mcmc Reference Manual

Generated by Doxygen 1.5.1

Tue Mar 27 16:03:38 2007

Contents

Chapter 1

mcmc Namespace Index

1.1 mcmc Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

mcmc Hierarchical Index

2.1 mcmc Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

mcmc Class Index

3.1 mcmc Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

mcmc File Index

4.1 mcmc File List

Here is a list of all documented files with brief descriptions:

Chapter 5

mcmc Namespace Documentation

5.1 Density Namespace Reference

Functions

- double dbeta (const double x, const double a, const double b, const bool give_log)
- double dbinom (const int k, const int n, double p, bool give_log)
- double dcauchy (const double x, const double l, const double s, const bool give_log)
- double dchisq (const double x, const double n, const bool give_log)
- double ddirch (const std::vector< double > &p, const std::vector< double > &a, const bool give_log, const bool include_const)
- double dexp (const double x, const double b, const bool give_log)
- double df (const double x, const double m, const double n, const bool give_log)
- double dgamma (const double x, const double shape, const double scale, const bool give_log)
- double dgeom (const unsigned x, const double p, const bool give_log)
- double dhyper (const unsigned x, const unsigned r, const unsigned b, const unsigned n, bool giveLog)
- double dinvgamma (const double y, const double shape, const double scale, const bool give_log)
- double dlnorm (const double x, const double mu, const double sigma, const bool give_log)
- double dlogis (const double x, const double m, const double s, const bool give_log)

- double dmulti (const std::vector< int > &n, const std::vector< double > &p, const bool give_log, const bool include_factorial)
- double dnbinom (const unsigned x, const double n, const double p, const bool give_log)
- double dnorm (const double x_in, const double mu, const double sigma, const bool give_log)
- double dpois (const unsigned x, const double lambda, const bool give_log)
- double dt (const double x, const double n, const bool give_log)
- double dweibull (const double x, const double a, const double b, const bool give_log)
- double gamln (const double x)
- double logChoose (const double n, const double k)
- double lbeta (const double a, const double b)
- double BetaEntropy (const double a, const double b)

Variables

- const double MinGammaPar
- const double MaxGammaPar

5.1.1 Detailed Description

Used to isolate functions for density evaluation.

Provides a variety of numerical density routines. All of them are derived from R. Details are provided in the comments accompanying each one.

5.1.2 Function Documentation

5.1.2.1 double Density::BetaEntropy (const double *a*, const double *b*)

Entropy of a beta distribution with parameters a and b.

Formula:

$$(a-1)(\Psi(a)-\Psi(a+b))+(b-1)(\Psi(b)-\Psi(a+b))///-\log(\beta(a,b))$$

Parameters:

a first parameter of the beta distribution

b second parameter of the beta distribution

$\Psi(x)$ is Euler's psi function (also known as the digamma function).

Definition at line 1999 of file Density.cpp.

References lbeta().

5.1.2.2 double Density::dbeta (const double *x*, const double *a*, const double *b*, const bool *give_log* = `false`)

Density of the beta distribution

Returns the probability density associated with a beta variate:

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$$

Parameters:

x

a

b

give_log Return log density?

$$\mathrm{E}(x) = \frac{a}{a+b}$$

$$\mathrm{Var}(x) = \frac{ab}{(a+b)^2(a+b+1)}$$

This implementation is derived from R. It assumes that the caller has ensured that a and b are positive.

Definition at line 841 of file Density.cpp.

Referenced by logQBeta().

5.1.2.3 double Density::dbinom (const int *k*, const int *n*, double *p*, bool *give_log*)

Density of the binomial distribution.

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Returns probability of getting k successes in n binomial trials with a probability p of success on each trial, if give_log == false. If give_log == true, returns the natural logarithm of the probability.

Parameters:

- ***k*** Number of successes
- ***n*** Number of trials
- ***p*** Probability of success on each trial
- ***give_log*** Return log probability?

$$\mathrm{E}(x) = np$$

$$\mathrm{Var}(x) = np(1-p)$$

The implementation uses dbinom_raw from R

Definition at line 890 of file Density.cpp.

5.1.2.4 double Density::dcauchy (const double *x*, const double *l*, const double *s*, const bool *give_log*)

Density of the Cauchy distribution

$$f(x) = \frac{1}{\pi \mathrm{s}(1 + (\frac{x-\mathrm{l}}{\mathrm{s}})^2)}$$

Parameters:

- ***x*** the x value at which the density is to be calculated
- ***l*** the location parameter
- ***s*** the scale parameter
- ***give_log*** return natural log of density if true

The expectation and variance of the Cauchy distribution are infinite. The mode is equal to the location parameter.

This implementation is adapted from R v2.0. The sanity checks for s $>$ 0 and ISNAN(x) have been removed.

Definition at line 932 of file Density.cpp.

5.1.2.5 double Density::dchisq (const double *x*, const double *n*, const bool *give_log*)

Density of the chi-squared distribution

$$f(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$$

Parameters:

- ***x*** the chi-squared variate whose density is desired
- ***n*** degrees of freedom for the chi-squared density
- ***give_log*** true if natural logarithm of density is desired

$$\mathrm{E}(x) = \mathrm{n}$$

$$\mathrm{Var}(x) = 2\mathrm{n}$$

From R v2.0. The implementation simply calls dgamma with shape = n/2 and scale = 2.

Definition at line 978 of file Density.cpp.

References dgamma().

5.1.2.6 double Density::ddirch (const std::vector< double > & *p*, const std::vector< double > & *a*, const bool *give_log* = `false`, const bool *include_const* = `true`)

Density of the Dirichlet distribution

A brute-force implementation of the Dirichlet density:

$$f(\mathbf{p})/// = \Gamma(\sum_k a_k) \prod_k \frac{p_k^{a_k - 1}}{\Gamma(a_k)}$$

Parameters:

- ***p*** Vector of probabilities
- ***a*** Vector of Dirichlet parameters
- ***give_log*** Return log density?
- ***include_const*** Include normalizing constant

Definition at line 995 of file Density.cpp.

References gamln(), and Util::log_dbl_min.

Referenced by logQDirch().

5.1.2.7 double Density::dexp (const double *x*, const double *b*, const bool *give_log*)

Exponential density

$$f(x) = \frac{1}{b}e^{-x/b}$$

Parameters:

x the exponential variate

b the parameter of the exponential density

give_log return log density if true

Returns:

0 or Util::log_dbl_min if x $<$ 0

$$\mathrm{E}(x) = \mathrm{b}$$

$$\mathrm{Var}(x) = \mathrm{b}$$

Derived from R v2.0. Does not propagate NaNs. Does not check to ensure b $>$ 0.

Definition at line 1053 of file Density.cpp.

5.1.2.8 double Density::df (const double *x*, const double *m*, const double *n*, const bool *give_log*)

Density of the F distribution

$$f(x) = \frac{\Gamma((m+n)/2)}{\Gamma(m/2)\Gamma(n/2)}(m/n)^{m/2}x^{m/2-1}(1+(m/n)x)^{-(m+n)/2}$$

Parameters:

x the F variate

m “numerator” degrees of freedom

n “denominator” degrees of freedom

give_log return log density if true

$$\mathrm{E}(x) = \frac{m}{m-2}, m > 2$$

$$\mathrm{Var}(x) = \frac{2m^2(n-2)}{n(m+2)}, n > 2$$

Derived from R v2.0. Does not do isnan() check on arguments. Does not check m $>$ 0 and n $>$ 0. Callers must ensure that these conditions are met.

Definition at line 1109 of file Density.cpp.

5.1.2.9 double Density::dgamma (const double *x*, const double *shape*, const double *scale*, const bool *give_log*)

Density of the gamma distribution.

Returns the probability density associated with a gamma variate:

$$f(x) = \frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s}$$

Parameters:

x A gamma variate (x)

shape Shape of the distribution (a)

scale Scale of the distribution (s)

give_log Return log density?

$$\mathrm{E}(x) = sa$$

$$\mathrm{Var}(x) = s^2 a$$

The implementation is derived from R.

Definition at line 1146 of file Density.cpp.

References Util::dbl_max.

Referenced by dchisq().

5.1.2.10 double Density::dgeom (const unsigned *x*, const double *p*, const bool *give_log*)

Density of the geometric distribution

$$f(x) = p(1-p)^x$$

Parameters:

x the (integer) geometric variate

p the parameter of the geometric distribution

give_log return log density if true

$$\mathrm{E}(x) = \frac{1-p}{p}$$

$$\text{Var}(x) = \frac{1-p}{p^2}$$

Derived from R v2.0. Does not check isnan() on x and p. Does not check for 0 < p < 1. Changed x to unsigned int, so check on it is no longer required.

Definition at line 1215 of file Density.cpp.

5.1.2.11 double Density::dhyper (const unsigned *x*, const unsigned *r*, const unsigned *b*, const unsigned *n*, bool *giveLog*)

Density of the hypergeometric distribution.

$$f(x) = \frac{\binom{r}{x}\binom{b}{n-x}}{\binom{r+b}{n}}$$

Returns the probability of choosing x white balls in a sample of size n from an urn with r white balls and b black balls (sampling without replacement.

Parameters:

- ***x*** The number of white balls in the sample
- ***r*** The number of white balls in the urn
- ***b*** The number of black balls in the urn
- ***n*** The sample size
- ***giveLog*** Return log probability?

$$\text{E}(x) = n(\frac{r}{r+b})$$

$$\text{Var}(x) = \frac{n(\frac{r}{r+b})(1-\frac{r}{r+b})((r+b)-n)}{r+b-1}$$

The code is modified from R v1.8.1 to take unsigned integer arguments rather than doubles.

Definition at line 1278 of file Density.cpp.

5.1.2.12 double Density::dinvgamma (const double *y*, const double *shape*, const double *scale*, const bool *give_log*)

Density of the inverse gamma distribution.

Returns the probability density associated with an inverse gamma variate:

$$f(y) = f(1/x)$$

$$f(y) = \frac{s^a}{\Gamma(a)} y^{-(a+1)} e^{-s/y}$$

Parameters:

y An inverse gamma variate (y)

shape Shape of the distribution (a)

scale Scale of the distribution (s)

give_log Return log density?

$$\mathrm{E}(x) = \frac{s}{a-1} \text{ for } a > 1$$

$$\mathrm{Var}(x) = \frac{s^2}{(a-1)^2(a-2)} \text{ for } a > 2$$

The implementation is based on the R implementation of dgamma(). It first calculates

$$p(y) = \frac{(s/y)^a e^{-s/y}}{\Gamma(a)}$$

$f(y)$ is then given by

$$f(y) = \frac{p(y)}{y}$$

Definition at line 1323 of file Density.cpp.

References gamln().

5.1.2.13 double Density::dlnorm (const double *x*, const double *mu*, const double *sigma*, const bool *give_log*)

Density of the lognormal distribution

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

Parameters:

x the lognormal variate

mu logarithm of the mean of the corresponding normal (μ)
sigma logarithm of the sd of the corresponding normal (σ)
give_log return log density if true

$$\begin{aligned} \mathrm{E}(x) &= e^{\mu+\sigma^2/2} \\ \mathrm{Var}(x) &= e^{2\mu+\sigma^2}(e^{\sigma^2}-1) \\ \mathrm{mode} &= \frac{e^{\mu}}{e^{\sigma^2}} \\ \mathrm{median} &= e^{\mu} \end{aligned}$$

Derived from R v2.0. Does not do isnan() checks on arguments. Does not check for sigma > 0.

Definition at line 1380 of file Density.cpp.

5.1.2.14 double Density::dlogis (const double *x*, const double *m*, const double *s*, const bool *give_log*)

Density of the logistic distribution

$$f(x) = \frac{1}{s}\frac{e^{\frac{x-m}{s}}}{(1+e^{\frac{x-m}{s}})^2}$$

or equivalently (dividing numerator and denominator by $e^{2\frac{x-m}{s}}$)

$$f(x) = \frac{1}{s}\frac{e^{\frac{-(x-m)}{s}}}{(1+e^{\frac{-(x-m)}{s}})^2}$$

Parameters:

x the logistic variate
m the location parameter
s the scale parameter
give_log return log density if true

$$\begin{aligned} \mathrm{E}(x) &= m \\ \mathrm{Var}(x) &= \frac{\pi^2 s^2}{3} \end{aligned}$$

Derived from R v2.0. Does not do isnan() checks on parameters. Does not check for scale > 0.

Definition at line 1429 of file Density.cpp.

5.1.2.15 double Density::dmulti (const std::vector< int > & *n*, const std::vector< double > & *p*, const bool *give_log* = `false`, const bool *include_factorial* = `false`)

Density of the multinomial distribution

A brute-force implementation of the multinomial density

$$f(\mathbf{n}) = \binom{\sum_i n_i}{n_1 \dots n_I} \prod_i p_i^{n_i}$$

Parameters:

n Vector of observations

p Vector of probabilities

give_log Return log probability

include_factorial Leave out combinatorial coefficient?

Definition at line 1452 of file Density.cpp.

References Util::dbl_min, gamln(), and Util::log_dbl_min.

5.1.2.16 double Density::dnbinom (const unsigned *x*, const double *n*, const double *p*, const bool *give_log*)

Density of the negative binomial distribution

$$f(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

Parameters:

x the negative binomial variate

n the “size” parameter

p the “probability” parameter

give_log return log density if true

$$\mathrm{E}(x) = \frac{x(1-p)}{p}$$
$$\mathrm{Var}(x) = \frac{x(1-p)}{p^2}$$

Derived from R v2.0. Does not check isnan() on arguments. Does not check 0 < p < 1 or n > 0. Allows non-integer n (as in R). Integer checks for x not needed, since it is passed as unsigned.

Definition at line 1524 of file Density.cpp.

5.1.2.17 double Density::dnorm (const double *x_in*, const double *mu*, const double *sigma*, const bool *give_log*)

Density of the normal distribution.

Returns the probability density associated with a normal variate:

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

Parameters:

x_in A normal variate (x)
mu Mean (μ)
sigma Standard deviation (σ)
give_log Return log density?

$$\mathrm{E}(x) = \mu$$
$$\text{Var}(x) = \sigma^2$$

This implementation is derived from Mathlib via R.

Definition at line 1577 of file Density.cpp.

Referenced by logQNorm().

5.1.2.18 double Density::dpois (const unsigned *x*, const double *lambda*, const bool *give_log*)

Density of the Poisson distribution

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Parameters:

x the Poisson variate
lambda the Poisson parameter (λ)
give_log return log density if true

$$\mathrm{E}(x) = \lambda$$
$$\text{Var}(x) = \lambda$$

Derived from R v2.0. No isnan() checks. Integer check on x discarded since it is unsigned.

Definition at line 1628 of file Density.cpp.

5.1.2.19 double Density::dt (const double *x*, const double *n*, const bool *give_log*)

Density of Student's t distribution

$$f(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu}\Gamma(\frac{\nu}{2})(1+\frac{x^2}{\nu})^{(\nu+1)/2}}$$

Parameters:

x the t deviate

n the degrees of freedom (ν)

give_log return log density if true

$$\mathrm{E}(x) = 0 \quad , \quad \nu > 1$$

$$\mathrm{Var}(x) = \frac{\nu}{\nu-2} \quad , \quad \nu > 2$$

Derived from R v2.0. Does not check isnan() or isfinite() on arguments. Does not verify n >= 0.

Definition at line 1680 of file Density.cpp.

Referenced by lot::hypergeom().

5.1.2.20 double Density::dweibull (const double *x*, const double *a*, const double *b*, const bool *give_log*)

Density of the Weibull distribution

$$f(x) = (\frac{a}{b})(\frac{x}{b})^{a-1}e^{-\frac{x}{b}^a}$$

Parameters:

x the Weibull variate

a the "shape" parameter

b the "scale" parameter

give_log return log density if true

$$\mathrm{E}(x) = b\Gamma(1+\frac{1}{a})$$

$$\mathrm{Var}(x) = b^2(\Gamma(1+\frac{2}{a}) - \Gamma(1+\frac{1}{a})^2)$$

Derived from R v2.0. Does not check isnan() on arguments. Does not check for finite x.

Definition at line 1730 of file Density.cpp.

5.1.2.21 double Density::gamln (const double *x*)

Natural log of the gamma function

Parameters:

x

from

NIST Guide to Available Math Software. Source for module GAMLN from package CMLIB. Retrieved from TIBER on Wed Apr 29 17:30:20 1998.

Definition at line 1785 of file Density.cpp.

Referenced by ddirch(), dinvgamma(), dmulti(), lbeta(), and logChoose().

5.1.2.22 double Density::lbeta (const double *a*, const double *b*)

Logarithm of $\beta(a, b)$.

Formula:

$$\log(\Gamma(a)) + \log(\Gamma(b)) - \log(\Gamma(a + b))$$

Parameters:

a

b

Definition at line 1981 of file Density.cpp.

References gamln().

Referenced by BetaEntropy().

5.1.2.23 double Density::logChoose (const double *n*, const double *k*)

Logarithm of n choose k.

Formula:

$$\log(\gamma(n + 1)) - \log(\gamma(k + 1)) - log(\gamma(n - k + 1))$$

Parameters:

n Sample size

k Number of successes

Definition at line 1967 of file Density.cpp.

References gamln().

5.1.3 Variable Documentation

5.1.3.1 const double Density::MaxGammaPar

Maximum value of parameters to be used in gamln()

Definition at line 45 of file Density.cpp.

5.1.3.2 const double Density::MinGammaPar

Minimum value of parameters to be used in gamln()

Definition at line 44 of file Density.cpp.

