Journal of Statistical Software

November 2013, Volume 55, Issue 9. *http://www.jstatsoft.org/*

On Circulant Embedding for Gaussian Random Fields in R

Tilman M. Davies
University of Otago

David J. Bryant
University of Otago

Abstract

The high-dimensionality typically associated with discretized approximations to Gaussian random fields is a considerable hinderance to computationally efficient methods for their simulation. Many direct approaches require spectral decompositions of the associated covariance matrix and so are unable to complete the solving process in a timely fashion, if at all. However under certain conditions, we may construct block-circulant versions of the covariance matrix at hand thereby allowing access to fast-Fourier methods to perform the required operations with impressive speed. We demonstrate how circulant embedding and subsequent simulation can be performed directly in the R language. The approach is currently implemented in C for the R package **RandomFields**, and used in the recently released package **lgcp**. Motivated by applications dealing with spatial point processes we restrict attention to stationary Gaussian fields on $\mathbb{R}^2$, where sparsity of the covariance matrix cannot necessarily be assumed.

Keywords: multivariate normal, fast-Fourier transform, point pattern.

1. Introduction

The utility of the *fast-Fourier transform* (FFT) in reducing the computational costs associated with the evaluation of complicated problems is well understood. It can occasionally be difficult, however, to interpret the rather abstract notions associated with translating this technique to fit given applications. To this end, authors of more detailed works provide generalized instructions on how it may be performed. For example, Wand and Jones (1995, Appendix D) demonstrate how the FFT is employed to dramatically increase the speed of kernel density estimation, and Dietrich and Newsam (1993) and Wood and Chan (1994) discuss use of the FFT in terms of multidimensional Gaussian fields, techniques subsequently accessed for spatial point pattern modeling using the log-Gaussian Cox process (LGCP) by Møller, Syversveen, and Waagepetersen (1998).

It can be especially useful, particularly from the point of view of users of the methodology, to express what may be considered as rather abstract mathematical notions in an applied, intuitively sensible fashion. Motivated primarily by the analysis of planar point patterns and their underlying intensity functions, the core aim of this work is to therefore augment, in a practical sense, the theory-based instructions in Appendix E of Møller and Waagepetersen (2004) for simulation of Gaussian fields on a restricted region $W \subset \mathbb{R}^2$ via *circulant embedding* and the two-dimensional FFT. This goal is achieved in the form of a simple tutorial in R (R Core Team 2013), with conceptual illustrations supported by a clear coding strategy.

The remainder of this paper is structured as follows. Section 2 introduces the aim of generating a realization of a spatially continuous Gaussian field on a finite subset of the plane by considering the discretization of the region into a set of disjoint cells. Section 3 outlines the procedure that must be taken in order to validate use of the FFT for the Gaussian field on W, by considering a torus-wrapped extension of a rectangular lattice over the region. The speed and efficiency of the FFT approach in generating single realizations within the local R environment is compared against alternative methods in Section 4, and Section 5 times multiple realizations. Concluding remarks are provided in Section 6.

2. Basic concept

Consider a stationary, spatially continuous Gaussian field $\mathcal{Y}$ on $\mathbb{R}^2$ with mean $\mu \equiv \mu(\boldsymbol{x})$ and stationary, isotropic covariance function $\sigma^2 r(\|\boldsymbol{x}-\boldsymbol{y}\|, \phi)$, where σ^2 is the (scalar) variance, $\|\cdot\|$ denotes Euclidean distance and r is a specified correlation function which also depends on a scale parameter ϕ. We are typically interested in $\mathcal{Y}$ on $W \subset \mathbb{R}^2$; $\boldsymbol{x}, \boldsymbol{y} \in W$, with W a bounded region with Lebesgue measure $|W| > 0$. To simulate a realization of $\mathcal{Y}$ in practice, we must discretize W into a suitably fine set of d disjoint cells, each of which is assumed to be a single component of the d-dimensional multivariate Gaussian variable Y. The simplest strategy (and as it happens, a strategy which works well for implementation of the FFT) is to construct a $(M+1) \times (N+1)$ grid over the encapsulating rectangle $\mathcal{A}_W$; $M, N \geq 1$, so that

$$Y \sim \mathrm{N}(\boldsymbol{\mu}, \Sigma_Y), \tag{1}$$

where, according to our assumptions about the process, $\boldsymbol{\mu}$ is the $d = MN$-dimensional vector of the stationary process mean μ, and Σ_Y is the $MN \times MN$-dimensional covariance matrix (owing to the fact we use the grid cell *centroids* to define the dependence structure – this will become clearer in a moment). The covariance matrix is computed from the $MN \times MN$-dimensional set of Euclidean distances D; see Equation 4 below.

Represent the maximum ranges spanning W (and hence $\mathcal{A}_W$) along the x- and y-axes with $R_x(\mathcal{A})$, and $R_y(\mathcal{A})$ respectively. Defining $\Delta_x = R_x(\mathcal{A})/M$ and $\Delta_y = R_y(\mathcal{A})/N$ to be the horizontal and vertical grid spacings respectively, a grid I is constructed over $\mathcal{A}_W$. This is given as

$$I = \left[(x_{\min} + \Delta_x m, y_{\min} + \Delta_y n) : m \in \{0, \ldots, M\}, n \in \{0, \ldots, N\}\right], \tag{2}$$

where obviously $R_x(I) \equiv R_x(\mathcal{A})$ and $R_y(I) \equiv R_y(\mathcal{A})$, and $x_{\min}$, $y_{\min}$ are the lowest (bottom left) coordinates of $\mathcal{A}_W$. Important calculations such as finding inter-cell distances and determining region inclusion are defined in terms of the lattice of cell centroids:

$$C^{(m,n)} = [x_{\min} + (m-0.5)\Delta_x, y_{\min} + (n-0.5)\Delta_y]; \quad m = 1, \ldots, M; n = 1, \ldots, N. \tag{3}$$

The covariance matrix is computed from the $MN \times MN$ cell-centroid-wise matrix of Euclidean distances D. Let scalar indices $i \in \{1, \ldots, MN\}$ and $j \in \{1, \ldots, MN\}$ uniquely reference each centroid (m, n) within I. Then

$$D[i, j] = \|C^{(i)} - C^{(j)}\|, \tag{4}$$

thus allowing

$$\Sigma_Y[i, j] = \sigma^2 r(D[i, j]; \phi).$$

For simplicity, we will set $M = N$ in this example. Figure 1 displays an irregular study region W upon which I defined as a 30×30 rectangular grid (hence 29×29 cells) has been superimposed. (Aside: This is the spatial window for the Chorley-Ribble cancer data available in the package **spatstat**; Baddeley and Turner 2005). The spatial window is obtained with

```
R> library("spatstat")
R> data("chorley", package = "spatstat")
R> W <- chorley$window
R> AW <- as.rectangle(W)
R> W

window: polygonal boundary
enclosing rectangle: [343.45, 366.45] x [410.41, 431.79] km
```

To construct I and obtain the grid centroids as per Equations 2 and 3, we enlist a helper function `grid.prep`, defined fully in the replication code in the supplementary files. Briefly, this function takes a polygonal (or rectangular) spatial window of class `owin` from **spatstat** as `W`, and requires the user to specify the number of cells in the horizontal `M` and vertical `N` directions (M and N in our present notation). The final argument, `ext`, is used in circulant embedding procedures and will be explained in Section 3.

```
R> M <- N <- 29
R> mygrid <- grid.prep(W = W, M = M, N = N, ext = 2)
```

The returned object `mygrid` is a named list comprised of numeric vectors giving the grid and centroid locations, as well as additional information relevant to the lattice which will be explained in the article as needed. Using `mygrid`, the code producing Figure 1 is given in the replication code in the supplementary files.

