

libfbm

0.1

Generated by Doxygen 1.8.1

Sat Sep 14 2013 11:21:19

Contents

1	Main Page	1
2	Namespace Index	4
2.1	Namespace List	4
3	Class Index	5
3.1	Class Hierarchy	5
4	Class Index	5
4.1	Class List	5
5	Namespace Documentation	6
5.1	libfbm Namespace Reference	6
6	Class Documentation	6
6.1	libfbm::FBMSteinContext Class Reference	6
6.1.1	Detailed Description	7
6.1.2	Constructor & Destructor Documentation	7
6.1.3	Member Function Documentation	8
6.2	libfbm::FGNContext Class Reference	8
6.2.1	Detailed Description	9
6.2.2	Constructor & Destructor Documentation	9
6.2.3	Member Function Documentation	9
6.3	libfbm::Field Class Reference	9
6.3.1	Detailed Description	10
6.3.2	Constructor & Destructor Documentation	10
6.3.3	Member Function Documentation	10
6.4	libfbm::FWSContext Class Reference	12
6.4.1	Detailed Description	12
6.4.2	Constructor & Destructor Documentation	12
6.4.3	Member Function Documentation	12
6.5	libfbm::GaussianRandomGenerator Class Reference	13
6.5.1	Detailed Description	13
6.5.2	Constructor & Destructor Documentation	13
6.5.3	Member Function Documentation	13
6.6	libfbm::RandomGenerator Class Reference	14
6.6.1	Detailed Description	14
6.6.2	Constructor & Destructor Documentation	14
6.6.3	Member Function Documentation	15
6.7	libfbm::SGPContext Class Reference	15

6.7.1	Detailed Description	16
6.7.2	Constructor & Destructor Documentation	16
6.7.3	Member Function Documentation	16
6.7.4	Friends And Related Function Documentation	17
6.8	libfbm::SGPTester Class Reference	18
6.8.1	Detailed Description	18
6.8.2	Member Function Documentation	18
6.9	libfbm::UniformRandomGenerator Class Reference	18
6.9.1	Detailed Description	19
6.9.2	Constructor & Destructor Documentation	19
6.9.3	Member Function Documentation	19
6.10	libfbm::zvec Class Reference	19
6.10.1	Detailed Description	20
6.10.2	Constructor & Destructor Documentation	20
6.10.3	Member Function Documentation	20
6.10.4	Friends And Related Function Documentation	21

1 Main Page

Author

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

This library generates or simulates "stationary Gaussian processes". Stationary means here that the covariance between two points does not depend on the individual coordinates of the points:

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

so that only the vector $\mathbf{x} - \mathbf{y}$ 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

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

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

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

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

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

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).

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 random field 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 generator, which is not "exact". And there are numeric errors that may creep up whenever calculations approach zero or infinity.

The library consists of two files (`libfbm.cpp` and `libfbm.hpp`) and depends on the GSL library for FFT functions and pseudo-random generators.

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., & 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., & 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., & Bassingthwaite, 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., & Botev, I. Z.

Spatial Process Generation

<http://www.maths.uq.edu.au/~kroese/ps/MCSpatial.pdf>

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::FWContext ctx(0.75, dim);
    // set random generator seed, optional
    ctx.getRandomGenerator().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
    ctx.getRandomGenerator().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:

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 GSL, 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. You may send a porn magazine, but you should first consider how that will reflect on your character ;)

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

2 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

libfbm	6
------------------------	---

3 Class Index

3.1 Class Hierarchy

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

libfbm::Field	9
libfbm::RandomGenerator	14
libfbm::GaussianRandomGenerator	13
libfbm::UniformRandomGenerator	18
libfbm::SGPContext	15
libfbm::FBMSteinContext	6
libfbm::FGNContext	8
libfbm::FWSContext	12
libfbm::SGPTester	18
libfbm::zvec	19

