See Also
interpolate1d interpolate1ddemo

Returns
Bool: T for success

1.5.17 randomnumbers - Tool

Package utility
Module mathematics

Random numbers from various statistical distributions
include "randomnumbers.g"

Constructors

randomnumbers	Construct an randomnumbers tool

Functions

binomial	Generate binomially distributed random numbers
discreteuniform	Generate uniformly distributed random integers
done	Delete the randomnumbers tool
erlang	Generate random numbers that have an Erlang distribution.
geometric	Generate geometrically distributed random numbers
hypergeometric	Generate random numbers that have an hyper-geometric distribution.
lognormal	Generate random numbers with a log-normal distribution
negativeexponential	Generate random numbers with a negative-exponential distribution
normal	Generate normally distributed random numbers
poisson	Generate Poisson distributed random numbers
reseed	Set the seed used by the random number generator.
type	Return the type of this tool
uniform	Generate uniformly distributed random numbers
weibull	Generate random numbers with a Weibull distribution

Description

The `randomnumbers` tool will generate pseudo-random numbers from a number of common distribution functions.
This tool will generate random numbers from the following distributions:

normal This is a continious distribution that is also known as the Gaussian distribution. The probability density function is given by:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-1}{2}\left(\frac{x-m}{\sigma}\right)^2\right) \tag{1.14}$$

where m is the mean and σ^2 is the variance.

uniform This is a continuous distribution where every number between the specified upper and lower bounds is equally probable. The returned values include the lower bound but not the upper bound. The probability density function is given by:

$$p(x) = \frac{1}{u-l} \tag{1.15}$$

where u is the upper and l is the lower bound.

discrete uniform This is a discrete distribution where all the integral values between the specified upper and lower bounds are equally probable. The returned values include the lower & upper bounds. The point probabilities are given by:

$$p(x) = \frac{1}{u-l} \tag{1.16}$$

where u is the upper and l is the lower bound and x is an integral value in the range $[l, u]$.

binomial This is a discrete distribution where integers between zero and a specified number of throws n may be returned. The point probabilities are given by:

$$p(x) = \left(\begin{array}{c} n \\ x \end{array}\right) p^x \, (1-p)^{n-x} \tag{1.17}$$

where n is the number of throws, p is the probability of of an individual throw and x is an integer in the range $[0, n]$.

poisson This is a discrete distribution where non-negative integers may be returned. The point probabilities are given by:

$$p(x) = \frac{e^{-m} m^x}{x!} \tag{1.18}$$

where m is the mean of of the distribution and x is an integer in the range $[0, \infty)$.

geometric This is a discrete distribution where non-negative integers may be returned. The point probabilities are given by:

$$p(x) = p\,(1-p)^x \tag{1.19}$$

where p is a probability and x is an integer in the range $[0, \infty)$.

hypergeometric This is a continuous distribution.

erlang This is a continuous distribution that is more commonly known as the Gamma distribution. The probability density function is given by:

$$p(x) = \frac{1}{\Gamma(\alpha+1)\,\beta^{\alpha+1}} x^{\alpha} e^{-x/\beta} \tag{1.20}$$

where the mean is given by $\beta(\alpha+1)$ and the variance is given by $\beta^2(\alpha+1)$. $\Gamma(x)$ is the gamma function.

lognormal This is a continuous distribution.

negative exponential This is a continuous distribution that is also known as the exponential distribution. The probability density function is given by:

$$p(x) = \frac{1}{m} \exp\left(\frac{-x}{m}\right) \tag{1.21}$$

where m is the mean. $p(x)$ is zero when x is negative. This distribution is sometimes known as the exponential distribution.

weibull This is a continuous distribution.

The builtin function `random()` of glish can be used to generate a sequence of uniformly distributed random numbers. These are random integers between optionally specified bounds. This is a discrete uniform distribution and hence is similar in functionality to the discreteuniform function in this tool.

See Also
Handbook of Mathematical Functions, Abramowitz and Stegun
The C++ class, Random.
The Glish built-in function, random.

Example
To use the functions in this tool we have to firstly, as shown in line 1, load the definition of the randomnumbers tool and then, as shown in line 2, construct a randomnumbers tool that will actually perform the computations.

```
include 'randomnumbers.g' #1
rand := randomnumbers();  #2
```

The `rand` tool is now ready for use. To generate a sequence of 100 random numbers which are normally distributed with a mean of 0.5 and variance of 1.0 the normal function is used. ie.,

```
x := rand.normal(0.5, 1.0, 100);
```

A 10 by 20 two-dimensional array of Poisson distributed random numbers, with a mean of 6.5, can be generated using,

```
x := rand.poisson(6.5, [10,20]);
```

The glish variable `x` would now have a shape of `[10,20]`, where each element of which will be a random number from a poisson distribution.
The random number generator will always produce the same sequence of numbers for a given set of seeds. To generate a new sequence of random numbers, one has to reset the random number generator by supplying a new seed. This can be done using:

```
rand.reseed(12.34);
```

where seed can be any number. If called with the unset value, which is the default, this function will use the glish random function to supply a seed.
Once you are finished with this tool you should call the done function to release all the memory used by this tool. If no other tools are using the server then this will also be shut down.

```
rand.done();
```

randomnumbers - Constructor

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Construct an randomnumbers tool

Synopsis
randomnumbers(host, forcenewserver)

Description
Use this constructor to create a randomnumbers tool. This is the only way to create such a tool. After construction all randomnumbers tools are identical and will produce the same sequence of randomnumbers. To change this you need to use the reseed function.
The host argument specifies which machine the precompiled server process should be started on. The default value of unset starts it on the same machine

as the one that glish is running on. Starting a server process on a remote machine allows you to use the memory and cpu on that machine. As the randomnumbers tool does not use much memory or processor time this is usually not worth it.
In order to run the server process on a remote machine you need to satisfy all the following conditions.

- It must be possible to start AIPS++ on the remote machine
- You must be able to log onto the remote machine without having to type a password
- The AIPSPATH environment variable must be defined on the remote machine. You may want to set this up in the relevant "dot" file eg., adding a line like `source /usr/local/aips++/aipsinit.csh` in your .cshrc file (for csh).

One quick way to check if all three conditions are met is to type, on your local machine:

```
rsh host 'echo \$AIPSPATH'
```

where host is replaced by the name of the remote machine. If the value of the AIPSPATH variable that is printed does not contain something like `aips-root architecture site host` and that all the values are correct for the remote machine you can be certain that starting the any AIPS++ server, on the remote host will not work
Each server process can only run one function at a time. Hence it is not possible, for example, to simultaneously calculate two different random numbers sequences using one server. To solve this you start two servers. The forcenewserver argument allows you to do this by overriding the default behaviour of having each randomnumbers tool share the same server.
For the randomnumbers tool, in 99.99% of all cases, the default values for the host and forcenewserver arguments will be fine.

Arguments

host	in	The host on which the server process should be run Allowed:string Default: unset
forcenewserver	in	Always start a new server Allowed:Boolean Default: F

Returns
A randomnumbers tool

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
rand.reseed(10);
print rand.uniform(-1, 3, 4);
print rand.discreteuniform(-1, 3, [4,2]);
rand.done()
```

This example creates a randomnumbers tool, and sets the seed to a new value. It then prints the values for random numbers from two different, but similar distributions. Finally the tool is shutdown to release all the memory it has consumed.

---

randomnumbers.binomial - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate binomially distributed random numbers

Synopsis
binomial(number, probability, shape)

Description
The binomial function will generate numbers that follow a binomial distribution. See the general tool description for a definition of the point probabilities.

The number, N, can be any positive integer and the probability must be a number between zero and one. The result will be integers in the range $0 \leq x \leq N$. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

number	in	Number of trials Allowed:integer Default: 1
probability	in	probability of success for one trial Allowed:double Default: 0.5
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
An integer scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
print rand.binomial(2, shape=100);
rand.done()
```

This example will print random numbers that correspond to tossing two coins one hundred time and counting the number of heads. Hence zero will appear one quarter of the time, one appear half of the time and two will appear one quarter of the time.

