Timescales Library

0.3

Generated by Doxygen 1.7.4

Mon Jul 18 2011 19:06:40

Contents

1 Main Page

1.1 Introduction

The Timescales library provides basic functions for lightcurve analysis. The target application is automated reduction of large time-series data sets.

The library is organized as a series of global functions under the kpftimes namespace, rather than as an object heirarchy. This architecture was chosen in part for its efficiency (i.e. avoiding the memory overhead of constructing objects to represent each periodogram or other function), but mainly for its simplicity. Each function performs a single, narrowly defined task, making it (hopefully!) easy to chain functions together into pipelines.

1.2 About this Documentation

New users will find the Module Documentation chapter the best starting point for learning about the Timescales API. There they will find a list of the main functions in the library, organized by category. The other chapters are more useful for people seeking to understand the code itself.

1.3 Installation and Use

Timescales itself should compile on any UNIX-like system. In many cases you need simply unpack the .tar contents into the appropriate directory, run `make`, and move libtimescales.a into a directory of your choice. If you do not use GCC, you may need to edit the makefile before you can build the library.

To use Timescales, include `<timescales.h>` in your source code (see examples/example.cpp). Timescales relies on the GNU Scientific Library (GSL) for some of its more complex mathematics, so you must link your program with *both* Timescales and GSL for it to run correctly. Check with your system administrator if you're not sure whether GSL is available on your machine.

1.4 Recent changes

1.4.1 0.2.1

- Added acWindow()

2 Test List

Member kpftimes::deltaT(const DoubleVec &times) Regular grid, length 1. Expected behavior: throws invalid_argument

Regular grid, length 2. Expected behavior: returns PNF = 2∗step = std::max_-value-std::min_value

Regular grid, length 100. Expected behavior: returns PNF = 100∗step = std::max_-value-std::min_value

Irregular grid, 2 values randomly chosen from [0, 1). Expected behavior: returns PNF = std::max_value-std::min_value

Irregular grid, 100 values randomly chosen from [0, 1). Expected behavior: returns PNF = std::max_value-std::min_value

Member kpftimes::FastTable::at(size_t x, size_t y) FastTable(4, 5).at(0, 2). Expected behavior: update reflected by separate .at().

FastTable(4, 5).at(3, 2). Expected behavior: update reflected by separate .at().

FastTable(4, 5).at(2, 0). Expected behavior: update reflected by separate .at().

FastTable(4, 5).at(2, 4). Expected behavior: update reflected by separate .at().

FastTable(1000, 1000).at(x,y) = x+y±x∗y iteration in both directions. Expected behavior: y-inner loop faster than x-inner loop for 10 out of 10 objects.

Member kpftimes::FastTable::FastTable(size_t dimX, size_t dimY) FastTable(-1, 1). Expected behavior: throw invalid_argument

FastTable(0, 1). Expected behavior: throw invalid_argument

FastTable(1, -1). Expected behavior: throw invalid_argument

FastTable(1, 0). Expected behavior: throw invalid_argument

FastTable(1, 1). Expected behavior: success. Can iterate over elements with external loop.

FastTable(5, 5). Expected behavior: success. Can iterate over elements with external loop.

FastTable(4, 5). Expected behavior: success. Can iterate over elements with external loop.

FastTable(5, 4). Expected behavior: success. Can iterate over elements with external loop.

Member kpftimes::FastTable::FastTable(const FastTable &otherTable) FastTable(1, 1). Expected behavior: success. Can iterate over elements with external loop and edit independently.

FastTable(4, 5). Expected behavior: success. Can iterate over elements with external loop and edit independently.

FastTable(5, 4). Expected behavior: success. Can iterate over elements with external loop and edit independently.

Member kpftimes::FastTable::operator=(const FastTable &otherTable) FastTable(1, 1) = FastTable(1, 1). Expected behavior: success. Can iterate over elements with external loop and edit independently.

FastTable(4, 5) = FastTable(4, 5). Expected behavior: success. Can iterate over elements with external loop and edit independently.

FastTable(4, 5) = FastTable(5, 5). Expected behavior: throw domain_error.

FastTable(5, 4) = FastTAble(5, 5). Expected behavior: throw domain_error.

Member kpftimes::isSortedAsc(const DoubleVec &list) Empty list. Expected behavior: true.

List of size 1. Expected behavior: true.

List of size 2, sorted asc. Expected behavior: true.

List of size 2, sorted desc. Expected behavior: false.

List of size 10, list[i] = i^2. Expected behavior: true.

List of size 10, list[i] = (i-5)^2. Expected behavior: false.

List of size 10, random elements. Expected behavior: false.

Member kpftimes::lsNormalEdf(const DoubleVec &times, const DoubleVec &freqs, DoubleVec &powers, DoubleV

A 1-element time series and nSims = 100. Expected behavior = throw invalid_argument

A 2-element time series, sorted with no duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.

A 2-element time series, sorted with duplicates, and nSims = 100. Expected behavior = throw invalid_argument

A 2-element time series, unsorted, and nSims = 100. Expected behavior = throw invalid_argument.

A 100-element nonuniformly sampled time series, sorted with no duplicates, and negative frequencies. Expected behavior = throw domain_error.

A 100-element nonuniformly sampled time series, sorted with no duplicates, and multiple zero frequencies. Expected behavior = matches result of running lomb-Scargle 100 times.

A 100-element uniformly sampled time series, sorted with no duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.

A 100-element nonuniformly sampled time series, sorted with no duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.

A 100-element nonuniformly sampled time series, sorted with no duplicates, and nSims = 0. Expected behavior = throw domain_error.

A 100-element nonuniformly sampled time series, sorted with no duplicates, and nSims = 1. Expected behavior = (probs = 1.0, powers = undefined)

A 100-element nonuniformly sampled time series, sorted with duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.

A 100-element nonuniformly sampled time series, unsorted. Expected behavior = throw invalid_argument.

Member kpftimes::maxFreq(const DoubleVec &times) Regular grid, length 1. Expected behavior: throws invalid_argument

Regular grid, length 2. Expected behavior: returns PNF = 1/(2∗step)

Regular grid, length 100. Expected behavior: returns PNF = 1/(2∗step)

Member kpftimes::mean(ForwardIterator first, ForwardIterator last) List of ints, length 0. Expected behavior: throw invalid_argument.