4 Class Index

4.1 Class List

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

libfbm::FBMSteinContext		
Stein's fractional Brownian motion context		6
libfbm::FGNContext		
Fractional gaussian noise context		8
libfbm::Field		
Gaussian process field		9
libfbm::FWSContext		
Fractional wiener surface context		12
libfbm::GaussianRandomGenerator		
Random generator producing standard normal random values		13
libfbm::RandomGenerator		
Random generator interface		14
libfbm::SGPContext		
Abstract stationary Gaussian process context		15
libfbm::SGPTester		
Class for testing the covariance of generated fields		18

libfbm::UniformRandomGenerator

Uniform random generator based on GSL, producing floating point numbers uniformly distributed in the range [0,1)

18

libfbm::zvec

Integer vector

19

5 Namespace Documentation

5.1 libfbm Namespace Reference

Classes

- class [RandomGenerator](#)
Random generator interface.
- class [UniformRandomGenerator](#)
Uniform random generator based on GSL, producing floating point numbers uniformly distributed in the range [0,1).
- class [GaussianRandomGenerator](#)
Random generator producing standard normal random values.
- class [zvec](#)
Integer vector.
- class [SGPContext](#)
Abstract stationary Gaussian process context.
- class [FGNContext](#)
Fractional gaussian noise context.
- class [FWSContext](#)
Fractional wiener surface context.
- class [Field](#)
Gaussian process field.
- class [SGPTester](#)
Class for testing the covariance of generated fields.
- class [FBMSteinContext](#)
Stein's fractional Brownian motion context.

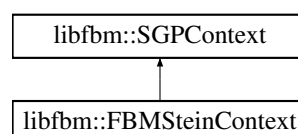
6 Class Documentation

6.1 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, double Rhint=-1)
Get the precomputed R value for the given H.

Protected Member Functions

- void [postProcess](#) ([Field](#) &field)
Postprocessor called after field generation.

6.1.1 Detailed Description

Stein's fractional Brownian motion context.

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

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

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

$$\text{Cov}(Z(\mathbf{x}), Z(\mathbf{y})) = c_0 - |\mathbf{x} - \mathbf{y}|^{2H} + c_2 \left(|\mathbf{x}|^2 + |\mathbf{y}|^2 \right).$$

The [FBMSteinContext](#) changes the actual generated 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 and should be ignored.

Definition at line 755 of file libfbm.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `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

<i>dim</i>	Dimensionality of the field wanted (1D, 2D, 3D, etc.).
<i>size</i>	Request field of size^dim.
<i>Rhint</i>	Hint that overrides the self-determined R for H>0.75, ignored for H<=0.75. See [3] for details.
<i>mapDim</i>	Expand dimension so that the resulting usable field is the size requested. True by default.

6.1.3 Member Function Documentation

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

Covariance function.

Implements [libfbm::SGPContext](#).

6.1.3.2 void libfbm::FBMSteinContext::disablePostProcessing ()

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

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

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

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

Definition at line 775 of file libfbm.hpp.

6.1.3.5 static double libfbm::FBMSteinContext::getRForH (double *H*, 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*.

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

Postprocessor called after field generation.

This is called automatically by [Field::generate\(\)](#).

Reimplemented from [libfbm::SGPContext](#).

6.1.3.7 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 $H \leq 0.75$, *R* is always 1.0. For $H > 0.75$, *R* must be increased. It is proven that the method 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:

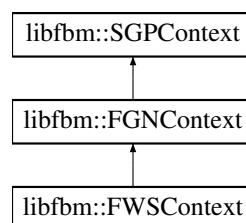
- libfbm.hpp

6.2 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

6.2.1 Detailed Description

Fractional gaussian noise context.

Definition at line 593 of file libfbm.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 libfbm::FGNContext::FGNContext (double H , const [zvec](#) & dim)

6.2.3 Member Function Documentation

6.2.3.1 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

6.3 libfbm::Field Class Reference

Gaussian process field.

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

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 [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.

6.3.1 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 624 of file libfbm.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 libfbm::Field::Field (SGPCContext & 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.

6.3.2.2 libfbm::Field::~Field ()

6.3.3 Member Function Documentation

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

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

Definition at line 697 of file libfbm.hpp.

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

Definition at line 705 of file libfbm.hpp.

6.3.3.3 void libfbm::Field::clear ()

Release all memory this object holds.

