

**A fast method for solving the
two-dimensional Helmholtz equation,
with Robbins boundary conditions.**

*Jef Hendrickx
Raf Vandebril
Marc Van Barel*

Report TW326, June 2001



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

A fast method for solving the two-dimensional Helmholtz equation, with Robbins boundary conditions.

Jef Hendrickx
Raf Vandebril
Marc Van Barel

Report TW 326, June 2001

Department of Computer Science, K.U.Leuven

Abstract

We present a fast direct method for solving the two-dimensional Helmholtz equation: on a rectangular grid $[0, a_1] \times [0, a_2]$ with Robbins boundary conditions.

Because we can solve the Helmholtz equation, with Neumann boundary conditions in a fast way using the discrete cosine transform, we can split the problem above into two smaller problems. One of these problems can be solved using the same techniques as in the Neumann-boundary case. The second, and the hardest problem of the two, can be solved using low displacement rank techniques.

When dividing $[0, a_1]$ into n_1 and $[0, a_2]$ into n_2 equal parts, the total complexity of the overall algorithm is $10 n_1 n_2 \log n_2 + O(n_1^2 + n_1 n_2)$, which gives us a fast algorithm.

Keywords : Helmholtz, Robbins boundary conditions, displacement rank, Neumann boundary conditions.

AMS(MOS) Classification : Primary : 35J05, Secondary : 65F05.

A fast method for solving the two-dimensional Helmholtz equation, with Robbins boundary conditions.*

Jef Hendrickx[†], Raf Vandebril[‡], Marc Van Barel[§]

27-03-2001

Abstract

We present a fast direct method for solving the two-dimensional Helmholtz equation:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \lambda \phi = f(x, y),$$

on a rectangular grid $[0, a_1] \times [0, a_2]$ with Robbins boundary conditions

$$\begin{aligned} \left(\frac{\partial \phi}{\partial x} - p_0 \phi \right) (0, y) &= \alpha_0(y), \\ \left(\frac{\partial \phi}{\partial x} - p_1 \phi \right) (a_1, y) &= \alpha_1(y), \\ \left(\frac{\partial \phi}{\partial y} - q_0 \phi \right) (x, 0) &= \beta_0(x), \\ \left(\frac{\partial \phi}{\partial y} - q_1 \phi \right) (x, a_2) &= \beta_1(x), \end{aligned}$$

where λ, p_0, p_1, q_0 and q_1 are constants.

*This research was partially supported by the Fund for Scientific Research–Flanders (FWO–V), project “SMA: Structured Matrices and their Applications” grant #G.0078.01, by the K.U.Leuven (Bijzonder Onderzoeksfonds), project “SLAP: Structured Linear Algebra Package,” grant #OT/00/16, by the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister’s Office for Science, Technology and Culture. The scientific responsibility rests with the authors

[†]E-mail: jef.hendrickx@prof.ehsal.be

[‡]E-mail: raf.vandebril@cs.kuleuven.ac.be

[§]E-mail: marc.vanbarel@cs.kuleuven.ac.be

Because we can solve the Helmholtz equation, with Neumann boundary conditions in a fast way using the discrete cosine transform, we can split the problem above into two smaller problems. One of these problems can be solved using the same techniques as in the Neumann-boundary case. The second, and the hardest problem of the two, can be solved using low displacement rank techniques.

When dividing $[0, a_1]$ into n_1 and $[0, a_2]$ into n_2 equal parts, the total complexity of the overall algorithm is $10n_1n_2 \log n_2 + O(n_1^2 + n_1n_2)$, which gives us a fast algorithm.

Keywords: Helmholtz, Robbins boundary conditions, displacement rank, Neumann boundary conditions.

1 The Neumann boundary problem

In this section we will briefly review a fast method to solve the Helmholtz equation with Neumann boundary conditions. This problem can easily be solved using a fast transformation. And as it will be shown, in the next section this is an important property in solving the Robbins boundary problem, in which we can twice use the techniques we are about to describe. In this point of view as mentioned in the abstract, the Robbins boundary problem can be split into two smaller problems of which one can be solved using the Neumann boundary problem. First we will describe the solution to this problem.

We take the Helmholtz equation,

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \lambda \phi = f(x, y), \quad (1)$$

with Neumann boundary conditions:

$$\frac{\partial \phi}{\partial y}(x, 0) = \beta_0(x), \quad \frac{\partial \phi}{\partial y}(x, a_2) = \beta_1(x), \quad 0 \leq x \leq a_1. \quad (2)$$

When using the classical five points difference scheme for $i = 0, 1, \dots, n_1$ and $j = 0, 1, \dots, n_2$, we get

$$u_{i,j} - \theta_1(u_{i+1,j} + u_{i-1,j}) - \theta_2(u_{i,j+1} + u_{i,j-1}) = -\delta f_{i,j} \quad (3)$$

with

$$h_1 = \frac{a_1}{n_1}, \quad h_2 = \frac{a_2}{n_2}, \quad \delta = \left(\frac{2}{h_1^2} + \frac{2}{h_2^2} - \lambda \right)^{-1}, \quad \theta_1 = \frac{\delta}{h_1^2}, \quad \theta_2 = \frac{\delta}{h_2^2} \quad (4)$$

and where $u_{i,j}$ is the approximated value of $\phi(x,y)$ in the grid point (x_i,y_j) and in the same way we define $f_{i,j}$ as the value of $f(x,y)$ in the grid point (x_i,y_j) . Approximating the boundary conditions with central differences we can substitute $u_{i,-1}$ and u_{i,n_2+1} by

$$u_{i,-1} = u_{i,1} - 2h_2\beta_{0,i}, \quad u_{i,n_2+1} = u_{i,n_2-1} + 2h_2\beta_{1,i}, \quad (5)$$

with $\beta_{k,i} = \beta_k(x_i)$. This gives us a system of equations $M_{N,N} \mathbf{u} = \mathbf{b}$ with unknowns $u_{i,j}$ for $i = 0, 1, \dots, n_1$ and $j = 0, 1, \dots, n_2$ where

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{n_2} \end{bmatrix}, \quad \mathbf{u}_j = \begin{bmatrix} u_{0,j} \\ u_{1,j} \\ \vdots \\ u_{n_1,j} \end{bmatrix}. \quad (6)$$

and $M_{N,N}$ is the following block matrix

$$M_{NN} = \begin{bmatrix} A & 2T & & & \\ T & A & T & & \\ & \ddots & \ddots & \ddots & \\ & & T & A & T \\ & & & 2T & A \end{bmatrix}, \quad (7)$$

where A and T are of the form: $T = -\theta_2 I$ and

$$A = \begin{bmatrix} 1 + 2h_1 p_0 \theta_1 & -2\theta_1 & & & \\ -\theta_1 & 1 & \ddots & & \\ & \ddots & \ddots & & -\theta_1 \\ & & & -2\theta_1 & 1 - 2h_1 p_1 \theta_1 \end{bmatrix}, \quad (8)$$

with θ_1 and θ_2 as in (4).

