Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Highlights

**A tutorial and open source software for the efficient evaluation of gravity and magnetic kernels**

*Computers and Geosciences xxx (xxxx) xxx*

Jarom D. Hogue, Rosemary Anne Renaut*, Saeed Vatankhah

- Fast matrix–vector multiplications for gravity and magnetic kernels.
- Matrices with Block Toeplitz–Toeplitz Block structure.
- Padded domains and variable depths.
- Open source software in MATLAB.
- Memory efficient and computationally powerful.

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

# A tutorial and open source software for the efficient evaluation of gravity and magnetic kernels

Jarom D. Hogue [a], Rosemary Anne Renaut [a,*], Saeed Vatankhah [b]

[a] School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ, USA
[b] Institute of Geophysics, University of Tehran, Tehran, Iran

ARTICLE INFO

*Keywords:*
Forward modeling
Fast Fourier Transform
Gravity
Magnetic

ABSTRACT

Fast computation of three-dimensional `gravity` and `magnetic` forward models is considered. When the measurement data is assumed to be obtained on a uniform grid which is staggered with respect to the discretization of the parameter volume, the resulting kernel sensitivity matrices exhibit block-Toeplitz–Toeplitz-block (BTTB) structure. These matrices are symmetric for the `gravity` problem but unsymmetric for the `magnetic` problem. In each case, the structure facilitates fast forward computation using two-dimensional fast Fourier transforms. The construction of the kernel matrices and the application of the transform for fast forward multiplication, for each problem, is carefully described. But, for purposes of comparison with the non-transform approach, the generation of the unique entries that define a given kernel matrix is also explained. It is also demonstrated how the matrices, and hence transforms, are adjusted when padding around the volume domain is introduced. The transform algorithms for fast forward matrix multiplication with the sensitivity matrix and its transpose, without the direct construction of the relevant matrices, are presented. Numerical experiments demonstrate the significant reduction in computation time and memory requirements that are achieved using the transform implementation. Thus, it becomes feasible, both in terms of reduced memory requirements and computational time, to implement the transform algorithms for large three-dimensional volumes. All presented algorithms, including with variable padding, are coded for optimal memory, storage and computation as an open source MATLAB code which can be adapted for any convolution kernel which generates a BTTB matrix, whether or not it is symmetric. This work, therefore, provides a general tool for the efficient simulation of `gravity` and `magnetic` field data, as well as any formulation which admits a sensitivity matrix with the required structure.

## 1. Introduction

Fast computation of geophysics kernel models has been considered by a number of authors, including calculation within the Fourier domain as in Li et al. (2018), Pilkington (1997), Shin et al. (2006) and Zhao et al. (2018), and through discretization of the operator and calculation in the spatial domain as in Chen and Liu (2018) and Zhang and Wong (2015). Pilkington (1997) introduced the use of the Fast Fourier Transform (FFT) for combining the evaluation of the `magnetic` kernel in the Fourier domain with the conjugate gradient method for solving the inverse problem to determine `magnetic` susceptibility from measured `magnetic` field data. Li et al. (2018) considered the use of the Gauss FFT for fast forward modeling of the `magnetic` kernel on an undulated surface, combined with spline interpolation of the surface data. Their work focused on the implementation of the model in the wave number domain and only applied the method for forward modeling. The Gauss FFT was also used by Zhao et al. (2018) for the development of a high accuracy forward modeling approach for the `gravity` kernel. Moreover, in earlier work, Shin et al. (2006) designed a Fortran code for fast forward and inverse modeling of the gravity model using the Fourier domain method using the FFT for achieving fast computation. On the other hand, Gómez-Ortiz and Agarwal (2005) provided a MATLAB code for computing the geometry of a density interface related to a known gravity anomaly by also employing the FFT to achieve fast computation, but which is not related to forward modeling for gravity models.

Bruun and Nielsen (2007), and subsequently, Zhang and Wong (2015) introduced the use of the Block-Toeplitz–Toeplitz-Block (BTTB) structure of the modeling sensitivity matrix for fast three-dimensional inversion of three-dimensional `gravity` and `magnetic` data. For a matrix with BTTB structure, it is possible to embed the information

* Corresponding author.
*E-mail addresses:* jdhogue@asu.edu (J.D. Hogue), renaut@asu.edu (R.A. Renaut), svatan@ut.ac.ir (S. Vatankhah).

https://doi.org/10.1016/j.cageo.2020.104575
Received 13 December 2019; Received in revised form 15 May 2020; Accepted 5 August 2020

within a matrix of Block-Circulant Circulant-Block (BCCB) structure that facilitates fast forward multiplication using a two-dimensional FFT (2DFFT), see e.g. Chan and Jin (2007) and Vogel (2002). For three-dimensional modeling, Zhang and Wong (2015) exploited the two-dimensional multi-layer structure of the kernel, that provides BTTB structure for each layer of the domain, and performed the inverse operation iteratively over all layers of the domain. The technique is flexible to depth layers of variable heights, and permits the inclusion of smoothness stabilizers in the inversion, for each layer of the domain. Zhang and Wong (2015) adopted the preconditioning of the BTTB matrix using optimal preconditioning operators as presented in Chan and Jin (2007) for implementing efficient and effective solvers for the inversion. On the other hand, Geng et al. (2019) used the BTTB structure for the development of a Bayesian inversion of gravity data, using an interesting inclusion of available a priori data, again referencing the Toeplitz literature, Chan and Jin (2007), but without providing extensive implementation details.

A fast forward modeling of the `gravity` field was developed by Chen and Liu (2018), using the three-dimensional modeling of the `gravity` kernel as given in e.g. Boulanger and Chouteau (2001) and Haáz (1953). Their work extends the techniques of Zhang and Wong (2015) for taking advantage of the symmetric BTTB structure of the sensitivity matrix associated with a single layer of the `gravity` kernel, but offers greater improvements in the implementation of the forward kernel through the presentation of an optimized calculation of the kernel entries in this matrix. When the stations are on a grid that is uniformly staggered with respect to the coordinate grid on the top surface of the coordinate domain, redundant operations in the calculation of the sensitivity matrix can be eliminated.

Here we consider the specific case in which uniform discretizations of first kind convolution Fredholm integral operators, combined with measurements obtained on uniform staggered grids in the $x$ and $y$ dimensions, yield sensitivity matrices with BTTB structure for each depth layer of the volume. These are symmetric for the `gravity` kernel, symmetric BTTB (symBTTB), Boulanger and Chouteau (2001), but symmetry is lost for the `magnetic` kernel, Bhaskara Rao and Ramesh Babu (1991). Hence further analysis for efficient computation of the general BTTB matrix is required. Here we demonstrate that it is feasible to optimize the calculation of the entries of the `magnetic` kernel matrix, but due to lack of symmetry the computation requirements are increased as compared to the `gravity` kernel. Still, remarkable savings in generating the matrix are achieved. Moreover, while it is not immediately possible to take advantage of BTTB structure when the stations are not on a uniform grid, the calculation of the underlying kernel matrices can still be optimized for arbitrary stations locations, by reuse of common vectors and arrays for each depth layer of the coordinate volume. Furthermore, it is also feasible to interpolate from stations that are not on a uniform grid to a uniform grid, without significant loss of accuracy, as discussed in Bruun and Nielsen (2007). Thus, the approach is useful also for a broader class of practical problems, and in particular due to reduced storage requirements, makes it possible to carry out forward modeling for a finer resolution of the volume domain.

*Overview of main scientific contributions.* Our approach implements and extends the BTTB algorithm for forward modeling with the `magnetic` kernel, and for the inclusion of padding around the domain. (i) We present a detailed derivation of the implementation of the algorithm presented in Chen and Liu (2018) for the forward modeling of the `gravity` problem; (ii) The algorithm is extended to include arbitrary domain padding in $x$ and $y$ directions; (iii) The use of the 2DFFT for matrix transpose multiplication is explained, (needed for solution of the associated inverse problem); (iv) The algorithm applies for general BTTB matrices; (v) The approaches are coded for optimal memory, storage and computation as an open source MATLAB code, https://github.com/renautra/FastBTTB with example simulations at https://math.la.asu.edu/~rosie/research/bttb.html. This can be adapted for any Earth modeling using a convolution kernel that can be used to generate a sensitivity matrix with BTTB structure for each depth layer of the domain when the measurement data are obtained on, or can be interpolated to, a uniform grid.

The paper is organized as follows. In Section 2 we present the general kernel-based forward model, and specifically convolutional kernels, Section 2.1. We demonstrate in Section 2.1.1 how the placement of the measurement stations as uniformly staggered with respect to the coordinate domain yields a distance vector for distances from coordinates to stations that is efficiently stored as a one-dimensional instead of two-dimensional vector, also for padded domains, Section 2.1.2. We then show in Section 2.2 how operators that are spatially invariant yield matrix operators with BTTB structure, and explicitly explain how the relevant entries of the matrices are calculated. In Section 3 we show how these entries are built into the formulation that facilitates the use of the 2DFFT, following the discussions in Chan and Ng (1996) and Vogel (2002). Specific examples are given in Section 4 for the efficient derivation of the entries in the operators for `gravity` and `magnetic` kernels, following Chen and Liu (2018) and Bhaskara Rao and Ramesh Babu (1991), in Sections 4.1 and 4.2, respectively. The presented numerical results in Section 5 validate that the given algorithms are efficient and facilitate forward modeling for problems that are significantly larger as compared to the case when the BTTB structure is not utilized for fast computation with the 2DFFT. Indeed, estimates of the storage requirements demonstrate that it is not possible to store the sensitivity matrices for large problems on standard desktop computers. Software availability is discussed in Section 6 and conclusions with topics for future work are discussed in Section 7. The adopted notation and algorithms are presented in Appendices A and B, respectively.

## 2. Forward modeling

We consider a forward model described by the Fredholm integral equation of the first kind

$$d(a,b,c) = \iiint h(a,b,c,x,y,z)\zeta(x,y,z)dx\,dy\,dz, \tag{1}$$

for which discretization leads to the forward model $\mathbf{d} = G\mathbf{m}$. Here $G$ is the sensitivity matrix, and $\mathbf{d}$ and $\mathbf{m}$ are the discretizations of $d$ and $\zeta$, respectively. We suppose that data measurements for $d(a,b,c)$, on the surface with $c = 0$, are made at $m = s_x s_y$ arbitrary station locations denoted by

$$s_{ij} = (a_{ij}, b_{ij}),\ 1 \le i \le s_x,\ 1 \le j \le s_y. \tag{2}$$

The volume domain, without padding, is discretized into $n = s_x s_y n_z$ uniform prisms, $c_{pqr}$, with coordinates[1]

$$\begin{aligned} x_{p-1} &= (p-1)\Delta_x \quad x_p = p\Delta_x, \quad 1 \le p \le s_x, \\ y_{q-1} &= (q-1)\Delta_y \quad y_q = q\Delta_y, \quad 1 \le q \le s_y, \\ z_{r-1} &= (r-1)\Delta_z \quad z_r = r\Delta_z, \quad 1 \le r \le n_z. \end{aligned} \tag{3}$$

The geometry is illustrated in Fig. 1, in which the configuration of station at location $(i,j)$ relative to volume prism $pqr$ is shown.

The entries in $G$ depend on the integral of kernel $h$ and correspond to the unit contribution from a given prism to a particular station. The ordering of the entries depends on the organization of the volume domain into a vector of length, $n = n_x n_y n_z$. We assume the depth-based multilayer model that yields

$$G = [G^{(1)}, G^{(2)}, \ldots, G^{(n_z)}]. \tag{4}$$

Each $G^{(r)}$ has size $s_x s_y \times n_x n_y = m \times n_r$, where there are $n_r$ prisms in layer $r$, and maps from the prisms in depth layer $r$, with depth coordinates $z_{r-1}$ and $z_r$, to the stations. Further, $G^{(r)}$ decomposes as a block matrix

[1] Note that there are, for example in the $x$−dimension, $s_x$ blocks and hence $s_x + 1$ coordinates describing these blocks.

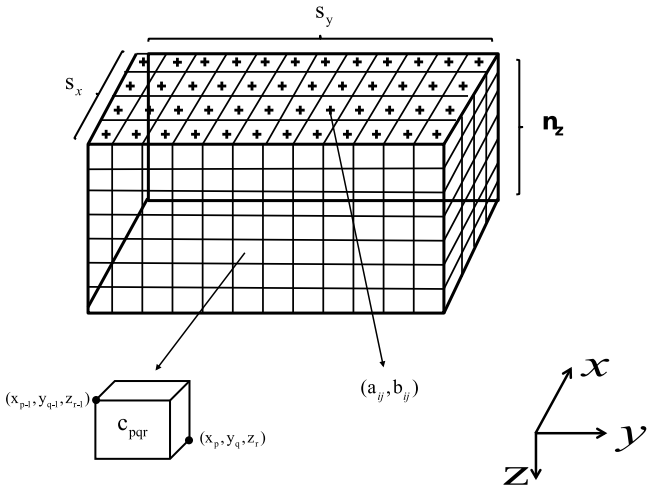



**Fig. 1.** The configuration of prism $pqr$ in the volume relative to a station at location $s_{ij} = (a_{ij}, b_{ij})$. The blocks in the $z$-direction define depth $z \geq 0$ pointing down. Here we assume there is one station located above each prism, giving $s_x$ and $s_y$ blocks in the $x$ and $y$-directions, respectively.