---

randomnumbers.discreteuniform - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate uniformly distributed random integers

Synopsis
discreteuniform(low, high, shape)

Description
The discreteuniform function will generate integers that are uniformly distributed between the specified lower and upper bounds. Both the lower and

upper bound are included in the range. See the general tool description for a more mathematical definition of the point probabilities.
The lower and upper bounds can be any integer and the lower bound cannot be larger than the upper bound. The result will be integers in the range low $\leq x \leq$ high. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

low	in	Lower bound Allowed:integer Default: -1
high	in	Upper bound Allowed:integer Default: 1
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
An integer scalar or array.

Example
See the example for the randomnumbers constructor.

---

randomnumbers.erlang - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate random numbers that have an Erlang distribution.

Synopsis
erlang(mean, variance, shape)

Description
The erlang function will generate numbers that follow a erlang, or Gamma

distribution. See the general tool description for a mathematical definition of the probability density function.
The mean can be any real number except zero and the variance must be a positive number. The result will be floating point numbers in the range $0 < x < \infty$ if the mean is positive and in the range $-\infty < x < 0$ if it is negative. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

mean	in	Mean of the distribution Allowed:double Default: 1.0
variance	in	Variance of the distribution Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
A floating point scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
print rand.erlang(4, 8);
rand.done()
```

This example will print a random number from the erlang distribution.

---

randomnumbers.geometric - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate geometrically distributed random numbers

Synopsis
geometric(probability, shape)

Description
The geometric function will generate numbers that follow a geometric distribution. See the general tool description for a mathematical definition of the point probabilities.
The probability must be a number between zero and one. The result will be integers in the range $0 \leq x \leq N$. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

probability	in	probability Allowed:double Default: 0.5
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
An integer scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
print rand.geometric();
rand.done()
```

This example will a random numbers from the geometric distribution that has a probability of 0.5.

---

randomnumbers.hypergeometric - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate random numbers that have an hyper-geometric distribution.

Synopsis
hypergeometric(mean, variance, shape)

Description

The hypergeometric function will generate numbers that follow a hyper-geometric distribution. I cannot find a mathematical definition of a probability density function that represents the output produced by this function.
The variance must be a positive number and the mean must be non-zero and less than the square-root of the variance. The result will be floating point numbers in the range $0 < x < \infty$ if the mean is positive and in the range $-\infty < x < 0$ if it is negative. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

mean	in	Mean of the distribution Allowed:double Default: 0.5
variance	in	Variance of the distribution Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
A floating point scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
numbers := rand.hypergeometric(2, 8, 1E3);
rand.done()
```

This example will calculate a thousand hypergeometric random numbers.

randomnumbers.normal - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate normally distributed random numbers

Synopsis

normal(mean, variance, shape)

Description

The normal function will generate numbers that follow a normal, or Gaussian distribution. See the general tool description for a mathematical definition of the probability density function.
The mean can be any real number and the variance must be a positive number. The result will be floating point numbers in the range $-\infty < x < \infty$.
The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

mean	in	Mean of the distribution Allowed:double Default: 0.0
variance	in	Variance of the distribution Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns

A floating point scalar or array.

Example

See the example in the randomnumbers tool description.

randomnumbers.lognormal - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate random numbers with a log-normal distribution

Synopsis
lognormal(mean, variance, shape)

Description
The lognormal function will generate numbers that follow a log-normal distribution. I cannot find a mathematical definition of a probability density function that represents the output produced by this function, but it appears to be similar to a Ricean distribution, which is the sum of the square of two Gaussian random variables.
The mean and variance can be any positive number. The result will be floating point numbers in the range $0 < x < \infty$. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

mean	in	Mean of the distribution Allowed:double Default: 1.0
variance	in	Variance of the distribution Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
A floating point scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
numbers := rand.lognormal(shape=[2,5]);
rand.done()
```

This example will calculate ten log-normal random numbers, storing the result in a two-dimensional array of shape [2,5]. The mean and variance both default to 1.0.

---

randomnumbers.negativeexponential - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate random numbers with a negative-exponential distribution

Synopsis
negativeexponential(mean, shape)

Description
The negativeexponential function will generate numbers that follow a negative-exponential or exponential distribution. See the general tool description for a mathematical definition of the probability density function. The mean can be any positive number. The result will be floating point numbers in the range $0 < x < \infty$. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

mean	in	Mean of the distribution Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
A floating point scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
numbers := rand.negativeexponential(2);
rand.done()
```

This example will calculate a random number from the negative-exponential distribution with a mean of two.

---

randomnumbers.poisson - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate Poisson distributed random numbers

Synopsis
poisson(mean, shape)

Description
The poisson function will generate numbers that follow a Poisson distribution. See the general tool description for a definition of the point probabilities. The mean can be any positive value. The result will be non-negative integers. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

mean	in	Mean of the distribution Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
An integer scalar or array.

Example
See the example in the randomnumbers tool description.

---

randomnumbers.uniform - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate uniformly distributed random numbers

Synopsis
uniform(low, high, shape)

Description

The uniform function will generate floating point numbers that are uniformly distributed between the specified lower and upper bounds. The lower is included in the range but the upper bond is not. See the general tool description for a more mathematical definition of the point probabilities. The lower and upper bounds can be any value and the lower bound must be less than the upper bound. The result will be integers in the range low $\leq x <$ high. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

low	in	Lower bound Allowed:integer Default: -1.0
high	in	Upper bound Allowed:integer Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
A floating point scalar or array.

Example
See the example for the randomnumbers constructor.

---

randomnumbers.weibull - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Generate random numbers with a Weibull distribution

Synopsis
weibull(alpha, beta, shape)

Description
The weibull function will generate numbers that follow a Weibull distribution. I cannot find a mathematical definition of a probability density function that represents the output produced by this function.
Alpha parameter must be a non-zero number and beta can be any value. The result will be floating point numbers in the range $0 < x < \infty$. The shape can be either an integer or a vector of integers where all the values are greater than zero. The default value of one will produce a scalar output.

Arguments

alpha	in	The alpha parameter Allowed:double Default: 1.0
beta	in	The beta parameter Allowed:double Default: 1.0
shape	in	Shape of the output array Allowed:integer or a vector of integers Default: 1

Returns
A floating point scalar or array.

Example

```
include 'randomnumbers.g'
rand := randomnumbers();
numbers := rand.weibull(1.5, 8, 12);
rand.done()
```

This example will calculate twelve weibull random numbers, storing the result in a vector.

---

randomnumbers.reseed - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Set the seed used by the random number generator.

Synopsis
reseed(seed)

Description
The pseudo-random number generator that is used by the randomnumbers tool can be set to different states, which will make it generate different random numbers. Specifying a number as the argument to this function you will produce a repeatable sequence of random numbers. Alternatively if you use the default value of unset then the glish random function will be used to generate the seed and the resulting random numbers will not be a repeatable sequence.

Arguments

seed	in	seed Allowed:double or unset Default: unset

Returns
T

Example
See the example in the randomnumbers tool description.

---

randomnumbers.done - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Delete the randomnumbers tool

Synopsis
done()

Description
The done function frees up all the memory associated with a randomnumbers tool. It will also shut down the server process if there are no other tools using the same server.

Returns
T

Example
See the examples in the randomnumbers tool description and the randomnumbers constructor.

---

randomnumbers.type - Function

Package	**utility**
Module	**mathematics**
Tool	**randomnumbers**

Return the type of this tool

Synopsis
type()

Description
This function returns the string 'randomnumbers'. It is used to determine if this glish variable is an randomnumbers tool.

Returns
string

1.5.18 randomnumbersdemo - Function

Package utility
Module mathematics

Demonstrate the randomnumbers tool

Synopsis
randomnumbersdemo()

Description
This function will demonstrate the the functions in the randomnumbers tool. It will send sample input and out put to the logger and then produce a pot showing examples of the different distributions available.

Returns
T

1.5.19 randomnumberstest - Function

Package utility
Module mathematics

Test the randomnumbers class

Synopsis
randomnumberstest()

Description
This class tests the randomnumbers tool by running all the functions and comparing the output with expected values. It does not no do a full statistical analysis. It will send, to the logger, an list of the tests being done and whether it succeeded or failed. This function will terminate, and return a fail, whenever a test does not succeed.