6.3.3.4 void libfbm::Field::generate ()

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 $2 \cdot (2 \cdot d_1) \cdot (2 \cdot d_2) \cdot \dots \cdot (2 \cdot d_n)$ doubles, where $d_1 \dots d_n$ are the dimensions of the generated process. Note that the precomputed eigenvalue cache size (if kept in memory) additionally adds half of that.

6.3.3.5 `const zvec& libfbm::Field::getDim () const` `[inline]`

Get the usable dimensions of this field.

Definition at line 694 of file libfbm.hpp.

6.3.3.6 `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 subtract the `strides[d]` value to the address.

Definition at line 718 of file libfbm.hpp.

6.3.3.7 `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 [FWSCContext](#) calls this automatically.

6.3.3.8 `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.

Definition at line 668 of file libfbm.hpp.

6.3.3.9 `double libfbm::Field::operator() (size_t x) const` `[inline]`

Definition at line 675 of file libfbm.hpp.

6.3.3.10 `double libfbm::Field::operator() (size_t x, size_t y) const` `[inline]`

Definition at line 677 of file libfbm.hpp.

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

Definition at line 679 of file libfbm.hpp.

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

Definition at line 681 of file libfbm.hpp.

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

Definition at line 683 of file libfbm.hpp.

6.3.3.14 `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:

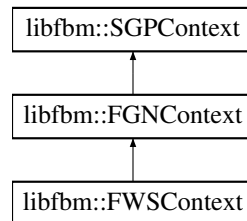
- libfbm.hpp

6.4 libfbm::FWSSContext Class Reference

Fractional wiener surface context.

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

Inheritance diagram for libfbm::FWSSContext:



Public Member Functions

- [FWSSContext](#) (double *H*, const [zvec](#) &*dim*)

Protected Member Functions

- void [postProcess](#) ([Field](#) &*field*)
Postprocessor called after field generation.

6.4.1 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 610 of file libfbm.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 `libfbm::FWSSContext::FWSSContext (double H, const zvec &dim)` `[inline]`

Definition at line 613 of file libfbm.hpp.

6.4.3 Member Function Documentation

6.4.3.1 `void libfbm::FWSSContext::postProcess (Field &field)` `[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:

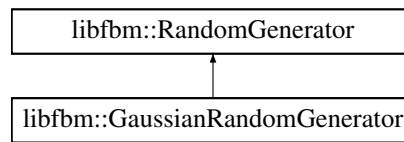
- libfbm.hpp

6.5 libfbm::GaussianRandomGenerator Class Reference

Random generator producing standard normal random values.

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

Inheritance diagram for libfbm::GaussianRandomGenerator:



Public Member Functions

- [GaussianRandomGenerator](#) ()
Construct using a [UniformRandomGenerator](#).
- [GaussianRandomGenerator](#) (const [RandomGenerator](#) *urg)
Construct using a uniform random generator urg.
- [GaussianRandomGenerator](#) (const [GaussianRandomGenerator](#) ©)
- const [GaussianRandomGenerator](#) & operator= (const [GaussianRandomGenerator](#) ©)
- [~GaussianRandomGenerator](#) ()
- void [setSeed](#) (unsigned long int seed)
- double [next](#) ()
Next random value.
- [RandomGenerator](#) * [clone](#) () const
Allocates a completely new random generator (in the heap using new), copying the state of this.

6.5.1 Detailed Description

Random generator producing standard normal random values.

It uses the Marsaglia polar method.

Definition at line 301 of file libfbm.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 libfbm::GaussianRandomGenerator::GaussianRandomGenerator ()

Construct using a [UniformRandomGenerator](#).

6.5.2.2 libfbm::GaussianRandomGenerator::GaussianRandomGenerator (const [RandomGenerator](#) * urg)

Construct using a uniform random generator *urg*.

Parameters

<i>urg</i>	A uniform random generator. It is cloned and stored internally.
------------	---

6.5.2.3 libfbm::GaussianRandomGenerator::GaussianRandomGenerator (const [GaussianRandomGenerator](#) & copy)

6.5.2.4 libfbm::GaussianRandomGenerator::~~GaussianRandomGenerator ()

6.5.3 Member Function Documentation

6.5.3.1 **RandomGenerator*** libfbm::GaussianRandomGenerator::clone () const [virtual]

Allocates a completely new random generator (in the heap using new), copying the state of this.

