Efficient Gaussian Copula Density Computation for Large-Scale Spatial Data: A Matérn-like GMRF Approach with Circulant and Folded Circulant Approximations

Brynjólfur Gauti Guðrúnar Jónsson

2024-08-15

# Introduction

## Problem Formulation

Consider a spatial field on a regular grid. Our objective is to compute the Gaussian copula density efficiently for this field. This computation involves:

1. Specifying an precision matrix that represents the spatial dependence structure.
2. Ensuring the implied covariance matrix has unit diagonal elements.
3. Computing the log determinant, , and the quadratic form where

## Review

Gaussian Markov Random Fields (GMRFs) and copulas are two powerful statistical tools, each offering unique strengths in modeling complex data structures. GMRFs excel in capturing spatial and temporal dependencies, particularly in fields such as environmental science, epidemiology, and image analysis. Their ability to represent local dependencies through sparse precision matrices makes them computationally attractive for high-dimensional problems. Copulas, on the other hand, provide a flexible framework for modeling multivariate dependencies, allowing separate specification of marginal distributions and their joint behavior.

The Gaussian copula, in particular, has gained popularity due to its interpretability and connection to the multivariate normal distribution. However, combining GMRFs with copulas has historically been computationally challenging, limiting their joint application to smaller datasets or simpler models.

Let be a multivariate random vector with marginal distribution functions for . The joint distribution function of can be written as:

where is the Gaussian copula defined by the GMRF precision matrix . The Gaussian copula is given by:

where is the joint cumulative distribution function of a multivariate normal distribution with mean vector and precision matrix , and is the inverse of the standard normal cumulative distribution function.

A critical requirement for the precision matrix governing the GMRF copula is that should have a unit diagonal, i.e. the marginal variance is equal to one everywhere.. This ensures it operates on the same scale as the transformed data, . However, this can be challenging as GMRFs are typically defined in terms of precision matrices that often imply non-unit marginal variances.

This paper presents a novel algorithm that bridges the gap between GMRFs and copulas, allowing for fast and efficient computation of Gaussian copula densities using GMRF precision structures. Our method focuses on creating a Matérn-like precision matrix with unit marginal variance and efficiently computing the multivariate Gaussian copula density of , where , .

The key innovation lies in leveraging the special structure of the precision matrix:

where is the precision matrix of a standardized one-dimensional AR(1) process with correlation and denotes the Kronecker product. By employing efficient eigendecomposition techniques, our method avoids explicit formation and inversion of the large precision matrix , making it particularly suitable for high-dimensional spatial data. In addition to the exact method, we show how the precision matrix can be approximated by a folded circulant matrix wich gives a large speed-up while preserving suitable boundary conditions.

# Methods

## Gaussian Copula Density Computation

The Gaussian copula density for a random vector with is given by:

where with , is the precision matrix, and is the identity matrix.

The log-density can be expressed as:

Our goal is to efficiently compute this log-density for large spatial fields.

## Precision Matrix Structure

We define the precision matrix as:

where is the precision matrix of a one-dimensional AR(1) process with correlation :

The matrix, , is then scaled so that its inverse, is a correlation matrix, i.e. .

## Computation Process

### Step 1: Eigendecomposition of

We first compute the eigendecomposition of both $:

where is the matrix of eigenvectors and is the diagonal matrix of eigenvalues. Then, because of how is defined, its eigendecomposition is:

We don’t work with the full eigendecomposition, but rather utilize the fact that the eigenvalues of are and their corresponding eigenvectors are to iterate over each value and vector pair to compute the density without forming the larger matrix.

### Step 2: Computation of Marginal Standard Deviations

In order to scale so that its inverse is a correlation matrix, we first calculate , . We then use these marginal standard deviations to scale the eigenvectors and values. The inverse of is given by:

The diagonal elements, , are given by:

This means that the ’th marginal variance, , is a weighted sum of the reciprocals of the eigenvalues of where the weights are the squares of the ’th value in each eigenvector. This means that we can calculate the marginal standard deviations by iterating over the eigenvalues and -vectors of and and cumulating their values according to the formula above, then taking the element-wise square roots.

### Step 3: Scaling the Eigendecomposition

To scale the eigendecomposition of using the marginal standard deviations, we define a diagonal matrix , where and scale the precision matrix as:

In practice, we don’t scale the whole eigendecomposition. Instead, we rescale each value/vector pair individually as we iterate over the eigenvectors and values of and to create the corresponding values and vectors for the larger matrix.

### Step 4: Efficient Computation of Log-Density

Using this scaled eigendecomposition, we efficiently compute:

1. Log-determinant: , where is after rescaling with marginal standard deviations.
2. Quadratic form: , where .

This approach allows us to calculate the density of the spatial copula by calculating and iterating over the spectral decomposition of the smaller matrices, avoiding the formation of alltogether.