5.2 keh Namespace Reference

Classes

- class Accumulate< false, T >
- class Accumulate< true, T >

5.3 lot_conditions Namespace Reference

5.4 MCMC Namespace Reference

Variables

- const double MinBetaPar = 1.0e-1

  minimum value allowed for beta parameter

- const double MaxBetaPar = 1.0e4

  maximum value allowed for beta parameter

- const double MinDirchPar = 1.0e-1

  minimum value allowed for Dirichlet parameter

- const double MaxDirchPar = 1.0e4

  maximum value allowed for Dirichlet parameter

5.5 Util Namespace Reference

Classes

- class PrintForVector

Functions

- template<class C> C vectorMin (std::vector< C > &v)
- template<class C> C vectorMax (std::vector< C > &v)
- double round (double x)
- double safeLog (double x)
- double sqr (double x)
- template<class C> void FlushVector (std::vector< C > &v)
- template<class Except, class Assertion> void Assert (Assertion assert)
- template<class To, class From> std::vector< To > vector_cast (std::vector< From > &x)
- double round (const double x)
- double safeLog (const double x)
- double sqr (const double x)

Variables

- const double dbl_eps = std::numeric_limits<double>::epsilon()

  minimum representable value of 1.0 - x

- const double dbl_max = std::numeric_limits<double>::max()

  maximum value of double

- const double dbl_min = std::numeric_limits<double>::min()

  minimum (positive) value of double

- const int int_max = std::numeric_limits<int>::max()

  maximum value of integer

- const int int_min = std::numeric_limits<int>::min()

  minimum value of integer

- const unsigned uint_max = std::numeric_limits<unsigned>::max()

  maximum value of unsigned integer

- const long long_max = std::numeric_limits<long>::max()

maximum value of long integer

- const long long_min = std::numeric_limits<long>::min()

  minimum value of long integer

- const unsigned long ulong_max = std::numeric_limits<unsigned long>::max()

  maximum value of unsigned long integer

- const double log_dbl_max = log(dbl_max)

  log(dbl_max)

- const double log_dbl_min = log(dbl_min)

  log(dbl_min)

5.5.1 Detailed Description

Used to isolate utility functions and numerical constants.

Provides a variety of numerical constants related to double precision and integer arithmetic, a small collection of utility functions, and for probability density functions.

5.5.2 Function Documentation

5.5.2.1 template<class Except, class Assertion> void Util::Assert (Assertion *assert*) `[inline]`

Assertion template for error checking

Parameters:

assert the assertion to check

This template throws an exception of type Except when the assertion of type Assertion is false. It is shamelessly adapted (stolen) from Stroustrup, The C++ Programming Language, 3rd ed.

Definition at line 115 of file util.h.

5.5.2.2 template<class C> void Util::FlushVector (std::vector< C > & *v*)

Empty a vector, and ensure that its capacity is zero

Parameters:

v The vector to be emptied

Uses the trick on p. 487 of Stroustrup (3rd ed.): create a temporary vector with zero capacity and swap it with the one to be erased.

Definition at line 101 of file util.h.

5.5.2.3 double Util::round (const double *x*)

Round a floating point number

Formula:

$$\text{floor}(x + 0.5)$$

Parameters:

x

Definition at line 48 of file util.cpp.

5.5.2.4 double Util::round (const double *x*)

Round a floating point number

Formula:

$$\text{floor}(x + 0.5)$$

Parameters:

x

Definition at line 48 of file util.cpp.

5.5.2.5 double Util::safeLog (const double *x*)

Returns log_dbl_min if x < log_dbl_min

Parameters:

x The logarithm to check

Definition at line 57 of file util.cpp.

References log_dbl_min.

Referenced by logQBeta(), logQDirch(), and logQNorm().

5.5.2.6 double Util::safeLog (const double *x*)

Returns log_dbl_min if x < log_dbl_min

Parameters:

x The logarithm to check

Definition at line 57 of file util.cpp.

References log_dbl_min.

Referenced by logQBeta(), logQDirch(), and logQNorm().

5.5.2.7 double Util::sqr (const double *x*)

Returns x^2

Parameters:

x

Definition at line 66 of file util.cpp.

Referenced by keh::Accumulate< true, T >::Accumulate(), and keh::Accumulate< false, T >::Accumulate().

5.5.2.8 double Util::sqr (const double *x*)

Returns x^2

Parameters:

x

Definition at line 66 of file util.cpp.

Referenced by keh::Accumulate< false, T >::Accumulate(), and keh::Accumulate< true, T >::Accumulate().

5.5.2.9 template<class To, class From> std::vector<To> Util::vector_cast (std::vector< From > & *x*) `[inline]`

Cast all elements of a vector from one type to another

Parameters:

x the vector with elements to cast

This template will cast elements of type From to elements of type To and return the resulting vector

Definition at line 127 of file util.h.

5.5.2.10 template<class C> C Util::vectorMax (std::vector< C > & *v*) `[inline]`

Returns maximum element in v.

This is a simple wrapper around std::max_element()

Parameters:

- ***v*** The vector whose maximum element is sought

Definition at line 85 of file util.h.

5.5.2.11 template<class C> C Util::vectorMin (std::vector< C > & *v*) `[inline]`

Returns minimum element in v.

This is a simple wrapper around std::min_element()

Parameters:

- ***v*** The vector whose minimum element is sought

Definition at line 74 of file util.h.

Chapter 6

mcmc Class Documentation

6.1 keh::Accumulate< false, T > Class Template Reference

```
#include <statistics.h>
```

Public Member Functions

- Accumulate (std::vector< T > &x, double &sum, double &sumsq, unsigned long &n)

6.1.1 Detailed Description

template<class T> class keh::Accumulate< false, T >

Designed for use with a vector<T>, where a method in T is called before this constructor to provide an implementation of operator∗ that returns an appropriate member from T (one that can be converted via default conversions to a double)

Definition at line 58 of file statistics.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 template<class T> keh::Accumulate< false, T >::Accumulate (std::vector< T > & *x*, double & *sum*, double & *sumsq*, unsigned long & *n*) `[inline]`

Constructor

Parameters:

- ***x*** vector of values
- ***sum*** sum of values
- ***sumsq*** sum of squared values
- ***n*** number of values, i.e., length of the vector

The constructor is useful because of its side effects, namely sum, sumsq, and n are adjusted and are accesible from the calling function

Definition at line 71 of file statistics.h.

References Util::sqr().

The documentation for this class was generated from the following file:

- statistics.h

6.2 keh::Accumulate< true, T > Class Template Reference

```
#include <statistics.h>
```

Public Member Functions

- Accumulate (std::vector< T > &x, double &sum, double &sumsq, unsigned long &n)

6.2.1 Detailed Description

template<class T> class keh::Accumulate< true, T >

Specialization of Accumulate for arithmetic types

Definition at line 94 of file statistics.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 template<class T> keh::Accumulate< true, T >::Accumulate (std::vector< T > & *x*, double & *sum*, double & *sumsq*, unsigned long & *n*) `[inline]`

Constructor

Parameters:

x	vector of values
sum	sum of values
sumsq	sum of squared values
n	number of values, i.e., length of the vector

The constructor is useful because of its side effects, namely sum, sumsq, and n are adjusted and are accesible from the calling function

Definition at line 107 of file statistics.h.

References Util::sqr().

The documentation for this class was generated from the following file:

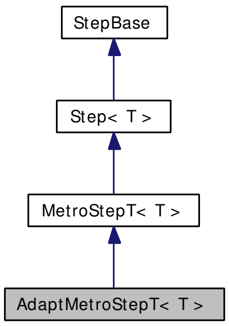
- statistics.h

6.3 AdaptMetroStepT< T > Class Template Reference

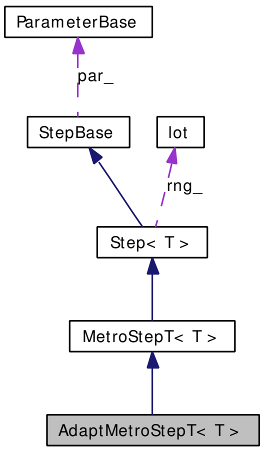
Implements Metropolis-Hastings step for a parameter.

`#include <MCMC.h>`

Inheritance diagram for AdaptMetroStepT< T >:



Collaboration diagram for AdaptMetroStepT< T >:



Public Member Functions

- AdaptMetroStepT (ParameterT< T > ∗parameter, const unsigned long nBurnin, const unsigned long adapt, const unsigned long interval)
- void DoStep (void)

6.3.1 Detailed Description

template<typename T> class AdaptMetroStepT< T >

Implements Metropolis-Hastings step for a parameter.

This version allows an adaptive phase to be included in during the burnin

Definition at line 420 of file MCMC.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 template<typename T> AdaptMetroStepT< T >::AdaptMetroStepT (ParameterT< T > ∗ *parameter*, const unsigned long *nBurnin*, const unsigned long *adapt*, const unsigned long *interval*) `[inline, explicit]`

Constructor

Parameters:

parameter	Pointer to the parameter associated with this step
nBurnin	Number of iterations in burn in
adapt	Number of iterations for adaptation
interval	Number of iterations between adaptive adjustments

Definition at line 429 of file MCMC.h.

6.3.3 Member Function Documentation

6.3.3.1 template<typename T> void AdaptMetroStepT< T >::DoStep (void) `[inline, virtual]`

Simple modification to allow adaptive phase of M-H sampling

Reimplemented from MetroStepT< T >.

Definition at line 440 of file MCMC.h.

References MetroStepT< T >::Accept().

The documentation for this class was generated from the following file:

- MCMC.h

6.4 BadCol Class Reference

Exception thrown on bad column index.

```
#include <DataTable.h>
```

6.4.1 Detailed Description

Exception thrown on bad column index.

Definition at line 70 of file DataTable.h.

The documentation for this class was generated from the following file:

- DataTable.h

6.5 BadCT Struct Reference

6.5.1 Detailed Description

Definition at line 65 of file MCMC.cpp.

The documentation for this struct was generated from the following file:

- MCMC.cpp

6.6 BadRow Class Reference

Exception thrown on bad row index.

```
#include <DataTable.h>
```

6.6.1 Detailed Description

Exception thrown on bad row index.

Definition at line 75 of file DataTable.h.

The documentation for this class was generated from the following file:

- DataTable.h

6.7 BadValue Struct Reference

6.7.1 Detailed Description

Definition at line 64 of file MCMC.cpp.

The documentation for this struct was generated from the following file:

- MCMC.cpp

6.8 DataTable< Type > Class Template Reference

Provides access to homogeneous tabular data.

`#include <DataTable.h>`

Public Member Functions

- DataTable (const bool columnLabels=true, const bool rowLabels=false)
- enum DataTableResult Read (const std::string fileName)
- void SetWidth (const std::string s)
- Type Value (const unsigned row, const unsigned col) const
- void SetValue (const unsigned row, const unsigned col, const Type value)
- std::string ColumnLabel (const unsigned index) const
- std::string RowLabel (const unsigned index) const
- std::vector< Type > RowVector (const unsigned row) const
- std::vector< Type > ColumnVector (const unsigned col) const
- void PrintTable (std::ostream &out=std::cout)
- void Flush (void)
- unsigned Rows (void) const
- unsigned Columns (void) const
- unsigned ColumnLabels (void) const
- void SetZero (void)

6.8.1 Detailed Description

template<typename Type> class DataTable< Type >

Provides access to homogeneous tabular data.

The DataTable class reads homogeneous tabular data, i.e., numerical data that is either all of type Type or that can be converted to Type using standard conversions. Rows or columns or both can be labeled, but labels are not required. A simple method for stream output of errors is also provided.

Definition at line 99 of file DataTable.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 template<typename Type> DataTable< Type >::DataTable (const bool *columnLabels* = `true`, const bool *rowLabels* = `false`) `[inline]`

Constructor – no default constructor is provided.

Parameters:

columnLabels Are columns labeled?

rowLabels Are rows labeled?

Initializes data structures. Use Read() to collect the data.

Definition at line 113 of file DataTable.h.

6.8.3 Member Function Documentation

6.8.3.1 template<typename Type> enum DataTableResult DataTable< Type >::Read (const std::string *fileName*) `[inline]`

Read data from a file.

Parameters:

fileName The name of the file from which data is to be read

Returns:

notEmptyError If the DataTable is not empty
labelError If there is an error reading column labels
valueError If there is an error reading values
openError If filename could not be opened for reading
readSuccess If everything works

The DataTable must be empty for data to be read. If it has been used before, Flush() must be used to re-initialize the internal state.

Definition at line 129 of file DataTable.h.

6.8.3.2 template<typename Type> void DataTable< Type >::SetWidth (const std::string *s*) `[inline]`

Sets width of output based on length of string

Parameters:

s The string used to set the width

Definition at line 152 of file DataTable.h.

6.8.3.3 template<typename Type> Type DataTable< Type >::Value (const unsigned *row*, const unsigned *col*) const `[inline]`

Value of the data at specified row and column

Parameters:

row Index of the data row

col Index of the data column

Definition at line 165 of file DataTable.h.

References argCheck_.

Referenced by DataTable< Type >::ColumnVector().

6.8.3.4 template<typename Type> void DataTable< Type >::SetValue (const unsigned *row*, const unsigned *col*, const Type *value*) `[inline]`

Set value of the data at specified row and column

Parameters:

row Index of the data row

col Index of the data column

value Value to be inserted

Definition at line 177 of file DataTable.h.

References argCheck_.

6.8.3.5 template<typename Type> std::string DataTable< Type >::ColumnLabel (const unsigned *index*) const `[inline]`

Label associated with a particular column index

Parameters:

index column index

Definition at line 189 of file DataTable.h.

References argCheck_.

6.8.3.6 template<typename Type> std::string DataTable< Type >::RowLabel (const unsigned *index*) const `[inline]`

Label associated with a particular row index

Parameters:

index row index

Definition at line 199 of file DataTable.h.

References argCheck_.

Referenced by DataTable< Type >::PrintTable().

6.8.3.7 template<typename Type> std::vector<Type> DataTable< Type >::RowVector (const unsigned *row*) const `[inline]`

An entire row of the data matrix

Parameters:

row Index of the data row

Definition at line 208 of file DataTable.h.

References argCheck_.

6.8.3.8 template<typename Type> std::vector<Type> DataTable< Type >::ColumnVector (const unsigned *col*) const `[inline]`

An entire column of the data matrix

Parameters:

col Index of the data column

Definition at line 217 of file DataTable.h.

References argCheck_, and DataTable< Type >::Value().

6.8.3.9 template<typename Type> void DataTable< Type >::PrintTable (std::ostream & *out* = `std::cout`) `[inline]`

Print the table to the specified stream

Parameters:

out The stream for output (defaults to std::cout)

Definition at line 230 of file DataTable.h.

References DataTable< Type >::RowLabel().

6.8.3.10 template<typename Type> void DataTable< Type >::Flush (void) `[inline]`

Re-initialize internal data structures.

Definition at line 244 of file DataTable.h.

6.8.3.11 template<typename Type> unsigned DataTable< Type >::Rows (void) const `[inline]`

Number of rows in the data

Definition at line 253 of file DataTable.h.

6.8.3.12 template<typename Type> unsigned DataTable< Type >::Columns (void) const `[inline]`

Number of columns in the data

Definition at line 259 of file DataTable.h.

6.8.3.13 template<typename Type> unsigned DataTable< Type >::ColumnLabels (void) const `[inline]`

Number of column labels

Definition at line 265 of file DataTable.h.

6.8.3.14 template<typename Type> void DataTable< Type >::SetZero (void) `[inline]`

Set all data elements to zero

Definition at line 271 of file DataTable.h.

The documentation for this class was generated from the following file:

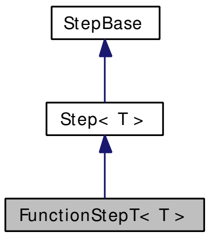
- DataTable.h

6.9 FunctionStepT< T > Class Template Reference

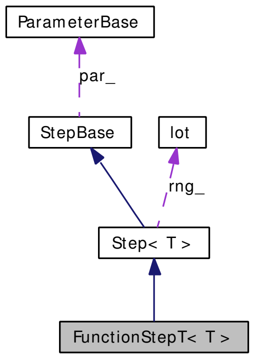
Implements a deterministic node.

`#include <MCMC.h>`

Inheritance diagram for FunctionStepT< T >:



Collaboration diagram for FunctionStepT< T >:



Public Member Functions

- FunctionStepT (ParameterT< T > ∗parameter)
- const T Value (void)

6.9.1 Detailed Description

template<typename T> class FunctionStepT< T >

Implements a deterministic node.

Often parameters of interest are deterministic functions of other statistical parameters. A FunctionStep allows us to express that relationship. DoStep() simply invokes the Function() associated with this parameter and returns the result.

Definition at line 539 of file MCMC.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 template<typename T> FunctionStepT< T >::FunctionStepT (ParameterT< T > ∗ *parameter*) `[inline, explicit]`

Constructor

Parameters:

parameter Pointer to parameter associated with this step

Definition at line 545 of file MCMC.h.

6.9.3 Member Function Documentation

6.9.3.1 template<typename T> const T FunctionStepT< T >::Value (void) `[inline]`

Return the value associated with this node

The value is returned by the Function() associated with this paramter.

Definition at line 553 of file MCMC.h.

The documentation for this class was generated from the following file:

- MCMC.h

6.10 lot Class Reference

Provides a series of random number generators.

`#include <lot.h>`

Public Types

- enum { RAN_POL = 1, RAN_KNU, RAN_MT }
- enum { PRECISE = 100, FAST }
- enum { OPEN = 1000, ZERO, ZERO_ONE }

Public Member Functions

- lot (int type=RAN_MT, int gType=ZERO)
- ∼lot (void)
- void set_generator (int type, int gType)
- long seed (void)
- long init_seed (void)
- void randomize (int spin=100)
- void set_seed (long s)
- void dememorize (int spin=100)
- void ran_start (long seed)
- void ranf_start (long seed)
- int random_int (int)
- long random_long (long maxval=Util::int_max)
- long ran_knu (void)
- double uniform (void)
- void ran_array (std::vector< long > &x, int n)
- void ranf_array (std::vector< double > &aa, int n)
- void MT_sgenrand (long seed)
- void MT_init_by_array (unsigned long ∗init_key, int key_length)
- void MT_R_initialize (int seed)
- unsigned long MT_genrand_int (void)
- double MT_genrand (void)
- double MT_genrand_with_zero (void)
- double MT_genrand_with_zero_one (void)
- bool Set_MT (int type)
- double beta (double aa, double bb)
- int binom (double nin, double pp)
- double cauchy (double l, double s)
- double chisq (double n)

- std::vector< double > dirichlet (std::vector< double > c)
- double expon (void)
- double exponential (const double lambda)
- double f (double m, double n)
- double gamma (double a, double scale=1.0)
- double geom (double p)
- int hypergeom (int nn1, int nn2, int kk)
- double igamma (const double a, const double s=1.0)
- double lnorm (double logmean, double logsd)
- double logis (double location, double scale)
- std::vector< int > multinom (unsigned n, const std::vector< double > &p)
- double nbinom (double n, double p)
- int poisson (double mu)
- double norm (const double mu, const double sd)
- double snorm (void)
- double t (double df)
- double weibull (double shape, double scale)
- double get_ran_u_0 (void) const

6.10.1 Detailed Description

Provides a series of random number generators.