List of ints, length 1. Expected behavior: return list[0]

List of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_mean to within 1e-6 in 10 out of 10 trials.

Vector of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_mean to within 1e-6 in 10 out of 10 trials.

Array of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_mean to within 1e-6 in 10 out of 10 trials.

Member kpftimes::pseudoNyquistFreq(const DoubleVec &times) Regular grid, length 1. Expected behavior: throws invalid_argument

Regular grid, length 2. Expected behavior: returns PNF = 1/(2∗step)

Regular grid, length 100. Expected behavior: returns PNF = 1/(2∗step)

Member kpftimes::variance(ForwardIterator first, ForwardIterator last) List of ints, length 1. Expected behavior: throw invalid_argument.

List of ints, length 2. Expected behavior: return lit[1]-list[0]

List of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_variance to within 1e-6 in 10 out of 10 trials.

Vector of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_variance to within 1e-6 in 10 out of 10 trials.

Array of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_variance to within 1e-6 in 10 out of 10 trials.

3 Todo List

Member kpftimes::acWindow(const DoubleVec &times, const DoubleVec &offsets, DoubleVec &wf)
Verify that input validation is worth the cost

Member kpftimes::acWindow(const DoubleVec &times, const DoubleVec &offsets, DoubleVec &wf, double maxF
Verify that input validation is worth the cost

Member kpftimes::autoCorr(const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, Doub
Allow autoCorr to run with arbitrary offset grids

Verify that input validation is worth the cost

Member kpftimes::autoCorr(const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, Doub
Verify that input validation is worth the cost

Member kpftimes::dft(const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &freqs, ComplexVec
Find a faster implementation of dft.

Verify that input validation is worth the cost

Member kpftimes::freqGen(const DoubleVec &times, DoubleVec &freq, double fMin, double fMax, double fStep)
How to test this? At all?

Member kpftimes::lombScargle(const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &freq, Dou
Find a faster algorithm

Optimize for multiple calls with similar [but not identical] values of times and freq, as would happen in a large survey where some epochs of some stars are removed for technical reasons.

Verify that input validation is worth the cost

Member kpftimes::lsThreshold(const DoubleVec &times, const DoubleVec &freq, double fap, long nSims)
Test the performance advantage AFTER refactoring

Verify that input validation is worth the cost

Member kpftimes::maxFreq(const DoubleVec &times) How to test this for irregular grids?

Member kpftimes::pseudoNyquistFreq(const DoubleVec &times) Come up with test cases for an irregular grid.

4 Bug List

Member kpftimes::autoCorr(const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, Doub
Many ACFs show a strong fringing effect. The fringes have a wavelength corresponding to the pseudo-Nyquist frequency, or to the highest peak frequency that is below the pseudo-Nyquist frequency if the power spectrum of the data is strongly peaked.

5 Module Index

5.1 Modules

Here is a list of all modules:

6 Class Index

6.1 Class List

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

7 File Index

7.1 File List

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

8 Module Documentation

8.1 Periodogram generation

Defines

- #define SCARGLE_SLOW 0

  A compiler flag controlling the strategy used for lsThreshold().

Functions

- void kpftimes::lombScargle (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &freq, DoubleVec &power) throw (std::invalid_argument, std::domain_error)

  Calculates the Lomb-Scargle periodogram for a time series.
- double kpftimes::lsThreshold (const DoubleVec &times, const DoubleVec &freq, double fap, long nSims) throw (std::invalid_argument, std::domain_error)

  Calculates the significance threshold for a Lomb-Scargle periodogram.
- void kpftimes::lsNormalEdf (const DoubleVec &times, const DoubleVec &freqs, DoubleVec &powers, DoubleVec &probs, long nSims) throw (std::invalid_argument, std::domain_error)

  Calculates the empirical distribution function of false peaks for a Lomb-Scargle periodogram.

8.1.1 Define Documentation

8.1.1.1 #define SCARGLE_SLOW 0

A compiler flag controlling the strategy used for lsThreshold().

If set to 0 or not set [the default], lsThreshold() is optimized for speed at the expense of memory. If set to 1, it is instead optimized for memory use at the expense of speed. The default setting should perform better on all but the oldest systems.

This flag is best set in the call to the compiler, rather than in the code itself.

8.1.2 Function Documentation

8.1.2.1 void kpftimes::lombScargle (const DoubleVec & *times*, const DoubleVec & *fluxes*, const DoubleVec & *freq*, DoubleVec & *power*) throw (std::invalid_argument, std::domain_error)

Calculates the Lomb-Scargle periodogram for a time series.

Parameters

in	*times*	Times at which data were taken
in	*fluxes*	Flux measurements of a source
in	*freq*	The frequency grid over which the periodogram should be calculated. See freqGen() for a quick way to generate a grid.
out	*power*	The periodogram power at each frequency.

Precondition

times contains at least two unique values

times is sorted in ascending order
fluxes is of the same length as times
fluxes has at least two unique values
fluxes[i] is the flux of the source at times[i], for all i
all elements of freq are >= 0

Postcondition

power is of the same length as freq
power[i] is the Lomb-Scargle periodogram evaluated at freq[i], for all i

Exceptions

domain_error	Thrown if negative frequencies are provided
invalid_argument	Thrown if any of the preconditions on the format of times or fluxes are violated.

Performance:

O(times.size() × freq.size()) time

O(times.size() + freq.size()) memory

Todo

Find a faster algorithm
Optimize for multiple calls with similar [but not identical] values of times and freq, as would happen in a large survey where some epochs of some stars are removed for technical reasons.
Verify that input validation is worth the cost

8.1.2.2 void kpftimes::IsNormalEdf (const DoubleVec & *times*, const DoubleVec & *freqs*, DoubleVec & *powers*, DoubleVec & *probs*, long *nSims*) throw (std::invalid_argument, std::domain_error)

Calculates the empirical distribution function of false peaks for a Lomb-Scargle periodogram.

This function is a generalization of IsThreshold().