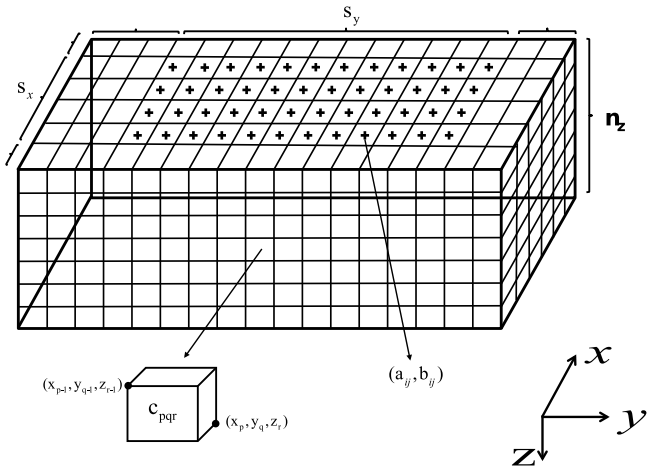



**Fig. 2.** The configuration of the volume domain with padding, explicitly assuming no stations in the padded regions.

with block entries $G^{(r)}_{jq}$, $1 \leq j \leq s_y$, $1 \leq q \leq n_y$ each of size $s_x \times n_x$. Equivalently, a given layer of the volume with $s_y n_y$ blocks is mapped to a one dimensional vector using **row-major** ordering; we sweep through the prisms in the slice for increasing $x$ and fixed $y$ direction. Entry $(G^{(r)})_{k\ell}$, $1 \leq k \leq s_x s_y$, $1 \leq \ell \leq n_x n_y$ represents the contribution from prism $\ell = (q-1)n_x + p$, for $1 \leq q \leq n_y$ and $1 \leq p \leq n_x$, for depth slice $r$, to station $k = (j-1)s_x + i$, $1 \leq j \leq s_y$, $1 \leq i \leq s_x$. Using $\tilde{h}(s_{ij})_{pqr}$ to denote the function that calculates the contribution to station $s_{ij}$ from prism $c_{pqr}$,

$$(G^{(r)})_{k\ell} = \tilde{h}(s_{ij})_{pqr}, \;\; k = (j-1)s_x + i, \;\; \ell = (q-1)n_x + p. \tag{5}$$

In the following we discuss the derivation of the matrix for a given layer $r$ and remove the dependence of $\tilde{h}$ on depth, using $\tilde{h}(s_{ij})_{pq}$ to indicate the contribution to station $s_{ij} = (a_{ij}, b_{ij})$ due to block number $p$ in $x$ and $q$ in $y$. Although the discussion is applied under the assumption of a uniform depth interval, $\Delta_z$, the approach applies equally well for problems in which the layers have different heights, $(\Delta_z)_r$, see e.g. Zhang and Wong (2015).

### 2.1. Spatially invariant kernels

Our discussion focuses on kernels that are spatially invariant in all dimensions:

$$h(a, b, c, x, y, z) = h(x-a, y-b, z-c).$$

For fixed $r$, calculation of $\tilde{h}(s_{ij})_{pqr}$ depends on the differences $(x-a)$ and $(y-b)$ for all station and prism coordinates, (2) and (3), respectively. Using matrices

$$\left.\begin{array}{ll} (DX)_{ij,p} = (x_p - a_{ij}) & 0 \leq p \leq n_x \\ (DY)_{ij,q} = (y_q - b_{ij}) & 0 \leq q \leq n_y \end{array}\right\} 1 \leq i \leq s_x, \;\; 1 \leq j \leq s_y,$$

the distances for block $pq$, $1 \leq p \leq n_x$ and $1 \leq q \leq n_y$, are obtained from distance matrices $(DX)_{p-1}$ and $(DX)_p$ in $x$, and likewise for $y$. For uniform prisms in the $x-$ and $y-$dimensions,

$$(DX)_p = (DX)_{p-1} + \Delta_x, \;\; 1 \leq p \leq n_x, \;\; (DY)_q = (DY)_{q-1} + \Delta_y, \;\; 1 \leq q \leq n_y,$$

and $(DX)_p$ and $(DY)_q$ can be obtained directly from $(DX)_0$ and $(DY)_0$. They are also independent of the layer, regardless of the locations of the stations relative to the prisms. It is practical, therefore, to store $(DX)_0$ and $(DY)_0$ entirely, and update an entire slice of the domain without recalculating $(DX)_0$ and $(DY)_0$ across layers. For the uniform station grid in which $a_{ij}$ is independent of $j$ and $b_{ij}$ is independent of $i$, the sizes of matrices $(DX)_0$ and $(DY)_0$ are reduced in the first dimension to $s_x$ and $s_y$, respectively, and greater optimization is achieved.

#### 2.1.1. Placement of the stations at the center of the cells

Following Boulanger and Chouteau (2001) and Chen and Liu (2018), suppose that the two coordinate systems for the stations and the volume domain, are uniformly staggered in the $x$–$y$ plane stations, such that $a_i = (i - \frac{1}{2})\Delta_x$, $1 \leq i \leq s_x$ and $b_j = (j - \frac{1}{2})\Delta_y$, $1 \leq j \leq s_y$. Then,

$$\begin{array}{llll} (DX)_{i,p} & = x_p - a_i & = (p-1)\Delta_x - (i - \frac{1}{2})\Delta_x & = (p - i - \frac{1}{2})\Delta_x, \quad 1 \leq p \leq n_x + 1 \\ (DY)_{j,q} & = y_q - b_j & = (q-1)\Delta_y - (j - \frac{1}{2})\Delta_y & = (q - j - \frac{1}{2})\Delta_y, \quad 1 \leq q \leq n_y + 1, \end{array} \tag{44}$$

and for all pairs of indices $(i, p)$ and $(j, q)$, the possible paired distances are obtained from the vectors

$$\begin{array}{ll} X_\ell & = (\ell - s_x - \frac{1}{2})\Delta_x, \quad 1 \leq \ell \leq 2s_x \\ Y_k & = (k - s_y - \frac{1}{2})\Delta_y, \quad 1 \leq k \leq 2s_y. \end{array} \tag{6}$$

#### 2.1.2. Introducing padding around the domain

Suppose now that padding is introduced around the domain, with an extra $p_{x_L}$ and $p_{x_R}$ blocks in the $x$-direction. Then the $x-$coordinates are $(-p_{x_L} : (s_x + p_{x_R}))\Delta_x$ for a total of $s_x$ coordinate blocks within the domain, but a total number of blocks $n_x = (s_x + p_{x_L} + p_{x_R})$. Blocks 1 to $p_{x_L}$ are in the padded region to the left of the domain, and blocks $s_x + p_{x_L} + 1$ to $n_x$ are within the padded region to the right. Thus, the coordinates of block $p$ are adjusted to $(p - p_{x_L} - 1)\Delta_x$ to $(p - p_{x_L})\Delta_x$, consistent with (3) for $p_{x_L} = 0$. Likewise, the $y$ coordinates extend from $-p_{y_L} : (s_y + p_{y_R})$ and $n_y = (s_y + p_{y_L} + p_{y_R})$, see Fig. 2. Hence, (6) is replaced by

$$\begin{array}{ll} X_\ell & = (\ell - (s_x + p_{x_L}) - \frac{1}{2})\Delta_x, \quad 1 \leq \ell \leq 2s_x + p_{x_L} + p_{x_R} = n_x + s_x \\ Y_k & = (k - (s_y + p_{y_L}) - \frac{1}{2})\Delta_y, \quad 1 \leq k \leq 2s_y + p_{y_L} + p_{y_R} = n_y + s_y. \end{array} \tag{7}$$

To calculate all possible paired distances we only need to store vectors $X$ and $Y$, as given by (7). This is negligible as compared to the entire storage of the $mn$ entries in the matrix $G$.

### 2.2. Matrix structure for spatially invariant kernels

We now consider the structure of the matrices that arise for spatially invariant kernels, for domains without padding in Section 2.2.1, and

then the modifications that are required when domain padding is introduced, Section 2.2.2. The summary of the discussion is detailed in the presented Algorithms in Appendix A, and can be ignored if the intent is to only use the provided codes.

*2.2.1. Symmetric kernel matrices with Toeplitz block structure without domain padding*

Suppose block matrix $G^{(r)}$ is symBTTB and is defined by its first row block $G^{(r)}_{1q} = G^{(r)}_q$, $1 \le q \le n_y$. Then, with $n_x = s_x$ and $n_y = s_y$,

$$G^{(r)} = \begin{bmatrix} G^{(r)}_1 & G^{(r)}_2 & G^{(r)}_3 & \cdots & G^{(r)}_{n_y} \\ G^{(r)}_2 & G^{(r)}_1 & G^{(r)}_2 & \cdots & G^{(r)}_{n_y-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ G^{(r)}_{s_y} & G^{(r)}_{s_y-1} & G^{(r)}_{s_y-2} & \cdots & G^{(r)}_1 \end{bmatrix}. \tag{8}$$

Each $G^{(r)}_q$ is symmetric and defined by its first row,

$$G^{(r)}_q = \begin{bmatrix} g_{1q} & g_{2q} & g_{3q} & \cdots & g_{n_x q} \\ g_{2q} & g_{1q} & g_{2q} & \cdots & g_{(n_x-1)q} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ g_{s_x q} & g_{(s_x-1)q} & g_{(s_x-2)q} & \cdots & g_{1q} \end{bmatrix}.$$

MATLAB notation can be used to write these matrices compactly in terms of the defining first row (column),

$$G^{(r)}_q = \text{toeplitz}(\mathbf{r}_q),\ \mathbf{r}_q = (g_{1q}, g_{2q}, g_{3q}, \dots, g_{n_x q}), \tag{9}$$

and, with abuse of the same notation as applied to matrices,

$$G^{(r)} = \text{toeplitz}(R),\ R = (G^{(r)}_1, G^{(r)}_2, G^{(r)}_3, \dots, G^{(r)}_{n_y}). \tag{10}$$

From (9) and (10) it is immediate, as discussed in Boulanger and Chouteau (2001) and Chen and Liu (2018), that the generation of $G^{(r)}$ requires only the calculation of its first row. But the first row represents the contributions of all prisms to the first station. Thus, $G^{(r)}$ requires only the calculation of

$$(G^{(r)})_{1\ell} = \tilde{h}(s_{11})_{pq},\ \ell = (q-1)n_x + p,\ 1 \le p \le n_x,\ 1 \le q \le n_y \quad \text{or} \tag{11}$$

$$(G^{(r)}_1)_{1:n_x n_y} = (\ \mathbf{r}_1 \mid \mathbf{r}_2 \mid \cdots \mid \mathbf{r}_{\mathbf{n_y}}\ ) \quad \text{where}$$

$$\mathbf{r}_q = (\tilde{h}(s_{11})_{1q}, \tilde{h}(s_{11})_{2q}, \tilde{h}(s_{11})_{3q}, \dots, \tilde{h}(s_{11})_{n_x q}),\ 1 \le q \le n_y. \tag{12}$$

Equivalently, it is sufficient to calculate only the distances $(DX)_{1,p} = (p-3/2)\Delta_x$ and $(DY)_{1,q} = (q-3/2)\Delta_y$, for $1 \le p \le n_x+1$ and $1 \le q \le n_y+1$, and (6) is replaced by

$$\begin{aligned} X_\ell &= (\ell - \tfrac{3}{2})\Delta_x, \quad 1 \le \ell \le (n_x+1) \\ Y_k &= (k - \tfrac{3}{2})\Delta_y, \quad 1 \le k \le (n_y+1). \end{aligned} \tag{13}$$

*2.2.2. Symmetric kernel matrices with Toeplitz block structure and domain padding*

Suppose now that the domain is padded in the $x$ and $y$ directions, with no real stations within the padded region. To illustrate we take a one-dimensional example with $s_x = 4$, $p_{x_L} = 2$ and $p_{x_R} = 1$. Suppose first that there are artificial stations in the first two blocks, blocks 1, 2, and in the final block, block 7. Then, the single square and symmetric Toeplitz that defines $G^{(r)}$ is

$$G^{(r)}_1 = \left[\begin{array}{ccc|ccc|c} g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & g_7 \\ g_2 & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 \\ \hline g_3 & g_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ g_4 & g_3 & g_2 & g_1 & g_2 & g_3 & g_4 \\ g_5 & g_4 & g_3 & g_2 & g_1 & g_2 & g_3 \\ g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_2 \\ \hline g_7 & g_6 & g_5 & g_4 & g_3 & g_2 & g_1 \end{array}\right] \begin{array}{l} \text{Station 1(Artificial)} \\ \text{Station 2(Artificial)} \\ \hline \text{Station 3} = p_{x_L} + 1 \\ \text{Station 4} \\ \text{Station 5} \\ \text{Station 6} = p_{x_L} + s_x \\ \hline \text{Station 7(Artificial)} \end{array}.$$

This depends on

$$\mathbf{r} = (g_1, g_2, g_3, g_4, g_5, g_6, g_7) = (\tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \tilde{h}(s_1)_3, \dots, \tilde{h}(s_1)_7). \tag{31}$$

But the contribution to the first real station due to all prisms is given by the third row, row $p_{x_L} + 1$ of $G^{(r)}_1$, which is

$$(G^{(r)}_1)_3 = \left(g_3, g_2, g_1, g_2, g_3, g_4, g_5\right), \tag{34}$$

and, using symmetry, the contributions for the real stations are determined by

$$\begin{aligned} (G^{(r)}_1)(p_{x_L}+1 : p_{x_L}+s_x, :) &= \begin{bmatrix} g_3 & g_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ g_4 & g_3 & g_2 & g_1 & g_2 & g_3 & g_4 \\ g_5 & g_4 & g_3 & g_2 & g_1 & g_2 & g_3 \\ g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_2 \end{bmatrix} \\ &= \text{toeplitz}(\mathbf{c}, \mathbf{r}) \text{ where} \\ \mathbf{c} &= (g_3, g_4, g_5, g_6) \text{ and } \mathbf{r} = (g_3, g_2, g_1, g_2, g_3, g_4, g_5). \end{aligned}$$

More generally, in one dimension,

$$\begin{aligned} \mathbf{c} &= (g_{p_{x_L}+1}, g_{p_{x_L}+2}, \dots, g_{p_{x_L}+s_x}) = (\tilde{h}(s_1)_{p_{x_L}+1}, \tilde{h}(s_1)_{p_{x_L}+2}, \dots, \tilde{h}(s_1)_{p_{x_L}+s_x}) \\ &\quad \text{and} \\ \mathbf{r} &= (g_{p_{x_L}+1}, \dots, g_2, g_1, g_2, \dots, g_{n_x - p_{x_L}}) \\ &= (\tilde{h}(s_1)_{p_{x_L}+1}, \dots, \tilde{h}(s_1)_2, \tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \dots, \tilde{h}(s_1)_{n_x - p_{x_L}}). \end{aligned}$$

Extending to the two-dimensional case, and assuming that the first artificial station is in the $(1,1)$ block of the padded domain, then $G^{(r)}_q$, for any $q$, is also Toeplitz and is given by

$$G^{(r)}_q = \text{toeplitz}(\mathbf{c}_q, \mathbf{r}_q),\ 1 \le q \le n_y, \tag{14}$$

$$\mathbf{c}_q = (\tilde{h}(s_{11})_{(p_{x_L}+1)q}, \tilde{h}(s_{11})_{(p_{x_L}+2)q}, \dots, \tilde{h}(s_{11})_{(p_{x_L}+s_x)q}) \text{ and} \tag{15}$$

$$\mathbf{r}_q = (\tilde{h}(s_{11})_{(p_{x_L}+1)q}, \dots, \tilde{h}(s_{11})_{2q}, \tilde{h}(s_{11})_{1q}, \tilde{h}(s_{11})_{2q}, \dots, \tilde{h}(s_{11})_{(n_x - p_{x_L})q}). \tag{16}$$

This is consistent with (11)–(12) for the unpadded case. But notice, also, that the maximum distance between station and coordinates in the $x-$coordinate is $\max(n_x - p_{x_L}, n_x - p_{x_R})\Delta_x = \max(s_x + p_{x_R}, s_x + p_{x_L})\Delta_x$.

It remains to apply the same argument to the structure of the matrix $G^{(r)}$, as to the structure of its individual components, to determine the structure of the symBTTB matrix when padding is applied. Then, consistent with (14)–(16), (10) is replaced by

$$G^{(r)} = \text{toeplitz}(C, R), \tag{17}$$

$$C = (G^{(r)}_{p_{y_L}+1}, \dots, G^{(r)}_{p_{y_L}+s_y}) \text{ and} \tag{18}$$

$$R = (G^{(r)}_{p_{y_L}+1}, \dots, G^{(r)}_2, G^{(r)}_1, G^{(r)}_2, G^{(r)}_3, \dots, G^{(r)}_{n_y - p_{y_L}}). \tag{19}$$

Moreover, since this matrix depends on the first row of the symmetric matrix, defined with respect to the artificial station at $s_{11}$, it is sufficient to still use (13) for the calculation of the relevant distances between the first station and all coordinate blocks. But, from (15)–(16), and (18)–(19), just as we do not calculate all entries $g_j$ in $G^{(r)}_q$, we also do not calculate all the blocks $G^{(r)}_q$, rather the blocks needed are for $q = 1 : \max(n_y - p_{y_L}, n_y - p_{y_R})$. Thus, some savings in memory and computation can be made, when padding is significant relative to $s_x$ and $s_y$, by using

$$\begin{aligned} X_\ell &= (\ell - \tfrac{3}{2})\Delta_x, \quad 1 \le \ell \le s_x + \max(p_{x_R}, p_{x_L}) + 1 \\ Y_k &= (k - \tfrac{3}{2})\Delta_y, \quad 1 \le k \le s_y + \max(p_{y_R}, p_{y_L}) + 1. \end{aligned} \tag{20}$$

### 2.2.3. Unsymmetric kernel matrices with block structure

Consider the unsymmetric BTTB matrix

$$G^{(r)} = \begin{bmatrix} G_1^{(r)} & G_2^{(r)} & G_3^{(r)} & \dots & \dots & G_{n_y}^{(r)} \\ \bar{G}_2^{(r)} & G_1^{(r)} & G_2^{(r)} & \dots & \dots & G_{n_y-1}^{(r)} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \bar{G}_{s_y}^{(r)} & \bar{G}_{s_y-1}^{(r)} & \bar{G}_{s_y-2}^{(r)} & \dots & \dots & G_1^{(r)} \end{bmatrix}, \tag{21}$$

where, without padding, $n_y = s_y$. $G^{(r)}$ depends on the first block row and column only, and, is given by

$$G^{(r)} = \text{toeplitz}(C, R),\ C = (G_1^{(r)}, \bar{G}_2^{(r)}, \dots, \bar{G}_{s_y}^{(r)}),\ R = (G_1^{(r)}, G_2^{(r)}, \dots, G_{s_y}^{(r)}). \tag{22}$$

We use $\bar{G}_j^{(r)}$ to denote the contributions below the diagonal, and $G_q^{(r)}$ for the contributions above the diagonal. None of the block matrices are symmetric and, therefore, to calculate $G^{(r)}$ it is necessary to calculate columns and rows that define $C$ and $R$. Calculating $R$ uses just the first row entries $G_q^{(r)}$, but since each of these is unsymmetric we need also the first columns $\mathbf{c}_q$ of each block in $G_q^{(r)}$.

Using (5), the $G_q^{(r)}$ are given by (14) with

$$\mathbf{r}_q = (\tilde{h}(s_{11})_{1q}, \tilde{h}(s_{11})_{2q}, \dots, \tilde{h}(s_{11})_{n_x q}),\ 1 \le q \le n_y \tag{23}$$

$$\mathbf{c}_q = (\tilde{h}(s_{11})_{1q}, \tilde{h}(s_{21})_{1q}, \dots, \tilde{h}(s_{s_x 1})_{1q}),\ 1 \le q \le n_y. \tag{24}$$

Rather than calculating all $(s_x n_x)n_y$ entries in the first block row of $G^{(r)}$, for each matrix of the block we calculate just $(s_x + n_x)n_y$ entries for its first row and column.

This leaves the calculation of the $\bar{G}_j^{(r)}$, $2 \le j \le s_y$, which by the Toeplitz structure only use $\bar{G}_j^{(r)}$, given by

$$\bar{G}_j^{(r)} = \text{toeplitz}(\bar{\mathbf{c}}_j, \bar{\mathbf{r}}_j),\ 2 \le j \le s_y, \tag{25}$$

$$\bar{\mathbf{c}}_j = (\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{2j})_{11}, \dots, \tilde{h}(s_{s_x j})_{11}),\ 2 \le j \le s_y, \text{ and}$$

$$\bar{\mathbf{r}}_j = (\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{1j})_{21}, \dots, \tilde{h}(s_{1j})_{n_x 1}),\ 2 \le j \le s_y. \tag{26}$$

This requires not only all distances between the first station and all prism coordinates, as in (13), but also for all stations and the first coordinate block (for the first column of $G^{(r)}$) which uses $X_\ell = (\ell + \frac{1}{2})\Delta_x$, $-s_x \le \ell \le 0$, and likewise for $Y_k$. Thus, we require the full set of differences (6).

### 2.2.4. Unsymmetric kernel matrices with block structure and domain padding

As for the discussion on domain padding with the symmetric kernel in Section 2.2.2, we first present an example using one dimension. We again assume $s_x = 4$, $p_{x_L} = 2$ and $p_{x_R} = 1$ and that there are artificial stations in the first two blocks, blocks 1, 2, and in the final block, block 7. Then, the single square but unsymmetric Toeplitz matrix that defines $G^{(r)}$ is

$$G_1^{(r)} = \left[\begin{array}{ccc|ccc|c} g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & g_7 \\ \gamma_2 & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 \\ \hline \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 \\ \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 \\ \gamma_6 & \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 \\ \hline \gamma_7 & \gamma_6 & \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 \end{array}\right] \begin{array}{l} \text{Station 1(Artificial)} \\ \text{Station 2(Artificial)} \\ \hline \text{Station 3} = p_{x_L} + 1 \\ \text{Station 4} \\ \text{Station 5} \\ \text{Station 6} = p_{x_L} + s_x \\ \hline \text{Station 7(Artificial)} \end{array}$$

This depends on

$$\mathbf{r} = (g_1, g_2, g_3, g_4, g_5, g_6, g_7) = (\tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \dots, \tilde{h}(s_1)_6, \tilde{h}(s_1)_7)$$

$$\mathbf{c} = (g_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6, \gamma_7) = (\tilde{h}(s_1)_1, \tilde{h}(s_2)_1, \dots, \tilde{h}(s_6)_1, \tilde{h}(s_7)_1).$$

But again the required rows of $G_1^{(r)}$ correspond to the real stations

$$(G_1^{(r)})(p_{x_L} + 1 : p_{x_L} + s_x, :) = \begin{bmatrix} \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 \\ \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 \\ \gamma_6 & \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 \end{bmatrix}$$

$$= \text{toeplitz}(\mathbf{c}, \mathbf{r}) \text{ where}$$

$$\mathbf{c} = (\gamma_3, \gamma_4, \gamma_5, \gamma_6) \text{ and } \mathbf{r} = (\gamma_3, \gamma_2, g_1, g_2, g_3, g_4, g_5).$$

More generally,

$$\mathbf{c} = (\gamma_{p_{x_L}+1}, \gamma_{p_{x_L}+2}, \dots, \gamma_{p_{x_L}+s_x}) = (\tilde{h}(s_{p_{x_L}+1})_1, \tilde{h}(s_{p_{x_L}+2})_1, \dots, \tilde{h}(s_{p_{x_L}+s_x})_1)$$

and

$$\mathbf{r} = (\gamma_{p_{x_L}+1}, \gamma_{p_{x_L}}, \dots, \gamma_2, g_1, g_2, \dots, g_{n_x - p_{x_L}})$$

$$= (\tilde{h}(s_{p_{x_L}+1})_1, \tilde{h}(s_{p_{x_L}})_1, \dots, \tilde{h}(s_2)_1, \tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \dots, \tilde{h}(s_1)_{n_x - p_{x_L}}).$$

Extending to the two-dimensional case, with the same assumptions as in Section 2.2.2, $G_q^{(r)}$ is obtained as

$$G_q^{(r)} = \text{toeplitz}(\mathbf{c}_q, \mathbf{r}_q),\ 1 \le q \le n_y, \tag{27}$$

$$\mathbf{c}_q = (\tilde{h}(s_{(p_{x_L}+1)1})_{1q}, \tilde{h}(s_{(p_{x_L}+2)1})_{1q}, \dots, \tilde{h}(s_{(p_{x_L}+s_x)1})_{1q}) \text{ and}$$

$$\mathbf{r}_q = (\tilde{h}(s_{(p_{x_L}+1)1})_{1q}, \tilde{h}(s_{p_{x_L}1})_{1q}, \dots, \tilde{h}(s_{21})_{1q} \tilde{h}(s_{11})_{1q}, \dots, \tilde{h}(s_{11})_{(n_x - p_{x_L})q}).$$

This is consistent with (23)–(24) for the unpadded case.

Turning to the column block entries, first observe that $\bar{G}_1^{(r)} = G_1^{(r)}$, and so we examine $\bar{G}_j^{(r)}$ which represents stations 1 to $n_x$ (both real and artificial) in the $x$-direction for a fixed $j$ coordinate in the $y$-direction. Then, with the same example for choices of $s_x$, $p_{x_L}$ and $p_{x_R}$,

$$\bar{G}_j^{(r)} = \left[\begin{array}{cc|cccc|c} \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 & \bar{g}_5 & \bar{g}_6 & \bar{g}_7 \\ \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 & \bar{g}_5 & \bar{g}_6 \\ \hline \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & g_4 & \bar{g}_5 \\ \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 \\ \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 \\ \bar{\gamma}_6 & \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 \\ \hline \bar{\gamma}_7 & \bar{\gamma}_6 & \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 \end{array}\right] \begin{array}{l} \text{Station 1(Artificial)} \\ \text{Station 2(Artificial)} \\ \hline \text{Station 3} = p_{x_L} + 1 \\ \text{Station 4} \\ \text{Station 5} \\ \text{Station 6} = p_{x_L} + s_x \\ \hline \text{Station 7(Artificial)} \end{array}$$

This depends on

$$\bar{\mathbf{r}} = (\bar{g}_1, \bar{g}_2, \bar{g}_3, \bar{g}_4, \bar{g}_5, \bar{g}_6, \bar{g}_7) = (\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{1j})_{21}, \dots, \tilde{h}(s_{1j})_{61}, \tilde{h}(s_{1j})_{71})$$

$$\bar{\mathbf{c}} = (\bar{g}_1, \bar{\gamma}_2, \bar{\gamma}_3, \bar{\gamma}_4, \bar{\gamma}_5, \bar{\gamma}_6, \bar{\gamma}_7) = (\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{2j})_{11}, \dots, \tilde{h}(s_{6j})_{11}, \tilde{h}(s_{7j})_{11}).$$

But again, since stations 1 to 2 and 7 are artificial, we only need

$$(\bar{G}_j^{(r)})(p_{x_L} + 1 : p_{x_L} + s_x, :) = \begin{bmatrix} \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 & \bar{g}_5 \\ \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 \\ \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 \\ \bar{\gamma}_6 & \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 \end{bmatrix}$$

$$= \text{toeplitz}(\mathbf{c}, \mathbf{r}) \text{ where}$$

$$\mathbf{c} = (\bar{\gamma}_3, \bar{\gamma}_4, \bar{\gamma}_5, \bar{\gamma}_6) \text{ and } \mathbf{r} = (\bar{\gamma}_3, \bar{\gamma}_2, \bar{g}_1, \bar{g}_2, \bar{g}_3, \bar{g}_4, \bar{g}_5).$$

Thus, in two dimensions, the first column block entries are $\bar{G}_j^{(r)}$, $1 \le j \le n_y$, with

$$\bar{G}_j^{(r)} = \text{toeplitz}(\bar{\mathbf{c}}_j, \bar{\mathbf{r}}_j),\ 1 \le j \le s_y + p_{y_L} + p_{y_R},$$

$$\bar{\mathbf{c}}_j = (\tilde{h}(s_{(p_{x_L}+1)j})_{11}, \tilde{h}(s_{(p_{x_L}+2)j})_{11}, \dots, \tilde{h}(s_{(p_{x_L}+s_x)j})_{11}) \text{ and} \tag{28}$$

$$\bar{\mathbf{r}}_j = (\tilde{h}(s_{(p_{x_L}+1)j})_{11}, \tilde{h}(s_{p_{x_L}j})_{11}, \dots, \tilde{h}(s_{2j})_{11}, \tilde{h}(s_{1j})_{11}, \dots, \tilde{h}(s_{1j})_{(n_x - p_{x_L})1}).$$

But now (22) is replaced by the block Toeplitz matrix

$$G^{(r)} = \text{toeplitz}(C, R), \tag{29}$$

$$C = (\bar{G}_{p_{y_L}+1}^{(r)}, \dots, \bar{G}_{p_{y_L}+s_y}^{(r)}) \text{ and}$$

$$R = (\bar{G}_{p_{y_L}+1}^{(r)}, \bar{G}_{p_{y_L}}^{(r)}, \dots, \bar{G}_2^{(r)}, G_1^{(r)}, \dots, G_{n_y - p_{y_L}}^{(r)}).$$

Here each block matrix is the subset of rows corresponding to the real stations, as noted in (27) and (28). Moreover, we conclude that (27) is

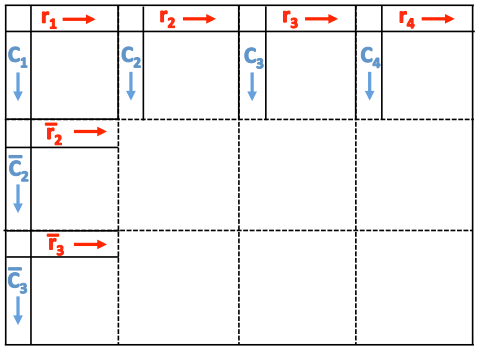



**Fig. 3.** Required entries from a matrix $G^{(r)}$ in order to generate all entries $G^{(r)}$ using the BTTB structure. This shows considerable savings can be accrued in calculating the entries of $G^{(r)}$ when using the structure.

applied only for $1 \le q \le n_y - p_{y_L} = s_y + p_{y_R}$ and (28) for $1 \le j \le p_{y_L} + s_y$, reducing the dimension of the required $Y$ in the $y-$direction. Likewise, $X$ is reduced because of the padding impacting the required entries for generating both $G_q^{(r)}$ and $\bar{G}_j^{(r)}$. Thus while the required vectors are given by (6), with $n_x$ replacing $s_x$ and $n_y$ replacing $s_y$, their lengths can be reduced as for the symmetric case, (20), by using

$$\begin{aligned} X_\ell &= (\ell - (s_x + \max(p_{x_R}, p_{x_L})) - \tfrac{1}{2})\Delta_x, \quad 1 \le \ell \le 2(s_x + \max(p_{x_R}, p_{x_L})) \\ Y_k &= (k - (s_y + \max(p_{y_R}, p_{y_L})) - \tfrac{1}{2})\Delta_y, \quad 1 \le k \le 2(s_y + \max(p_{y_R}, p_{y_L})). \end{aligned} \tag{30}$$

This effectively assumes the calculation of $\bar{G}_1^{(r)}$ as well as $G_1^{(r)}$, whereas only one is calculated in practice, since $\bar{G}_1^{(r)} = G_1^{(r)}$.

The plot in Fig. 3 illustrates the unique entries from $G^{(r)}$ that define its BTTB structure. We note that, while the discussion is focused on the situation in which observation points are on a uniform grid and exactly staggered with respect to the coordinate domain, it is sufficient for generating matrices with the BTTB structure that the observation points are uniform with respect to a coordinate domain which is uniform in the $x-$ and $y-$ dimensions.

## 3. Circulant operators and the 2D FFT

**Definition 1** (*Circulant*)**.** The Toeplitz matrix in which the defining vectors $\mathbf{c}$ and $\mathbf{r}$, each of length $2s_x - 1$, have entries that are related by $r_i = c_{(2s_x+1-i)}$ for $2 \le i \le 2s_x - 1$, is **circulant**.

A circulant matrix is defined solely by its first column or first row. Here we will use the first column. Any Toeplitz matrix can be embedded in its circulant extension, as illustrated for the simple example with $s_x = n_x = 3$

$$\left[\begin{array}{ccc|cc} g_1 & g_2 & g_3 & \gamma_3 & \gamma_2 \\ \gamma_2 & g_1 & g_2 & g_3 & \gamma_3 \\ \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 \\ \hline g_3 & \gamma_3 & \gamma_2 & g_1 & g_2 \\ g_2 & g_3 & \gamma_3 & \gamma_2 & g_1 \end{array}\right].$$

In the same way, a BTTB matrix can be embedded in a block circulant matrix. Thus, matrices (8) and (21) can be embedded in block circulant matrices, in which also each Toeplitz block $G_q^{(r)}$ and $\bar{G}_q^{(r)}$ is embedded in a $(2s_x - 1) \times (2s_x - 1)$ circulant matrix. This yields a matrix that is Block Circulant with Circulant Blocks (BCCB). It is the structure of a BCCB matrix that facilitates the use of the 2DFFT to efficiently evaluate forward matrix multiplication with a BTTB matrix. Specifically, BTTB matrix–vector multiplication can be applied at reduced computational cost by using a BCCB extension combined with the FFT for implementing a discrete convolution, Vogel (2002). The required components that provide the FFT approach are now discussed.

**Definition 2** (*Exchange Matrix*)**.** The **exchange** matrix is the $m \times m$ matrix $J_m$ which is everywhere 0 except for 1's on the principal counter diagonal.

Given arbitrary vector $\mathbf{x}$ of length $m$, with entries $x_i$, $1 \le i \le m$, then $J_m\mathbf{x} = \mathbf{y}$ where $\mathbf{y}_i = \mathbf{x}_{m-i+1}$; it is the vector with the order of the entries reversed. Equivalently, for matrix $A$ with rows $\mathbf{a}_i$, $1 \le i \le m$, then $J_m A = B$ where $B$ is the matrix with rows in reverse order, $\mathbf{b}_i = \mathbf{a}_{m-i+1}$. Further, multiplying on the right reorders the columns in reverse order. Specifically, $J_m^T = J_m$, and thus $\mathbf{y}^T = (J_m\mathbf{x})^T = \mathbf{x}^T J_m^T = \mathbf{x}^T J_m$ and the column entries of $\mathbf{y}$ are in reverse order as compared to $\mathbf{x}$. In the same way, $AJ_m$ gives the matrix with the columns in reverse order. In MATLAB the exchange matrix is implemented using the functions `flipud` and `fliplr`, for "up-down" and "left–right", for multiplication with $J_m$ on the left and right, respectively.

The exchange matrix yields a compact notation for the entries that define the circulant extension of a Toeplitz matrix. For matrix $G_q^{(r)}$ which depends on $\mathbf{r}_q$, as given in (9), then the defining first row for the circulant extension for each symmetric $G_q^{(r)}$ is given by

$$\mathbf{r}_q^{\text{ext}} = \begin{pmatrix} \mathbf{r}_q \\ J_{s_x-1}\mathbf{r}_q(2:s_x) \end{pmatrix}. \tag{31}$$

For the unsymmetric case for $G_q^{(r)}$, as given in (23)–(24), the circulant extension uses

$$\mathbf{c}_q^{\text{ext}} = \begin{pmatrix} \mathbf{c}_q \\ J_{s_x-1}\mathbf{r}_q(2:s_x) \end{pmatrix} \text{ and } \mathbf{r}_q^{\text{ext}} = \begin{pmatrix} \mathbf{r}_q \\ J_{s_x-1}\mathbf{c}_q(2:s_x) \end{pmatrix}. \tag{32}$$

An equivalent expression applies for the circulant extension for each $\bar{G}_j^{(r)}$ as defined in (25)–(26) using the extension for $\bar{\mathbf{c}}_j$ and $\bar{\mathbf{r}}_j$. While (31) and (32) can be used as the defining vectors to explicitly generate the extensions $(G_j^{(r)})^{\text{circ}}$ and $(\bar{G}_j^{(r)})^{\text{circ}}$ as Toeplitz matrices, again using $\texttt{toeplitz}(\mathbf{c}_j^{\text{ext}}, \mathbf{r}_j^{\text{ext}})$, we note that the intent is to define the vectors that define the extensions but not to generate the extensions. Moreover, $\mathbf{r}_j^{\text{ext}}$ is as noted defined explicitly from $\mathbf{c}_q^{\text{ext}}$ and we focus entirely on the columns $\mathbf{c}_j^{\text{ext}}$. We also note that this definition for generating the extension differs from that used in Li et al. (2018), Vogel (2002) and Zhang and Wong (2015); the extra 0 is omitted for convenience. We also directly define the circulant extension instead of performing a series of transformations.

We now turn to the defining set of vectors needed for the circulant extension of (21), which depends on its first block row and column as given in (22). Using the same analogy as with the block Toeplitz matrix and using Definition 1, the extension for $G^{(r)}$ requires the extensions of $C$ and $R$ in (22), thus for entries $G_j^{(r)}$ and $\bar{G}_j^{(r)}$ for $1 \le j \le s_x$. Moreover, the circulant extension, as in the one-dimensional case will depend entirely either on the extension of $C$ or $R$, denoted by $C^{\text{ext}}$ and $R^{\text{ext}}$, but again we do not form $\texttt{toeplitz}(C^{\text{ext}}, R^{\text{ext}})$. We assume the use of the extension for $C$ only, and note that $C^{\text{ext}}$ is completely defined by $T^{\text{circ}}$, dropping the dependence on slice $r$. Then, using (32)

$$T^{\text{circ}} = \begin{pmatrix} \bar{\mathbf{c}}_1^{\text{ext}} & \cdots & \bar{\mathbf{c}}_{s_y}^{\text{ext}} & \mathbf{c}_{s_y}^{\text{ext}} & \cdots & \mathbf{c}_2^{\text{ext}} \end{pmatrix}, \tag{33}$$

and is of size $(2s_x - 1) \times (2s_y - 1)$

Using $\mathbf{u} = \texttt{vec}(U)$ to denote the vectorization of matrix $U$, the two-dimensional convolution product $G^{(r)}\mathbf{u}$, can be computed using $T^{\text{circ}}$ which defines the circular extension of $G^{(r)}$, Vogel (2002). Suppose that $(G^{(r)})^{\text{circ}}$ is defined by $T^{\text{circ}}$, and let $\mathbf{w} = \texttt{vec}(W)$. Then, the reshaped convolution product $\texttt{array}((G^{(r)})^{\text{circ}}\mathbf{w})$, where `array()` is the inverse of `vec()`, can be computed by

$$\texttt{array}((G^{(r)})^{\text{circ}}\mathbf{w}) = T^{\text{circ}} \star W = \texttt{ifft2}(\texttt{fft2}(T^{\text{circ}}) \cdot * \texttt{fft2}(W)). \tag{34}$$

Here $\star$ denotes convolution, `fft2` denotes the two-dimensional FFT, and `ifft2` denotes the inverse two-dimensional FFT, and we introduce

$\hat{T}^{\text{circ}} = \texttt{fft2}(T^{\text{circ}})$ and $\hat{W} = \texttt{fft2}(W)$. To obtain $G^{(r)}\mathbf{u}$ from this product, notice that $G^{(r)}$ is in the upper left block of $(G^{(r)})^{\text{circ}}$. Thus, defining $W$ of size $(2s_x - 1) \times (2s_y - 1)$ by

$$W = \begin{bmatrix} U & 0_{s_x(s_y-1)} \\ 0_{(s_x-1)s_y} & 0_{(s_x-1)(s_y-1)} \end{bmatrix}, \tag{35}$$

using $0_{mn}$ to denote a matrix of zeros of size $m \times n$, and with $U$ of size $s_x \times s_y$, $\texttt{array}(G^{(r)}\mathbf{u})$ is the upper left $s_x \times s_y$ block of $\texttt{array}((G^{(r)})^{\text{circ}}\mathbf{w})$ in (34). Moreover, $\texttt{array}((G^{(r)})^{\text{circ}})$ does not need to be formed explicitly for this product. Instead elements of $T^{\text{circ}}$ are calculated using (33).

It is immediate that a set of equivalent steps can be used to calculate $(G^{(r)})^T\mathbf{v}$, where $\mathbf{v} = \texttt{vec}(V)$ for matrix $V$, since $(G^{(r)})^T$ is also BTTB, and the defining first column for $((G^{(r)})^{\text{circ}})^T$ is the first row of $(G^{(r)})^{\text{circ}}$. Thus, as shown by Bruun and Nielsen (2007), multiplication using the transpose matrix is then represented by the circulant extension as

$$\texttt{array}(((G^{(r)})^{\text{circ}})^T\mathbf{v}) = \texttt{ifft2}(\texttt{conj}(\texttt{fft2}(T^{\text{circ}})) \cdot * \texttt{fft2}(\bar{W})). \tag{36}$$

Although computing $\texttt{conj}(\texttt{fft2}(T^{\text{circ}}))$ once and storing the result may provide small computational reduction in some situations, this is generally unnecessary.

### 3.1. Convolution with domain padding

Suppose that padding is introduced around the domain and, consistent with (14) and (28), assume that the indices for $\mathbf{r}_p$ and $\mathbf{c}_q$ are from the first row and column of the padded domain. Then, for the case of the symBTTB matrix, (31) is replaced by

$$\mathbf{r}_q^{\text{ext}} = \begin{pmatrix} \mathbf{r}_q \\ J_{s_x-1}\mathbf{c}_q(2:s_x) \end{pmatrix}, \quad \mathbf{c}_q^{\text{ext}} = \begin{pmatrix} \mathbf{c}_q \\ J_{n_x-1}\mathbf{r}_q(2:n_x) \end{pmatrix}, \tag{37}$$

where $\mathbf{r}_q$ and $\mathbf{c}_q$ are defined using (15) and (16), respectively. But now since $\mathbf{r}_q$ is defined by $\mathbf{c}_q$ for the symmetric case we can use just $\mathbf{c}_q^{\text{ext}}$. Each vector is of length $s_x$ for $\mathbf{c}_q$ and $n_x - 1$ for $J_{n_x-1}(\mathbf{r}_q(2:n_x))$. Then using (18), the BCCB extension is defined by the replacement of (33) by the matrix

$$T^{\text{circ}} = \left( \begin{array}{ccccccc|ccc} \mathbf{c}^{\text{ext}}_{1+p_{y_L}} & \cdots & \mathbf{c}^{\text{ext}}_{s_y+p_{y_L}} & \mathbf{c}^{\text{ext}}_{s_y+p_{y_R}} & \cdots & \mathbf{c}^{\text{ext}}_{2} & \mathbf{c}^{\text{ext}}_{1} & \cdots & \mathbf{c}^{\text{ext}}_{p_{y_L}} \end{array} \right), \tag{38}$$

of size $(s_x + n_x - 1) \times (s_y + n_y - 1)$. For the unsymmetric case (38) is replaced by

$$T^{\text{circ}} = \left( \begin{array}{ccccccc|ccc} \bar{\mathbf{c}}^{\text{ext}}_{1+p_{y_L}} & \cdots & \bar{\mathbf{c}}^{\text{ext}}_{s_y+p_{y_L}} & \mathbf{c}^{\text{ext}}_{s_y+p_{y_R}} & \cdots & \mathbf{c}^{\text{ext}}_{2} & \bar{\mathbf{c}}^{\text{ext}}_{1} & \cdots & \bar{\mathbf{c}}^{\text{ext}}_{p_{y_L}} \end{array} \right), \tag{39}$$

where $\bar{\mathbf{c}}_j^{\text{ext}}$ is obtained as in (37) but using $\bar{\mathbf{c}}_j$ and $\bar{\mathbf{r}}_j$ from (28). In any case in which $p_{y_L} = 0$ the end block is removed. Moreover, due to $G^{(r)}\mathbf{u}$ of size $s_x s_y$, when $\mathbf{u}$ is of size $n_x n_y$, the definition of $W$ in (35) is replaced by

$$W = \begin{bmatrix} U & 0_{n_x(s_y-1)} \\ 0_{(s_x-1)n_y} & 0_{(s_x-1)(s_y-1)} \end{bmatrix}, \quad U \in \mathcal{R}^{n_x \times n_y}. \tag{40}$$

The transpose operation can still be obtained directly from $T^{\text{circ}}$, but notice that for $\mathbf{v}$ of size $s_x s_y$, $(G^{(r)})^T\mathbf{v}$ is of size $n_x n_y$, and in this case (35) is replaced by

$$\bar{W} = \begin{bmatrix} V & 0_{s_x(n_y-1)} \\ 0_{(n_x-1)s_y} & 0_{(n_x-1)(n_y-1)} \end{bmatrix}, \quad V \in \mathcal{R}^{s_x \times s_y}. \tag{41}$$

We illustrate in Fig. 4 the matrix $T^{\text{circ}}$ that is generated using the row and column entries from the BTTB matrix $G^{(r)}$ as shown in Fig. 3.

## 4. Optimizing the calculations for specific kernels

Chen and Liu (2018) demonstrated that considerable savings are realized in the generation of $T^{\text{circ}}$ through optimized calculations of the entries for forward modeling of the `gravity` problem. Here we focus on both improving that optimization, and with the generation of an optimized and stable calculation for the entries of the `magnetic` kernel, Bhaskara Rao and Ramesh Babu (1991).

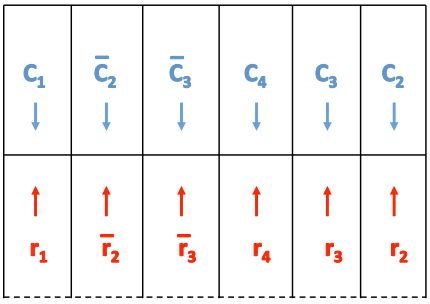



**Fig. 4.** The configuration of $T^{\text{circ}}$, where the arrow denotes the direction of the vector in ascending order. The dotted line indicates that the first elements of each $\mathbf{r}_q$ and $\bar{\mathbf{r}}_q$ are omitted in the construction of $T^{\text{circ}}$.

### 4.1. Gravity kernel calculation

The `gravity` kernel generates a symBTTB matrix for each slice in depth ($z$-direction). According to Boulanger and Chouteau (2001), and as used in Chen and Liu (2018), the contribution of the kernel from the prism at point $(p, q)$ on the volume grid, (where $x$ points East and $y$ points North), to the station at location $(a, b, 0)$, is given by

$$\tilde{h}(a,b,c)_{pq} = \gamma \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}(-1)^i(-1)^j(-1)^k \left( \mathrm{Z}_k \arctan\frac{\mathrm{X}_i\Upsilon_j}{\mathrm{Z}_k\mathrm{P}_{ij}^k} - \mathrm{X}_i \ln\left(\mathrm{P}_{ij}^k + \Upsilon_j\right) - \Upsilon_j \ln\left(\mathrm{P}_{ij}^k + \mathrm{X}_i\right) \right).$$

Here $\gamma$ is the gravitational constant and

$$\begin{aligned} \mathrm{X}_1 &= x_{p-1} - a, & \mathrm{X}_2 &= x_p - a \\ \Upsilon_1 &= y_{q-1} - b & \Upsilon_2 &= y_q - b \\ \mathrm{Z}_1 &= z_{r-1} - c, & \mathrm{Z}_2 &= z_r - c \\ R_{ij}^2 &= \mathrm{X}_i^2 + \Upsilon_j^2 & \mathrm{P}_{ij}^k &= \sqrt{R_{ij}^2 + \mathrm{Z}_k^2}. \end{aligned} \tag{42}$$

Specifically, we need

$$\sum_i\sum_j(-1)^{i+j+1}\left(\left(\mathrm{Z}_1 \arctan\frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_1(R_1)_{ij}} - \mathrm{Z}_2 \arctan\frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_2(R_2)_{ij}}\right) - \mathrm{X}_i\left(\ln((R_1)_{ij} + \Upsilon_j) - \ln((R_2)_{ij} + \Upsilon_j)\right) - \Upsilon_j\left(\ln((R_1)_{ij} + \mathrm{X}_i) - \ln((R_2)_{ij} + \mathrm{X}_i)\right)\right).$$

Here

$$R_1 = \sqrt{R.\hat{}2 + \mathrm{Z}_1^2}, \quad R_2 = \sqrt{R.\hat{}2 + \mathrm{Z}_2^2},$$

and operations involve elementwise powers and multiplications. Using the notation in Chen and Liu (2018), we write the summand, ignoring $\sum_j(-1)^{i+j+1}$, as

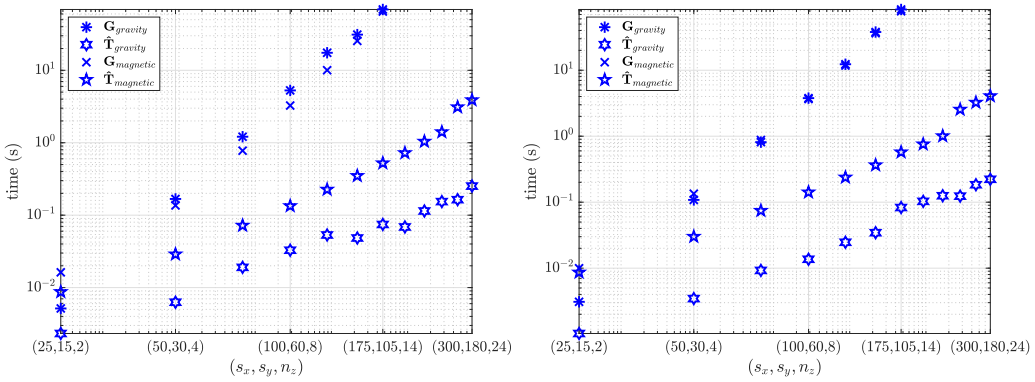
$$\left(\left(\mathrm{Z}_1(CM5)_{ij} - \mathrm{Z}_2(CM6)_{ij}\right) - \mathrm{X}_i\left((CM3)_{ij} - (CM4)_{ij}\right) - \Upsilon_j\left((CM1)_{ij} - (CM2)_{ij}\right)\right).$$

Notice that $(CM3)_{ij} - (CM4)_{ij}$ is a logarithmic difference (and also for $(CM1)_{ij} - (CM2)_{ij}$). Thus, the differences can be replaced by

$$CMX = \ln\frac{\mathrm{X} + R_1}{\mathrm{X} + R_2}, \text{ and } CMY = \ln\frac{\Upsilon + R_1}{\Upsilon + R_2}. \tag{43}$$

Moreover, we can directly calculate

$$CM5Z = \mathrm{Z}_1 \arctan\frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_1(R_1)_{ij}}, \text{ and } CM6Z = \mathrm{Z}_2 \arctan\frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_2(R_2)_{ij}}.$$




(a) $p = 0$ (b) $p = 0.05$

**Fig. 5.** Running times for generating $G$ using Algorithms 1 and 3 and $\hat{T}$ using Algorithms 2 and 4 with 0% padding in Fig. 5(a), and with 5% padding in Fig. 5(b).

Hence, the summand of the triple sum is replaced by

$$\left(((CM5Z)_{ij} - (CM6Z)_{ij}\right) - \mathrm{X}_i(CMY)_{ij} - \Upsilon_j(CMX)_{ij}\big) = CM_{ij}.$$

Now, $\mathrm{X}_1$, $\mathrm{X}_2$ are entries from $X$, and $\Upsilon_1$, $\Upsilon_2$ are entries from $Y$. Thus, given (7) and assuming that $X$ and $Y$ are stored in row vectors we can form matrices $XY = X(:) * Y$ and $R^2 = X(:).\hat{}2 + Y.\hat{}2$ which are of size $(n_x+1)\times(n_y+1)$, and are independent of the $z$ coordinates. Thus, we save substantial computation by only calculating $XY$ and $R^2$ once for all slices, and for each slice we only calculate one row of the matrix. Since these are based on matrices we can calculate the double sum for multiple coordinates by shifting each $ij$ matrix to the right in $i$ and in $j$ with the appropriate sign, and obtain the entire sum in one line by correct indexing into the matrices. Suppose that $CM$ has size $(n_x+1)\times(n_y+1)$ then we obtain a matrix that can be reshaped to a row vector

$$\begin{aligned} g = CM(1:n_x, 1:n_y) - CM(1:n_x, 2:n_y+1) - CM(2:n_x+1, 1:n_y) \\ + CM(2:n_x+1, 2:n_x+1) \\ \tilde{h}(a_1, b_1, 0) = -\gamma g(:). \end{aligned}$$

Thus, we calculate the first row of depth block $r$ by an evaluation of (5) for all coordinate contributions to station 1 in one step. Note that the simplification (43) is a further optimization of the calculation of entries for $G$ as compared to that given in Chen and Liu (2018). The details of the use and application of the `gravity` problem in the context of forward algorithms for symBTTB matrices are provided in Algorithms 1–2 with the `gravity` function in Algorithm 5.

### 4.2. Magnetic kernel

We now extend the idea of the optimization of the `gravity` kernel calculation to the calculation of the `magnetic` kernel, under the assumption that there is no remanence magnetization or self-demagnetization, so that the magnetization vector is parallel to the Earth's `magnetic` field.[2] Although the `magnetic` kernel is not symmetric, its discretization does lead to a BTTB matrix, and hence the discussion of Section 2.2.3 is relevant. We apply the simplifications of Bhaskara Rao and Ramesh Babu (1991) for the evaluation of the `magnetic` kernel. Using their notation Bhaskara Rao and Ramesh Babu (1991, eq. (3))

$$\tilde{h}(a,b,0)_{pq} = \tilde{H}(G_1 \ln F_1 + G_2 \ln F_2 + G_3 \ln F_3 + G_4 F_4 + G_5 F_5). \tag{44}$$

The constants $g_i = \tilde{H}G_i$[3] depend on the volume orientation and `magnetic` constants, Bhaskara Rao and Ramesh Babu (1991), and here we assume that $x$ points North and $y$ points East. Taking advantage of the notation in (42), the variables $F_i$ are given by

$$F_1 = \frac{(\mathrm{P}^2_{11}+\mathrm{X}_1)(\mathrm{P}^1_{21}+\mathrm{X}_2)(\mathrm{P}^1_{12}+\mathrm{X}_1)(\mathrm{P}^2_{22}+\mathrm{X}_2)}{(\mathrm{P}^1_{11}+\mathrm{X}_1)(\mathrm{P}^2_{21}+\mathrm{X}_2)(\mathrm{P}^2_{12}+\mathrm{X}_1)(\mathrm{P}^1_{22}+\mathrm{X}_2)}$$

$$F_2 = \frac{(\mathrm{P}^2_{11}+\Upsilon_1)(\mathrm{P}^1_{21}+\Upsilon_1)(\mathrm{P}^1_{12}+\Upsilon_2)(\mathrm{P}^2_{22}+\Upsilon_2)}{(\mathrm{P}^1_{11}+\Upsilon_1)(\mathrm{P}^2_{21}+\Upsilon_1)(\mathrm{P}^2_{12}+\Upsilon_2)(\mathrm{P}^1_{22}+\Upsilon_2)}$$

$$F_3 = \frac{(\mathrm{P}^2_{11}+\mathrm{Z}_2)(\mathrm{P}^1_{21}+\mathrm{Z}_1)(\mathrm{P}^1_{12}+\mathrm{Z}_1)(\mathrm{P}^2_{22}+\mathrm{Z}_2)}{(\mathrm{P}^1_{11}+\mathrm{Z}_2)(\mathrm{P}^2_{21}+\mathrm{Z}_1)(\mathrm{P}^2_{12}+\mathrm{Z}_1)(\mathrm{P}^1_{22}+\mathrm{Z}_2)}$$

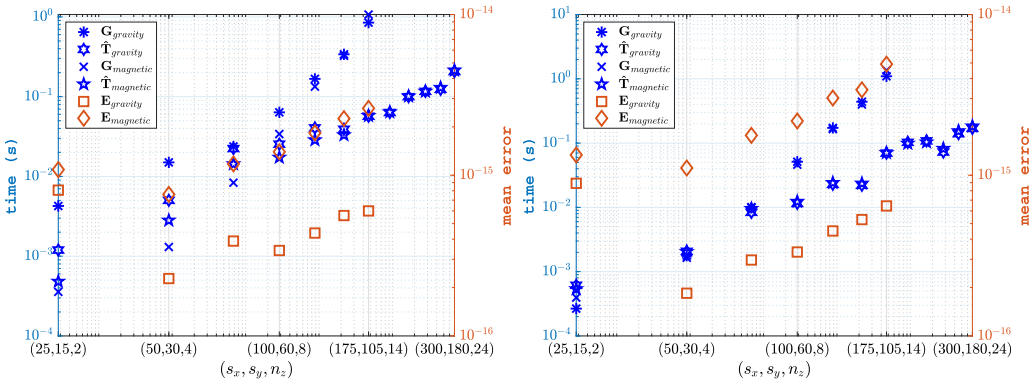
$$\begin{aligned} F_4 = \arctan\frac{\mathrm{X}_2\mathrm{Z}_2}{\mathrm{P}^2_{22}\Upsilon_2} - \arctan\frac{\mathrm{X}_1\mathrm{Z}_2}{\mathrm{P}^2_{12}\Upsilon_2} - \arctan\frac{\mathrm{X}_2\mathrm{Z}_2}{\mathrm{P}^2_{21}\Upsilon_1} + \arctan\frac{\mathrm{X}_1\mathrm{Z}_2}{\mathrm{P}^2_{11}\Upsilon_1} - \\ \arctan\frac{\mathrm{X}_2\mathrm{Z}_1}{\mathrm{P}^1_{22}\Upsilon_2} + \arctan\frac{\mathrm{X}_1\mathrm{Z}_1}{\mathrm{P}^1_{12}\Upsilon_2} + \arctan\frac{\mathrm{X}_2\mathrm{Z}_1}{\mathrm{P}^1_{21}\Upsilon_1} \\ - \arctan\frac{\mathrm{X}_1\mathrm{Z}_1}{\mathrm{P}^1_{11}\Upsilon_1} \end{aligned}$$

$$\begin{aligned} F_5 = \arctan\frac{\Upsilon_2\mathrm{Z}_2}{\mathrm{P}^2_{22}\mathrm{X}_2} - \arctan\frac{\Upsilon_2\mathrm{Z}_2}{\mathrm{P}^2_{12}\mathrm{X}_1} - \arctan\frac{\Upsilon_1\mathrm{Z}_2}{\mathrm{P}^2_{21}\mathrm{X}_2} + \arctan\frac{\Upsilon_1\mathrm{Z}_2}{\mathrm{P}^2_{11}\mathrm{X}_1} - \\ \arctan\frac{\Upsilon_2\mathrm{Z}_1}{\mathrm{P}^1_{22}\mathrm{X}_2} + \arctan\frac{\Upsilon_2\mathrm{Z}_1}{\mathrm{P}^1_{12}\mathrm{X}_1} + \arctan\frac{\Upsilon_1\mathrm{Z}_1}{\mathrm{P}^1_{21}\mathrm{X}_2} \\ - \arctan\frac{\Upsilon_1\mathrm{Z}_1}{\mathrm{P}^1_{11}\mathrm{X}_1}. \end{aligned}$$

Hence calculating (44) we can use (42) to calculate $X$, $Y$ and $R^2$ once for all slices. We note that minor computational savings may be made by calculating for example $R_1 + \mathrm{X}_1$ within the calculations for $\tilde{h}$ but these are not calculations that can be made independent of the given slice. It may appear also that one could calculate ratios $\mathrm{X}/\Upsilon$, with modification of the calculations for $F_4$ and $F_5$, but the stable calculation of the arctan requires the ratios as given. Otherwise sign changes in the numerator or denominator passed to arctan can lead to changes in the obtained angle. In MATLAB we use `atan2` rather than `atan` for improved stability in the calculation of the angle. The details of the use and application of the `magnetic` problem in the context of the forward algorithms for BTTB matrices are provided in Algorithms 3–4 with the `magnetic` function in Algorithm 6.

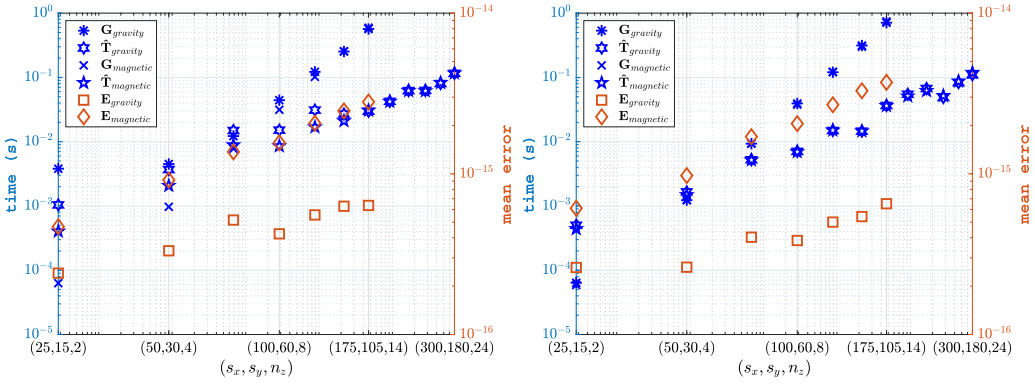
[2] We assume that the total field is measured in nano Teslas; introducing a scaling factor $10^9$ in the definitions.

[3] Note that constants $G_i$ are unrelated to the sensitivity matrices.




(a) $p = 0$ (b) $p = 0.05$

**Fig. 6.** Mean running times over 100 trials for forward multiplication using $G_{\text{gravity}}$, $G_{\text{magnetic}}$, $T_{\text{gravity}}$, and $T_{\text{magnetic}}$ (left $y$-axis) and mean errors over 100 trials $E_{\text{gravity}}$ and $E_{\text{magnetic}}$ for forward multiplication using $G_{\text{gravity}}$ versus $T_{\text{gravity}}$, and $G_{\text{magnetic}}$ versus $T_{\text{magnetic}}$ respectively (right $y$-axis) are shown for 0% padding in Fig. 6(a), and 5% padding in Fig. 6(b).




(a) $p = 0$ (b) $p = 0.05$

**Fig. 7.** Mean running times over 100 trials for transpose multiplication using $G_{\text{gravity}}$, $G_{\text{magnetic}}$, $T_{\text{gravity}}$, and $T_{\text{magnetic}}$ (left $y$-axis) and mean errors over 100 trials $E_{\text{gravity}}$ and $E_{\text{magnetic}}$ for transpose multiplication using $G_{\text{gravity}}$ versus $T_{\text{gravity}}$, and $G_{\text{magnetic}}$ versus $T_{\text{magnetic}}$ respectively (right $y$-axis) are shown for 0% padding in Fig. 7(a), and 5% padding in Fig. 7(b).

**Table 1**
Dimensions of the volume used in the experiments labeled as problems 1 to 12 corresponding to scaling each dimension in $(25, 15, 2)$ by the problem number (Prob.) and increasing $m$ by a factor 8 for each row. In the last two columns the memory requirements, in Gigabytes (GB), calculated in MATLAB, where the entries for rows 7 to 12 are the estimates given by MATLAB using try zeros(m,n), which gives an exception for matrices that are too large for storage in the given environment and reports the estimated requirements in GB to just one decimal place.

| Prob. | $(s_x, s_y, n_z)$ | $m$ | $n$ | $n$ $(p = 0.05\%)$ | GB $G$ | GB $\hat{T}^{\text{circ}}$ |
|---|---|---|---|---|---|---|
| 1 | (25, 15, 2) | 375 | 750 | 918 | .000225 | .000005 |
| 2 | (50, 30, 4) | 1500 | 6000 | 7616 | .007200 | .000037 |
| 3 | (75, 45, 6) | 3375 | 20 250 | 24 402 | .054675 | .000127 |
| 4 | (100, 60, 8) | 6000 | 48 000 | 58 080 | .230400 | .000303 |
| 5 | (125, 75, 10) | 9375 | 93 750 | 113 710 | .703125 | .000594 |
| 6 | (150, 90, 12) | 13 500 | 162 000 | 199 200 | 17.4960 | .001028 |
| 7 | (175, 105, 14) | 18 375 | 257 250 | 310 730 | 35.2 | .001634 |
| 8 | (200, 120, 16) | 24 000 | 384 000 | 464 640 | 68.7 | .002441 |
| 9 | (225, 135, 18) | 30 375 | 546 750 | 662 450 | 123.7 | .003478 |
| 10 | (250, 150, 20) | 37 500 | 750 000 | 916 320 | 209.5 | .004774 |
| 11 | (275, 165, 22) | 45 375 | 998 250 | 1 206 500 | 337.5 | .006358 |
| 12 | (300, 180, 24) | 54 000 | 1 296 000 | 1 568 200 | 521.4 | .008258 |

## 5. Numerical validation

We now validate the fast and efficient methods for generating both the symmetric and unsymmetric kernels relating to `gravity` and `magnetic` problems. We compare the computational cost of direct calculation of the entries of the matrix $G$ that are required for matrix multiplications, with the entries that are required for the transform implementation of the multiplications. We also compare the storage requirements for these matrices. Thus, we compare Algorithms 1 and 2 with all entries calculated using Algorithm 5, and Algorithms 3 and 4 with all entries calculated using Algorithm 6, for the symmetric `gravity`, and unsymmetric `magnetic` kernels, respectively. The 12 problem sizes considered are detailed in Table 1. They are generated by taking $(s_x, s_y, n_z) = (25, 15, 2)$, and then scaling each dimension by 1 to 12 for the test cases. We compare the cases with $p = 0\%$ and $p = 5\%$ padding across $x$ and $y$ dimensions, rounded to the nearest integer. Thus, $m = s_x s_y$, and $n = \lfloor (1+p)s_x \rfloor \lfloor (1+p)s_y \rfloor n_z$. All computations use MATLAB release 2019b implemented on a desktop computer with an Intel(R) Xeon (R) Gold 6138 processor (2.00 GHz) and 256 GB RAM.

First, note that the last two columns of Table 1 report the estimated memory requirement to store the arrays $G$ and $\hat{T}^{\text{circ}}$, which is independent of whether this is for the `gravity` or the `magnetic` problem. These are the results without padding, but the difference between the padded and unpadded case is insignificant in comparison to the memory requirements for each of these arrays. For the problem of size $m \times n$, matrix $G$ has $mn$ entries, corresponding to $8mn$ bytes and complex array $\hat{T}^{\text{circ}}$ uses approximately $8m$ entries for each depth layer, for a total of $8mn_z = 8n$ entries or $64n$ bytes, here using that one floating point number uses 8 bytes and noting that 1 byte is $10^{-9}$ GB.

**Table A.2**
Parameters and variables in the codes. The parameters are defined in Table A.3.

| | |
|---|---|
| prob_params | $s_x, s_y, n_z, p_{x_L}, p_{x_R}, p_{y_L}, p_{y_R}, n_x, n_y, m, n, n_r, p_x, p_y$ |
| $gsx$, $gsy$, $gsz$ | Grid sizes $\Delta_x$, $\Delta_y$ and $\Delta_z$ |
| That | That.forward = $\hat{T}^{\text{circ}}$, That.transpose = $\hat{\tilde{T}}^{\text{circ}}$ |
| z_blocks | Depth coordinates, increasing, $z_r$ |
| $D$ | Declination of geomagnetic field and magnetization vector |
| $I$ | Inclination of geomagnetic field and magnetization vector |
| $F$ | Intensity of the geomagnetic field in nT ($10^{-9}$ F in T) |
| $H = \frac{10^{-9}F}{4\pi}$ | Magnetic field intensity (A/m) in SI units |
| $\tilde{H} = 10^9 H = \frac{F}{4\pi}$ | Assumes the field is measured in nT |

**Algorithm 1:** $G$ = sym_BTTB($gsx, gsy, z\_blocks$, prob_params)
Entries of padded symBTTB matrix. Function gravity.

**Input**: See Table A.2 for details;
$gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
prob_params required parameters
**Output**: symBTTB real matrix $G$ of size $m \times n$.

Extract parameters from prob_params ;
Initialize zero arrays: $G$, $Gr$, $Grq$;
Sizes: $nX = s_x + \max(p_{x_L}, p_{x_R})$, $nY = s_y + \max(p_{y_L}, p_{y_R})$;
Form distance arrays $X$ and $Y$ according to (20);
Form $X2 = X^2$, $Y2 = Y^2$, $XY = X(:) * Y$ and $R = X2(:) + Y2$;
**for** $r = 1 : n_z$ **do**
  Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
  Calculate slice response at first station:
  $g$ = gravity($z_1, z_2, X, Y, XY, R$);
  **for** $q = 1 : nY$ **do**
    Extract $\mathbf{c}_q$, $\mathbf{r}_q$ from $g$ : use (15), (16);
    Generate: $Grq$ = toeplitz($\mathbf{c}_q, \mathbf{r}_q$): use (14) ;
  **end**
  **for** $j = [p_{y_L} + 1 : -1 : 2, 1 : s_y + p_{y_R}]$ **do**
    Build first row of $Gr$ using (19);
  **end**
  **for** $j = 2 : s_y$ **do**
    Build $j^{\text{th}}$ row of $Gr$ using (17) and (18);
  **end**
  Assign: $Gr$ to $r^{\text{th}}$ block of $G$;
**end**

**Algorithm 2:** That = sym_BTTBFFT($gsx, gsy, z\_blocks$, prob_params)
Transform of padded symBTTB matrix. Function gravity.

**Input**: See Table A.2 for details;
$gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
prob_params required parameters
**Output**: Array That

Extract parameters from prob_params ;
Initialize zero arrays: $T$ and That for $T^{\text{circ}}$ and $\hat{T}^{\text{circ}}$;
Sizes: $nX = s_x + \max(p_{x_L}, p_{x_R})$, $nY = s_y + \max(p_{y_L}, p_{y_R})$;
Form distance arrays $X$ and $Y$ according to (20);
Form $X2 = X^2$, $Y2 = Y^2$, $XY = X(:) * Y$ and $R = X2(:) + Y2$;
**for** $r = 1 : n_z$ **do**
  Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
  Calculate slice response at first station:
  $g$ = gravity($z_1, z_2, X, Y, XY, R$);
  **for** $j = [1 + p_{y_L} : s_y + p_{y_L}.s_y + p_{y_R} : -1 : 21 : p_{y_L}]$ **do**
    Extract $\mathbf{r}_j$ from $g$ : use (16);
    Augment column of $T$, use (38) ;
  **end**
  Take FFT of $T$: That$(:, :, r)$ = fft2($T$) ;
**end**

In the results, we reference the kernels generated by Algorithms 1, 3, 2, and 4 as $G_{\texttt{gravity}}$, $G_{\texttt{magnetic}}$, $T_{\texttt{gravity}}$, and $T_{\texttt{magnetic}}$ respectively. These values are plotted on a "log–log" scale in Fig. 5, without and with padding in Figs. 5(a) and 5(b), respectively. The problem sizes are given as relevant triples on the $x$-axis. The problem cases from 8 to 12 for the direct calculation of $G$ are too large to fit in memory on the given computer. It can be seen that the generation of $G$ is effectively independent of the gravity or magnetic kernels; $G_{\texttt{gravity}}$, $G_{\texttt{magnetic}}$ are comparable. But the requirement to calculate extra entries for the unsymmetric magnetic kernel is also seen; $T_{\texttt{gravity}} < T_{\texttt{magnetic}}$. On the other hand, the significant savings in generating just the transform matrices, as indicated by timings $T_{\texttt{gravity}}$, and $T_{\texttt{magnetic}}$, as compared to $G_{\texttt{gravity}}$, and $G_{\texttt{magnetic}}$ is evident. There is a considerable computational advantage to the use of the transform for calculating the required components that are needed for evaluating matrix–vector products for these structured kernel matrices.

Of greater significance is the comparison of the computational cost of direct matrix multiplications, $\mathbf{b} = G\mathbf{u}$ and $\mathbf{d} = G^T\mathbf{v}$, as compared with the transform implementations for these products, using Algorithm 7. We consistently partition $\mathbf{u} \in \mathcal{R}^{n_x n_y n_z}$ into $n_z$ blocks, $\mathbf{u}_r \in \mathcal{R}^{n_x n_y}$, $1 \le r \le n_z$. Then,

$$G\mathbf{u} = \sum_{r=1}^{n_z} G^{(r)}\mathbf{u}_r, \text{ and } \qquad G^T\mathbf{v} = \begin{bmatrix} \left(G^{(1)}\right)^T\mathbf{v} \\ \left(G^{(2)}\right)^T\mathbf{v} \\ \vdots \\ \left(G^{(n_z)}\right)^T\mathbf{v} \end{bmatrix}, \tag{22}$$

**Table A.3**
Notation adopted in the discussion.

| | | | |
|---|---|---|---|
| $s_x$ | # true stations in $x$ | $s_{ij} = (a_{ij}, b_{ij})$ | Station location |
| $s_y$ | # true stations in $y$ | $m = s_x s_y$ | # measurements |
| $n_x, n_y, n_z$ | # coordinate blocks in $x$, $y$, $z$ | $\Delta_x, \Delta_y, \Delta_z$ | Grid sizes in $x$, $y$, $z$ |
| $p_{x_L}, p_{x_R}, p_x$ | Left, right, total padding: $x$ | $n_x$ | $n_x = s_x + p_x$ |
| $p_{y_L}, p_{y_R}, p_y$ | Left, right,total, padding: $y$ | $n_y$ | $n_y = s_y + p_y$ |
| $x_p$ | $x_p = (p - 1 - p_{x_L})\Delta_x$, $1 \le p \le n_x + 1$ | $n = n_x n_y n_z$ | Volume Dimension |
| $y_q$ | $y_q = (q - 1 - p_{y_L})\Delta_y$, $1 \le q \le n_y + 1$ | $n_r = n_x n_y$ | Layer Dimension |
| $z_r$ | $z_r = (r-1)\Delta_z$, $1 \le r \le n_z + 1$ | $c_{pqr}$ | Prism $pqr$ in $xyz$ |
| $d$, $h$, $\zeta$ | Forward Model see (1) | $\tilde{h}(s_{ij})_{pqr}$ | Projection $c_{pqr}$ to $s_{ij}$ |
| $G \in \mathcal{R}^{m \times n}$ | $(G^{(r)})_{k\ell} = \tilde{h}(s_{ij})_{pqr}$ See (5) | $G^{(r)} \in \mathcal{R}^{m \times n_r}$ | Depth $r$ Contribution |
| $G_q^{(r)} \in \mathcal{R}^{s_x \times n_x}$ | $G_q^{(r)} = G_{1q}^{(r)}$, $1 \le q \le n_y$ | $\bar{G}_j^{(r)} \in \mathcal{R}^{s_x \times n_x}$ | $\bar{G}_j^{(r)} = \bar{G}_{j1}^{(r)}$, $1 \le j \le s_y$ |
| $\mathbf{c}_q$, $\mathbf{r}_q$ | $G_q^{(r)}$ = toeplitz($\mathbf{c}_q, \mathbf{r}_q$) | $\bar{\mathbf{c}}_j$, $\bar{\mathbf{r}}_j$ | $\bar{G}_j^{(r)}$ = toeplitz($\bar{\mathbf{c}}_j, \bar{\mathbf{r}}_j$) |
| BTTB | Block Toeplitz–Toeplitz blocks | symBTTB | Symmetric BTTB |
| BCCB | Block Circulant–Circulant blocks | $J_m$ Definition 2 | Exchange matrix |
| $\mathbf{c}_q^{\text{ext}}$, $\mathbf{r}_q^{\text{ext}}$ | Defining $(G^{(r)})^{\text{circ}}$ | $\bar{\mathbf{c}}_j^{\text{ext}}$, $\bar{\mathbf{r}}_j^{\text{ext}}$ | Defining $(\bar{G}^{(r)})^{\text{circ}}$ |
| $T^{\text{circ}}$ | Components of BCCB | $\hat{T}^{\text{circ}}$ | fft2($T^{\text{circ}}$) : 2DFFT |

**Algorithm 3:** $G = \texttt{BTTB}(gsx, gsy, z\_blocks, \texttt{prob\_params}, D, I, H)$
Entries of padded BTTB matrix, Fig. 3. Function `magnetic`.

**Input**: See Table A.2 for details;
$gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
`prob_params` required parameters
**Output**: BTTB real matrix $G$ of size $m \times n$.

Extract parameters from `prob_params` ;
Calculate constants $g_i = \tilde{H}G_i$ for (44), Bhaskara Rao and Ramesh Babu (1991, (3));
Initialize zero arrays: $G$, $Gr$ and row and column cell arrays, Fig. 3;
Sizes: $nX = s_x + \max(p_{x_L}, p_{x_R})$, $nY = s_y + \max(p_{y_L}, p_{y_R})$;
Form distance arrays $X$ and $Y$ according to (30);
Form $X2 = X^2$, $Y2 = Y^2$, and $R = X2(:) + Y2$;
**for** $r = 1 : n_z$ **do**
  Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
  Calculate $grow\{1\} = \tilde{h}(11)_{pq}$, $1 \le p \le nX$, $1 \le q \le nY$;
  **for** $j = 2 : s_y + p_{y_L}$ **do**
    Calculate $grow\{j\} = \tilde{h}(1j)_{p1}$, $1 \le p \le nX$;
  **end**
  Calculate : $gcol\{1\} = \tilde{h}(ij)_{11}$, $1 \le i \le nX$, $1 \le j \le nY$;
  **for** $q = 2 : s_y + p_{y_R}$ **do**
    Calculate : $gcol\{q\} = \tilde{h}(i1)_{1q}$, $1 \le i \le nX$;
  **end**
  **for** $j = p_{y_L} + 1 : -1 : 2$ **do**
    Generate $Gjr$: using $gcol\{1\}$ and $grow\{j\}$, (28) for $R$ in (29);
  **end**
  **for** $q = 1 : s_y + p_{y_R}$ **do**
    Generate $Gqr$: using $gcol\{q\}$ and $grow\{1\}$, (27) for $R$ in (29);
  **end**
  **for** $j = p_{y_L} + 2 : -1 : 2, 1 : s_y + p_{y_L}$ **do**
    Generate $Grj$: using $gcol\{1\}$ and $grow\{j\}$, (28) for $C$ in (29);
  **end**
  Build $Gr$ in (29) using $C$ and $R$;
  Assign: $Gr$ to $r^{\text{th}}$ block of $G$;
**end**

**Algorithm 4:** $\text{That} = \texttt{BTTBFFT}(gsx, gsy, z\_blocks, \texttt{prob\_params}, D, I, H)$
Transform of padded BTTB matrix, Fig. 3. Function `magnetic`.

**Input**: See Table A.2 for details;
$gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
`prob_params` required parameters;
$D$, $I$, $H$ declination, inclination and intensity of magnetization
**Output**: Array That

Extract parameters from `prob_params` ;
Calculate constants $g_i = \tilde{H}G_i$ for (44), Bhaskara Rao and Ramesh Babu (1991, (3));
Initialize zero arrays: $T$ and That for $T^{\text{circ}}$ and $\hat{T}^{\text{circ}}$;
Initialize zero arrays for and row and column cell arrays, see Fig. 3;
Sizes: $nX = s_x + \max(p_{x_L}, p_{x_R})$, $nY = s_y + \max(p_{y_L}, p_{y_R})$;
Form distance arrays $X$ and $Y$ according to (30);
Form $X2 = X^2$, $Y2 = Y^2$, and $R = X2(:) + Y2$;
**for** $r = 1 : n_z$ **do**
  Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
  Calculate $grow\{1\} = \tilde{h}(11)_{pq}$, $1 \le p \le nX$, $1 \le q \le nY$;
  **for** $j = 2 : s_y + p_{y_L}$ **do**
    Calculate $grow\{j\} = \tilde{h}(1j)_{p1}$, $1 \le p \le nX$;
  **end**
  Calculate : $gcol\{1\} = \tilde{h}(ij)_{11}$, $1 \le i \le nX$, $1 \le j \le nY$;
  **for** $q = 2 : s_y + p_{y_R}$ **do**
    Calculate : $gcol\{q\} = \tilde{h}(i1)_{1q}$, $1 \le i \le nX$;
  **end**
  **for** $j = p_{y_L} + 1 : p_{y_L} + s_y$ **do**
    Augment column of $T$, $gcol\{1\}$ and $grow\{j\}$, (28) with (39) ;
  **end**
  **for** $q = s_y + p_{y_R} : -1 : 2$ **do**
    Augment column of $T$, $gcol\{q\}$ and $grow\{1\}$, (27) with (39) ;
  **end**
  **for** $j = 1 : p_{y_L}$ **do**
    Augment column of $T$, $gcol\{1\}$ and $grow\{j\}$, (28) with (39) ;
  **end**
  Take FFT of $T$: $\text{That}(:, :, r) = \texttt{fft2}(T)$;
**end**

where $\mathbf{v} \in \mathcal{R}^{s_x s_y}$. 100 copies of vectors $\mathbf{u} \in \mathcal{R}^n$ and $\mathbf{v} \in \mathcal{R}^m$ are randomly generated and the mean times for calculating the products over all 100 trials, for each problem size, are recorded. We also record the differences over all trials in the generation of **b** and **d** obtained directly for $G_{\texttt{gravity}}$ and $G_{\texttt{magnetic}}$ and by Algorithm 7 for $T_{\texttt{gravity}}$ and $T_{\texttt{magnetic}}$. Then, $E_{\texttt{gravity}}$ and $E_{\texttt{magnetic}}$ are the mean values of the relative 2-norm of the difference between the results produced by $G_{\texttt{gravity}}$ versus $T_{\texttt{gravity}}$, and for $G_{\texttt{magnetic}}$ versus $T_{\texttt{magnetic}}$, respectively, for both forward and transpose operations. The results are illustrated in Figs. 6 and 7 for the generation of $G\mathbf{u}$ and $G^T\mathbf{v}$, respectively. In each case the timing is reported on the left $y$-axis and the error on the right $y$-axis. Again all plots are on the "log–log" scale, and Figs. 6(a) and 7(a), and 6(b) and 7(b), are without and with padding, respectively. Figs. 6 and 7 show significant reductions in mean running time when implemented without the direct calculation of the matrices. Moreover, the results are comparable, $E_{\texttt{gravity}} \lesssim 10\epsilon$ for both forward and transpose operations, and $E_{\texttt{magnetic}} \lesssim 10^2\epsilon$, where $\epsilon$ is the machine accuracy. Thus, in all cases, $T_{\texttt{gravity}}$ and $T_{\texttt{magnetic}}$ show a significant reduction in mean running time for large problems, and allow much larger systems to be represented. Indeed, the largest test case for $T_{\texttt{gravity}}$ and $T_{\texttt{magnetic}}$ is by no means a limiting factor, and it is possible to represent much larger kernels.

**Remark 1** (MATLAB `fft2`)**.** The MATLAB `fft2` function determines an optimal algorithm for a given problem size. On the first call for a given problem size, `fft2` uses the function `fftw` to determine optimal parameters for the Fourier transform. Thus, the first time `fft2` is called generally takes longer than subsequent instances. We mitigate this effect by first removing the variable `dwisdom` within `fftw`, and then setting the planner within `fftw` to `exhaustive`. The obtained `dwisdom` is saved. This process is repeated for generating $\hat{T}$, forward multiplication, and transpose multiplication. Then for each trial, the appropriate stored values for `dwisdom` are loaded before each use of `fft2`. Hence the results are not contaminated by artificially high costs of the first run of `fftw` for each problem case.

## 6. Data availability and software package

The software consists of the main functions to calculate the BTTB and symBTTB matrices, with padding, and the circulant matrices $T^{\text{circ}}$ that are needed for the 2DFFT. Also provided is a simple script to test the algorithms using the `gravity` and `magnetic` kernels. All the algorithms are described in Appendix B and the software is open source and available at https://github.com/renautra/FastBTTB, and

**Algorithm 5:** $g = \texttt{gravity}(z_1, z_2, X, Y, XY, R)$
Entries of sensitivity matrix $G$ for the `gravity` problem.

**Input**: Depth coordinates $z_1$ and $z_2$ for the slice;
$X$: Distances of $x-$coordinates from station 1 size nx;
$Y$: Distances of $y-$coordinates from station 1 size ny;
$XY$: the product $X(:).*Y$ which is a matrix of size $(nx+1)\times(ny+1)$;
$R$: the matrix of size $(nx+1)\times(ny+1)$ of entries $X(:).\hat{}2$ and $Y.\hat{}2$;
**Output**: Response vector $g$ of length $(nx+1)(ny+1)$;

$[nx, ny] = \text{size}(R)$;
$R_1 = \text{sqrt}(R + z_1^2)$;
$R_2 = \text{sqrt}(R + z_2^2)$;
$CMX = (\log((X(:) + R_1)./(X(:) + R_2))).*Y$;
$CMY = (\log((Y + R_1)./(Y + R_2))).*X(:)$;
$CM5Z = \text{atan2}(XY, R_1 z_1)z_1$;
$CM6Z = \text{atan2}(XY, R_2 z_2)z_2$;
$CM56 = CM5Z - CM6Z$;
$CM = (CM56 - CMY - CMX)\gamma$;
$g = -(CM(1:nx-1, 1:ny-1) - CM(1:nx-1, 2:ny) - CM(2:nx, 1:ny-1) + CM(2:nx, 2:ny))$;

**Algorithm 6:** $g = \text{magnetic}(z_1, z_2, X, Y, R, gc))$
Entries of sensitivity matrix $G$ for the `magnetic` problem.

**Input**: Depth coordinates $z_1$ and $z_2$ for the slice;
$X$: Distances of $x-$coordinates from station ;
$Y$: Distances of $y-$coordinates from station ;
$R$: Matrix of entries $X(:)^2$ and $Y^2$;
$gc$ vector of constants, Bhaskara Rao and Ramesh Babu (1991, 3);
**Output**: Response vector $g$ of length $(\ell+1)(k+1)$;

$\ell = \text{length}(X) - 1; k = \text{length}(Y) - 1$;
$R_1 = \text{sqrt}(R + z_1^2)$;
$R_2 = \text{sqrt}(R + z_2^2)$;
$F_1 = ((R_2(1:\ell, 1:k) + X(1:\ell))./(R_1(1:\ell, 1:k) + X(1:\ell))).*((R_1(2:\ell+1, 1:k) + X(2:\ell+1))./(R_2(2:\ell+1, 1:k) + X(2:\ell+1))).*((R_1(1:\ell+1, 2:k+1) + X(1:\ell))./(R_2(1:\ell+1, 2:k+1) + X(1:\ell))).*((R_2(2:\ell+1, 2:k+1) + X(2:\ell+1))./(R_1(2:\ell+1, 2:k+1) + X(2:\ell+1)))$;
$F_2 = ((R_2(1:\ell, 1:k) + Y(1:k))./(R_1(1:\ell, 1:k) + Y(1:k))).*((R_1(2:\ell+1, 1:k) + Y(1:k))./(R_2(2:\ell+1, 1:k) + Y(1:k))).*((R_1(1:\ell+1, 2:k+1) + Y(2:k+1))./(R_2(1:\ell+1, 2:k+1) + Y(2:k+1))).*((R_2(2:\ell+1, 2:k+1) + Y(2:k+1))./(R_1(2:\ell+1, 2:k+1) + Y(2:k+1)))$;
$F_3 = ((R_2(1:\ell, 1:k) + z2)./(R_1(1:\ell, 1:k) + z1)).*((R_1(2:\ell+1, 1:k) + z1)./(R_2(2:\ell+1, 1:k) + z2)).*((R_1(1:\ell+1, 2:k+1) + z1)./(R_2(1:\ell+1, 2:k+1) + z2)).*((R_2(2:\ell+1, 2:k+1) + z2)./(R_1(2:\ell+1, 2:k+1) + z1))$;
$F_4 = \text{atan2}(X(2:\ell+1)z_2, R_2(2:\ell+1, 2:k+1).*Y(2:k+1)) - \text{atan2}(X(1:\ell)z_2, R_2(1:\ell+1, 2:k+1).*Y(2:k+1)) - \text{atan2}(X(2:\ell+1)z_2, R_2(2:\ell+1, 1:k).*Y(1:k)) + \text{atan2}(X(1:\ell)z_2, R_2(1:\ell, 1:k).*Y(1:k)) - \text{atan2}(X(2:\ell+1)z_1, R_1(2:\ell+1, 2:k+1).*Y(2:k+1)) + \text{atan2}(X(1:\ell)z_1, R_1(1:\ell+1, 2:k+1).*Y(2:k+1)) + \text{atan2}(X(2:\ell+1)z_1, R_1(2:\ell+1, 1:k).*Y(1:k)) - \text{atan2}(X(1:\ell)z_1, R_1(1:\ell, 1:k).*Y(1:k))$;
$F_5 = \text{atan2}(Y(2:k+1)z_2, R_2(2:\ell+1, 2:k+1).*X(2:\ell+1)) - \text{atan2}(Y(2:k+1)z_2, R_2(1:\ell+1, 2:k+1).*X(1:\ell)) - \text{atan2}(Y(1:k)z_2, R_2(2:\ell+1, 1:k).*X(2:\ell+1)) + \text{atan2}(Y(1:k)z_2, R_2(1:\ell, 1:k).*X(1:\ell)) - \text{atan2}(Y(2:k+1)z_1, R_1(2:\ell+1, 2:k+1).*X(2:\ell+1)) + \text{atan2}(Y(2:k+1)z_1, R_1(1:\ell+1, 2:k+1).*X(1:\ell)) + \text{atan2}(Y(1:k)z_1, R_1(2:\ell+1, 1:k).*X(2:\ell+1)) - \text{atan2}(Y(1:k)z_1, R_1(1:\ell, 1:k).*X(1:\ell))$;
$g = (gc(1)*\log(F_1) + gc(2)*\log(F_2) + gc(3)*\log(F_3) + gc(4)*F_4 + gc(5)*F_5)$;
$g = g(:)$;

described at https://math.la.asu.edu/~rosie/research/bttb.html. Provided are the scripts that are used to generate the results presented in the paper. The variables used in the codes are described in Tables A.2 and A.3. The `TestingScript.m` is easily modified to generate new examples and can be tested within different hardware configurations and versions of MATLAB. A safety test for memory usage in generating large scale examples is provided at the initialization of each problem size, so that problems too large to fit in memory will not be used in generating the matrix $G$ directly. The presented implementation assumes uniform grid sizes in the $x$ and $y$ dimensions, i.e. fixed $\Delta_x$ and $\Delta_y$ throughout the domain, but using depth layers of different heights, different $\Delta_z$, is easily implemented by appropriately picking the input coordinate vector *z_blocks*.

## 7. Conclusions and future work

We have provided a description of the generation of efficient codes for implementing forward and transpose operations with BTTB matrices. These are used in geophysical forward modeling when the kernels are of convolution type and generate matrices with the required structure, which occurs when the observation points are on a uniform grid. Efficient generation of matrix operations with minimal storage makes it feasible to perform large three-dimensional modeling with these kernels. A novelty of this work, beyond existing descriptions in the literature, is the development of the approach for the `magnetic` kernel and the inclusion of padding in the coordinate volume. The approach for finding operations $G^T\mathbf{m}$ explicitly given knowledge of the BTTB structure of $G$ and its BCCB embedding is provided. It should be noted that while the algorithm requires that the observation points are on a uniform grid, this is not necessarily a limiting factor of the approach, since interpolation from non-uniform to uniform points is possible. Moreover, while it is assumed that the volume is discretized so that the observation points occur at one point on the surface for each prism of the coordinate domain, there is no requirement that the prisms are all uniform in the depth dimension, and it is feasible therefore to implement with different resolutions in the depth dimension of the subsurface volume. Thus, the developed software can be integrated into an inverse modeling problem, in which given data $\mathbf{d}$, model parameters $\mathbf{m}$ are desired. This is planned for future work.

## CRediT authorship contribution statement

**Jarom D. Hogue:** Wrote the codes for implementing the BTTB operations using transforms, Wrote the section describing the use of transforms. **Rosemary Anne Renaut:** Developed the gravity and magnetic codes, Efficient derivation of the entries of the matrices. Drafted the paper. **Saeed Vatankhah:** Provided geophysics expertise for all components of the manuscript, Provided all the initial functions for magnetic and gravity kernels.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

Rosemary Renaut acknowledges the support of NSF grant DMS 1913136: "Approximate Singular Value Expansions and Solutions of Ill-Posed Problems".

**Algorithm 7:** $\mathbf{b} = \texttt{mult\_BTTB}(\text{That}, \mathbf{x}, t, \texttt{prob\_params})$

This algorithm calculates the forward and transpose multiplication, $G\mathbf{x}$, or $G^T\mathbf{x}$ as described in Section 3 using the embedding of the BTTB matrix in a BCCB matrix and the 2DFFT. The transform of (38) or (39) for symBTTB and BTTB, respectively, is precomputed and provided in That. See Table A.2 for definitions of input parameters.

**Input**: That for $\hat{T}$: see Table A.2;
$\mathbf{x}$ : vector for forward or transpose multiplication;
$t$ : 1 or 2 for forward or transpose multiplication, respectively;
`prob_params` : required parameters see Table A.2
**Output**: vector: $\mathbf{b}$ of size $m$ or $n$, for $t = 1, 2$, respectively.

```
Extract parameters from prob_params ;
Initialize zero array for b and W;
if t == 2 then
    Initialize W according to (41) ;
    Take transform of W: Ŵ = fft2(W);
end
for j = 1 : n_z % For all layers of domain do
    switch t do
        case 1
            Initialize W according to (40);
            Take transform of W: Ŵ = fft2(W);
            Form convolution (34):
            W = real(ifft2(That(:, :, j)· ∗ Ŵ));
            Extract and accumulate top left block:
            b = b + reshape(W(1 : s_x, 1 : s_y), m, 1);
        end
        case 2
            Form convolution (34):
            Z = real(ifft2(conj(That(:, :, j)· ∗ Ŵ)));
            Extract top left block and assign to output:
            b((j − 1)n_r + 1 : jn_r) = reshape(Z(1 : n_x, 1 : n_y), n_r, 1);
        end
    endsw
end
```

## Appendix A. Notation and parameter definitions

The notation and parameters is detailed in two tables. The first is for the variables used for the codes, and the second for the notation in the paper.

## Appendix B. Algorithms

An overview of the required algorithms as described in Sections 2–3 are provided in Algorithms 1–2 using the `gravity` function in Algorithm 5 and in Algorithms 3–4 with the `magnetic` function in Algorithm 6. The convolution multiplication is provided in Algorithm 7.

## References


Bhaskara Rao, D., Ramesh Babu, N., 1991. A rapid method for three-dimensional modeling of magnetic anomalies. Geophysics 56 (11), 1729–1737.

Boulanger, Olivier, Chouteau, Michel, 2001. Constraints in 3D gravity inversion. Geophys. Prospect. (ISSN: 1365-2478) 49 (2), 265–280. http://dx.doi.org/10.1046/j.1365-2478.2001.00254.x.

Bruun, Christian Eske, Nielsen, Trine Brandt, 2007. Algorithms and Software for Large-Scale Geophysical Reconstructions (Master's thesis). Technical University of Denmark, DTU, DK-2800 Kgs., Lyngby, Denmark.

Chan, Raymond Hon-Fu, Jin, Xiao-Qing, 2007. An Introduction to Iterative Toeplitz Solvers. Society for Industrial and Applied Mathematics, http://dx.doi.org/10.1137/1.9780898718850, URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718850.

Chan, Raymond H., Ng, Michael K., 1996. Conjugate gradient methods for Toeplitz systems. SIAM Rev. (ISSN: 00361445) 38 (3), 427–482, URL http://www.jstor.org/stable/2132496.

Chen, Longwei, Liu, Lanbo, 2018. Fast and accurate forward modelling of gravity field using prismatic grids. Geophys. J. Int. (ISSN: 0956-540X) 216 (2), 1062–1071. http://dx.doi.org/10.1093/gji/ggy480.

Geng, Meixia, Kim Welford, J., Farquharson, Colin G., Hu, Xiangyun, 2019. Gravity modeling for crustal-scale models of rifted continental margins using a constrained 3D inversion method. Geophysics 84 (4), G25–G39. http://dx.doi.org/10.1190/geo2018-0134.1.

Gómez-Ortiz, David, Agarwal, Bhrigu N.P., 2005. 3DINVER.M: a MATLAB program to invert the gravity anomaly over a 3D horizontal density interface by Parker–Oldenburg's algorithm. Comput. Geosci. (ISSN: 0098-3004) 31 (4), 513–520. http://dx.doi.org/10.1016/j.cageo.2004.11.004, URL http://www.sciencedirect.com/science/article/pii/S0098300404002262.

Haáz, István Béla, 1953. Relations between the potential of the attraction of the mass contained in a finite rectangular prism and its first and second derivatives. Geophys. Trans. II 7, 57–66.

Li, Kun, Chen, Long-Wei, Chen, Qing-Rui, Dai, Shi-Kun, Zhang, Qian-Jiang, Zhao, Dong-Dong, Ling, Jia-Xuan, 2018. Fast 3D forward modeling of the magnetic field and gradient tensor on an undulated surface. Appl. Geophys. (ISSN: 1993-0658) 15 (3), 500–512. http://dx.doi.org/10.1007/s11770-018-0690-9.

Pilkington, Mark, 1997. 3-D magnetic imaging using conjugate gradients. Geophysics (ISSN: 0016-8033) 62 (4), 1132–1142. http://dx.doi.org/10.1190/1.1444214.

Shin, Young Hong, Choi, Kwang Sun, Xu, Houze, 2006. Three-dimensional forward and inverse models for gravity fields based on the Fast Fourier Transform. Comput. Geosci. (ISSN: 0098-3004) 32 (6), 727–738. http://dx.doi.org/10.1016/j.cageo.2005.10.002, URL http://www.sciencedirect.com/science/article/pii/S0098300405002268.

Vogel, Curt, 2002. Computational Methods for Inverse Problems. Society for Industrial and Applied Mathematics, Philadelphia, http://dx.doi.org/10.1137/1.9780898717570, URL http://epubs.siam.org/doi/abs/10.1137/1.9780898717570.

Zhang, Yile, Wong, Yau Shu, 2015. BTTB-based numerical schemes for three-dimensional gravity field inversion. Geophys. J. Int. (ISSN: 0956-540X) 203 (1), 243–256. http://dx.doi.org/10.1093/gji/ggv301.

Zhao, Guangdong, Chen, Bo, Chen, Longwei, Liu, Jianxin, Ren, Zhengyong, 2018. High-accuracy 3D Fourier forward modeling of gravity field based on the Gauss-FFT technique. J. Appl. Geophys. (ISSN: 0926-9851) 150, 294–303. http://dx.doi.org/10.1016/j.jappgeo.2018.01.002, URL http://www.sciencedirect.com/science/article/pii/S0926985117301751.