It is a well-known fact that a matrix like

$$B = \begin{bmatrix} a & 2t & & & \\ t & a & t & & \\ & \ddots & \ddots & \ddots & \\ & & t & a & t \\ & & & 2t & a \end{bmatrix}$$

of size $n + 1$ with a and t scalars, can be diagonalized in the following sense

$$B = C\Lambda C$$

with $C = (c_{i,j})$ and $\Lambda = \text{diag}\{\lambda_0, \dots, \lambda_n\}$, where

$$c_{i,j} = \begin{cases} \sqrt{2/n} 1/2, & \text{voor } j = 0, & i = 0, \dots, n, \\ \sqrt{2/n} \cos\left(\frac{ij\pi}{n}\right) & \text{voor } j = 1, \dots, n-1, & i = 0, \dots, n, \\ \sqrt{2/n} (-1)^i 1/2, & \text{voor } j = n, & i = 0, \dots, n, \end{cases}$$

and

$$\lambda_i = a + 2t \cos \frac{i\pi}{n}, \quad i = 0, \dots, n.$$

To prove this, write the matrix B as $aI + tX_{00}$ with X_{00} as in (17), and then use the formula (29). A more compact form for the matrix C is

$$C = \sqrt{\frac{2}{n}} \left[\varepsilon_j \cos \frac{ij\pi}{n} \right]_{i,j=0}^n,$$

with

$$\varepsilon_k = \begin{cases} 1/2 & \text{voor } k = 0, n \\ 1 & \text{voor } k = 1, 2, \dots, n-1 \end{cases}. \quad (9)$$

(When we explicitly want to denote the size of the matrix C we denote it as C_{n+1} .) The matrix C is the matrix of one of the versions of the discrete cosine transform (*DCT*), [1]. This matrix is not symmetric, nor persymmetric but satisfies $(C)^{-1} = C$.

In a similar way as in the scalar case it can be shown that the matrix M_{NN} can be factorised in the following way, where \otimes stands for the Kronecker product,

$$M_{NN} = (C \otimes I) \Lambda (C \otimes I), \quad (10)$$

with Λ a block diagonal matrix with blocks

$$\Lambda_j = A + 2T \cos \frac{j\pi}{n_2}, \quad j = 0, \dots, n_2. \quad (11)$$

We now present the algorithm for solving the Helmholtz equation with the Neumann boundary conditions. We use an operator called *Vec* which means that we make a long vector from a matrix by placing all the columns of the matrix one below the other.

algorithm 1 Solving $M_{N,N}\mathbf{u} = \mathbf{b}$ where $M_{N,N}$ is the block tridiagonal matrix of size $(n_1 + 1) \times (n_2 + 1)$, given by (7), with A and T of size $n_1 + 1$ as in (8). Let $\mathbf{b} = \text{vec}V$ with $V \in \mathbb{R}^{(n_1+1) \times (n_2+1)}$ (and the same for $\mathbf{z} = \text{vec}Z$, $\mathbf{y} = \text{vec}Y$, $\mathbf{u} = \text{vec}U$).

1. $Z = VC_{n_2+1}$
2. for $j = 0, \dots, n_2$:
solve the systems of equations $(A + 2T \cos \frac{j\pi}{n_2}) \mathbf{y}_j = \mathbf{z}_j$ for \mathbf{y}_j
3. $U = YC_{n_2+1}$

The transformations in the first and the final step are cosine transformations. Such a transformation of length n can be done using the Fast Fourier Transform (FFT) in $2.5n \log n$ flops (under some weak assumptions about the size n of the matrix [3, 4]), so we get a complexity of $5n_1n_2 \log n_2 + 8n_1n_2$, because A and T are tridiagonal.

2 The Robbins boundary problem

We are now ready to deduce a fast direct method for solving the Helmholtz equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \lambda \phi = f(x, y),$$

with Robbins boundary conditions

$$\begin{aligned} \left(\frac{\partial \phi}{\partial x} - p_0 \phi \right) (0, y) &= \alpha_0(y), \\ \left(\frac{\partial \phi}{\partial x} - p_1 \phi \right) (a_1, y) &= \alpha_1(y), \\ \left(\frac{\partial \phi}{\partial y} - q_0 \phi \right) (x, 0) &= \beta_0(x), \\ \left(\frac{\partial \phi}{\partial y} - q_1 \phi \right) (x, a_2) &= \beta_1(x). \end{aligned} \tag{12}$$

We place an $n_1 \times n_2$ grid on the rectangle $[0, a_1] \times [0, a_2]$ and discretize the Helmholtz equation. The Robbins boundary conditions are approximated using central differences,

$$\begin{aligned}
\frac{1}{2h_1} (u_{1,j} - u_{-1,j}) - p_0 u_{0,j} &\approx \alpha_{0,j}, \\
\frac{1}{2h_1} (u_{n_1+1,j} - u_{n_1-1,j}) - p_1 u_{n_1,j} &\approx \alpha_{1,j}, \\
\frac{1}{2h_2} (u_{i,1} - u_{i,-1}) - q_0 u_{i,0} &\approx \beta_{0,i}, \\
\frac{1}{2h_2} (u_{i,n_2+1} - u_{i,n_2-1}) - q_1 u_{i,n_2} &\approx \beta_{1,i},
\end{aligned} \tag{13}$$

where $h_1 = a_1/n_1$ and $h_2 = a_2/n_2$. When using the classical five points difference scheme for the Helmholtz equation for $i = 0, 1, \dots, n_1$ and $j = 0, 1, \dots, n_2$, we get

$$u_{i,j} - \theta_1(u_{i+1,j} + u_{i-1,j}) - \theta_2(u_{i,j+1} + u_{i,j-1}) = -\delta f_{i,j}, \tag{14}$$

with δ, θ_1 and θ_2 as in (4).

Using the equations (13) to eliminate the unknown parameters $u_{-1,j}$, $u_{n_1+1,j}$, $u_{i,-1}$ and u_{i,n_2+1} , we get the following linear system with unknowns $u_{i,j}$ similar to (6)

$$M_{RR}\mathbf{u} = \mathbf{b}, \tag{15}$$

$$M_{RR} = \begin{bmatrix} A & 2T & & & \\ T & A & T & & \\ & \ddots & \ddots & \ddots & \\ & & T & A & T \\ & & & 2T & A \end{bmatrix} + 2h_2\theta_2 \begin{bmatrix} q_0 I & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & -q_1 I \end{bmatrix}, \tag{16}$$

where T and A are the same as in (8). We can write M_{RR} in the following form:

$$M_{RR} = I \otimes A + X_{00} \otimes T + \Gamma \otimes I,$$

with

$$X_{00} = \begin{bmatrix} 0 & 2 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 2 & 0 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} \gamma_0 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & \gamma_1 \end{bmatrix}, \tag{17}$$

where $\gamma_0 = 2h_2\theta_2q_0$ and $\gamma_1 = -2h_2\theta_2q_1$. It is not hard to see that the problem above can be written as

$$M_{RR} = M_{NN} + \Gamma \otimes I,$$

with

$$M_{NN} = I \otimes A + X_{00} \otimes T \quad (18)$$

which is exactly the matrix of the Neumann problem as described in the previous section. We rearrange the equation $M_{RR}\mathbf{u} = \mathbf{b}$ like

$$\mathbf{u} = M_{NN}^{-1}\mathbf{b} - M_{NN}^{-1} \begin{bmatrix} \gamma_0\mathbf{u}_0 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \gamma_1\mathbf{u}_{n_2} \end{bmatrix}. \quad (19)$$

It is obvious from the equation above that if we can find the two vectors $\mathbf{u}_0 = (u_{i,0})_{i=0}^{n_1}$ and $\mathbf{u}_{n_2} = (u_{i,n_2})_{i=0}^{n_1}$, we can solve the problem by successively solving two M_{NN} problems, for which the algorithm can be found in the previous section.

3 Low displacement rank matrices

In this section we will use the theory of low displacement rank matrices to find a solution for \mathbf{u}_0 and \mathbf{u}_{n_2} in a fast way. We will use the following notation $B = M_{NN}^{-1}$ and $\mathbf{z} = M_{NN}^{-1}\mathbf{b}$. When partitioning the matrices B and M_{RR} in subblocks the first and the last equation of (19) in \mathbf{u}_0 and \mathbf{u}_{n_2} become:

$$\begin{cases} (I + \gamma_0 B_{0,0})\mathbf{u}_0 + \gamma_1 B_{0,n_2}\mathbf{u}_{n_2} = \mathbf{z}_0 \\ \gamma_0 B_{n_2,0}\mathbf{u}_0 + (I + \gamma_1 B_{n_2,n_2})\mathbf{u}_{n_2} = \mathbf{z}_{n_2} \end{cases}. \quad (20)$$

Because $B_{n_2,n_2} = B_{0,0}$ and $B_{n_2,0} = B_{0,n_2}$, (this will become clear later on, look at formula (24)) we only have to determine the matrices $B_0 = B_{0,0}$ and $B_{n_2} = B_{n_2,0}$ to solve the problem.

We could find them immediately by using the following formula:

$$M_{NN} \begin{bmatrix} B_0 \\ \star \\ B_{n_2} \end{bmatrix} = \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix}$$

but this would give us a computational cost of $O((n_1 n_2)^3)$ flops, which is too much.

However, because the matrices are highly structured, block tridiagonal, and almost block Toeplitz, we can expect a low displacement rank, so we can solve the system of equations with a much lower complexity.

But first we will review the definition, and some important properties of S -matrices. More information about this topic can be found in [6]. We will briefly describe the notation used in the following definitions: The symbol \otimes denotes the *Kronecker product* of matrices [8]. For a $(p \times q)$ -matrix $A = [a_{i,j}]$ and a $(r \times s)$ -matrix $B = [b_{i,j}]$ the Kroneckerproduct $C = B \otimes A$ is a $(pr \times qs)$ -matrix defined as

$$C = \begin{bmatrix} b_{1,1}A & \cdots & b_{1,s}A \\ \vdots & & \vdots \\ b_{r,1}A & \cdots & b_{r,s}A \end{bmatrix}.$$

It is obvious that the matrix C has a block structure with r block rows and s block columns. We denote the (i_2, j_2) th block submatrix of C with C_{i_2, j_2} . The (i_1, j_1) th element of C_{i_2, j_2} is denoted as $C_{i_1, j_1; i_2, j_2}$ and equals $a_{i_1, j_1} b_{i_2, j_2}$. These notations can easily be extended to Kronecker products of 3 or more matrices.

Denote by S_{n-1} the orthogonal sine transform of dimension $n - 1$:

$$S_{n-1} = \left(\sqrt{\frac{2}{n}} \sin \frac{ij\pi}{n} \right)_{i,j=1}^{n-1}, \quad (21)$$

then an S -matrix is defined as follows:

Definition 1 Starting with the orthogonal matrices S as defined in (21) we associate to the sequence of numbers $\{m_1, m_2, \dots, m_k\}$ the following sequence of orthogonal matrices

$$Q^{(s)} = S_{m_s} \otimes Q^{(s-1)}, \quad s = 2, \dots, k,$$

with

$$Q^{(1)} = S_{m_1}.$$

Every matrix of the following form

$$S_k = Q^{(k)} \Lambda Q^{(k)} \quad (22)$$

with Λ an arbitrary diagonal matrix of dimension $m_1 m_2 \cdots m_k$ is called an S_k -matrix associated to the sequence $\{m_1, m_2, \dots, m_k\}$, or briefly an S_k -matrix or S -matrix.

Generalizing the definition of an S -matrix we get:

Definition 2 When Q_1, Q_2, \dots, Q_k are nonsingular square matrices of dimensions m_1, m_2, \dots, m_k , we define the matrices

$$Q^{(s)}$$

recursively as follows:

$$Q^{(s)} = Q_s \otimes Q^{(s-1)}, \quad s = 2, \dots, k,$$

$$Q^{(1)} = Q_1.$$

Then, every matrix of the following form

$$S_k = Q^{(k)} \Lambda \left(Q^{(k)} \right)^{-1}, \quad (23)$$

where the Kronecker-submatrices of Λ satisfy

$$\Lambda_{i_s, j_s; i_{s+1}, j_{s+1}; \dots; i_k, j_k} = 0 \text{ when } i_s \neq j_s, Q_s \neq I_{m_s} \text{ and } s = 1, \dots, k,$$

is called a generalized S_k -matrix associated to the sequence of matrices $\{Q_1, Q_2, \dots, Q_k\}$ or simply a generalized S -matrix.