Parameters

in	*times*	Times at which data were taken
in	*freqs*	The frequency grid over which the periodogram was calculated.
out	*powers*	The power levels at which the EDF is measured
out	*probs*	probs[i] contains the estimated probability that a periodogram calculated from white noise has a peak less than or equal to powers[i]
in	*nSims*	Number of white noise simulations to find the EDF.

Precondition

times contains at least two unique values
times is sorted in ascending order
all elements of freq are >= 0
nSims >> 1

Postcondition

powers.size() == probs.size() == nSims
powers is sorted in ascending order
probs is sorted in ascending order
powers and probs represent an empirical distribution function, such that probs[i] = EDF(powers[i]). The EDF is that of the peak power level observed in the periodograms of observations of an uncorrelated Gaussian noise source, if the uncorrelated source is sampled at the cadence represented by times and the periodogram is measured at frequencies freq.

Exceptions

domain_error	Thrown if negative frequencies or nonpositive nSims are provided.
invalid_argument	Thrown if the preconditions on the format of times are violated.

Performance:

O(times.size() × freq.size() × nSims) time

O(times.size() × freq.size()) memory by default, or O(times.size() + freq.size()) memory if SCARGLE_SLOW is set

Remarks

The intended use is that lsRandomEdf() will accompany a call, or multiple calls, to lombScargle() with the same values of times and freq. Although lsRandomEdf() is optimized for multiple calculations to a degree that lombScargle() cannot, it will still run roughly nSims/10 times longer than lombScargle() itself. Don't run it after every real periodogram!

Test

A 1-element time series and nSims = 100. Expected behavior = throw invalid_argument
A 2-element time series, sorted with no duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.
A 2-element time series, sorted with duplicates, and nSims = 100. Expected behavior = throw invalid_argument
A 2-element time series, unsorted, and nSims = 100. Expected behavior = throw invalid_argument.

A 100-element nonuniformly sampled time series, sorted with no duplicates, and negative frequencies. Expected behavior = throw domain_error.
A 100-element nonuniformly sampled time series, sorted with no duplicates, and multiple zero frequencies. Expected behavior = matches result of running lomb-Scargle 100 times.
A 100-element uniformly sampled time series, sorted with no duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.
A 100-element nonuniformly sampled time series, sorted with no duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.
A 100-element nonuniformly sampled time series, sorted with no duplicates, and nSims = 0. Expected behavior = throw domain_error.
A 100-element nonuniformly sampled time series, sorted with no duplicates, and nSims = 1. Expected behavior = (probs = 1.0, powers = undefined)
A 100-element nonuniformly sampled time series, sorted with duplicates, and nSims = 100. Expected behavior = matches result of running lombScargle 100 times.
A 100-element nonuniformly sampled time series, unsorted. Expected behavior = throw invalid_argument.

8.1.2.3 double kpftimes::lsThreshold (const DoubleVec & *times,* const DoubleVec & *freq,* double *fap,* long *nSims*) throw (std::invalid_argument, std::domain_error)

Calculates the significance threshold for a Lomb-Scargle periodogram.

Parameters

in	*times*	Times at which data were taken
in	*freq*	The frequency grid over which the periodogram was calculated.
in	*fap*	Desired false alarm probability
in	*nSims*	Number of white noise simulations to find the FAP power level.

Returns

The peak power level that will be reached, with probability fap, in a periodogram of white noise.

Precondition

times contains at least two unique values
times is sorted in ascending order
all elements of freq are $>=$ 0
0 $<$ fap $<$ 1
nSims $>=$ 1
fap $\times$ nSims $>>$ 1

Postcondition

The function returns the peak power level observed in the periodograms of (1-fap) of

observations of an uncorrelated Gaussian noise source, if the uncorrelated source is sampled at the cadence represented by times and the periodogram is measured at frequencies freq.

Exceptions

domain_error	Thrown if negative frequencies, fap, or nSims are provided.
invalid_argument	Thrown if any of the preconditions on the format of times or the values of fap and nSims are violated.

Performance:

O(times.size() × freq.size() × nSims) time

O(times.size() × freq.size()) memory by default, or O(times.size() + freq.size()) memory if SCARGLE_SLOW is set

Remarks

The intended use is that lsThreshold() will accompany a call, or multiple calls, to lombScargle() with the same values of times and freq. Although lsThreshold() is optimized for multiple calculations to a degree that lombScargle() cannot, it will still run roughly nSims/10 times longer than lombScargle() itself. Don't run it after every real periodogram!
The significance threshold is a strong function of the observing cadence (times), and a weaker function of the frequency grid (freq). The latter dependence arises because in finer grids a (random or observed) periodogram peak is more likely to be sampled close to its maximum value.

Todo

Test the performance advantage AFTER refactoring
Verify that input validation is worth the cost

8.2 Autocorrelation function generation

Functions

- void kpftimes::autoCorr (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, DoubleVec &acf) throw (std::invalid_argument)

  Calculates the autocorrelation function for a time series.
- void kpftimes::autoCorr (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, DoubleVec &acf, double maxFreq) throw (std::invalid_argument)

  Calculates the autocorrelation function for a time series.

- void kpftimes::acWindow (const DoubleVec &times, const DoubleVec &offsets, DoubleVec &wf) throw (std::invalid_argument)

  Calculates the autocorrelation window function for a time sampling.
- void kpftimes::acWindow (const DoubleVec &times, const DoubleVec &offsets, DoubleVec &wf, double maxFreq) throw (std::invalid_argument)

  Calculates the autocorrelation window function for a time sampling.

8.2.1 Function Documentation

8.2.1.1 void kpftimes::acWindow (const DoubleVec & *times*, const DoubleVec & *offsets*, DoubleVec & *wf*) throw (std::invalid_argument)

Calculates the autocorrelation window function for a time sampling.

Parameters

in	*times*	Times at which data were taken
in	*offsets*	The time grid over which the autocorrelation function should be calculated.
out	*wf*	The value of the window function at each offset.

Precondition

times contains at least two unique values
times is sorted in ascending order
offsets is uniformly sampled from 0 to some maximum offset. This requirement will be relaxed in future versions.

