libfbm

0.3

Generated by Doxygen 1.8.1

Fri Sep 27 2013 00:53:58

Contents

1 Main Page

Author

Indrek Mandre indrek(at)mare.ee http://www.mare.ee/indrek/libfbm/

This library generates (simulates) multi-dimensional random fields:

- stationary Gaussian processes (SGP), with a user given covariance function
- fractional Brownian motion (fBm)
- random fields with power spectrum as power law

Stationary means here that the covariance between two points does not depend on the individual coordinates of the points:

$$\mathrm{Cov}(Z(\mathbf{x}), Z(\mathbf{y})) \equiv \rho(\mathbf{x}-\mathbf{y}),$$

so that only the vector $x-y$ is used. Gaussian means that the distribution of individual values in the field is gaussian and the process can be completely described by its mean and covariance matrix, that is when

$$[Z(\mathbf{x}_1), \ldots, Z(\mathbf{x}_m)]^T \sim \mathcal{N}(\mu, \Sigma)$$

We say the SGP is isotropic, if the covariance depends only on the distance between the two points, that is

$$\mathrm{Cov}(Z(\mathbf{x}), Z(\mathbf{y})) \equiv \rho(|\mathbf{x}-\mathbf{y}|).$$

We say the SGP is even, if the covariance function has the same value in case of reflection in one dimension, that is when:

$$\mathrm{Cov}(Z(\mathbf{x}), Z(\mathbf{y})) \equiv \rho(|x_1-y_1|, \ldots, |x_d-y_d|).$$

Library supports arbitrary-dimensional generation, that is 1D, 2D, 3D, 4D, etc. However, there is a hardcoded limit of 8 (can be changed at library compile time).

For a random engine the library makes use of SFMT [8]. Normal distribution is achieved with an efficient ziggurat algorithm (rejection sampling). Emphasis of the library is on fast bulk-generation of random fields.

With the help of some post-processing also fractional Brownian motion or surfaces (or higher-dimensional fields) can be generated. For the one-dimensional (1D) case this can be done simply by generating the increment process (fractional Gaussian noise), and summing elements to get the fBm itself. For higher-dimensional cases a more involved process is used by Stein [3]. Note that Stein's definition of fBm is different compared to the traditional (strict mathematical) definition – fields generated with his methods also adhere to this change.

Calculation of a SGP involves two N-dimensional fast Fourier transforms. One of them is pre-calculated and can be cached on the hard disk to save memory. For this reason the first invocation of generate() on a given field object may be at least twice as slow as the following calls (unless already cached on HD). The library is meant to be used in Monte Carlo simulations where thousands or millions of field instances are required.

The first Fourier transform results in eigenvalues. For the algorithm to work these must be positive. Sometimes, due to numeric errors or the nature of cov() (the embedded covariance matrix is not positive semi-definite), they can come out negative. When this happens the library complains to stderr. Here are some hints on how to overcome this:

- from [2]: use of a nugget effect. Add a small value (say 1e-12) to Cov(0) to avoid numerical instabilities that result in negative eigenvalues.
- from [1] and [2]: increase field size (or reduce the scaling of the Cov() function) to make the matrix positive-semidefinite.
- from [1]: if there are only a few negative eigenvalues, the library sets them to 0 and scales the other values to compensate. However, I've not tested how well this works – this is bound to introduce distortions.
- for the Stein's fBm method, increase the value of R. Note that this comes at the expense of increased memory usage and computation time. Conversely, reducing R to minimum that "works" will result in reduced memory usage and computational expense.
- Stein's fBm method [3] uses molding of the isotropic covariance function at edges to make it positive semi-definite. At the expense of losing some space at the edges one can still embed a given isotropic covariance function and save a lot of computational time as opposed to idea in tip 2 (simply increasing the field size which sometimes is not feasible). See [3] for more details.
- use a highly-composite or power-of-two for field sizes as the FFT functions are more efficient with these (both space and time).
- use common sense, test stuff out, fiddle with parameters. This library is no silver bullet. Make sure what you get is what you wanted.

The used methods in this library are derived from [1], [2] and [3]. Also [5] contains useful information and implementation for matlab/octave. Note that these methods are mathematically "exact". That is they are not approximations. Still I should add that here by default we use a pseudo-random random number generataror, which is not "exact". And there are numeric errors that may creep up whenever calculations approach zero or infinity.

The library depends on the FFTW3 library for FFT functions. You can get it from http://www.fftw.org/ but most Linux distribution have a native package for easy installation.

I haven't really tested the library that much for non-isotropic or un-even covariance functions. Hopefully, it works properly. If you run into trouble you can use SGPTester to test that the generated process has the expected covariances.

References:
[1] Wood, A. T. A. and Chan, G.
Simulation of Stationary Gaussian Processes in [0, 1]^d.
Journal of Computational and Graphical Statistics, 3(4), 409-432 (1994).
doi:10.1080/10618600.1994.10474655
[2] Dietrich, C. R. and Newsam, G. N.
Fast and Exact Simulation of Stationary Gaussian Processes through
Circulant Embedding of the Covariance Matrix.
SIAM Journal on Scientific Computing, 18(4), 1088-1107 (1997).
doi:10.1137/S1064827592240555
[3] Stein, M. L.
Fast and Exact Simulation of Fractional Brownian Surfaces.
Journal of Computational and Graphical Statistics, 11(3), 587-599 (2002).
doi:10.1198/106186002466
[4] Qian, H., Raymond, G. M. and Bassingthwaighte, J. B.
On two-dimensional fractional Brownian motion and fractional Brownian random field.
J. Phys. A: Math. Gen., 31, L527-L535 (1998).
doi:10.1088/0305-4470/31/28/002
[5] Kroese, D. P. and Botev, I. Z.
Spatial Process Generation.
http://www.maths.uq.edu.au/~kroese/ps/MCSpatial.pdf
[6] Timmer, J. and König, M.
On generating power law noise.
Astronomy and Astrophysics 300: 707 (1995).
[7] Marsaglia, G. and Tsang, W. W.
The Ziggurat Method for Generating Random Variables.
Journal of Statistical Software, 5(8), 1-7 (2000).
[8] Saito, M. and Matsumoto, M.
SIMD-Oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator
Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer, 607-622 (2008).
doi:10.1007/978-3-540-85912-3_26