As mentioned before and shown in [7] the matrix $M_{N,N}$ can be factorized in the following sense:

$$M_{NN} = (C \otimes I) \Lambda (C \otimes I), \quad (24)$$

with Λ a block diagonal matrix with blocks:

$$\Lambda_k = A + 2T \cos \frac{k\pi}{n_2}, \quad k = 0, \dots, n_2. \quad (25)$$

Because $(C \otimes I)^{(-1)} = (C \otimes I)$ it is easy to deduce the following formula:

$$n_2 B_0 = \frac{1}{2} \Lambda_0^{-1} + \frac{1}{2} \Lambda_{n_2}^{-1} + \sum_{k=1}^{n_2-1} \Lambda_k^{-1} = \sum_{k=0}^{n_2} \Lambda_k^{-1}, \quad (26)$$

where the double accent in the sommation means that we have to multiply the first and the last term by $1/2$. In a similar way we can find

$$n_2 B_{n_2} = R_{n_2} = \sum_{k=0}^{n_2} (-1)^k \Lambda_k^{-1}. \quad (27)$$

The matrices Λ_k are tridiagonal matrices and can be inverted easily. On the other hand these matrices can be written as

$$\Lambda_k = M_k + E,$$

with

$$E = \begin{bmatrix} \varepsilon_0 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & \varepsilon_1 \end{bmatrix},$$

where $\varepsilon_0 = 2h_1 p_0 \theta_1$, $\varepsilon_1 = -2h_1 p_1 \theta_1$ and

$$M_k = (A - E) + 2T \cos \frac{k\pi}{n_2},$$

which is a generalized S -matrix associated to C .

To compute $\Lambda_k^{(-1)}$ we will use the displacement operator $\nabla_{\{X_{00}, X_{00}\}}(\cdot)$,

$$\nabla_{\{X_{00}, X_{00}\}}(R) = X_{00}R - RX_{00}. \quad (28)$$

Unfortunately we cannot reconstruct the complete matrix R from the image of R under this displacement operator so we will have to keep some extra information in order to determine the matrix R uniquely.

First we will start by constructing the generators of the matrices Λ_k^{-1} . Remark that the matrix X_{00} is a generalized S -matrix associated to C (C is the matrix of the DCT), which means that

$$X_{00} = CD(\mathbf{c})C, \quad (29)$$

with

$$D(\mathbf{c}) = \text{diag}\{\mathbf{c}\} = 2 \text{diag}\left\{1, \cos \frac{\pi}{n_1}, \dots, \cos \frac{(n_1 - 1)\pi}{n_1}, -1\right\}.$$

Because they are both generalized S -matrices X_{00} and M_k are commutative and we find

$$\begin{aligned}\nabla_{\{X_{00}, X_{00}\}}(\Lambda_k) &= (X_{00}M_k - M_kX_{00}) + (X_{00}E - EX_{00}) \\ &= X_{00}E - EX_{00} \\ &= \begin{bmatrix} 0 & -2\varepsilon_0 & & & & \\ \varepsilon_0 & 0 & 0 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 0 & 0 & \varepsilon_1 & \\ & & & -2\varepsilon_1 & 0 & \end{bmatrix}.\end{aligned}$$

It is easily seen that the displacement rank is 4, and we can construct simple generators

$$\nabla_{\{X_{00}, X_{00}\}}(\Lambda_k) = -\varepsilon_0(2\mathbf{e}_0\mathbf{e}_1^T - \mathbf{e}_1\mathbf{e}_0^T) - \varepsilon_1(2\mathbf{e}_{n_1}\mathbf{e}_{n_1-1}^T - \mathbf{e}_{n_1-1}\mathbf{e}_{n_1}^T).$$

where \mathbf{e}_j denotes the j th unit vector of length $n_1 + 1$. We can rewrite the formula as

$$\begin{aligned}X_{00}\Lambda_k^{-1} - \Lambda_k^{-1}X_{00} &= -\Lambda_k^{-1}(X_{00}\Lambda_k - \Lambda_kX_{00})\Lambda_k^{-1} \\ &= \varepsilon_0\Lambda_k^{-1}(2\mathbf{e}_0\mathbf{e}_1^T - \mathbf{e}_1\mathbf{e}_0^T)\Lambda_k^{-1} + \varepsilon_1\Lambda_k^{-1}(2\mathbf{e}_{n_1}\mathbf{e}_{n_1-1}^T - \mathbf{e}_{n_1-1}\mathbf{e}_{n_1}^T)\Lambda_k^{-1}.\end{aligned}\quad (30)$$

But in fact it is possible to find more convenient generators, using the following lemma.

Lemma 1 *Take R_0 to be an arbitrary square matrix of size $n_1 + 1$ where R_0 has the following structure*

$$R_0 = \begin{bmatrix} \star & 2\alpha & 0 & \cdots & 0 \\ \alpha & \star & \star & \cdots & \star \\ 0 & \star & \star & \cdots & \star \\ \vdots & \star & \star & \cdots & \star \\ 0 & \star & \star & \cdots & \star \end{bmatrix},$$

with α an arbitrary number different from 0, then we have that

$$R_0^{-1}(2\mathbf{e}_0\mathbf{e}_1^T - \mathbf{e}_1\mathbf{e}_0^T)R_0^{-1} = \frac{1}{\alpha}(\mathbf{f}_0\mathbf{e}_0^T - \mathbf{e}_0\mathbf{g}_0^T),$$

with $\mathbf{f}_0 = R_0^{-1}\mathbf{e}_0$ and $\mathbf{g}_0 = R_0^{-T}\mathbf{e}_0$.

Proof :

Because we don't want to overload the notation, we write n when we mean n_1 . We also denote R_0, R_0^{-1} en $K = 2\mathbf{e}_0^{(n+1)}\mathbf{e}_1^{(n+1)T} - \mathbf{e}_1^{(n+1)}\mathbf{e}_0^{(n+1)T}$ in partitioned form

$$R_0 = \begin{bmatrix} \zeta & 2\alpha\mathbf{e}_0^{(n)T} \\ \alpha\mathbf{e}_0^{(n)} & R_\star \end{bmatrix}, \quad R_0^{-1} = \begin{bmatrix} \beta & \mathbf{c}^T \\ \mathbf{d} & E \end{bmatrix},$$

$$K = \begin{bmatrix} 0 & 2\mathbf{e}_0^{(n)T} \\ -\mathbf{e}_0^{(n)} & 0 \end{bmatrix}$$

with $\zeta, \alpha, \beta \in \mathbb{R}$, $\mathbf{c}, \mathbf{d} \in \mathbb{R}^{n \times 1}$ and $R_\star, E \in \mathbb{R}^{n \times n}$. According to the definition of \mathbf{f}_0 and \mathbf{g}_0 we get

$$\mathbf{f}_0 = \begin{pmatrix} \beta \\ \mathbf{d} \end{pmatrix}, \quad \mathbf{g}_0 = \begin{pmatrix} \beta \\ \mathbf{c} \end{pmatrix}. \quad (31)$$

