ParaCircE, a parallel Gaussian Random Field (GRF) generator

Simon Legrand, Géraldine Pichot

ParaCircE, a parallel Gaussian Random Field (GRF) generator

Simon Legrand[1] Géraldine Pichot[1]

[1] Inria Paris and ENPC

SIAM GS21

Contents

Contents

Gaussian Random Fields

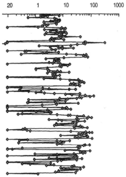
Context

In hydrogeology, permeability/transmissivity are often modeled by **second order stationary fields**, with **lognormal** distributions.

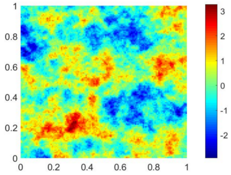
Classical applications are:

- Groundwater ressources management
- Pollution propagation studies
- Waste storage studies in deep geological media

Sand and gravel deposits in Switzerland, Gelhar [1993]



permeability (md)



Matérn isotrope covariance, $L=1$, $n=128^2$, $\lambda=0.125$

Contents

Gaussian Random Fields

1D General algorithm

Let $\Omega \in \mathbb{R}$. To generate $\boldsymbol{y}$, a Gaussian vector, with zero mean and covariance matrix $\boldsymbol{R}$, defined on a $N+1$ points regular grid of Ω:

1. Factorize $\boldsymbol{R} = \boldsymbol{B}\boldsymbol{B}^{\boldsymbol{T}}$
2. Generate $\boldsymbol{\theta} = (\theta_0, ..., \theta_N)^T$, a realization of standard normal variables.
3. One realization is obtained with $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{\theta}$

Among existing methods

- Cholesky factorization, efficient on small problems, but $O(N^3)$
- Karhunen Loève series expansion, expensive when $\lambda << L$
- H-Matrices [Feischl et al., 2018], suited for unstructured meshes
- **Circulant Embedding** [Dietrich et al., 1993], [Graham et al., 2018], with exact correlation structure, suited for large regular grids.

Gaussian Random Fields

1D General algorithm

Let $\Omega \in \mathbb{R}$. To generate $\boldsymbol{y}$, a Gaussian vector, with zero mean and covariance matrix $\boldsymbol{R}$, defined on a $N+1$ points regular grid of Ω:

1. Factorize $\boldsymbol{R} = \boldsymbol{B}\boldsymbol{B}^{\boldsymbol{T}}$
2. Generate $\boldsymbol{\theta} = (\theta_0, ..., \theta_N)^T$, a realization of standard normal variables.
3. One realization is obtained with $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{\theta}$

Among existing methods

- Cholesky factorization, efficient on small problems, but $O(N^3)$
- Karhunen Loève series expansion, expensive when $\lambda << L$
- H-Matrices [Feischl et al., 2018], suited for unstructured meshes
- **Circulant Embedding** [Dietrich et al., 1993], [Graham et al., 2018], with exact correlation structure, suited for large regular grids.

Contents

1D Circulant Embedding algorithm

1. Sample the covariance function to get the vector $\boldsymbol{a} = (c_0, ..., c_N, c_{N-1}, ..., c_1) \in \mathbb{R}^{2N}$, first row of a circulant matrix $\boldsymbol{A}$.
2. If N is large enough, $\boldsymbol{A}$ is SPD, we compute its eigenvalues with FFT, otherwise increase N (padding). $\boldsymbol{s} = \boldsymbol{F}\boldsymbol{a}$ and set $\boldsymbol{D} = Diag(\boldsymbol{s})$
3. Generate realizations of standard normal vectors of size $2N$ $\theta = \theta^{re} + i\theta^{im}$
4. Apply iFFT to compute $\boldsymbol{C}^*\theta$, with $\boldsymbol{C}^* = \frac{1}{\sqrt{2N}}\boldsymbol{F}^*\boldsymbol{D}^{\frac{1}{2}}$
5. Retrieve **two** GRF $\boldsymbol{Y}_1 = (\boldsymbol{C}^*\theta)^{re}(0:N), \quad \boldsymbol{Y}_2 = (\boldsymbol{C}^*\theta)^{im}(0:N)$

1D Circulant Embedding algorithm

1. Sample the covariance function to get the vector
   $\boldsymbol{a} = (c_0, ..., c_N, c_{N-1}, ..., c_1) \in \mathbb{R}^{2N}$,
   first row of a circulant matrix $\boldsymbol{A}$.
2. If N is large enough, $\boldsymbol{A}$ is SPD, we compute its eigenvalues with FFT, otherwise increase N (padding).
   $\boldsymbol{s} = \boldsymbol{F}\boldsymbol{a}$ and set $\boldsymbol{D} = Diag(\boldsymbol{s})$
3. Generate realizations of standard normal vectors of size $2N$
   $\theta = \theta^{re} + i\theta^{im}$
4. Apply iFFT to compute
   $\boldsymbol{C}^*\theta$, with $\boldsymbol{C}^* = \frac{1}{\sqrt{2N}}\boldsymbol{F}^*\boldsymbol{D}^{\frac{1}{2}}$
5. Retrieve **two** GRF
   $\boldsymbol{Y}_1 = (\boldsymbol{C}^*\theta)^{re}(0:N), \quad \boldsymbol{Y}_2 = (\boldsymbol{C}^*\theta)^{im}(0:N)$