## Circulant and Folded Circulant Approximations

While the eigendecomposition method provides an exact solution, it can be computationally expensive for very large spatial fields. To address this, we introduce circulant and folded circulant approximations that offer potential computational advantages.

### Circulant Matrices

A circulant matrix is a special kind of matrix where each row is a cyclic shift of the row above it. It can be fully specified by its first row or column, called the base :

The base vector completely determines the circulant matrix and plays a crucial role in efficient computations. In particular:

1. The eigenvalues of are given by the Discrete Fourier Transform (DFT) of :
2. Matrix-vector multiplication can be performed using the FFT:
3. When is non singular, then the inverse is circulant and thus determined by its base:

These properties allow for much faster computations than for general matrices.

### Block Circulant Matrices

For two-dimensional spatial fields, we use block circulant matrices with circulant blocks (BCCB). An matrix C is block circulant if it has the form:

where each is itself a circulant matrix.

For a BCCB matrix, we define a base matrix , which is an matrix where each column is the base vector of the corresponding circulant block. This base matrix completely determines the BCCB matrix and is central to efficient computations:

1. The eigenvalues of are given by the 2D DFT of .
2. Matrix-vector multiplication can be performed using the 2D FFT.
3. When is non singular, then the inverse is also a BCCB matrix and thus determined by its base matrix.

### Approximations for

Let be the precision matrix of a one-dimensional AR(1) process with correlation . The exact form of is:

#### Circulant Approximation

The circulant approximation to , denoted as , is:

This approximation treats the first and last observations as neighbors, effectively wrapping the data around a circle.

#### Folded Circulant Approximation

The folded circulant approximation, , is based on a reflected version of the data. We double the data by reflecting it, giving us the data . We then model this doubled data with a circulant matrix. If written out as an matrix, it takes the form:

This approximation modifies the first and last diagonal elements to account for the reflection of the data. As now is the first and last data point, then we avoid the circular dependence from the regular circulant approximation.

### Extension to the Full Q Matrix

For a two-dimensional spatial field on an grid, we construct the full precision matrix Q using a Kronecker sum:

where denotes the Kronecker product, is the identity matrix, and is a smoothness parameter.

When we approximate with a circulant matrix, this Kronecker sum results in a block-circulant matrix with circulant blocks (BCCB). To see this, let’s consider the case where for simplicity:

Now, let the two AR(1) matrices be approximated by a circulant matrix with base vector . Then:

and is defined similarly. The Kronecker product results in a block matrix where each block is a scalar multiple of :

Similarly, results in a block diagonal matrix where each block is a copy of :

The sum of these two matrices is a block-circulant matrix with circulant blocks:

where each is a circulant matrix. Specifically,

This BCCB structure allows us to use 2D FFT for efficient computations. The base matrix for this BCCB structure is:

This base matrix captures the structure of the precision matrix and allows for efficient computation of eigenvalues using the 2D Fast Fourier Transform (FFT), enabling rapid calculation of the log-determinant and quadratic forms needed for the Gaussian copula density.

## Computation with Circulant Approximation

When using the circulant approximation, we leverage the efficient computation properties of block circulant matrices with circulant blocks (BCCB). This approach significantly reduces the computational complexity, especially for large spatial fields. Here’s the step-by-step process:

### 1. Construct the Base Matrix

First, we construct the base matrix for our BCCB approximation of . For an grid, is an matrix:

This base matrix encapsulates the structure of our Matérn-like precision matrix.

### 2. Compute Initial Eigenvalues

We compute the initial eigenvalues of using the 2D Fast Fourier Transform (FFT) of :

where ν is the smoothness parameter.

### 3. Compute Marginal Variance and Rescale Eigenvalues

An important property of Block Circulant with Circulant Blocks (BCCB) matrices is that the inverse of a BCCB matrix is also a BCCB matrix, and the marginal variance is the first element in its first circulant block. We use this to efficiently compute the marginal variance and rescale the eigenvalues:

1. Compute the element-wise inverse of :
2. Compute the base of using inverse 2D FFT:
3. The marginal variance is given by the first element of :
4. Rescale the eigenvalues:

This process ensures that the resulting precision matrix will have unit marginal variances, as required for the Gaussian copula.

### 4. Compute Log-Determinant

The log-determinant of the scaled can be efficiently calculated as the sum of the logarithms of the scaled eigenvalues:

### 5. Compute Quadratic Form

To compute the quadratic form , we use the following steps:

1. Compute the 2D FFT of z:
2. Multiply element-wise with the scaled eigenvalues:
3. Compute the inverse 2D FFT:
4. Compute the dot product:

### 6. Compute the Log-Density

Finally, we can compute the log-density of the Gaussian copula:

where .