This class provides a series of random number generators. Three different uniform random number generators are provided. By default the Mersenne twister is used. Sources are acknowledged in the source code by including copyright information (where appropriate).

It is given the name "lot" because the noun lot is defined as "an object used in deciding something by chance" according to The New Merriam-Webster Dictionary,

To seed the random number generator with a specific value (useful for debugging stochastic simulations), do the following:

```
lot rng;
rng.set_seed(1234L);
```

You can, of course pick any number you like.

Definition at line 67 of file lot.h.

6.10.2 Member Enumeration Documentation

6.10.2.1 anonymous enum

Enumerator:

- ***RAN_POL*** Paul's original RNG from J. Monahan, NCSU.
- ***RAN_KNU*** Knuth's rng.c translated to C++.
- ***RAN_MT*** Mersenne twister MT19937.

Definition at line 69 of file lot.h.

6.10.2.2 anonymous enum

Enumerator:

- ***PRECISE*** use double version of uniform with RAN_KNU
- ***FAST*** retrieve double from uniform long with RAN_KNU

Definition at line 75 of file lot.h.

6.10.2.3 anonymous enum

Enumerator:

- ***OPEN*** uniform on (0,1) with RAN_MT
- ***ZERO*** uniform on [0,1) with RAN_MT
- ***ZERO_ONE*** uniform on [0,1] with RAN_MT

Definition at line 80 of file lot.h.

6.10.3 Constructor & Destructor Documentation

6.10.3.1 lot::lot (int *type* = `RAN_MT`, int *gType* = `ZERO`)

Default constructor

Uses Mersenne twister on [0,1) by default and selects integer implementation of Knuth generator by default, using long -> double conversion for uniform on [0,1) instead of direct calculations with floating point

Parameters:

- ***type*** RAN_POL (Lewis), RAN_KNU (Knuth), RAN_MT (Mersenne Twister)

gType RAN_KNU: PRECISE (floating point), FAST (long -> double) FAST is default because PRECISE != WITH_ZERO RAN_MT: OPEN (0,1), WITH_ZERO [0,1), ZERO_ONE [0,1]

Definition at line 236 of file lot.cpp.

References set_generator().

6.10.3.2 lot::∼lot (void)

Destructor

Currently empty. Nothing to clean up

Definition at line 250 of file lot.cpp.

6.10.4 Member Function Documentation

6.10.4.1 void lot::set_generator (int *type*, int *gType*)

Set uniform RNG type

Parameters:

type RAN_POL, RAN_KNU, or RAN_MT

gType PRECISE or FAST (RAN_KNU) DEFAULT, WITH_ZERO, or ZERO_-ONE (RAN_MT)

Definition at line 259 of file lot.cpp.

References MT_genrand(), MT_sgenrand(), PRECISE, RAN_KNU, RAN_MT, RAN_POL, ran_start(), randomize(), ranf_start(), and Set_MT().

Referenced by lot().

6.10.4.2 long lot::seed (void) `[inline]`

First seed from RAN_POL

Definition at line 92 of file lot.h.

6.10.4.3 long lot::init_seed (void) `[inline]`

Second seed from RAN_POL

Definition at line 98 of file lot.h.

6.10.4.4 void lot::randomize (int *spin* = `100`)

Initializes RAN_POL

Parameters:

spin (default 100) passed to dememorize()

Initializes seeds with current system time and calls dememorize to "warm up" random number generator

Definition at line 317 of file lot.cpp.

References dememorize().

Referenced by set_generator().

6.10.4.5 void lot::set_seed (long *s*)

Initialize RNG with known seed

Parameters:

s Seed

Definition at line 328 of file lot.cpp.

References MT_sgenrand(), RAN_KNU, RAN_MT, RAN_POL, ran_start(), and ranf_start().

6.10.4.6 void lot::dememorize (int *spin* = `100`)

Used with RAN_POL to "warm up" generator

Parameters:

spin number of preliminary calls to uniform()

Definition at line 304 of file lot.cpp.

References uniform().

Referenced by randomize().

6.10.4.7 void lot::ran_start (long *seed*)

Initialize the Knuth long RNG

Definition at line 567 of file lot.cpp.

References ran_array(), and t().

Referenced by set_generator(), and set_seed().

6.10.4.8 void lot::ranf_start (long *seed*)

Initialize the Knuth double RNG

Definition at line 443 of file lot.cpp.

References ranf_array(), and t().

Referenced by set_generator(), and set_seed().

6.10.4.9 int lot::random_int (int *maxval*)

Returns a random int in [0,maxval-1]

Parameters:

maxval

Definition at line 878 of file lot.cpp.

References uniform().

6.10.4.10 long lot::random_long (long *maxval* = `Util::int_max`)

Returns a random long in [0,maxval-1]

Parameters:

maxval

Definition at line 863 of file lot.cpp.

References uniform().

6.10.4.11 long lot::ran_knu (void) `[inline]`

Uniform random integer in [0, Util::int_max-1] with RAN_KNU

Definition at line 115 of file lot.h.

6.10.4.12 double lot::uniform (void) `[inline]`

Uniform random number

RAN_POL & RAN_KNU produce uniform on [0,1) RAN_MT produces uniform on [0,1) by default RAN_MT can produce uniform on (0,1) or [0,1]

See also:

Set_MT

Definition at line 126 of file lot.h.

Referenced by dememorize(), expon(), random_int(), and random_long().

6.10.4.13 void lot::ran_array (std::vector< long > & *aa*, int *n*)

Updates long array of Knuth generator

Parameters:

aa the vector of values in which to update

n the number of values to update, note n == aa.size() assumed

Definition at line 534 of file lot.cpp.

Referenced by ran_start().

6.10.4.14 void lot::ranf_array (std::vector< double > & *aa*, int *n*)

Updates floating point array of Knuth generator

Parameters:

aa the vector of values in which to update

n the number of values to update, note n == aa.size() assumed

Definition at line 410 of file lot.cpp.

Referenced by ranf_start().

6.10.4.15 void lot::MT_sgenrand (long *seed*)

Initializes Mersenne twister RNG

Parameters:

seed

This code is translated directly from:

`http://www.math.keio.ac.jp/matumoto/CODES/MT2002/mt19937ar.c`

Here are the accompanying comments

A C-program for MT19937, with initialization improved 2002/1/26. Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed) or init_by_array(init_key, key_length).

Any feedback is very welcome. `http://www.math.keio.ac.jp/matumoto/emt.html` email: `matumoto@math.keio.ac.jp`

Definition at line 725 of file lot.cpp.

Referenced by MT_genrand_int(), MT_init_by_array(), set_generator(), and set_seed().

6.10.4.16 void lot::MT_init_by_array (unsigned long $*$ *init_key*, int *key_length*)

Initialize Mersenne twister with an array

Parameters:

init_key is the array for initializing keys

key_length is its length

Definition at line 745 of file lot.cpp.

References MT_sgenrand().

6.10.4.17 void lot::MT_R_initialize (int *seed*)

Initialize array directly with algorithm from R

Parameters:

seed Probably useful only for testing purposes. I wrote it to allow me to check norm(), binom(), beta(), etc. against R. It only produces the same sequence as R when seed == 1.

Definition at line 787 of file lot.cpp.

6.10.4.18 unsigned long lot::MT_genrand_int (void)

generate random integer on [0,0xffffffff]-interval

Definition at line 800 of file lot.cpp.

References MT_sgenrand().

Referenced by MT_genrand(), MT_genrand_with_zero(), and MT_genrand_with_-zero_one().

6.10.4.19 double lot::MT_genrand (void)

generate random double uniform on (0,1)

Definition at line 839 of file lot.cpp.

References MT_genrand_int().

Referenced by set_generator(), and Set_MT().

6.10.4.20 double lot::MT_genrand_with_zero (void)

generate random double uniform on [0,1)

Definition at line 846 of file lot.cpp.

References MT_genrand_int().

Referenced by Set_MT().

6.10.4.21 double lot::MT_genrand_with_zero_one (void)

generate random double uniform on [0,1]

Definition at line 853 of file lot.cpp.

References MT_genrand_int().

Referenced by Set_MT().

6.10.4.22 bool lot::Set_MT (int *type*)

Set MT type

Parameters:

type DEFAULT (0,1), ZERO [0,1), ZERO_ONE [0,1]

Leaves MT type unchanged if type is not one of DEFAULT, ZERO, or ZERO_ONE. Changes from RAN_POL or RAN_KNU to RAN_MT.

Definition at line 354 of file lot.cpp.

References MT_genrand(), MT_genrand_with_zero(), MT_genrand_with_zero_one(), OPEN, RAN_MT, ZERO, and ZERO_ONE.

Referenced by set_generator().

6.10.4.23 double lot::beta (double *aa*, double *bb*)

Returns a random beta variate

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$$

Parameters:

aa The first beta parameter (a)

bb The second beta parameter (b)

Returns a random variable from a beta distribution with parameters aa and bb.

NOTE: Checks for aa, bb > 0 not included

NOTE: R uses RNGs uniform on [0,1). To ensure consistency with that well tested code make sure that you have Set_MT(ZERO), or the equivalent, if you are using the Mersenne-Twister. ZERO is the default.

This implementation is derived from R v1.9.0

Definition at line 938 of file lot.cpp.

References gamma(), t(), and unif_rand.

Referenced by proposeBeta().

6.10.4.24 int lot::binom (double *nin*, double *pp*)

Returns a random binomial variate

$$f(x) = \binom{n}{k} p^k (1-p)^{n-k}$$

Parameters:

nin sample size (n)

pp probability of success on each trial (p)

$$\mathrm{E}(x) = np$$
$$\mathrm{Var}(x) = np(1-p)$$

This implementation is derived from R v1.9.0.

NOTE: Checks for finite nin and pp not included. Check for nin == floor(n+0.5) not included

Definition at line 1070 of file lot.cpp.

References f(), repeat, and unif_rand.

Referenced by multinom().

6.10.4.25 double lot::cauchy (double *l*, double *s*)

Returns a Cauchy variate

$$f(x) = \frac{1}{\pi \mathrm{s}(1 + (\frac{x-\mathrm{l}}{\mathrm{s}})^2)}$$

Parameters:

- ***l*** the location parameter
- ***s*** the scale parameter

The expectation and variance of the Cauchy distribution are infinite. The mode is equal to the location parameter.

Modified from R v2.0. Does not check isnan() on arguments. Does not check that arguments are finite

Definition at line 1252 of file lot.cpp.

References unif_rand.

6.10.4.26 double lot::chisq (double *n*)

Returns a chi-squared variate

$$f(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$$

Parameters:

- ***n*** degrees of freedom for the chi-squared density

$$\mathrm{E}(x) = \mathrm{n}$$

$$\mathrm{Var}(x) = 2\mathrm{n}$$

Derived from R v2.0. Does not check isfinite() on argument.

Definition at line 1298 of file lot.cpp.

References gamma().

Referenced by f(), and t().

6.10.4.27 std::vector< double > lot::dirichlet (std::vector< double > *c*)

Returns a vector of Dirichlet variates

$$f(\mathbf{x}) = \frac{\Gamma(\sum_i x_i)}{\prod_i \Gamma(x_i)} \prod_i x_i^{c_i}$$

Parameters:

- ***c*** [vector<double>] parameters of the Dirichlet

Definition at line 1310 of file lot.cpp.

References gamma().

Referenced by proposeDirch().

6.10.4.28 double lot::expon (void)

Returns a random value from an exponential distribution with mean 1.

$$f(x) = e^{-1}$$

$$\mathrm{E}(x) = 1$$
$$\mathrm{Var}(x) = 1$$

originally derived from R v1.8.1

NOTE: R uses RNGs uniform on [0,1). To ensure consistency with that well tested code make sure that you have Set_MT(ZERO), or the equivalent, if you are using the Mersenne-Twister. ZERO is the default.

Definition at line 1384 of file lot.cpp.

References uniform().

Referenced by exponential().

6.10.4.29 double lot::exponential (const double *lambda*) `[inline]`

Exponential random deviate

Parameters:

lambda The exponential parameter (λ)

$$f(x) = \lambda e^{-\lambda x}$$

$$\mu = \frac{1}{\lambda}$$
$$\sigma^2 = \left(\frac{1}{\lambda}\right)^2$$

Calls expon() to do the real work

Definition at line 162 of file lot.h.

References expon().

6.10.4.30 double lot::f (double *m*, double *n*)

Returns an F variate

$$f(x) = \frac{\Gamma((m+n)/2)}{\Gamma(m/2)\Gamma(n/2)}(m/n)^{m/2}x^{m/2-1}(1+(m/n)x)^{-(m+n)/2}$$

Parameters:

m “numerator” degrees of freedom

n “denominator” degrees of freedom

$$\mathrm{E}(x) = \frac{m}{m-2}, m > 2$$

$$\mathrm{Var}(x) = \frac{2m^2(n-2)}{n(m+2)}, n > 2$$

Derived from R v2.0. Does not check arguments for isnan() or isfinite().

Definition at line 1480 of file lot.cpp.

References chisq().

Referenced by binom(), and hypergeom().

6.10.4.31 double lot::gamma (double *a*, double *scale* = `1.0`)

Gamma random deviate

Parameters:

a Shape

scale Scale (σ, defaults to 1.0)

$$f(x) = \frac{1}{\sigma^a\Gamma(a)}x^{a-1}e^{-x/\sigma}$$

$$\mu = a\sigma$$

$$\sigma^2 = a\sigma^2$$

NOTE: Checks for finite scale and shape parameters not included

NOTE: R uses RNGs uniform on [0,1). To ensure consistency with that well tested code make sure that you have Set_MT(ZERO), or the equivalent, if you are using the Mersenne-Twister. ZERO is the default.

Derived from R v1.9.0

Definition at line 1551 of file lot.cpp.

References exp_rand, norm_rand, repeat, t(), and unif_rand.

Referenced by beta(), chisq(), dirichlet(), igamma(), and nbinom().

6.10.4.32 double lot::geom (double *p*)

Returns a geometric deviate

$$f(x) = p(1-p)^x$$

Parameters:

p the parameter of the geometric distribution

$$\mathrm{E}(x) = \frac{1-p}{p}$$

$$\mathrm{Var}(x) = \frac{1-p}{p^2}$$

Derived from R v2.0. Does not check isnan() on x and p. Does not check for 0 < p < 1.

Definition at line 1754 of file lot.cpp.

References exp_rand, and poisson().

6.10.4.33 int lot::hypergeom (int *nn1*, int *nn2*, int *kk*)

Returns a hypergeometric variate

$$f(x) = \frac{\binom{w}{x}\binom{b}{n-x}}{\binom{w+b}{n}}$$

Parameters:

nn1 The number of white balls in the urn (w)

nn2 The number of black balls in the urn (b)

kk The sample size (n)

$$\mathrm{E}(x) = n(\frac{w}{w+b})$$

$$\mathrm{Var}(x) = \frac{n(\frac{w}{w+b})(1-\frac{w}{w+b})((w+b)-n)}{w+b-1}$$

The code is modified from R v2.0 to take unsigned integer arguments rather than doubles. isfinite() checks are no longer needed. Check for n < r + b not done.

Definition at line 1812 of file lot.cpp.

References Density::dt(), f(), t(), and unif_rand.

6.10.4.34 double lot::igamma (const double *a*, const double *s* = `1.0`) `[inline]`

Inverse gamma random deviate

$$f(1/x) = \frac{1}{s^a\Gamma(a)}x^{a-1}e^{-x/s}$$

or equivalently

$$f(y) = \frac{\lambda^a(1/y)^{a+1}e^{-\lambda/y}}{\Gamma(a)}$$

$$\lambda = 1/s$$

Parameters:

a Shape (a)

s Scale (s)

$$\mathrm{E}(x) = \frac{\lambda}{a-1} \quad , \quad a > 1$$

$$\mathrm{Var}(x) = \frac{\lambda^2}{(a-1)^2(a-2)} \quad , \quad a > 2$$

Definition at line 184 of file lot.h.

References gamma().

6.10.4.35 double lot::lnorm (double *logmean*, double *logsd*)

Returns a log-normal deviate

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

Parameters:

logmean logarithm of the mean of the corresponding normal (μ)

logsd logarithm of the sd of the corresponding normal (σ)

$$\begin{aligned}\mathrm{E}(x) &= e^{\mu+\sigma^2/2} \\ \mathrm{Var}(x) &= e^{2\mu+\sigma^2}(e^{\sigma^2}-1) \\ \text{mode} &= \frac{e^{\mu}}{e^{\sigma^2}} \\ \text{median} &= e^{\mu}\end{aligned}$$

Derived from R v2.0. Does not do isnan() checks on arguments. Does not check for sigma > 0.

Definition at line 2093 of file lot.cpp.

References norm().