Taking the product of R_0^{-1}, K and R_0^{-1} gives us

$$R_0^{-1}KR_0^{-1} = \begin{bmatrix} \beta \left(2\mathbf{e}_0^{(n)T} \mathbf{d} - \mathbf{c}^T \mathbf{e}_0^{(n)} \right) & 2\beta\mathbf{e}_0^{(n)T} E - \mathbf{c}^T \mathbf{e}_0^{(n)} \mathbf{c}^T \\ 2\mathbf{d}\mathbf{e}_0^{(n)T} \mathbf{d} - \beta E \mathbf{e}_0^{(n)} & 2\mathbf{d}\mathbf{e}_0^{(n)T} E - E \mathbf{e}_0^{(n)} \mathbf{c}^T \end{bmatrix}. \quad (32)$$

On the other hand, using the fact that $R_0^{-1}R_0 = I_{n+1}$ we can deduce the following relationships:

$$\begin{aligned} \mathbf{c}^T \mathbf{e}_0^{(n)} &= \frac{1}{\alpha}(1 - \zeta\beta) \\ E \mathbf{e}_0^{(n)} &= -\frac{1}{\alpha}\zeta\mathbf{d} \\ \mathbf{d}\mathbf{e}_0^{(n)T} &= \frac{1}{2\alpha}(I_n - ER_\star) \end{aligned}.$$

In the same way, using the fact that $R_0R_0^{-1} = I_{n+1}$ we get

$$\begin{aligned} \mathbf{e}_0^{(n)T} \mathbf{d} &= \frac{1}{2\alpha}(1 - \zeta\beta) \\ \mathbf{e}_0^{(n)T} E &= -\frac{1}{2\alpha}\zeta\mathbf{c}^T \\ \mathbf{e}_0^{(n)} \mathbf{c}^T &= \frac{1}{\alpha}(I_n - R_\star E) \end{aligned}.$$

Using these equations in (32), we get

$$R_0^{-1}KR_0^{-1} = \begin{bmatrix} 0 & -\mathbf{c}^T/\alpha \\ \mathbf{d}/\alpha & 0 \end{bmatrix} = \frac{1}{\alpha} \left(\begin{pmatrix} \beta \\ \mathbf{d} \end{pmatrix} \mathbf{e}_0^{(n+1)T} - \mathbf{e}_0^{(n+1)} \begin{pmatrix} \beta & \mathbf{c}^T \end{pmatrix} \right).$$

According to (31) this is what we had to prove. ■

In the same way we can deduce that for a matrix R_{n_1} of the form

$$R_{n_1} = \begin{bmatrix} \star & \star & \cdots & \star & 0 \\ \vdots & & & \vdots & \vdots \\ \star & \cdots & \star & \star & 0 \\ \star & \cdots & \star & \star & \alpha \\ 0 & \cdots & 0 & 2\alpha & \star \end{bmatrix}$$

again with α an arbitrary number different from zero, the following equation holds

$$R_{n_1}^{-1}(2\mathbf{e}_{n_1}\mathbf{e}_{n_1-1}^T - \mathbf{e}_{n_1-1}\mathbf{e}_{n_1}^T)R_{n_1}^{-1} = \frac{1}{\alpha}(\mathbf{f}_{n_1}\mathbf{e}_{n_1}^T - \mathbf{e}_{n_1}\mathbf{g}_{n_1}^T),$$

with $\mathbf{f}_{n_1} = R_{n_1}^{-1}\mathbf{e}_{n_1}$ and $\mathbf{g}_{n_1} = R_{n_1}^{-T}\mathbf{e}_{n_1}$.

When $R = \Lambda_k$ we only have to calculate \mathbf{f}_0 and \mathbf{f}_{n_1} . Indeed, take the diagonal matrix $D = \text{diag}\{1, 2, \dots, 2, 1\}$, then its easy to check that $\Lambda_k^T D = D\Lambda_k$, $D\mathbf{e}_0 = \mathbf{e}_0$ and $D\mathbf{e}_{n_1} = \mathbf{e}_{n_1}$. Take $\mathbf{f}_0 = \Lambda_k^{-1}\mathbf{e}_0$ and $\mathbf{f}_{n_1} = \Lambda_k^{-1}\mathbf{e}_{n_1}$, then we have:

$$\Lambda_k^T(D\mathbf{f}_0) = D\Lambda_k\mathbf{f}_0 = D\mathbf{e}_0 = \mathbf{e}_0,$$

$$\Lambda_k^T(D\mathbf{f}_{n_1}) = D\Lambda_k\mathbf{f}_{n_1} = D\mathbf{e}_{n_1} = \mathbf{e}_{n_1}.$$

Now we can calculate \mathbf{g}_0 and \mathbf{g}_{n_1} using \mathbf{f}_0 and \mathbf{f}_{n_1} :

$$\mathbf{g}_0 = D\mathbf{f}_0, \quad \mathbf{g}_{n_1} = D\mathbf{f}_{n_1}.$$

Eventually we can rewrite (30) and get

$$\nabla_{\{X_{00}, X_{00}\}}(\Lambda_k^{-1}) = \frac{\varepsilon_0}{\theta_1}(\mathbf{x}_k\mathbf{e}_0^T - \mathbf{e}_0(D\mathbf{x}_k)^T) + \frac{\varepsilon_1}{\theta_1}(\mathbf{y}_k\mathbf{e}_{n_1}^T - \mathbf{e}_{n_1}(D\mathbf{y}_k)^T), \quad (33)$$

with

$$\mathbf{x}_k = \Lambda_k^{-1}\mathbf{e}_0 \quad (34)$$

and

$$\mathbf{y}_k = \Lambda_k^{-1}\mathbf{e}_{n_1}.$$

Using the equations (26) and (27) we find

$$X_{00}B_0 - B_0X_{00} = \mathbf{x}_+\mathbf{e}_0^T - \mathbf{e}_0(D\mathbf{x}_+)^T + \mathbf{y}_+\mathbf{e}_{n_1}^T - \mathbf{e}_{n_1}(D\mathbf{y}_+)^T \quad (35)$$

with

$$\mathbf{x}_+ = \left(\frac{-1}{n_2} \right) \left(\frac{\varepsilon_0}{\theta_1} \sum_{k=0}^{n_2''} \mathbf{x}_k \right) \quad (36)$$

and

$$\mathbf{y}_+ = \left(\frac{-1}{n_2} \right) \left(\frac{\varepsilon_1}{\theta_1} \sum_{k=0}^{n_2''} \mathbf{y}_k \right).$$

We see now that the matrix B_0 also has displacement rank 4, and we can find the generators using the formula here above. In a completely similar way we can deduce the following relations for B_{n_2}

$$\nabla_{\{X_{00}, X_{00}\}}(B_{n_2}) = \mathbf{x}_- \mathbf{e}_0^T - \mathbf{e}_0 (D\mathbf{x}_-)^T + \mathbf{y}_- \mathbf{e}_{n_1}^T - \mathbf{e}_{n_1} (D\mathbf{y}_-)^T \quad (37)$$

with

$$\mathbf{x}_- = \left(\frac{-1}{n_2} \right) \left(\frac{\varepsilon_0}{\theta_1} \sum_{k=0}^{n_2''} (-1)^k \mathbf{x}_k \right)$$

and

$$\mathbf{y}_- = \left(\frac{-1}{n_2} \right) \left(\frac{\varepsilon_1}{\theta_1} \sum_{k=0}^{n_2''} (-1)^k \mathbf{y}_k \right).$$

Hence both B_0 and B_{n_2} have a low displacement rank, and we can calculate their generators.