1D Circulant Embedding algorithm

1. Sample the covariance function to get the vector $\boldsymbol{a} = (c_0, ..., c_N, c_{N-1}, ..., c_1) \in \mathbb{R}^{2N}$, first row of a circulant matrix $\boldsymbol{A}$.
2. If N is large enough, $\boldsymbol{A}$ is SPD, we compute its eigenvalues with FFT, otherwise increase N (padding).
   $\boldsymbol{s} = \boldsymbol{F}\boldsymbol{a}$ and set $\boldsymbol{D} = Diag(\boldsymbol{s})$
3. Generate realizations of standard normal vectors of size $2N$
   $\theta = \theta^{re} + i\theta^{im}$
4. Apply iFFT to compute
   $\boldsymbol{C}^*\theta$, with $\boldsymbol{C}^* = \frac{1}{\sqrt{2N}}\boldsymbol{F}^*\boldsymbol{D}^{\frac{1}{2}}$
5. Retrieve **two** GRF
   $\boldsymbol{Y}_1 = (\boldsymbol{C}^*\theta)^{re}(0:N), \quad \boldsymbol{Y}_2 = (\boldsymbol{C}^*\theta)^{im}(0:N)$

1D Circulant Embedding algorithm

1. Sample the covariance function to get the vector
   $\boldsymbol{a} = (c_0, ..., c_N, c_{N-1}, ..., c_1) \in \mathbb{R}^{2N}$,
   first row of a circulant matrix $\boldsymbol{A}$.
2. If N is large enough, $\boldsymbol{A}$ is SPD, we compute its eigenvalues with FFT, otherwise increase N (padding).
   $\boldsymbol{s} = \boldsymbol{F}\boldsymbol{a}$ and set $\boldsymbol{D} = Diag(\boldsymbol{s})$
3. Generate realizations of standard normal vectors of size $2N$
   $\theta = \theta^{re} + i\theta^{im}$
4. Apply iFFT to compute
   $\boldsymbol{C}^*\theta$, with $\boldsymbol{C}^* = \frac{1}{\sqrt{2N}}\boldsymbol{F}^*\boldsymbol{D}^{\frac{1}{2}}$
5. Retrieve **two** GRF
   $\boldsymbol{Y}_1 = (\boldsymbol{C}^*\theta)^{re}(0:N), \quad \boldsymbol{Y}_2 = (\boldsymbol{C}^*\theta)^{im}(0:N)$

1D Circulant Embedding algorithm

1. Sample the covariance function to get the vector
   $\boldsymbol{a} = (c_0, ..., c_N, c_{N-1}, ..., c_1) \in \mathbb{R}^{2N}$,
   first row of a circulant matrix $\boldsymbol{A}$.
2. If N is large enough, $\boldsymbol{A}$ is SPD, we compute its eigenvalues with FFT, otherwise increase N (padding).
   $\boldsymbol{s} = \boldsymbol{F}\boldsymbol{a}$ and set $\boldsymbol{D} = Diag(\boldsymbol{s})$
3. Generate realizations of standard normal vectors of size $2N$
   $\theta = \theta^{re} + i\theta^{im}$
4. Apply iFFT to compute
   $\boldsymbol{C}^*\theta$, with $\boldsymbol{C}^* = \frac{1}{\sqrt{2N}}\boldsymbol{F}^*\boldsymbol{D}^{\frac{1}{2}}$
5. Retrieve **two** GRF
   $\boldsymbol{Y}_1 = (\boldsymbol{C}^*\theta)^{re}(0:N), \quad \boldsymbol{Y}_2 = (\boldsymbol{C}^*\theta)^{im}(0:N)$

Contents

Features

- **Matérn** family of covariance (example in 2D)

$$\rho_{2D}(\mathbf{x}, \nu, \boldsymbol{\lambda}) = \kappa(\frac{\sqrt{\lambda_2^2\, x^2 + \lambda_1^2\, y^2}}{\lambda_1\, \lambda_2}, \nu),$$

with

$$\kappa(r, \nu) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}\, r)^{\nu} K_{\nu}(\sqrt{2\nu}\, r),$$

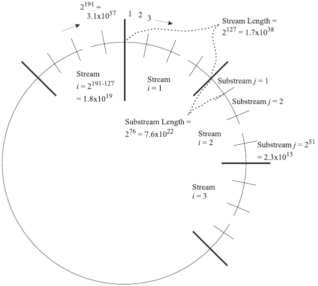
  - $\boldsymbol{x} = (x, y)$
  - Γ the gamma function
  - K_ν the modified Bessel function of the second kind
  - λ_i the correlation length along direction i
  - $\nu > 0$ a smoothness parameter.

- **Parallel** (MPI)
  One direction parallelism (FFTW). The reproducibility is guaranteed whatever the number of processes (RngStream).

Features

- **Independance** and **reproducibility** thanks to the RngStream library [L'Ecuyer et al., 2002]

2^{191}-periodic set of pseudo random numbers, divided into **Streams** and **Substreams**.



- **Accelerated** minimum domain size computation

  If padding is required, save computational resources by approximating the required domain size to get positive eigenvalues [Graham et al., 2018, Legrand et al., 2021].

Contents

Technical overview

- C++17/MPI headers (templated) library
  $\rightarrow$ Static dimension 2D/3D
  $\rightarrow$ Variable numerical precision (double/long double)
- Dependencies
  - MPI
  - FFTW_MPI [Frigo et al., 2005]
  - RngStream [L'Ecuyer et al., 2002]
- Installation with **CMake**
  $\rightarrow$ Portability
  $\rightarrow$ Easy dependencies management

Contents

ParaCircE API

Data structure instantiation

- Domain

```
DiscretDomain2D<double> d{
  {{lx_min, lx_max, nx}, {ly_min, ly_max, ny}}
};
```

- Correlation function

```
auto f = create_function2D<double>(correl_type,
                                    lcx, lcy);
```

- Field generator

```
auto generator = GRF(MPI_COMM_WORLD, d, f);
```

Field generation

```
auto field = generator.generate(nx_local);
```

ParaCircE API

Data structure instantiation

- Domain

```
DiscretDomain2D<double> d{
   {{lx_min, lx_max, nx}, {ly_min, ly_max, ny}}
};
```

- Correlation function

```
auto f = create_function2D<double>(correl_type,
                                    lcx, lcy);
```

- Field generator

```
auto generator = GRF(MPI_COMM_WORLD, d, f);
```

Field generation

```
auto field = generator.generate(nx_local);
```

ParaCircE API

Data structure instantiation

- Domain

```
DiscretDomain2D<double> d{
  {{lx_min, lx_max, nx}, {ly_min, ly_max, ny}}
};
```

- Correlation function

```
auto f = create_function2D<double>(correl_type,
                                    lcx, lcy);
```

- Field generator

```
auto generator = GRF(MPI_COMM_WORLD, d, f);
```

Field generation

```
auto field = generator.generate(nx_local);
```

ParaCircE API

Data structure instantiation

- Domain

```
DiscretDomain2D<double> d{
  {{lx_min, lx_max, nx}, {ly_min, ly_max, ny}}
};
```

- Correlation function

```
auto f = create_function2D<double>(correl_type,
                                   lcx, lcy);
```

- Field generator

```
auto generator = GRF(MPI_COMM_WORLD, d, f);
```

Field generation

```
auto field = generator.generate(nx_local);
```

ParaCircE API

For **Monte-Carlo** use, change stream/substream of the pseudo RNG RngStream.
$//generator.set\_state(stream, substream);$

Field generation

```
...
for (size_t i=0; i<nb_mc; i++)
{
  generator.set_state(0, i);
  auto field = generator.generate(nx_local);
  ...
}
```

Contents

Tests environment

- Instrumentation performed with Score-P/Scalasca tools
- Compilation options: *-O3*
- Execution on Inria Paris CLEPS cluster
  - Xeon 5218, 2.4GHz
  - 192Go RAM 2667MHz
  - 100Gb/s Inifiniband network
- 1 core/MPI process
- **Exclusive** access to the nodes
- No hyperthreading

Contents

Strong scaling

- **Fixed** size problem (2048*512*512)
- **Increasing number** of processes

Maximum acceleration given by Amdhal law:

$$A(N) = \frac{1}{s + \frac{p}{N}}$$

N: Number of tasks
s: sequential time fraction = 0.014
p: parallel time fraction = 0.986
Total time $\approx$ 41min

Results

- Scales like FFTW

Strong scaling

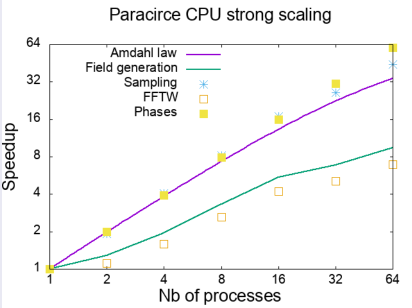
- **Fixed** size problem (2048*512*512)
- **Increasing number** of processes