Postcondition

wf is of the same length as offsets
wf[i] is the Scargle autocorrelation function evaluated at offsets[i], for all i

Exceptions

invalid_argument	Thrown if the preconditions on times, or offsets are violated.

Performance:

O(times.size() × offsets.size()) time

Todo

Verify that input validation is worth the cost

8.2.1.2 void kpftimes::acWindow (const DoubleVec & *times*, const DoubleVec & *offsets*, DoubleVec & *wf*, double *maxFreq*) throw (std::invalid_argument)

Calculates the autocorrelation window function for a time sampling.

Parameters

in	*times*	Times at which data were taken
in	*offsets*	The time grid over which the autocorrelation function should be calculated.
in	*maxFreq*	The maximum frequency to consider when calculating the autocorrelation function.
out	*wf*	The value of the window function at each offset.

Warning

This version of acWindow is highly volatile and may be removed from the library in future versions. I recommend its use only for testing of the autocorrelation algorithm. Wherever possible, use acWindow(), which has a stable (or at least forward-compatible) spec.

Precondition

times contains at least two unique values
times is sorted in ascending order
offsets contains at least two elements
offsets contains only nonnegative values
offsets is uniformly sampled from 0 to some maximum offset. This requirement will be relaxed in future versions.
maxFreq is positive

Postcondition

wf is of the same length as offsets
wf[i] is the Scargle autocorrelation function evaluated at offsets[i], for all i

Exceptions

std::invalid_argument	Thrown if the preconditions on times, offsets, or maxFreq are violated.

Performance:

O(times.size() × offsets.size()) time

Todo

Verify that input validation is worth the cost

8.2.1.3 void kpftimes::autoCorr (const DoubleVec & *times,* const DoubleVec & *fluxes,* const DoubleVec & *offsets,* DoubleVec & *acf*) throw (std::invalid_argument)

Calculates the autocorrelation function for a time series.

Parameters

in	*times*	Times at which data were taken
in	*fluxes*	Flux measurements of a source
in	*offsets*	The time grid over which the autocorrelation function should be calculated.
out	*acf*	The value of the autocorrelation function at each offset.

Precondition

times contains at least two unique values
times is sorted in ascending order
fluxes is of the same length as times
fluxes[i] is the flux of the source at times[i], for all i
offsets is uniformly sampled from 0 to some maximum offset. This requirement will be relaxed in future versions.

Postcondition

acf is of the same length as offsets
acf[i] is the Scargle autocorrelation function evaluated at offsets[i], for all i

Exceptions

invalid_argument	Thrown if the preconditions on times, offsets, or length(fluxes) are violated.

Performance:

O(times.size() × offsets.size()) time

Todo

Allow autoCorr to run with arbitrary offset grids
Verify that input validation is worth the cost

Bug

Many ACFs show a strong fringing effect. The fringes have a wavelength corresponding to the pseudo-Nyquist frequency, or to the highest peak frequency that is below the pseudo-Nyquist frequency if the power spectrum of the data is strongly peaked.

8.2.1.4 void kpftimes::autoCorr (const DoubleVec & *times,* const DoubleVec & *fluxes,* const DoubleVec & *offsets,* DoubleVec & *acf,* double *maxFreq*) throw (std::invalid_argument)

Calculates the autocorrelation function for a time series.

Parameters

in	*times*	Times at which data were taken
in	*fluxes*	Flux measurements of a source
in	*offsets*	The time grid over which the autocorrelation function should be calculated.
in	*maxFreq*	The maximum frequency to consider when calculating the autocorrelation function.
out	*acf*	The value of the autocorrelation function at each offset.

Note

Increasing maxFreq will increase the time resolution of acf at the cost of making the entire function noisier.

Warning

This version of autoCorr is highly volatile and may be removed from the library in future versions. I recommend its use only for testing of the autocorrelation algorithm. Wherever possible, use autoCorr(), which has a stable (or at least forward-compatible) spec.

Precondition

times contains at least two unique values
times is sorted in ascending order
fluxes is of the same length as times
fluxes[i] is the flux of the source at times[i], for all i
offsets contains at least two elements
offsets contains only nonnegative values
offsets is uniformly sampled from 0 to some maximum offset. This requirement will be relaxed in future versions.
maxFreq is positive

Postcondition

acf is of the same length as offsets
acf[i] is the Scargle autocorrelation function evaluated at offsets[i], for all i

Exceptions

std::invalid_argument	Thrown if the preconditions on times, offsets, length(fluxes), or maxFreq are violated.

Performance:

O(times.size() × offsets.size()) time

Todo

Verify that input validation is worth the cost

8.3 Frequency/offset grid generation

Functions

- double kpftimes::deltaT (const DoubleVec &times) throw (std::invalid_argument)

  Returns the time interval covered by the data.
- double kpftimes::maxFreq (const DoubleVec &times) throw (std::invalid_argument)

  Returns the highest frequency that can be probed by the data.
- void kpftimes::freqGen (const DoubleVec &times, DoubleVec &freq) throw (std::invalid_argument)

  Creates a frequency grid that can be fed to time series analysis functions.
- void kpftimes::freqGen (const DoubleVec &times, DoubleVec &freq, double fMin, double fMax) throw (std::invalid_argument)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void kpftimes::freqGen (const DoubleVec &times, DoubleVec &freq, double fMin, double fMax, double fStep) throw (std::invalid_argument)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

8.3.1 Function Documentation

8.3.1.1 double kpftimes::deltaT (const DoubleVec & *times*) throw (std::invalid_argument)

Returns the time interval covered by the data.

Parameters

in	*times*	Times at which data were taken

Returns

The length of time between the earliest observation in times and the latest observation in times, in whatever units times is in.

Precondition

times contains at least two unique values
times is sorted in ascending order

Exceptions

invalid_argument	Thrown if preconditions violated.

Performance:

O(times.size()) time

Test

Regular grid, length 1. Expected behavior: throws invalid_argument
Regular grid, length 2. Expected behavior: returns PNF = 2∗step = std::max_value-std::min_value
Regular grid, length 100. Expected behavior: returns PNF = 100∗step = std::max_-value-std::min_value
Irregular grid, 2 values randomly chosen from [0, 1). Expected behavior: returns PNF = std::max_value-std::min_value
Irregular grid, 100 values randomly chosen from [0, 1). Expected behavior: returns PNF = std::max_value-std::min_value

8.3.1.2 void kpftimes::freqGen (const DoubleVec & *times*, DoubleVec & *freq*, double *fMin*, double *fMax*, double *fStep*) throw (std::invalid_argument)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a frequency grid that can be fed to time series analysis functions.