## Computation with Folded Circulant Approximation

The folded circulant approximation offers an alternative approach that can provide better accuracy near the edges of the spatial field. This method is based on the idea of reflecting the data along each coordinate axis, effectively doubling the size of the field. Other than that, the algorithmic implementation is the same except that the circulant approximation matrices to are now .

First, we reflect the data along each coordinate axis. For a 2D spatial field represented by an matrix, the reflected data takes the form:

This reflection creates a matrix. The matrix is then stacked in lexicographic order before entering into the quadratic forms.

# Results

## Computational Efficiency

Table 1 presents the results of a benchmark comparing the time it takes to evaluate the gaussian copula density described above. For each grid size, we report the computation time for the exact method and the two approximations, along with the speed-up factor relative to the exact method. Each calculation was performed twenty times and the median times are shown in the table.

Table 1. Benchmarking how long it takes to evaluate the density of a Mátern()-like field with correlation parameter , scaled to have unit marginal variance.

| Q\_size | Cholesky | Eigen | Eigen Speed-Up | Circulant | Circulant Speed-Up | Folded | Folded Speed-Up |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 100x100 | 27.14µs | 96.86µs | 0.28x | 28.3µs | 0.96x | 35.88µs | 0.76x |
| 400x400 | 335.09µs | 270.48µs | 1.24x | 35.0µs | 9.58x | 113.12µs | 2.96x |
| 900x900 | 1.95ms | 758.11µs | 2.57x | 87.4µs | 22.28x | 178.17µs | 10.93x |
| 1600x1600 | 6.08ms | 1.96ms | 3.11x | 110.4µs | 55.12x | 332.16µs | 18.31x |
| 2500x2500 | 15.24ms | 4.43ms | 3.44x | 141.0µs | 108.07x | 489.25µs | 31.15x |
| 3600x3600 | 31.28ms | 8.53ms | 3.67x | 175.2µs | 178.52x | 653.15µs | 47.89x |
| 4900x4900 | 68.73ms | 16.37ms | 4.2x | 229.8µs | 299.02x | 864.75µs | 79.48x |
| 6400x6400 | 121.65ms | 30.95ms | 3.93x | 300.1µs | 405.38x | 1.16ms | 104.96x |
| 8100x8100 | 196.57ms | 47.82ms | 4.11x | 470.5µs | 417.83x | 1.55ms | 126.68x |
| 10000x10000 | 288.79ms | 74.03ms | 3.9x | 429.8µs | 671.87x | 1.74ms | 166.37x |

# Appendix

## Cholesky Methods

Standard methods of evaluating multivariate normal densities using the Cholesky decomposition were implemented to compare with the new methods for benchmarking.

### Unscaled Precision Matrix

#### Precision Matrix Construction

We start by constructing the precision matrix for a 2D Matérn field on a grid of size :

where denotes the Kronecker product, and are 1D precision matrices for the x and y dimensions respectively (typically AR(1)-like structures), and and are identity matrices of appropriate sizes.

#### Density Computation

For a Matérn field with smoothness parameter , we need to work with . We can efficiently compute the log-determinant, , and the quadratic form, , without explicitly forming . To do this, we compute the Cholesky decomposition, , where L is a lower triangular matrix, and make use of the following equations:

#### Algorithm

1. Construct
2. Compute Cholesky decomposition
3. Compute log-determinant:
4. For each observation :
   1. Initialize
   2. For from 0 to :
      * If is even:
      * If is odd:
   3. Compute quadratic form
5. Compute log-density:

### Scaled Precision Matrix

#### Precision Matrix Construction

We start by constructing the precision matrix for a 2D Matérn field on a grid of size :

where denotes the Kronecker product, and are 1D precision matrices for the x and y dimensions respectively (typically AR(1)-like structures), and and are identity matrices of appropriate sizes. We will then have to work with the matrix .

To ensure unit marginal variances, we need to scale this precision matrix. Let be a diagonal matrix where , and . The scaled precision matrix is then:

#### Efficient Computation of Scaling Matrix D

1. Compute the Cholesky decomposition of the original
2. Compute , so that .
3. We then calculate the entries in using the following steps:
   1. For , , the column-wise norm of .
   2. For , we use the column-wise norm of
   3. For , we use the column-wise norm of

#### Log determinant

1. First, note that
2. We can compute directly from the diagonal elements of D, i.e.
3. For , we can use the properties of the Cholesky decomposition:
4. Combining these, we get

#### Quadratic Form

1. First, note that
2. Let . We can compute this element-wise as
3. Now we compute as in the unscaled case.

#### Algorithm

Putting it all together, here’s the algorithm for computing the log-density of the Gaussian copula using the scaled precision matrix:

1. Construct
2. Compute Cholesky decomposition
3. Compute and use it to compute D as described earlier
4. Compute log-determinant:
5. For each observation :
   1. Compute
   2. Compute as in the unscaled case.
6. Compute log-density: