Calculation of Average Covariance Using Fast Fourier Transform (FFT)

Yongshe Liu, Yuanlin Jiang* & Phaedon Kyriakidis†

Stanford Center for Reservoir Forecasting
Petroleum Engineering Department
Stanford University

April 11, 2006

Abstract

Block-related covariance calculation is an essential part in any block data kriging or simulation algorithm. Such calculation takes almost all of the CPU time if there are many blocks or the block size is large. The computation of block-to-block covariances via the Fast Fourier Transform (FFT) provides an efficient solution to this problem. In this report, the methodology of the FFT covariance calculation approach is presented. We propose an hybrid FFT and traditional integration approach. The C++ code for this algorithm is developed and some test results prove the code to be fast and accurate.

*SUPRI-B, Petroleum Engineering Department, Stanford University
†Department of Geography, University of California at Santa Barbara
1 Introduction

In the implementation of block data integration, the fast calculation of block average covariance values is a critical issue. Since we assume that any block datum is a linear spatial average of point values at locations discretizing that block, all block-related covariances call for averaging point covariance values (Journel & Huijbregts 1978, Hansen et al. 2004, Liu 2005). But many multiple numerical summations can be very slow, especially when blocks are of different sizes and shapes. The block covariance calculation may take up to 98% CPU time in the present bdsim code for stochastic simulation accounting for block data in presence of block data (Liu & Journel 2005). Building on the characteristics of a covariance matrix, a Fast Fourier Transform (FFT) approach has been proposed to compute the block-related covariance (Dietrich & Newsam 1997, Nowak et al. 2003, Kyriakidis et al. August 2005).

Based on the recent work of Kyriakidis (2005), this report suggests a hybrid approach, a combination of FFT and numerical integration, to compute block-related covariances. A corresponding C++ code has been developed. Our goal is to develop a stand alone geostatistical library for block-related covariance calculation.

2 Methodology

We first briefly recall the FFT covariance calculation methodology using a 1D example (Kyriakidis et al. August 2005). The established GSLIB notations are used (Deutsch & Journel 1998).

2.1 1D case

The basic idea of this method is that the covariance between any two blocks \(g\) and \(k\) can be computed as the product of the three Fourier transforms: (i) the block \(g\) location information, (ii) the point spatial covariance model, (iii) the block \(k\) location information.

More precisely, define a 1D field as a regular grid with \(N\) nodes. A block is defined as a set of \(n\) cells with id numbers: \(i = n_1, \ldots, n_2\). A row vector defining the block location
is written as

\[ b_k(i) = \begin{cases} 
1/n, & \text{if } i = n_1, \ldots, n_2 \\
0, & \text{otherwise}
\end{cases} \]

where \( k \) is the block index if there are multiple blocks, see Figure 1(a). The value \( 1/n \) is assigned to each block location to allow later averaging of point covariances over that block.

Given a point-support stationary covariance model \( C(h) \), where \( h \) is the separation distance, the covariance matrix \( C_p \) between any two locations \( x_i \) and \( x_j \) within a 1D field of dimension \( N \) can be written as: \( C_p = [C_{i,j}] \), where \( C_{i,j} = C(x_j - x_i) = \text{cov}\{Z(x_j), Z(x_i)\} \) with \( i = 1, \ldots, N \), \( j = 1, \ldots, N \) and \( Z(x) \) is the stationary random function. In matrix format, \( C_p \) is written as:

\[
C_p = \begin{bmatrix}
C(0) & C(1) & \cdots & C(N-2) & C(N-1) \\
C(1) & C(0) & \cdots & C(N-3) & C(N-2) \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
C(N-2) & C(N-3) & \cdots & C(0) & C(1) \\
C(N-1) & C(N-2) & \cdots & C(1) & C(0)
\end{bmatrix}
\]

Note that \( C_p \) is a symmetric Toeplitz matrix (Bini 1995).

From the block location row vector \( b_k \) and the point covariance matrix \( C_p \), the block-to-point covariance is written as:

\[
C_{bp} = C_p b_k^T
\]

where \( b_k^T \) is transpose of \( b_k \). Note that the product of row \( i \) of \( C_p \) and \( b_k^T \) is the average covariance value between block \( k \) and point \( i \):

\[
C_{ki} = \sum_{j=1}^{N} C_{ij} b_k(j)
\]

where \( b_k(j) \) is the \( j \)th element of block \( b_k \).

Consider another block \( g \) with block location vector \( b_g \). The average covariance between block \( g \) and block \( k \) is written as:

\[
C_{bgk} = b_g C_{bp} = b_g C_p b_k^T
\]
Expression (3) indicates that the covariance between blocks $g$ and $k$ equals the spatial average of the block-to-point covariances of block $k$ over all point locations within block $g$.

Figure 2 gives an calculation example to illustrate Equations (2) and (3). The field envelope is composed of 5 cells: $P_i, i = 1 \cdots 5$. Block $k$ is constituted by two cells (red $P_1$ and $P_2$ in Figure 2) and its location vector is written as $b_k = [1/2, 1/2, 0, 0, 0]$. Block $g$ is constituted by three cells (blue $P_2, P_3$ and $P_4$), and its location vector is written as $b_k = [0, 1/3, 1/3, 1/3, 0]$. The point-to-point covariance is given in Figure 2. From Equation (2), we obtain:

$$C_{bp} = C_{pb}^T = [C_{p1b_k}, C_{p2b_k}, C_{p3b_k}, C_{p4b_k}, C_{p5b_k}]$$

This is the block-to-point covariance value for block $k$. Then from Equation (3), we get the block-to-block covariance:

$$C_{bg} = b_g C_{bp} = \frac{1}{3} (C_{p2b_k} + C_{p3b_k} + C_{p4b_k})$$

which is the average of the block-to-point covariance values over the 3 locations of block $g$.

In order to build a circulant matrix then capitalize on its spectral decomposition properties, the $(N \times N)$ matrix $C_p$ is extended to a twice larger matrix $\tilde{C}_p$ of dimension $(2N \times 2N)$:

$$\tilde{C}_p = \begin{bmatrix}
C(0) & C(1) & \cdots & C(N-1) & C(N) & C(N-1) & \cdots & C(1) \\
C(1) & C(0) & \cdots & C(N-2) & C(N-1) & C(N) & \cdots & C(2) \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
C(N-1) & C(N-2) & \cdots & C(0) & C(1) & C(2) & \cdots & C(N) \\
C(N) & C(N-1) & \cdots & C(1) & C(0) & C(1) & \cdots & C(N-1) \\
C(N-1) & C(N) & \cdots & C(2) & C(1) & C(2) & \cdots & C(N-2) \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
C(1) & C(2) & \cdots & C(N) & C(N-1) & C(N-2) & \cdots & C(0)
\end{bmatrix}$$

(4)
where \( \tilde{C}_p \) is fully determined by its first row or first column; all other rows or columns can be obtained by shifting the first row or column, respectively. The spectral decomposition of this circulant matrix \( \tilde{C}_p \) can now be written as (Strang 1998):

\[
\tilde{C}_p = \tilde{F} \tilde{E} \tilde{F}^H
\]  

(5)

where \( \tilde{E} = \text{diag}\{\tilde{e}(j), j = 1, \cdots, 2N\} \) denotes the \((2N \times 2N)\) diagonal matrix of eigenvalues of \( \tilde{C}_p \), \( \tilde{F} \) is the \((2N \times 2N)\) Fourier matrix, and \( \tilde{F}^H \) is the Hermitian transpose of \( \tilde{F} \). When \( \tilde{F}^H \) is multiplied with a vector, say \( v \), it yields the forward Discrete Fourier Transform (DFT) of this vector, denoted as \( \mathcal{F}(v) \). When \( \tilde{F} \) is multiplied with a vector \( v \), it yields the inverse DFT of this vector, denoted as \( \mathcal{F}^{-1}(v) \). In practice, DFT can be efficiently computed via FFT.

The two block location vectors \( \mathbf{b}_g \) and \( \mathbf{b}_k \) are accordingly extended to \( \tilde{\mathbf{b}}_g \) and \( \tilde{\mathbf{b}}_k \) of size \( 2N \times 1 \) by padding zero values. For example,

\[
\tilde{\mathbf{b}}_k(i) = \begin{cases} 
1/n, & \text{if } i = n_1, \cdots, n_2 \\
0, & \text{otherwise}
\end{cases}
\]

The block-to-block covariance \( C_{\mathbf{b}_g \mathbf{b}_k} \) Expression (3) then changes to:

\[ C_{\mathbf{b}_g \mathbf{b}_k} = \tilde{\mathbf{b}}_g^T \tilde{C}_p \tilde{\mathbf{b}}_k \]  

(6)

where \( \tilde{\mathbf{b}}_g^T \) is the transpose of \( \tilde{\mathbf{b}}_g \).

Substituting Expression (5) into Equation (6) yields:

\[
C_{\mathbf{b}_g \mathbf{b}_k} = \tilde{\mathbf{b}}_g^T (\tilde{F} \tilde{E} \tilde{F}^H) \tilde{\mathbf{b}}_k = (\tilde{\mathbf{b}}_g^T \tilde{F} \tilde{E} \tilde{F}^H \tilde{\mathbf{b}}_k)
\]

(7)

\[
= (\tilde{F}^H \tilde{\mathbf{b}}_g)^H \tilde{E} (\tilde{F}^H \tilde{\mathbf{b}}_k), \text{ then using the property of } \tilde{F} \text{ mentioned above}
\]

\[
= \mathcal{F}(\tilde{\mathbf{b}}_g)^H \tilde{E} \mathcal{F}(\tilde{\mathbf{b}}_k) = \tilde{\mathbf{f}}_g^H \tilde{\mathbf{f}}_k = \sum_{j=1}^{2N} \tilde{f}_g(j) \tilde{e}(j) \tilde{f}_k(j)
\]

where \( \mathcal{F} \) denotes forward FFT, \( \tilde{\mathbf{f}}_g = \mathcal{F}(\tilde{\mathbf{b}}_g) \), \( \tilde{\mathbf{f}}_g^H \) is the Hermitian transpose of \( \tilde{\mathbf{f}}_g \), \( \tilde{\mathbf{f}}_k = \mathcal{F}(\tilde{\mathbf{b}}_k) \), \( \tilde{\mathbf{f}}_g(j) \) is the \( j \)-th entry of vector \( \tilde{\mathbf{f}}_g \), \( \tilde{\mathbf{f}}_k(j) \) is the \( j \)-th entry of vector \( \tilde{\mathbf{f}}_k \), \( \tilde{f}_g(j) \) is the complex conjugate of \( \tilde{f}_g(j) \).
and $\tilde{e}(j)$ is the $j$ entry of $\tilde{e}$, the eigenvalue vector of $\tilde{C}_p$.

$\tilde{e}$ is computed by the FFT of the first row of $\tilde{C}_p$, which will be derived hereafter.

Since $\tilde{F}$ is a Hermitian matrix, $\tilde{F}^H \tilde{F} = \tilde{F} \tilde{F}^H = I$. Thus, Expression (5) is written as:

$$\tilde{C}_p \tilde{F} = \tilde{F} \tilde{E}$$

Since $\tilde{C}_p$ is fully determined by its first row $\tilde{C}_p(1, \cdot)$, the expression above reduces to:

$$\tilde{C}_p(1, \cdot) \tilde{F} = \tilde{F}(1, \cdot) \tilde{E}$$

where $\tilde{F}(1, \cdot)$ is the first row of the Fourier matrix, that is a vector of $(1 \times 2N)$ with constant value $\frac{1}{\sqrt{2N}}$ (Nowak et al. 2003).

The right hand side of the above equation then becomes:

$$\frac{1}{\sqrt{2N}} \tilde{e}, \text{ where } \tilde{e} = [\tilde{e}(j), j = 1, \cdots, 2N]$$

Taking the Hermitian transpose of both sides of the above expression, we get:

$$\tilde{e} = \sqrt{2N} \tilde{F}^H \tilde{C}_p(\cdot, 1) \Rightarrow \tilde{e} = \mathcal{F}(\tilde{C}_p(1, \cdot)) \text{ since } \tilde{C}_p \text{ is symmetric} \quad (8)$$

From Equation (6) and (8), it appears that the 1D block-to-block covariance $C_{b_g b_k}$ can be computed through the product of three vectors: the two forward FFTs of the extended block location vectors $\mathcal{F}(\tilde{b}_g)$ and $\mathcal{F}(\tilde{b}_k)$ and the forward FFT of the first row of the extended point-to-point covariance matrix $\mathcal{F}(\tilde{C}_p(1, \cdot))$.

### 2.2 2D and 3D cases

The major differences between 1D and 2D applications are (1) how to build the extended circulant covariance matrix $\tilde{C}_p$, and (2) which part of the matrix needs to be stored. For a 1D block with $N$ cells, its $(2N \times 2N)$ covariance matrix $\tilde{C}_p$ is given in Expression (4).

A 1D block configuration and the first row of $\tilde{C}_p$ are shown in Figure 1(a). This first row is the only covariance matrix information that need to be stored in the 1D case.

A $N \times M$ 2D field can be considered to be composed of $M$ blocks with $N$ cells within each block, see the $M$ 1D blocks with different colors in Figure 1(b). The 1D extended circulant covariance matrix consists of $2M \times 2M$ sub-matrices blocks, see Figure 1(c).
Thus, the 2D case can be seen as two nested sets of the 1D case, each with a different scale. For the finer scale, the field is of dimension $N$, see the color bars in Figure 1(b). Each of the block submatrix in Figure 1(c) has a structure similar to that of the 1D extended matrix (4). For the larger scale, each of the color bar in Figure 1(b) is considered a cell, hence it can be seen as a 1D case. The whole block submatrices again have a structure similar to the 1D extended matrix (4), that is each of the block submatrix can be fully determined by its first row of elements. In summary, the first row of the block submatrix is fully determined from the first row of the whole matrix (the top color row in Figure 1(c)). Similarly, the first row of the block submatrices determines the whole block submatrices. Therefore, the first row of elements of the whole matrix fully determines the 2D field covariance matrix.

Take a closer look at the first row of elements of the extended covariance matrix $\tilde{C}_p$. In $\tilde{C}_p$, the $k$th row in block entry $(i, j)$ represents the covariance between the $k$th cell in block $i$ and the whole block $j$. Thus, each of color bar in the top row in Figure 1(c) is the extended covariance matrix between the 1st cell in block 1 (the cell (1,1) in Figure 1(b)) and any of the $M$ blocks in Figure 1(b). We rearrange these first row color bars in the covariance matrix and stack them vertically with the left red one on the top and the right red one on the bottom (Figure 3): this results in a shifted point covariance map of dimension $2N \times 2M$, see Figure 4(b). If this map is shifted between quadrants 1 and 3, and between quadrants 2 and 4 (Figure 5(a)), the traditional covariance map of dimension $2N \times 2M$ is obtained, see Figure 4(a). This means that for the 2D case, the $\tilde{e}$ term in Expression (8) can be obtained by FFT of the shifted covariance map of Figure 4(b). If the block location data vectors are extended to the same size as the shifted covariance map, Equation (6) can be used to compute the 2D block-to-block covariance.

A 3D case calls for a covariance cube. In 3D there are 4 shifts: quadrant 1 against 8, quadrant 2 against 7, quadrant 3 against 6 and quadrant 4 against 5, see Figure 5(b).

2.3 Hybrid FFT and integration method

In the traditional method, the block-to-block covariance $C_{bb}$ is computed by averaging the point covariances over the block volumes (Journel & Huijbregts 1978, Liu 2005). In
the full FFT method, $C_{bb}$ is computed by the product of three terms: FFT of the first block location vector, FFT of the point covariance and FFT of the second block location vector. We propose a method combining these two methods. Forward FFT is performed on one block location vector, then on the covariance map. Inverse FFT is performed on the product of these two FFTs. This inverse FFT yields the block-to-point covariance $C_{bp}$ map. The corresponding analytical derivation is as follows.

From Equation (8), $C_{bp}$ is written as:

$$C_{bp} = (\tilde{F}\tilde{E}\tilde{F}^H)b_k = \tilde{F}\tilde{E}(\tilde{F}^Hb_k) = \tilde{F}(\tilde{E}F(b_k)) = F^{-1}\tilde{E}F(b_k)$$

This algorithm has two advantages in terms of implementation. First, once the $C_{bp}$ map of one block is stored (provided enough memory is available for its storage), the $C_{bb}$ covariance can be easily computed by integration over the locations of the other block. This is written:

$$C_{b_bk} = \int_{v_k} C_{b_bp}(u)du$$

where $v_k$ denotes the locations of block $b_k$, $C_{b_bp}$ is the block $b$ to point covariance, $u$ is a running location variable.

Second, any block-to-point covariance value required by a kriging matrix can be quickly retrieved from the stored $C_{bp}$ map. Compared to the traditional integration method, this hybrid method would significantly reduce the CPU cost when there are many blocks. This is because all the covariances between any point in one block and any point in the other block must be computed for each pair of blocks in the integration method, while in the hybrid method it is just a simple average of block-to-point covariances over all locations within the second block. The block-to-point covariance map is computed only once and stored.

Compared to the full FFT method, this hybrid method has a valuable byproduct: the $C_{bp}$ map. Indeed the full FFT method only gives one value of the block-to-block
covariance and does not give the block-to-point covariance. With the full FFT method, if the $C_{bp}$ map is necessary (for example, to build a kriging matrix), it would have to be calculated separately. This is a consequence of the double for-loop being decomposed as a forward and inverse FFT computation (used to compute $C_{bp}$) and a single for-loop (used to compute $C_{bb}$). This approach is faster but requires much more storage.

As an example, Figure 6(a) shows two blocks: one being a curvilinear ray and the second a ball, both defined on a $(N \times N)$ field envelope grid. One approach to compute $C_{bb}$ is to first extend the ray field envelope grid to $2N \times 2N$ (Figure 6(b)) and compute the ray-to-point covariance $C_{bp}$ map on that extended field envelope (Figure 6(c)). Then the ray-to-ball covariance is computed by averaging the $C_{bp}$ values over all ball locations. Another approach to compute $C_{bb}$ is to first extend the ball field envelope grid to $2N \times 2N$ (Figure 6(d)) and compute the ball-to-point covariance $C_{bp}$ map on the extended field envelope (Figure 6(e)). Then the ray-to-ball covariance is computed by averaging the $C_{bp}$ values over all ray locations. These two methods result into similar covariance values (within reasonable error range), similar to the results obtained from a Matlab code implementing both the traditional integration method and the full FFT method, see Table 1.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Computed $C_{bb}$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid FFT &amp; Integration using ray-point cov.</td>
<td>0.57955036</td>
</tr>
<tr>
<td>Hybrid FFT &amp; Integration using ball-point cov.</td>
<td>0.57955036</td>
</tr>
<tr>
<td>FFT (Matlab code)</td>
<td>0.57955078</td>
</tr>
<tr>
<td>Traditional integration (Matlab code)</td>
<td>0.57955035</td>
</tr>
</tbody>
</table>

Table 1: Ray-to-ball covariance obtained with different methods

2.4 Global v.s. local FFT

The discussion above calls for performing the FFT globally over the whole field envelope. Actually, in practice, there is no need to evaluate the spatial correlation or the covariance map ($C_p$, $C_{bp}$ or $C_{bb}$) beyond the correlation range. This implies that the FFT may
be performed locally as long as the field envelope is large enough to cover each whole block, that is a smaller rectangle or cube containing the block can be used as a local field envelope. Note that such local FFT technique could not be applied to the full FFT block covariance calculation method because this full FFT calls for different blocks sharing the same field envelope grid; therefore the three FFT vectors can not be multiplied together. The hybrid (FFT and integration) method can consider a local FFT because the FFT vector of the first block location vector defined on a local field envelope can be multiplied with the FFT vector of the covariance map defined on the same local field envelope. This local covariance map is extracted from the larger covariance map of the original field envelope grid. The size of local field envelope should be equal to or larger than the maximum correlation range involved, which means the blocks located outside this local envelope field have no correlation with the original block. After this local FFT, the averaging step is performed over the second block location. If the second block is outside of the local field envelope, then \( C_{bb} = 0 \). If it is partially located within the local envelope, only the inside part is counted. This local FFT approach has not been implemented yet.

The differences between these 3 methods, traditional integration, full FFT and hybrid methods, are summarized in Figure 7.

3 Implementation

The C++ code was written to be general by avoiding calling into the Geostatistics Template Library (GSTL)(Remy 2001) or SGEMS (Remy 2004).

3.1 Flowchart

Figure 8 gives the flowchart of the implementation of the hybrid block covariance calculation approach. There are two major tasks. One is to calculate the point-to-point covariance map and the other is to deal with the block location vectors. For the first task, a covariance model is defined and is used to compute the covariance map on the extended field envelope grid. Then this extended map is shifted as mentioned before (Figure 4(b)),
and FFT is performed on the shifted covariance map. Note that this covariance map FFT is done only once for a given field envelope. This FFT result is then stored for all future usages. For the second task, the block field envelope grid is extended to the same size as the covariance map by padding zero values. Then FFT is performed on it. This block FFT must be repeated, however, for all blocks even if these blocks are of same size. The two results from block FFT and FFT of the covariance map are multiplied. After inverse FFT, the block-to-point covariance map over the field envelope grid is obtained. Note that only the first quadrant (left-upper quadrant in Figure 6(c) and 6(e)), which coincides with the original field envelope, will be used. From this block-to-point covariance map, the covariance between the block and any point location within the field envelope can be retrieved; similarly the covariance between this block and any other block located in the same field envelope can be computed by simply averaging the block-to-point covariance values over the locations discretizing the second block.

3.2 Performance of different FFT code packages

In all previous algorithms, the FFT is the most important tool. In the full FFT approach, there are three FFT operations (FFT of the 2 block location vectors and FFT of the covariance map). In the hybrid (FFT and integration) method, there are two FFT (FFT of the block location vector and FFT of the covariance map) plus one inverse FFT operations (inverse FFT of the product of the previous FFT results). These FFT or inverse FFT operations taken together account for more than 80% of the total CPU time. Thus, choosing a highly efficient FFT program is critical. Three different FFT C source codes were tested in this study. They are, the FFT code in the C Numerical Recipes book (Press et al. 1996), the package FFTN (Singleton 1968, Olesen 1997) and the package FFTW (Frigo & Johnson 2005). The Numerical Recipes code is simple and easy to understand, but it requires the number of input data to be a power of 2. The FFTN code can handle any size of input data and multiple dimension cases, but it is not very efficient CPU-wise. The FFTW code is an advanced FFT package developed at the MIT Laboratory for Computer Sciences, www.fftw.org. The authors claim that it is one of the fastest. In our test, it did give the best performance. An important feature
of **FFTW** is that it first tries several algorithms to find the fastest one for the specific input data, save that information and apply it to any similar input encountered later on. Therefore, the first FFT operation is a bit slower but the following FFT operations are very fast. **FFTW** has been retained in our C++ code. The performances of different FFT codes are shown in the next section.

## 4 Case study results

In this section, we perform tests using the different techniques for block covariance computation. All the tests are done on a mid-performance laptop with 1.8GHz CPU and 512M RAM.

### 4.1 Sensitivity to the change of field grid size

In this study, two blocks of equal size located at the upper-left and lower-right corners of the field are considered. There are $10 \times 10$ cells within each block, see Figure 9(a). The size of the field envelope changes from $10 \times 10$ to $800 \times 800$. We consider an isotropic spherical covariance with isotropic range of 2000, zero nugget and unit sill. The dimension of each elementary grid cell is constant equal to 1. The change in CPU time with increasing size of the field using the traditional integration method (Matlab code), the full FFT method (Matlab code), and the hybrid methods are given in Figure 9(b). The hybrid method includes three C++ codes implemented with the three types of FFT approaches mentioned before. Several points can be made from this figure. First, the CPU cost of the traditional integration method (the bottom blue curve in Figure 9(b)) is smaller than those of the two FTT methods because the numbers of cells in the two blocks are small. Second, The CPU time cost of the integration method is constant because the numbers of cells within the two blocks are constant, and it is not affected by the increase in size of the field envelope. The CPU costs of the those FFT methods increase with the increase in size of the field envelope, because this increases the size of the input data vector into the FFT or inverse FFT operations. Third, the CPU cost of the **FFTW** C++ code (red curve) is lowest among all FFT methods; it is more than 5 times faster than Kyriakidis
CASE STUDY RESULTS

(2005) original Matlab FFT code (pink curve). There are several steps in the numerical recipes FFT C++ code (black curve) which require the size of input data to be a power of 2. If that is not the case, the code extends the size to a power of 2. Assuming that the block covariances computed from the integration method are correct, the errors are the differences between these correct results and results from the FFT methods. Figure 9(c) gives the errors for the Matlab FFT code and the C++ code (all three different C++ codes using FFTW, numerical recipes FFT and FFTN produce about the same results). Note that even though the Matlab FFT code produces negligible error (red curve), the C++ codes get more accurate results (blue curve). Thus, the C++ code using FFTW is both much faster and more accurate than the Matlab FFT code.

4.2 Sensitivity to block sizes

In this section, the field envelope grid of dimension 500×500 is fixed; and only two blocks are considered. One is located at the upper-left corner of the field and the other is at the lower-right corner of the field (Figure 10(a)). The dimension of each elementary cell is constant at 1. The common size of the two blocks is varied. We consider an isotropic spherical covariance model with range 1000, zero nugget and unit sill. The changes in CPU time with increase of the common block size using the traditional integration method (Matlab code), the full FFT method (Matlab code), and the hybrid method (C++ code with FFTW) are given in Figure 10(b). It appears that the CPU cost of the traditional integration method (blue curve in Figure 10(b)) increases very fast as the number of cells in the two blocks increases. Conversely, the CPU costs of the two FFT methods are constant because the field envelope size remains constant. Again, the CPU time of the FFTW C++ code (green curve) is more than 5 times lower than Kyriakidis (2005) original Matlab FFT code (pink curve). Figure 10(c) gives the errors from the Matlab FFT code and the FFTW C++ code. The C++ code provides the most accurate results.
4 CASE STUDY RESULTS

<table>
<thead>
<tr>
<th>Block covariance</th>
<th>Find grid case</th>
<th>Coarse grid case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tradi.(Matlab)</td>
<td>0.174381745</td>
<td>0.174380700</td>
</tr>
<tr>
<td>FFT(Matlab)</td>
<td>0.174381625</td>
<td>0.174380810</td>
</tr>
<tr>
<td>Hybrid(C++)</td>
<td>0.174381745</td>
<td>0.174380700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU time (sec.)</th>
<th>Tradi.(Matlab)</th>
<th>FFT(Matlab)</th>
<th>Hybrid(C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>139.53</td>
<td>21.79</td>
<td>3.27</td>
</tr>
<tr>
<td></td>
<td>0.11</td>
<td>0.36</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 2: Block covariance calculation using different block discretization grids

4.3 Impact of block discretization size on block covariance

If a block is discretized into a large number of cells, it will make the computation expensive. It turns out that if coarser discretization cells are used, this not only reduces the computation time but also gets reasonably accurate block covariance values (Journel & Huijbregts 1978). This is illustrated by an example. In Figure 11, two blocks with size of 100×100 are located at the two corners of the field. The blocks in Figure 11(a) are defined over a finely gridded field envelope 1000×1000 with each of the block discretized by 100×100 cells, while the blocks in Figure 11(b) are defined over a coarsely gridded field envelope 100×100 with each of the block discretized by 10×10 cells. The same covariance model as that in Section 4.1 is used. Three different codes (traditional integration Matlab code, full FFT Matlab code and hybrid C++ code with FFTW) are used to compute the average covariance between these two blocks. The results are shown in Table 2. It appears that the calculated covariance values from the coarser grid are sufficiently close to those from the fine grid with a significantly less CPU cost. As to speed, the traditional method has the most significant CPU cost reduction and again the C++ code is the fastest. This indicates that regridding large blocks with coarser grids should be considered to save CPU time.
5 Conclusions

In summary, the following remarks can be drawn from this report:

- FFT is a very efficient tool for computing block covariance values. It is much faster than the traditional integration method when blocks are composed of a large number of cells or there are a large number of blocks in the study field.

- The CPU cost of traditional integration method is sensitive to the block discretization size, while that of FFT method is sensitive to the size of the field envelope.

- Similar to the traditional integration method, the FFT block covariance method can be applied to blocks with arbitrary shape.

- Compared to the full FFT method, the proposed hybrid method is more efficient. It can generate the block-to-point covariance map as a byproduct. Also it can be applied to local field envelopes. Both block-to-point covariance map and local FFT are important for geostatistical operations.

- The newly developed C++ have several advantages over Kyriakidis’s original Matlab code. It is much faster and produces more accurate results.

6 Acknowledgments

The authors would like to thank Professor Andre G. Journel for his suggestions, advices and correction of this manuscript.
References


Liu, Y. (2005), Crosswell tomographic inversion with block kriging, in ‘the 18th SCRF annual meeting report’.


Olesen, M. (1997), ‘Multivariate complex Fourier transform, computed in place using mixed-radix Fast Fourier Transform algorithm (c version)’, Queen’s University at Kingston.


Figure 1: 1D and 2D fields and the corresponding extended covariance matrices for FFT
Envelope field (N cells):

\[
P_1 \quad P_2 \quad P_3 \quad P_4 \quad P_5
\]

Block \( k \): \( b_k = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \end{bmatrix} \)

Block \( g \): \( b_g = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 \end{bmatrix} \)

Point-to-point covariance matrix:

\[
\mathbf{C}_p = \begin{bmatrix}
C(0) & C(1) & C(2) & C(3) & C(4) \\
C(1) & C(0) & C(1) & C(2) & C(3) \\
C(2) & C(1) & C(0) & C(1) & C(2) \\
C(3) & C(2) & C(1) & C(0) & C(1) \\
C(4) & C(3) & C(2) & C(1) & C(0)
\end{bmatrix}
\]

Block-to-point covariance map:

\[
\mathbf{C}_{b_p} = \mathbf{C}_p b_k^\top = \begin{bmatrix}
0.5(C(0)+C(1)) & 0.5(C(1)+C(0)) & 0.5(C(2)+C(1)) & 0.5(C(3)+C(2)) & 0.5(C(4)+C(3))
\end{bmatrix}
\]

Block-to-block covariance map:

\[
\mathbf{C}_{b_p} = b_g \mathbf{C}_{b_p} = b_g \mathbf{C}_p b_k^\top = \frac{1}{3} \left( \mathbf{C}_{p,b_p} + \mathbf{C}_{p,b_h} + \mathbf{C}_{p,b_0} \right)
\]

Figure 2: An example of block covariance calculation

Figure 3: The covariance map obtained by stacking the first color row in Figure 1(c)
Figure 4: 2D covariance map and its shifted result
(a) 2D case

(b) 3D case

Figure 5: Quadrant shift for 2D and 3D cases
(a) Configuration of two data blocks

(b) Extended grid of the ray block  

(c) Extended block-to-point covariance map of the ray block

(d) Extended grid of the ball block

(e) Extended block-to-point covariance map of the ball block

Figure 6: Computation of $C_{bb}$ by hybrid (FFT and integration) method
<table>
<thead>
<tr>
<th>Operations (2-block case)</th>
<th>Operations (n-block case)</th>
<th>Applicable to local FFT method</th>
<th>Impact of field envelope size</th>
<th>$C_{bp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integration method</strong></td>
<td>$C_p$ calculated $M \times M$ times</td>
<td>$C_p$ calculated $\frac{n(n-1)}{n} \times (M \times M)$ times</td>
<td>Not relevant</td>
<td>$C_p$ calculated $n$ times for each $C_{bp}$</td>
</tr>
<tr>
<td><strong>Full FFT method</strong></td>
<td>• 3 FFT</td>
<td>• $n+1$ FFT</td>
<td>No</td>
<td>Field size affects FFT calculation time</td>
</tr>
<tr>
<td></td>
<td>• 2 multiplications</td>
<td>• $n-1$ multiplications</td>
<td>All blocks must share the same field envelope</td>
<td>Can not calculate $C_{bp}$</td>
</tr>
<tr>
<td></td>
<td>• Not affected by block size $n$</td>
<td>• Not affected by block size $n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hybrid FFT and integration method</strong></td>
<td>• 3 FFT (2 forward and 1 inverse)</td>
<td>• $2n-1$ FFT</td>
<td>Yes</td>
<td>Envelope size affects FFT calculation time</td>
</tr>
<tr>
<td></td>
<td>• 1 multiplication</td>
<td>• $n(n-1)$ multiplications</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Block size affects the integration, but negligible.</td>
<td>• Block size affects the integration, but negligible.</td>
<td></td>
<td>$C_{bp}$ map as a byproduct</td>
</tr>
</tbody>
</table>

Figure 7: Comparing three block covariance calculation methods (all blocks have the same number $M$ of cells)
Figure 8: Flowchart of block covariance computation using the proposed hybrid FFT and integration method.
Figure 9: Sensitivity to field envelope size
Figure 10: Sensitivity to number of cells within blocks
Figure 11: Two block configurations with different field envelope grids