Implements [libfbm::RandomGenerator](#).

6.5.3.2 **double** libfbm::GaussianRandomGenerator::next () [virtual]

Next random value.

Implements [libfbm::RandomGenerator](#).

6.5.3.3 **const GaussianRandomGenerator&** libfbm::GaussianRandomGenerator::operator= (const GaussianRandomGenerator & copy)

6.5.3.4 **void** libfbm::GaussianRandomGenerator::setSeed (unsigned long int seed) [virtual]

Implements [libfbm::RandomGenerator](#).

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

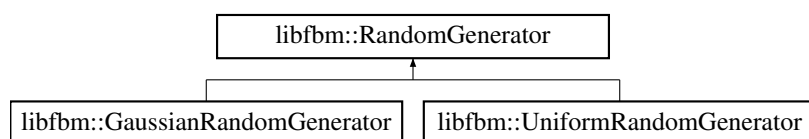
- libfbm.hpp

6.6 libfbm::RandomGenerator Class Reference

Random generator interface.

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

Inheritance diagram for libfbm::RandomGenerator:



Public Member Functions

- [RandomGenerator](#) ()
- virtual void [setSeed](#) (unsigned long int seed)=0
- virtual double [next](#) ()=0
Next random value.
- virtual [RandomGenerator](#) * [clone](#) () const =0
Allocates a completely new random generator (in the heap using new), copying the state of this.
- virtual [~RandomGenerator](#) ()

6.6.1 Detailed Description

Random generator interface.

Definition at line 263 of file libfbm.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 **libfbm::RandomGenerator::RandomGenerator** () [inline]

Definition at line 266 of file libfbm.hpp.

6.6.2.2 virtual libfbm::RandomGenerator::~~RandomGenerator () [virtual]

6.6.3 Member Function Documentation

6.6.3.1 virtual RandomGenerator* libfbm::RandomGenerator::clone () const [pure virtual]

Allocates a completely new random generator (in the heap using new), copying the state of this.

Implemented in [libfbm::GaussianRandomGenerator](#), and [libfbm::UniformRandomGenerator](#).

6.6.3.2 virtual double libfbm::RandomGenerator::next () [pure virtual]

Next random value.

Implemented in [libfbm::GaussianRandomGenerator](#), and [libfbm::UniformRandomGenerator](#).

6.6.3.3 virtual void libfbm::RandomGenerator::setSeed (unsigned long int seed) [pure virtual]

Implemented in [libfbm::GaussianRandomGenerator](#), and [libfbm::UniformRandomGenerator](#).

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

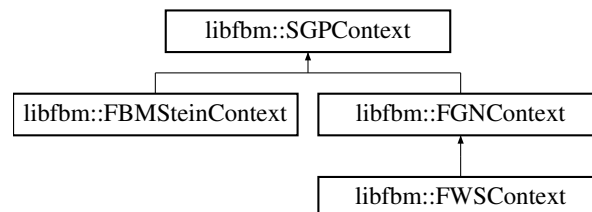
- libfbm.hpp

6.7 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.
- void [setRandomGenerator](#) (const [RandomGenerator](#) &newRng)
Set the standard normal random value generator.
- [RandomGenerator](#) & [getRandomGenerator](#) ()
- const [zvec](#) & [getDim](#) () const
Get the usable dimension of the generated field.
- const [zvec](#) & [getFieldDim](#) () const
Get the physical dimension of the generated field.
- double [nextRng](#) () const
- void [setCacheDir](#) (const std::string &cacheDir)
Set the cache directory.

- `size_t badEigenCount () const`
Return the number of bad eigenvalues encountered.
- `void initCache (bool forceRecalc=false)`
Initialize the cache.

Protected Member Functions

- `virtual void postProcess (Field &field)`
Postprocessor called after field generation.
- `void setScaleResult (double f)`
Scale the output values by factor f.