Using the horizontal and vertical centroid coordinates (`mcens` and `ncens` respectively) from the list returned by `grid.prep`, we compute the inter-cell distance matrix D. This is straightforwardly achieved by taking (squared) differences between axis-specific 'coordinate matrices' with transposed versions of themselves:

```
R> cent <- expand.grid(mygrid$mcens, mygrid$ncens)
R> mmat <- matrix(rep(cent[, 1], M * N), M * N, M * N)
R> nmat <- matrix(rep(cent[, 2], M * N), M * N, M * N)
R> D <- sqrt((mmat - t(mmat))^2 + (nmat - t(nmat))^2)
```

Now assume, for the sake of illustration, we have defined r to be the exponential correlation function such that $r(u) = \exp\{-|u|/\phi\}$. After specifying desired variance and scale parameters for the process, we find the covariance matrix corresponding to Y as

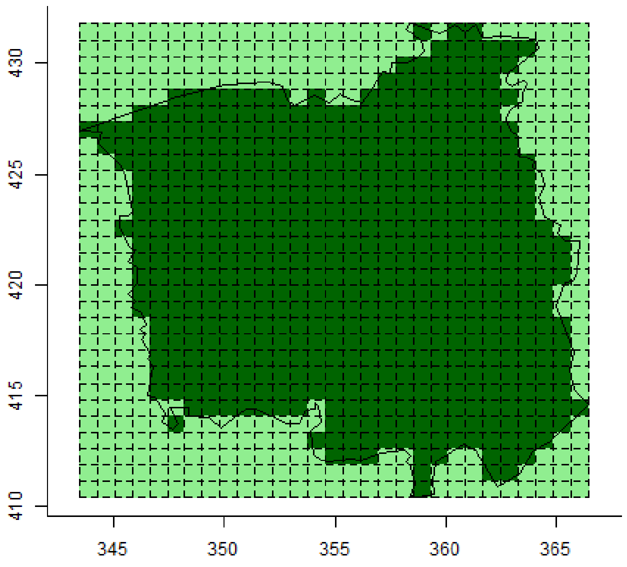

Figure 1: The regular rectangular grid I constructed over $\mathcal{A}_W$. A realization of Y will initially occur over all cells in I. Dark green shading indicates those cells whose centroids fall inside the irregular W, light green otherwise.

```
R> sigma <- 5
R> phi <- 1
R> r <- function(u, phi) exp(-abs(u)/phi)
R> SIGMA.Y <- sigma^2 * r(D, phi)
R> dim(SIGMA.Y)

[1] 841 841

R> SIGMA.Y[1:6, 1:6]

           [,1]      [,2]      [,3]      [,4]      [,5]       [,6]
[1,] 25.0000000 11.310962  5.117515  2.315361  1.047558  0.4739557
[2,] 11.3109624 25.000000 11.310962  5.117515  2.315361  1.0475583
[3,]  5.1175148 11.310962 25.000000 11.310962  5.117515  2.3153607
[4,]  2.3153607  5.117515 11.310962 25.000000 11.310962  5.1175148
[5,]  1.0475583  2.315361  5.117515 11.310962 25.000000 11.3109624
[6,]  0.4739557  1.047558  2.315361  5.117515 11.310962 25.0000000
```

Assuming provision of a mean value μ, we are now ready to simulate realizations of this particular field. For simulation of $Y(\boldsymbol{u})$, $\boldsymbol{u} \in \mathcal{A}_W$, we require the eigendecomposition of Σ_Y; an operation that can be computed faster and more efficiently with the FFT (in comparison to other methods) when dealing with *circulant*, and in our two-dimensional application, *block circulant* matrices. Albeit symmetrical, Σ_Y is in its current form not block-circulant. To overcome this issue, and therefore have access to the Fourier transform methods, we must artificially embed Σ_Y in its own circulant matrix.

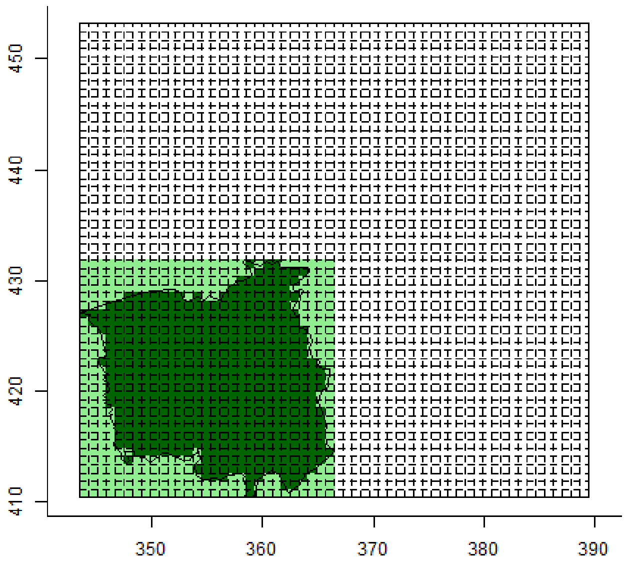

Figure 2: The extended rectangular grid I_{ext} over $\mathcal{A}_{\text{ext},W}$. Once more, dark green shading indicates those cells whose centroids fall inside the irregular W, light green otherwise.

3. Circulant embedding

Circulant embedding of Σ_Y is achieved by considering Y on an extended $(M_{\text{ext}}+1)\times(N_{\text{ext}}+1)$ grid at least twice the cell-size of the original, in the sense that $M_{\text{ext}} \geq 2M$ and $N_{\text{ext}} \geq 2N$. This is necessary to avoid the behavior of the field within the sub-region of interest, namely W, becoming contaminated by so-called 'wrap-around' effects owing to the periodic nature of the Fourier transform.

Denote the extended encapsulating rectangle by $\mathcal{A}_{W,\text{ext}}$. We now consider simulation of

$$Y_{\text{ext}} \sim \mathrm{N}(\boldsymbol{\mu}_{\text{ext}}, \Sigma_{Y_{\text{ext}}}), \tag{5}$$

where $\boldsymbol{\mu}_{\text{ext}}$ and $\Sigma_{Y_{\text{ext}}}$ are defined as in Equation 1, except now increased in size to correspond to the extended grid. The new grid will be denoted with I_{ext}:

$$I_{\text{ext}} = \left[(x_{\min} + \Delta_x m, y_{\min} + \Delta_y n) : m \in \{0,\ldots,M_{\text{ext}}\}, n \in \{0,\ldots,N_{\text{ext}}\}\right], \tag{6}$$

giving a new set of centroids

$$C_{\text{ext}}^{(m,n)} = [x_{\min} + (m-0.5)\Delta_x, y_{\min} + (n-0.5)\Delta_y]; \quad m = 1,\ldots,M_{\text{ext}}, n = 1,\ldots,N_{\text{ext}}. \tag{7}$$

Suppose we wish to extend I from the previous section the minimum permitted amount, setting $M_{\text{ext}} = N_{\text{ext}} = 58$. In our earlier execution of `grid.prep` this is precisely what was achieved by setting `ext = 2` (the extension factor). Included in `mygrid` are vectors corresponding to the coordinates of I_{ext} and $C_{\text{ext}}^{(m,n)}$ for this extended grid size. Figure 2 shows the original grid I extended to I_{ext}; code is given in the supplementary files.

