SCARF3D

A Scalable Three-Dimensional Random Field Generator

Version 2.4

User's Manual

Walter Imperatori
Swiss Seismological Service, ETH Zurich
`walter.imperatori@sed.ethz.ch`
`https://github.com/flurinursli/scarf3d`

Contents

1 Introduction

SCARF3D is a numerical package to efficiently generate large-scale, two- and three-dimensional random fields with prescribed power spectrum. It is primarily designed to be used as a library, that is, to be invoked by other computer programs to generate random perturbations directly on the nodes of structured and unstructured meshes (thus avoiding costly intermediate I/O operations), but it can be used also as a standalone program if desired. The code is highly flexible and allows users to easily generate continuous multi-resolution random fields across mesh refinement interfaces. It implements two different algorithms that can be selected depending on the available hardware resources and problem size. The package relies on the popular MPI and FFTW libraries and it has been tested from desktop machines up to hybrid CPU/GPU supercomputers, showing very good scaling properties. The simulated random fields closely follow the desired auto-correlation function.
Although designed to generate 3D heterogeneous structures of the Earth crust, the library may be useful in any scientific field in which the characterization of random but correlated variations of physical parameters on a numerical grid are of interest.

1.1 Random fields, autocorrelation functions and power spectral densities

Random fields can be used to characterize the distribution of any parameter that cannot be otherwise represented in a deterministic way. These fields have specific statistical properties described by an **autocorrelation function** (ACF) and its Fourier transform, known as the **power spectral density function** (PSDF). Given a scalar field $\xi(\mathbf{x})$ that is a random function of coordinate $\mathbf{x}$, a randomly perturbed parameter $p(\mathbf{x})$ can be written as:

$$p(\mathbf{x}) = p_0 + \delta p(\mathbf{x}) = p_0[1 + \xi(\mathbf{x})] \tag{1.1}$$

where p_0 is its mean value. Note that $\xi(\mathbf{x}) = \delta p(\mathbf{x})/p_0$ is a non-dimensional quantity and it represents the fractional fluctuation of parameter p.
SCARF3D can be used to generate field $\xi(\mathbf{x})$ based on three different ACFs: Gaussian, Von Karman and Exponential. These are defined as follows:

$$\begin{aligned} R_{gs}(\mathbf{x}) &= \sigma^2 e^{-r^2} \\ R_{vk}(\mathbf{x}) &= \frac{\sigma^2 2^{(1-\nu)}}{\Gamma(\nu)} r^\nu H_\nu(r) \\ R_{ex}(\mathbf{x}) &= \sigma^2 e^{-r} \end{aligned} \tag{1.2}$$

where Γ is the Gamma function, ν the Hurst exponent, H is the Bessel function of the second kind of order ν and $r = \sqrt{\sum_{i=1}^{D} x_i^2/a_i^2}$, being a the correlation length and D the dimension (either 2 or 3).
The corresponding PSDFs are:

$$\begin{aligned} S_{gs}(\mathbf{k}) &= e^{-m/4}\sigma^2\pi^{D/2}\prod_{i=1}^{D} a_i \\ S_{vk}(\mathbf{k}) &= \frac{2^D\sigma^2\pi^{D/2}\Gamma(\nu + D/2)\prod_{i=1}^{D} a_i}{\Gamma(\nu)(1+m)^{\nu+D/2}} \\ S_{ex}(\mathbf{k}) &= \frac{(D-1)2^{D-1}\sigma^2\pi\prod_{i=1}^{D} a_i}{(1+m)^{(D+1)/2}} \end{aligned} \tag{1.3}$$

where $m = \sum_{i=1}^{D} k_i^2 a_i^2$, being k the wave number. Note that the exponential ACF is a special case of the Von Karman ACF having $\nu = 0.5$.
When the correlation length a is identical along the main axes, the resulting random field will be completely isotropic. Anisotropy can be enforced by setting direction-dependent correlation values: in this case perturbations will be elongated but still along the main axes. *SCARF3D* allows to arbitrarily orient the perturbations (see Figure 5 in [1]) by means of rotation angles described in Section 3.1.1.
Our library implements two different algorithms based on the **Fourier Integral Method**

(FIM) [e.g., 2] and the **Spectral Representation Method** (SRM) [3]. Their implementation in *SCARF3D* is described in detail in [1]. Both methods rely on the evaluation of PSDFs. Interested users may easily add other correlation functions whose spectral density is defined analytically. Section 3.4 provides some guidelines showing how this can be achieved.