Example of generating 1D fractional Brownian motion:

```
#include <stdio.h>
#include <libfbm.hpp>

int main()
{
  // we want one-dimensional (1D) fractional Brownian motion of size 1024
  libfbm::zvec dim(1);
  dim[0] = 1024;
  // create context for 1D fBm with exponent H=0.75
  libfbm::FWSContext ctx(0.75, dim);
  // set random generator seed, optional
  libfbm::gaussianRandomGenerator.setSeed
      (1);
  // set caching path, optional
  ctx.setCacheDir("/tmp/fbm");
  // create the field
  libfbm::Field fbm(ctx, true);
  // allocates memory and generates the fBm
  fbm.generate();
  // print it out
  for ( int i = 0; i < dim[0]; i++ )
      printf("%g\n", fbm(i));
```

```
  // generate another version of the fBm
  fbm.generate();
  // ..
  return 0;
}
```

Example of generating 2D fractional Brownian surfaces:

```
#include <stdio.h>
#include <libfbm.hpp>

int main()
{
  // create context for 2D fBs with exponent H=0.75 and size 64x64
  libfbm::FBMSteinContext ctx(0.75, 2, 64);
  // set random generator seed, optional
  libfbm::gaussianRandomGenerator.setSeed
      (1);
  // set caching path, optional
  ctx.setCacheDir("/tmp/fbm");
  // create the field
  libfbm::Field fbm(ctx, true);
  // allocates memory and generates the fBm
  fbm.generate();
  // print it out
  for ( int y = 0; y < fbm.getDim()[1]; y++ )
    {
      for ( int x = 0; x < fbm.getDim()[0]; x++ )
          printf("%s%g", x ? " " : "", fbm(x, y));
      printf("\n");
    }
  // generate another version of the fBm
  fbm.generate();
  // ..
  return 0;
}
```

Example of generating 2D random process with custom covariance function:

```
#include <stdio.h>
#include <math.h>
#include <libfbm.hpp>

// our own exponential unisotropic covariance function:
struct MyContext : public libfbm::SGPContext
{
  MyContext(const libfbm::zvec& dim) : libfbm::SGPContext(dim, dim,
       "myctx") { }

  double cov(const libfbm::zvec& r)
  {
    double xp = 0;
    for ( size_t i = 0; i < r.size(); i++ )
        xp += -fabs(r[i]) / (5 + i * 17);
    return exp(xp);
  }
};

int main()
{
  libfbm::zvec dim(2);
  dim[0] = 64;
  dim[1] = 64;
  MyContext ctx(dim);
  libfbm::Field fbm(ctx, true);
  fbm.generate();
  for ( int y = 0; y < dim[1]; y++ )
    {
      for ( int x = 0; x < dim[0]; x++ )
          printf("%s%g", x ? " " : "", fbm(x, y));
      printf("\n");
    }
  return 0;
}
```

And here's some legalese, the copyright notice for libfbm:

```
Copyright (c) 2013, Indrek Mandre <indrek(at)mare.ee>
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

a Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.
b Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Note that this software also uses FFTW, which is under the GPL. It should be trivial to make changes to use a different FFT stack.

Now despite the claims before, this software is not actually free. In case you used it in research resulting in a published paper, or a thesis you defended, you must send me a book. One book per paper. Any book will do. It can be used. I do personally enjoy fiction.

Take care and have fun!!

Indrek Mandre - indrek(at)mare.ee - Institute of Cybernetics at the Tallinn University of Technology - Akadeemia tee 21, 12618, Tallinn, Estonia, EU

This library incorporates SFMT, with the following license:

```
Copyright (c) 2006,2007 Mutsuo Saito, Makoto Matsumoto and Hiroshima
University.
Copyright (c) 2012 Mutsuo Saito, Makoto Matsumoto, Hiroshima University
and The University of Tokyo.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