6.10.4.36 double lot::logis (double *location*, double *scale*)

Returns a logistic variate

$$f(x) = \frac{1}{s}\frac{e^{\frac{x-m}{s}}}{(1+e^{\frac{x-m}{s}})^2}$$

or equivalently (dividing numerator and denominator by $e^{2\frac{x-m}{s}}$)

$$f(x) = \frac{1}{s}\frac{e^{\frac{-(x-m)}{s}}}{(1+e^{\frac{-(x-m)}{s}})^2}$$

Parameters:

location the location parameter (m)

scale the scale parameter (s)

$$\mathrm{E}(x) = m$$

$$\mathrm{Var}(x) = \frac{\pi^2 s^2}{3}$$

Derived from R v2.0. Does not do isnan() checks on parameters. Does not check for scale > 0.

Definition at line 2133 of file lot.cpp.

6.10.4.37 std::vector< int > lot::multinom (unsigned *n*, const std::vector< double > & *p*)

$$f(\mathbf{n}) = \binom{\sum_i n_i}{n_1 \ldots n_I} \prod_i p_i^{n_i}$$

Parameters:

n Sample size ($\sum_i n_i$)

p Vector of probabilities (p_i)

Derived from R v2.0. Safety checks eliminated. User must ensure that $\sum_i p_i = 1$ and $p_i >= 0$. $n > 0$ guaranteed, because n is unsigned.

Definition at line 2183 of file lot.cpp.

References binom().

6.10.4.38 double lot::nbinom (double *n*, double *p*)

Returns a negative binomial variate

$$f(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

Parameters:

n the “size” parameter

p the “probability” parameter

$$\mathrm{E}(x) = \frac{x(1-p)}{p}$$

$$\mathrm{Var}(x) = \frac{x(1-p)}{p^2}$$

Derived from R v2.0. Does not check isfinite() on arguments or ensure that p is in [0, 1).

Definition at line 2270 of file lot.cpp.

References gamma(), and poisson().

6.10.4.39 int lot::poisson (double *mu*)

Returns Poisson deviate

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Parameters:

mu mean of the Poisson distribution (λ)

$$\mathrm{E}(x) = \mu$$

$$\mathrm{Var}(x) = \mu$$

derived from R v1.9.0

NOTE: Check for finite mu not included

NOTE: R uses RNGs uniform on [0,1). To ensure consistency with that well tested code make sure that you have Set_MT(ZERO), or the equivalent, if you are using the Mersenne-Twister. ZERO is the default.

Definition at line 2324 of file lot.cpp.

References exp_rand, norm_rand, repeat, t(), and unif_rand.

Referenced by geom(), and nbinom().

6.10.4.40 double lot::norm (const double *mu*, const double *sd*) `[inline]`

Normal random deviate

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

Parameters:

mu Mean (μ)

sd Standard deviation (σ)

$$\mathrm{E}(x) = \mu$$
$$\mathrm{Var}(x) = \sigma^2$$

Calls snorm() to do the real work.

Definition at line 204 of file lot.h.

References snorm().

Referenced by lnorm(), and proposeNorm().

6.10.4.41 double lot::snorm (void)

Returns standard normal deviate

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

Kinderman-Ramage standard normal generator from R v1.9.0

$$\mathrm{E}(x) = 0$$
$$\mathrm{Var}(x) = 1$$

NOTE: R uses RNGs uniform on [0,1). To ensure consistency with that well tested code make sure that you have Set_MT(ZERO), or the equivalent, if you are using the Mersenne-Twister. ZERO is the default.

Definition at line 2506 of file lot.cpp.

References repeat, and unif_rand.

Referenced by norm().

6.10.4.42 double lot::t (double *df*)

Returns a t variate

$$f(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu}\Gamma(\frac{\nu}{2})(1+\frac{x^2}{\nu})^{(\nu+1)/2}}$$

Parameters:

df the degrees of freedom (ν)

$$\mathrm{E}(x) = 0 \quad , \quad \nu > 1$$
$$\mathrm{Var}(x) = \frac{\nu}{\nu - 2} \quad , \quad \nu > 2$$

Derived from R v2.0. Does not check isnan() or isfinite() on argument.

Definition at line 2594 of file lot.cpp.

References chisq(), and norm_rand.

Referenced by beta(), gamma(), hypergeom(), poisson(), ran_start(), and ranf_start().

6.10.4.43 double lot::weibull (double *shape*, double *scale*)

Returns a Weibull variate

$$f(x) = (\frac{a}{b})(\frac{x}{b})^{a-1}e^{-\frac{x}{b}^a}$$

Parameters:

shape the "shape" parameter (a)

scale the "scale" parameter (b)

$$\mathrm{E}(x) = b\Gamma(1 + \frac{1}{a})$$
$$\mathrm{Var}(x) = b^2(\Gamma(1 + \frac{2}{a}) - \Gamma(1 + \frac{1}{a})^2)$$

Derived from R v2.0. Does not check isnan() on arguments. Does not check for $a > 0$ and $b > 0$.

Definition at line 2638 of file lot.cpp.

References unif_rand.

6.10.4.44 double lot::get_ran_u_0 (void) const `[inline]`

First element of internal array in RAN_KNU

Public only to allow testing of PRECISE

Definition at line 215 of file lot.h.

The documentation for this class was generated from the following files:

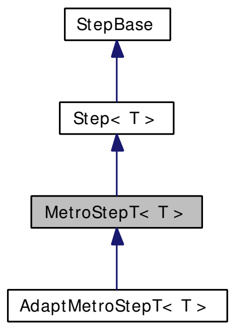
- lot.h
- lot.cpp

6.11 MetroStepT< T > Class Template Reference

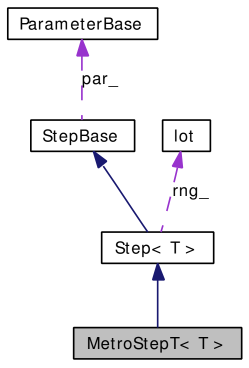
Implements Metropolis-Hastings step for a parameter.

```
#include <MCMC.h>
```

Inheritance diagram for MetroStepT< T >:



Collaboration diagram for MetroStepT< T >:



Public Member Functions

- MetroStepT (ParameterT< T > ∗parameter)

Protected Member Functions

- bool Accept (T &newX, T &oldX)
- void Assign (const T &x)
- virtual void DoStep (void)

Protected Attributes

- ParameterT< T > * met_

  pointer to parameter associated with this step

6.11.1 Detailed Description

template<typename T> class MetroStepT< T >

Implements Metropolis-Hastings step for a parameter.

Definition at line 303 of file MCMC.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 template<typename T> MetroStepT< T >::MetroStepT (ParameterT< T > * *parameter*) `[inline, explicit]`

Constructor

Parameters:

parameter Pointer to the parameter associated with this step

Definition at line 309 of file MCMC.h.

6.11.3 Member Function Documentation

6.11.3.1 template<typename T> bool MetroStepT< T >::Accept (T & *newX*, T & *oldX*) `[inline, protected]`

Accept or reject

Parameters:

newX Proposed value

oldX Old value

Returns:

true for accept
false for reject

Definition at line 324 of file MCMC.h.

References MetroStepT< T >::met_.

Referenced by AdaptMetroStepT< T >::DoStep(), and MetroStepT< T >::DoStep().

6.11.3.2 template<typename T> void MetroStepT< T >::Assign (const T & *x*) `[inline, protected]`

Assign value to parameter associated with this step

Parameters:

x The value to assign

Definition at line 342 of file MCMC.h.

References MetroStepT< T >::met_.

6.11.3.3 template<typename T> virtual void MetroStepT< T >::DoStep (void) `[inline, protected, virtual]`

Get a new value from an M-H step

Implements StepBase.

Reimplemented in AdaptMetroStepT< T >.

Definition at line 348 of file MCMC.h.

References MetroStepT< T >::Accept(), and MetroStepT< T >::met_.

The documentation for this class was generated from the following file:

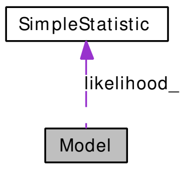
- MCMC.h

6.12 Model Class Reference

Implements the statistical model.

```
#include <MCMC.h>
```

Collaboration diagram for Model:



Public Member Functions

- virtual ∼Model (void)
- void Simulation (std::ostream &outf, bool interimReport)
- virtual void Summarize (int i, std::ostream &outf)
- std::string Label (int i) const
- void ReportDic (std::ostream &outf)
- virtual void Record (const SampleVector &p)
- virtual void InterimReport (std::ostream &outf, std::string header, int progress, int goal)
- virtual void ReportHead (std::ostream &outf)
- virtual void Report (std::ostream &outf, const int lastPar=-1)
- virtual double Llike (const SampleVector &par) const
- void SetBurnIn (const int nBurnin)
- void SetSample (const int nSample)
- void SetThin (const int thin)

Protected Member Functions

- Model (int nBurnin, int nSample, int nThin, bool calc=false, bool useMedian=false)
- double percent (int k, int n) const
- double percent (const std::vector< int > &x, int n) const
- double percent (const std::vector< std::vector< int > > &x, int n) const
- virtual SampleVector Parameters (void) const

Protected Attributes

- int nBurnin_

  number of iterations for burn in

- int nSample_

  number of iterations for sample

- int nThin_

  number of iterations between saving results

- unsigned nElem_

  number of parameters in the model

- SimpleStatistic likelihood_

  stores likelihood for DIC calculations

- ModelSteps step_

  vector of parameters to sample

- Results results_

  vector (boost::any) of stored results

- boost::format ∗ summaryFormat_

  boost::format for summary output

6.12.1 Detailed Description

Implements the statistical model.

To build a model, derive a new class from Model. In its constructor, push Steps of the appropriate type onto step_. If you're happy with the default reports, the only other thing you'll need to do is to provide accessor functions to values of the parameters in step_ (or make them public so that Parameters can access them directly).

If you want to calculate DIC for the model, you'll have to provide an appropriate override for Llike(), but everything else will be taken care of automatically.

Definition at line 613 of file MCMC.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Model::∼Model (void) `[virtual]`

Destructor

Definition at line 737 of file MCMC.cpp.

6.12.2.2 Model::Model (int *nBurnin*, int *nSample*, int *nThin*, bool *calc* = `false`, bool *useMedian* = `false`) `[protected]`

Constructor

Parameters:

nBurnin Number of iterations for "burn in"

nSample Number of iterations for "sample"

nThin Keep every nThin'th sample

calc false == no lLike not overridden, true == lLike overriden (there's probably a way to figure this out, but I haven't worked on it yet). Default is false

useMedian false == use posterior mean in DIC calculation, true == use posterior median in DIC calculation. Default is false.

Definition at line 728 of file MCMC.cpp.

6.12.3 Member Function Documentation

6.12.3.1 void Model::Simulation (std::ostream & *outf*, bool *interimReport*)

Invoke simulation to do the analysis

Parameters:

outf Output stream for progress display

interimReport Produce progress display?

Definition at line 745 of file MCMC.cpp.

References InterimReport(), nBurnin_, nElem_, nSample_, nThin_, Parameters(), Record(), and step_.

6.12.3.2 void Model::Summarize (int *i*, std::ostream & *outf*) `[virtual]`

Produce the summary statistics for each parameter

Parameters:

- ***i*** Index of the parameter being reported on
- ***outf*** The stream on which the report is being written

Definition at line 867 of file MCMC.cpp.

References Label(), SimpleStatistic::Mean(), nSample_, nThin_, quantile(), results_, SimpleStatistic::StdDev(), and summaryFormat_.

Referenced by Report().

6.12.3.3 std::string Model::Label (int *i*) const

Produce the parameter label

Parameters:

- ***i*** Index of the parameter

Definition at line 884 of file MCMC.cpp.

References step_.

Referenced by Summarize().

6.12.3.4 void Model::ReportDic (std::ostream & *outf*)

Calculate and report DIC statistics

Definition at line 901 of file MCMC.cpp.

References likelihood_, Llike(), SimpleStatistic::Mean(), nElem_, quantile(), results_, and SimpleStatistic::Variance().

Referenced by Report().

6.12.3.5 void Model::Record (const SampleVector & *p*) `[virtual]`

Keep track of current parameter values

Parameters:

- ***p*** Current parameter values

If you're analysing a big model with lots of parameters, only some of which are of primary interest, you may want to override Record() to either keep track only of the few parameters that are of primary interest, either discarding the others or writing them to disk.

Definition at line 775 of file MCMC.cpp.

References SimpleStatistic::Add(), likelihood_, Llike(), and results_.

Referenced by Simulation().

6.12.3.6 void Model::InterimReport (std::ostream & *outf*, std::string *header*, int *progress*, int *goal*) `[virtual]`

Produce a progress display

If you're writing a sampler under a GUI and want a progress display, you'll definitely want to override this.

Definition at line 788 of file MCMC.cpp.

Referenced by Simulation().

6.12.3.7 void Model::ReportHead (std::ostream & *outf*) `[virtual]`

Produce the report header

Parameters:

outf The stream on which the report is being written

Definition at line 852 of file MCMC.cpp.

References summaryFormat_.

Referenced by Report().

6.12.3.8 void Model::Report (std::ostream & *outf*, const int *lastPar* = `-1`) `[virtual]`

Produce a final report

Parameters:

outf The stream on which the report is to be written

lastPar -1 = all parameters, k+1 = k parameters

The report includes a header, the label for each parameter, and the posterior mean, standard deviation, 2.5%, 50%, and 97.5tiles.

Definition at line 811 of file MCMC.cpp.

References ReportDic(), ReportHead(), results_, and Summarize().

6.12.3.9 double Model::Llike (const SampleVector & *par*) const `[virtual]`

Likelihood

Parameters:

par A vector of parameters from which to calculate the likelihood

Definition at line 894 of file MCMC.cpp.

Referenced by Record(), and ReportDic().

6.12.3.10 void Model::SetBurnIn (const int *nBurnin*)

Set the number of iterations in the burn-in period

Parameters:

nBurnin the number of burn-in iterations

Definition at line 825 of file MCMC.cpp.

References nBurnin_.

6.12.3.11 void Model::SetSample (const int *nSample*)

Set the number of iterations in the sample period

Parameters:

nSample the number of sample iterations

Definition at line 834 of file MCMC.cpp.

References nSample_.

6.12.3.12 void Model::SetThin (const int *thin*)

Set the thinning interval (the interval between saved samples)

thin the thinning interval

Definition at line 843 of file MCMC.cpp.

References nThin_.

6.12.3.13 double Model::percent (int *k*, int *n*) const `[protected]`

percent – integer/integer

Parameters:

- ***k*** Count
- ***n*** Sample size

Definition at line 959 of file MCMC.cpp.

Referenced by percent().

6.12.3.14 double Model::percent (const std::vector< int > & *x*, int *n*) const `[protected]`

percent – average percent across a vector

Parameters:

- ***x*** Vector of counts
- ***n*** Sample size

Definition at line 969 of file MCMC.cpp.

References SimpleStatistic::Add(), SimpleStatistic::Mean(), and percent().

6.12.3.15 double Model::percent (const std::vector< std::vector< int > > & *x*, int *n*) const `[protected]`

percent – average percent across a vector of vectors

Parameters:

- ***x*** Vector of vector of counts
- ***n*** Sample size

Definition at line 984 of file MCMC.cpp.

References SimpleStatistic::Add(), SimpleStatistic::Mean(), and percent().

6.12.3.16 SampleVector Model::Parameters (void) const `[protected, virtual]`

Return parameter vector associated with this step

Values in the sample vector are stored as boost::any. Retrieving them will require an appropriate boost::any_cast<>

Definition at line 1020 of file MCMC.cpp.

References step_.

Referenced by Simulation().

The documentation for this class was generated from the following files:

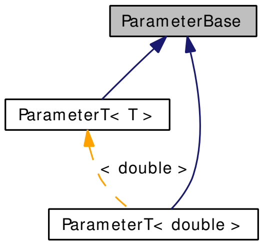
- MCMC.h
- MCMC.cpp

6.13 ParameterBase Class Reference

Pure virtual class provide interface for Parameter.

```
#include <MCMC.h>
```

Inheritance diagram for ParameterBase:



Public Member Functions

- ParameterBase (std::string label)
- virtual ∼ParameterBase (void)
- virtual void Adapt (const double accept)
- void Assign (const boost::any &value)
- std::string Label (void) const
- void SetLabel (const std::string &label)
- const boost::any Value (void) const

6.13.1 Detailed Description

Pure virtual class provide interface for Parameter.

Definition at line 84 of file MCMC.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 ParameterBase::ParameterBase (std::string *label*)

Constructor

Parameters:

label a string identifier for the parameter

Used internally to construct Parameter classes. Not intended for direct use.

Definition at line 309 of file MCMC.cpp.

6.13.2.2 ParameterBase::∼ParameterBase (void) `[virtual]`

Destructor

Definition at line 315 of file MCMC.cpp.

6.13.3 Member Function Documentation

6.13.3.1 void ParameterBase::Adapt (const double *accept*) `[virtual]`

Adapt the proposal distribution in a Metropolis-Hastings sampler

Parameters:

accept the current acceptance percentage

Definition at line 323 of file MCMC.cpp.

6.13.3.2 void ParameterBase::Assign (const boost::any & *value*)

sets value of the parameter

Parameters:

value the value to be assigned

Definition at line 351 of file MCMC.cpp.

6.13.3.3 std::string ParameterBase::Label (void) const

Returns string identifying current parameter

The default value is an empty string. Replace it by including an argument to the constructor

Definition at line 376 of file MCMC.cpp.

Referenced by StepBase::Label().

6.13.3.4 void ParameterBase::SetLabel (const std::string & *label*)

Set the label associated with this parameter

Parameters:

label the label string

Definition at line 385 of file MCMC.cpp.

Referenced by StepBase::SetLabel().

6.13.3.5 const boost::any ParameterBase::Value (void) const

Value of the parameter

Reimplemented in ParameterT< T >, and ParameterT< double >.

Definition at line 413 of file MCMC.cpp.

Referenced by StepBase::aValue().