The grid itself is trivial to compute; this function therefore exists mainly as a convenient wrapper for the most commonly needed grids.

Parameters

in	*times*	Times at which data were taken
out	*freq*	The returned frequency grid
in	*fMin*	The requested frequency range
in	*fMax*	The requested frequency range
in	*fStep*	The interval at which freq will sample the frequency range in multiples of 1/(Delta t).

Precondition

times contains at least two unique values
times is sorted in ascending order

fMin $<$ fMax
0 $<$ fStep

Postcondition

freq is a grid of frequencies from fMin to fMax, spaced in units of fStep$*$1/T, where T is the time interval covered.
freq is sorted in ascending order

Exceptions

invalid_argument	Thrown if preconditions violated.

Todo

How to test this? At all?

8.3.1.3 void kpftimes::freqGen (const DoubleVec & *times*, DoubleVec & *freq*, double *fMin*, double *fMax*) throw (std::invalid_argument)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a frequency grid that can be fed to time series analysis functions.

The grid itself is trivial to compute; this function therefore exists mainly as a convenient wrapper for the most commonly needed grids.

Parameters

in	*times*	Times at which data were taken
out	*freq*	The returned frequency grid
in	*fMin*	The requested frequency range
in	*fMax*	The requested frequency range

Precondition

times contains at least two unique values
times is sorted in ascending order
0 $<=$ fMin $<$ fMax

Postcondition

freq is a grid of frequencies from fMin to fMax, spaced in units of 1/2T, where T is the time interval covered.
freq is sorted in ascending order

Exceptions

invalid_argument	Thrown if preconditions violated.

8.3.1.4 void kpftimes::freqGen (const DoubleVec & *times,* DoubleVec & *freq*) throw (std::invalid_argument)

Creates a frequency grid that can be fed to time series analysis functions.

The grid itself is trivial to compute; this function therefore exists mainly as a convenient wrapper for the most commonly needed grids.

Parameters

in	*times*	Times at which data were taken
out	*freq*	The returned frequency grid

Precondition

times contains at least two unique values
times is sorted in ascending order

Postcondition

freq is a grid of frequencies from 0 to pseudoNyquistFreq(times) , spaced in units of 1/2T, where T is the time interval covered.
freq is sorted in ascending order

Exceptions

invalid_argument	Thrown if preconditions violated.

8.3.1.5 double kpftimes::maxFreq (const DoubleVec & *times*) throw (std::invalid_argument)

Returns the highest frequency that can be probed by the data.

This is defined as 1/2dt, where dt $>$ 0 is the **smallest** time interval between any two observations.

Parameters

in	*times*	Times at which data were taken

Returns

The highest meaningful frequency, in the inverse of whatever units times is in.

Precondition

times contains at least two unique values
times is sorted in ascending order

Exceptions

invalid_argument	Thrown if preconditions violated.

Performance:

O(times.size()) time

Test

Regular grid, length 1. Expected behavior: throws invalid_argument
Regular grid, length 2. Expected behavior: returns PNF = 1/(2∗step)
Regular grid, length 100. Expected behavior: returns PNF = 1/(2∗step)

Todo

How to test this for irregular grids?

8.4 Utility functions

Classes

- class kpftimes::FastTable

  Two-dimensional dynamically allocated table.

Functions

- void kpftimes::dft (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &freqs, ComplexVec &dft)

  Calculates the discrete Fourier transform for a list of times and fluxes.
- bool kpftimes::isSortedAsc (const DoubleVec &list)

  Tests whether the argument is sorted.
- template<class ForwardIterator >
  std::iterator_traits< ForwardIterator >::value_type kpftimes::mean (ForwardIterator first, ForwardIterator last) throw (std::invalid_argument)

  Finds the mean of the values in a generic container object.
- template<class ForwardIterator >
  std::iterator_traits< ForwardIterator >::value_type kpftimes::variance (ForwardIterator first, ForwardIterator last) throw (std::invalid_argument)

  Finds the variance of the values in a generic container object.
- double kpftimes::pseudoNyquistFreq (const DoubleVec &times) throw (std::invalid_argument)

  Returns the pseudo-Nyquist frequency for a grid of observations.

8.4.1 Function Documentation

8.4.1.1 void kpftimes::dft (const DoubleVec & *times*, const DoubleVec & *fluxes*, const DoubleVec & *freqs*, ComplexVec & *dft*)

Calculates the discrete Fourier transform for a list of times and fluxes.

Parameters

in	*times*	Times at which data were taken
in	*fluxes*	Flux measurements of a source
in	*freqs*	The frequency grid over which the DFT should be calculated. See freqGen() for a quick way to generate a grid.
out	*dft*	Fourier transform at each frequency.

Precondition

times contains at least two unique values
times is sorted in ascending order
fluxes is of the same length as times
fluxes[i] is the flux of the source at times[i], for all i
all elements of freq are >= 0

Postcondition

dft is of the same length as freq
dft[i] is the discrete Fourier transform evaluated at freq[i], for all i

Exceptions

domain_error	Thrown if negative frequencies are provided
invalid_argument	Thrown if any of the preconditions on the format of times or fluxes are violated.

Performance:

O(times.size() × freqs.size()) time

Todo

Find a faster implementation of dft.
Verify that input validation is worth the cost

8.4.1.2 bool kpftimes::isSortedAsc (const DoubleVec & *list*)

Tests whether the argument is sorted.

Parameters

in	*list*	Times at which data were taken

Returns

TRUE if and only if the argument is sorted in ascending order.

Performance:

O(list.size()) time

Test

Empty list. Expected behavior: true.
List of size 1. Expected behavior: true.
List of size 2, sorted asc. Expected behavior: true.
List of size 2, sorted desc. Expected behavior: false.
List of size 10, list[i] = i^2. Expected behavior: true.
List of size 10, list[i] = (i-5)^2. Expected behavior: false.
List of size 10, random elements. Expected behavior: false.