    a Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    b Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.
    c Neither the names of Hiroshima University, The University of
      Tokyo nor the names of its contributors may be used to endorse
      or promote products derived from this software without specific
      prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

```
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Note that the authors of SFMT have nothing to do with libfbm.

TODO:

- Maybe differentiate between dimension, dimensionality and size. Introduce rank? Quite a bit confusion in the API due to this right now. Bad indrek.
- SIMD optimization for the Gaussian number generation.

2 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

3 Class Index

3.1 Class Hierarchy

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

4 Class Index

4.1 Class List

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

5 Namespace Documentation

5.1 libfbm Namespace Reference

Classes

- class GaussianGenerator

  Normally distributed random number generator interface.
- class SFMTGaussianGenerator

  SIMD-oriented Fast Mersenne Twister (SFMT) and Ziggurat sampling based gaussian random generator.
- class Seeder

Seed generator for random number generators.

- class zvec

  Integer vector.
- class SGPContext

  Abstract stationary Gaussian process context.
- class FGNContext

  Fractional gaussian noise context.
- class FWSContext

  Fractional wiener surface context.
- class AbstractField

  Base class for Field and PLFPSField.
- class Field

  Gaussian process field.
- class SGPTester

  Class for testing the covariance of generated fields.
- class FBMSteinContext

  Stein's fractional Brownian motion context.
- class PowerLawContext

  Power law covariance context.
- class PLFPSField

  Power-law power spectrum field.

Variables

- SFMTGaussianGenerator gaussianRandomGenerator

  Global random generator used by default in Field::generate().

5.1.1 Variable Documentation

5.1.1.1 SFMTGaussianGenerator libfbm::gaussianRandomGenerator

Global random generator used by default in Field::generate().

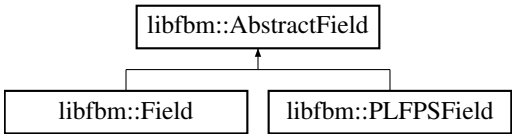
6 Class Documentation

6.1 libfbm::AbstractField Class Reference

Base class for Field and PLFPSField.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::AbstractField:



Public Member Functions

- AbstractField ()
- virtual ~AbstractField ()
- virtual void generate ()=0

  Generate the field.
- virtual void generate (GaussianGenerator &rng)=0

  Generate the field using random generator rng.
- virtual void clear ()=0

  Release all memory this object holds.
- virtual const zvec & getDim () const =0

  Get the usable dimensions of this field.
- double operator() (const zvec &p) const

  Get field value at the given point.
- double operator() (size_t x) const
- double operator() (size_t x, size_t y) const
- double operator() (size_t x, size_t y, size_t z) const
- double operator() (size_t x, size_t y, size_t z, size_t u) const
- double operator() (size_t x, size_t y, size_t z, size_t u, size_t v) const
- double & at (const zvec &p)

  Return a reference to the internal array element at point p.
- const double & at (const zvec &p) const
- const size_t * getStrides () const

  Get the strides array.

Protected Attributes

- std::vector< double > Z
- double * datap
- size_t muls [LIBFBM_MAX_DIM]

6.1.1 Detailed Description

Base class for Field and PLFPSField.

Definition at line 719 of file libfbm.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 libfbm::AbstractField::AbstractField ( ) [inline]

Definition at line 722 of file libfbm.hpp.

6.1.2.2 virtual libfbm::AbstractField::~AbstractField ( ) [virtual]

6.1.3 Member Function Documentation

6.1.3.1 double& libfbm::AbstractField::at (const zvec & *p*) [inline]

Return a reference to the internal array element at point *p*.

Reimplemented in libfbm::Field.

Definition at line 758 of file libfbm.hpp.

6.1.3.2 const double& libfbm::AbstractField::at (const zvec & *p*) const [inline]

Reimplemented in libfbm::Field.

Definition at line 766 of file libfbm.hpp.

6.1.3.3 virtual void libfbm::AbstractField::clear ( ) [pure virtual]

Release all memory this object holds.

Implemented in libfbm::PLFPSField, and libfbm::Field.

6.1.3.4 virtual void libfbm::AbstractField::generate ( ) [pure virtual]

Generate the field.

Implemented in libfbm::PLFPSField, and libfbm::Field.

6.1.3.5 virtual void libfbm::AbstractField::generate (GaussianGenerator & *rng*) [pure virtual]

Generate the field using random generator rng.

The random generator should produce normally distributed numbers.

Implemented in libfbm::PLFPSField, and libfbm::Field.

6.1.3.6 virtual const zvec& libfbm::AbstractField::getDim ( ) const [pure virtual]

Get the usable dimensions of this field.

Implemented in libfbm::PLFPSField, and libfbm::Field.

6.1.3.7 const size_t* libfbm::AbstractField::getStrides ( ) const [inline]

Get the strides array.

Internally, the data is kept in a huge single array of doubles. You can get addresses into this array using the at() member functions. If you want to quickly move to the next or previous element in a given dimension d, you can add or substract the strides[d] value to the address.

Reimplemented in libfbm::Field.

Definition at line 779 of file libfbm.hpp.

6.1.3.8 double libfbm::AbstractField::operator() (const zvec & *p*) const [inline]

Get field value at the given point.

Note that you are responsible for correct point indices. No checking is done internally and invalid values can lead to crashes.

Reimplemented in libfbm::Field.

Definition at line 739 of file libfbm.hpp.

6.1.3.9 double libfbm::AbstractField::operator() (size_t *x*) const [inline]

Reimplemented in libfbm::Field.

Definition at line 746 of file libfbm.hpp.

6.1.3.10 double libfbm::AbstractField::operator() (size_t *x*, size_t *y*) const [inline]

Reimplemented in libfbm::Field.

Definition at line 748 of file libfbm.hpp.

6.1.3.11 double libfbm::AbstractField::operator() (size_t x, size_t y, size_t z) const [inline]

Reimplemented in libfbm::Field.

Definition at line 750 of file libfbm.hpp.

6.1.3.12 double libfbm::AbstractField::operator() (size_t x, size_t y, size_t z, size_t u) const [inline]

Reimplemented in libfbm::Field.

Definition at line 752 of file libfbm.hpp.

6.1.3.13 double libfbm::AbstractField::operator() (size_t x, size_t y, size_t z, size_t u, size_t v) const [inline]

Reimplemented in libfbm::Field.

Definition at line 754 of file libfbm.hpp.

6.1.4 Member Data Documentation

6.1.4.1 double* libfbm::AbstractField::datap [protected]

Definition at line 783 of file libfbm.hpp.

6.1.4.2 size_t libfbm::AbstractField::muls[LIBFBM_MAX_DIM] [protected]

Definition at line 784 of file libfbm.hpp.

6.1.4.3 std::vector<double> libfbm::AbstractField::Z [protected]

Definition at line 782 of file libfbm.hpp.

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