The block-circulant covariance matrix $\Sigma_{Y_{\text{ext}}}$ is now obtained using the extended cell-centroid-wise Euclidean distance matrix D_{ext}. This is not, however, computed with 'raw' Euclidean

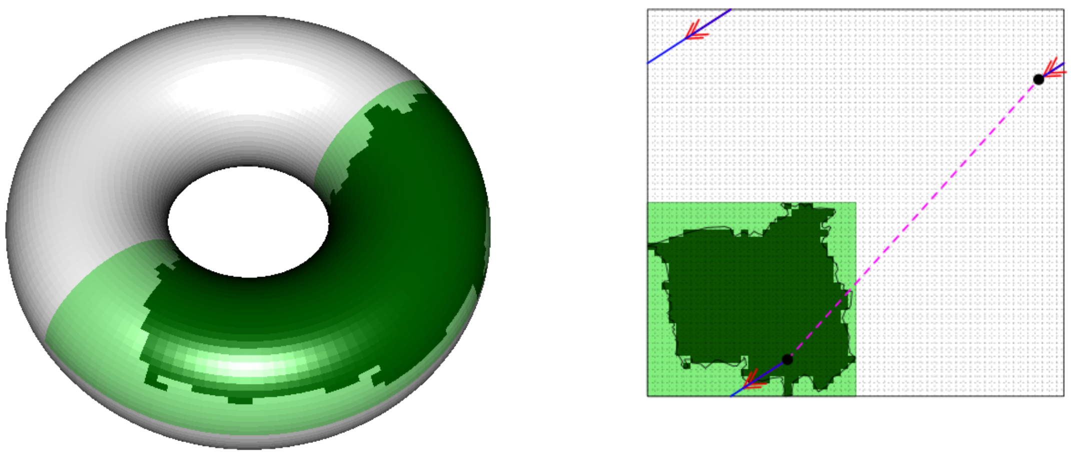

Figure 3: Left: An abstract 'inflated' representation of the extended lattice of cells wrapped on a torus. Right: Comparing raw (dashed, pink) and minimum torus (solid, blue) Euclidean distance calculation between two distinct cell centroids (•, black) on the extended lattice. Directional markers (<<, red) on the solid lines indicate the continuity of the extended grid as a torus.

distances between distinct centroids in C_{ext}. To satisfy the circulant requirement for implementation of the FFT, the distances upon which $\Sigma_{Y_{\text{ext}}}$ is defined must correspond to wrapping the extended lattice on a torus and only thereafter finding the *minimum* Euclidean distances. Wood and Chan (1994), Section 6 of Møller *et al.* (1998) and Appendix E in Møller and Waagepetersen (2004) give details on how this is achieved. Briefly,

$$\begin{aligned} D_{\text{ext}}[i,j] = \Big\{ &\min\left[\text{abs}_m(i,j), R_x(\mathcal{A}_{W,\text{ext}}) - \text{abs}_m(i,j)\right]^2 \\ &+ \min\left[\text{abs}_n(i,j), R_y(\mathcal{A}_{W,\text{ext}}) - \text{abs}_n(i,j)\right]^2 \Big\}^{\frac{1}{2}}, \end{aligned} \tag{8}$$

where $R_x(\mathcal{A}_{W,\text{ext}})$ and $R_y(\mathcal{A}_{W,\text{ext}})$ represent the width and height of $\mathcal{A}_{W,\text{ext}}$. Furthermore, denoting the horizontal and vertical positions of centroid i with $C^{(i)}_{\text{ext},m}$ and $C^{(i)}_{\text{ext},n}$ respectively, $\text{abs}_m(i,j) = |C^{(i)}_{\text{ext},m} - C^{(j)}_{\text{ext},m}|$; subscript n indicating vertical differences.

Figure 3 gives an abstract impression of the torus-wrapping procedure with respect to the study region of interest (3D plotting achieved using package **rgl**; see Adler and Murdoch 2012), and shows the difference between what we will term the 'raw' Euclidean distance on the extended grid, and the minimum distance on the torus-wrapped version thereof, between two arbitrarily chosen locations. As we can see for this particular pair of centroids, the distance matrix would include the torus distance.

To compute the required circulant difference matrix D_{ext} for our example, we utilize the components `M.ext` and `N.ext` (scalars – the extended grid size); `cell.width` and `cell.height` (scalars – the cell size); and `mcens.ext` and `ncens.ext` (vectors of length `M.ext` and `N.ext` respectively – the centroids of the extended grid along both axes) from `mygrid`. The following code computes $R_x(\mathcal{A}_{W,\text{ext}})$ (`Rx`) and $R_y(\mathcal{A}_{W,\text{ext}})$ (`Ry`), and uses the same axis-specific coordinate matrix transpose-differencing as earlier. Equation 8 is directly reflected in the last three lines:

```
R> Rx <- mygrid$M.ext * mygrid$cell.width
R> Ry <- mygrid$N.ext * mygrid$cell.height
R> MN.ext <- mygrid$M.ext * mygrid$N.ext
R> cent.ext <- expand.grid(mygrid$mcens.ext, mygrid$ncens.ext)
R> mmat.ext <- matrix(rep(cent.ext[, 1], MN.ext), MN.ext, MN.ext)
R> nmat.ext <- matrix(rep(cent.ext[, 2], MN.ext), MN.ext, MN.ext)
R> mmat.diff <- mmat.ext - t(mmat.ext)
R> nmat.diff <- nmat.ext - t(nmat.ext)
R> mmat.torus <- pmin(abs(mmat.diff), Rx - abs(mmat.diff))
R> nmat.torus <- pmin(abs(nmat.diff), Ry - abs(nmat.diff))
R> D.ext <- sqrt(mmat.torus^2 + nmat.torus^2)
```

Finally, the covariance matrix is again simply computed with:

```
R> SIGMA.Y.ext <- sigma^2 * r(D.ext, phi)
R> dim(SIGMA.Y.ext)

[1] 3364 3364
```

As simulation will require eigendecomposition of the covariance matrix, we will for illustrative purposes store the eigenvalues of Σ_{Y_ext}; displaying here the first six:

```
R> ext.eigs <- eigen(SIGMA.Y.ext, symmetric = TRUE, only.values = TRUE)
R> ext.eigs <- ext.eigs$values
R> head(ext.eigs)

[1] 272.9771 265.6322 265.6322 264.5067 264.5067 257.5406
```

The block circulant nature of the extended, torus-wrapped covariance matrix means all the required information is stored within the *first row* of Σ_{Y_ext} should we desire fast decomposition using the FFT. This suggests the above commands, resulting in a large $M_\text{ext}N_\text{ext} \times M_\text{ext}N_\text{ext} = 3364 \times 3364$ structure, exhibit much redundancy. Under our current coding strategy, the first-row entries correspond to the torus distances from the first, 'bottom-left' centroid to the others. Via Equation 7, we would therefore only need to compare $C_\text{ext}^{(1,1)}$ to $C_\text{ext}^{(m,n)}$, instead of the full $C_\text{ext}^{(m,n)}$ to $C_\text{ext}^{(m,n)}$; $m \in \{1,\ldots,M_\text{ext}\}, n \in \{1,\ldots,N_\text{ext}\}$, to obtain the torus-wrapped distances needed. Eigendecomposition is then able to be performed on the corresponding covariance values from application of r, once these results are trivially arranged in a (substantially smaller!) $M_\text{ext} \times N_\text{ext}$ matrix we will denote with $\tilde{C}$ (explicitly named `SIGMA.Y.ext.row1` below). This construct is referred to as the *base matrix* by Rue and Held (2005). Observe:

```
R> m.abs.diff.row1 <- abs(mygrid$mcens.ext[1] - mygrid$mcens.ext)
R> m.diff.row1 <- pmin(m.abs.diff.row1, Rx - m.abs.diff.row1)
```

```
R> n.abs.diff.row1 <- abs(mygrid$ncens.ext[1] - mygrid$ncens.ext)
R> n.diff.row1 <- pmin(n.abs.diff.row1, Ry - n.abs.diff.row1)
R> cent.ext.row1 <- expand.grid(m.diff.row1, n.diff.row1)
R> D.ext.row1 <- matrix(sqrt(cent.ext.row1[, 1]^2 + cent.ext.row1[, 2]^2),
+    mygrid$M.ext, mygrid$N.ext)
R> SIGMA.Y.ext.row1 <- sigma^2 * r(D.ext.row1, phi)
R> dim(SIGMA.Y.ext.row1)
```

```
[1] 58 58
```