The documentation for this class was generated from the following files:

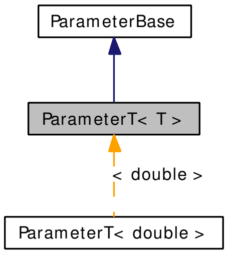
- MCMC.h
- MCMC.cpp

6.14 ParameterT< T > Class Template Reference

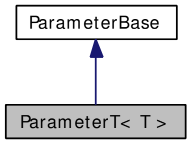
Base class for model parameters.

```
#include <MCMC.h>
```

Inheritance diagram for ParameterT< T >:



Collaboration diagram for ParameterT< T >:



Public Member Functions

- ParameterT (std::string label)
- virtual ∼ParameterT (void)
- virtual double llike (const T x) const
- virtual double lPrior (const T x) const
- virtual T propose (const T current) const
- virtual double lQ (const T x, const T y) const
- void Assign (const T &x)
- virtual const T Function (const bool doCalc=true) const
- virtual const T Value (void) const

6.14.1 Detailed Description

template<typename T> class ParameterT< T >

Base class for model parameters.

ParameterT is one of the three workhorses in the simulation framework. Each parameter in the statistical model must be derived separately from ParameterT. It is not necessary to declare and define separate classes for parameters that share the same probabilistic structure, e.g., allele counts in different populations or measurements of yields in different experimental blocks, but each parameter must be pushed separately onto the step_ stack in Model.

It is often useful for the derived class to store a pointer to the model of which it is part. This allows the Model class to define access functions to the values of other parameters, facilitating calculation of the full conditionals.

lPrior() and llike(), the log prior and log likelihood for a particular parameter respectively, are used only as a sum. Thus, if it is more convenient to write the full conditional in a single function, either may be used. I typically find it easiest to write lPrior() as expressing the "probability" of the parameter, given hyperparameters on which it depends and llike() as expressing the "probability" of parameters (or data that depend on this one. But your mileage may vary.

Definition at line 125 of file MCMC.h.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 template<typename T> ParameterT< T >::ParameterT (std::string *label*) `[inline]`

Constructor

Parameters:

label a string identifier for the parameter

Definition at line 131 of file MCMC.h.

6.14.2.2 template<typename T> virtual ParameterT< T >::~ParameterT (void) `[inline, virtual]`

Destructor

Definition at line 137 of file MCMC.h.

6.14.3 Member Function Documentation

6.14.3.1 template<typename T> virtual double ParameterT< T >::llike (const T *x*) const `[inline, virtual]`

Returns likelihood associated with current parameter

Parameters:

x Value of the parameter in the current iteration

Definition at line 148 of file MCMC.h.

6.14.3.2 template<typename T> virtual double ParameterT< T >::lPrior (const T *x*) const `[inline, virtual]`

Returns prior associated with current parameter

Parameters:

x Value of the parameter in the current iteration

Definition at line 156 of file MCMC.h.

6.14.3.3 template<typename T> virtual T ParameterT< T >::propose (const T *current*) const `[inline, virtual]`

Propose a new value for a parameter in an M-H step

Parameters:

current Value of the parameter in the current iteration

Definition at line 164 of file MCMC.h.

6.14.3.4 template<typename T> virtual double ParameterT< T >::lQ (const T *x*, const T *y*) const `[inline, virtual]`

Probability of proposing x, given starting from y

Parameters:

x

y

Definition at line 173 of file MCMC.h.

6.14.3.5 template<typename T> void ParameterT< T >::Assign (const T & *x*) `[inline]`

Assign value

Parameters:

x Value to assign to this parameter

Definition at line 180 of file MCMC.h.

Referenced by SliceStep::DoStep().

6.14.3.6 template<typename T> virtual const T ParameterT< T >::Function (const bool *doCalc* = `true`) const `[inline, virtual]`

Function used to update a deterministic node

Definition at line 186 of file MCMC.h.

6.14.3.7 template<typename T> virtual const T ParameterT< T >::Value (void) const `[inline, virtual]`

Value of the parameter

Reimplemented from ParameterBase.

Definition at line 192 of file MCMC.h.

Referenced by SliceStep::DoStep(), StepBase::dVecValue(), StepBase::iValue(), StepBase::iVecValue(), and StepBase::Value().

The documentation for this class was generated from the following file:

- MCMC.h

6.15 Util::PrintForVector< T > Class Template Reference

`#include <util.h>`

Public Member Functions

- PrintForVector (std::ostream &os)
- void operator() (const T &x)

6.15.1 Detailed Description

template<class T> class Util::PrintForVector< T >

This template provides a helper class for the template function below for std::vector<T> operator <<

Definition at line 142 of file util.h.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 template<class T> Util::PrintForVector< T >::PrintForVector (std::ostream & *os*) `[inline, explicit]`

Constructor

Definition at line 146 of file util.h.

6.15.3 Member Function Documentation

6.15.3.1 template<class T> void Util::PrintForVector< T >::operator() (const T & *x*) `[inline]`

allows for_each to provide appropriate output for vector elements

Definition at line 151 of file util.h.

The documentation for this class was generated from the following file:

- util.h

6.16 ratio Class Reference

Provides a ratio class.

Public Member Functions

- ratio (void)
- ratio (const ratio &r)
- ratio & operator+= (const ratio &r)
- ratio & operator+= (const double d)
- ratio & operator/= (const ratio &r)
- ratio & operator/= (double d)
- ratio & operator= (const ratio &r)
- bool operator== (const ratio &r) const
- bool operator!= (const ratio &r) const
- double make_double (void) const
- double Top (void) const
- double Bottom (void) const

6.16.1 Detailed Description

Provides a ratio class.

The ratio class allows numerators and denominators to be summed separately. Access to each is provided.

Definition at line 34 of file ratio.h.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 ratio::ratio (void)

Default constructor.

Initializes private variables

Definition at line 42 of file ratio.cpp.

6.16.2.2 ratio::ratio (const ratio & *r*)

Copy constructor.

Initializes private variables from existing ratio

Definition at line 50 of file ratio.cpp.

6.16.3 Member Function Documentation

6.16.3.1 ratio & ratio::operator+= (const ratio & *r*)

Add ratio to current ratio.

Parameters:

r The ratio to add

Definition at line 59 of file ratio.cpp.

References bottom_, and top_.

6.16.3.2 ratio & ratio::operator+= (const double *d*)

Add double to current ratio.

Adds the d to both numerator and denominator

Parameters:

d The double value to add

Definition at line 72 of file ratio.cpp.

6.16.3.3 ratio & ratio::operator/= (const ratio & *r*)

Divide one ratio by another.

Numerator divided by numerator. Denominator divided by denominator.

Parameters:

r The ratio to be used in the "denominator" of the division

Definition at line 85 of file ratio.cpp.

References bottom_, and top_.

6.16.3.4 ratio & ratio::operator/= (double *d*)

Divide a ratio by a double.

Numerator and denominator both divided by d.

Parameters:

d

Definition at line 98 of file ratio.cpp.

6.16.3.5 ratio & ratio::operator= (const ratio & *r*)

Assignment operator.

Parameters:

r The value being assigned

Definition at line 109 of file ratio.cpp.

References bottom_, and top_.

6.16.3.6 bool ratio::operator== (const ratio & *r*) const

Equality test.

Equal if and only if top_ and bottom_ of both ratios are equal.

Parameters:

r The value being compared

Definition at line 122 of file ratio.cpp.

References bottom_, and top_.

6.16.3.7 bool ratio::operator!= (const ratio & *r*) const

Inequality test.

Not equal if top_s or bottom_s are unequal

Parameters:

r The value being compared

Definition at line 133 of file ratio.cpp.

References bottom_, and top_.

6.16.3.8 double ratio::make_double (void) const

Value of ratio

Definition at line 140 of file ratio.cpp.

References Util::long_max, and Util::long_min.

6.16.3.9 double ratio::Top (void) const

Numerator of ratio

Definition at line 155 of file ratio.cpp.

Referenced by Statistic::Add().

6.16.3.10 double ratio::Bottom (void) const

Denominator of ratio

Definition at line 162 of file ratio.cpp.

Referenced by Statistic::Add().

The documentation for this class was generated from the following files:

- ratio.h
- ratio.cpp

6.17 SimpleStatistic Class Reference

Implements a class for summary statistics.

Public Member Functions

- SimpleStatistic (void)
- template<class T> SimpleStatistic (std::vector< T > &x)
- void Add (double x)
- double Mean (void) const
- double Variance (void) const
- double StdDev (void) const
- void Clear (void)

6.17.1 Detailed Description

Implements a class for summary statistics.

The SimpleStatistic class allows easy calculation of simple summary statistics. Unlike Statistic, it cannot be applied to ratio data. In addition to adding single values, with Add(), an entire vector of values can be added in the constructor.

Example:

```
SimpleStatistic stat;
stat.Add(1.0);
stat.Add(2.0);
stat.Add(3.0);
stat.Add(4.0);
stat.Add(5.0);
double mean   = stat.Mean();   // mean = 3.000
double variance  = stat.Variance(); // variance = 2.500
double stddev  = stat.StdDev();  // stddev = 1.581
stat.Clear();  // clears internal data for new calculation

vector<double> x(5);
for (int i = 0; i < 5; ++i) {
  x[i] = i;
}
SimpleStatistic vecStat(x);
double mean   = stat.Mean();   // mean = 3.000
double variance  = stat.Variance(); // variance = 2.500
double stddev  = stat.StdDev();  // stddev = 1.581
```

Definition at line 178 of file statistics.h.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 SimpleStatistic::SimpleStatistic (void)

Default constructor.

The default constructor simply initializes the internal data. After initializaiton, data is added with Add().

Definition at line 250 of file statistics.cpp.

6.17.2.2 template<class T> SimpleStatistic::SimpleStatistic (std::vector< T > & *x*) `[inline]`

Constructor – initialize with a vector.

May be used with any vector having an iterator that can produce a double. This may be a simple vector<double> (or any other vector whose elements can be converted to double by default conversions, but it could also be a vector<T>, where a method in T is called before this constructor to provide an implementation of operator∗ that returns an appropriate member from T (one that can be converted via default conversions to a double).

Parameters:

x The vector to use in calculations

Definition at line 195 of file statistics.h.

6.17.3 Member Function Documentation

6.17.3.1 void SimpleStatistic::Add (double *x*)

Add a double value to the statistic.

Parameters:

x Value to add

Definition at line 259 of file statistics.cpp.

Referenced by Model::percent(), and Model::Record().

6.17.3.2 double SimpleStatistic::Mean (void) const

Returns mean of the data.

Definition at line 268 of file statistics.cpp.

Referenced by Model::percent(), Model::ReportDic(), and Model::Summarize().

6.17.3.3 double SimpleStatistic::Variance (void) const

Returns variance of the data.

Definition at line 275 of file statistics.cpp.

Referenced by Model::ReportDic(), and StdDev().

6.17.3.4 double SimpleStatistic::StdDev (void) const

Returns standard deviation of the data.

Definition at line 282 of file statistics.cpp.

References Variance().

Referenced by Model::Summarize().

6.17.3.5 void SimpleStatistic::Clear (void)

Re-initializes internal data for new calculations.

Definition at line 289 of file statistics.cpp.

The documentation for this class was generated from the following files:

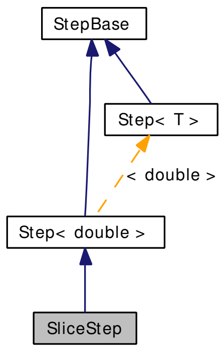
- statistics.h
- statistics.cpp

6.18 SliceStep Class Reference

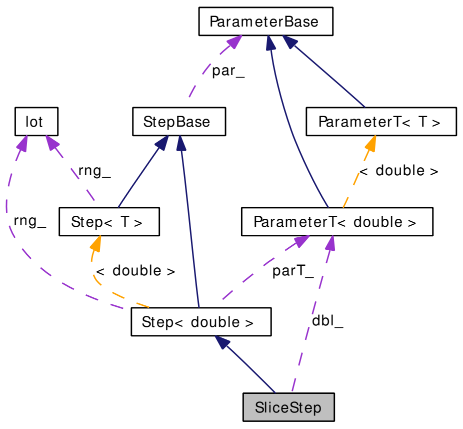
Implements a slice sampler.

`#include <MCMC.h>`

Inheritance diagram for SliceStep:



Collaboration diagram for SliceStep:



Public Member Functions

- SliceStep (ParameterT< double > ∗parameter)
- void SetBounds (double low, double high)
- void SetW (double w)
- void SetM (int m)

Protected Member Functions

- virtual void DoStep (void)

6.18.1 Detailed Description

Implements a slice sampler.

Note: only univariate slices are supported

ParameterT constructors must provide appropriate step width, parameter->W(), and number of tries, parameter->M(), if defaults (1 and MUnbounded, respectively) are to be avoided. lowBound_ and highBound_ defaults must be reset if parameter range is bounded, e.g., frequencies in [0,1].

Definition at line 496 of file MCMC.h.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 SliceStep::SliceStep (ParameterT$<$ double $>$ $*$ *parameter*) `[explicit]`

Constructor

Parameters:

parameter Pointer to the parameter associated with this step

The "step out" width is set to 1, unless the parameter has defined W() and returns a value $>$ 0.0. The "step out" procedure is set to do an unlimited number of steps, unless the parameter has defined M() and returns a value $>$ 0.

Notice that the defaults will result in the slice sampler adapting the "step out" width as the simulation proceeds. Adapting the width is permissible **only** when the distribution is unimodal. If you're not sure that your distribution is unimodal, you should define W() and M() to return values that seem reasonable to you.

Definition at line 553 of file MCMC.cpp.

6.18.3 Member Function Documentation

6.18.3.1 void SliceStep::SetBounds (double *low*, double *high*) `[virtual]`

Sets upper and lower limits on values allowed

Parameters:

low Lower bound on parameter

high Upper bound on parameter

For parameters that are bounded, e.g., frequencies on [0,1], setting bounds on allowable values helps avoid numerical problems in evaluating likelihoods and priors.

Implements StepBase.

Definition at line 573 of file MCMC.cpp.

References Util::dbl_max, StepBase::highBound_, StepBase::lowBound_, and StepBase::w_.

6.18.3.2 void SliceStep::SetW (double *w*) `[virtual]`

Reset step width

Parameters:

w New step width

Implements StepBase.

Definition at line 586 of file MCMC.cpp.

References StepBase::w_.

6.18.3.3 void SliceStep::SetM (int *m*) `[virtual]`

Reset number of steps out allowed

Parameters:

m New number of steps out allowed

Implements StepBase.

Definition at line 595 of file MCMC.cpp.

References StepBase::m_.

6.18.3.4 void SliceStep::DoStep (void) `[protected, virtual]`

Return a new value from the slice sampler

Implements StepBase.

Definition at line 611 of file MCMC.cpp.

References ParameterT< T >::Assign(), and ParameterT< T >::Value().

The documentation for this class was generated from the following files:

- MCMC.h
- MCMC.cpp

6.19 Statistic Class Reference

Implements a class for summary statistics.

Public Member Functions

- Statistic (void)
- void Add (double)
- void Add (ratio)
- long N (void)
- double Sum (void)
- double SumSq (void)
- double Mean (void)
- double Variance (void)
- double StdDev (void)
- double CV (void)
- Statistic & operator+= (double v)
- Statistic & operator+= (ratio r)

Friends

- std::ostream & operator<< (std::ostream &, Statistic &)

6.19.1 Detailed Description

Implements a class for summary statistics.

The Statistic class allows easy calculation of summary statistics. When applied to ratio data the mean is the ratio of the means, and the variance is calculated from the ratio of the sums of squares. Specifically, let

$$s = \frac{\sum_i x_{i,top}}{\sum_i x_{i,bottom}}$$

$$ss = \frac{\sum_i x_{i,top}^2}{\sum_i x_{i,bottom}^2}$$

then Mean() returns

$$s/n$$

and Variance() returns

$$(ss - (s * s)/n)/(n - 1)$$

Example:

```
Statistic stat;
stat.Add(1.0);
stat.Add(2.0);
stat.Add(3.0);
stat.Add(4.0);
stat.Add(5.0);
long  n     = stat.N();     // n = 5
double mean    = stat.Mean();    // mean = 3.000
double variance  = stat.Variance(); // variance = 2.500
double stddev   = stat.StdDev();  // stddev = 1.581
double cv     = stat.CV();    // cv = 0.527
```

Definition at line 129 of file statistics.h.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Statistic::Statistic (void)

Constructor

The constructor for statistic simply initializes all internal values in preparation for calculating statistics. After initializtion, data is added with Add().

Definition at line 74 of file statistics.cpp.

6.19.3 Member Function Documentation

6.19.3.1 void Statistic::Add (double *v*)

Add a double value to the statistic.

Parameters:

v The value to add

Definition at line 152 of file statistics.cpp.

Referenced by operator+=().

6.19.3.2 void Statistic::Add (ratio *r*)

Add a ratio to the statistic.

Parameters:

r The ratio to add

Definition at line 164 of file statistics.cpp.

References ratio::Bottom(), and ratio::Top().

6.19.3.3 long Statistic::N (void) `[inline]`

returns sample size

Definition at line 156 of file statistics.h.

Referenced by operator<<().

6.19.3.4 double Statistic::Sum (void) `[inline]`

returns sum of sample values

Definition at line 161 of file statistics.h.

6.19.3.5 double Statistic::SumSq (void) `[inline]`

returns sum of squared sample values

Definition at line 166 of file statistics.h.

6.19.3.6 double Statistic::Mean (void)

Returns arithmetic mean of the data.

Definition at line 118 of file statistics.cpp.

Referenced by operator<<().

6.19.3.7 double Statistic::Variance (void)

Returns variance of the data.

Definition at line 126 of file statistics.cpp.

Referenced by operator<<().

6.19.3.8 double Statistic::StdDev (void)

Returns standard deviation of the data.

Definition at line 134 of file statistics.cpp.

Referenced by operator<<().

6.19.3.9 double Statistic::CV (void)

Returns coefficient of variation of the data.

Definition at line 142 of file statistics.cpp.

Referenced by operator<<().

6.19.3.10 Statistic & Statistic::operator+= (double *v*)

Add a double value to the statistic

Parameters:

v The value to add

Definition at line 177 of file statistics.cpp.