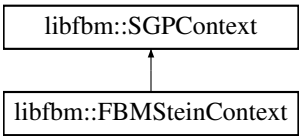
- libfbm.hpp

6.2 libfbm::FBMSteinContext Class Reference

Stein's fractional Brownian motion context.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::FBMSteinContext:



Public Member Functions

- FBMSteinContext (double H, size_t dim, size_t size, double Rhint=-1, bool mapDim=true)

  Construct the Stein's FBM context object of size size^dim.
- double cov (const zvec &zvec)

  Covariance function.
- void disablePostProcessing ()
- double getR () const

  Get the parameter R described in [3].

Static Public Member Functions

- static size_t userSize2FieldSize (size_t size, double R)

  Get the required field size for the given usable user size and R.
- static size_t fieldSize2UserSize (size_t size, double R)

  Map the field dimension into the resulting usable fBm field dimension.
- static double getRForH (double H, size_t dim, double Rhint=-1)

  Get the precomputed R value for the given H.

Protected Member Functions

- void postProcess (Field &field, GaussianGenerator &rng)

  Postprocessor called after field generation.

6.2.1 Detailed Description

Stein's fractional Brownian motion context.

Stein [3] defines fBm as a random process Z with a power-law increment variance:

$$\mathrm{Var}\left(|Z(\mathbf{x})-Z(\mathbf{y})|^2\right) \propto |\mathbf{x}-\mathbf{y}|^{2H},$$

however, the covariance does not match that of a "standard" fBm. Indeed, for Stein it is actually

$$\mathrm{Cov}(Z(\mathbf{x}),Z(\mathbf{y})) = c0 - |\mathbf{x}-\mathbf{y}|^{2H} + c_2\left(|\mathbf{x}|^2+|\mathbf{y}^2|\right).$$

The FBMSteinContext changes the actual genereated field dimension. However, only part of it, that is the dimension given at construction is usable, the rest is used for the Stein's embedding method. This means the getDim() and getFieldDim() may have different values. The Field::getDim() reflects the usable area.

Definition at line 928 of file libfbm.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 libfbm::FBMSteinContext::FBMSteinContext (double *H*, size_t *dim*, size_t *size*, double *Rhint* = -1, bool *mapDim* = true)

Construct the Stein's FBM context object of size size^dim.

Parameters

H	The Hurst exponent.
dim	Dimensionality of the field wanted (1D, 2D, 3D, etc.).
size	Request field of size^dim.
Rhint	Hint that overrides the self-determined R (see getRforH()). See [3] for details but in general 1<=R<=2. Larger R wastes more space and takes more time to calculate; smaller R may result in negative eigenvalues and a failed calculation. So one strives towards smallest R possible that works (no negative eigenvalues). As field size is increased (or field dimension), R must also increase. If left negative, the program attempts to estimate the value based on parameter H and dimensionality, but this estimate is conservative.
mapDim	Expand dimension so that the resulting usable field is the size requested. True by default.

The automatic R detections seems to work well for 2D case. For higher-dimensional cases you may have to fiddle with it manually.

6.2.3 Member Function Documentation

6.2.3.1 double libfbm::FBMSteinContext::cov (const zvec & *p*) [virtual]

Covariance function.

Implements libfbm::SGPContext.

6.2.3.2 void libfbm::FBMSteinContext::disablePostProcessing ( )

6.2.3.3 static size_t libfbm::FBMSteinContext::fieldSize2UserSize (size_t *size*, double *R*) [static]

Map the field dimension into the resulting usable fBm field dimension.

6.2.3.4 double libfbm::FBMSteinContext::getR ( ) const [inline]

Get the parameter *R* described in [3].

Definition at line 958 of file libfbm.hpp.

6.2.3.5 static double libfbm::FBMSteinContext::getRForH (double *H*, size_t *dim*, double *Rhint* = -1) [static]

Get the precomputed R value for the given H.

This function is quantized, so is not optimal. Also R depends a bit on the size of the field. This here is conservative. However, it could be it fails for some cases, then you need to provide your own correct R. The returned value is a very conservative 2.0 when dim >= 3, so for 3D and higher-dimensional cases it makes sense to find your own optimal R value.

Parameters

Rhint	If equal or greater than 1, return Rhint.

6.2.3.6 void libfbm::FBMSteinContext::postProcess (Field & *field*, GaussianGenerator & *rng*) [protected], [virtual]

Postprocessor called after field generation.

This is called automatically by Field::generate().

Reimplemented from libfbm::SGPContext.

6.2.3.7 static size_t libfbm::FBMSteinContext::userSize2FieldSize (size_t *size*, double *R*) [static]

Get the required field size for the given usable user *size* and *R*.

Stein's method [2] requires a larger field to generate a field for the required initial dimensions. This size depends on the chosen R. For 2D and H <= 0.75, R is always 1.0. For H > 0.75, R must be increased. It is proven that the methdod works with R=2.0. However, in practice R=1.3 at H=0.999 works just fine.

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

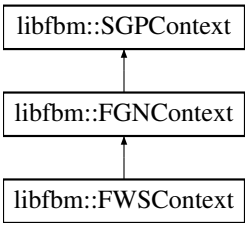
- libfbm.hpp

6.3 libfbm::FGNContext Class Reference

Fractional gaussian noise context.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::FGNContext:



Public Member Functions

- FGNContext (double H, const zvec &dim)
- double cov (const zvec &zvec)

  Covariance function.

Additional Inherited Members

6.3.1 Detailed Description

Fractional gaussian noise context.

The covariance function is

$$\rho(\mathbf{r}) = \prod_{i=1}^{d} \frac{1}{2}\left[|r_i - 1|^{2H} - 2|r_i|^{2H} + |r_i + 1|^{2H}\right],$$

where $0 < H < 1$. See [4]. For 1D cases the generated process is the increment process of fractional Brownian motion.

Definition at line 692 of file libfbm.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 libfbm::FGNContext::FGNContext (double *H,* const zvec & *dim*)

6.3.3 Member Function Documentation

6.3.3.1 double libfbm::FGNContext::cov (const zvec & *p*) [virtual]

Covariance function.

Implements libfbm::SGPContext.

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

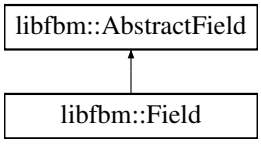
- libfbm.hpp

6.4 libfbm::Field Class Reference

Gaussian process field.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::Field:



Public Member Functions

- Field (SGPContext &context, bool allowCorrelated=false)

  Construct the field using given context.
- ~Field ()
- void setBufferSize (size_t bufferSize)

  Set the internal buffer size used when reading cached precomputed data from the HD.
- void generate ()

  Generate the next random process.
- void generate (GaussianGenerator &rng)

  Generate the field using random generator rng.
- void clear ()

  Release all memory this object holds.
- double operator() (const zvec &p) const

  Access field value at given point.
- double operator() (size_t x) const
- double operator() (size_t x, size_t y) const
- double operator() (size_t x, size_t y, size_t z) const
- double operator() (size_t x, size_t y, size_t z, size_t u) const