8.4.1.3 template<class ForwardIterator > std::iterator_traits<ForwardIterator>::value_type kpftimes::mean (ForwardIterator *first,* ForwardIterator *last*) throw (std::invalid_argument)

Finds the mean of the values in a generic container object.

The container class is accessed using first and last iterators, as in the C++ STL convention, and the variance is computed over the interval [first, last).

Template Parameters

ForwardIterator	The iterator type for the container over which the mean is to be calculated

Parameters

in	*first*	Forward iterator marking the first element in the container.
out	*last*	Forward iterator marking the position after the last element in the container.

Returns

The arithmetic mean of the elements between first, inclusive, and last, exclusive. The return type is that of the elements pointed to by the first and last iterators.

Precondition

first is "before" last in the sense that incrementing first repeatedly would reach last.

There is at least one element in the interval [first, last)

Exceptions

invalid_argument	Thrown if there are not enough elements.

Test

List of ints, length 0. Expected behavior: throw invalid_argument.
List of ints, length 1. Expected behavior: return list[0]
List of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_mean to within 1e-6 in 10 out of 10 trials.
Vector of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_mean to within 1e-6 in 10 out of 10 trials.
Array of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_mean to within 1e-6 in 10 out of 10 trials.

8.4.1.4 double kpftimes::pseudoNyquistFreq (const DoubleVec & *times*) throw (std::invalid_argument)

Returns the pseudo-Nyquist frequency for a grid of observations.

The pseudo-Nyquist frequency is defined as N/2T, where N is the number of observations and T is the length of the time interval covered by the data.

Parameters

in	*times*	Times at which data were taken

Returns

The pseudo-Nyquist frequency, in the inverse of whatever units times is in.

Precondition

times contains at least two unique values
times is sorted in ascending order

Exceptions

invalid_argument	Thrown if preconditions violated.

Performance:

O(times.size()) time

Test

Regular grid, length 1. Expected behavior: throws invalid_argument
Regular grid, length 2. Expected behavior: returns PNF = 1/(2∗step)

Regular grid, length 100. Expected behavior: returns PNF = 1/(2∗step)

Todo

Come up with test cases for an irregular grid.

8.4.1.5 template<class ForwardIterator > std::iterator_traits<ForwardIterator>::value_type kpftimes::variance (ForwardIterator *first*, ForwardIterator *last*) throw (std::invalid_argument)

Finds the variance of the values in a generic container object.

The container class is accessed using first and last iterators, as in the C++ STL convention, and the mean is computed over the interval [first, last).

Template Parameters

ForwardIterator	The iterator type for the container over which the variance is to be calculated

Parameters

in	*first*	Forward iterator marking the first element in the container.
out	*last*	Forward iterator marking the position after the last element in the container.

Returns

The (unbiased) sample variance of the elements between first, inclusive, and last, exclusive. The return type is that of the elements pointed to by the first and last iterators.

Precondition

first is "before" last in the sense that incrementing first repeatedly would reach last. There is at least two elements in the interval [first, last)

Exceptions

invalid_argument	Thrown if there are not enough elements.

Test

List of ints, length 1. Expected behavior: throw invalid_argument.
List of ints, length 2. Expected behavior: return lit[1]-list[0]
List of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_variance to within 1e-6 in 10 out of 10 trials.
Vector of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_variance to within 1e-6 in 10 out of 10 trials.

Array of ints, length 100, randomly generated. Expected behavior: agrees with gsl_stats_variance to within 1e-6 in 10 out of 10 trials.

9 Class Documentation

9.1 kpftimes::FastTable Class Reference

Two-dimensional dynamically allocated table.

Public Member Functions

- FastTable (size_t dimX, size_t dimY)

  Constructs but does not initialize a table of fixed size.
- FastTable (const FastTable &otherTable)

  Constructs an independent copy of the given object, containing identical data.
- FastTable & operator= (const FastTable &otherTable)

  Alters the existing object to be identical to otherTable.
- double & at (size_t x, size_t y)

  Access function to allow reads and writes of a table element.

9.1.1 Detailed Description

Two-dimensional dynamically allocated table.

The table is designed to allow fast (i.e. localized) scans along the Y axis. It has barely any other functionality.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 kpftimes::FastTable::FastTable (size_t *initX*, size_t *initY*)

Constructs but does not initialize a table of fixed size.

Parameters

in	*initX*	The first dimension of the table.
in	*initY*	The first dimension of the table.

Precondition

initX $>$ 0 and initY $>$ 0

Postcondition

The object represents an initX by initY table of values. The table elements are not constrained.

Exceptions

std::invalid_argument	Thrown if the table dimensions aren't valid.

Note

Once the FastTable object has been constructed, its dimensions cannot be changed.

Test

FastTable(-1, 1). Expected behavior: throw invalid_argument
FastTable(0, 1). Expected behavior: throw invalid_argument
FastTable(1, -1). Expected behavior: throw invalid_argument
FastTable(1, 0). Expected behavior: throw invalid_argument
FastTable(1, 1). Expected behavior: success. Can iterate over elements with external loop.
FastTable(5, 5). Expected behavior: success. Can iterate over elements with external loop.
FastTable(4, 5). Expected behavior: success. Can iterate over elements with external loop.
FastTable(5, 4). Expected behavior: success. Can iterate over elements with external loop.

9.1.2.2 kpftimes::FastTable::FastTable (const FastTable & *other*)

Constructs an independent copy of the given object, containing identical data.

Parameters

in	*other*	The object to copy.

Postcondition

The object is indistinguishable from other, but may be modified or deleted independently.

Test

FastTable(1, 1). Expected behavior: success. Can iterate over elements with external loop and edit independently.
FastTable(4, 5). Expected behavior: success. Can iterate over elements with external loop and edit independently.
FastTable(5, 4). Expected behavior: success. Can iterate over elements with external loop and edit independently.

9.1.3 Member Function Documentation

9.1.3.1 double & kpftimes::FastTable::at (size_t *x*, size_t *y*)

Access function to allow reads and writes of a table element.

Elements with the same x coordinate and adjacent y coordinates are adjacent in memory.

Parameters

in	*x*	The first-dimension coordinate to be read.
in	*y*	The second-dimension coordinate to be read.

Returns

An lvalued reference to the element Table[x, y]

Precondition

0 <= x < dimX
0 <= y < dimY

Warning

In the interests of speed-at-all-costs, preconditions are *not* checked.

Test

FastTable(4, 5).at(0, 2). Expected behavior: update reflected by separate .at().
FastTable(4, 5).at(3, 2). Expected behavior: update reflected by separate .at().
FastTable(4, 5).at(2, 0). Expected behavior: update reflected by separate .at().
FastTable(4, 5).at(2, 4). Expected behavior: update reflected by separate .at().
FastTable(1000, 1000).at(x,y) = x+y±x∗y iteration in both directions. Expected behavior: y-inner loop faster than x-inner loop for 10 out of 10 objects.

9.1.3.2 kpftimes::FastTable & FastTable::operator= (const FastTable & *other*)

Alters the existing object to be identical to otherTable.

Parameters

in	*other*	The object to copy.

Precondition

This object has the same dimensions as other.

Postcondition

The object is indistinguishable from other, but may be modified or deleted independently.

Exceptions

domain_error	Thrown if the two tables have mismatched dimensions.

Test

FastTable(1, 1) = FastTable(1, 1). Expected behavior: success. Can iterate over elements with external loop and edit independently.
FastTable(4, 5) = FastTable(4, 5). Expected behavior: success. Can iterate over elements with external loop and edit independently.
FastTable(4, 5) = FastTable(5, 5). Expected behavior: throw domain_error.
FastTable(5, 4) = FastTAble(5, 5). Expected behavior: throw domain_error.

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

- utils.h
- utils.cpp

10 File Documentation

10.1 autocorr.cpp File Reference

Autocorrelation function for unevenly sampled data.

10.1.1 Detailed Description

Autocorrelation function for unevenly sampled data.

Author

Krzysztof Findeisen

Date

Created February 16, 2011
Last modified April 15, 2011

10.2 dft.cpp File Reference

Implements the irregularly-sampled discrete Fourier transform.

10.2.1 Detailed Description

Implements the irregularly-sampled discrete Fourier transform.

Author

Krzysztof Findeisen

Date

Created February 13, 2011
Last modified April 14, 2011

10.3 dft.h File Reference

Computes the irregularly-sampled discrete Fourier transform.

Typedefs

- typedef std::vector< double > DoubleVec

  A convenient shorthand for vectors of doubles.
- typedef std::vector< std::complex< double > > ComplexVec

  A convenient shorthand for vectors of complex numbers.

Functions

- void kpftimes::dft (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &freqs, ComplexVec &dft)

  Calculates the discrete Fourier transform for a list of times and fluxes.

10.3.1 Detailed Description

Computes the irregularly-sampled discrete Fourier transform.

Author

Krzysztof Findeisen

Date

Created February 13, 2011
Last modified April 14, 2011

10.4 freqgen.cpp File Reference

Frequency generators for Scargle functions.

10.4.1 Detailed Description

Frequency generators for Scargle functions.

Author

Krzysztof Findeisen

Date

Created February 16, 2011
Last modified April 13, 2011

10.5 scargle.cpp File Reference

Computes the Lomb-Scargle periodogram of an unevenly sampled lightcurve.

Defines

- #define SCARGLE_SLOW 0

  A compiler flag controlling the strategy used for lsThreshold().
- #define **PI** 3.1415927

10.5.1 Detailed Description

Computes the Lomb-Scargle periodogram of an unevenly sampled lightcurve.

Author

Krzysztof Findeisen

Date

Derived from scargle.pro (by Joern Wilms et al.) January 25, 2010
Last modified April 13, 2011

10.6 specialfreqs.cpp File Reference

Computes characteristic frequencies of a sampling cadence.

10.6.1 Detailed Description

Computes characteristic frequencies of a sampling cadence.

Author

Krzysztof Findeisen

Date

Created April 13, 2011
Last modified April 13, 2011

10.7 tests/driver.cpp File Reference

Test code for Timescales.

Functions

- void runLsNormalEdfTests ()

  Runs all test cases defined for lsNormalEdf().
- void runAutoCorrTests ()

  Runs all test cases defined for autoCorr().
- int main ()

  Runs all test cases----------.

10.7.1 Detailed Description

Test code for Timescales.

Author

Krzysztof Findeisen

Date

Created May 23, 2011
Last modified May 23, 2011

10.7.2 Function Documentation

10.7.2.1 int main ( )

Runs all test cases----------.

Returns

Status code 0

10.7.2.2 void runAutoCorrTests ( )

Runs all test cases defined for autoCorr().

Results are printed to stdout.

10.7.2.3 void runLsNormalEdfTests ( )

Runs all test cases defined for lsNormalEdf().

Results are printed to stdout.

10.8 tests/unit_autoCorr.cpp File Reference

Performs unit testing of the function kpftimes::autoCorr()

Functions

- bool testAcf (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, bool expectInvalid, const std::string &outFile) throw ()

  Carries out a generic test of the autoCorr() function.
- void runAutoCorrTests ()

  Runs all test cases defined for autoCorr().

10.8.1 Detailed Description

Performs unit testing of the function kpftimes::autoCorr()

Author

Krzysztof Findeisen

Date

Created July 15, 2011
Last modified July 15, 2011

10.8.2 Function Documentation

10.8.2.1 void runAutoCorrTests ( )

Runs all test cases defined for autoCorr().

Results are printed to stdout.

10.8.2.2 bool testAcf (const DoubleVec & *times,* const DoubleVec & *fluxes,* const DoubleVec & *offsets,* bool *expectInvalid,* const std::string & *outFile*) throw ()

Carries out a generic test of the autoCorr() function.

Parameters

in	*times*	The times array to be passed to autoCorr().
in	*fluxes*	The flux array to be passed to autoCorr().
in	*offsets*	The offset array to be passed to autoCorr().
in	*expectInvalid*	Whether the correct behavior is throwing a \ invalid_argument exception.
in	*outFile*	The name of the file to which to print the ACF. No printing if empty string.

Returns

true if test passed, false if test failed

Warning

: missing automated oracle for ACFs

10.9 tests/unit_lsNormalEdf.cpp File Reference

Performs unit testing of the function kpftimes::lsNormalEdf()

Functions

- bool testEdf (const DoubleVec &times, const DoubleVec &freqs, int nSims, bool expectInvalid, bool expectDomain) throw ()

  Carries out a generic test of the lsNormalEdf() function.
- void runLsNormalEdfTests ()

  Runs all test cases defined for lsNormalEdf().

10.9.1 Detailed Description

Performs unit testing of the function kpftimes::lsNormalEdf()

Author

Krzysztof Findeisen

Date

Created May 19, 2011
Last modified May 24, 2011

10.9.2 Function Documentation

10.9.2.1 void runLsNormalEdfTests ( )

Runs all test cases defined for lsNormalEdf().

Results are printed to stdout.

10.9.2.2 bool testEdf (const DoubleVec & *times*, const DoubleVec & *freqs*, int *nSims*, bool *expectInvalid*, bool *expectDomain*) throw ()

Carries out a generic test of the lsNormalEdf() function.

Parameters

in	*times*	The times array to be passed to lsNormalEdf().
in	*freqs*	The frequency array to be passed to lsNormalEdf().
in	*nSims*	The number of simulations to request of lsNormalEdf().
in	*expectInvalid*	Whether the correct behavior is throwing a \ invalid_argument exception. May be set simultaneously with expectDomain, in which case catching either invalid_argument or domain_behavior is considered correct behavior.
in	*expectDomain*	Whether the correct behavior is throwing a \ domain_error exception. May be set simultaneously with expectInvalid, in which case catching either invalid_argument or domain_behavior is considered correct behavior.

Returns

true if test passed, false if test failed

Compare highestPeak to powers in a statistical sense. For now, do this by visual inspection of plots. Eventually we want to implement a KS or (preferably) AD test to automate the comparison. GSL doesn't have either, believe it or not.

Warning

: missing automated oracle for EDFs

10.10 timescales.h File Reference

Primary header for Krzysztof's timescales library.

Typedefs

- typedef std::vector< double > DoubleVec

  A convenient shorthand for vectors of doubles.

Functions

- void kpftimes::lombScargle (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &freq, DoubleVec &power) throw (std::invalid_argument, std::domain_error)

  Calculates the Lomb-Scargle periodogram for a time series.
- double kpftimes::lsThreshold (const DoubleVec &times, const DoubleVec &freq, double fap, long nSims) throw (std::invalid_argument, std::domain_error)

  Calculates the significance threshold for a Lomb-Scargle periodogram.
- void kpftimes::lsNormalEdf (const DoubleVec &times, const DoubleVec &freqs, DoubleVec &powers, DoubleVec &probs, long nSims) throw (std::invalid_argument, std::domain_error)

  Calculates the empirical distribution function of false peaks for a Lomb-Scargle periodogram.
- void kpftimes::autoCorr (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, DoubleVec &acf) throw (std::invalid_argument)

  Calculates the autocorrelation function for a time series.
- void kpftimes::autoCorr (const DoubleVec &times, const DoubleVec &fluxes, const DoubleVec &offsets, DoubleVec &acf, double maxFreq) throw (std::invalid_argument)

  Calculates the autocorrelation function for a time series.
- void kpftimes::acWindow (const DoubleVec &times, const DoubleVec &offsets, DoubleVec &wf) throw (std::invalid_argument)

  Calculates the autocorrelation window function for a time sampling.
- void kpftimes::acWindow (const DoubleVec &times, const DoubleVec &offsets, DoubleVec &wf, double maxFreq) throw (std::invalid_argument)

  Calculates the autocorrelation window function for a time sampling.
- double kpftimes::deltaT (const DoubleVec &times) throw (std::invalid_argument)

  Returns the time interval covered by the data.
- double kpftimes::pseudoNyquistFreq (const DoubleVec &times) throw (std::invalid_argument)

  Returns the pseudo-Nyquist frequency for a grid of observations.
- double kpftimes::maxFreq (const DoubleVec &times) throw (std::invalid_argument)

  Returns the highest frequency that can be probed by the data.

- void kpftimes::freqGen (const DoubleVec &times, DoubleVec &freq) throw (std::invalid_argument)

  Creates a frequency grid that can be fed to time series analysis functions.
- void kpftimes::freqGen (const DoubleVec &times, DoubleVec &freq, double fMin, double fMax) throw (std::invalid_argument)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void kpftimes::freqGen (const DoubleVec &times, DoubleVec &freq, double fMin, double fMax, double fStep) throw (std::invalid_argument)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

10.10.1 Detailed Description

Primary header for Krzysztof's timescales library.

Author

Krzysztof Findeisen

Date

Created January 25, 2010
Last modified April 13, 2011

10.11 utils.cpp File Reference

Implements support code for the library.

10.11.1 Detailed Description

Implements support code for the library. None of these routines are intended as part of the public API.

Author

Krzysztof Findeisen

Date

Created April 13, 2011
Last modified April 13, 2011

10.12 utils.h File Reference

Support code for the library.

Classes

- class kpftimes::FastTable

  Two-dimensional dynamically allocated table.

Typedefs

- typedef std::vector< double > DoubleVec

  A convenient shorthand for vectors of doubles.

Functions

- bool kpftimes::isSortedAsc (const DoubleVec &list)

  Tests whether the argument is sorted.

- template<class ForwardIterator >
  std::iterator_traits< ForwardIterator >::value_type kpftimes::mean (ForwardIterator first, ForwardIterator last) throw (std::invalid_argument)

  Finds the mean of the values in a generic container object.

- template<class ForwardIterator >
  std::iterator_traits< ForwardIterator >::value_type kpftimes::variance (ForwardIterator first, ForwardIterator last) throw (std::invalid_argument)

  Finds the variance of the values in a generic container object.

10.12.1 Detailed Description

Support code for the library. None of these routines are intended as part of the public API.

Author

Krzysztof Findeisen

Date

Created April 13, 2011
Last modified April 13, 2011