References Add().

6.19.3.11 Statistic & Statistic::operator+= (ratio *r*)

Add a ratio to the statistic.

Parameters:

r The ratio to add

Definition at line 187 of file statistics.cpp.

References Add().

6.19.4 Friends And Related Function Documentation

6.19.4.1 std::ostream& operator<< (std::ostream & *out*, Statistic & *st*) `[friend]`

Stream output for Statistic.

Reports sample size, mean, variance, standard deviation, and coefficient of variation, each preceded by a tab and appearing on a new line.

Parameters:

out The output stream

st The statistics

Definition at line 202 of file statistics.cpp.

The documentation for this class was generated from the following files:

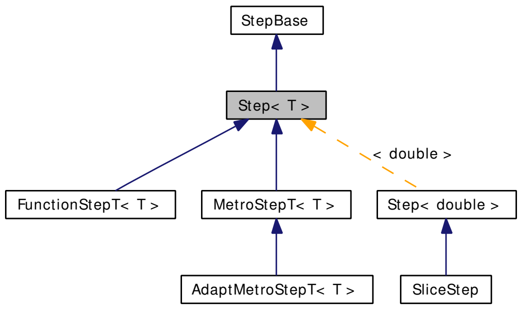
- statistics.h
- statistics.cpp

6.20 Step< T > Class Template Reference

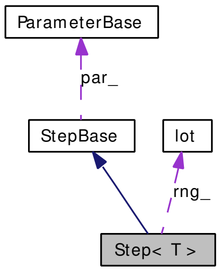
Base class for all steps associated with one parameter type.

`#include <MCMC.h>`

Inheritance diagram for Step< T >:



Collaboration diagram for Step< T >:



Public Member Functions

- Step (ParameterT< T > *parameter)
- virtual ~Step (void)

Protected Member Functions

- Step (ParameterT< T > *parameter, double w, double m, double lowBound, double highBound)

Protected Attributes

- lot & rng_

  shared random number generator

- ParameterT< T > * parT_

  pointer to parameter associated with this step

6.20.1 Detailed Description

template<typename T> class Step< T >

Base class for all steps associated with one parameter type.

Definition at line 265 of file MCMC.h.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 template<typename T> Step< T >::Step (ParameterT< T > * *parameter*) `[inline, explicit]`

Constructor

Parameters:

parameter Pointer to the parameter associated with this step

Definition at line 271 of file MCMC.h.

6.20.2.2 template<typename T> virtual Step< T >::~Step (void) `[inline, virtual]`

Destructor

Definition at line 278 of file MCMC.h.

6.20.2.3 template<typename T> Step< T >::Step (ParameterT< T > * *parameter*, double *w*, double *m*, double *lowBound*, double *highBound*) `[inline, protected]`

Constructor

Parameters:

parameter	Pointer to the parameter associated with this step
w	step width for slice sampler
m	maximum number of steps for slice sampler
lowBound	minimum value of parameter allowed
highBound	maximum value of parameter allowed

Definition at line 289 of file MCMC.h.

The documentation for this class was generated from the following file:

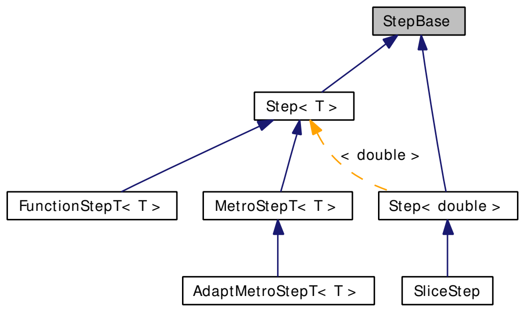
- MCMC.h

6.21 StepBase Class Reference

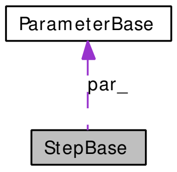
Base class for steps associated with different parameter types.

`#include <MCMC.h>`

Inheritance diagram for StepBase:



Collaboration diagram for StepBase:



Public Member Functions

- StepBase (ParameterBase ∗parameter, unsigned long accept, unsigned long ct, const double w, const double m, const double lowBound, const double highBound)
- virtual ∼StepBase (void)
- virtual void DoStep (void)=0
- virtual void SetBounds (double low, double high)=0
- virtual void SetW (double w)=0
- virtual void SetM (int m)=0
- const boost::any aValue (void) const
- const double Value (void) const
- const int iValue (void) const
- const std::vector< double > dVecValue (void) const
- const std::vector< int > iVecValue (void) const

- int accept (void) const
- std::string Label (void) const
- ParameterBase ∗ Par (void) const
- void ResetAccept (void)
- void SetLabel (const std::string &label)

Protected Attributes

- ParameterBase ∗ par_

  holds the data and methods

- unsigned long accept_

  acceptance count for M-H

- unsigned long ct_

  number of choices so far

- double w_

  slice width

- double m_

  maximum number of steps in slice sampler

- double lowBound_

  smallest value of parameter allowed

- double highBound_

  largest value of parameter allowed

6.21.1 Detailed Description

Base class for steps associated with different parameter types.

Definition at line 205 of file MCMC.h.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 StepBase::StepBase (ParameterBase ∗ *parameter*, unsigned long *accept*, unsigned long *ct*, const double *w*, const double *m*, const double *lowBound*, const double *highBound*)

Constructor

Used internally to construct Step classes. Not intended for direct use.

Definition at line 436 of file MCMC.cpp.

6.21.2.2 StepBase::∼StepBase (void) `[virtual]`

Destructor

Definition at line 445 of file MCMC.cpp.

6.21.3 Member Function Documentation

6.21.3.1 virtual void StepBase::DoStep (void) `[pure virtual]`

Select new parameter from sampler

Implemented in MetroStepT< T >, AdaptMetroStepT< T >, and SliceStep.

6.21.3.2 virtual void StepBase::SetBounds (double *low*, double *high*) `[pure virtual]`

Set bounds on parameter value

Parameters:

low lower bound

high upper bound

Implemented in SliceStep.

6.21.3.3 virtual void StepBase::SetW (double *w*) `[pure virtual]`

Set width of step in slice sampler

Parameters:

w width

Implemented in SliceStep.

6.21.3.4 virtual void StepBase::SetM (int *m*) `[pure virtual]`

Set maximum number of steps in slice sampler

Parameters:

m maximum number of steps allowed

Implemented in SliceStep.

6.21.3.5 const boost::any StepBase::aValue (void) const

Returns parameter value of the current step

Definition at line 490 of file MCMC.cpp.

References par_, and ParameterBase::Value().

6.21.3.6 const double StepBase::Value (void) const

Returns parameter value of the current step

Assumes value stored as double

Definition at line 499 of file MCMC.cpp.

References par_, and ParameterT< T >::Value().

6.21.3.7 const int StepBase::iValue (void) const

Returns parameter value of the current step

Assumes value stored as integer

Definition at line 509 of file MCMC.cpp.

References par_, and ParameterT< T >::Value().

6.21.3.8 const vector< double > StepBase::dVecValue (void) const

Returns parameter value of the current step

Assumes value stored as vector<double>

Definition at line 519 of file MCMC.cpp.

References par_, and ParameterT< T >::Value().

6.21.3.9 const vector< int > StepBase::iVecValue (void) const

Returns parameter value of the current step

Assumes value stored as vector<int>

Definition at line 529 of file MCMC.cpp.

References par_, and ParameterT< T >::Value().

6.21.3.10 int StepBase::accept (void) const

Number of proposals accepted (for M-H)

Definition at line 451 of file MCMC.cpp.

References accept_.

6.21.3.11 std::string StepBase::Label (void) const

Label of parameter associated with this step

Definition at line 458 of file MCMC.cpp.

References ParameterBase::Label(), and par_.

6.21.3.12 ParameterBase ∗ StepBase::Par (void) const

Parameter associated with this step

Definition at line 465 of file MCMC.cpp.

References par_.

6.21.3.13 void StepBase::ResetAccept (void)

Reset acceptance statistics (for M-H)

Definition at line 472 of file MCMC.cpp.

References accept_.

6.21.3.14 void StepBase::SetLabel (const std::string & *label*)

Set the label associated with this step

Parameters:

label the label string

Definition at line 481 of file MCMC.cpp.

References par_, and ParameterBase::SetLabel().

The documentation for this class was generated from the following files:

- MCMC.h
- MCMC.cpp

Chapter 7

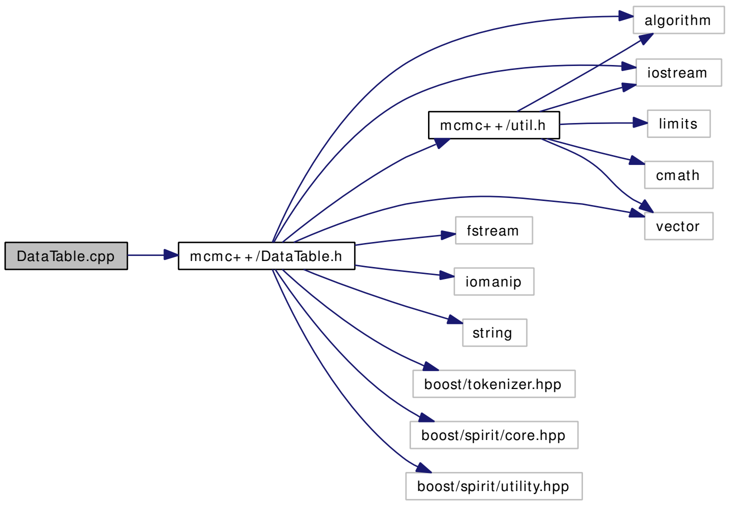
mcmc File Documentation

7.1 DataTable.cpp File Reference

Implementation of stream output for errors from DataTable class.

```
#include "mcmc++/DataTable.h"
```

Include dependency graph for DataTable.cpp:



Functions

- std::ostream & operator<< (std::ostream &out, enum DataTableResult result)

7.1.1 Detailed Description

Implementation of stream output for errors from DataTable class.

Author:

Kent Holsinger

Date:

2005-05-18

Definition in file DataTable.cpp.

7.1.2 Function Documentation

7.1.2.1 std::ostream& operator<< (std::ostream & *out*, enum DataTableResult *result*)

Stream output for DataTable errors

Parameters:

out The stream for output

result The error identifier

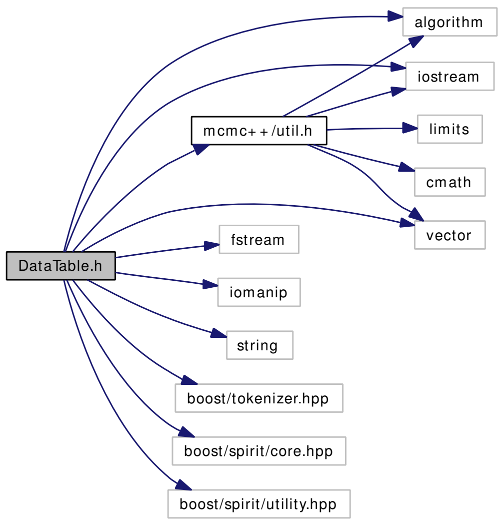
Definition at line 35 of file DataTable.cpp.

7.2 DataTable.h File Reference

Provides DataTable class for access to tabular data.

```
#include <algorithm>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>
#include <boost/tokenizer.hpp>
#include <boost/spirit/core.hpp>
#include <boost/spirit/utility.hpp>
#include "mcmc++/util.h"
```

Include dependency graph for DataTable.h:



Classes

- class BadCol

  Exception thrown on bad column index.

- class BadRow

Exception thrown on bad row index.

- class DataTable< Type >

  Provides access to homogeneous tabular data.

Defines

- #define __DATATABLE_H
- #define argCheck_ 1

Enumerations

- enum DataTableResult

Functions

- std::ostream & operator<< (std::ostream &out, enum DataTableResult result)

7.2.1 Detailed Description

Provides DataTable class for access to tabular data.

The DataTable class reads homogeneous tabular data, i.e., numerical data that is either all of the same type or that can be converted to the base type of the data table using standard conversions. Rows or columns or both can be labeled, but labels are not required.

Author:

Kent Holsinger

Date:

2004-06-26

Definition in file DataTable.h.

7.2.2 Define Documentation

7.2.2.1 #define argCheck_ 1

argCheck_ controls whether row and column indexes are bounds checked before use

Defaults to 1 (true) unles NDEBUG is defined

Definition at line 85 of file DataTable.h.

Referenced by DataTable< Type >::ColumnLabel(), DataTable< Type >::ColumnVector(), DataTable< Type >::RowLabel(), DataTable< Type >::RowVector(), DataTable< Type >::SetValue(), and DataTable< Type >::Value().

7.2.3 Enumeration Type Documentation

7.2.3.1 enum DataTableResult

enum DataTableResult

codes used to determine whether read was successful and the type of error, if not

Definition at line 55 of file DataTable.h.

7.2.4 Function Documentation

7.2.4.1 std::ostream& operator<< (std::ostream & *out*, enum DataTableResult *result*)

Stream output for DataTable errors

Parameters:

out The stream for output

result The error identifier

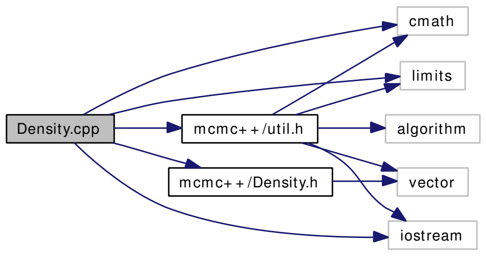
Definition at line 35 of file DataTable.cpp.

7.3 Density.cpp File Reference

Collects a variety of constants and function in namespace Util.

```
#include <cmath>
#include <limits>
#include "mcmc++/Density.h"
#include "mcmc++/util.h"
#include <iostream>
```

Include dependency graph for Density.cpp:



Functions

- double Density::dbeta (const double x, const double a, const double b, const bool give_log)
- double Density::dbinom (const int k, const int n, double p, bool give_log)
- double Density::dcauchy (const double x, const double l, const double s, const bool give_log)
- double Density::dchisq (const double x, const double n, const bool give_log)
- double Density::ddirch (const std::vector< double > &p, const std::vector< double > &a, const bool give_log, const bool include_const)
- double Density::dexp (const double x, const double b, const bool give_log)
- double Density::df (const double x, const double m, const double n, const bool give_log)
- double Density::dgamma (const double x, const double shape, const double scale, const bool give_log)
- double Density::dgeom (const unsigned x, const double p, const bool give_log)
- double Density::dhyper (const unsigned x, const unsigned r, const unsigned b, const unsigned n, bool giveLog)
- double Density::dinvgamma (const double y, const double shape, const double scale, const bool give_log)

- double Density::dlnorm (const double x, const double mu, const double sigma, const bool give_log)
- double Density::dlogis (const double x, const double m, const double s, const bool give_log)
- double Density::dmulti (const std::vector< int > &n, const std::vector< double > &p, const bool give_log, const bool include_factorial)
- double Density::dnbinom (const unsigned x, const double n, const double p, const bool give_log)
- double Density::dnorm (const double x_in, const double mu, const double sigma, const bool give_log)
- double Density::dpois (const unsigned x, const double lambda, const bool give_-log)
- double Density::dt (const double x, const double n, const bool give_log)
- double Density::dweibull (const double x, const double a, const double b, const bool give_log)
- double Density::gamln (const double x)
- double Density::logChoose (const double n, const double k)
- double Density::lbeta (const double a, const double b)
- double Density::BetaEntropy (const double a, const double b)

7.3.1 Detailed Description

Collects a variety of constants and function in namespace Util.

Provides definitions for a variety of functions related to numerical evaluation of probability densities.

All are declared in namespace Density with an eye towards avoiding naming conflicts.

Author:

Kent Holsinger

Date:

2005-05-18

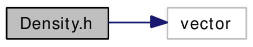
Definition in file Density.cpp.

7.4 Density.h File Reference

Collects a variety of constants and function in namespace Util.

`#include <vector>`

Include dependency graph for Density.h:



Namespaces

- namespace Density

Defines

- #define __DENSITY_H

Functions

- double Density::dbeta (const double x, const double a, const double b, const bool give_log)
- double Density::dbinom (const int k, const int n, double p, bool give_log)
- double Density::dcauchy (const double x, const double l, const double s, const bool give_log)
- double Density::dchisq (const double x, const double n, const bool give_log)
- double Density::ddirch (const std::vector< double > &p, const std::vector< double > &a, const bool give_log, const bool include_const)
- double Density::dexp (const double x, const double b, const bool give_log)
- double Density::df (const double x, const double m, const double n, const bool give_log)
- double Density::dgeom (const unsigned x, const double p, const bool give_log)
- double Density::dgamma (const double x, const double shape, const double scale, const bool give_log)
- double Density::dhyper (const unsigned x, const unsigned r, const unsigned b, const unsigned n, bool giveLog)
- double Density::dinvgamma (const double y, const double shape, const double scale, const bool give_log)
- double Density::dlnorm (const double x, const double mu, const double sigma, const bool give_log)
- double Density::dlogis (const double x, const double m, const double s, const bool give_log)

- double Density::dmulti (const std::vector< int > &n, const std::vector< double > &p, const bool give_log, const bool include_factorial)
- double Density::dnbinom (const unsigned x, const double n, const double p, const bool give_log)
- double Density::dnorm (const double x_in, const double mu, const double sigma, const bool give_log)
- double Density::dpois (const unsigned x, const double lambda, const bool give_-log)
- double Density::dt (const double x, const double n, const bool give_log)
- double Density::dweibull (const double x, const double a, const double b, const bool give_log)
- double Density::BetaEntropy (const double a, const double b)
- double Density::logChoose (const double n, const double k)
- double Density::gamln (const double x)
- double Density::lbeta (const double a, const double b)

Variables

- const double Density::MinGammaPar
- const double Density::MaxGammaPar

7.4.1 Detailed Description

Collects a variety of constants and function in namespace Util.

Provides definitions for a variety of functions related to numerical evaluation of probability densities.

