysis of Monte Carlo results to produce confidence bounds on the mean of a probabilistic result. These bounds reflect uncertainty in the probability distribution due to the finite number of Monte Carlo realizations - as the number of realizations is increased, the limits become narrower. The confidence bounds on the mean are displayed in the Statistics section of the Distribution Summary dialog when viewing Distribution Results if the **Confidence Bounds** checkbox is checked.

Read More: *Viewing a Distribution Summary* on page 670

This approach to compute the confidence bounds uses the t distribution, which is strictly valid only if the underlying distribution is normal. The 5% and 95% confidence bounds on the population mean are calculated as defined below:

$$\text{5\% Bound} = \overline{X} - t_{0.05}\frac{s_x}{\sqrt{n}}$$

$$\text{95\% Bound} = \overline{X} + t_{0.05}\frac{s_x}{\sqrt{n}}$$

where:

$\overline{X}$ is the sample mean;
$t_{0.05}$ is the 5% value of the t distribution for n-1 degrees of freedom;
s_x is the sample standard deviation; and
n is the number of samples (realizations).

As the number of realizations, n, becomes large, the Central Limit theorem becomes effective, the t distribution approaches the normal distribution, and the assumption of normality is no longer required. This may generally be assumed to occur for n in the order of 30 to 100 realizations even for results of highly-skewed distributions.

Note: These confidence bound calculations are strictly only applicable for purely random sampling. When using Latin Hypercube sampling, these bounds overestimate the uncertainty in the mean.

Computing and Displaying Confidence Bounds on CDFs and CCDFs

GoldSim is able to perform a statistical analysis of Monte Carlo results to produce confidence bounds on the resulting probability distribution curves. These bounds reflect uncertainty in the probability distribution due to the finite number of Monte Carlo realizations - as the number of realizations is increased, the limits become narrower. The confidence bounds are displayed when viewing Distribution Results if the **Show Confidence Bounds** button in the Distribution display window is pressed.

The confidence bounds appear as different-colored curves on the probability plots produced by the GoldSim user interface. The bounds represent 5% and 95% confidence limits on the distribution value at each probability level. Confidence bounds cannot be displayed for PDFs.

The theory used to produce the confidence bounds has several limitations:

- The bounds on distributions can only be calculated for cases where all realizations have equal weights. If importance sampling is used for any parameter is used, GoldSim will not display confidence bounds.
- The confidence bounds cannot be calculated for values less than the smallest result, or greater than the largest result. As a result of this, the confidence-bound curves do not generally reach all of the way to the tails of the result plots.

In cases with relatively few stochastic parameters, Latin Hypercube sampling can increase the accuracy of the probability distributions. The confidence bounds are not able to reflect this improvement, and as a result will be conservatively wide in such cases.

Theory: Bounds on Cumulative Probability

Suppose we have calculated and sorted in ascending order n random results r_i from a distribution. What can we say about the q^{th} quantile x_q (e.g., q=0.9) of the underlying distribution?

Each random result had a probability of q that its value would be less than or equal to the actual q^{th} quantile x_q. The total number of results less than x_q was therefore random and binomially distributed, with the likelihood of exactly i results <= x_q being:

$$P(i) = P(r_i \le x_q \le r_{i+1}) = \binom{n}{i} q^i (1-q)^{n-i} = \frac{n!}{i!(n-i)!} q^i (1-q)^{n-i}$$

Note that there may be a finite probability that the value of x_q is less than the first or greater than the largest result: for example, if 100 realizations r_i were sampled, there would be a probability of 0.366 that the 0.99 quantile exceeded the largest result. The 100 realization probability distribution for $x_{0.99}$ is as follows:

Between Results	Probability	Cumulative Probability
<94	0.0000	0.0000
94 and 95	0.0005	0.0005
95 and 96	0.003	0.0035
96 and 97	0.015	0.0185
97 and 98	0.061	0.0795
98 and 99	0.1849	0.2644
99 and 100	0.370	0.6344
> 100	0.366	1

GoldSim assumes that the probability defined by the equation presented above is uniformly distributed over the range from r_i to r_{i+1}, and interpolates into the Monte Carlo results-list to find the 5% and 95% cumulative probability levels for x_q.

Using the above probability distribution, it is also possible to calculate the expected value of x_q. This approach appears (by experimentation) to provide a slightly more reliable estimate of x_q than the conventional Monte Carlo approach of directly interpolating into the results-list. The expected value of x_q is calculated by summing the product of the probability of x_q lying between each pair of results and the average of the corresponding pair of result-values:

$$\overline{x_q} = \sum_{i=1}^{n} P(i) \frac{(r_{i+1} + r_i)}{2}$$

When using this equation to estimate a very high or low quantile, a problem arises when the probability level, q, is near to 0 or 1, as there can be a significant probability that x_q lies outside the range of results. In the table presented above, for example, there is a 0.366 chance of $q_{0.99}$ exceeding the largest result. In such cases, an estimate of the expected value of x_q can be found by extrapolating from the probabilities within the range of the results. Obviously, however there are limits to extrapolation and without knowledge of the actual distributional form no extrapolation would produce a reliable estimate of (say) the 0.9999 quantile if only 100 realizations had been performed.

In evaluating the binomial distribution for large values of n, large numbers can be generated which can cause numerical difficulties. To avoid these difficulties, when the number of realizations (n) is greater than 100, GoldSim uses either the normal or Poisson approximations to the binomial distribution. The Poisson approximation is used when i or (n-i) is less than 20 and the normal distribution is used otherwise. These approximations are described in any introductory statistics text.

Computing the Conditional Tail Expectation

The Conditional Tail Expectation (CTE) is the expected value of the result given that it lies above a specified value or quantile. The CTE is displayed in the 'Calculator' portions of the Stochastic and Result Distribution elements, and is an optional output type for a Monte Carlo SubModel element.

Read More: *Specifying the Distribution for a Stochastic Element* on page 142; *Viewing a Distribution Summary* on page 670; *Creating the Output Interface to a SubModel* on page 1047

For a tail starting at value k, the CTE is defined as:

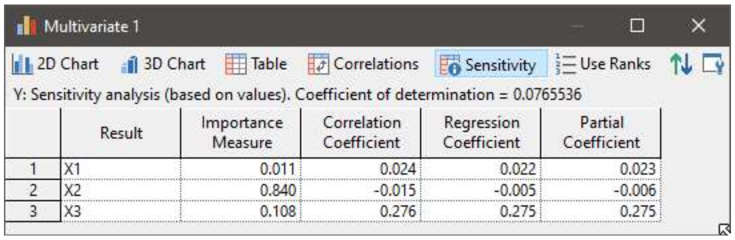
$$CTE_k = \frac{1}{1 - F(k)} \int_k^{\infty} x\, f(x) dx$$

where:

FK) is the cumulative probability of value k; and
f(x) is the probability density at value x.

Computing Sensitivity Analysis Measures

GoldSim provides a number of statistical sensitivity analyses through the multi-variate result display option. If you press the **Sensitivity Analysis…** button from the Multi-Variate Result dialog, a table such as this is displayed:



Y: Sensitivity analysis (based on values). Coefficient of determination = 0.0765536

	Result	Importance Measure	Correlation Coefficient	Regression Coefficient	Partial Coefficient
1	X1	0.011	0.024	0.022	0.023
2	X2	0.840	-0.015	-0.005	-0.006
3	X3	0.108	0.276	0.275	0.275

This table displays measures of the sensitivity of the selected result (the output from which you selected **Multi-Variate Result…**) to the selected input variables.

Note: You can control whether the sensitivity analyses are based on the values or the ranks of the variables and the result via the **Use Ranks** button at the top of the display.

The measures that GoldSim computes are:

- Coefficient of determination;
- Correlation coefficients;
- Standardized regression coefficients (SRC);
- Partial correlation coefficients; and
- Importance measures.

The manner in which each of these measures is computed is described below.

Read More: *Viewing a Sensitivity Analysis Table* on page 747

Computing the Coefficient of Determination

The coefficient of determination represents the fraction of the total variance in the result that can be explained based on a linear (regression) relationship to the input variables (i.e., Result = aX + bY + cZ + …). The closer this value is to 1, the better that the relationship between the result and the variables can be explained with a linear model.

The formulation for the coefficient of determination (R^2) can be found in Iman (1985). Note that if all of the variables on uncorrelated, then:

$$R^2 = \sum_i C_i$$

where Ci is the correlation coefficient for variable i.

Computing Correlation Coefficients

Rank (Spearman) or value (Pearson) correlation coefficients range between -1 and +1, and express the extent to which there is a linear relationship between the selected result and an input variable.

The value correlation coefficient is computed as follows:

$$C_{rp} = \frac{\sum_{i-1}^{n}(p_i - m_p)(r_i - m_r)}{\sqrt{\sum_{i=1}^{n}(p_i - m_p)^2 \sum_{i=1}^{n}(r_i - m_r)^2}}$$

where:

C_{rp} is the value correlation coefficient between r and p;
n is the number of selected data points (realizations);
p_i is the value of output p for realization i;
r_i is the value of output r for realization i;
m_p is the mean value of output p; and
m_r is the mean value of output r.

Note that the value correlation coefficient as defined here provides a measure of the linear relationship between two variables.

Furthermore, it may be affected by a few aberrant pairs (i.e., a good alignment of a few extreme pairs can dramatically improve an otherwise poor correlation coefficient; likewise, an otherwise good correlation could be ruined by the poor alignment of a few extreme pairs).

To overcome these problems, the value correlation coefficient can be supplemented by the rank correlation coefficient.

The rank correlation coefficient (also referred to as the ***Spearman rank correlation coefficient***) is computed using the same equation as that of the value correlation coefficient with the ranks of the data values being used rather than the actual values:

$$\mathbf{C_{rp,rank}} = \frac{\sum_{i-1}^{n}(\mathbf{Rp_i} - \mathbf{m_{Rp}})(\mathbf{Rr_i} - \mathbf{m_{Rr}})}{\sqrt{\sum_{i=1}^{n}(\mathbf{Rp_i} - \mathbf{m_{Rp}})^2 \sum_{i=1}^{n}(\mathbf{Rr_i} - \mathbf{m_{Rr}})^2}}$$

where:

$C_{rp,rank}$ is the rank correlation coefficient between r and p;
n is the number of selected data points (realizations);
Rp_i is the rank (from 1 to n) of output p for realization i;
Rr_i is the rank (from 1 to n) of output r for realization i;
m_p is the mean value of the rank of output p; and
m_r is the mean value of the rank of output r.

In GoldSim, the ranks of equal values are the same. For example, if the lowest two realizations of an output were the same, their ranks would both be 1.5 (the mean of 1 and 2), with the third lowest value being assigned a rank of 3.

Note: A correlation coefficient cannot be computed for a pair of outputs if one of the outputs has a standard deviation of zero (i.e., is constant). In this case, GoldSim sets the correlation coefficient to zero.

Computing Standardized Regression Coefficients (SRC)

Standardized regression coefficients range between -1 and +1 and provide a normalized measure of the linear relationship between variables and the result. They are the regression coefficients found when all of the variables (and the result) are transformed and expressed in terms of the number of standard deviations away from their mean. GoldSim's formulation is based on Iman et al (1985).

The use of standardized regression coefficients to identify the importance of correlated input variables is described in Iman et al (1985). In this approach the correlation matrix C for the input variables is augmented, with an additional row/column assigned for the result (GoldSim uses the first row/column for this purpose).

The standardized regression coefficients are based on the inverse of the augmented correlation matrix, and are found by dividing the terms in the augmented column of the matrix by the negative of the augmented diagonal term:

$$\mathbf{SRC_{y,i}} = \frac{-\mathbf{c_{iy}}}{\mathbf{c_{yy}}}$$

where:

$SRC_{y,i}$ is the standardized regression coefficient of the result (y) with respect to input variable X_i;
c_{iy} is the off-diagonal term in the inverted correlation matrix for row i, column y; and
c_{yy} is the diagonal term in the inverted correlation matrix corresponding to the result y.

Computing Partial Correlation Coefficients

Partial correlation coefficients reflect the extent to which there is a linear relationship between the selected result and an input variable, after removing the effects of any linear relationships between the other input variables and both the result and the input variable in question. For systems where some of the input variables may be correlated, the partial correlation coefficients represent the "unique" contribution of each input to the result. GoldSim's formulation is based on Iman et al (1985).

The partial correlation coefficient $P_{y,i}$ is calculated as:

$$\mathbf{P_{y,i}} = \frac{-\mathbf{c_{iy}}}{\sqrt{\mathbf{c_{ii}\,c_{yy}}}}$$

where:

$P_{y,i}$ is the partial correlation coefficient of the result (y) to input variable X_i;
c_{iy} the off-diagonal term in the inverted correlation matrix for row i, column y; and
c_{ii} and c_{yy} are the diagonal terms in the inverted correlation matrix corresponding to the input variable and the result, respectively.

Note that if any two or more of the input variables are linearly dependent, for example if they are perfectly correlated, then the correlation matrix becomes singular and cannot be inverted. GoldSim, which uses Choleski decomposition to compute the inverse, will automatically adjust the correlation matrix if necessary in order to compute the partial correlation coefficients. This adjustment, which takes the form of 'weakening' of the off-diagonal terms, does not affect the displayed correlation coefficients.

Computing Importance Measures

If there is a nonlinear, non-monotonic relationship between an input variable and the result, conventional correlation coefficients may not reveal the relationship. The Importance Measure computed by GoldSim is a normalized version of the measure E[V(Y|Xi)] discussed in Saltelli and Tarantola (2002). The Saltelli and Tarantola measure represents the expected value of the variance if the input variable Xi was not uncertain. Thus, the smaller this value, the more the input variable controls the result.

The GoldSim version of this measure is normalized as follows:

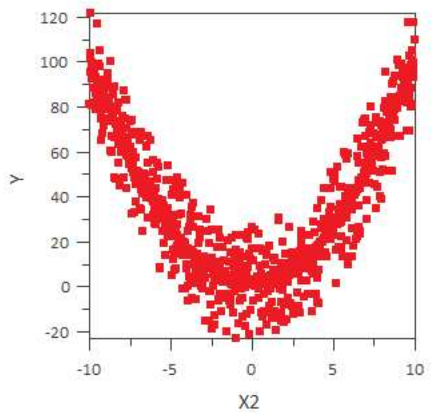
$$\mathbf{M_{y,i}} = \mathbf{1} - \frac{\mathbf{E\left[V_y(Y|X_i)\right]}}{\mathbf{V_y}}$$

where:

$M_{y,i}$ is the GoldSim importance measure for the sensitivity of the result (y) to input variable X_i;
V_y is the current variance in the result y; and
$E[V_y(Y|X_i)]$ is the expected value of V_y if the input variable X_i was perfectly known.

Thus, GoldSim's Importance Measure $M_{y,i}$ represents the fraction of the result variance that is explained by X_i. Note that, like the correlation coefficients and scatter-plots, the importance measure can be calculated using either values or ranks, as specified in the main property window for the multivariate element.

GoldSim calculates $M_{y,i}$ numerically, using the following method: Construct a 2-d scatter plot of the results, with the X_i values on the horizontal axis and the result on the vertical axis. If X_i is very important to the result, then this plot will show a clustering around a central line or curve, such as in this example (in which X_2 is important):



Subdivide the plot into N_s vertical segments based on the ranks of the X_i-values, where N_s equals the square root of the number of model realizations. For each of the segments, estimate the variance of Y within that segment by using a weighting function that assigns decreasing weights to the results as their distance from the center of the segment increases. Then compute the average value of the variance over all of the segments, i.e. $E[V_y (Y|X_i)]$. For the weighting function, GoldSim uses the density of a beta distribution having a mean equal to the X-value at the center of the segment, a standard deviation equal to the segment width, and upper and lower bounds corresponding to the range of X-values.

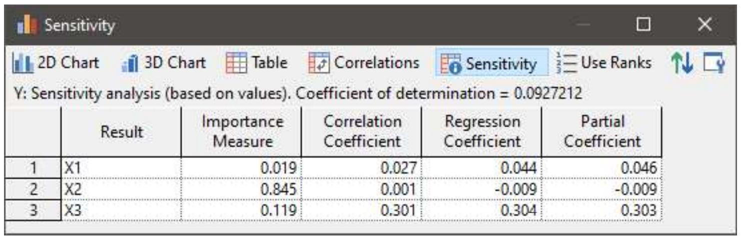
The example plot above is based on calculating:

$$\mathbf{Y = X_1 + X_1^2 + X_3^3}$$

where:

X_1 is a random variable with a U(1,2) distribution;
X_2 is a random variable with a U(-10,10) distribution; and
X_3 is a random variable with a U(-3,3) distribution.

The plot shown above, plotting Y vs X_2, clearly reveals the importance of X_2 in this model. Conventional correlation analysis is completely unable to recognize this sensitivity, as indicated by the correlation coefficient of effectively zero (0.001) in the screen-shot below. However, the importance measure for X_2 is very high (0.845). That is, X_2 is identified as the most important variable:



Y: Sensitivity analysis (based on values). Coefficient of determination = 0.0927212

	Result	Importance Measure	Correlation Coefficient	Regression Coefficient	Partial Coefficient
1	X1	0.019	0.027	0.044	0.046
2	X2	0.845	0.001	-0.009	-0.009
3	X3	0.119	0.301	0.304	0.303

Appendix B References

The references cited in this appendix are listed below.

Cox, D.R. and H.D. Miller, 1965. The Theory of Stochastic Processes, Chapman and Hall, New York.

Benjamin, J.R. and C.A. Cornell, 1970. Probability, Statistics, and Decision for Civil Engineers, McGraw-Hill, New York.

Embrechts, Paul., Lindskog, Filip and Alexander McNeil, 2001, "Modelling Dependence with Copulas and Applications to Risk Management," Department of Mathematics, Swiss Federal Institute of Technology, Zurich (http://www.defaultrisk.com/pp_corr_19.htm).

Iman, R.L. 1982. Statistical Methods for Including Uncertainties Associated with the Geologic Isolation of Radioactive Waste Which Allow for a Comparison with Licensing Criteria. Proceedings of the Symposium on Uncertainties Associated with the Regulation of the Geologic Disposal of High-Level Radioactive Waste, Gatlinburg, Tennessee, March 9-13, 1981. Kocher, D.C., ed. NUREG/CP-0022. 145-157. Washington, D.C.: U.S. Nuclear Regulatory Commission. TIC: 213069.

Iman, R.L. and W.J. Conover, 1982. A Distribution-Free Approach to Inducing Rank Correlation Among Input Variables, Communications in Statistics: Simulation and Computation, 11(3), pp 311-334,

Iman, R.L. et al., 1985. A FORTRAN Program and User's Guide for the Calculation of Partial Correlation and Standardized Regression Coefficients, NUREG/CR-4122, SAND85-0044.

L'Ecuyer, P., 1988, Communications of the ACM, vol. 31, pp. 742-744.

McKay, M.D., Conover, W. J., and Beckman, R. J., 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code, Technometrics, 21, 239-245.

Saltelli, A. and S. Tarantola, 2002. On the Relative Importance of Input Factors in Mathematical Models: Safety Assessment for Nuclear Waste Disposal, J. Am. Stat. Ass., Vol. 97, No. 459.

Appendix C: Implementing External (DLL) Elements

Begin at the beginning and go on till you come to the end; then stop.

Lewis Carroll, *Alice in Wonderland*

Appendix Overview

GoldSim allows you to develop separate program modules (written in C, C++, Pascal, FORTRAN or other compatible programming languages) which can then be directly coupled with the main GoldSim algorithms. These user-defined modules are referred to as external functions, and are linked into GoldSim as DLLs (Dynamic Link Libraries) at run time. GoldSim interfaces with the DLL via an External element.

Read More: *External (DLL) Elements* on page 995

Integrating your external program module into GoldSim requires that you develop a "wrapper" or "shell" around the function and compile it into a DLL. This appendix discusses the details of how external functions must be coded and compiled.

In this Appendix

This appendix discusses the following:

- Understanding External (DLL) Elements
- Implementing an External Function
- External Function Examples
- External Function Calling Sequence
- DLL Calling Details

Understanding External (DLL) Elements

with External elements to do calculations or other manipulations that are not included in the standard capabilities of GoldSim. The external function facility allows special purpose calculations or manipulations to be accomplished with more flexibility, speed or complexity than with the standard GoldSim element types.

The external functions are bound to the GoldSim executable code at run time using DLL technology. The DLL files should be present in the same folder as the GoldSim .gsm file, in the same folder as the GoldSim executable file, or elsewhere in the user’s path.

Note that these functions are external to GoldSim and are not covered by the standard GoldSim verification process. The user is responsible for any necessary testing of external functions.

Every external function is called by GoldSim with specific requests. The requests include initialization, returning the function version number, performing a normal calculation, and "cleaning up" after a simulation. The function name and argument list (the set of input and output data for the function) are specified by the GoldSim user when setting up the External element.

External functions should provide their own error handling, message handling, file management and memory management if required. It is essential that when it receives a “clean up” request, an external function should release any dynamically acquired memory and close any open files.

Note: In the case of an error condition, the external function should always return an error code to GoldSim, so that the user can be informed about the problem and the simulation can be terminated cleanly with no memory leaks.

Implementing an External Function

Important Restrictions

The implementation of the external function is left to the programmer, but several restrictions apply so that the functions can be correctly called by GoldSim. They are:

- The function return value is ignored. For example, use void functions in C/C++, or subroutines in FORTRAN.
- Data are passed from GoldSim to the external function and back again to GoldSim via arrays of double precision floating point input and output arguments.
- Input arguments must not be modified by the external function. Doing so may cause memory corruption in the DLL and/or GoldSim.
- Each function must manage its own initialization and memory allocation, and its own messages to the user (if any).
- Unique external function (or subroutine) names must be defined in each DLL. In C++, function names are case-sensitive, while in FORTRAN, the case of the external subroutine name is determined from the DLL build options. In all instances, the function name specified in GoldSim is case-sensitive, and must agree with the case specified in the DLL.

- All files required to run your specific DLL must be properly installed on the machine where GoldSim is running. This includes any additional runtime libraries required by your DLL.
- Most development environments allow both ***static*** and ***dynamic*** binding of runtime libraries to DLLs. DLLs built with static binding are stand-alone, with all run-time library references included in the DLL file. However, if a DLL is built with dynamic binding, the runtime libraries are not included. For a DLL with dynamic binding, the runtime libraries (e. g. msvcrt.dll for Visual C++ or libifcoremd.dll for Intel Visual Fortran) must be present and in the environment PATH variable on the machine where GoldSim is running.

External Function Format

When calling methods from the DLL, GoldSim always expects the following C/C++ function signature:

```
extern "C" void __declspec(dllexport)
MyExternalFcn(int XFMethod,
int*    XFState,
double* inargs,
double* outargs)
```

The extern "C" specifies C language style linkage between GoldSim and the DLL, and __declspec(dllexport) makes the function visible outside the DLL.

For Intel Visual Fortran, the following subroutine signature can be used:

```
subroutine my_external_fcn(xfmethod, xfstate, inargs, outargs)
!DEC$ ATTRIBUTES dllexport,c :: add_mult_scalars
!DEC$ ATTRIBUTES value       :: nmethod
!DEC$ ATTRIBUTES reference   :: nstatus
!DEC$ ATTRIBUTES reference   :: dinputs
!DEC$ ATTRIBUTES reference   :: doutputs
```

Other FORTRAN development environments may require different attributes for the proper linkage to GoldSim.

Note: Arrays in C/C++ start at zero, rather than one (the default for FORTRAN). Array indices in this section use the C/C++ convention.

The arguments are:

Argument	Definition
int XFmethod;	Action that the external function must perform (see table below)
int* XFState;	Returned value success 0 or fatal 1
double* inargs;	Array of input arguments
double* outargs;	This array returns different information when different XFmethod values are passed to the external function

The values for XFmethod are:

0	XF_INITIALIZE	Initialize (called at the beginning of each realization). No arguments are passed on this call.
1	XF_ CALCULATION	Normal calculation. Total number of output arguments are returned on this call. outargs[0] = 1st result from the function; outargs[1] = 2nd result from the function; etc.
2	XF_REP_ VERSION	External functions report their versions. No input arguments are passed on this call. outargs[0] = external function version, e.g., 1.10
3	XF_REP_ ARGUMENTS	External functions report the # of input and output arguments. No input arguments are passed on this call. outargs[0] = # of input arguments. outargs [1] = # of output arguments.
99	XF_CLEANUP	Close any open files, and optionally release dynamic memory at the end of a simulation. No arguments are passed on this call.

The returned values for XFState are:

0	OK, continue GoldSim
> 0 and < 99	terminate GoldSim
99	OK, unload the DLL

The following two return codes may only be used for an XF_CALCULATION call:

-1	Fatal error, error message pointer returned
-2	More result memory required; the total amount (in doubles) required is returned in outargs[0]. This may be required of the external function is returning a table or time series definition. (Note that the additional memory is retained until the end of the simulation.)

The memory for the inargs and outargs arrays is allocated by GoldSim at the start of the simulation, based upon the inputs and outputs specified in the Interface tab of the External element properties dialog. The number of input and output arguments is verified during the XF_REP_ARGUMENTS request. GoldSim counts up the number of input and output arguments it expects, and compares it to the numbers for each reported by the DLL.

Warning: It is the responsibility of the external function to ensure that it does not write to any output arguments beyond the number reported for XF_REP_ARGUMENTS. Doing so may cause memory corruption in GoldSim.

Argument Checking

GoldSim calculates the total number of input and output arguments by summing over the the inputs and outputs specified on the Interface tab of the External element properties dialog. Note that each scalar input or output counts as one argument, while array inputs or outputs count as the size of the array (rows * columns). The calculated totals are then compared to the external function’s reported number of input and output arguments. If the number of arguments defined by GoldSim does not agree with the number reported by the external function, GoldSim terminates with an error message.

However, note the following exceptions:

- If outargs[0] is returned as -1, the number of input arguments is not tested. This allows for functions where the DLL does not know in advance the number of input argument that it will receive.
- If the external function will be returning definitions for one or more Tables or Time Series (see below), GoldSim will not know in advance how long the Table definitions will be. In this case, the external function should specify a value for outargs[1] greater than or equal to the actual total number of arguments that may be returned. GoldSim will allocate this amount of memory for outargs. Note that this can be reset during the simulation by returning XFState=-2.

The Input and Output Argument Arrays

The content of input and output argument arrays is determined from the Interface tab of the External element properties dialog.

The following points should be noted:

- Input or outputs are processed in the order specified in the interface. All data from one input or output is contiguous, followed by the data from the next input or output.
- Scalar inputs and outputs are mapped to a single array argument.
- Vector input and output items are specified in order, one argument per item, according to the order specified in the array label.
- Matrix input and output items are specified item-by-item, with all items in one row together, followed by all items in the next row, and so on.
- Lookup Table definitions can be specified in the output interface, via a special format.
- Time Series definitions can be specified as inputs or outputs, also via a special format.

Lookup Table Definitions

External functions can also return Table definition elements to GoldSim. A table definition requires a specific sequence of values, depending whether it is a 1-D, 2-D, or 3-D table.

The sequence for a 1-D table is as follows:

- number of dimensions (in this case, 1)
- the number of rows
- row value1, row value 2, …, row value n
- dependent value 1, dependent value 2, …, dependent value n

The sequence for a 2-D table is as follows:

- number of dimensions (in this case, 2)
- the number of rows, the number of columns
- row value1, row value 2, …, row value n
- column value 1, column value 2, …, column value n
- dependent(row 1, column 1), …, dependent(row 1,column n)
- dependent(row 2, column 1), …, dependent(row 2,column n)

- …
- dependent(row n, column 1), …, dependent(row n, column n)

Warning: This is different than the sequence used to read in an ASCII text file for a 2-D table.

The sequence for a 3-D table is as follows:

- number of dimensions (must be 3)
- the number of rows, the number of columns, the number of layers
- row value1, row value 2, …, row value y
- column value 1, column value 2, …, column value x
- layer value1, layer value 2, …, layer value z
- dependent(row 1, column 1, layer 1), …, dependent(row 1,column x, layer 1)
- dependent(row 2, column 1, layer 1), …, dependent(row 2,column x, layer 1)
- …
- dependent(row y, column 1, layer 1), …, dependent(row y, column x, layer 1)
- .
  .
  .
- dependent(row 1, column 1, layer z), …, dependent(row 1,column x, layer z)
- dependent(row 2, column 1, layer z), …, dependent(row 2,column x, layer z)
- …
- dependent(row y, column 1, layer 1), …, dependent(row y, column x, layer z)

Warning: This is different than the sequence used to read in an ASCII text file for a 3-D table.

Time Series Definitions

External functions can also read and return Time Series Definition. A Time Series Definition consists of the following specific sequence of values.

1. The number 20 (this informs GoldSim that this is a Time Series)
2. The number -3 (this is a format number that infoms GoldSim what version of the time series format is expected)
3. Calendar-baed index: 0 if elapsed time; 1 if dates
4. An index (0,1,2,3) indicating what the data represents (0=instantaneous value, 1=constant value over the next time interval, 2=change over the next time interval, 3=discrete change)
5. The number of rows (0 for scalar time series)
6. The number of columns (0 for scalar and vector time series)
7. Number of series

8. For each series, the following is repeated:
    - The total number of time points in the series
    - Time point 1, Time point 2, …, Time point n

    The structure of the remainder of the file depends on whether the Time Series Definition represents a scalar, a vector, or a matrix.

    For a scalar, the next sequence of values is as follows:
    - Value 1[time point 1], Value 2[time point 2], …, Value[time point n]

    For a vector, the next sequence of values is as follows:
    - Value[row1, time point 1], Value[row1, time point 2], …, Value[row1, time point n]
    - Value[row2, time point 1], Value[row2, time point 2], …, Value[row2, time point n]
    - …
    - Value[rowr, time point 1], Value[rowr, time point 2], …, Value[rowr, time point n]

    For a matrix, the next sequence of values is as follows:
    - Value[row1, column1, time point 1], Value[row1, column1, time point 2], …, Value[row1, column1, time point n]
    - Value[row1, column2, time point 1], Value[row1, column2, time point 2], …, Value[row1, column2, time point n]
    - …
    - Value[row1, columnc, time point 1], Value[row1, columnc, time point 2], …, Value[row1, columnc, time point n]
    - .
      .
      .
    - Value[rowr, column1, time point 1], Value[rowr, column1, time point 2], …, Value[rowr, column1, time point n]
    - Value[rowr, column2, time point 1], Value[rowr, column2, time point 2], …, Value[rowr, column2, time point n]
    - …
    - Value[rowr, columnc, time point 1], Value[rowr, columnc, time point 2], …, Value[rowr, columnc, time point n]

External Function Examples

The following is a simple example implementation of DLL code that will work with an External element. This code takes two scalar elements as input, and returns the sum and product to GoldSim. For simplicity, this code is written in C, implemented with Visual C++. This external function can be used with the External.gsm example which can be found in the External subfolder of the General Examples folder in your GoldSim directory (accessed by selecting **File | Open Example...** from the main menu).

```
// Global enumerations, useful for C-style implementations
//
// XFMethodID identifies the method types, used to identify
// the phase of the simulation that is currently in progress.
//
//    XF_INITIALIZE    - Called after DLL is loaded and before each realization.
//    XF_CALCULATE     - Called during the simulation, each time the inputs change.
//    XF_REP_VERSION   - Called after DLL load to report the external fcn version number.
//    XF_REP_ARGUMENTS - Called after DLL load to report the number of input
//                       and output arguments.
//    XF_CLEANUP       - Called before the DLL is unloaded.
//
enum XFMethodID
{
XF_INITIALIZE = 0, XF_CALCULATE = 1, XF_REP_VERSION = 2, XF_REP_ARGUMENTS = 3,
XF_CLEANUP = 99
};
// XFStatusID identifies the return codes for external functions.
//
//     XF_SUCCESS          - Call completed successfully.
//     XF_CLEANUP_NOW      - Call was successful, but GoldSim should clean up
//                           and unload the DLL immediately.
//     XF_FAILURE          - Failure (no error information returned).
//     XF_FAILURE_WITH_MSG - Failure, with DLL-supplied error message available.
//                           Address of error message is returned in the first element
//                           of the output arguments array.
//     XF_INCREASE_MEMORY  - Failed because the memory allocated for output arguments
//                           is too small.  GoldSim will increase the size of the
//                           output argument array and try again.
//
enum XFStatusID
{
XF_SUCCESS = 0, XF_FAILURE = 1, XF_CLEANUP_NOW = 99, XF_FAILURE_WITH_MSG = -1,
XF_INCREASE_MEMORY   = -2
};
////////////////////////////////////////////////////////////////////////////////
// AddMultScalarsInC
//     Adds and multiplies two input scalar values (C Language implementation).
//-----------------------------------------------------------------------------
extern "C" void __declspec(dllexport) AddMultScalarsInC(int     methodID,
int*    status,
double* inargs,
double* outargs)
{
*status = XF_SUCCESS;
switch ( methodID )
{
case  XF_INITIALIZE:
break;                              // nothing required
case  XF_REP_VERSION:
outargs[0] = 1.03;
break;
case  XF_REP_ARGUMENTS:
outargs[0] = 2.0;                   // 2 scalar inputs expected
outargs[1] = 2.0;                   // 2 scalar outputs returned
break;
case  XF_CALCULATE:
outargs[0] = inargs[0] + inargs[1];   // return the sum
outargs[1] = inargs[0]*inargs[1];     // return the product
break;
case  XF_CLEANUP:
break;                                // No clean-up required
default:
*status = XF_FAILURE;
break;
}
}
```

The following code is the same algorithm, implemented in Intel Visual Fortran.

```
! Utility module to specify the GoldSim parameter constants
module gs_parameters
implicit none
! Parameters to identify the method types, which indicate the phase of the
! simulation that is currently in progress.
!
! INITIALIZE       - Called after DLL is loaded and before each realization.
! CALCULATE        - Called during the simulation, each time the inputs change.
! REPORT_VERSION   - Called after DLL load to report the external fcn version number.
! REPORT_ARGUMENTS - Called after DLL load to report the number of input and output
!                    arguments.
! CLEANUP          - Called before the DLL is unloaded.
integer(4), parameter :: INITIALIZE       =  0
integer(4), parameter :: CALCULATE        =  1
integer(4), parameter :: REPORT_VERSION   =  2
integer(4), parameter :: REPORT_ARGUMENTS =  3
integer(4), parameter :: CLEANUP          = 99
! Parameters to identify the return codes for external functions.
!
! SUCCESS          - Call completed successfully.
! CLEANUP_NOW      - Call was successful, but GoldSim should clean up
!                    and unload the DLL immediately.
! FAILURE          - Failure (no error information returned).
! FAILURE_WITH_MSG - Failure, with DLL-supplied error message available.
!                    Address of error message is returned in the first element
!                    of the output arguments array.
! INCREASE_MEMORY  - Failed because the memory allocated for output arguments
!                    is too small. GoldSim will increase the size of the output
!                    argument array and try again.
integer(4), parameter :: SUCCESS          =  0
integer(4), parameter :: FAILURE          =  1
integer(4), parameter :: CLEANUP_NOW      = 99
integer(4), parameter :: FAILURE_WITH_MSG = -1
integer(4), parameter :: INCREASE_MEMORY  = -2
end module gs_parameters
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! add_mult_scalars
!     Adds and multiplies two input scalar values.
!-----------------------------------------------------------------------
subroutine add_mult_scalars(method_id, status, inargs, outargs)
!DEC$ ATTRIBUTES dllexport,c :: add_mult_scalars
!DEC$ ATTRIBUTES value       :: method_id
!DEC$ ATTRIBUTES reference   :: status
!DEC$ ATTRIBUTES reference   :: inargs
!DEC$ ATTRIBUTES reference   :: outargs
use gs_parameters
implicit none
real(8),    parameter :: VERSION  = 1.03
integer(4), parameter :: NINPUTS  = 2 ! Two scalar inputs expected
integer(4), parameter :: NOUTPUTS = 2 ! Two scalar outputs returned
integer(4) method_id, status
real(8)    inargs(*), outargs(*)
select case (method_id)
```

```
case (INITIALIZE)
status = SUCCESS
case (REPORT_VERSION)
outargs(1) = VERSION
status = SUCCESS
case (REPORT_ARGUMENTS)
outargs(1) = NINPUTS
outargs(2) = NOUTPUTS
status = SUCCESS
case (CALCULATE)
outargs(1) = inargs(1) + inargs(2)      ! return the sum
outargs(2) = inargs(1)*inargs(2)        ! return the product
status = SUCCESS
case (CLEANUP)
status = SUCCESS
case default
status = FAILURE
end select
end subroutine add_mult_scalars
```

Additional DLL code examples can be found (including examples for arrays, Lookup Tables, and Time Series Elements) can be found in the GoldSim install directory. In addition to the source files, example solution and project files for Microsoft Visual C++ and Intel Visual Fortran are also included (under General Examples/External).

External Function Calling Sequence

GoldSim makes calls to the DLL external function at different times during the course of a GoldSim simulation:

- Before the simulation starts (while checking model integrity).
- The first time that the External element values are calculated.
- Every time that the input values of the External element change.
- Before each (subsequent) realization.
- After each realization (Cleanup After Realization option only).
- After the simulation finishes.
- If any DLL external function call returns a value of 99.

GoldSim users can control when the DLL is unloaded via the Unload After Each Use and Cleanup After Realization options. These options are selected via checkboxes in the External element properties dialog ("Unload DLL after each use" and "Run Cleanup after each realization"). As the name implies, Unload After Each Use will clean up (XFMethod = XF_CLEANUP) and unload the DLL after each calculation call (XFMethod = XFCalculate). Similarly, Cleanup After Realization will clean up and unload the DLL at the end of each realization (if the DLL is loaded).

The DLL external function code can also control when the DLL is unloaded. If any call to a DLL external function returns a status of 99, GoldSim will treat the call as a success, but will clean up and unload the DLL immediately after processing the returned data.

Before the Simulation

GoldSim calls the DLL external function before the simulation starts, as part of the process to check the validity of the model. The main reason is to get the number of input and output arguments, to insure that they are consistent with what is specified in the Interface tab. The call sequence is as follows:

1. Load the DLL and check for the existence of the external function
2. Ask for the version number (XFMethod = XF_REP_VERSION).
3. Ask for the number of input and output arguments (XFMethod = XF_REP_ARGUMENTS), and compare to the values determined from the External element interface.
4. Clean up (XFMethod = XF_CLEANUP) and unload the DLL.

If any of these calls fail, or if the number of input or output arguments in step 3 are inconsistent, GoldSim will display an error message and return to the previous state (Edit or Ready mode).

During Each Realization

During each realization, GoldSim will call the DLL external function when the corresponding External element needs to be updated. This happens the first time that the Element is referenced, and every subsequent time-step or event update where an input to the Element changes. In this case, the following call sequence is used:

1. Check to see if the DLL is loaded. If so, skip to step 6.
2. Load the DLL and check for the existence of the external function
3. Ask for the version number (XFMethod = XF_REP_VERSION), and write it to the log file.
4. Ask for the number of input and output arguments (XFMethod = XF_REP_ARGUMENTS).
5. Initialize the DLL (XFMethod = XF_INITIALIZE).
6. Calculate (XFMethod = XF_CALCULATE)
7. If Unload After Each Use is specified, clean up (XFMethod = XF_CALCULATE) and unload the DLL.

Before Each Realization

Before each realization, GoldSim will check to see if the DLL is loaded for each External element. If the DLL is loaded, GoldSim will reinitialize the DLL (XFMethod = XF_INITIALIZE).

After Each Realization

If the Cleanup After Realization option is specified, and the DLL is loaded, GoldSim will clean up (XFMethod = XF_CLEANUP) and unload the DLL after each realization.

After the Simulation

After the simulation completes (either successfully or after a fatal error), GoldSim will clean up (XFMethod = XF_ CLEANUP) and unload the DLL if it is still loaded.

DLL Calling Details

GoldSim can call DLL external functions by two different mechanisms, depending upon the Separate Process Space option. For each External element, the GoldSim user can select this option in the properties dialog, via the "Run in separate process space" checkbox. If this option is not selected, the DLL will be loaded with the Win32 LoadLibrary function. As a result, this DLL will share the same memory address space as the GoldSim process, and any memory used in the DLL will be charged to the GoldSim process. By default, External elements are created without the Separate Process Space option enabled.

If the Separate Process Space option is selected, GoldSim will call the DLL using a Component Object Model (COM) interface. The interface is implemented as a COM LocalServer executable, which is started when GoldSim first requests to load the DLL. Once the LocalServer is started, it loads the DLL into its own memory address space (separate from GoldSim), and acts a proxy between GoldSim and the DLL for all external function requests. After the DLL is unloaded, GoldSim frees the COM interface and the LocalServer executable terminates. Because this option loads the DLL into a separate memory space, it may be a better option for DLLs with a large memory footprint.

64-Bit DLL Support

GoldSim also supports loading of external DLLs that are built as 64-bit libraries. The main advantage for a 64-bit DLL is a significant increase in the amount of virtual memory available (from 4GB to 8TB). Migrating DLL code from 32-bit to 64-bit requires minimal (if any) changes; just install the 64-bit compilers for Visual C++ or Visual Fortran and build with a 64-bit target. No change is GoldSim model configuration is required to use 64-bit DLLs, since GoldSim will automatically determine the type of DLL and call the appropriate interface. However, the following caveats do apply when using 64-bit DLLs in GoldSim:

- 64-bit DLLs can only be run on a computer with a 64-bit Windows operating system. If a DLL needs to run on both 32-bit and 64-bit Windows, it should be a 32-bit DLL (which will run on both 64-bit and 32-bit OS).
- 64-bit DLLs must run in a separate process space.
- For Distributed Processing runs, models that contain 64-bit DLLs can only be launched from a 64-bit Windows operating system. GoldSim will inspect the model to see if it contains a 64-bit DLL, and will disconnect all slaves that are running a 32-bit Windows OS.

Returning Error Messages from External Functions

To help GoldSim users debug problems with DLL external functions, GoldSim lets users send an error message from the DLL back to GoldSim through the External element interface when the call to an external function fails. The error message is then displayed to the user in a pop-up dialog.

The DLL external function signals the presence of an error message by returning a status value of -2. When GoldSim processes the results of the DLL external function call, it will interpret the first element of the output arguments arrray (outargs in our source-code example) as a pointer to a memory location where the error string can be found. The memory containing the string must have static scope, so that it will still be available when GoldSim retrieves the string. The string must also be NULL-terminated, even when returning from a FORTRAN DLL. If either of these recommendations are not followed, GoldSim will likely crash when it tries to display the error message.

The following code is an example of a C language function that will properly handle passing a message from a DLL external function to GoldSim. The ULONG_PTR is cast to different types on 64-bit (unsigned long) and 32-bit (unsigned __int64), so that it will work for building both 32-bit and 64-bit binaries.

```
// Utility method used to simplify the sending of an error message to GoldSim
void CopyMsgToOutputs(const char* sMsg, double* outargs)
{
// Static character array used to hold the error message.
// This needs to be static so that it will be "in scope" when it is read by GoldSim.
static char sBuffer[200];
// Clear out any old data from the buffer
memset(sBuffer, 0, sizeof(sBuffer));
// Cast the first output array element as a pointer.
// ULONG_PTR is used because it will work for both 32-bit and 64-bit DLLs
ULONG_PTR* pAddr = (ULONG_PTR*) outargs;
// Copy the string data supplied into the static buffer.
// Safer version of string copy is used. If your compiler does not support this you
// should remove the comments in front of the next line and add to the next one.
// strncpy(sBuffer, sMsg, sizeof(sBuffer) - 1);
strncpy_s(sBuffer, sMsg, sizeof(sBuffer) - 1);
// Copy the static buffer pointer to the first output array element.
*pAddr = (ULONG_PTR) sBuffer;
}
```

For FORTRAN DLLs, the following code performs the same function:

```
! Utility subroutine to simplify sending an error message to GoldSim
subroutine copy_msg_to_outputs(smsg, outargs)
implicit none
character(*) smsg
real(8) outargs(*)
! "Static" character buffer, used to store the error message so
! that it can be returned to GoldSim.
character(80), save :: sbuffer
! Create a shared memory variable that can be interpreted either as integer or real.
integer(8) ioutput1
real(8)    doutput1
equivalence(ioutput1, doutput1)
! Copy the message into the buffer.  Truncate it if it is too long
! Make sure that it is null terminated!
if (len(smsg) .lt. 80) then
sbuffer = smsg // char(0)
else
sbuffer = smsg(1:79) // char(0)
end if
! Since we are sending back to C++, we need an actual address.
! The "loc" function is not standard, but it is supported by all
! compilers that we checked.
ioutput1 = loc(sbuffer)
outargs(1) = doutput1
end subroutine copy_msg_to_outputs
```

Appendix D: GoldSim Units Database

364.4 Smoots plus 1 ear.

Official length of the Harvard Bridge

Appendix Overview

One of the more powerful features of GoldSim is that it is dimensionally-aware. You enable this capability by assigning dimensions to the outputs (and hence to the inputs) of the elements in your model. GoldSim has an extensive internal database of units and conversion factors. This appendix lists all of the units and conversion factors that are built into GoldSim.

Built-in Units and Conversion Factors

All units in GoldSim are defined relative to the following basic units:

- meter (m)
- kilogram (kg)
- second (s)
- Kelvin temperature (K)
- ampere (amp)
- radian (rad)
- Candela (cd)
- Mole (mol)
- Item (item)

The following table summarizes all of the built-in units and conversion factors within GoldSim, organized by category:

Category	Unit	Abbreviation	Definition
Acceleration	Mean Acceleration of earth’s gravity	gee	9.80665 m/s2
Angle	Degree of Arc	°	0.017453293 rad
Angle	One cycle/revolution	cycle	6.2831853 rad
Angle	Degree of arc	deg	0.017453293 rad
Angle	Minute of Arc	minarc	0.00029088821 rad
Angle	Radian	rad	1 rad
Angle	Revolution (cycle)	rev	6.2831853 rad
Angle	Second of Arc	secarc	4.8481368E-06 rad
Angular Frequency	Revolutions per minute	pm	6°/s
Area	Acre	ac	4046.856422 m2
Area	Acre	acre	4046.856422 m2
Area	Hectare	ha	10000 m2
Capacitance	Farad	Fa	1 s4-amp2/kg-m2
Charge	Coulomb of Charge	Co	1 s-amp
Charge	GigaCoulomb of Charge	GCo	1E+9 s-amp

Category	Unit	Abbreviation	Definition
Charge	KiloCoulomb of charge	kCo	1E+3 s-amp
Charge	MegaCoulomb of charge	MCo	1E+6 s-amp
Charge	MicroCoulomb of charge	uCo	1E-06 s-amp
Charge	MilliCoulomb of charge	mCo	1E-3 s-amp
Charge	NanoCoulomb of charge	nCo	1E-09 s-amp
Charge	PicoCoulomb of charge	pCo	1E-12 s-amp
Charge	TeraCoulomb of charge	TCo	1E+12 s-amp
Currency	US Dollar	$	user defined
Currency	Euro	EUR	user defined
Currency	British Pound	GBP	user defined
Currency	Japanese Yen	YEN	user defined
Currency	Australian Dollar	AUD	user defined
Currency	Brazilian Real	BRL	user defined
Currency	Canadian Dollar	CAD	user defined
Currency	Chinese Yuan	CNY	user defined
Currency	Czech Koruna	CZK	user defined
Currency	Danish Krona	DKK	user defined
Currency	Hong Kong Dollar	HKD	user defined
Currency	Hungarian Forint	HUF	user defined
Currency	Mexican Peso	MXN	user defined
Currency	New Zealand Dollar	NZD	user defined
Currency	Norwegian Krone	NOK	user defined
Currency	Russian Rouble	RUB	user defined
Currency	Singapore Dollar	SGD	user defined
Currency	Swedish Krona	SEK	user defined
Currency	Swiss Franc	CHF	user defined
Currency	South African Rand	ZAR	user defined
Currency	Thousand US Dollar	k$	1000 $

Category	Unit	Abbreviation	Definition
Currency	Thousand Euro	kEUR	1000 EUR
Currency	Thousand British Pound	kGBP	1000 GBP
Currency	Thousand Japanese Yen	kYEN	1000 YEN
Currency	Thousand Australian Dollar	kAUD	1000 SUD
Currency	Thousand Brazilian Real	kBRL	1000 BRL
Currency	Thousand Canadian Dollar	kCAD	1000 CAD
Currency	Thousand Chinese Yuan	kCNY	1000 CNY
Currency	Thousand Czech Koruna	kCZK	1000 CZK
Currency	Thousand Danish Krone	kDKK	1000 DKK
Currency	Thousand Hong Kong Dollar	kHKD	1000 HKD
Currency	Thousand Hungarian Forint	kHUF	1000 HUF
Currency	Thousand Mexican Peso	kMXN	1000 MXN
Currency	Thousand New Zealand Dollar	kNZD	1000 NZD
Currency	Thousand Norwegian Krone	kNOK	1000 NOK
Currency	Thousand Russian Rouble	kRUB	1000 RUB
Currency	Thousand Singapore Dollar	kSGD	1000 SGD
Currency	Thousand Swedish Krona	kSEK	1000 SEK
Currency	Thousand Swiss Franc	kCHF	1000 CHF
Currency	Thousand South African Rand	kZAR	1000 ZAR
Currency	Million US Dollar	M$	1E+6 $
Currency	Million Euro	MEUR	1E+6 EUR
Currency	Million British Pound	MGBP	1E+6 GBP
Currency	Million Japanese Yen	MYEN	1E+6 YEN
Currency	Million Australian Dollar	MSUD	1E+6 SUD
Currency	Million Brazilian Real	MBRL	1E+6 BRL
Currency	Million Canadian Dollar	MCAD	1E+6 CAD
Currency	Million Chinese Yuan	MCNY	1E+6 CNY
Currency	Million Czech Koruna	MCZK	1E+6 CZK

Category	Unit	Abbreviation	Definition
Currency	Million Danish Krone	MDKK	1E+6 DKK
Currency	Million Hong Kong Dollar	MHKD	1E+6 HKD
Currency	Million Hungarian Forint	MHUF	1E+6 HUF
Currency	Million Mexican Peso	MMXN	1E+6 MXN
Currency	Million New Zealand Dollar	MNZD	1E+6 NZD
Currency	Million Norwegian Krone	MNOK	1E+6 NOK
Currency	Million Russian Rouble	MRUB	1E+6 RUB
Currency	Million Singapore Dollar	MSGD	1E+6 SGD
Currency	Million Swedish Krona	MSEK	1E+6 SEK
Currency	Million Swiss Franc	MCHF	1E+6 CHF
Currency	Million South African Rand	MZAR	1E+6 ZAR
Current	Ampere	amp	1 amp
Current	GigaAmpere	Gamp	1E+9 amp
Current	KiloAmpere	kamp	1E+3 amp
Current	MegaAmpere	Mamp	1E+6 amp
Current	MicroAmpere	uamp	1E-6 amp
Current	MilliAmpere	mamp	1E-3 amp
Current	NanoAmpere	namp	1E-9 amp
Current	PicoAmpere	pamp	1E-12 amp
Current	TeraAmpere	Tamp	1E+12 amp
Dose	Sievert (dose equivalent)	sV	1 m2/s2
Dose	GigaSievert (dose equivalent)	GSv	1E+9 m2/s2
Dose	KiloSievert (dose equivalent)	kSv	1E+3 m2/s2
Dose	MegaSievert (dose equivalent)	MSv	1E+6 m2/s2
Dose	MicroSievert (dose equivalent)	uSv	1E-6 m2/s2
Dose	MilliSievert (dose equivalent)	mSv	1E-3 m2/s2
Dose	NanoSievert (dose equivalent)	nSv	1E-9 m2/s2
Dose	PicoSievert (dose equivalent)	pSv	1E-12 m2/s2

Category	Unit	Abbreviation	Definition
Dose	TeraSievert (dose equivalent)	TSv	1E+12 m2/s2
Dose	Gray (dose absorbed)	Gy	1 m2/s2
Dose	GigaGray (dose absorbed)	GGy	1E+9 m2/s2
Dose	KiloGray (dose absorbed)	kGy	1E+3 m2/s2
Dose	MegaGray (dose absorbed)	MGy	1E+6 m2/s2
Dose	MicroGray (dose absorbed)	uGy	1E-6 m2/s2
Dose	MilliGray (dose absorbed)	mGy	1E-3 m2/s2
Dose	NanoGray (dose absorbed)	nGy	1E-9 m2/s2
Dose	PicoGray (dose absorbed)	pGy	1E-12 m2/s2
Dose	TeraGray (dose absorbed)	TGy	1E+12 m2/s2
Dose	RAD dose	RADD	1E-2 m2/s2
Dose	REM dose	REM	1E-2 m2/s2
Dose	MilliREM dose	mREM	1E-5 m2/s2
Electrical Resistance	Ohm	ohm	1 kg-m2/s3-amp2
Electrical Resistance	GigaOhm	Gohm	1E+9 kg-m2/s3-amp2
Electrical Resistance	KiloOhm	kohm	1E+3 kg-m2/s3-amp2
Electrical Resistance	MegaOhm	Mohm	1E+6 kg-m2/s3-amp2
Electrical Resistance	MicroOhm	uohm	1E-6 kg-m2/s3-amp2
Electrical Resistance	MilliOhm	mohm	1E-3 kg-m2/s3-amp2
Electrical Resistance	NanoOhm	nohm	1E-9 kg-m2/s3-amp2
Electrical Resistance	PicoOhm	pohm	1E-12 kg-m2/s3-amp2
Electrical Resistance	TeraOhm	Tohm	1E+12 kg-m2/s3-amp2
Energy	British Thermal Unit (Int’l)	BTU	1055.056 kg-m2/s2
Energy	Calorie	cal	4.1868 kg-m2/s2
Energy	GigaCalorie	Gcal	4.1868E+9 kg-m2/s2
Energy	KiloCalorie	kcal	4.1868E+3 kg-m2/s2
Energy	MegaCalorie	Mcal	4.1868E+6 kg-m2/s2
Energy	MicroCalorie	ucal	4.1868E-6 kg-m2/s2