Earlier, we computed and stored the eigenvalues of the *full* extended covariance matrix from above as `ext.eigs`. Using `SIGMA.Y.ext.row1`, the same decomposition is able to be performed using R's `fft` function (package **stats**), which provides the fast discrete Fourier transform operations:

```
R> ext.eigs.row1 <- rev(sort(Re(fft(SIGMA.Y.ext.row1, inverse = TRUE))))
R> head(ext.row1.eigs)
```

```
[1] 272.9771 265.6322 265.6322 264.5067 264.5067 257.5406
```

In fact, this code can be seen as a direct reflection of the theoretical comments in Appendix E of Møller and Waagepetersen (2004) on page 258. Using the same lattice definitions as we do, they note that the matrix of the eigenvalues for $\Sigma_{Y_{\text{ext}}}$, $\tilde{\Lambda}$, is found via the operation $\sqrt{M_{\text{ext}}N_{\text{ext}}} \times \bar{F}_{M_{\text{ext}}}\tilde{C}\bar{F}_{N_{\text{ext}}}$, where F_L represents the normalized $L \times L$ discrete Fourier transform matrix, $\bar{F}_L$ its inverse, and $\tilde{C}$ is, as mentioned above, the $M_{\text{ext}} \times N_{\text{ext}}$ matrix of the first row of the torus-wrapped extended covariance matrix precisely as we have constructed the object `SIGMA.Y.ext.row1`. As the R command `fft` computes the *unnormalized* Fourier transform, the leading $\sqrt{M_{\text{ext}}N_{\text{ext}}}$ term in the expression above is incorporated automatically, and the transformation operation itself is simply achieved by executing `fft` on `SIGMA.Y.ext.row1`, setting the argument `inverse = TRUE`.

Further inspection of the sorted eigenvalue vectors `ext.eigs` and `ext.eigs.row1` (each are of length 58^2) shows us that identical results are obtained. Thus, the usefulness of the FFT in this situation in terms of speed and efficiency begins to become apparent – only the computation and storage of the relatively small $M_{\text{ext}} \times N_{\text{ext}}$ structure has been required in order for decomposition to take place.

4. Single field realizations

Here, we compare the specific R commands and approximate execution times for simulation of single variates of a Gaussian random field on the specific W studied in the previous two sections. The FFT approach will be compared to two other well-known methods: the eigenvalue decomposition (EIGEN) and the Cholesky decomposition (CHOL). A nominal example of code using circulant embedding/FFT for generating a Gaussian random field, via the `GaussRF` function available in the R package **RandomFields** (Schlather 2012), is presented in the replication code in the supplementary files.

4.1. Methods

Generation of the multivariate normal variables which follow the distribution of interest in all three cases amounts to the initial, computationally cheap, generation of a standard normal vector of the appropriate length, followed by a particular set of matrix operations involving the decomposed matrix structures.

The eigenvalue decomposition uses numerical routines to identify the matrices Q and Λ, where Q is an orthonormal $MN \times MN$ matrix of the eigenvectors of the specified covariance matrix Σ_Y, and Λ is the corresponding diagonal matrix of eigenvalues. It is subsequently recognized that $Q\Lambda Q^\top = \Sigma_Y$. To generate a realization of the Gaussian random field following Σ_Y, the steps are:

1. Given a $MN \times MN$ covariance matrix, compute the orthonormal Q and diagonal Λ such that $Q\Lambda Q^\top = \Sigma_Y$.
2. Generate a standard normal vector Z with dimension $MN \times 1$.
3. Recover the realization by computing $Q\Lambda^{\odot 1/2}Z$, where we use $B^{\odot b}$ to denote the element-wise power of the matrix B to the scalar b.

In R, the `eigen` function is used to find Q and Λ, which uses algorithms from the LAPACK library (Anderson *et al.* 2000, see also `http://www.netlib.org/lapack/`). The construction of Q and Λ in Step 2 takes $O(M^3N^3)$ time on a dense covariance matrix. Once these are computed and stored, generating Z in Step 1 and calculating $Q\lambda^{\odot 1/2}Z$ takes $O(M^2N^2)$ time.

The Cholesky decomposition (or 'factorization') is a common alternative. For a positive definite symmetric matrix A, the Cholesky factorization involves finding an upper-triangular matrix R with positive diagonal entries, such that $A = R^\top R$. Algorithms for finding R are typically faster than computing the eigenvalues and eigenvectors explicitly as required above. For simulation of the Gaussian random field using CHOL,

1. Given a $MN \times MN$ covariance matrix, compute R.
2. Generate a standard normal vector Z with dimension $MN \times 1$.
3. Recover the realization by computing $R^\top Z$.

In R, `chol` computes the upper-triangular matrix R. The computation has the same overall time complexity as eigenvector decomposition, namely $O(M^3N^3)$ for the decomposition (assuming a dense covariance matrix, see below), and then $O(M^2N^2)$ per sample (Golub and Van Loan 1989; Gneiting, Ševčíková, Percival, Schlather, and Jiang 2006).

Finally, given the base matrix $\tilde{C}$ (the $M_{\text{ext}} \times N_{\text{ext}}$ structure containing the first row of the extended, torus-wrapped covariance matrix $\Sigma_{Y_{\text{ext}}}$), simulation under the FFT involves the following steps (for which more detail is available in Appendix E of Møller and Waagepetersen 2004):

1. Compute the $M_{\text{ext}} \times N_{\text{ext}}$ matrix of eigenvalues, $\tilde{\Lambda}$, by pre- and post-multiplication of $\tilde{C}$ using the unnormalized inverse Fourier transform matrices of size $M_{\text{ext}} \times M_{\text{ext}}$ and $N_{\text{ext}} \times N_{\text{ext}}$ respectively.

2. Generate a standard normal vector $\tilde{Z}$ with dimension $M_{\text{ext}}N_{\text{ext}} \times 1$.
3. Find the normalized Fourier transform of $\tilde{Z}$, by pre- and post-multiplication of $\tilde{Z}$ using the normalized Fourier transform matrices of size $M_{\text{ext}} \times M_{\text{ext}}$ and $N_{\text{ext}} \times N_{\text{ext}}$ respectively.
4. Compute $\tilde{Z} \circ \tilde{\Lambda}^{\odot 1/2}$, where $\circ$ denotes the Hadamard product.
5. Recover the realization via the normalized, inverse Fourier transform of the quantity calculated in the previous step.

The R function `fft` is responsible for the calculations in Steps 1, 3, and 5 directly above. The use of the FFT results in a significant improvement in time complexity over eigenvector decomposition and Cholesky decomposition (Golub and Van Loan 1989). Each application of the (two-dimensional) FFT takes $O(MN \log MN)$ time, which is therefore the time complexity of the first and every subsequent sample (Møller and Waagepetersen 2004).

4.2. Generation

For our computational illustrations, we will remain with the exponential r with $\sigma^2 = 25$ and $\phi = 1$. Simulations can be performed for any positive, finite values of these parameters as well as any one of a host of other possible functional forms for r, on the proviso that the resulting covariance matrix is positive definite (and in some cases, positive semidefinite). See the documentation for the function `CovarianceFct` from package **RandomFields** (Schlather 2012) for some examples. In all cases, we will set the stationary mean $\mu = 0$.

Execution time for the following examples is best split into two parts: 1) those operations which only need to be performed once regardless of how many variables wish to be generated, i.e., eigen- or Cholesky-decomposition of the covariance matrix, or inverse discrete Fourier transformation of the base matrix; and 2) those operations to generate the standard normal vector of the required dimensions and perform any additional matrix multiplications to recover the appropriate realization (truncated to W). Although the simulations in this section are focused on generating single realizations only, we investigate the relatively minor additional computational impact of the generation of multiple realizations in Section 5.

For ease of presentation, a small function `timings` to compute execution times is defined (see the code in the supplementary files) and used in the following simulations. In each example, this function summarizes the elapsed time taken for the decomposition step (the difference between `t1` and `t2` below) as `Decomp`, the time to actually generate the desired realization (the difference between `t2` and `t3` below) as `Generate`, and the sum of both `Decomp` and `Generate` as `Total elapsed`. Reported times are based on the first author's local desktop machine running Microsoft Windows 7 Professional with an AMD Phenom II X6 1100T 3.30GHz processor and 4.00GB RAM.

Beginning with EIGEN and our illustrative resolution of a 29×29 field on W, we use `SIGMA.Y` from Section 2 to run

```
R> t1 <- Sys.time()
R> eigs <- eigen(SIGMA.Y, symmetric = TRUE)
R> decomp.eigen.29 <- eigs$vectors %*% diag(sqrt(eigs$values))
R> t2 <- Sys.time()
```

```
R> std <- rnorm(M * N);
R> realz <- as.vector(decomp.eigen.29 %*% matrix(std))
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.eigen.29 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

      Decomp        Generate Total elapsed
 1.5888 secs 0.06239986 secs   1.6512 secs
```

This time can be improved upon by implementing CHOL:

```
R> t1 <- Sys.time()
R> decomp.chol.29 <- chol(SIGMA.Y)
R> t2 <- Sys.time()
R> std <- rnorm(M * N)
R> realz <- as.vector(t(decomp.chol.29) %*% matrix(std))
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.chol.29 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

         Decomp        Generate Total elapsed
 0.1499999 secs 0.09000015 secs     0.24 secs
```

The FFT method improves further over CHOL:

```
R> t1 <- Sys.time()
R> d <- dim(SIGMA.Y.ext.row1)
R> dp <- prod(d)
R> sdp <- sqrt(dp)
R> prefix <- sqrt(Re(fft(SIGMA.Y.ext.row1, TRUE)))
R> t2 <- Sys.time()
R> std <- rnorm(dp)
R> realz <- prefix * (fft(matrix(std, d[1], d[2]))/sdp)
R> realz <- as.vector(Re(fft(realz, TRUE)/sdp)[1:M, 1:N])
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.fft.29 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

 Decomp        Generate   Total elapsed
 0 secs 0.01559901 secs 0.01559901 secs
```

The generated fields are given in the top row of Figure 4. For many applications, such as our current interest in the consideration of planar point process intensity functions, this relatively coarse discretization of the field over W may be inappropriate. At the very least the resolution of the grid must be fine enough to adequately capture the scale of dependence present – important small-scale behavior in the field may be missed if the distance between adjacent cell centroids is too great (this is dependent on our choice of correlation function and the value of any scaling parameter such as ϕ). However, even modest increases in resolution can lead to high computational expense. Overwrite the original lattice to give a 64×64 centroid grid and compute the finer covariance structures:

```
R> M <- N <- 64
R> mygrid <- grid.prep(W = W, M = M, N = N, ext = 2)
R> cent <- expand.grid(mygrid$mcens, mygrid$ncens)
R> mmat <- matrix(rep(cent[, 1], M * N), M * N, M * N)
R> nmat <- matrix(rep(cent[, 2], M * N), M * N, M * N)
R> D <- sqrt((mmat - t(mmat))^2 + (nmat - t(nmat))^2)
R> SIGMA.Y <- sigma^2 * r(D,phi)
R> Rx <- mygrid$M.ext * mygrid$cell.width
R> Ry <- mygrid$N.ext * mygrid$cell.height
R> m.abs.diff.row1 <- abs(mygrid$mcens.ext[1] - mygrid$mcens.ext)
R> m.diff.row1 <- pmin(m.abs.diff.row1, Rx - m.abs.diff.row1)
R> n.abs.diff.row1 <- abs(mygrid$ncens.ext[1] - mygrid$ncens.ext)
R> n.diff.row1 <- pmin(n.abs.diff.row1, Ry - n.abs.diff.row1)
R> cent.ext.row1 <- expand.grid(m.diff.row1, n.diff.row1)
R> D.ext.row1 <- matrix(sqrt(cent.ext.row1[, 1]^2 + cent.ext.row1[, 2]^2),
+     mygrid$M.ext, mygrid$N.ext)
R> SIGMA.Y.ext.row1 <- sigma^2 * r(D.ext.row1, phi)
```

The resolution of the new centroid grid was chosen as 64×64 here for the benefit of the FFT generations to follow. It is noted that with our current use of the FFT, computational efficiency can be optimized if M and N are powers of two; cf. Section 3 of Wood and Chan (1994).

Once more beginning with EIGEN:

```
R> t1 <- Sys.time()
R> eigs <- eigen(SIGMA.Y, symmetric = TRUE)
R> decomp.eigen.64 <- eigs$vectors %*% diag(sqrt(eigs$values))
R> t2 <- Sys.time()
R> std <- rnorm(M * N);
R> realz <- as.vector(decomp.eigen.64 %*% matrix(std))
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.eigen.64 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

        Decomp       Generate Total elapsed
 4.052507 mins 0.1871998 secs 4.055627 mins
```

This time, we note a marked increase in the time taken to perform the spectral decomposition – over four minutes. Using CHOL with this finer lattice also gives a noticeably increased execution time, though it is not so severe as EIGEN:

```
R> t1 <- Sys.time()
R> decomp.chol.64 <- chol(SIGMA.Y)
R> t2 <- Sys.time()
R> std <- rnorm(M * N)
R> realz <- as.vector(t(decomp.chol.64) %*% matrix(std))
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.chol.64 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

      Decomp   Generate Total elapsed
 14.4924 secs 1.638 secs  16.1304 secs
```

The FFT method, however, remains extremely cheap:

```
R> t1 <- Sys.time()
R> d <- dim(SIGMA.Y.ext.row1)
R> dp <- prod(d)
R> sdp <- sqrt(dp)
R> prefix <- sqrt(Re(fft(SIGMA.Y.ext.row1, TRUE)))
R> t2 <- Sys.time()
R> std <- rnorm(dp)
R> realz <- prefix * (fft(matrix(std, d[1], d[2]))/sdp)
R> realz <- as.vector(Re(fft(realz, TRUE)/sdp)[1:M, 1:N])
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.fft.64 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

 Decomp        Generate   Total elapsed
 0 secs 0.01559997 secs 0.01559997 secs
```

The computational demands have become apparent owing to the fact that the covariance matrix has now increased in size to a 4096×4096 structure, from the original dimensions of 841×841. These realizations are displayed in the second row of Figure 4.