- double operator() (size_t x, size_t y, size_t z, size_t u, size_t v) const
- void integrate ()

  Sum the generated field in each dimension.
- const zvec & getDim () const

  Get the usable dimensions of this field.
- double & at (const zvec &p)

  Return a reference to the internal array element at point p.
- const double & at (const zvec &p) const
- const size_t * getStrides () const

  Get the strides array.

Additional Inherited Members

6.4.1 Detailed Description

Gaussian process field.

This class holds the resulting data. Note that when accessing the data you must be sure the indexes are correct as no internal checking is done here (the same as for a general C/C++ array/vector).

Definition at line 796 of file libfbm.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 libfbm::Field::Field (SGPContext & *context*, bool *allowCorrelated* = `false`)

Construct the field using given context.

Actual memory allocation is done at generate(). At each generate() two groups of random fields are generated, each group containing 2^d cross-correlated processes. So at each generate() we get exactly two uncorrelated random processes. One of them is cached and switched in at the next call to generate(). However, if you want to also use the correlated fields, set *allowCorrelated* to *true*. So for 2D we get 4 instances, for 3D 8 instances, etc. Note that the field holds onto the context object pointer, so it must stick around until the field is destroyed.

6.4.2.2 libfbm::Field::~Field ( )

6.4.3 Member Function Documentation

6.4.3.1 double& libfbm::Field::at (const zvec & *p*) [inline]

Return a reference to the internal array element at point *p*.

Reimplemented from libfbm::AbstractField.

Definition at line 870 of file libfbm.hpp.

6.4.3.2 const double& libfbm::Field::at (const zvec & *p*) const [inline]

Reimplemented from libfbm::AbstractField.

Definition at line 878 of file libfbm.hpp.

6.4.3.3 void libfbm::Field::clear ( ) [virtual]

Release all memory this object holds.

Implements libfbm::AbstractField.

6.4.3.4 void libfbm::Field::generate ( ) [virtual]

Generate the next random process.

At first calling this function allocates the necessary memory. Note that multiple instances of random processes are generated at one calculation, which are quickly pulled in at next call to generate(). See the constructor for more information. The memory allocation size is 2∗(2∗d1)∗(2∗d2)∗...∗(2∗dn) doubles, where d1..dn are the dimensions of the generated process. Note that the precomputed eigenvalue cache size (if kept in memory) additionally adds half of that.

Implements libfbm::AbstractField.

6.4.3.5 void libfbm::Field::generate (GaussianGenerator & *rng*) [virtual]

Generate the field using random generator rng.

The random generator should produce normally distributed numbers.

Implements libfbm::AbstractField.

6.4.3.6 const zvec& libfbm::Field::getDim ( ) const [inline], [virtual]

Get the usable dimensions of this field.

Implements libfbm::AbstractField.

Definition at line 867 of file libfbm.hpp.

6.4.3.7 const size_t∗ libfbm::Field::getStrides ( ) const [inline]

Get the strides array.

Internally, the data is kept in a huge single array of doubles. You can get addresses into this array using the at() member functions. If you want to quickly move to the next or previous element in a given dimension d, you can add or substract the strides[d] value to the address.

Reimplemented from libfbm::AbstractField.

Definition at line 891 of file libfbm.hpp.

6.4.3.8 void libfbm::Field::integrate ( )

Sum the generated field in each dimension.

Can be used to generate 1D fractional Brownian motion from the 1D fractional Gaussian noise. For 2D case a strange fractional Wiener surface is produced (not very useful). Note that FWSContext calls this automatically.

6.4.3.9 double libfbm::Field::operator() (const zvec & *p*) const [inline]

Access field value at given point.

Note that you are responsible for correct point indices. No checking is done internally and invalid values can lead to crashes.

Reimplemented from libfbm::AbstractField.

Definition at line 841 of file libfbm.hpp.

6.4.3.10 double libfbm::Field::operator() (size_t *x*) const [inline]

Reimplemented from libfbm::AbstractField.

Definition at line 848 of file libfbm.hpp.

6.4.3.11 double libfbm::Field::operator() (size_t *x*, size_t *y*) const [inline]

Reimplemented from libfbm::AbstractField.

Definition at line 850 of file libfbm.hpp.

6.4.3.12 double libfbm::Field::operator() (size_t *x*, size_t *y*, size_t *z*) const [inline]

Reimplemented from libfbm::AbstractField.

Definition at line 852 of file libfbm.hpp.

6.4.3.13 double libfbm::Field::operator() (size_t *x*, size_t *y*, size_t *z*, size_t *u*) const [inline]

Reimplemented from libfbm::AbstractField.

Definition at line 854 of file libfbm.hpp.

6.4.3.14 double libfbm::Field::operator() (size_t *x*, size_t *y*, size_t *z*, size_t *u*, size_t *v*) const [inline]

Reimplemented from libfbm::AbstractField.

Definition at line 856 of file libfbm.hpp.

6.4.3.15 void libfbm::Field::setBufferSize (size_t *bufferSize*)

Set the internal buffer size used when reading cached precomputed data from the HD.

This is 65536 by default. Don't worry, your OS is supposed to cache often-used files in memory anyway, so probably there are no actual file reads involved if you run your program properly. Just fast memor-to-memory copies.

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

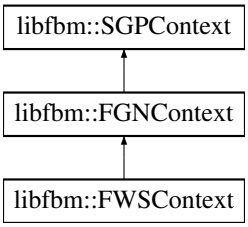
- libfbm.hpp

6.5 libfbm::FWSContext Class Reference

Fractional wiener surface context.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::FWSContext:



Public Member Functions

- FWSContext (double H, const zvec &dim)

Protected Member Functions

- void postProcess (Field &field, GaussianGenerator &rng)

  Postprocessor called after field generation.

6.5.1 Detailed Description

Fractional wiener surface context.

1D case it is equivalent to fractional Brownian motion, for higher-dimensional cases the result is a "fractional Wiener surface", see the paper by Qian [4]. It uses the same covariance function as fractional Gaussian noise, except at post-processing the field is integrated (summed).

Definition at line 709 of file libfbm.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 libfbm::FWSContext::FWSContext (double *H,* const zvec & *dim*) [inline]

Definition at line 712 of file libfbm.hpp.

6.5.3 Member Function Documentation

6.5.3.1 void libfbm::FWSContext::postProcess (Field & *field,* GaussianGenerator & *rng*) [protected], [virtual]

Postprocessor called after field generation.

This is called automatically by Field::generate().

Reimplemented from libfbm::SGPContext.

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

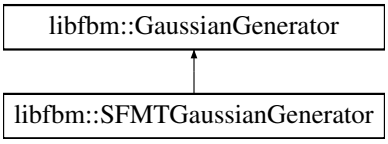
- libfbm.hpp

6.6 libfbm::GaussianGenerator Class Reference

Normally distributed random number generator interface.