Friends

- class `Field`

6.7.1 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 parameters affecting `cov()` into the constructor's `cacheName` argument.

Definition at line 526 of file `libfbm.hpp`.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `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 parameters affecting `cov()`, they should be incorporated into `cacheName`. It is used in the file system path so must be sane in that regard.

6.7.2.2 `virtual libfbm::SGPContext::~~SGPContext () [virtual]`

6.7.3 Member Function Documentation

6.7.3.1 `size_t libfbm::SGPContext::badEigenCount () const [inline]`

Return the number of bad eigenvalues encountered.

Definition at line 555 of file `libfbm.hpp`.

6.7.3.2 `virtual double libfbm::SGPContext::cov (const zvec & p) [pure virtual]`

Covariance function.

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

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

Get the usable dimension of the generated field.

Definition at line 546 of file `libfbm.hpp`.

6.7.3.4 `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 548 of file libfbm.hpp.

6.7.3.5 `RandomGenerator& libfbm::SGPContext::getRandomGenerator ()` `[inline]`

Definition at line 543 of file libfbm.hpp.

6.7.3.6 `void 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

<i>forceRecalc</i>	Force recalculation and rewriting of the cache.
--------------------	---

6.7.3.7 `double libfbm::SGPContext::nextRng () const` `[inline]`

Definition at line 549 of file libfbm.hpp.

6.7.3.8 `virtual void libfbm::SGPContext::postProcess (Field & field)` `[protected], [virtual]`

Postprocessor called after field generation.

This is called automatically by [Field::generate\(\)](#).

Reimplemented in [libfbm::FBMSteinContext](#), and [libfbm::FWSContext](#).

6.7.3.9 `void libfbm::SGPContext::setCacheDir (const std::string & cacheDir)`

Set the cache directory.

Must be called before construction of Field-s.

6.7.3.10 `void libfbm::SGPContext::setRandomGenerator (const RandomGenerator & newRng)`

Set the standard normal random value generator.

It is cloned and kept internally.

6.7.3.11 `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 574 of file libfbm.hpp.

6.7.4 Friends And Related Function Documentation**6.7.4.1** `friend class Field` `[friend]`

Definition at line 586 of file libfbm.hpp.

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

- libfbm.hpp

6.8 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.

6.8.1 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 739 of file `libfbm.hpp`.

6.8.2 Member Function Documentation

6.8.2.1 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:

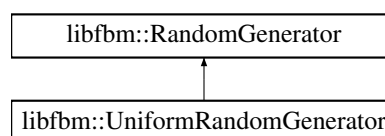
- `libfbm.hpp`

6.9 libfbm::UniformRandomGenerator Class Reference

Uniform random generator based on GSL, producing floating point numbers uniformly distributed in the range [0,1).

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

Inheritance diagram for `libfbm::UniformRandomGenerator`:



Public Member Functions

- [UniformRandomGenerator](#) ()
- [UniformRandomGenerator](#) (unsigned long int seed, const gsl_rng_type *type)
- [UniformRandomGenerator](#) (const [UniformRandomGenerator](#) ©)
- [~UniformRandomGenerator](#) ()
- const [UniformRandomGenerator](#) & [operator=](#) (const [UniformRandomGenerator](#) ©)
- void [setSeed](#) (unsigned long int seed)
- double [next](#) ()
Next random value.

- [RandomGenerator](#) * [clone](#) () const

Allocates a completely new random generator (in the heap using new), copying the state of this.

6.9.1 Detailed Description

Uniform random generator based on GSL, producing floating point numbers uniformly distributed in the range [0,1).

The default `gsl_rng_type` use is `gsl_rng_mt19937`.

Definition at line 281 of file `libfbm.hpp`.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 `libfbm::UniformRandomGenerator::UniformRandomGenerator ()`

6.9.2.2 `libfbm::UniformRandomGenerator::UniformRandomGenerator (unsigned long int seed, const gsl_rng_type * type)`

6.9.2.3 `libfbm::UniformRandomGenerator::UniformRandomGenerator (const UniformRandomGenerator & copy)`

6.9.2.4 `libfbm::UniformRandomGenerator::~~UniformRandomGenerator ()`

6.9.3 Member Function Documentation

6.9.3.1 `RandomGenerator* libfbm::UniformRandomGenerator::clone () const` `[virtual]`

Allocates a completely new random generator (in the heap using new), copying the state of this.