So how fine can we go before the computational demands associated with constructing the various quantities required for simulation become prohibitive? For a given application, a 64×64 lattice of cells may still be considered too coarse. We can expect some point at which all the above approaches, even FFT, will be practically infeasible if we continue to set higher and higher resolutions. Let us further increase the resolution to $2^7 \times 2^7 = 128 \times 128$, achieved by setting `M <- N <- 128`. Implementation of both EIGEN and CHOL requires the full $MN \times MN = 128^2 \times 128^2$ `SIGMA.Y`. But in attempting to create it,

```
R> M <- N <- 128
R> mygrid <- grid.prep(W = W, M = M, N = N, ext = 2)
R> cent <- expand.grid(mygrid$mcens, mygrid$ncens)
R> mmat <- matrix(rep(cent[, 1], M * N), M * N, M * N)
```

```
Error: cannot allocate vector of size 2.0 Gb
```

and R understandably refuses to complete constructing the appropriately-sized matrix due to memory constraints (though these options can be tailored to a certain extent within the R environment, the presence of such an error is on its own concerning). However, as the FFT method requires only the $M_{\text{ext}} \times N_{\text{ext}} = 256 \times 256$ `SIGMA.Y.ext.row1`,

```
R> M <- N <- 128
R> mygrid <- grid.prep(W = W, M = M, N = N, ext = 2)
R> cent <- expand.grid(mygrid$mcens, mygrid$ncens)
R> Rx <- mygrid$M.ext * mygrid$cell.width
R> Ry <- mygrid$N.ext * mygrid$cell.height
R> m.abs.diff.row1 <- abs(mygrid$mcens.ext[1] - mygrid$mcens.ext)
R> m.diff.row1 <- pmin(m.abs.diff.row1, Rx - m.abs.diff.row1)
R> n.abs.diff.row1 <- abs(mygrid$ncens.ext[1] - mygrid$ncens.ext)
R> n.diff.row1 <- pmin(n.abs.diff.row1, Ry - n.abs.diff.row1)
R> cent.ext.row1 <- expand.grid(m.diff.row1, n.diff.row1)
R> D.ext.row1 <- matrix(sqrt(cent.ext.row1[, 1]^2 + cent.ext.row1[, 2]^2),
+    mygrid$M.ext, mygrid$N.ext)
R> SIGMA.Y.ext.row1 <- sigma^2 * r(D.ext.row1, phi)
```

and no such memory impediment occurs. FFT-based simulation can then proceed exactly as performed earlier:

```
R> t1 <- Sys.time()
R> d <- dim(SIGMA.Y.ext.row1)
R> dp <- prod(d)
R> sdp <- sqrt(dp)
R> prefix <- sqrt(Re(fft(SIGMA.Y.ext.row1, TRUE)))
R> t2 <- Sys.time()
R> std <- rnorm(dp)
R> realz <- prefix * (fft(matrix(std, d[1], d[2]))/sdp)
R> realz <- as.vector(Re(fft(realz, TRUE)/sdp)[1:M, 1:N])
R> realz[!inside.owin(x = cent[, 1], y = cent[, 2], w = W)] <- NA
R> realization.fft.128 <- matrix(realz, M, N, byrow = TRUE)
R> t3 <- Sys.time()
R> timings(t1, t2, t3)

         Decomp        Generate   Total elapsed
 0.0156002 secs 0.03119993 secs 0.04680014 secs
```

and still, the execution time is barely noticeable. This realization is given in the third row of Figure 4.

The improvement in efficiency for FFT over CHOL and EIGEN does not stop there. We continued to experiment with ever-finer grids for FFT, stopping at `M <- 2049` and `N <- 2049`, at which point the basic `expand.grid` commands used to define the centroid objects `cent` and `cent.ext.row1`, consisting of a staggering $2048^2 = 4,194,304$ and $4096^2 = 16,777,216$

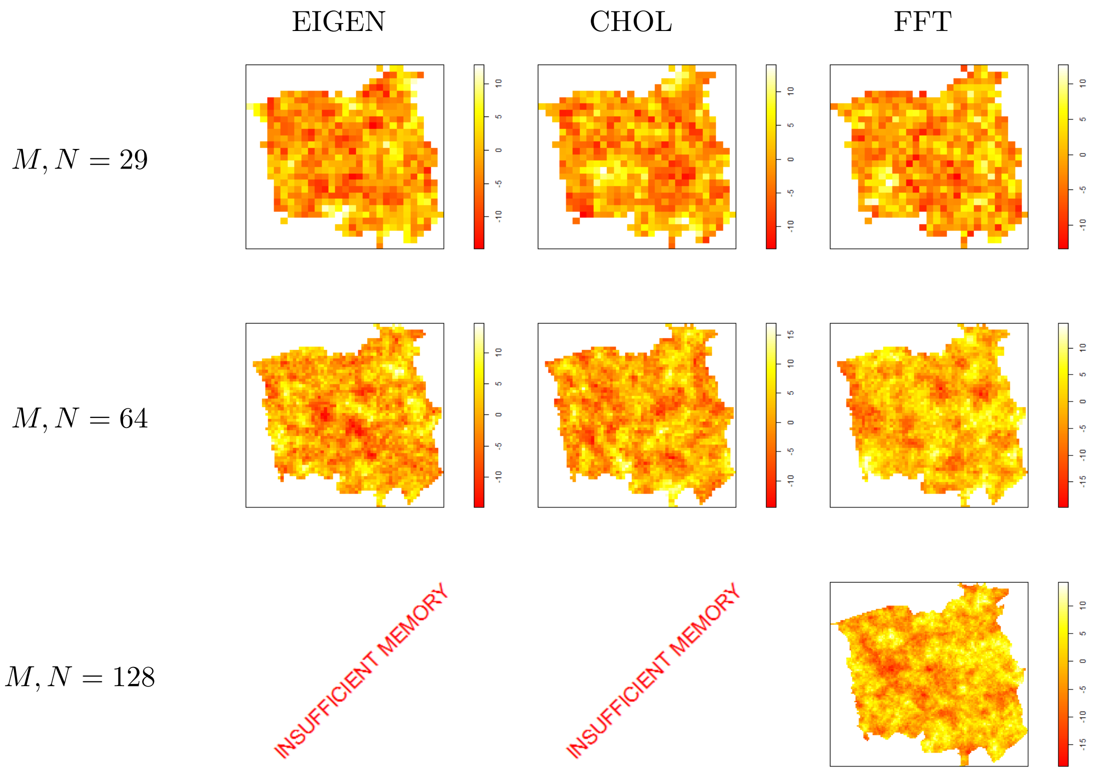

Figure 4: Field realizations within W via the eigenvalue and Cholesky decompositions, as well as the FFT approach, for varying resolutions.

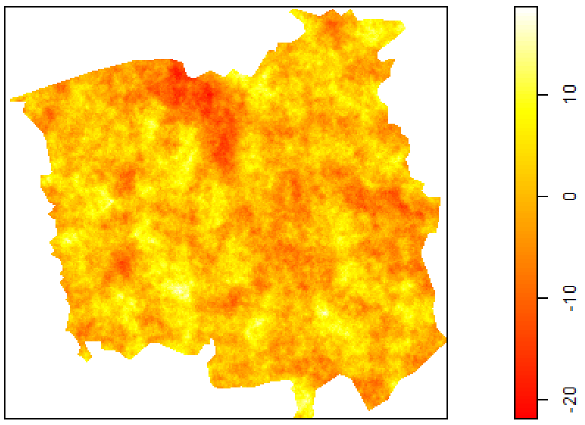

Figure 5: The 'super-fine' FFT realization within W based on a 2048×2048 rectangular centroid grid.

individual coordinates respectively, created their own modest computational delays. Even so, the total elapsed time for a realization of this 'super-fine' field using the FFT methods in the same way as above took only 55 seconds, and this is shown separately in Figure 5.