```
#include <libfbm.hpp>
```

Inheritance diagram for libfbm::GaussianGenerator:



Public Member Functions

- virtual ~GaussianGenerator ()
- virtual void setSeed (uint32_t seed)=0
- virtual void setSeed (const unsigned char *seed_array, size_t len)=0
- virtual double getDouble ()=0
- virtual void getDouble (double *array, size_t len)=0

6.6.1 Detailed Description

Normally distributed random number generator interface.

You can provide your own by extending this class and passing it to Field::generate().

Definition at line 328 of file libfbm.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 virtual libfbm::GaussianGenerator::~GaussianGenerator ( ) [virtual]

6.6.3 Member Function Documentation

6.6.3.1 virtual double libfbm::GaussianGenerator::getDouble ( ) [pure virtual]

Implemented in libfbm::SFMTGaussianGenerator.

6.6.3.2 virtual void libfbm::GaussianGenerator::getDouble (double * *array*, size_t *len*) [pure virtual]

Implemented in libfbm::SFMTGaussianGenerator.

6.6.3.3 virtual void libfbm::GaussianGenerator::setSeed (uint32_t *seed*) [pure virtual]

Implemented in libfbm::SFMTGaussianGenerator.

6.6.3.4 virtual void libfbm::GaussianGenerator::setSeed (const unsigned char * *seed_array*, size_t *len*) [pure virtual]

Implemented in libfbm::SFMTGaussianGenerator.

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

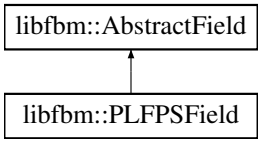
- libfbm.hpp

6.7 libfbm::PLFPSField Class Reference

Power-law power spectrum field.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::PLFPSField:



Public Member Functions

- PLFPSField (const zvec &dim, double H)
- ~PLFPSField ()
- void generate ()

  Generate the field.
- void generate (GaussianGenerator &rng)

  Generate the field using random generator rng.
- void clear ()

  Release all memory this object holds.
- const zvec & getDim () const

  Get the usable dimensions of this field.

Additional Inherited Members

6.7.1 Detailed Description

Power-law power spectrum field.

Loosely based on [6]. The power spectum is given as

$$S(f) \sim 1/f^{(2H+d)}.$$

We fill a complex matrix with gaussian random values. We don't use symmetric conjugation. The random values are scaled based on frequency, that is they are scaled using the power law above as scaling factor. The values are also globally scaled so that the resulting field has variance 1.0. Then a FFT is applied. This result in two fields, one in the real part, the other in the complex part. The real part is exported first, the next generate() switches to the imaginary part. Note that the fields in this case are symmetric. The 0 frequency component is taken as 0. The scaling factors are kept in memory, so total memory usage is 3x that of field dimension.

Definition at line 1023 of file libfbm.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 libfbm::PLFPSField::PLFPSField (const zvec & *dim*, double *H*)

6.7.2.2 libfbm::PLFPSField::~PLFPSField ( )

6.7.3 Member Function Documentation

6.7.3.1 void libfbm::PLFPSField::clear ( ) [virtual]

Release all memory this object holds.

Implements libfbm::AbstractField.

6.7.3.2 void libfbm::PLFPSField::generate ( ) [virtual]

Generate the field.

Implements libfbm::AbstractField.

6.7.3.3 void libfbm::PLFPSField::generate (GaussianGenerator & *rng*) [virtual]

Generate the field using random generator rng.

The random generator should produce normally distributed numbers.

Implements libfbm::AbstractField.

6.7.3.4 const zvec& libfbm::PLFPSField::getDim ( ) const [inline], [virtual]

Get the usable dimensions of this field.

Implements libfbm::AbstractField.

Definition at line 1034 of file libfbm.hpp.

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

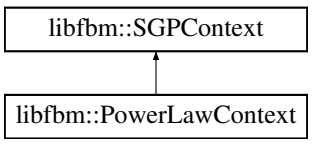
- libfbm.hpp

6.8 libfbm::PowerLawContext Class Reference

Power law covariance context.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::PowerLawContext:



Public Member Functions

- PowerLawContext (double H, const zvec &dim, double Var=2.0)
  Constructor.
- double cov (const zvec &dim)
  Covariance function.

Additional Inherited Members

6.8.1 Detailed Description

Power law covariance context.

$$\rho(\mathbf{r}) = |\mathbf{r}|^{2H},$$

where $H < 0$.

Definition at line 993 of file libfbm.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 libfbm::PowerLawContext::PowerLawContext (double *H,* const zvec & *dim,* double *Var* = 2.0)

Constructor.

Var determines the variance or cov(0). This parameter strongly affects whether circular matrix embedding works or not. You can decrease this to 1 when $H << 0$. Note that as H approaches 0 the field becomes distorted. H=0.01 should still be quite fine.

6.8.3 Member Function Documentation

6.8.3.1 double libfbm::PowerLawContext::cov (const zvec & *p*) [virtual]

Covariance function.

Implements libfbm::SGPContext.

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

- libfbm.hpp

6.9 libfbm::Seeder Class Reference

Seed generator for random number generators.

`#include <libfbm.hpp>`

Public Member Functions

- Seeder (const char *path)
  Construct the Seeder.
- ~Seeder ()
- void seed (uint32_t index)
- size_t size () const
- operator const unsigned char * () const

6.9.1 Detailed Description

Seed generator for random number generators.

The Mersenne twister has a known weakness, that is with a small integer seed the first generated values are not very random. That is it takes a while for the generator to "work in". To get over this one can use a larger seed. This class generates such seeds from an integer index and a datafile containing random bits. On Unix one can easily generate such a seed pool file by executing:

```
dd if=/dev/urandom of=seedpool.dat bs=1024 count=1024
```

The seeds generated are 256 bytes long and contain blocks from the seed pool xor-ed with the given index. I'm not a specialist in random number generation, so I hope this is adequate.

Definition at line 385 of file libfbm.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 libfbm::Seeder::Seeder (const char * *path*)

Construct the Seeder.

The program aborts if the path cannot be opened or seed pool is too small (4kB).

6.9.2.2 libfbm::Seeder::∼Seeder ( )

6.9.3 Member Function Documentation

6.9.3.1 libfbm::Seeder::operator const unsigned char ∗ ( ) const [inline]

Definition at line 396 of file libfbm.hpp.

6.9.3.2 void libfbm::Seeder::seed (uint32_t *index*)

6.9.3.3 size_t libfbm::Seeder::size ( ) const [inline]

Definition at line 395 of file libfbm.hpp.

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

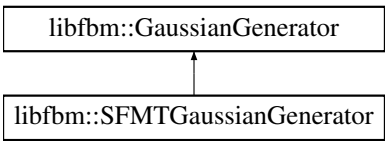
- libfbm.hpp

6.10 libfbm::SFMTGaussianGenerator Class Reference

SIMD-oriented Fast Mersenne Twister (SFMT) and Ziggurat sampling based gaussian random generator.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::SFMTGaussianGenerator:



Public Member Functions

- SFMTGaussianGenerator (uint32_t seed=0)
- SFMTGaussianGenerator (const unsigned char ∗seed_array, size_t len)
- ∼SFMTGaussianGenerator ()
- void setSeed (uint32_t seed)
- void setSeed (const unsigned char ∗seed_array, size_t len)
- double getDouble ()
- void getDouble (double ∗array, size_t len)

6.10.1 Detailed Description

SIMD-oriented Fast Mersenne Twister (SFMT) and Ziggurat sampling based gaussian random generator.

Very fast. Makes use of the excellent SFMT library by Mutsuo Saito and Makoto Matsumoto, http://www.-math.sci.hiroshima-u.ac.jp/∼m-mat/MT/SFMT/, also see [8]. It assumes that the CPU supports SSE2. Gaussian distribution is generated by a variation of the ziggurat algorithm (basically layered rejection sampling). The ziggurat algorithm used is slightly different from the one described by Marsaglia in [7], here we select layers that cover an equal area of the distribution (instead of layers having similar overall area).

Definition at line 351 of file libfbm.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 libfbm::SFMTGaussianGenerator::SFMTGaussianGenerator (uint32_t *seed* = 0)

6.10.2.2 libfbm::SFMTGaussianGenerator::SFMTGaussianGenerator (const unsigned char * *seed_array*, size_t *len*)

6.10.2.3 libfbm::SFMTGaussianGenerator::~SFMTGaussianGenerator ( )

6.10.3 Member Function Documentation

6.10.3.1 double libfbm::SFMTGaussianGenerator::getDouble ( ) [virtual]

Implements libfbm::GaussianGenerator.

6.10.3.2 void libfbm::SFMTGaussianGenerator::getDouble (double * *array*, size_t *len*) [virtual]

Implements libfbm::GaussianGenerator.

6.10.3.3 void libfbm::SFMTGaussianGenerator::setSeed (uint32_t *seed*) [virtual]

Implements libfbm::GaussianGenerator.

6.10.3.4 void libfbm::SFMTGaussianGenerator::setSeed (const unsigned char * *seed_array*, size_t *len*) [virtual]

Implements libfbm::GaussianGenerator.

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

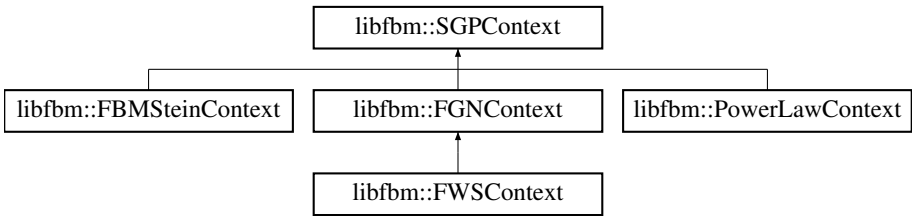
- libfbm.hpp

6.11 libfbm::SGPContext Class Reference

Abstract stationary Gaussian process context.

`#include <libfbm.hpp>`

Inheritance diagram for libfbm::SGPContext:



Public Member Functions

- SGPContext (const zvec &fieldDim, const zvec &userDim, const std::string &cacheName)
  Constructor.
- virtual ~SGPContext ()
- virtual double cov (const zvec &p)=0
  Covariance function.
- const zvec & getDim () const
  Get the usable dimension of the generated field.
- const zvec & getFieldDim () const
  Get the physical dimension of the generated field.
- void setCacheDir (const std::string &cacheDir)
  Set the cache directory.
- size_t badEigenCount () const

Return the number of bad eigenvalues encountered.

- bool initCache (bool forceRecalc=false)

  Initialize the cache.

Protected Member Functions

- virtual void postProcess (Field &field, GaussianGenerator &rng)

  Postprocessor called after field generation.
- void setScaleResult (double f)

  Scale the output values by factor f.

Friends

- class Field

6.11.1 Detailed Description

Abstract stationary Gaussian process context.

Used by the Field. This class provides random number generation and filesystem based caching. To define your own covariance function, you must create a class that extends this. Word of warning: if you have enabled caching, but change cov(), the results from the old version are loaded. To overcome this you must manually delete the old cache. To overcome this you should incorporate paramters affecting cov() into the constructor's cacheName argument.

Definition at line 627 of file libfbm.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 libfbm::SGPContext::SGPContext (const zvec & *fieldDim,* const zvec & *userDim,* const std::string & *cacheName*)

Constructor.

The *cacheName* is the cache identifier used to distinct between cache files on the HD. So if your extension of this class has some paramters affecting cov(), they should be incorporated into cacheName. It is used in the file system path so must be sane in that regard.

6.11.2.2 virtual libfbm::SGPContext::~SGPContext ( ) [virtual]

6.11.3 Member Function Documentation

6.11.3.1 size_t libfbm::SGPContext::badEigenCount ( ) const [inline]

Return the number of bad eigenvalues encountered.

Definition at line 650 of file libfbm.hpp.

6.11.3.2 virtual double libfbm::SGPContext::cov (const zvec & *p*) [pure virtual]

Covariance function.

Implemented in libfbm::PowerLawContext, libfbm::FBMSteinContext, and libfbm::FGNContext.

6.11.3.3 const zvec& libfbm::SGPContext::getDim ( ) const [inline]

Get the usable dimension of the generated field.

Definition at line 642 of file libfbm.hpp.

