**DFNgen V2.0 Documentation**

**Exponential Distribution Class Implementation**

**Introduction**

This document is intended for new developers working on DFNgen. It covers the implementation of the ‘*Distributions*’ class, and its composed exponential distribution class ‘*ExpDist*’ in DFNgen V2.0.

During DFNgen 2.0 development, new functionality was needed to allow for the control of the range of numbers produced by the exponential distribution. Previously, DFNgen V2.0 was developed using the C++ standard library, *random.*

**Need for a Customized Exponential Distribution**

There was need to control the minimum fracture size for exponential distributed fracture families for research purposes. Also, all fracture radii must always be greater than the minimum feature size *h.*

The exponential distribution favors small numbers that caused a lot of re-sampling when the distribution generated fracture radii of less than *h* or smaller than the user’s defined minimum radius. Re-sampling the standard library’s exponential distribution when the distribution produced numbers outside of the user’s defined ranged was found to be very inefficient and could halt program execution when the exponential mean did not match the range which the user had chosen. The program could re-sample the distribution thousands of times before an acceptable radius was generated.

With the standard library’s implementation, complete randomness is forced from the distribution. There was no way to control the range of numbers produced by the distribution. A way of limiting the output of the distribution was needed that did not involve re-sampling.

**Implementation Overview**

Our implementation uses the CDF determine the random variable range from which we need to sample. When the inverse CDF is sampled uniformly between 0 and 1, an exponential distribution will be produced that matches that of the standard library’s exponential distribution output. By limiting the random variable range, we can sample between the users desired minimum and maximum without generating numbers outside of that range.

To limit the range of output, we use the exponential CDF formula:

**rv = 1 – e (-lambda \* output)**, where rv is the random variable needed to produce ***output*** when plugged into the inverse CDF function:

***output =* -log(1-rv)/lambda.**

When the user’s defined minimum and maximum are plugged in to **o*utput***, we get the range which the distribution should be sample from in order to get a exponential distribution bounded by the users defined minimum and maximum.

These variables, the range to sample the exponential distribution, are saved to *minDistInput* and *maxDistInput* in the family’s corresponding *Shape* structure.

**Implementation Details**

ExpDist

Distributions

Our implementation uses composition for increased modularity and to increase the ease of adding additional distribution types in the future.

The *ExpDist* class is a sub-class of the *Distributions* class. This allows the programmer to only create one instance of the *Distributions* class, and the ExpDist class and any other distribution classes added in the future will be automatically set up and initialized by *Distributions’* constructor.

**Distributions Class**

The *Distributions* class contains functions and variables that are needed to initialize the *ExpDist* class, and likely other distribution classes added in the future. It also contains the *ExpDist* class within it.

When the *Distributions* class is created, its constructor function is called. This function creates and initializes the *ExpDist* class within the *Distributions* class.

One of the issues with the exponential distribution is that if given 1.0 as a random variable, the distribution returns inf. To maximize the range of numbers which can be produced, we need to know the largest value less than 1.0 that the computer is able to produce.

The *Distributions* class has a function called *getMaxDecimalDouble()*. During *Distributions* creation, *getMaxDecimalDouble()* returns the largest number less than 1, e.g. 0.999….9, to its maximum precision. This variable is saved to variable *maxInput* in the *Distributions* class. It is also passed to the *ExpDist* class during its creation.

Also in the *Distributions* class constructor, the function *checkDistributionUserInput()* is called. This function error checks user exponential input options and finishes initializing the exponential distribution. The function is written with the expectation for other distributions to be added and will be easy to modify.

In *checkDistributionUserInput()*, *minDistInput and maxDistInput* are initialized for each family using exponential distribution (see Implementation Overview). Error checks are performed to ensure *minDistInput* and *maxDistInput* are within the machines capabilities to produce. If they are set very high, plugging in *maxInput* (see above) into the distribution can produce a number smaller than the requested maximum, and possibly minimum. If the user defined maximum cannot be produced stochastically, the user is warned and the user defined maximum is set to the largest possible number that the machine can produce. The minimum is then checked to ensure it is still less than the maximum. If it is not, the error is reported to the user and the program terminates. Otherwise, everything is OK and the *ExpDist* class is ready to use.

**ExpDist Class**

After the *ExpDist* class has been initialized, the *getValue()* function can be used to return random numbers from the exponential distribution. The function has been overloaded to either be given the random input variable (random variable between 0 and 1) as an argument, or be given a range between 0 and 1 to generate random input variables from.

**Other Details**

The C++ standard random library is still used for generating uniform random reals. The 64-bit Mersenne twister engine random generator is the random generator used for all DFNgen’s random variables. It is created in *main()* and passed as a reference to the *Distributions* class during its creation.