Generally speaking, we may be content in practice with resolutions far coarser than the super-fine example, provided of course we can adequately capture the scale of dependence. However, the fact that we already encounter difficulties using the EIGEN and CHOL decomposition methods for resolutions not much finer than a 64×64 lattice, as well as the computational expense already exhibited at those levels, makes the FFT approach very appealing. There is however a key issue in the use of FFT in this context: stationarity and isotropy must be imposed in order for the block-Toeplitz extended covariance matrix to be valid. Where this cannot be safely assumed, the Cholesky decomposition may be a more sensible option. Nevertheless, these constraints on a given Gaussian field are not overly restrictive in practice, with a wide variety of surfaces still capable of being generated by controlling the form of r and any parameters influencing it (ϕ), as well as the variance σ^2.

An additional requirement for successful implementation of the FFT mechanism, mentioned earlier, is that the extended torus-wrapped covariance matrix be positive semidefinite. There is no way to guarantee this in a given application, though Møller *et al.* (1998) and Møller and Waagepetersen (2004) remark that this has rarely been a problem for them in practice, provided a suitable level of discretization is employed, comments supported by our experiences. Wood and Chan (1994) and these later works recommend choosing larger lattice extensions as a possible remedy in the event negative eigenvalues are encountered, while Gneiting *et al.* (2006) explore the use of appropriately modified covariance functions.

4.3. A note on sparse covariance structures

The running time calculations made above all assumed a dense covariance matrix. There has been a considerable amount of work developing efficient algorithms for the case when the covariance matrix or *precision matrix* (inverse of the covariance matrix) is banded, or more generally, sparse. In both cases there can be significant improvements in time complexity, sometimes giving algorithms which are even faster than those based on FFT.

A matrix B has bandwidth b_w if all of its non-zero entries lie on the diagonal or on one of the first b_w lower or upper diagonals, that is, if $B_{ij} > 0$ implies $|i - j| \leq b_w$. Martin and Wilkinson (1965) showed that when B is positive definite there exists a Cholesky factorization $B = R^\top R$ such that R also has bandwidth at most b_w and, furthermore, R can be obtained in time *linear* in the number of rows or columns of B.

Banded matrices of this form occur frequently when simulating Gaussian *Markov* random fields. For these simulations, an entry i, j of the precision matrix Σ^{-1} is non-zero only if i and j are neighbors. Hence bandedness arises from small neighborhood sizes, a property which Rue (2001) uses to great effect when designing efficient samplers. He observes that careful permutation of the nodes can lead to substantial reductions in bandwidth, and presents heuristic algorithms for finding good permutations (finding the optimal permutation is an NP-hard problem; see Papadimitriou 1976).

Note that the covariance matrix of a Gaussian random field will not be bounded unless the correlation function becomes identically zero for more than close distances.

In other applications, the covariance or precision matrices may not have low bandwidth, but they can still be sparse, with few non-zero entries. An advantage of sparse matrices, with appropriate data structures (e.g., Furrer and Sain 2010), is that computations like multiplying a vector by a matrix are essentially linear in the number of non-zero entries of the matrix. This fact forms the foundation of sparse iterative algorithms (reviewed in Golub and Van

Loan 1989) which can be used to solve massive linear systems without ever constructing a copy of the matrix.

Sparse iterative samplers (Schneider and Willsky 2003; Parker and Fox 2012) follow a similar strategy except that they produce samples instead of solutions. The algorithms are iterative, in the sense that they produce a sequence of (correlated) samples. Each sample comes closer and closer to the target distribution, the rate of convergence in both cases being sub-quadratic. Under the unrealistic assumption of exact arithmetic, the algorithms can produce a sample with $O(MN)$ matrix-vector multiplications. Most importantly these methods, like the circulant embedding approach, do not require storage of the complete covariance or precision matrices as they make use of the matrix only via a matrix-vector multiplication.

5. Timing multiple realizations

The previous section showed that the main contributor to execution time when generating single realizations using EIGEN, CHOL, and FFT, are the operations performing the decomposition step. This cost need not be repeated should one require multiple realizations of the Gaussian field for a given W, covariance matrix, and grid resolution. Repetition of code is restricted to generation of a new standard normal vector of the appropriate dimension, and any associated matrix multiplication steps required to transform this vector into one reflecting the desired covariance structure. As such, it is of interest to briefly examine the nature of the increase in computational expense from generation of a single variate to generation of multiple variates.

To this end, we employ the code demonstrated in the previous two sections and evaluate the execution times for simulation of Gaussian fields on the unit square (zero mean, exponential correlation function with $\sigma^2 = 2$ and $\phi = 0.05$). Generation is performed for varying grid resolutions $M = N = 5, 10, \ldots, 65, 70$, and the number of realizations S is varied with $S = 1, 10, 100, 1000$. Figure 6 shows the recorded times in each of these situations for EIGEN, CHOL, and FFT. The leftmost plot gives the times for the unrepeated decomposition step. Time taken to generate the desired number of realizations is given in the second column, and moving down we see that the increase in generation time appears to be directly proportional to the increasing S. The total elapsed time i.e., the sum of the decomposition and generation steps, is provided in the right column.

The experimental results closely match the expectations from theory. From above, we would expect a per-sample time complexity of $O(M^2N^2) = O(N^4)$ for EIGEN and CHOL, and $O(MN\log(MN)) = O(N^2\log(N))$ complexity for FFT, ignoring in all cases the computation time for the original decompositions.

6. Concluding remarks

Introduced by Dietrich and Newsam (1993) and Wood and Chan (1994) in the context of Gaussian fields, it is clear that FFT is an invaluable tool when simulating these stochastic processes. The seminal work of Møller *et al.* (1998) discusses the use of this approach for the unconditional and conditional simulation of LGCP, where the logarithm of the planar intensity function is assumed to be driven by a latent Gaussian process. This was followed by theoretical instructions on circulant embedding of the covariance matrix associated with