Maximum acceleration given by Amdhal law:

$$A(N) = \frac{1}{s + \frac{p}{N}}$$

N: Number of tasks
s: sequential time fraction = 0.014
p: parallel time fraction = 0.986
Total time $\approx$ 41min

Results



- Scales like FFTW

Strong scaling

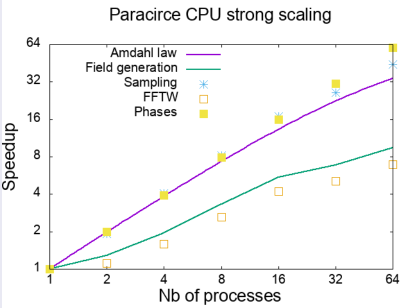
- **Fixed** size problem (2048*512*512)
- **Increasing number** of processes

Maximum acceleration given by Amdhal law:

$$A(N) = \frac{1}{s + \frac{p}{N}}$$

N: Number of tasks
s: sequential time fraction = 0.014
p: parallel time fraction = 0.986
Total time $\approx$ 41min

Results



- Scales like FFTW

Contents

Memory scaling

- The goal of ParaCircE is to generate **large** fields.
- Main constraint: **Memory occupation** (**R**esident **S**et **S**ize)

Circulant Embedding algorithm implies **symmetrizing** the domain.
In dimension d:

$$RSS \approx complexSize * (\mathbf{2}N)^d$$

For $N = 512, d = 3$, with a complex size of 128 bits:

$$RSS \approx 16 * 8 * 512^3 \approx 16GB$$

Memory scaling

- The goal of ParaCircE is to generate **large** fields.
- Main constraint: **Memory occupation** (**R**esident **S**et **S**ize)

Circulant Embedding algorithm implies **symmetrizing** the domain.
In dimension d:

$$RSS \approx complexSize * (\mathbf{2}N)^d$$

For $N = 512, d = 3$, with a complex size of 128 bits:

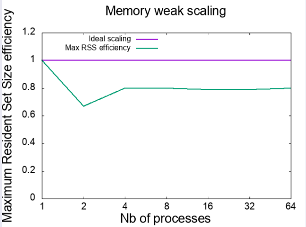
$$RSS \approx 16 * 8 * 512^3 \approx 16GB$$

Memory weak scaling

- Both problem size and number of tasks increase
- **Constant workload** per task, $\approx 512^3$ samples

Results

Nb_proc	Time(min)	Max RSS(GB)	RSS Efficiency
1	7.8	15.9	1.00
2	11.2	23.9	0.67
4	16.8	20.0	0.80
8	23.2	20.0	0.80
16	31.0	20.1	0.79
32	41.4	20.1	0.79
64	70.3	20.0	0.80



Relatively **low** execution time

- $\sim$**134 millions** elements: 1 proc $\rightarrow$ **7min48s**
- $\sim$**8.6 billions** elements: 64 proc $\rightarrow$ **1h10min** (would require 1TB in sequential)

Constant memory scaling!

Conclusion

- **Easy** to install and use
- **Fast** even for very large cases
- **Accelerated** method for automated padding computation

Perspectives

- Periodic field generation
- Multithread
- What do you need?

Gitlab repository: https://gitlab.inria.fr/slegrand/paracirce

Conclusion

- **Easy** to install and use
- **Fast** even for very large cases
- **Accelerated** method for automated padding computation

Perspectives

- Periodic field generation
- Multithread
- What do you need?

Gitlab repository: https://gitlab.inria.fr/slegrand/paracirce

For Further Reading I

LEcuyer, Pierre and Simard, Richard and Chen, E. and Kelton, David
An Object-Oriented Random-Number Package With Many Long Streams And Substreams
Operations Research, 2002, Vol. 50, pp. 1073-1075

I. G. Graham, F. Y. Kuo, D. Nuyens, R. Scheichl, and I. H. Sloan
Analysis of Circulant Embedding Methods for Sampling Stationary Random Fields
SIAM Journal on Numerical Analysis, 2018, Vol. 56, No. 3 : pp. 1871-1895

Simon Legrand, Geraldine Pichot, Nathanael Tepakbong
Algorithms to speed up the generation of stationary Gaussian Random Fields with the Circulant Embedding method
HAL 2021

For Further Reading II

M. Frigo and S. G. Johnson.
The design and implementation of fftw3.
Proc. IEEE 93, 2:216–231, 2005.

M. Feischl, F.Y. Kuo, and I.H. Sloan.
Fast random field generation with h-matrices.
Numer. Math., 140:639676, 2018.

C. R. Dietrich and G. N. Newsam.
A fast and exact method for multidimensional gaussian stochastic simulations.
Water Resources Research, 29(8):2861–2869, 1993.