Category	Unit	Abbreviation	Definition
Energy	MilliCalorie	mcal	4.1868E-3 kg-m2/s2
Energy	NanoCalorie	ncal	4.1868E-9 kg-m2/s2
Energy	PicoCalorie	pcal	4.1868E-12 kg-m2/s2
Energy	TeraCalorie	Tcal	4.1868E+12 kg-m2/s2
Energy	Joule	J	1 kg-m2/s2
Energy	GigaJoule	GJ	1E+9 kg-m2/s2
Energy	KiloJoule	kJ	1E+3 kg-m2/s2
Energy	MegaJoule	MJ	1E+6 kg-m2/s2
Energy	MicroJoule	uJ	1E-6 kg-m2/s2
Energy	MilliJoule	mJ	1E-3 kg-m2/s2
Energy	NanoJoule	nJ	1E-9 kg-m2/s2
Energy	PicoJoule	pJ	1E-12 kg-m2/s2
Energy	TeraJoule	TJ	1E+12 kg-m2/s2
Energy	Electron Volt	eV	1.60E-19 kg-m2/s2
Energy	GigaElectron Volt	GeV	1.60E-10 kg-m2/s2
Energy	KiloElectron Volt	keV	1.60E-16 kg-m2/s2
Energy	MegaElectron Volt	MeV	1.60E-13 kg-m2/s2
Energy	MicroElectron Volt	ueV	1.60E-25 kg-m2/s2
Energy	MilliElectronVvolt	meV	1.60E-22 kg-m2/s2
Energy	NanoElectron Volt	neV	1.60E-28 kg-m2/s2
Energy	PicoElectron Volt	peV	1.60E-31 kg-m2/s2
Energy	TeraElectron Volt	TeV	1.60E-07 kg-m2/s2
Energy	Kilowatt-hour	kwh	3600000 kg-m2/s2
Flux (Volume)	Acre-feet/day	afd	0.01427641 m3/s
Flux (Volume)	US Barrels/day	bpd	1.84E-06 m3/s
Flux (Volume)	Cubic feet per second	cfs	0.028316847 m3/s
Flux (Volume)	US Gallons per minute	gpm	6.31E-05 m3/s
Flux (Volume)	Million gallons per day	MGD	0.043812639 m3/s

Category	Unit	Abbreviation	Definition
Force	Newton	N	1 kg-m/s2
Force	GigaNewton	GN	1E+9 kg-m/s2
Force	KiloNewton	kN	1E+3 kg-m/s2
Force	MegaNewton	MN	1E+6 kg-m/s2
Force	MicroNewton	uN	1E-6 kg-m/s2
Force	MilliNewton	mN	1E-3 kg-m/s2
Force	NanoNewton	nN	1E-9 kg-m/s2
Force	PicoNewton	pN	1E-12 kg-m/s2
Force	TeraNewton	TN	1E+12 kg-m/s2
Force	Dyne	dyne	1E-05 kg-m/s2
Force	Gram Force	gf	9.80665E-3 kg-m/s2
Force	Kilogram Force	kgf	9.80665 kg-m/s2
Force	Milligram Force	mgf	9.80665E-6 kg-m/s2
Force	Pound Force	lbf	4.448221909 kg-m/s2
Force	1,000 Pound Force	kip	4.448221909E+3 kg-m/s2
Force	Ton Force	tonf	8.896443819E+3 kg-m/s2
Force	Ounce Force	ozf	0.278013869 kg-m/s2
Frequency (non-angular, Rate)	Hertz (frequency)	Hz	1 s-1
Frequency (non-angular, Rate)	GigaHertz (frequency)	GHz	1E+9s-1
Frequency (non-angular, Rate)	KiloHertz (frequency)	kHz	1E+3 s-1
Frequency (non-angular, Rate)	MegaHertz (frequency)	MHz	1E+6 s-1
Frequency (non-angular, Rate)	MicroHertz (frequency)	uHz	1E-6 s-1
Frequency (non-angular, Rate)	MilliHertz (frequency)	mHz	1E-3 s-1
Frequency (non-angular, Rate)	NanoHertz (frequency)	nHz	1E-9 s-1

Category	Unit	Abbreviation	Definition
Frequency (non-angular, Rate)	PicoHertz (frequency)	pHz	1E-12s-1
Frequency (non-angular, Rate)	TeraHertz (frequency)	THz	1E+12 s-1
Frequency (non-angular, Rate)	Becquerel	Bq	1 s-1
Frequency (non-angular, Rate)	GigaBecquerel	GBq	1E+9s-1
Frequency (non-angular, Rate)	KiloBecquerel	kBq	1E+3 s-1
Frequency (non-angular, Rate)	MegaBecquerel	MBq	1E+6 s-1
Frequency (non-angular, Rate)	MicroBecquerel	uBq	1E-6 s-1
Frequency (non-angular, Rate)	MilliBecquerel	mBq	1E-3 s-1
Frequency (non-angular, Rate)	NanoBecquerel	nBq	1E-9 s-1
Frequency (non-angular, Rate)	PicoBecquerel	pBq	1E-12s-1
Frequency (non-angular, Rate)	TeraBecquerel	TBq	1E+12 s-1
Frequency (non-angular, Rate)	Curie	Ci	3.7E+10 s-1
Frequency (non-angular, Rate)	MicroCurie	mCi	3.7E+4 s-1
Frequency (non-angular, Rate)	PicoCurie	pCi	3.7E-2 s-1
Illuminance	Lambert	lamb	10000 cd/m2
Illuminance	Lux (1 lm/m2)	lx	1 cd/m2
Inverse Area (mileage)	Miles per Gallon	mpg	425143.68321 m-2
Items	Items	item	dimensionless
Items	Persons	pers	1 item

Category	Unit	Abbreviation	Definition
Items	Thousand Persons	kpers	1E+3 item
Items	Million Persons	Mpers	1E+6 item
Length	Meter	m	1 m
Length	CentiMeter	cm	1E-2 m
Length	GigaMeter	Gm	1E+9 m
Length	KiloNewton	km	1E+3 m
Length	MegaMeter	Mm	1E+6 m
Length	MicroMeter	um	1E-6 m
Length	MilliMeter	mm	1E-3 m
Length	NanoMeter	nm	1E-9 m
Length	PicoMeter	pm	1E-12 m
Length	TeraMeter	Tm	1E+12 m
Length	Angstrom	Ang	1E-10 m
Length	Foot (US)	ft, ’	0.3048 m
Length	Inch	in, ”	0.0254 m
Length	Yard	yard	0.9144 m
Length	Mil=0.001 inch	mil	2.54E-5 m
Length	Mile	mi	1609.344 m
Length	Nautical Mile	naut	1852 m
Length	Rod	rd	5.0292 m
Length	Fathom	fath	1.8288 m
Length	Furlong	flng	201.168 m
Length	Light Year	ly	9.46E+15 m
Luminous Intensity	Candela	cd	1 cd
Luminous Intensity	GigaCandela	Gcd	1E+9 cd
Luminous Intensity	KiloCandela	kcd	1E+3 cd
Luminous Intensity	MegaCandela	Mcd	1E+6 cd
Luminous Intensity	MicroCandela	ucd	1E-6 cd

Category	Unit	Abbreviation	Definition
Luminous Intensity	MilliCandela	mcd	1E-3 cd
Luminous Intensity	NanoCandela	ncd	1E-9 cd
Luminous Intensity	PicoCandela	pcd	1E-12 cd
Luminous Intensity	TeraCandela	Tcd	1E+12 cd
Luminous Intensity	Lumen (cd/Steradian)	lm	0.079577472 cd
Mass	Gram	g	1E-3 kg
Mass	GigaGram	Gg	1E+6 kg
Mass	KiloGram	kg	1 kg
Mass	MegaGram	Mg	1E+3 kg
Mass	MicroGram	ug	1E-9 kg
Mass	MilliGram	mg	1E-6 kg
Mass	NanoGram	ng	1E-12 kg
Mass	PicoGram	pg	1E-15 kg
Mass	TeraGram	Tg	1E+9 kg
Mass	Tonne	tonne	1E+3 kg
Mass	Slug	slug	14.5939039 kg
Mass	Ounce (mass)	ozm	0.028349525 kg
Mass	Pound (mass)	lbm	0.4535924 kg
Mass	Ton (mass)	tonm	907.1848 kg
Math Constants	Parts per billion	ppb	1E-9
Math Constants	Parts per million	ppm	1E-6
Math Constants	Percentage	%	0.01
Permeability (seepage)	Darcy	Darcy	9.87E-13 m2
Permeability (seepage)	MilliDarcy	md	9.87E-16 m2
Power	Watt	W	1 kg-m2/s3
Power	GigaW	GW	1E+9 kg-m2/s3
Power	KiloW	kW	1E+3 kg-m2/s3

Category	Unit	Abbreviation	Definition
Power	MegaW	MW	1E+6 kg-m2/s3
Power	MicroW	uW	1E-6 kg-m2/s3
Power	MilliW	mW	1E-3 kg-m2/s3
Power	NanoW	nW	1E-9 kg-m2/s3
Power	PicoW	pW	1E-12 kg-m2/s3
Power	TeraW	TW	1E+12 kg-m2/s3
Power	Horsepower (550 ft-lb/sec)	hp	745.6999209 kg-m2/s3
Pressure, stress	Pascal	Pa	1 kg/m-s2
Pressure, stress	GigaMeter	GPa	1E+9 kg/m-s2
Pressure, stress	KiloNewton	kPa	1E+3 kg/m-s2
Pressure, stress	MegaMeter	MPa	1E+6 kg/m-s2
Pressure, stress	MicroMeter	uPa	1E-6 kg/m-s2
Pressure, stress	MilliMeter	mPa	1E-3 kg/m-s2
Pressure, stress	NanoMeter	nPa	1E-9 kg/m-s2
Pressure, stress	PicoMeter	pPa	1E-12 kg/m-s2
Pressure, stress	TeraMeter	TPa	1E+12 kg/m-s2
Pressure, stress	Bar	bar	1E+5 kg/m-s2
Pressure, stress	GigaBar	Gbar	1E+14 kg/m-s2
Pressure, stress	KiloBar	kbar	1E+8 kg/m-s2
Pressure, stress	MegaBar	Mbar	1E+11 kg/m-s2
Pressure, stress	MicroBar	ubar	1E-1 kg/m-s2
Pressure, stress	MilliBar	mbar	1E+2 kg/m-s2
Pressure, stress	NanoBar	nbar	1E-4 kg/m-s2
Pressure, stress	PicoBar	pbar	1E-7kg/m-s2
Pressure, stress	TeraBar	Tbar	1E+17 kg/m-s2
Pressure, stress	KiloPond	kp	98066.5 kg-m/s2
Pressure, stress	Pound per square foot	psf	47.88026215 kg/m-s2
Pressure, stress	Pound per square inch	psi	6894.757749 kg/m-s2

Category	Unit	Abbreviation	Definition
Pressure, stress	Torr (mm Hg)	torr	133.3221913 kg/m-s2
Pressure, stress	Atmosphere	atm	101325 kg/m-s2
Quantity of Matter	Mole	mol	1 mol
Quantity of Matter	GigaMole	Gmol	1E+9 mol
Quantity of Matter	KiloMole	kmol	1E+3 mol
Quantity of Matter	MegaMole	Mmol	1E+6 mol
Quantity of Matter	MicroMole	umol	1E-6 mol
Quantity of Matter	MilliMole	mmol	1E-3 mol
Quantity of Matter	NanoMole	nmol	1E-9 mol
Quantity of Matter	PicoMole	pmol	1E-12 mol
Quantity of Matter	TeraMole	Tmol	1E+12 mol
Temperature	Kelvin temperature	K	1 K
Temperature	GigaKelvin temperature	GK	1E+9 K
Temperature	KiloKelvin temperature	kK	1E+3 K
Temperature	MegaKelvin temperature	MK	1E+6 K
Temperature	MicroKelvin temperature	uK	1E-6 K
Temperature	MilliKelvin temperature	mK	1E-3 K
Temperature	NanoKelvin temperature	nK	1E-9 K
Temperature	PicoKelvin temperature	pK	1E-12 K
Temperature	TeraKelvin temperature	TK	1E+12 K
Temperature	Rankine temperature	R	0.555555556 K

Category	Unit	Abbreviation	Definition
Temperature	Celsius temperature	C	1 K
Temperature	Fahrenheit temperature	F	0.555555556 K
Temperature	Celsius degree	Cdeg	1 K
Temperature	Fahrenheit degree	Fdeg	0.555555556 K
Time	Second	s, sec	1 s
Time	GigaSecond	Gs	1E+9 s
Time	KiloSecond	ks	1E+3 s
Time	MegaSecond	Ms	1E+6 s
Time	MicroSecond	us	1E-6 s
Time	MilliSecond	ms	1E-3 s
Time	NanoSecond	ns	1E-9 s
Time	PicoSecond	ps	1E-12 s
Time	TeraSecond	Ts	1E+12 s
Time	Minute	min	60 s
Time	Hour	hr	3600 s
Time	Week	week	604800 s
Time	Day	d, day	86400 s
Time	Month	mon	2629800 s
Time	Year	yr, a	31557600 s
Time	Date	date	86400 s
Time	Date and time	datetime	86400 s
Velocity	Feet per minute	fpm	0.00508 m/s
Velocity	Feet per second	fps	0.3048 m/s
Velocity	Kilometer per hour	kph	0.277777778 m/s
Velocity	Knots	kt	0.514444444 m/s
Velocity	Miles per hour	mph	0.44704 m/s
Viscosity (Absolute)	Centipose	cp	0.001 kg/m/s

Category	Unit	Abbreviation	Definition
Viscosity (Absolute)	Poise	poise	0.1 kg/m/s
Viscosity (Kinematic)	Stoke	stoke	0.0001 m2/s
Voltage	Volt	V	1 kg-m2/s3-amp
Voltage	GigaVolt	GV	1E+9 kg-m2/s3-amp
Voltage	KiloVolt	kV	1E+3 kg-m2/s3-amp
Voltage	MegaVolt	MV	1E+6 kg-m2/s3-amp
Voltage	MicroVolt	uV	1E-6 kg-m2/s3-amp
Voltage	MilliVolt	mV	1E-3 kg-m2/s3-amp
Voltage	NanoVolt	nV	1E-9 kg-m2/s3-amp
Voltage	PicoVolt	pV	1E-12 kg-m2/s3-amp
Voltage	TeraVolt	TV	1E+12 kg-m2/s3-amp
Volume	Litre	L, l	1E-3 m3
Volume	GigaLitre	GL, Gl	1E+6 m3
Volume	KiloLitre	kL, kl	1 m3
Volume	MegaLitre	ML, Ml	1E+3 m3
Volume	MicroLitre	uL, ul	1E-9 m3
Volume	MilliLitre	mL, ml	1E-6 m3
Volume	NanoLitre	nL, nl	1E-12 m3
Volume	PicoLitre	pL, pl	1E-15 m3
Volume	TeraLitre	TL, Tl	1E+9 m3
Volume	Cubic Centimeter	cc	1E-6 m3
Volume	Gallon (US)	gal, galus	3.785412E-3 m3
Volume	Gallon (Imperial)	gali	4.54609E-3 m3
Volume	Quart (US)	qt	9.46353E-4 m3
Volume	Pint(US)	pint	4.73177E-4 m3
Volume	Cup (US)	cup	2.36588E-4 m3
Volume	Fluid ounce	floz	2.96E-5 m3
Volume	Tablespoon	tbsp	1.48E-5 m3

Category	Unit	Abbreviation	Definition
Volume	Teaspoon	tsp	4.93E-6 m3
Volume	Standard cubic foot	stcf	0.028316847 m3
Volume	Bushel	bushel	0.03523907 m3
Volume	Barrel (US, oil)	bbl	0.1589873 m3
Volume	Barrel (US, dry)	bbldry	0.11563 m3
Volume	Barrel (US, liquid)	bblliq	0.11924 m3
Volume	Acre-feet	af	1.2348183754752E+3 m3
Volume	Thousand Acre-feet	kf	1.2348183754752E+6 m3
Volume	Million Acre-feet	Maf	1.2348183754752E+9 m3

Appendix E: Database Input File Formats

Art and science cannot exist but in minutely organized particulars.

William Blake, *To the Public*

Appendix Overview

In simulations which require a great deal of input, it may be desirable for the simulation model to access the various data sources directly to ensure the quality of the data transfer.

To facilitate this, GoldSim data entry elements can be linked directly to an ODBC-compliant database.

Read More: *Linking Elements to a Database* on page 1094

After defining the linkage, you can then instruct GoldSim to download the data at any time. When it does this, GoldSim internally records the time and date at which the download occurred, along with other reference information retrieved from the database (e.g., document references), and this is stored with the model in the Run Log. This allows you to confirm that the correct data were loaded into your model, and provides very strong and defensible quality control over your model input data.

GoldSim can import from three different types of databases: a Generic Database, a Simple GoldSim Database, and an Extended GoldSim Database. This appendix describes the details of the structure and format for each of these three database types.

In this Appendix

This appendix discusses the following:

- Creating a Generic Database
- Creating a Simple GoldSim Database
- Creating an Extended GoldSim Database

Creating a Generic Database

The generic database format requirements are very general:

- The database must be ODBC compliant;
- The selected table must have a field (column) which contains unique IDs which will be used to map data to specific GoldSim elements. These IDs would typically be the GoldSim element names (but they do not have to be).
- The table must have one or more columns containing the data items to be downloaded.

Note that by assigning multiple records and using multiple fields for each ID, you can download vector and matrix data from the generic database.

Read More: *Downloading from a Generic Database* on page 1097

The file GenericDatabase.gsm in the General Examples/Database folder of your GoldSim directory (accessed by selecting **File | Open Example...** from the main menu) provides an example of how to use a Generic database. This folder also includes the sample database referenced by this file (GenericDatabase.accdb), created using Microsoft Access. In order to use the GoldSim file, you will need to add the database as data sources to your computer.

Read More: *Adding Data Sources to Your Computer* on page 1095

Creating a Simple GoldSim Database

A Simple GoldSim database contains the following three tables:

- tbl_Parameter
- tbl_Parameter_Reference
- tbl_Probability_Value_Pairs

Each of these tables is discussed in detail below.

Simple Database Parameter Table

The Parameter Table (tbl_Parameter) must contain the following fields (which must be named as shown below):

Field Name	Type	Description
UID	AutoNumber	Unique integer number assigned to each element.
Parameter_ Name	Text	The ID of the GoldSim element. Note that the length of an element ID in GoldSim is limited to 30 characters.
Parameter_ Path	Memo	The full path to the element in GoldSim. This attribute is optional. GoldSim tries to find a record set in the database using the name and path information. If this query is not successful it will try again just querying the name. If a path is specified it must end with '\'.

Field Name	Type	Description
Type_Code	Text	Code to describe the parameter type (see below).
ModDate	Date/Time	Last Date that the parameter properties were changed. Note that this field is for information purposes only and is not used by GoldSim.
Current	Yes/No	This version of the parameter (this record) is considered active in GoldSim if this flag is set to Yes. Note that only one parameter with the same name (and path) should have the current flag set to true.
Description	Text	Description of the parameter, inserted into the Description field for the element.
Unit	Text	Unit abbreviation for the parameter. See Appendix D for unit abbreviations. The unit should be specified without parentheses or brackets.
Arg_1	Number (double)	Value for the first argument for the parameter. All parameters have at least one argument.
Arg_2	Number (double)	Value for the second argument for the parameter
Arg_3	Number (double)	Value for the third argument for the parameter
Arg_4	Number (double)	Value for the fourth argument for the parameter

Note: Particular care must be taken when importing Stochastics from a database, as different Stochastics could have different dimensions (e.g., a Binomial is always dimensionless; a Boolean is always a condition). If a Stochastic is defined in your model as a particular type of distribution whose output is inconsistent with that of the distribution that you are trying to import from the database, GoldSim will display an error.

Note: Care must also be taken when importing Stochastics that represent temperatures. This is because temperatures have an absolute reference (i.e., absolute zero). Differences between temperatures are expressed in ‘deg’ units (Cdeg, Fdeg). This can lead to errors when importing Stochastics, since GoldSim assumes a single unit for the distribution parameters, while some types of distributions would require two different types of units. For example, a Normal distribution representing a temperature would require the Mean to be specified in absolute units (e.g., C) and the Standard Deviation to be specified in difference units (e.g., Cdeg). If faced with this problem, there are several approaches for addressing this: 1) Use a distribution type where all parameters have the same units (e.g., triangular, uniform, cumulative); or 2) Specify and import the distributions as dimensionless and then apply a unit to the sampled value using an Expression element.

Read More: *Dealing with Temperature Units* on page 92

Warning: The special GoldSim units “date” and “datetime” cannot be used when importing from a database.

Note that the Parameter_Path does not need to be defined (it can be blank). In this case, it does not matter where the element exists in the model. If you specify a path, it must start and end with a backslash (e.g., \container1\container2\ ). If you want to specify the top-level Container (the model root), you should use a single backslash.

If you specify a path, GoldSim first tries to find a record with the specified element name and path. If it does not find it, it tries again, ignoring the path.

If GoldSim finds multiple records with the same name and path, and both have the Current field marked "Yes", it will issue an error message (i.e., if multiple records in the database have the same name and path, only one record can have the Current flag set to Yes).

Simple Database Parameter Type Codes and Arguments

The codes for the various parameter types, and their required arguments are shown below:

Distribution	Type_ Code	Arg_1	Arg_2	Arg_3	Arg_4
constant (Data element)	100	Value			
constant (vector Data element)	101	Number of rows	1 (Number of columns)		
constant (matrix Data element)	102	Number of rows	Number of columns		
uniform	2100	Minimum	Maximum		
log-uniform	2101	Minimum	Maximum		
normal	2200	Mean	Std. Deviation		
truncated normal	2202	Mean	Std. Deviation	Minimum	Maximum
log-normal (geometic input)	2300	Geometric Mean	Geometric Std. Deviation		
truncated log-normal (geometric input)	2302	Geometric Mean	Geometric Std. Deviation	Minimum	Maximum
log-normal (true mean input)	2330	True Mean	True Std. Deviation		
truncated log-normal (true mean input)	2332	True Mean	True Std. Deviation	Minimum	Maximum
triangular	2400	Minimum	Most Likely	Maximum	

Distribution	Type_ Code	Arg_1	Arg_2	Arg_3	Arg_4
log-triangular	2401	Minimum	Most Likely	Maximum	
triangular (10/90)	2402	10th percentile	Most Likely	90th percentile	
log-triangular (10/90)	2403	10th percentile	Most Likely	90th percentile	
cumulative	2500	references Probability_ Value_Pairs table (see below)			
log-cumulative	2501	references Probability_ Value_Pairs table (see below)			
discrete	2600	references Probability_ Value_Pairs table (see below)			
poisson	2700	Expected Value			
beta (generalized)	2800	Mean	Std. Deviation	Minimum	Maximum
beta (success, failure)	2804	Successes	Failures		
BetaPERT	4200	Minimum	Most Likely	Maximum	
BetarPERT 10/90	4201	10th percentile	Most Likely	90th percentile	
gamma	2900	Mean	Std. Deviation		
truncated gamma	2902	Mean	Std. Deviation	Minimum	Maximum
weibull	3000	Minimum	Weibull Slope	Mean-Minimum	
truncated weibull	3002	Minimum	Weibull Slope	Mean-Minimum	Maximum
binomial	3100	# of Picks (Batch size)	Probability of Success		
boolean	3200	Probability of True			

Distribution	Type_ Code	Arg_1	Arg_2	Arg_3	Arg_4
Student's t	3300	Degrees of freedom			
exponential	3400	Mean			
pareto	3500	a	b		
truncated Pareto	3502	a	b	Maximum	
negative binomial	3600	Successes	Probability of Success		
extreme value (minimum)	3800	Location	Scale		
extreme value (maximum)	3803	Location	Scale		
extreme probability (minimum)	3900	Number of Samples			
extreme probability (maximum)	3903	Number of Samples			
Pearson type III	4000	Location	Scale	Shape	
sampled results (no extrapolation)	4100	references Probability_ Value_Pairs table (see below)			
sampled results (extrapolation)	4103	references Probability_ Value_Pairs table (see below)			

Note that for the Discrete, Cumulative and Sampled Results distributions, the first argument references the val_ pair_UID field in the Probability Value Pairs table (described below).

Simple Database Parameter Reference Table

The Parameter Reference Table (tbl_Parameter_Reference) allows you to specify reference information for the element.

The table has the following fields:

Field Name	Type	Description
Parameter_ UID	Text	Primary Key – link from UID in Parameter Table
GS_ Parameter_ Note	Text	The text in this field overwrites the Note associated with the element in GoldSim. If left blank here, the Note in GoldSim is not overwritten.

- When GoldSim imports text from a database into a GoldSim Note, it automatically converts text to hyperlinks in the Note under the following circumstances:
- Any text beginning the prefixes listed below is converted to a hyperlink. The hyperlink terminates when a space in encountered. As a result, hyperlinks with spaces will not be recognized by GoldSim.
    - http://
    - www.
    - ftp://
    - ftp.
    - file://
- If the character @ is encountered, and text before and after the @ not delimited by a space or a line break will be considered part of the hyperlink.

Simple Database Array Values Table

The Array Values Table (tbl_Array_Values) is used to store an arbitrary number of array values for a vector or matrix Data element.

The table has the following fields:

Field Name	Typed	Description
ID	Number	Unique integer number assigned to each row in the table.
Array_UID	Number	The parameter ID from the Parameter Table (tbl_Parameter) identifying the Data element.
Value	Number	The value for the specified row and column of the array.
Row	Number	The row of the array. This index is 1-based.
Column	Number	The column of the array. This index is 1-based. It must be set to 1 for vectors.

Simple Database Probability Value Pairs Table

The Probability Value Pair Table (tbl_Probability_Value_Pairs) is used to store an arbitrary number of value pairs used in defining discrete, cumulative and sampled results distributions. No other type of element uses this table.

The table has the following fields:

Field Name	Type	Description
Val_pair_ UID	Number	Identifies a set of value-probability-pairs that are used by discrete, cumulative and sampled results distributions. This distribution must specify this ID in arg_ 1 in tbl_Parameter.

Field Name	Type	Description
Probability	Number	The probability (for discrete distributions) or cumulative probability (for Cumulative distributions) of the data pair (dimensionless). Ignored for sampled results distributions.
Value	Number	The value corresponding with the defined probability level for Discrete, Cumulative and Sampled Results distributions. Uses the unit defined in the parameter table.

Simple Database Example File and Database Template

The file SimpleDatabase.gsm in the General Examples/Database folder of your GoldSim directory (accessed by selecting File | Open Example... from the main menu) provides an example of how to use a Simple GoldSim database. This folder also includes the sample database referenced by this file (SimpleDatabase.accdb), created using Microsoft Access. In order to use the GoldSim file, you will need to add the database as data sources to your computer.

Read More: *Adding Data Sources to Your Computer* on page 1095

The Database subfolder also includes a template for creating Simple GoldSim databases (SimpleDatabaseTemplate.accdb). This template file includes two additional tables (providing parameter type codes and unit abbreviations) which can be used to support the implementation of custom forms which allow the user to pick a parameter type code and unit from a list.

Creating an Extended GoldSim Database

An Extended GoldSim database must contain the following three tables:

- GS_Parameter
- GS_Parameter_Value
- GS_Value_Component

Each table is described in detail below.

Note: The tables described below can contain additional fields not used by GoldSim. When importing information, GoldSim will ignore any extra fields.

Extended Database Parameter Table

The Parameter Table (GS_Parameter) has one record for each linked Element. It contains basic descriptive information about the Element, and an index (Parameter_ID) which links it into the GS_Parameter_Value table. It must contain the following fields:

Number	Field	Description
1	Parameter_ ID	Unique integer number assigned to each element
2	Parameter_ Name	Key field which must match the GoldSim element ID
3	Description	Text description of the element
4	Units	String with GoldSim abbreviations for units of the data (see Appendix D for correct unit abbreviations)
5	Parameter_ Code	Code for parameter type (see table below)
6	indep_row_ units	String with GoldSim abbreviations for units of a table element's Row independent variable (only required for tables)
7	indep_col_ units	String with GoldSim abbreviations for units of the 2-D table element's Column independent variable (only required for 2-D tables)
8	bc_date	The default effective date for this item, in format YYYY-MM-DD HH:MM:SS. If no effective date is specified when downloading data, this date is used for this particular element.

The Parameter_Code for item is as follows:

Parameter_Code	Element Type
100	Data element
1nn	Array Data element, where nn is the # of columns (01 for vectors)
2100	Stochastic: Uniform
2101	Stochastic: Log-Uniform
2200	Stochastic: Normal
2202	Stochastic: Truncated Normal
2300	Stochastic: Log-Normal (geometric mean)
2302	Stochastic: Truncated Log-Normal (geometric mean)
2330	Stochastic: Log-Normal (true mean)
2332	Stochastic: Truncated Log-Normal (true mean)
2400	Stochastic: Triangular
2401	Stochastic: Log-Triangular
2402	Stochastic: Triangular (10/90)

Parameter_Code	Element Type
2403	Stochastic: Log-Triangular (10/90)
2500	Stochastic: Cumulative
2501	Stochastic: Log-cumulative
2600	Stochastic: Discrete
2700	Stochastic: Poisson
2800	Stochastic: Generalized Beta
2804	Stochastic: Beta (Success, Failure)
2900	Stochastic: Gamma
2902	Stochastic: Truncated Gamma
3000	Stochastic: Weibull
3002	Stochastic: Truncated Weibull
3100	Stochastic: Binomial
3200	Stochastic: Boolean
3300	Stochastic: Student’s t
3400	Stochastic: Exponential
3502	Stochastic: Truncated Pareto
3600	Stochastic: Negative Binomial
3800	Stochastic: Extreme Value (minimum)
3803	Stochastic: Extreme Value (maximum)
3900	Stochastic: Extreme Probability (minimum)
3903	Stochastic: Extreme Probability (maximum)
4000	Stochastic: Pearson Type III
4100	Stochastic: Sampled Results
4103	Stochastic: Sampled Results (extrapolation)
4200	Stochastic: BetaPERT
4201	Stochastic: BetaPERT (10/90)
5100	1-D Table
52nn	2-D Table, where nn is the no. of columns
	File element (no code required)

Note: Particular care must be taken when importing Stochastics from a database, as different Stochastics could have different dimensions (e.g., a Binomial is always dimensionless; a Boolean is always a condition). If a Stochastic is defined in your model as a paticular type of distribution whose output is inconsistent with that of the distribution that you are trying to import from the database, GoldSim will display an error.

Note: Care must also be taken when importing Stochastics that represent temperatures. This is because temperatures have an absolute reference (i.e., absolute zero). Differences between temperatures are expressed in 'deg' units (Cdeg, Fdeg). This can lead to errors when importing Stochastics, since GoldSim assumes a single unit for the distribution parameters, while some types of distributions would require two different types of units. For example, a Normal distribution representing a temperature would require the Mean to be specifed in absolute units (e.g., C) and the Standard Deviation to be specified in difference units (e.g., Cdeg). If faced with this problem, there are several approaches for addressing this: 1) Use a distribution type where all parameters have the same units (e.g., triangular, uniform, cumulative); or 2) Specify and import the distributions as dimensionless and then apply a unit to the sampled value using an Expression element.

Read More: *Dealing with Temperature Units* on page 92

Warning: The special GoldSim units "date" and "datetime" cannot be used when importing from a database.

Extended Database Parameter Value Table

The Parameter Value Table (GS_Parameter_Value) has one record for each Element and each effective date. Each record must have a unique Effective_Date and Value_ID. The Value_ID is an index which is used to link into the actual data values, which are stored in table GS_Value_Component. The GS_Parameter_Value table must contain the following fields:

Number	Field	Description
1	Parameter_ID	Key to link from items in table GS_Parameter
2	Value_ID	Unique integer key for value record(s) in table GS_Value_Component
3	Effective_ Date	The effective date for this item, in format YYYY-MM-DD HH:MM:SS
4	Reference_ Document	Written by GoldSim to the Run Log (an optional string)
5	Document_ ID	Written by GoldSim to the Run Log (an optional string)
6	DTN	Written by GoldSim to the Run Log (an optional string)
7	MOL	Written by GoldSim to the Run Log (an optional string)
8	DOC_Path	Network path for the source document (required only for File elements, as discussed below)

Number	Field	Description
9	DOC_SIG	CRC signature for File source document (a string required only for File elements)
10	Parameter_ Note	Text that is imported into the element's Note. This must be a "Memo" field, and does not support rich text or HTML. Only plain text can be imported.

The CRC signature is an alphanumeric code that can be used to uniquely identify whether the file contents have changed. When you download a file, GoldSim compares the CRC signature of the downloaded file with the original signature that was stored in the database. If these are not identical (indicating that the file has been changed), the download will fail.

You can generate a CRC signature for a file using the EFIViewer, a small utility program that is installed with GoldSim.

Extended Database Value Component Table

The Value Component Table (GS_Value_Component) stores the actual data values. There are one or more records for each data value. Each record must contain the following fields:

Number	Field	Description
1	Value_ID	The index used to link from the GS_Parameter_Value table
2	Component_ID	Unique index
3	Type_Code	Row number for sorting rows in vectors and matrices (used only arrays).
4-63	Value_Column_1, …, Value_ Column_60	Data values, to support tables and matrices with up to 60 columns.

For a Data element, the data value is stored in Value_Column_1.

For vector Data elements, the data for each item is stored in Value_Column_1, and there should be one record for each row (item) in the vector. For matrix Data elements, the data for each row of the matrix is stored in Value_ Column_1 through Value_Column_nn (where nn is as specified in the Parameter_Code field of the Parameter Table), and there should be one record for each row of the matrix.

Note: For matrix Data elements, data values for each matrix row must be defined in fields 4 to 4+N, in consecutive order (N being the number of columns as specified in the Parameter_Code field of the Parameter table). The DB fields must be labeled Value_Column_1 to Value_Column_N. Other field labels are not supported by GoldSim.

For a 1-D Table element, the independent variable values are stored in Value_Column_1, and the dependent variable values are stored in Value_Column_2. There is one record for each row in the table.

For a 2-D Table element, the row independent variable values are stored in Value_Column_1, and the dependent variable values are stored in Value_Column_2 through Value_Column_n, where n is one greater than the number of columns in the table. The first record for a 2-D Table element contains values for the column independent variable in Value_Column_2 through Value_Column_n, and the successive records contain values for the Row independent variable followed by values for the dependent variable.

Note: The row independent variable values must be defined in field 4, labeled Value_Column_1. Other field labels are not supported by GoldSim. The dependent variable values must in field 5 through field N+4, and must be labeled labeled Value_Column_2 Value_Column_N. Other field labels are not supported by GoldSim.

Note: Matrices and 2-D tables can have no more than 60 columns.

For Stochastic elements other than discrete, cumulative and sampled results, the arguments are stored, in sequence, in the Value_Column_1 … Value_Column_n entries. The order of the arguments for each type of Stochastic is listed below:

Stochastic	Order of Arguments
Uniform	minimum, maximum
Log-Uniform	minimum, maximum
Normal	mean, standard deviation
Truncated Normal	mean, standard deviation, minimum, maximum
Log-Normal (geometric)	geometric mean, geometric standard deviation
Log-Normal (true mean)	true mean, true standard deviation
Truncated Log-Normal (true mean)	true mean, true standard deviation, minimum, maximum
Triangular	minimum (or 10%), most likely, maximum (or 90%)
Log-Triangular	minimum (or 10%), most likely, maximum (or 90%)
Poisson	expected value
Beta	successes, failures
Generalized Beta	mean, standard deviation, minimum, maximum
BetaPERT	minimum (or 10%), most likely, maximum (or 90%)
Gamma	mean, standard deviation
Truncated Gamma	mean, standard deviation, minimum, maximum
Weibull	minimum, slope, mean-mimimum
Truncated Weibull	minimum, slope, mean-minimum, maximum
Binomial	# picks, probability of success
Boolean	probability of true

Stochastic	Order of Arguments
Student’s t	degrees of freedom
Exponential	mean
Pareto	a, b
Truncated Pareto	a, b, maximum
Negative Binomial	successes, probability of success
Extreme Value (minimum)	location, scale
Extreme Value (maximum)	location, scale
Extreme Probability (minimum)	number of samples
Extreme Probability (maximum)	number of samples
Pearson Type III	location, scale, shape

For a discrete, cumulative or sampled results Stochastic element type, there are multiple rows in the table, with one row for each value. Value_Column_1 contains the probability values, and Value_Column_2 contains the result values. For a sampled results distribution, there is only one value (the result) and this is placed in Value_Column_2 (Value_Column_1 is ignored).

For a Sampled Results distribution, there are multiple rows in the table, with one row for each value. Value_ Column_2 contains the result values. Any probabilities entered in Value_Column_1 are ignored (all results have equal weights).

Extended Database Example File

The file ExtendedDatabase.gsm in the General Examples/Database folder of your GoldSim directory (accessed by selecting **File | Open Example...** from the main menu) provides an example of how to use an Extended GoldSim database. This folder also includes the sample database referenced by this file (ExtendedDatabase.accdb), created using Microsoft Access. In order to use the GoldSim file, you will need to add the database as data sources to your computer.

Read More: *Adding Data Sources to Your Computer* on page 1095

Appendix F: Integration Methods and Timestepping Algorithm

On two occasions I have been asked [by members of Parliament], 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Charles Babbage

Appendix Overview

The elements and links in a GoldSim model represent a system of equations. Except in the simplest cases, these are systems of differential equations, and they are often nonlinear and discontinuous. In general, the systems of equations that GoldSim must solve will not have an analytical solution (i.e., they cannot be solved exactly), and must be solved numerically (i.e., using an algorithm that provides a numerical approximation to the actual solution).

In order to effectively use GoldSim, particularly for complex problems, it is important to have a basic understanding of the factors affecting the accuracy of your model, and the nature of the numerical approximations used by GoldSim.

This appendix provides a brief discussion of these numerical algorithms.

In this Appendix

This appendix discusses the following:

- Factors Affecting the Accuracy of Simulation Models
- Primary Numerical Approximations in GoldSim
- Summary of GoldSim's Dynamic Timestepping Algorithm

Factors Affecting the Accuracy of Simulation Models

A simulation model is an abstract representation of an actual (or hypothetical) system. By definition, it is a simplification of reality, with the goals being to include those aspects that are assumed to be important and omit those which are considered to be nonessential, so as to derive useful predictions of system performance in an efficient manner.

In addition, for most real world systems, there will be significant uncertainty regarding the processes that are controlling the system and the parameter values describing those processes. As a result, the most important factor impacting the accuracy of your model is the degree to which your conceptual and mathematical model have captured reality and the degree to which you have quantitatively represented your uncertainty in the system. In most real world systems that you would simulate in GoldSim, the uncertainty in the simulated result due to your uncertainty in the processes and parameters controlling the system will be far greater than any inaccuracies introduced by the numerical solution method. As a result, it will generally be more worthwhile for you to spend your time ensuring that your model captures the key aspects of the system realistically rather that worrying about the numerical accuracy of the solution.

Having said that, it is still important to understand the nature of the inaccuracies that can arise from GoldSim's numerical approximations in order to ensure that these do indeed remain small. The primary numerical factors affecting the accuracy of a GoldSim model are as follows:

- **Integrating Differential Equations:** GoldSim solves differential equations by numerically integrating them (via Stocks and Delays). This numerical integration is the largest potential source of numerical inaccuracies in a GoldSim model.
- **Solving Coupled Equations:** In some cases, the system you wish to model may include coupled equations or coupled differential equations. Solutions of these types of equations can be computationally-intensive (e.g., nonlinear coupled differential equations). For some specific types of coupled systems, GoldSim provides fast and accurate solution techniques. In other cases, these equations must be solved approximately using Previous Value elements. The use of Previous Value elements in this case can introduce numerical approximations that are of the same order as those introduced via numerical integration of ordinary differential equations.

Read More: *Using Advanced Algorithms to Solve Coupled Equations* on page 1208

- **Representing Discrete Events:** In many real world systems, discrete events occur which impose discontinuous changes onto the system. Superimposing such discontinuities onto a continuously-varying system discretized in time can introduce inaccuracies. GoldSim provides a powerful timestepping algorithm for accurately representing such systems.

These items are discussed in the sections below.

Note: Round-off error is sometimes noted as an important source of error in simulation models. Although this can indeed be important for some specialized engineering and science simulation tools, given the nature of GoldSim applications, and the fact that GoldSim uses double-precision to carry out its calculations, it is highly unlikely that round-off error could ever have a noticeable impact on a GoldSim model.

Primary Numerical Approximations in GoldSim

The primary numerical approximations in GoldSim are summarized below.

GoldSim Numerical Integration Algorithm

Stocks (Integrators, Reservoirs and Pools) represent time integrals of the form:

$$\mathbf{Value} = \mathbf{Intitial\ Value} + \int (\mathbf{Rate\ of\ Change})\ \mathbf{dt}$$

The Rate of Change, of course, can be a function of time.

In this case, we are solving the following differential equation:

$$\frac{\mathbf{dValue}}{\mathbf{dt}} = \mathbf{Rate\ of\ Change} \qquad \mathbf{Value_{t=0}} = \mathbf{Intial\ Value}$$

Numerically, GoldSim approximates the integral shown above as a sum:

$$\mathbf{Value(t_n)} = \mathbf{Intial\ Value} + \sum_{i=1}^{n} \mathbf{Rate\ of\ Change(t_i - \Delta t_i)\ \Delta t_i}$$

where

Δt_i is the timestep length just prior to time t_i (typically this will be constant in the simulation);
Rate of Change(t_i - Δt_i) is the Rate of Change at time = t_i - Δt_i; and
Value(t_i) is the value at end of timestep i.