Implements [libfbm::RandomGenerator](#).

6.9.3.2 `double libfbm::UniformRandomGenerator::next ()` `[virtual]`

Next random value.

Implements [libfbm::RandomGenerator](#).

6.9.3.3 `const UniformRandomGenerator& libfbm::UniformRandomGenerator::operator= (const UniformRandomGenerator & copy)`

6.9.3.4 `void libfbm::UniformRandomGenerator::setSeed (unsigned long int seed)` `[virtual]`

Implements [libfbm::RandomGenerator](#).

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

- `libfbm.hpp`

6.10 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)

6.10.1 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 379 of file libfbm.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 libfbm::zvec::zvec (size_t dim) [inline]

Definition at line 382 of file libfbm.hpp.

6.10.3 Member Function Documentation

6.10.3.1 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 391 of file libfbm.hpp.

6.10.3.2 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 409 of file libfbm.hpp.

6.10.3.3 `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 427 of file libfbm.hpp.

6.10.3.4 `size_t libfbm::zvec::index (size_t base) const [inline]`

Serialized position of the vector.

Definition at line 445 of file libfbm.hpp.

6.10.3.5 `size_t libfbm::zvec::index (const zvec & dim) const [inline]`

Serialized position of the vector.

Definition at line 458 of file libfbm.hpp.

6.10.3.6 `int& libfbm::zvec::operator[] (size_t i) [inline]`

Definition at line 384 of file libfbm.hpp.

6.10.3.7 `const int& libfbm::zvec::operator[] (size_t i) const [inline]`

Definition at line 385 of file libfbm.hpp.

6.10.3.8 `size_t libfbm::zvec::size () const [inline]`

Definition at line 387 of file libfbm.hpp.

6.10.3.9 `void libfbm::zvec::zero () [inline]`

Zero all vector components.

Definition at line 471 of file libfbm.hpp.

6.10.4 Friends And Related Function Documentation

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

Definition at line 493 of file libfbm.hpp.

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

Definition at line 501 of file libfbm.hpp.

6.10.4.3 `zvec operator+ (const zvec & l, const zvec & r) [friend]`

Definition at line 477 of file libfbm.hpp.

6.10.4.4 `zvec operator- (const zvec & l, const zvec & r) [friend]`

Definition at line 485 of file libfbm.hpp.