2 Installation

2.1 Introduction

In this chapter we describe how to install *SCARF3D* on Unix/Linux systems and the necessary system requirements. Note that it can be installed either as a library or as a stand-alone program, depending on your needs. The code has been successfully tested on a range of computer systems, from desktop machines to hybrid CPU/GPU supercomputers.

2.2 Download

SCARF3D is freely available at `https://github.com/flurinursli/scarf3d`. The downloaded zip file contains the following items:

- `examples/`: sample Fortran, C and C++ driver programs
- `postprocessing/`: Matlab/Octave scripts to visualize and analyze the output of the driver programs
- `src/`: source files to build the library
- `standalone/`: source files to build the stand-alone program
- `Makefile`: the makefile itself
- `Makefile.inc`: file with user-defined input parameters for `make`
- `common.mk`: file with default compilers flags
- `License.txt`: the GNU GPL license
- `Manual.pdf`: this manual

2.3 Hardware and Memory Requirements

SCARF3D is a library for high-performance computing applications and, as such, well suited for supercomputers equipped with multiple computing nodes and GPU accelerators interconnected by a fast network. However, it can also run on more modest

equipment, compatibly with the desired model size/resolution. Although both algorithm implemented can run on CPUs, we recommend the use of GPUs for the SRM. Consumer GPUs (e.g. Nvidia RTX2060) can be used, too, but performance may be severely hindered.
The two implemented algorithms have different memory footprints: assuming that N_x, N_y and N_z are the total number of grid points along each direction, the SRM allocates about

$$\frac{N_x \cdot N_y \cdot N_z \cdot p}{1024^3} \tag{2.1}$$

GB of memory (p is the arithmetic precision in bytes), while the FIM 4 times more. For example, generating a random field of 500^3 points in single precision (4 bytes) with the SRM requires in total 0.47GB, while about 1.86GB with the FIM. The actual memory consumption for the FIM is even larger if borders padding and/or arbitrarily oriented heterogeneity (see Table 3.2) are allowed [1].

2.4 Compilers

SCARF3D can be compiled using any compiler suite supporting FORTRAN (standard 2003), ANSI C and C++11, provided the required libraries are already installed (see Section 2.5). We have successfully built *SCARF3D* using the following compilers:

- **GCC**, version 7.4, 9.2, 10.1, `https://gcc.gnu.org/`
- **Intel**, version 19.1, `https://software.intel.com/content/www/us/en/develop/tools/compilers.html`
- **PGI** Community Edition, version 19.10, `https://www.pgroup.com/`

The GCC and PGI Community Edition can be obtained for free, while Intel compilers require a commercial license. Other suites can be used, but the `common.mk` file must be first modified accordingly (see Section 2.6).
Note that the SRM algorithm relies on **OpenACC** directives for GPU acceleration: these are supported only by the GCC and PGI suites, with the latter providing a more mature implementation. GPU offloading based on **OpenMP** directives (standard 4 or higher) will be considered in a future release. This implies that the SRM algorithm can only run (very slowly!) on CPUs if Intel compilers are selected.

2.5 External Libraries

In order to build *SCARF3D*, the following external libraries must be present in your system:

- **MPI** (the Message Passing Interface), standard 3 or higher, `https://www.mpi-forum.org/`
- **FFTW** (the Fastest Fourier Transform in the West), version 3.0 or higher (version 3.3.8 recommended), `http://www.fftw.org/`
- **TRNG** (Tina's Random Number Generator), version 4.20 or higher, `https://numbercrunch.de/trng/`

MPI provides support for message passing on parallel machines and it is required even for building *SCARF3D* on single core systems. A popular open-source implementation is OpenMPI (`https://www.open-mpi.org/`). FFTW is a widely distributed collection of fast routines to compute the discrete Fourier transform (DFT). TRNG is a free C++ pseudo-random number generator library suitable for parallel programs that has become part of the C++11 standard. In our numerical package it is used to guarantee hardware- and compiler-independent random heterogeneity distributions.
Instructions on how to install these libraries can be found on the respective websites. Although not strictly necessary, we recommend building these libraries and *SCARF3D* using the same compiler suite. In our experience, TRNG is the only library that may not be available by default in some systems. In this case we recommend to simply install it in your local home folder using mild optimization flags: since the random number generator is called outside the most time-consuming routines, users must not worry about relevant performance penalties if a sub-optimal installation is performed.