Note that the Value at a given time is a function of the Rate of Change at previous timesteps (but is not a function of the Rate of Change at the current time).

This particular integration method is referred to as ***Euler integration***. It is the simplest and most common method for numerically solving such integrals. The key assumption in the method is that the rate remains constant over a timestep. The validity of this assumption is a function of the length of the timestep and the timescale over which the rate is changing. The assumption is reasonable if the timestep is sufficiently small.

To get a feeling for the errors that can be produced by Euler integration, consider the following very simple integral:

$$\mathbf{Value} = \mathbf{Initial\ Value} + \int -\mathbf{k} * \mathbf{Value\ dt}$$

where k is a constant. This is the equation for simple (exponential) first-order decay, and the analytical solution is:

$$\mathbf{Value} = \mathbf{Initial\ Value} * \mathbf{e^{-kt}}$$

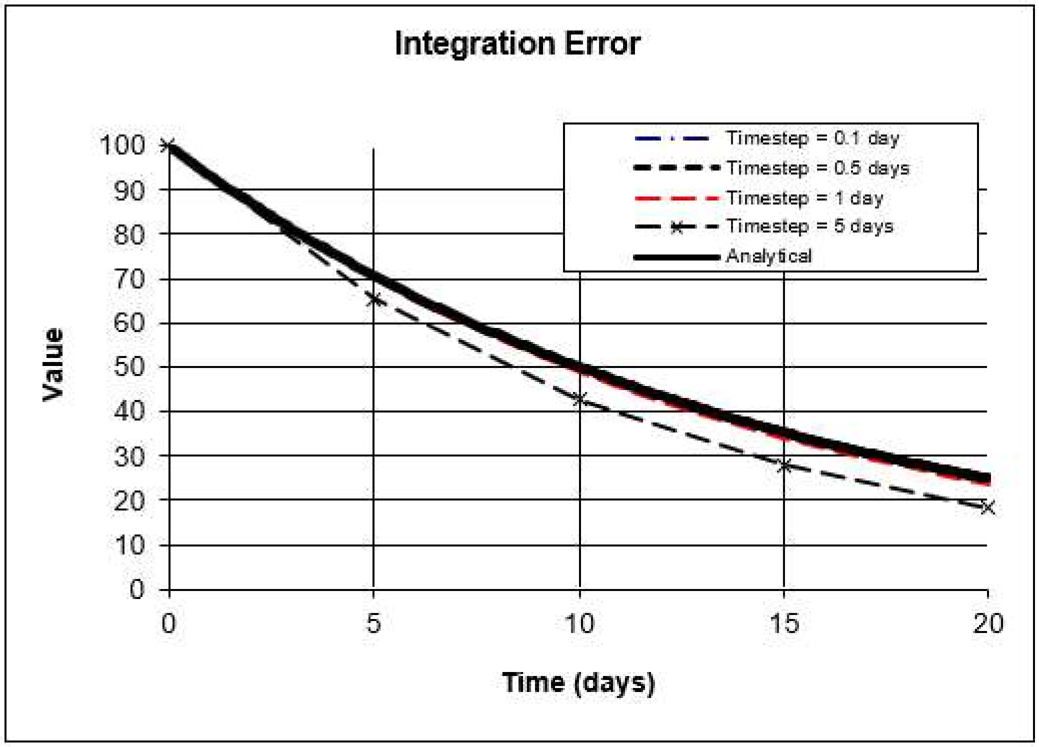
The timescale of the dynamic process can be quantified in terms of a ***half-life*** (the time it takes the current value to decay to half of its initial value). The half-life is equal to 0.693/k. The table below compares the results of computing the Value analytically and numerically (by comparing the results at 20 days assuming an initial value of 100 and a half-life of 10 days):

Solution	Value at 20 days	% Error*
Analytical Solution	25.00	-
Timestep = 5 days	18.24	9.0%

Solution	Value at 20 days	% Error*
Timestep = 1 days	23.78	1.6%
Timestep = 0.5 days	24.40	0.8%
Timestep = 0.1 days	24.89	0.1%

** Error computed as (Simulated-Analytical)/(Analytical-Initial Value)*

A plot of these results is shown below:



As can be seen, with the exception of the 5 day timestep, the Euler integration method is relatively accurate. In fact, for most systems that you will be simulating using GoldSim, a numerical error of several percent is likely to be acceptable (and much smaller than the error caused by the uncertainty in the initial conditions, the parameters, and the conceptual model). As a general rule, your timestep should be 1/3 to 1/10 of the timescale of the fastest process being simulated in your model.

Read More: *Selecting the Proper Timestep* on page 1206

Warning: The magnitude of the integration error depends on the nature of the model. In stable models that are dominated by negative feedback loops (and hence tend toward equilibrium), the errors tend to diminish with time. Systems that are unstable and grow exponentially or oscillate with no damping tend to accumulate errors over time. For these types of systems (e.g., a swinging pendulum), a very small timestep may be required to accurately simulate the system using Euler integration.

Note: In cases where a small timestep is required to maintain accuracy, this can be done in a very computational efficient way by using Containers with Internal Clocks, which allow you to locally use a much smaller timestep.

Read More: *Specifying Containers with Internal Clocks* on page 499

Other Integration Methods

Although the Euler method is simple and commonly used, other integration methods exist which can achieve higher accuracy with a larger timestep (e.g., Runga-Kutta, variable timestep methods). Because these methods can use a much larger timestep without losing accuracy, they are more computationally efficient.

These methods, while being important for some types of systems (e.g., sustained oscillators like a pendulum), are not incorporated into GoldSim for the following reasons:

- Higher order methods are most useful when simulating physical systems in which the mathematical model, initial conditions and input parameters are known very precisely, and small integration errors can be significant. For the most part, the kinds of systems that you will be simulating using GoldSim can be handled effectively using Euler integration.
- Higher-order methods work best when simulating continuously-varying systems. They do not deal well with systems that behave discontinuously and/or are impacted by discrete changes. Most real world systems do not vary continuously, and GoldSim therefore provides powerful algorithms for accurately superimposing discrete changes (discontinuities) onto a continuously-varying system. These algorithms are incompatible with higher-order integration methods.

Read More: *Accurately Simulating Discrete Events that Occur Between Timesteps* on page 1207

- For some kinds of systems (e.g., mass or heat transport using the Contaminant Transport Module), GoldSim uses powerful algorithms to accurately solve nonlinear coupled differential equations. These algorithms are incompatible with higher-order integration methods.

Read More: *Using Advanced Algorithms to Solve Coupled Equations* on page 1208

As mentioned above, in cases where a small timestep is required to maintain accuracy using Euler integration, this can be done in a very computational efficient way by using Containers with Internal Clocks, which allow you to locally use a much smaller timestep.

Read More: *Specifying Containers with Internal Clocks* on page 499

Approximate Solutions to Coupled Equations

In some situations, you may wish to simulate a static or dynamic processes in which the variables in the system are coupled such that they respond instantaneously to each other, with no time lags. In GoldSim, these are referred to as recursive loops, and they are treated differently from feedback loops.

If you encounter a system such as this in one of your models, you can handle it in one of two ways. First, you could solve the system of equations directly either prior to running the model (e.g., using substitution), or dynamically while running the model (e.g., using an External element or GoldSim's matrix functions to solve the appropriate equations). Note, however, that for many complex models (e.g., non-linear coupled equations), the solution could be very computationally intensive.

GoldSim offers an alternative: solve the equations approximately and iteratively using ***Previous Value elements***. A Previous Value element allows you to reference the previous value (i.e., the previous timestep) of an output.

Read More: *Creating Recursive Loops Using Previous Value Elements* on page 1023

Selecting the Proper Timestep

As a general rule, your timestep should be 3 to 10 times shorter than the timescale of the fastest process being simulated in your model. In simple systems, you should be able to determine the timescales of the processes involved. In more complex models, however, it may be difficult to determine the timescales of all the processes involved.

In addition, for some kinds of models (e.g., some oscillating systems), this general rule may not be sufficient and you may need a smaller timestep).

Therefore, after building your model, you should carry out the following experiment:

1. Carry out an expected value or median value simulation.
2. Reduce the timestep length by half (increase the number of timesteps by a factor of 2), rerun the model, and compare results.
3. Continue to half the timestep length until the differences between successive simulations are acceptable.

Summary of GoldSim's Dynamic Timestepping Algorithm

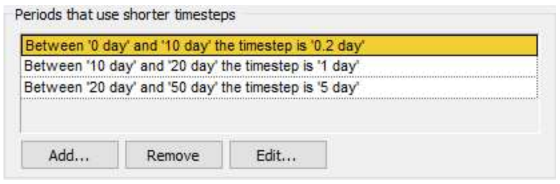
GoldSim provides a powerful timestepping algorithm that can dynamically adjust to more accurately represent discrete events, respond to rapidly changing variables in your model, represent specified SubSystems in your model using different timesteps, and accurately represent some special kinds of systems (e.g., mass and heat transport within the Contaminant Transport Module).

These features are discussed in detail elsewhere in this document. To complement the rest of the information provided in this appendix, however, these features are summarized here.

Defining Specific Periods with Shorter Timesteps

GoldSim allows you to increase or decrease the timestep length according to a specified schedule during a simulation (e.g., start with a small timestep, and then telescope out to a larger timestep). This can be useful, for example, if you know that early in a simulation, parameters are changing rapidly, and hence you need a smaller timestep.

You do this by defining Periods Steps, which have different durations and timestep lengths. An example of pre-specified time periods is shown below:


Periods that use shorter timesteps

Between '0 day' and '10 day' the timestep is '0.2 day'
Between '10 day' and '20 day' the timestep is '1 day'
Between '20 day' and '50 day' the timestep is '5 day'

In this example, the model uses a 0.2day timestep for the first 10 days, a 1 day timestep between 10 days and 20 days, and a 5 day timestep between 20 days and 50 days (Although not indicated here, it then reverts to the default timestep of 10 days for the remainder of the simulation).

Read More: *Adding Shorter Timesteps Over Defined Periods* on page 492

Dynamically Adjusting the Timestep

Although defining shorter timesteps over defined periods can be very useful, you must fully specify them prior to running the simulation. That is, you must know how you would like to alter your timestep prior to running the model. In some cases, however, it may not be possible to do this. That is, in complex systems (particularly ones with uncertain parameters), variables may change at different rates in different realizations, in ways that you cannot predict prior to running the model.

To better simulate these kinds of systems, GoldSim provides an advanced feature that allows you to dynamically adjust the timestep during a simulation (i.e., insert "internal" timesteps) based on the values of specified parameters in your model. For example, you could instruct GoldSim to use a timestep of 1 day if X was greater than Y, and 10 days if X was less than or equal to Y. Similarly, you could instruct GoldSim to use a short timestep for a period of 10 days after a particular event occurs, and then return to the default timestep.

Read More: *Dynamically Controlling the Timestep* on page 496

Assigning Different Timesteps to SubSystems

In addition to providing a dynamic timestepping algorithm on a global scale (i.e., for the entire model), GoldSim also enables you to apply dynamic timestepping to specific Containers. This allows you to specify different timesteps for different parts (i.e., Containers) in your model. For example, if one part of your model represented dynamics that changed very rapidly (requiring a 1 day timestep), while the rest of the model represented dynamics that changed much more slowly (requiring a 10 day timestep), you could assign a 10 day timestep to the model globally, and a 1 day timestep to the container representing the SubSystem that changed rapidly.

Read More: *Specifying Containers with Internal Clocks* on page 499

Accurately Simulating Discrete Events that Occur Between Timesteps

In some cases, events or other changes in the model may not fall exactly on a scheduled update. That is, some events or changes may actually occur between scheduled updates of the model. These trigger an "unscheduled update" of the model. Unscheduled updates are timesteps that are dynamically inserted by GoldSim during the simulation in order to more accurately simulate the system. That is, they are not specified directly prior to running the model. GoldSim inserts them automatically (and, generally, without you needing to be aware of it).

"Unscheduled updates" can be generated in the following ways:

- When events are ouput by a Timed Event, Event Delay, Discrete Change Delay or Time Series element;
- By manually specifying a dynamic timestep (i.e., dynamically controlling the time between updates);
- When a stock (e.g., Reservoir or Pool element) reaches an upper or lower bound;
- When a Resource becomes exhausted;
- When any element is triggered by an At Stock Test, At Date or At Etime triggering event; and
- By some specialized elements in GoldSim extension modules (Action and Function elements in the Reliability Module, Fund elements in the Financial Module, and Cell elements in the Flow Module and Contaminant Transport Module).

When any of these events occur, GoldSim automatically inserts an unscheduled update at the exact time that the event or change occurs. For example, if you had specified a one day timestep, and a Timed Event occurs at 33.65 days (i.e., between the scheduled one-day updates), GoldSim would insert an unscheduled update at 33.65 days.

Read More: *Understanding Timestepping in GoldSim* on page 481

By default, scheduled updates are always dynamically inserted by GoldSim. However, in some (rare) cases, you may want to prevent unscheduled updates from being inserted. For example, if your model included a specialized algorithm that was designed based on the assumption that the timestep was constant, inserting unscheduled updates could invalidate the algorithm. To support such situations, GoldSim allows you to disable unscheduled updates.

Warning: Because unscheduled updates are intended to more accurately represent a complex dynamic system, disabling this feature should be done with caution, and is generally not recommended.

Read More: o*Controlling Unscheduled Updates* on page 495

Using Advanced Algorithms to Solve Coupled Equations

For some special types of systems, GoldSim provides additional dynamic timestepping algorithms (different from the timestep algorithms described above) to more accurately solve these equations. For example, the Contaminant Transport Module utilizes dynamic timestep adjustment to solve the coupled differential equations associated with mass and heat transport.

This algorithm allows GoldSim to handle “stiff” systems (systems with widely varying time constants) as well as nonlinear aspects of the system in a very accurate and computationally efficient manner. This algorithm is discussed in the GoldSim Contaminant Transport Module User’s Guide.