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

- libfbm.hpp

Index

- ~Field
 - libfbm::Field, 9
- ~GaussianRandomGenerator
 - libfbm::GaussianRandomGenerator, 13
- ~RandomGenerator
 - libfbm::RandomGenerator, 14
- ~SGPContext
 - libfbm::SGPContext, 16
- ~UniformRandomGenerator
 - libfbm::UniformRandomGenerator, 18
- at
 - libfbm::Field, 10
- badEigenCount
 - libfbm::SGPContext, 16
- clear
 - libfbm::Field, 10
- clone
 - libfbm::GaussianRandomGenerator, 13
 - libfbm::RandomGenerator, 14
 - libfbm::UniformRandomGenerator, 18
- cov
 - libfbm::FBMSteinContext, 7
 - libfbm::FGNContext, 8
 - libfbm::SGPContext, 16
- disablePostProcessing
 - libfbm::FBMSteinContext, 7
- FBMSteinContext
 - libfbm::FBMSteinContext, 7
- FGNContext
 - libfbm::FGNContext, 8
- FWSContext
 - libfbm::FWSContext, 12
- Field
 - libfbm::Field, 9
 - libfbm::SGPContext, 17
- fieldSize2UserSize
 - libfbm::FBMSteinContext, 7
- GaussianRandomGenerator
 - libfbm::GaussianRandomGenerator, 13
- generate
 - libfbm::Field, 10
- getDim
 - libfbm::Field, 10
 - libfbm::SGPContext, 16
- getFieldDim
 - libfbm::SGPContext, 16
- getR
 - libfbm::FBMSteinContext, 7
- getRForH
 - libfbm::FBMSteinContext, 7
- getRandomGenerator
 - libfbm::SGPContext, 16
- getStrides
 - libfbm::Field, 10
- increment
 - libfbm::zvec, 20
- increment_but
 - libfbm::zvec, 20
- index
 - libfbm::zvec, 20
- initCache
 - libfbm::SGPContext, 16
- integrate
 - libfbm::Field, 10
- libfbm, 5
 - libfbm::FBMSteinContext, 6
 - cov, 7
 - disablePostProcessing, 7
 - FBMSteinContext, 7
 - fieldSize2UserSize, 7
 - getR, 7
 - getRForH, 7
 - postProcess, 7
 - userSize2FieldSize, 8
 - libfbm::FGNContext, 8
 - cov, 8
 - FGNContext, 8
 - libfbm::FWSContext, 11
 - FWSContext, 12
 - postProcess, 12
 - libfbm::Field, 9
 - ~Field, 9
 - at, 10
 - clear, 10
 - Field, 9
 - generate, 10
 - getDim, 10
 - getStrides, 10
 - integrate, 10
 - operator(), 10, 11
 - setBufferSize, 11
 - libfbm::GaussianRandomGenerator, 12
 - ~GaussianRandomGenerator, 13
 - clone, 13
 - GaussianRandomGenerator, 13
 - next, 13
 - operator=, 13
 - setSeed, 13
 - libfbm::RandomGenerator, 13
 - ~RandomGenerator, 14
 - clone, 14
 - next, 14
 - RandomGenerator, 14
 - setSeed, 14
 - libfbm::SGPContext, 14

- ~SGPContext, 16
- badEigenCount, 16
- cov, 16
- Field, 17
- getDim, 16
- getFieldDim, 16
- getRandomGenerator, 16
- initCache, 16
- nextRng, 16
- postProcess, 16
- SGPContext, 16
- setCacheDir, 16
- setRandomGenerator, 17
- setScaleResult, 17
- libfbm::SGPTester, 17
 - test, 17
- libfbm::UniformRandomGenerator, 18
 - ~UniformRandomGenerator, 18
 - clone, 18
 - next, 18
 - operator=, 19
 - setSeed, 19
 - UniformRandomGenerator, 18
- libfbm::zvec, 19
 - increment, 20
 - increment_but, 20
 - index, 20
 - operator*, 21
 - operator+, 21
 - operator-, 21
 - size, 20
 - zero, 21
 - zvec, 20
- next
 - libfbm::GaussianRandomGenerator, 13
 - libfbm::RandomGenerator, 14
 - libfbm::UniformRandomGenerator, 18
- nextRng
 - libfbm::SGPContext, 16
- operator*
 - libfbm::zvec, 21
- operator()
 - libfbm::Field, 10, 11
- operator+
 - libfbm::zvec, 21
- operator-
 - libfbm::zvec, 21
- operator=
 - libfbm::GaussianRandomGenerator, 13
 - libfbm::UniformRandomGenerator, 19
- postProcess
 - libfbm::FBMSteinContext, 7
 - libfbm::FWSContext, 12
 - libfbm::SGPContext, 16
- RandomGenerator
 - libfbm::RandomGenerator, 14
- SGPContext
 - libfbm::SGPContext, 16
- setBufferSize
 - libfbm::Field, 11
- setCacheDir
 - libfbm::SGPContext, 16
- setRandomGenerator
 - libfbm::SGPContext, 17
- setScaleResult
 - libfbm::SGPContext, 17
- setSeed
 - libfbm::GaussianRandomGenerator, 13
 - libfbm::RandomGenerator, 14
 - libfbm::UniformRandomGenerator, 19
- size
 - libfbm::zvec, 20
- test
 - libfbm::SGPTester, 17
- UniformRandomGenerator
 - libfbm::UniformRandomGenerator, 18
- userSize2FieldSize
 - libfbm::FBMSteinContext, 8
- zero
 - libfbm::zvec, 21
- zvec
 - libfbm::zvec, 20