2.6 How to compile

SCARF3D can be built using **make**. The process is controlled by environmental variables defined in the provided `Makefile.inc` and summarized in Table 2.1.
In general, we recommend building *SCARF3D* in single-precision since this translates into better performing code with minimal loss of accuracy [1]. Also, disabling `TIMING` reduces communication overhead and execution time. Debugging mode should be used exclusively for troubleshooting. The `PFS` flag is relevant only if the native *SCARF3D* I/O routines are exploited. Note that enabling `DIP` degrades performance sensibly as the actual internal grid for the Fourier Transform is enlarged: if possible, users should rely on the SRM algorithm to generate random field with arbitrarily oriented heterogeneity.
An additional file, `common.mk`, defines a set of environmental variables with compiler-specific optimization and debugging flags. Those suggested in the file for the tested compilers are, in our experience, very good but interested users can modify them or eventually add new flags appropriate for other suites. Note that the second part of `common.mk` should not be modified in any case.

Variable	Meaning	Accepted values
`FC,C,C++`	Fortran, C and C++ MPI wrappers.	(any)
`COMPILER`	Define compilers suite. Set manually if env. variable `$VENDOR` is undefined.	*gcc/intel/pgi*
`PRECISION`	Build library in single- or double-precision.	***single***/*double*
`DEBUG`	Enable verbose output and set compiler diagnostic flags.	*yes/**no***
`TIMING`	Enable timing of main routines.	*yes/**no***
`PFS`	Target parallel file systems.	***yes***/*no*
`SPECTRAL`	Enable testing SRM algorithm in driver programs.	*yes/no*
`DIP`	Allow arbitrarily oriented perturbations in FIM algorithm.	*yes/**no***
`TRNG_DIR`	Path to the TRNG library.	(any)
`FFTW_DIR`	Path to the FFTW library.	(any)

Table 2.1: Environmental variables in `Makefile.inc` and their meaning. Recommended values are in **bold**.

Once all variables are duly set, to build *SCARF3D* as a library type:

```
make lib
```

The library can be then installed in a specific location:

```
make install prefix=/my/preferred/location/
```

To build the driver programs, type:

```
make examples
```

Note that this step will also compile the library if necessary. The resulting executables, named `driverF.exe`, `driverC.exe` and `driverCPP.exe`, are located in the corresponding folders under `examples/`.
It is also possible to create both library and sample programs at the same time:

```
make all
```

On the other hand, to build *SCARF3D* as stand-alone program type:

```
make exe
```

The executable file, named `scarf3d.exe`, can be found in the `standalone/` folder.

3 Using SCARF3D

In this chapter we describe how to use *SCARF3D* as a library and as a stand-alone program. We also illustrate its basic I/O routines for Cartesian meshes. *SCARF3D* is distributed with sample FORTRAN, C and C++ driver programs located under `examples/` and its sub-folders (see Section 2.6 to compile them). These driver programs, beside showcasing the use of both FIM and SRM algorithms to generate basic two- and three-dimensional random fields, allow the interested user, together with the visualization scripts described in Section 3.3, to reproduce figure 2 and 3 of [1].

3.1 As a library

SCARF3D has been developed to be used mainly as a library. This way, for instance, it is possible to generate random perturbations of physical parameters directly on the nodes of a mesh.
All interfaces share the same structure. The workflow is based on four steps:

- **Initialization:** allocate resources, setup variables for optional I/O
- **Execution:** the random field is calculated using a specific seed number
- **Output:** the random field is written to disk (optional)
- **Finalization:** release resources

In a typical application, where most of the parameters are kept constant, the **initialization** and **finalization** steps must be invoked only once, whereas the **execution** step can be called as many times as necessary. This particular structure allows users to produce many random field realizations by simply changing the seed number (see also Section 3.1.2).
The **output** step is optional: normally users would exploit the I/O routines of the embedding program to store the random field or the perturbed physical parameters to disk, if desired. However, it is also possible to use *SCARF3D*'s own I/O facilities to output the random field for, e.g., checking purposes.
In the following sections we first describe all the input and output parameters, and then focus on the FORTRAN, C and C++ interfaces. Note that the C and C++ interfaces serve merely as wrappers for the underlying FORTRAN code.