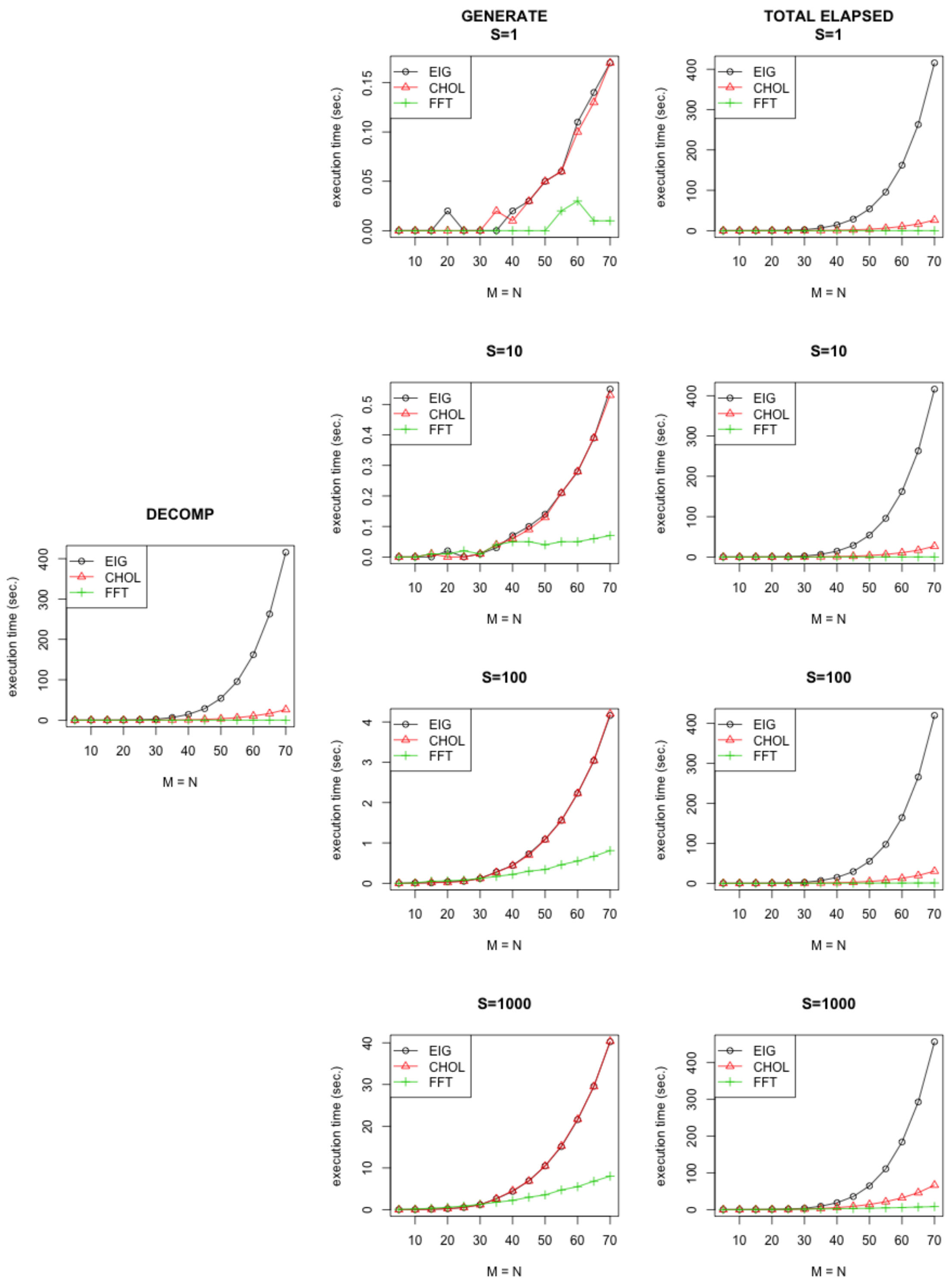

Figure 6: DECOMP, GENERATE and TOTAL ELAPSED timings for multiple field realizations using EIGEN, CHOL, and FFT.

a given LGCP, and subsequent implementation of the two-dimensional FFT, in Møller and Waagepetersen (2004).

It is important that interesting and useful technical achievements continue to be taken advantage of by applied researchers. The main goal of this work was therefore to augment the historical efforts with respect to the FFT and Gaussian processes in an applied sense, providing a clear translation from the technical notes into R code alongside intuitive visuals. The benefits afforded by working with the two-dimensional FFT when simulating stationary Gaussian fields (as opposed to alternative methods) were also made clear, as computation time and efficiency was also highlighted. It is in fact only slight modifications of the showcased code which drives the relevant operations in the recently released R package **lgcp** (Taylor, Davies, Rowlingson, and Diggle 2013); currently available on the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/`. Package **lgcp** provides the functionality to handle spatial and spatiotemporal modeling using the LGCP, where FFT support plays a large part in the practical feasibility of the methods. Also of interest is the impressive R package **RandomFields**, programmed primarily in C, which contains a suite of functions surrounding stochastic processes.

Acknowledgments

The authors are indebted to an anonymous referee and an editor for their constructive feedback which dramatically improved this work.

References

Adler D, Murdoch D (2012). ***rgl**: 3D Visualization Device System (OpenGL).* R package version 0.92.894, URL `http://CRAN.R-project.org/package=rgl`.

Anderson E, Bai Z, Bischof C, Blackford LS, Demmel J, Dongarra J, Croz JD, Greenhaum A, Hammerling S, McKenney A, Sorensen D (2000). ***LAPACK** Users Guide.* Society for Industrial and Applied Mathematics, Philadelphia, 3rd edition.

Baddeley AJ, Turner R (2005). "**spatstat**: An R Package for Analyzing Spatial Point Patterns." *Journal of Statistical Software*, **12**(6), 1–42. URL `http://www.jstatsoft.org/v12/i06/`.

Dietrich CR, Newsam GN (1993). "A Fast and Exact Method for Multidimensional Gaussian Stochastic Simulations." *Water Resources Research*, **29**(8), 2861–2869.

Furrer R, Sain SR (2010). "**spam**: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields." *Journal of Statistical Software*, **36**(10), 1–25. URL `http://www.jstatsoft.org/v36/i10/`.

Gneiting T, Ševčíková H, Percival DB, Schlather M, Jiang Y (2006). "Fast and Exact Simulation of Large Gaussian Lattice Systems in $\mathbb{R}^2$: Exploring the Limits." *Journal of Computational and Graphical Statistics*, **15**(3), 483–501.

Golub GH, Van Loan CF (1989). *Matrix Computations*. 2nd edition. Johns Hopkins University Press, London.

Martin RS, Wilkinson JH (1965). "Symmetric Decomposition of Positive Definite Band Matrices." *Numerische Mathematik*, **7**(5), 355–361.

Møller J, Syversveen AR, Waagepetersen RP (1998). "Log Gaussian Cox Processes." *Scandianvian Journal of Statistics*, **25**(3), 451–482.

Møller J, Waagepetersen RP (2004). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall, Boca Raton.

Papadimitriou CH (1976). "The NP-Completeness of the Bandwidth Minimization Problem." *Computing*, **16**(3), 263–270.

Parker A, Fox C (2012). "Sampling Gaussian Distributions in Krylov Spaces with Conjugate Gradients." *SIAM Journal on Scientific Computing*, **34**(3), B312–B334.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rue H (2001). "Fast Sampling of Gaussian Markov Random Fields." *Journal of the Royal Statistical Society B*, **63**(2), 325–338.

Rue H, Held L (2005). *Gaussian Markov Random Fields: Theory and Applications*. Chapman and Hall, Boca Raton.

Schlather M (2012). ***RandomFields**: Simulation and Analysis of Random Fields*. R package version 2.0.59, URL http://CRAN.R-project.org/package=RandomFields.

Schneider MK, Willsky AS (2003). "Krylov Subspace Method for Covariance Approximation and Random Processes and Fields." *Multidimensional Systems and Signal Processing*, **14**(4), 295–318.

Taylor BM, Davies TM, Rowlingson BS, Diggle PJ (2013). "**lgcp**: An R Package for Inference with Spatial and Spatio-Temporal Log-Gaussian Cox Processes." *Journal of Statistical Software*, **52**(4), 1–40. URL http://www.jstatsoft.org/v52/i04/.

Wand MP, Jones MC (1995). *Kernel Smoothing*. Chapman and Hall, Boca Raton.

Wood ATA, Chan G (1994). "Simulation of Stationary Gaussian Processes in $[0, 1]^d$." *Journal of Computational and Graphical Statistics*, **3**(4), 409–432.

Affiliation:

Tilman M. Davies
Department of Mathematics & Statistics
University of Otago
PO Box 56
Dunedin, New Zealand 9054
E-mail: tdavies@maths.otago.ac.nz
URL: http://www.maths.otago.ac.nz/home/department/staff/
_staffscript.php?s=tilman_davies

David J. Bryant
Allan Wilson Centre for Molecular Ecology and Evolution
Department of Mathematics & Statistics
University of Otago
PO Box 56
Dunedin, New Zealand 9054
E-mail: dbryant@maths.otago.ac.nz
URL: http://www.maths.otago.ac.nz/~dbryant/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 55, Issue 9 *Submitted:* 2012-01-16
November 2013 *Accepted:* 2013-04-05