All are declared in namespace Density with an eye towards avoiding naming conflicts.

Author:

Kent Holsinger

Date:

2004-07-03

Definition in file Density.h.

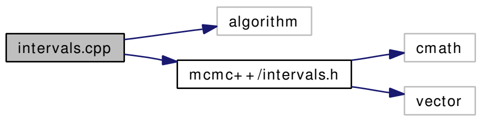
7.5 intervals.cpp File Reference

defintions for quantile(), p_value(), and hpd()

`#include <algorithm>`

`#include "mcmc++/intervals.h"`

Include dependency graph for intervals.cpp:



Functions

- double quantile (vector< double > &ox, const double p)
- double p_value (vector< double > &x, const double p)
- vector< double > hpd (vector< double > &x, const double p)

7.5.1 Detailed Description

defintions for quantile(), p_value(), and hpd()

Author:

Kent Holsinger

Date:

2005-05-18

Definition in file intervals.cpp.

7.5.2 Function Documentation

7.5.2.1 vector<double> hpd (vector< double > & *x*, const double *p*)

Returns HPD interval, low in x[0], high in x[1].

Returns lower limit of HPD interval in first element of vector, upper limit of HPD interval in second element of vector. This method is an implementation of the Chen-Shao HPD estimation algorithm.

Notice that as a side effect of calling this routine, the input vector is sorted.

Parameters:

x The vector

p The credible interval desired, e.g., 0.95 for 95%

Definition at line 95 of file intervals.cpp.

7.5.2.2 double p_value (vector< double > & *x*, const double *p*)

Returns P(x <= p).

Notice that as a side effect of calling this routine, the input vector is sorted.

Parameters:

x The vector

p The desired P-value

Definition at line 72 of file intervals.cpp.

7.5.2.3 double quantile (vector< double > & *ox*, const double *p*)

Returns the sample quantile corresponding to p.

This implementation corresponds to Type 8 continuous sample types of Hyndman and Fan (American Statistician, 50:361-365; 1996)

Notice that as a side effect of calling this routine, the input vector is sorted.

Parameters:

ox The vector from which to calculate the quantile

p The desired quantile

Definition at line 46 of file intervals.cpp.

Referenced by Model::ReportDic(), and Model::Summarize().

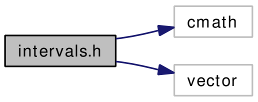
7.6 intervals.h File Reference

declarations for quantile(), p_value(), and hpd()

```
#include <cmath>
```

```
#include <vector>
```

Include dependency graph for intervals.h:



Defines

- #define __INTERVALS_H

7.6.1 Detailed Description

declarations for quantile(), p_value(), and hpd()

Author:

Kent Holsinger

Date:

2004-06-26

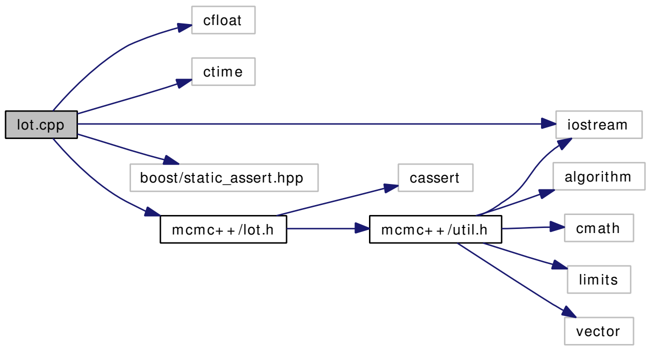
Definition in file intervals.h.

7.7 lot.cpp File Reference

Random number generators.

```
#include <cfloat>
#include <ctime>
#include <iostream>
#include <boost/static_assert.hpp>
#include "mcmc++/lot.h"
```

Include dependency graph for lot.cpp:



Namespaces

- namespace lot_conditions

Defines

- #define unif_rand() uniform()
- #define exp_rand() expon()
- #define norm_rand() snorm()
- #define repeat for (;;)
- #define expmax log(DBL_MAX)

7.7.1 Detailed Description

Random number generators.

A series of random number generators are provided here. Any one of three uniform random number generators can be chosen. By default the Mersenne twister is used.

Author:

Kent Holsinger & Paul Lewis

Date:

2005-05-18

The random number generators from R have been checked for numerical accuracy with the routines in R v2.0. See lotTest.cpp for the specific small set of test run. In every case the results differ from those reported by R by less than 1.0e-11, and are exact for integer random variables.

Definition in file lot.cpp.

7.7.2 Define Documentation

7.7.2.1 #define exp_rand() expon()

Macro for compatibility with RNGs from R

Definition at line 62 of file lot.cpp.

Referenced by lot::gamma(), lot::geom(), and lot::poisson().

7.7.2.2 #define norm_rand() snorm()

Macro for compatibility with RNGs from R

Definition at line 65 of file lot.cpp.

Referenced by lot::gamma(), lot::poisson(), and lot::t().

7.7.2.3 #define repeat for (;;)

Macro for compatibility with RNGs from R

Definition at line 68 of file lot.cpp.

Referenced by lot::binom(), lot::gamma(), lot::poisson(), and lot::snorm().

7.7.2.4 #define unif_rand() uniform()

Macro for compatibility with RNGs from R

Definition at line 59 of file lot.cpp.

Referenced by lot::beta(), lot::binom(), lot::cauchy(), lot::gamma(), lot::hypergeom(), lot::poisson(), lot::snorm(), and lot::weibull().

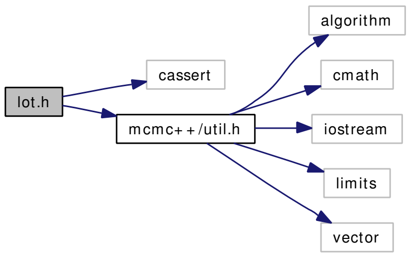
7.8 lot.h File Reference

Random number generators.

`#include <cassert>`

`#include "mcmc++/util.h"`

Include dependency graph for lot.h:



Classes

- class lot

  Provides a series of random number generators.

Typedefs

- typedef std::vector< double > DblVect

7.8.1 Detailed Description

Random number generators.

A series of random number generators are provided here. Any one of three uniform random number generators can be chosen. By default the Mersenne twister is used.

Author:

Kent Holsinger & Paul Lewis

Date:

2004-06-26

Definition in file lot.h.

7.8.2 Typedef Documentation

7.8.2.1 typedef std::vector<double> DblVect

typedef to allow more succint declarations for some variables

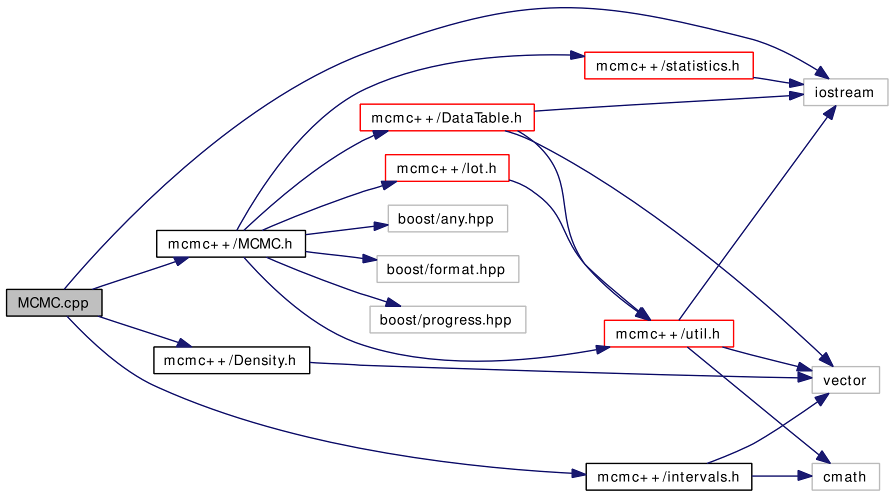
Definition at line 42 of file lot.h.

7.9 MCMC.cpp File Reference

Definitions of classes for MCMC evaluation of Bayesian models.

```
#include <iostream>
#include "mcmc++/Density.h"
#include "mcmc++/MCMC.h"
#include "mcmc++/intervals.h"
```

Include dependency graph for MCMC.cpp:



Classes

- struct BadValue
- struct BadCT

Defines

- #define argCheck_ 1
- #define checkValue_(x) ;

Typedefs

- typedef ParameterT< double > dPar

- typedef ParameterT< int > iPar
- typedef ParameterT< std::vector< double > > dVecPar
- typedef ParameterT< std::vector< int > > iVecPar
- typedef ParameterT< boost::any > aPar
- typedef const ParameterT< double > cdPar
- typedef const ParameterT< int > ciPar
- typedef const ParameterT< std::vector< double > > cdVecPar
- typedef const ParameterT< std::vector< int > > ciVecPar
- typedef const ParameterT< boost::any > caPar

Functions

- lot & GetRNG (void)
- double safeFreq (const double p, const double zero)
- double invSafeFreq (const double x, const double zero)
- std::vector< double > safeFreq (const std::vector< double > &p, const double zero)
- std::vector< double > invSafeFreq (const std::vector< double > &x, const double zero)
- double proposeBeta (const double mean, const double denom, const double tolerance)
- double logQBeta (const double newMean, const double oldMean, const double denom, const double tolerance)
- std::vector< double > proposeDirch (const std::vector< double > &q, const double denom, const double tolerance)
- double logQDirch (const std::vector< double > &newMean, const std::vector< double > &oldMean, const double denom, const double tolerance)
- double proposeNorm (const double mean, const double variance)
- double logQNorm (const double newMean, const double oldMean, const double variance)
- double safeBetaPar (const double x)
- void safeDirchPar (std::vector< double > &x)

Variables

- const double MCMC_ZERO_FREQ = 1.00e-14

7.9.1 Detailed Description

Definitions of classes for MCMC evaluation of Bayesian models.

The Parameter, Step, and Model classes defined here are the core classes that do all of the work. To implement a Bayesian model using these classes each parameter in the model should be derived from Paramater and should override llike(), at a minimum. lprior() should be overridden for anything other than a flat prior. (Notice that the prior will be improper unless overriden when the parameter has an unbounded domain.) The model is derived from Model, and each parameter is pushed onto a stack (step_), with a specified Step type (MetroStep, SliceStep, or FunctionStep).

Author:

Kent Holsinger

Date:

2005-05-18

Definition in file MCMC.cpp.

7.9.2 Define Documentation

7.9.2.1 #define argCheck_ 1

argCheck_ controls whether arguments are bounds checked before use

Defaults to 1 (true) unles NDEBUG is defined

Definition at line 61 of file MCMC.cpp.

7.9.2.2 #define checkValue_(x) ;

Macro to determine whether a value isnan() and print __FILE__ and __LINE__ if it is (disabled when argCheck == 0 (i.e., when NDEBUG is defined during compilation

Definition at line 79 of file MCMC.cpp.

7.9.3 Typedef Documentation

7.9.3.1 typedef ParameterT<boost::any> aPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 344 of file MCMC.cpp.

7.9.3.2 typedef const ParameterT<boost::any> caPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 408 of file MCMC.cpp.

7.9.3.3 typedef const ParameterT<double> cdPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 392 of file MCMC.cpp.

7.9.3.4 typedef const ParameterT<std::vector<double> > cdVecPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 400 of file MCMC.cpp.

7.9.3.5 typedef const ParameterT<int> ciPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 396 of file MCMC.cpp.

7.9.3.6 typedef const ParameterT<std::vector<int> > ciVecPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 404 of file MCMC.cpp.

7.9.3.7 typedef ParameterT<double> dPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 328 of file MCMC.cpp.

7.9.3.8 typedef ParameterT<std::vector<double> > dVecPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 336 of file MCMC.cpp.

7.9.3.9 typedef ParameterT<int> iPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 332 of file MCMC.cpp.

7.9.3.10 typedef ParameterT<std::vector<int> > iVecPar

internal typedef, defined here so that it is invisible and inaccessible to users of the library

Definition at line 340 of file MCMC.cpp.

7.9.4 Function Documentation

7.9.4.1 lot& GetRNG (void)

Returns a reference to the internal random number generator

Definition at line 104 of file MCMC.cpp.

7.9.4.2 std::vector<double> invSafeFreq (const std::vector< double > & *x*, const double *zero*)

Inverts a safeFreq()ed vector

Parameters:

x The vector to invert

zero The original guard (not checked)

Definition at line 153 of file MCMC.cpp.

7.9.4.3 double invSafeFreq (const double *x*, const double *zero*)

Converts a safeFreq()ed x back

Parameters:

x Frequency to convert back

zero Value used in original guard (not verified)

Definition at line 124 of file MCMC.cpp.

7.9.4.4 double logQBeta (const double *newMean*, const double *oldMean*, const double *denom*, const double *tolerance*)

lQ for a beta proposal

Parameters:

newMean New value proposed
oldMean Old value from which proposed
denom Effective sample size
tolerance Minimum value allowed in original proposal

Definition at line 196 of file MCMC.cpp.

References Density::dbeta(), invSafeFreq(), safeBetaPar(), and Util::safeLog().

7.9.4.5 double logQDirch (const std::vector< double > & *newMean*, const std::vector< double > & *oldMean*, const double *denom*, const double *tolerance*)

lQ for a Dirichlet proposal

Parameters:

newMean New value proposed
oldMean Old value from which proposed
denom Effective sample size
tolerance Minimum value allowed in original proposal

Definition at line 242 of file MCMC.cpp.

References Density::ddirch(), invSafeFreq(), and Util::safeLog().

7.9.4.6 double logQNorm (const double *newMean*, const double *oldMean*, const double *variance*)

lQ for a normal proposal

Parameters:

newMean New value proposed
oldMean Old value from which proposed
variance Variance of the proposal distribution

Definition at line 271 of file MCMC.cpp.

References Density::dnorm(), and Util::safeLog().

7.9.4.7 double proposeBeta (const double *mean*, const double *denom*, const double *tolerance*)

Propose a new beta variate for an M-H step

Parameters:

mean Mean for beta distribution

denom Effective sample size

tolerance Minimum value allowed (to avoid exact 0s and 1s)

The parameters of the beta distribution are given by

$$\alpha = denom \times mean$$

$$\beta = denom \times (1 - mean)$$

where alpha and beta are guarded by safeBetaPar()

Definition at line 178 of file MCMC.cpp.

References lot::beta(), safeBetaPar(), and safeFreq().

7.9.4.8 std::vector<double> proposeDirch (const std::vector< double > & *q*, const double *denom*, const double *tolerance*)

Propose a new Dirichlet vector for an M-H step

Parameters:

q Mean of the proposal distribution

denom Effective sample size

tolerance Minimum value allowed (to avoid exact 0s and 1s)

The parameters of the beta distribution are given by

$$\alpha = denom \times mean$$

$$\beta = denom \times (1 - mean)$$

where alpha and beta are guarded by safeBetaPar()

Definition at line 220 of file MCMC.cpp.

References lot::dirichlet(), safeDirchPar(), and safeFreq().

7.9.4.9 double proposeNorm (const double *mean*, const double *variance*)

Propose a new normal variate for an M-H step

Parameters:

mean Mean of the proposal distribution

variance Variance of the proposal distribution

Definition at line 261 of file MCMC.cpp.

References lot::norm().

7.9.4.10 double safeBetaPar (const double *x*)

Ensures MinBetaPar <= x <= MaxBetaPar

Parameters:

x Frequency to guard

Definition at line 282 of file MCMC.cpp.

References MCMC::MaxBetaPar, and MCMC::MinBetaPar.

Referenced by logQBeta(), and proposeBeta().

7.9.4.11 void safeDirchPar (std::vector< double > & *x*)

Ensures MinBetaPar <= x <= MaxBetaPar for all elements of a Dirichlet parameter vector

Parameters:

x Frequency to guard

Definition at line 293 of file MCMC.cpp.

References MCMC::MaxDirchPar, and MCMC::MinDirchPar.

Referenced by proposeDirch().

7.9.4.12 std::vector<double> safeFreq (const std::vector< double > & *p*, const double *zero*)

Ensures that all values in vector are greater than zero and less than 1.0 - n∗zero, where n is p.size()

Parameters:

p Frequency vector to guard

zero Value used for guard (MCMC_ZERO_FREQ default)

Assumes all elements of p are in [0, 1]

Definition at line 138 of file MCMC.cpp.

7.9.4.13 double safeFreq (const double *p*, const double *zero*)

Ensures zero < p < 1-zero

Parameters:

p Frequency to guard

zero Minimum value of p allowed

Assumes all elements of p are in [0, 1]

Definition at line 115 of file MCMC.cpp.

7.9.5 Variable Documentation

7.9.5.1 const double MCMC_ZERO_FREQ = 1.00e-14

default "zero" value for safeFreq()

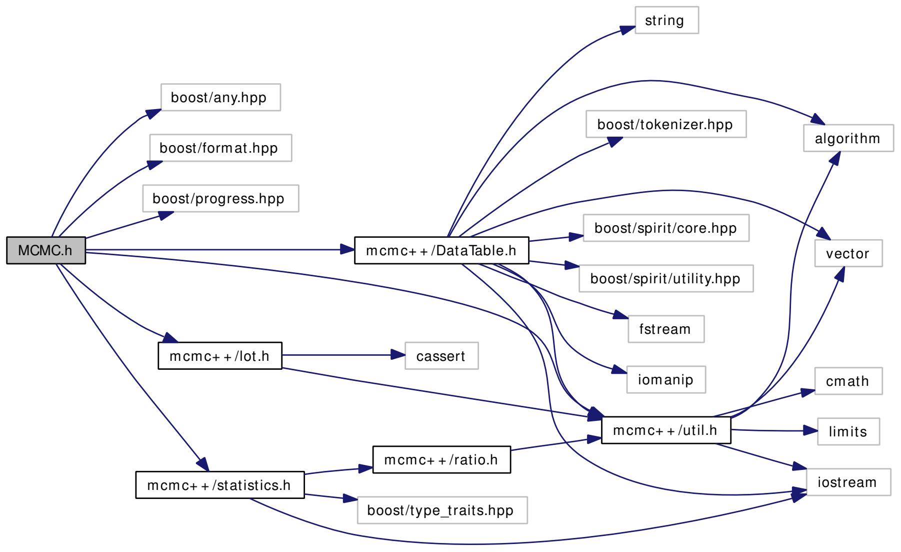
Definition at line 100 of file MCMC.cpp.

7.10 MCMC.h File Reference

Definitions of classes for MCMC evaluation of Bayesian models.

```
#include <boost/any.hpp>
#include <boost/format.hpp>
#include <boost/progress.hpp>
#include "mcmc++/DataTable.h"
#include "mcmc++/lot.h"
#include "mcmc++/util.h"
#include "mcmc++/statistics.h"
```

Include dependency graph for MCMC.h:



Namespaces

- namespace MCMC

Classes

- class ParameterBase

  Pure virtual class provide interface for Parameter.

- class ParameterT< T >

  Base class for model parameters.

- class StepBase

  Base class for steps associated with different parameter types.

- class Step< T >

  Base class for all steps associated with one parameter type.

- class MetroStepT< T >

  Implements Metropolis-Hastings step for a parameter.

- class AdaptMetroStepT< T >

  Implements Metropolis-Hastings step for a parameter.

- class SliceStep

  Implements a slice sampler.

- class FunctionStepT< T >

  Implements a deterministic node.

- class Model

  Implements the statistical model.

Defines

- #define __MCMC_H

Typedefs

- typedef std::vector< StepBase $*$ > ModelSteps

  pointer to each Step in the Model

- typedef std::vector< boost::any > SampleVector

  parameters at one iteration

- typedef SampleVector::const_iterator SampleIter

  paramter iterator

- typedef std::vector< SampleVector > Results

SampleVectors for all iterations.

- typedef Results::const_iterator ResultsIter

  iterator over iterations

- typedef ParameterT< double > Parameter
- typedef FunctionStepT< double > FunctionStep
- typedef MetroStepT< double > MetroStep

Functions

- lot & GetRNG ()
- double safeFreq (double p, double zero=MCMC_ZERO_FREQ)
- double invSafeFreq (double x, double zero=MCMC_ZERO_FREQ)
- std::vector< double > safeFreq (const std::vector< double > &p, const double zero)
- std::vector< double > invSafeFreq (const std::vector< double > &x, const double zero=MCMC_ZERO_FREQ)
- double proposeBeta (double mean, double denom, double tolerance)
- double logQBeta (double newMean, double oldMean, double denom, double tolerance)
- std::vector< double > proposeDirch (const std::vector< double > &mean, double denom, double tolerance)
- double logQDirch (const std::vector< double > &newMean, const std::vector< double > &oldMean, double denom, double tolerance)
- double proposeNorm (double mean, double variance)
- double logQNorm (double newMean, double oldMean, double variance)
- double safeBetaPar (double x)
- void safeDirchPar (std::vector< double > &x)

Variables

- const double MCMC_ZERO_FREQ
- const double MCMC::MinBetaPar = 1.0e-1

  minimum value allowed for beta parameter

- const double MCMC::MaxBetaPar = 1.0e4

  maximum value allowed for beta parameter

- const double MCMC::MinDirchPar = 1.0e-1

  minimum value allowed for Dirichlet parameter

- const double MCMC::MaxDirchPar = 1.0e4

  maximum value allowed for Dirichlet parameter

7.10.1 Detailed Description

Definitions of classes for MCMC evaluation of Bayesian models.

The ParameterT, StepT, and Model classes defined here are the core classes that do all of the work. To implement a Bayesian model using these classes each parameter in the model should be derived from Paramater and should override llike(), at a minimum. lprior() should be overridden for anything other than a flat prior. (Notice that the prior will be improper unless overriden when the parameter has an unbounded domain.) The model is derived from Model, and each parameter is pushed onto a stack (step_), with a specified Step type (MetroStep, SliceStep, or FunctionStep).

Author:

Kent Holsinger

Date:

2004-07-03

Definition in file MCMC.h.

7.10.2 Typedef Documentation

7.10.2.1 typedef FunctionStepT<double> FunctionStep

typedef for convenienc access to double paramters

Definition at line 671 of file MCMC.h.

7.10.2.2 typedef MetroStepT<double> MetroStep

typedef for convenienc access to double paramters

Definition at line 674 of file MCMC.h.

7.10.2.3 typedef ParameterT<double> Parameter

typedef for convenienc access to double paramters

Definition at line 668 of file MCMC.h.

7.10.3 Function Documentation

7.10.3.1 lot& GetRNG (void)

Returns a reference to the internal random number generator

Definition at line 104 of file MCMC.cpp.

7.10.3.2 std::vector<double> invSafeFreq (const std::vector< double > & *x*, const double *zero*)

Inverts a safeFreq()ed vector

Parameters:

x The vector to invert

zero The original guard (not checked)

Definition at line 153 of file MCMC.cpp.

7.10.3.3 double invSafeFreq (const double *x*, const double *zero*)

Converts a safeFreq()ed x back

Parameters:

x Frequency to convert back

zero Value used in original guard (not verified)

Definition at line 124 of file MCMC.cpp.

Referenced by logQBeta(), and logQDirch().

7.10.3.4 double logQBeta (const double *newMean*, const double *oldMean*, const double *denom*, const double *tolerance*)

lQ for a beta proposal

Parameters:

newMean New value proposed

oldMean Old value from which proposed

denom Effective sample size

tolerance Minimum value allowed in original proposal

Definition at line 196 of file MCMC.cpp.

References Density::dbeta(), invSafeFreq(), safeBetaPar(), and Util::safeLog().

7.10.3.5 double logQDirch (const std::vector< double > & *newMean*, const std::vector< double > & *oldMean*, const double *denom*, const double *tolerance*)

lQ for a Dirichlet proposal

Parameters:

newMean New value proposed

oldMean Old value from which proposed

denom Effective sample size

tolerance Minimum value allowed in original proposal

Definition at line 242 of file MCMC.cpp.

References Density::ddirch(), invSafeFreq(), and Util::safeLog().

7.10.3.6 double logQNorm (const double *newMean*, const double *oldMean*, const double *variance*)

lQ for a normal proposal

Parameters:

newMean New value proposed

oldMean Old value from which proposed

variance Variance of the proposal distribution

Definition at line 271 of file MCMC.cpp.

References Density::dnorm(), and Util::safeLog().

7.10.3.7 double proposeBeta (const double *mean*, const double *denom*, const double *tolerance*)

Propose a new beta variate for an M-H step

Parameters:

mean Mean for beta distribution

denom Effective sample size
tolerance Minimum value allowed (to avoid exact 0s and 1s)

The parameters of the beta distribution are given by

$$\alpha = denom \times mean$$
$$\beta = denom \times (1 - mean)$$

where alpha and beta are guarded by safeBetaPar()

Definition at line 178 of file MCMC.cpp.

References lot::beta(), safeBetaPar(), and safeFreq().

7.10.3.8 std::vector<double> proposeDirch (const std::vector< double > & *q*, const double *denom*, const double *tolerance*)

Propose a new Dirichlet vector for an M-H step

Parameters:

q Mean of the proposal distribution
denom Effective sample size
tolerance Minimum value allowed (to avoid exact 0s and 1s)

The parameters of the beta distribution are given by

$$\alpha = denom \times mean$$
$$\beta = denom \times (1 - mean)$$

where alpha and beta are guarded by safeBetaPar()

Definition at line 220 of file MCMC.cpp.

References lot::dirichlet(), safeDirchPar(), and safeFreq().

7.10.3.9 double proposeNorm (const double *mean*, const double *variance*)

Propose a new normal variate for an M-H step

Parameters:

mean Mean of the proposal distribution
variance Variance of the proposal distribution

Definition at line 261 of file MCMC.cpp.

References lot::norm().

7.10.3.10 double safeBetaPar (const double *x*)

Ensures MinBetaPar <= x <= MaxBetaPar

Parameters:

x Frequency to guard

Definition at line 282 of file MCMC.cpp.

References MCMC::MaxBetaPar, and MCMC::MinBetaPar.

Referenced by logQBeta(), and proposeBeta().

7.10.3.11 void safeDirchPar (std::vector< double > & *x*)

Ensures MinBetaPar <= x <= MaxBetaPar for all elements of a Dirichlet parameter vector

Parameters:

x Frequency to guard

Definition at line 293 of file MCMC.cpp.

References MCMC::MaxDirchPar, and MCMC::MinDirchPar.

Referenced by proposeDirch().

7.10.3.12 std::vector<double> safeFreq (const std::vector< double > & *p*, const double *zero*)

Ensures that all values in vector are greater than zero and less than 1.0 - n∗zero, where n is p.size()

Parameters:

p Frequency vector to guard

zero Value used for guard (MCMC_ZERO_FREQ default)

Assumes all elements of p are in [0, 1]

Definition at line 138 of file MCMC.cpp.

7.10.3.13 double safeFreq (const double *p*, const double *zero*)

Ensures zero < p < 1-zero

Parameters:

- ***p*** Frequency to guard
- ***zero*** Minimum value of p allowed

Assumes all elements of p are in [0, 1]

Definition at line 115 of file MCMC.cpp.

Referenced by proposeBeta(), and proposeDirch().

7.10.4 Variable Documentation

7.10.4.1 const double MCMC_ZERO_FREQ

default "zero" value for safeFreq()

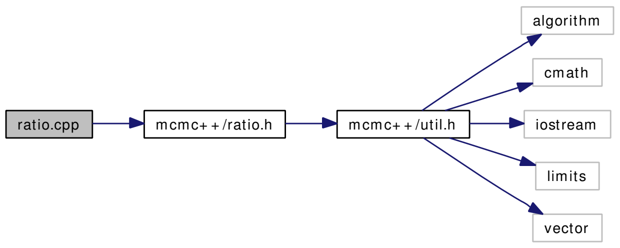
Definition at line 100 of file MCMC.cpp.

7.11 ratio.cpp File Reference

Provides a simple ratio class.

```
#include "mcmc++/ratio.h"
```

Include dependency graph for ratio.cpp:



7.11.1 Detailed Description

Provides a simple ratio class.

Author:

Kent Holsinger

Date:

2005-05-18

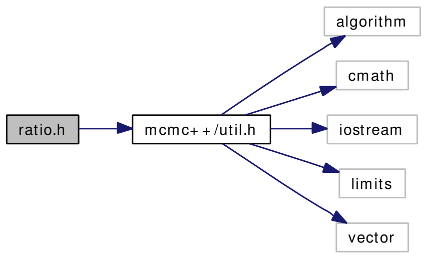
Definition in file ratio.cpp.

7.12 ratio.h File Reference

Provides a simple ratio class.

`#include "mcmc++/util.h"`

Include dependency graph for ratio.h:



Classes

- class ratio

  Provides a ratio class.

Defines

- #define __RATIO_H

7.12.1 Detailed Description

Provides a simple ratio class.

Author:

Kent Holsinger

Date:

2004-06-26

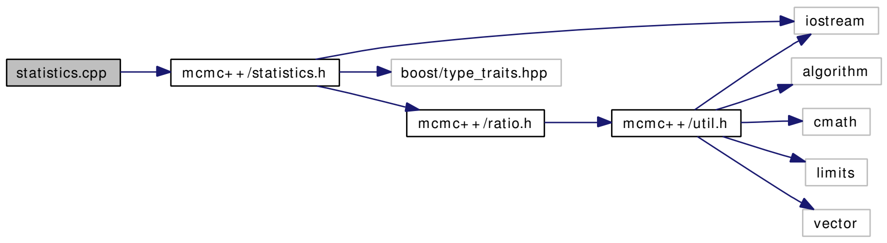
Definition in file ratio.h.

7.13 statistics.cpp File Reference

Classes for descriptive statistics.

```
#include "mcmc++/statistics.h"
```

Include dependency graph for statistics.cpp:



Functions

- std::ostream & operator<< (std::ostream &out, Statistic &st)

7.13.1 Detailed Description

Classes for descriptive statistics.

This file provides two statistical classes: Statistic and SimpleStatistic. As the names suggest, Statistic is more complete. It includes methods for standard deviation and coefficient of variation as well as mean and variance. It can also calculate statistics on ratios (using ratio.h)

Author:

Kent Holsinger

Date:

2005-05-18

Definition in file statistics.cpp.

7.13.2 Function Documentation

7.13.2.1 std::ostream& operator<< (std::ostream & *out*, Statistic & *st*)

Stream output for Statistic.

Reports sample size, mean, variance, standard deviation, and coefficient of variation, each preceded by a tab and appearing on a new line.

Parameters:

- ***out*** The output stream
- ***st*** The statistics

Definition at line 202 of file statistics.cpp.

References Statistic::CV(), Statistic::Mean(), Statistic::N(), Statistic::StdDev(), and Statistic::Variance().

7.14 statistics.h File Reference

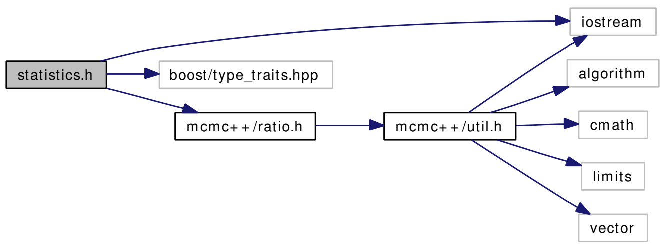
Classes for descriptive statistics.

`#include <iostream>`

`#include <boost/type_traits.hpp>`

`#include "mcmc++/ratio.h"`

Include dependency graph for statistics.h:



Namespaces

- namespace keh

Classes

- class keh::Accumulate< false, T >
- class keh::Accumulate< true, T >
- class Statistic

  Implements a class for summary statistics.

- class SimpleStatistic

  Implements a class for summary statistics.

Defines

- #define __STATISTI_H

7.14.1 Detailed Description

Classes for descriptive statistics.

This header file provides two statistical classes: Statistic and SimpleStatistic. As the names suggest, Statistic is more complete. It includes methods for standard deviation and coefficient of variation as well as mean and variance. It can also calculate statistics on ratios (using ratio.h)

Author:

Kent Holsinger

Date:

2004-06-26

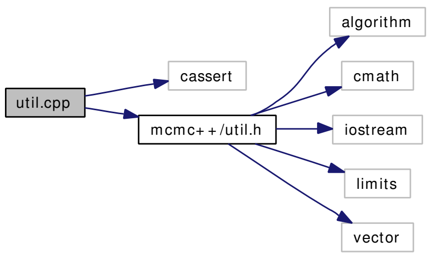
Definition in file statistics.h.

7.15 util.cpp File Reference

Collects a variety of constants and function in namespace Util.

```
#include <cassert>
#include "mcmc++/util.h"
```

Include dependency graph for util.cpp:



Functions

- double Util::round (const double x)
- double Util::safeLog (const double x)
- double Util::sqr (const double x)

7.15.1 Detailed Description

Collects a variety of constants and function in namespace Util.

Provides definitions for a variety of numerical constants related to double precision and integer arithmetic and for a small collection of utility functions.

All are declared in namespace Util with an eye towards avoiding naming conflicts.

Author:

Kent Holsinger

Date:

2005-05-18

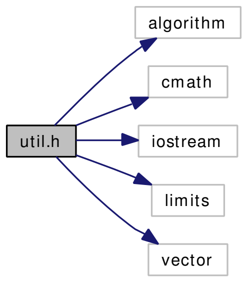
Definition in file util.cpp.

7.16 util.h File Reference

Collects a variety of constants and function in namespace Util.

```
#include <algorithm>
#include <cmath>
#include <iostream>
#include <limits>
#include <vector>
```

Include dependency graph for util.h:



Namespaces

- namespace Util

Classes

- class Util::PrintForVector< T >

Defines

- #define __UTIL_H

Functions

- template<class C> C Util::vectorMin (std::vector< C > &v)
- template<class C> C Util::vectorMax (std::vector< C > &v)
- double Util::round (double x)
- double Util::safeLog (double x)

- double Util::sqr (double x)
- template<class C> void Util::FlushVector (std::vector< C > &v)
- template<class Except, class Assertion> void Util::Assert (Assertion assert)
- template<class To, class From> std::vector< To > Util::vector_cast (std::vector< From > &x)
- template<class T> std::ostream & operator<< (std::ostream &os, std::vector< T > x)

Variables

- const double Util::dbl_eps = std::numeric_limits<double>::epsilon()

  minimum representable value of 1.0 - x

- const double Util::dbl_max = std::numeric_limits<double>::max()

  maximum value of double

- const double Util::dbl_min = std::numeric_limits<double>::min()

  minimum (positive) value of double

- const int Util::int_max = std::numeric_limits<int>::max()

  maximum value of integer

- const int Util::int_min = std::numeric_limits<int>::min()

  minimum value of integer

- const unsigned Util::uint_max = std::numeric_limits<unsigned>::max()

  maximum value of unsigned integer

- const long Util::long_max = std::numeric_limits<long>::max()

  maximum value of long integer

- const long Util::long_min = std::numeric_limits<long>::min()

  minimum value of long integer

- const unsigned long Util::ulong_max = std::numeric_limits<unsigned long>::max()

  maximum value of unsigned long integer

- const double Util::log_dbl_max = log(dbl_max)

  log(dbl_max)

- const double Util::log_dbl_min = log(dbl_min)

  log(dbl_min)

7.16.1 Detailed Description

Collects a variety of constants and function in namespace Util.

Provides definitions for a variety of numerical constants related to double precision and integer arithmetic and for a small collection of utility functions.

All are declared in namespace Util with an eye towards avoiding naming conflicts.

Author:

Kent Holsinger

Date:

2004-06-26

Definition in file util.h.

7.16.2 Function Documentation

7.16.2.1 template<class T> std::ostream& operator<< (std::ostream & *os*, std::vector< T > *x*)

ostream& operator<< for vectors

Parameters:

os the ostream

x the vector

Definition at line 172 of file util.h.