3.1.1 Initialization

In the initialization stage users define a set of parameters, some of which are optional. Table 3.1 list all the mandatory parameters, including a brief description, type and allowed values. Note that in the following `real` may indicate either single- or double-precision floating point numbers while `vector` and `array` refer to 1D and 2D arrays, respectively.

Variable	Description	Type	Value
`fs`[1]	First index of input mesh for calling MPI process	int, vector	> 0
`fe`[1]	Last index of input mesh for calling MPI process	int, vector	$\geq$ `fs`
`ds`[1]	Grid-step of input mesh	real, scalar	> 0
`x`[2]	Location of mesh nodes along x-axis for calling MPI process	real, vector	any
`y`[2]	Location of mesh nodes along y-axis for calling MPI process	real, vector	any
`z`[2,3]	Location of mesh nodes along z-axis for calling MPI process	real, vector	any
`dh`[2]	Space sampling interval	real, scalar	> 0
`acf`	Autocorrelation function	int, scalar	$[0, 2]$[a]
`cl`	Correlation length	real, vector	> 0
`sigma`	Continuous standard deviation	real, scalar	> 0
`hurst`[4]	Hurst exponent	real, scalar	$(0, 1]$

[a] 0 = Von Karman, 1 = Gaussian, 2 = User-Defined
[1] Cartesian meshes
[2] non-Cartesian meshes
[3] 3D models only
[4] required for Von Karman ACFs

Table 3.1: Mandatory input parameters.

Most parameters are self-explanatory. For Cartesian grids it is convenient - but not mandatory - to work with indices since the vertices are points on an integer lattice. Moreover working with indices may result in somewhat faster execution of the SRM algorithm on consumer GPU cards, as memory latency-bound effects are less exposed. For non-Cartesian grids, parameter `dh` determines the smallest resolvable heterogeneity or, equivalently, the maximum wave number in the spectral domain. It should be equal to or larger than the actual maximum inter-node distance to avoid aliasing effects. Care must be taken for vector parameters `fs, fe` and `cl`: these must all have either 2 or 3 elements, depending if users want to generate two- or three-dimensional random fields. Mixing vectors of different length will lead to unpredictable behavior.

Variable	Description	Type	Value	Default
`method`	Select algorithm of choice	int, scalar	0, 1[a]	0
`ds`[1]	Maximum desired resolution	real, scalar	> 0	`dh`
`dh`[2]	Space sampling interval	real, scalar	> 0	`ds`
`pad`	Avoid wrap-around effects in FIM	int, scalar	0, 1	0
`poi`	Points where the random field will be muted and/or tapered	real, array	any	none
`mute`	Muting radius	real, scalar	≥ 0	0
`taper`	Taper radius	real, scalar	≥ 0	0
`alpha`	Rotation angle around z-axis	real, scalar		0
`beta`[3]	Rotation angle around y-axis	real, scalar		0
`gamma`[3]	Rotation angle around x-axis	real, scalar		0
`rescale`	Normalize perturbations to continuous standard deviation	int, scalar	0, 1	0
`nc`	Location of first mesh node along each cartesian direction	real, vector		*see text*
`fc`	Location of last mesh node along each cartesian direction	real, vector		*see text*

[a] 0 = FIM, 1 = SRM
[1] non-Cartesian meshes
[2] Cartesian meshes
[3] 3D models only

Table 3.2: Optional input parameters.

Table 3.2 lists the optional input parameters, including their default values. It should be noted that the FIM is the default algorithm in *SCARF3D*. Setting variables `ds` and `dh` can be useful in those cases when the actual maximum wave number (determined by `ds`) must be smaller than the Nyquist wave number (controlled by `dh`): for instance when the same random field is desired on grids with different resolution or when work-

ing with refined meshes. In practice, the PSDF is low-pass filtered between these two wave numbers. Note that if both optional parameters are left blank, Nyquist and actual maximum wave numbers coincide and no filtering takes place.
Users may also face situations when the random field should be zeroed at some specific locations. In seismology, this may be necessary at grid nodes representing seismic sources to avoid altering the scalar moment [1] or, more in general, to have smooth physical properties inside absorbing boundaries. Such particular locations can be specified via `poi`, whose first dimension must be a 2- or 3-elements vector according to the parameters in Table 3.1. The random field within a given radius from these locations can be muted and/or tapered based on parameters `mute` and `taper`, respectively.
Anisotropic random perturbations are, by default, oriented along the major axes. They can be arbitrarily rotated along a specific direction according to parameter `alpha`, `beta` and `gamma`. These describe a rotation (in degrees) around the z-, y- and x-axis, respectively. Assuming the right-hand convention, perturbations are rotated counterclockwise for negative angles. As noted in Section 2.6, rotation is not recommended for the FIM and, unless enabled at compile-time, these parameters are ignored for such algorithm.
By default, continuous and discrete (actual) standard deviations do not coincide. The latter will be always smaller than the former because of spectral truncation [4]. The difference between the two can be used to infer how well the target power spectral density is sampled. Users can set `rescale` such that the computed random field has exactly the desired standard deviation, although this will lead to a discrete medium with different continuous properties.
The final two optional parameters, `nc` and `fc`, can be used to set the *near-corner* and *far-corner* of a model, i.e. they mark the position of the closest and farthest grid node for the calling MPI process. If not specified, their values are derived from the input nodes (see Table 3.1). `nc` and `fc` can be used, for instance, to generate the same perturbations distribution on two meshes having different size.
In any case we strongly advice users to check carefully the units of all input parameters: these must be consistent in order to produce meaningful results. For instance, specifying grid-step of the input mesh (`ds`) in meters and correlation length (`cl`) in kilometers will lead almost certainly to unexpected results.

3.1.2 Execution

Random perturbations can be generated for a given set of input parameters by calling a single routine, whose interface is reported in Section 3.1.4, 3.1.5 and 3.1.6, as many times as necessary. It requires three parameters, all mandatory, described in Table 3.3 (here `Intent` discriminates between input and output arguments).

Parameter	Description	Type	Intent
`seed`	Seed number to initialize the random number generator	int, scalar	In
`field`	The computed random field	real, array	Out
`stats`	Information on the generated perturbations	real, vector	Out

Table 3.3: Parameters for the execution routine.

Useful information is stored in `stats`, a 8-element real vector where the first two entries are non-zero if spectral truncation at small (first element) or large (second element) wave numbers likely occurred during the calculations (according to [4]). The third and fourth elements return the discrete standard deviation and mean of the whole random field. The remaining elements are meaningless, unless the library was built with the `TIMING` flag (see Section 2.6). In this case, if the FIM algorithm is chosen, elements from the fifth to the last position indicate the elapsed time (in seconds) to compute the spectrum, to apply symmetry conditions, to compute inverse FFTs and to interpolate the random field back onto the input mesh, respectively. For the SRM algorithm, the fifth and sixth element return how much time was spent in the CPU and the GPU, while the remaining two are not used.

3.1.3 Output

SCARF3D has two basic routines to write random fields to disk as binary files and their interface is described in Section 3.1.4, 3.1.5 and 3.1.6. Note that these routines can handle Cartesian meshes only and users are therefore requested to provide their own routines for more irregular grids. The required input parameters, shown in Table 3.4, vary a bit depending on whether the whole model or a single slice must be saved.

Parameter	Description	Type	Value
`npts`	Total input grid points along each direction	int, vector	> 0
`axis`[a]	Axis along which the field is sliced	char	"x", "y", "z"
`plane`[a]	Location of the slice in terms of grid points	int, scalar	[1, npts]
`field`	The computed random field	real, array	
`filename`	Name of the output file	string	any
`nwriters`[b,*]	Information on the generated perturbations	int, scalar	[1, N_p]

[a] slices only
[b] volumetric output only
[*] optional argument

Table 3.4: Parameters required by the output routines.

The optional argument `nwriters` is used to set the number of MPI processes that can write concurrently to disk. If not specified, or if the library was compiled for a serial file system (see Section 2.6), it is assumed equal to one.
Note that the C and C++ interfaces (Section 3.1.5, 3.1.6) require an additional integer parameter, `nd`, indicating whether the random field is two- or three-dimensional. Obviously slices can be written only for three-dimensional data.
All binary files follow column-major ordering: for volumetric output this means that the x- and z-coordinate correspond to the fastest and slowest changing index, respectively. The same order holds also for slices.

3.1.4 FORTRAN interface

Initialization: two overloaded subroutines, one for Cartesian and one for generic meshes.

```
  scarf_initialize(fs, fe, ds, acf, cl, sigma, [method], [hurst], [dh], [poi],
[mute], [taper], [rescale], [pad], [nc], [fc], [alpha], [beta], [gamma])

  scarf_initialize(dh, acf, cl, sigma, x, y, [z], [method], [hurst], [ds],
[poi], [mute], [taper], [rescale], [pad], [nc], [fc], [alpha], [beta], [gamma])
```

They must be called only once for a given set of input parameters. Optional arguments (marked by brackets) may be entered in any order if keywords are used.

Execution: single subroutine, it may be called as many times as necessary.

```
scarf_execute(seed, field, stats)
```

Output (optional): two overloaded subroutines for volume and slice output (3D case only)

```
scarf_io(n, axis, plane, field, filename)
```

```
scarf_io(n, field, filename, [nwriters])
```

These may be called as many times as necessary. Optional arguments are marked by brackets.

Finalization: single subroutine

```
scarf_finalize()
```

It must be called only once, also before a new initialization takes place.

3.1.5 C interface

Initialization: since C does not accept optional arguments, these are passed via the C structure `scarf_opt` defined in the header file `scarf3d.h`. For instance one may declare

```
struct scarf_opt options;
```

and then set all the members to their default value by calling

```
scarf_opt_init(&options);
```

Note that this function expects an address as input. After modifying the desired optional arguments (Table 3.2), the library can be finally initialized:

```
  scarf_cart_initialize(const int nd, const int fs[], const int fe[], const
real ds, const int acf, const real cl[], const real sigma, struct scarf_opt
*var)
```

```
  scarf_nocart_initialize(const int nd, const int npts, const real* x, const
real* y, const real* z, const real dh, const int acf, const real cl[], const
real sigma, struct scarf_opt *var)
```

As in the FORTRAN case, these functions must be called only once and also before a new initialization takes place.

Execution:

```
  scarf_execute(const int seed, real* field, real stats[])
```

This function may be called as many times as necessary.

Output (optional): two distinct functions for volume and slice output (3D case only)

```
  scarf_io_vol(const int nd, const int npts[], const real* field, const char
fname[], const int* nwriters)

  scarf_io_slice(const int npts[], const char axis[], const int plane, const
real* field, const char fname[])
```

These may be called as many times as necessary.

Finalization:

```
  scarf_finalize()
```

This function must called only once, also before a new initialization takes place.

3.1.6 C++ interface

The `Scarf3D` C++ class methods are contained in the header file `scarf3d.h` and described more in detail here below.

Initialization: two overloaded methods, one for Cartesian and one for generic meshes. Optional parameters, if necessary, can be set via a structure (described in Section 3.1.5) before the methods are invoked. The structure is declared as `option` under the `Scarf3D` namespace.

```
  Initialize(const int nd, const int fs[], const int fe[], const real ds,
const int acf, const real cl[], const real sigma)

  Initialize(const int nd, const int npts, const real* x, const real* y, const
real* z, const real dh, const int acf, const real cl[], const real sigma)
```

Note that the algorithm (FIM or SRM) can be selected when instantiating the template, e.g.:

```
  Scarf3D::Initialize<fim> obj(const int nd, ...)
```

Both methods must be called only once for a given set of input parameters.

Execution:

```
  execute(const int seed, real* field, real stats[])
```

This method may be called as many times as necessary.

Output (optional): two overloaded methods for volume and slice output (3D case only)

```
  io(const int nd, const int npts[], const real* field, const char fname[],
const int* nwriters = nullptr)

  io(const int npts[], const char axis[], const int plane, const real* field,
const char fname[])
```

They may be called as many times as necessary.

Finalization: a destructor in C++ is called automatically when the object goes out of scope. However, if required, it can be explicitly called as follows:

```
  ~Initialize()
```

In any case, the method must be called only once and also before a new initialization takes place.

Users should notice that vector and arrays in Table 3.1, 3.2, 3.3 and 3.4 are represented by pointers in C and C++.