6.11.3.4 const zvec& libfbm::SGPContext::getFieldDim ( ) const [inline]

Get the physical dimension of the generated field.

It may be greater than the usable dimension returned by getDim().

Definition at line 644 of file libfbm.hpp.

6.11.3.5 bool libfbm::SGPContext::initCache (bool *forceRecalc* = false)

Initialize the cache.

Either loads from cache (quick) or generates (slow). This is called automatically by Field::generate().

Parameters

forceRecalc	Force recalculation and rewriting of the cache.

Returns

Returns false in case negative eigenvalues were found. Otherwise returns true.

6.11.3.6 virtual void libfbm::SGPContext::postProcess (Field & *field*, GaussianGenerator & *rng*) [protected], [virtual]

Postprocessor called after field generation.

This is called automatically by Field::generate().

Reimplemented in libfbm::FBMSteinContext, and libfbm::FWSContext.

6.11.3.7 void libfbm::SGPContext::setCacheDir (const std::string & *cacheDir*)

Set the cache directory.

Must be called before construction of Field-s.

6.11.3.8 void libfbm::SGPContext::setScaleResult (double *f*) [inline], [protected]

Scale the output values by factor f.

This is assembled into the cached FFT matrix so must be called before initCache or Field::generate(). If you change the value, you must either erase the cache or incorporate it into cacheName.

Definition at line 670 of file libfbm.hpp.

6.11.4 Friends And Related Function Documentation

6.11.4.1 friend class Field [friend]

Definition at line 681 of file libfbm.hpp.

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

- libfbm.hpp

6.12 libfbm::SGPTester Class Reference

Class for testing the covariance of generated fields.

`#include <libfbm.hpp>`

Public Member Functions

- void test (SGPContext &context, size_t printInterval=1)

  Run an infinite test.

6.12.1 Detailed Description

Class for testing the covariance of generated fields.

Note that in case of FBMSteinContext you should disable post-processing. This basically generates fields, directly calculates covariances and then compares them to the cov() function provided by the context.

Definition at line 910 of file libfbm.hpp.

6.12.2 Member Function Documentation

6.12.2.1 void libfbm::SGPTester::test (SGPContext & *context*, size_t *printInterval* = 1)

Run an infinite test.

Print status after every *printInterval* generation.

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

- libfbm.hpp

6.13 libfbm::zvec Class Reference

Integer vector.

`#include <libfbm.hpp>`

Public Member Functions

- zvec (size_t dim)
- int & operator[] (size_t i)
- const int & operator[] (size_t i) const
- size_t size () const
- bool increment (int base)

  Iterate the vector value using the given base.
- bool increment (const zvec &dim)

  Iterate the vector value using the given dimensions.
- bool increment_but (const zvec &dim, size_t hold)

  Iterate the vector value using the given dimensions except for the dimension with index hold.
- size_t index (size_t base) const

  Serialized position of the vector.
- size_t index (const zvec &dim) const

  Serialized position of the vector.
- void zero ()

  Zero all vector components.

Friends

- zvec operator+ (const zvec &l, const zvec &r)
- zvec operator- (const zvec &l, const zvec &r)
- zvec operator* (int f, const zvec &v)
- zvec operator* (const zvec &v, int f)

6.13.1 Detailed Description

Integer vector.

The maximum dimension of the vector is hardcoded in the LIBFBM_MAX_DIM (it is 8 by default).

Definition at line 408 of file libfbm.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 libfbm::zvec::zvec (size_t *dim*) [inline]

Definition at line 411 of file libfbm.hpp.

6.13.3 Member Function Documentation

6.13.3.1 bool libfbm::zvec::increment (int *base*) [inline]

Iterate the vector value using the given base.

Returns

Returns false when values at all indices are equal to base.

Definition at line 420 of file libfbm.hpp.

6.13.3.2 bool libfbm::zvec::increment (const zvec & *dim*) [inline]

Iterate the vector value using the given dimensions.

dim.size() must be the same as vector size.

Returns

Returns false when values at all indices are equal to their dimension.

Definition at line 438 of file libfbm.hpp.

6.13.3.3 bool libfbm::zvec::increment_but (const zvec & *dim,* size_t *hold*) [inline]

Iterate the vector value using the given dimensions except for the dimension with index hold.

dim.size() must be the same as vector size.

Returns

Returns false when values at all indices are equal to their dimension.

Definition at line 456 of file libfbm.hpp.

6.13.3.4 size_t libfbm::zvec::index (size_t *base*) const [inline]

Serialized position of the vector.

Definition at line 474 of file libfbm.hpp.

6.13.3.5 size_t libfbm::zvec::index (const zvec & *dim*) const [inline]

Serialized position of the vector.

Definition at line 487 of file libfbm.hpp.

6.13.3.6 int& libfbm::zvec::operator[] (size_t *i*) [inline]

Definition at line 413 of file libfbm.hpp.

6.13.3.7 const int& libfbm::zvec::operator[] (size_t *i*) const [inline]

Definition at line 414 of file libfbm.hpp.

6.13.3.8 size_t libfbm::zvec::size ( ) const [inline]

Definition at line 416 of file libfbm.hpp.

6.13.3.9 void libfbm::zvec::zero ( ) [inline]

Zero all vector components.

Definition at line 500 of file libfbm.hpp.

6.13.4 Friends And Related Function Documentation

6.13.4.1 zvec operator* (int *f*, const zvec & *v*) [friend]

Definition at line 522 of file libfbm.hpp.

6.13.4.2 zvec operator* (const zvec & *v*, int *f*) [friend]

Definition at line 530 of file libfbm.hpp.

6.13.4.3 zvec operator+ (const zvec & *l*, const zvec & *r*) [friend]

Definition at line 506 of file libfbm.hpp.

6.13.4.4 zvec operator- (const zvec & *l*, const zvec & *r*) [friend]

Definition at line 514 of file libfbm.hpp.

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

- libfbm.hpp

Index