3.2 As a stand-alone program

SCARF3D can be used as a stand-alone program to generate random fields on Cartesian meshes only. In this case a plain text file must provide the input parameters. These can be entered in any order but must be introduced by a commented line, marked by the "#" symbol, containing the related keyword. For instance, to indicate a 5% standard deviation users must type:

```
# sigma
0.05
```

The following keywords, listed in alphabetic order, are accepted: `acf`, `alpha`, `beta`, `cl`, `dimension`, `ds`, `filename`, `gamma`, `hurst`, `method`, `mute`, `npts`, `nwriters`, `pad`, `poi`, `rescale`, `seed`, `taper`. Note that `dimension` can be either `2` or `3` since it indicates whether a two- or three-dimensional random field will be generated. For the exact meaning of the other keywords see Tables 3.1, 3.2, 3.3 and 3.4. `poi` can be repeated as many times as necessary. We remind that all the non-scalar parameters must conform to `dimension`, otherwise an error will be triggered.
Once the necessary input parameters are saved into a file called, e.g., *input.txt*, the program can be executed as:

```
mpirun -np 4 scarf3d.exe input.txt
```

assuming OpenMPI is used. The program will then echo to standard output the input parameters and any error encountered during execution. The computed random field is saved to disk as a single binary file as described at the end of Section 3.1.3.

3.3 Visualization

Folder `postprocessing/` contains two Matlab/Octave scripts, `vizme.m` and `scarf3d.m`, that can be used to visualize and analyze random fields generated by *SCARF3D* on Cartesian meshes and saved to disk as binary files via its own I/O routines. These include also the random fields created by the sample driver programs under `examples/`. In particular, `scarf3d.m` allows to compare continuous and discrete isotropic ACFs (including their corresponding PSDF) at regular intervals along the major axes (see figure 2 and 3 in [1]) and it is therefore useful to evaluate how accurately the target ACF was sampled. The list of arguments is described in detail in the header of each script.

3.4 User-defined power spectral density functions

SCARF3D is flexible and allow users to compute random fields characterized by ACFs other than those listed in Section 1.1. However, the following requirements must be met:

- an analytical expression for the corresponding PSDF exists
- the PSDF depends on two parameters at most

Users must implement the corresponding PSDF in the FORTRAN module `psdf.f90` located under `src/`. The new routine must be a function with one input argument only,

typically representing the wave number or a derived quantity (as variable m in Eq. 1.3, for instance). Other parameters (e.g., fractal dimension, etc.) may be accessed by module association by overloading argument `Hurst` (see Table 3.1).
For efficiency's sake, users may want to evaluate wavenumber-independent quantities outside their function. For the FIM algorithm, the resulting value can be assigned to variable `const` defined in subroutine `compute_spectrum` (located in `src/scarflib_fim.f90`). The same variable can be found in subroutine `scarf3d_(un)structured_srm` (located in `src/scarflib_srm.f90`) for the SRM algorithm. Note that in this case `const` may contain also a scaling factor necessary for the inequality $k^{(D-1)} \cdot S(k) \leq 1$ to hold (S, k and D have the same meaning as in 1.1), since this is required by the implemented acceptance-rejection method.
We stress that the argument of the PSDF is $k \cdot a$ and k for the FIM and SRM algorithm, respectively. For the latter, dependency on the correlation length is expressed by Eq. 9 of [3]. This implies that, for both algorithms, correlation length-independent ACFs (e.g. fractal) require setting `cl` (Table 3.1) to unity.

License

References

[1] Walter Imperatori, P. Martin Mai, and Donat Fäh. “SCARF3D: a scalable library to efficiently generate large-scale, three-dimensional random fields”. In: *Comput. Geosci.* Submitted ().

[2] E. Pardo-Igúzquiza and M. Chica-Olmo. “The Fourier Integral Method: An efficient spectral method for simulation of random fields”. In: *Math. Geol.* 25.2 (Feb. 1993), pp. 177–217. ISSN: 08828121. DOI: 10.1007/BF00893272.

[3] Ludovic Räss, Dmitriy Kolyukhin, and Alexander Minakov. “Efficient parallel random field generator for large 3-D geophysical problems”. In: *Comput. Geosci.* 131 (2019), pp. 158–169. ISSN: 00983004. DOI: 10.1016/j.cageo.2019.06.007.

[4] Lena Frenje and Christopher Juhlin. “Scattering attenuation: 2-D and 3-D finite difference simulations vs. theory”. In: *J. Appl. Geophys.* 44.1 (Jan. 2000), pp. 33–46. ISSN: 09269851. DOI: 10.1016/S0926-9851(00)00003-3.

Index