Before solving the special case (20), we take a look at the more general problem, how we can solve a system of equations with a low $\{X_{00}, X_{00}\}$ displacement rank. Therefore we take R to be an arbitrary square matrix of size $n+1$, for which

$$\nabla_{\{X_{00}, X_{00}\}}(R) = FG^T,$$

with F and G $(n+1) \times r$ -matrices. We already know that we can diagonalize X_{00} by using a cosine transformation.

Using the same techniques as in [5, 1] we can take another displacement operator, and transform R into another class of structured matrices. First of all we will give the definition of a generalized Cauchy matrix.

Definition 3 Let $c = (c_i)_1^n$ and $d = (d_j)_1^n$ be fixed n -tuples of numbers and $A = [a_{ij}]_{i,j=1}^n$ a given matrix. The Cauchy rank of a matrix is the rank r of

$$\nabla_{c,d}(A) = [(c_i - d_j) a_{ij}]_{i,j=1}^n$$

When r is small with respect to the order of A , then A will be called a generalized Cauchy matrix. For a classical Cauchy matrix $(c_i - d_j)a_{ij} = 1$ for all i and j , and has therefore, Cauchy rank 1.

When we take $\hat{R} = CRC$, then according to (28) and (29) we find that

$$\nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{R}) = D(\mathbf{c})\hat{R} - \hat{R}D(\mathbf{c}) = \hat{F}\hat{G}^T \quad (38)$$

with $\hat{F} = CF$ and $\hat{G} = C^T G$, such that \hat{R} is a generalized Cauchy matrix. Using the generators we can almost completely reconstruct the matrix \hat{R} . Only the diagonal elements are lost. So, we still have to find the diagonal of \hat{R} which can be constructed by first calculating $\mathbf{u} = \hat{R}\mathbf{1} = CRC\mathbf{1}$ with $\mathbf{1} = [1 \ \dots \ 1]^T$. Using this we can find the diagonal of \hat{R} using the following formula:

$$d_i = u_i - \sum_{j \neq i}^{0, n_1} \frac{\hat{\mathbf{f}}_i^T \hat{\mathbf{g}}_j}{c_i - c_j}, \quad i = 0, \dots, n_1.$$

We can now use the LU-CAUCHY algorithm of Bojanczyk and Heinig [2], to construct the LU factorization of the matrix \hat{R} knowing the generators and its diagonal elements. We can also implement row pivoting in this algorithm [5].

In this final paragraph, we will solve the equations (20). The matrix of the system of equations is

$$B = \begin{pmatrix} I + \gamma_0 B_0 & \gamma_1 B_{n_2} \\ \gamma_0 B_{n_2} & I + \gamma_1 B_0 \end{pmatrix}. \quad (39)$$

When we take a look at the four subblocks of the matrix we know that they all have a low $\{X_{00}, X_{00}\}$ -displacement rank. This means that the matrix B will have a low displacement rank according to the following displacement operator $\nabla_{\{X, X\}}$ with X the direct sum of X_{00} with itself:

$$X = X_{00} \oplus X_{00} = \begin{pmatrix} X_{00} & 0 \\ 0 & X_{00} \end{pmatrix}.$$

According to (29) we have

$$X_{00} \oplus X_{00} = (C \oplus C) (D(\mathbf{c}) \oplus D(\mathbf{c})) (C \oplus C),$$

so we can also transform B to a generalized Cauchy-like matrix \hat{B} :

$$\hat{B} = (C \oplus C) B (C \oplus C),$$

using the displacement operator

$$\nabla_{\{D(\mathbf{c}) \oplus D(\mathbf{c}), D(\mathbf{c}) \oplus D(\mathbf{c})\}}(\cdot). \quad (40)$$

Note that

$$\hat{B} = \begin{pmatrix} I + \gamma_0 \hat{B}_0 & \gamma_1 \hat{B}_{n_2} \\ \gamma_0 \hat{B}_{n_2} & I + \gamma_1 \hat{B}_0 \end{pmatrix},$$

so we can calculate the generators of \hat{B} using these of $\hat{B}_0 = CB_0C$ and $\hat{B}_{n_2} = CB_{n_2}C$. By multiplying (35) on both sides with C , using (29) and the fact that $C^T D = DC$ we obtain the following relationships

$$\nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{B}_0) = \hat{\mathbf{x}}_+(D\hat{\mathbf{e}}_0)^T - \hat{\mathbf{e}}_0(D\hat{\mathbf{x}}_+)^T + \hat{\mathbf{y}}_+(D\hat{\mathbf{e}}_{n_1})^T - \hat{\mathbf{e}}_{n_1}(D\hat{\mathbf{y}}_+)^T \quad (41)$$

and in the same way we get

$$\nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{B}_{n_2}) = \hat{\mathbf{x}}_-(D\hat{\mathbf{e}}_0)^T - \hat{\mathbf{e}}_0(D\hat{\mathbf{x}}_-)^T + \hat{\mathbf{y}}_-(D\hat{\mathbf{e}}_{n_1})^T - \hat{\mathbf{e}}_{n_1}(D\hat{\mathbf{y}}_-)^T, \quad (42)$$

with $\hat{\mathbf{x}}_{\pm} = C\mathbf{x}_{\pm}$, $\hat{\mathbf{y}}_{\pm} = C\mathbf{y}_{\pm}$, $\hat{\mathbf{e}}_i = C\mathbf{e}_i$, for $i = 0, \dots, n_1$. We can now write the displacement of \hat{B} as

$$\nabla_{\{D(\mathbf{c}) \oplus D(\mathbf{c}), D(\mathbf{c}) \oplus D(\mathbf{c})\}}(\hat{B}) = \begin{pmatrix} \gamma_0 \nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{B}_0) & \gamma_1 \nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{B}_{n_2}) \\ \gamma_0 \nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{B}_{n_2}) & \gamma_1 \nabla_{\{D(\mathbf{c}), D(\mathbf{c})\}}(\hat{B}_0) \end{pmatrix},$$

and we can calculate the generators

$$\begin{aligned} & \nabla_{\{D(\mathbf{c}) \oplus D(\mathbf{c}), D(\mathbf{c}) \oplus D(\mathbf{c})\}}(\hat{B}) \\ &= \gamma_0 \begin{pmatrix} \hat{\mathbf{x}}_+ \\ \hat{\mathbf{x}}_- \end{pmatrix} \begin{pmatrix} (D\hat{\mathbf{e}}_0)^T & \mathbf{0} \end{pmatrix} - \begin{pmatrix} \hat{\mathbf{e}}_0 \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} \gamma_0 (D\hat{\mathbf{x}}_+)^T & \gamma_1 (D\hat{\mathbf{x}}_-)^T \end{pmatrix} \\ &+ \gamma_0 \begin{pmatrix} \hat{\mathbf{y}}_+ \\ \hat{\mathbf{y}}_- \end{pmatrix} \begin{pmatrix} (D\hat{\mathbf{e}}_{n_1})^T & \mathbf{0} \end{pmatrix} - \begin{pmatrix} \hat{\mathbf{e}}_{n_1} \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} \gamma_0 (D\hat{\mathbf{y}}_+)^T & \gamma_1 (D\hat{\mathbf{y}}_-)^T \end{pmatrix} \\ &+ \gamma_1 \begin{pmatrix} \hat{\mathbf{x}}_- \\ \hat{\mathbf{x}}_+ \end{pmatrix} \begin{pmatrix} \mathbf{0} & (D\hat{\mathbf{e}}_0)^T \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \hat{\mathbf{e}}_0 \end{pmatrix} \begin{pmatrix} \gamma_0 (D\hat{\mathbf{x}}_-)^T & \gamma_1 (D\hat{\mathbf{x}}_+)^T \end{pmatrix} \\ &+ \gamma_1 \begin{pmatrix} \hat{\mathbf{y}}_- \\ \hat{\mathbf{y}}_+ \end{pmatrix} \begin{pmatrix} \mathbf{0} & (D\hat{\mathbf{e}}_{n_1})^T \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \hat{\mathbf{e}}_{n_1} \end{pmatrix} \begin{pmatrix} \gamma_0 (D\hat{\mathbf{y}}_-)^T & \gamma_1 (D\hat{\mathbf{y}}_+)^T \end{pmatrix}. \end{aligned}$$

In order to reconstruct the matrix \hat{B} completely, besides the generators, we also need the diagonals of \hat{B}_0 and \hat{B}_{n_2} . Remark that we need to reconstruct the diagonals of the four subblocks of the matrix, to completely restore \hat{B} , that is why we have to adapt the LU-CAUCHY algorithm to this special situation.

To reconstruct the diagonal \mathbf{d}_+ of \hat{B}_0 , we first calculate $\mathbf{u} = \hat{B}_0 \mathbf{1}$, which we can rewrite using (26) as

$$\mathbf{u} = \left(\frac{1}{n_2} \right) \left(\sum_{k=0}^{n_2} \hat{\Lambda}_k^{-1} \mathbf{1} \right),$$

with $\hat{\Lambda}_k^{-1} = C \Lambda_k^{-1} C$. Remark that $\mathbf{1} = \sqrt{2n_1} \hat{\mathbf{e}}_0$ and when we denote $\hat{\mathbf{x}}_k = C \mathbf{x}_k$ we find using (34) that $\hat{\mathbf{x}}_k = \hat{\Lambda}_k^{-1} \hat{\mathbf{e}}_0$, such that

$$\mathbf{u} = \left(\frac{\sqrt{2n_1}}{n_2} \right) \sum_{k=0}^{n_2} \hat{\Lambda}_k^{-1} \hat{\mathbf{e}}_0 = \left(\frac{\sqrt{2n_1}}{n_2} \right) \sum_{k=0}^{n_2} \hat{\mathbf{x}}_k.$$

Because of (36) we get

$$\mathbf{u} = -\sqrt{2n_1} \frac{\theta_1}{\varepsilon_0} \hat{\mathbf{x}}_+.$$

In order to find the i th component of the diagonal \mathbf{d}_+ we have to subtract from u_i the nondiagonal elements of the i th row of \hat{B}_0 which we can find using the generators of \hat{B}_0 . So finally we find for $i = 0, \dots, n_1$

$$d_{+i} = -\sqrt{2n_1} \frac{\theta_1}{\varepsilon_0} \hat{x}_{+i} - \sqrt{\frac{2}{n_1}} \sum_{j \neq i}^{0, n_1} \frac{\hat{x}_{+i} - \hat{x}_{+j} + (-1)^j \hat{y}_{+i} - (-1)^i \hat{y}_{+j}}{c_i - c_j}.$$

In a completely analogous manner we can find the diagonal \mathbf{d}_- of \hat{B}_{n_2}

$$d_{-i} = -\sqrt{2n_1} \frac{\theta_1}{\varepsilon_0} \hat{x}_{-i} - \sqrt{\frac{2}{n_1}} \sum_{j \neq i}^{0, n_1} \frac{\hat{x}_{-i} - \hat{x}_{-j} + (-1)^j \hat{y}_{-i} - (-1)^i \hat{y}_{-j}}{c_i - c_j}.$$

That is everything we need to deduce the algorithm that finds the LU factorization of \hat{B} .

4 The Algorithm

First of all we present the algorithm which calculates the LU factorization of the matrix $\hat{B} = (C \oplus C) B (C \oplus C)$ with B given by (39). This LU-factorization can later on be used to solve (20).

algorithm 2 *Calculating the LU factorization of the matrix B given by (39).*

1. Calculate the generators of the matrices Λ_k^{-1} (cf. formula (33)).
For $k = 0, \dots, n_2$, calculate

$$\mathbf{x}_k = \Lambda_k^{-1} \mathbf{e}_0, \quad \mathbf{y}_k = \Lambda_k^{-1} \mathbf{e}_{n_1}.$$

2. Calculate the generators for B_0 and B_{n_2} (cf. formula (35)).
Calculate

$$\begin{aligned} \mathbf{x}_+ &= \frac{(-1)}{n_2} \left(\frac{\boldsymbol{\varepsilon}_0}{\boldsymbol{\theta}_1} \sum_{k=0}^{n_2} \mathbf{x}_k \right), & \mathbf{x}_- &= \frac{(-1)}{n_2} \left(\frac{\boldsymbol{\varepsilon}_0}{\boldsymbol{\theta}_1} \sum_{k=0}^{n_2} (-1)^k \mathbf{x}_k \right), \\ \mathbf{y}_+ &= \frac{(-1)}{n_2} \left(\frac{\boldsymbol{\varepsilon}_1}{\boldsymbol{\theta}_1} \sum_{k=0}^{n_2} \mathbf{y}_k \right), & \mathbf{y}_- &= \frac{(-1)}{n_2} \left(\frac{\boldsymbol{\varepsilon}_1}{\boldsymbol{\theta}_1} \sum_{k=0}^{n_2} (-1)^k \mathbf{y}_k \right). \end{aligned}$$

3. Transform to the generators for \hat{B}_0 and \hat{B}_{n_2} (cf. formula (41) and (42)).
Calculate

$$\begin{aligned} \hat{\mathbf{x}}_+ &= C\mathbf{x}_+, & \hat{\mathbf{x}}_- &= C\mathbf{x}_-, \\ \hat{\mathbf{y}}_+ &= C\mathbf{y}_+, & \hat{\mathbf{y}}_- &= C\mathbf{y}_-, \\ \hat{\mathbf{e}}_0 &= C\mathbf{e}_0, & \hat{\mathbf{e}}_{n_1} &= C\mathbf{e}_{n_1}. \end{aligned}$$

4. Calculate the diagonal \mathbf{d} of \hat{B} and of the $(1, 2)$ - en $(2, 1)$ -subblock of \hat{B} (resp. \mathbf{d}_U en \mathbf{d}_L).
Calculate for $i = 1, \dots, n_1$:

$$d_{+i} = -\sqrt{2n_1} \frac{\theta_1}{\varepsilon_0} \hat{x}_{+i} - \sqrt{\frac{2}{n_1}} \sum_{j \neq i}^{0, n_1} \frac{\hat{x}_{+i} - \hat{x}_{+j} + (-1)^j \hat{y}_{+i} - (-1)^i \hat{y}_{+j}}{c_i - c_j},$$

$$d_{-i} = -\sqrt{2n_1} \frac{\theta_1}{\varepsilon_0} \hat{x}_{-i} - \sqrt{\frac{2}{n_1}} \sum_{j \neq i}^{0, n_1} \frac{\hat{x}_{-i} - \hat{x}_{-j} + (-1)^j \hat{y}_{-i} - (-1)^i \hat{y}_{-j}}{c_i - c_j},$$

$$\mathbf{d} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \begin{pmatrix} \gamma_0 \mathbf{d}_+ \\ \gamma_1 \mathbf{d}_+ \end{pmatrix},$$

$$\mathbf{d}_L = \gamma_0 \mathbf{d}_-, \quad \mathbf{d}_U = \gamma_1 \mathbf{d}_-,$$

where $c_i = c_{i+n_1+1} = 2 \cos \frac{i\pi}{n_1}$, for $i = 0, 1, \dots, n_1$.

5. Initialise the generators \hat{F} and \hat{G} of \hat{B} .

With the following notations

$$\begin{aligned} \nabla_{\{D(\mathbf{c}) \oplus D(\mathbf{c}), D(\mathbf{c}) \oplus D(\mathbf{c})\}}(\hat{B}) &= \hat{F} \hat{G}^T \\ &= \begin{pmatrix} \phi_0^T \\ \phi_1^T \\ \vdots \\ \phi_{2n_1+1}^T \end{pmatrix} (\psi_0 \ \psi_1 \ \dots \ \psi_{2n_1+1}) \end{aligned}$$

We have to give \hat{F} and \hat{G} the following values

$$\hat{F} = \begin{bmatrix} \gamma_0 \hat{\mathbf{x}}_+ & \gamma_0 \hat{\mathbf{y}}_+ & -\hat{\mathbf{e}}_0 & -\hat{\mathbf{e}}_{n_1} & \gamma_1 \hat{\mathbf{x}}_- & \gamma_1 \hat{\mathbf{y}}_- & \mathbf{0} & \mathbf{0} \\ \gamma_0 \hat{\mathbf{x}}_- & \gamma_0 \hat{\mathbf{y}}_- & \mathbf{0} & \mathbf{0} & \gamma_1 \hat{\mathbf{x}}_+ & \gamma_1 \hat{\mathbf{y}}_+ & -\hat{\mathbf{e}}_0 & -\hat{\mathbf{e}}_{n_1} \end{bmatrix}$$

and

$$\hat{G} = \begin{bmatrix} D\hat{\mathbf{e}}_0 & D\hat{\mathbf{e}}_{n_1} & \gamma_0 D\hat{\mathbf{x}}_+ & \gamma_0 D\hat{\mathbf{y}}_+ & \mathbf{0} & \mathbf{0} & \gamma_0 D\hat{\mathbf{x}}_- & \gamma_0 D\hat{\mathbf{y}}_- \\ \mathbf{0} & \mathbf{0} & \gamma_1 D\hat{\mathbf{x}}_- & \gamma_1 D\hat{\mathbf{y}}_- & D\hat{\mathbf{e}}_0 & D\hat{\mathbf{e}}_{n_1} & \gamma_1 D\hat{\mathbf{x}}_+ & \gamma_1 D\hat{\mathbf{y}}_+ \end{bmatrix}.$$

6. The LU-CAUCHY algorithm adapted to this situation (cf. [2])

for $k = 0, 1, \dots, 2n_1 + 1$, calculate

(a) Calculate column k of L and row k of U from the generators.

$$l_{kk} = d_k$$

For $j = k + 1, k + 2, \dots, 2n_1 + 1$,

$$i. \ l_{jk} = \begin{cases} d_{Lk} & \text{if } j = n_1 + 1 + k \\ \frac{\phi_j^T \psi_k}{c_j - c_k} & \text{else} \end{cases}$$

$$ii. \ u_{kj} = \begin{cases} d_{Uk} & \text{if } j = n_1 + 1 + k \\ \frac{\phi_k^T \psi_j}{c_k - c_j} & \text{else} \end{cases}$$

$$u_{kk} = l_{kk} \text{ en } l_{kk} = 1.$$

For $j = k + 1, k + 2, \dots, 2n_1 + 1$, calculate

$$l_{jk} = \frac{l_{jk}}{u_{kk}}$$

(b) Calculate the Schurcomplement.

For $j = k + 1, k + 2, \dots, 2n_1 + 1$, calculate

$$i. \ \phi_j = \phi_j - l_{jk} \phi_k$$

$$ii. \ t = \frac{u_{kj}}{u_{kk}}$$

- iii. $\Psi_j = \Psi_j - t\Psi_k$
- iv. $d_j = d_j - l_{jk}u_{kj}$
- v. If $k \leq n_1$,
 - $d_{Lj} = d_{Lj} - l_{n_1+1+j,k}u_{kj}$
 - $d_{Uj} = d_{Uj} - l_{jk}u_{k,n_1+1+j}$

In the same way as in [5] we can apply row pivoting after step 6(a)i, this will improve the numerical stability of the algorithm. The complexity of the algorithm presented here is $O(n_1n_2) + O(n_1^2)$. Indeed, step 1 requires solving $n_2 + 1$ tridiagonal systems of equations of size $n_1 + 1$, in step 2 and 4 we have to add $n_2 + 1$ vectors of length $n_1 + 1$ and in the final step for every k (from 0 to n_1) $72(2n_1 - k)$ flops have to be executed. The complexity of the other steps is of a lower magnitude.

When we want to solve (20) we use the fact that

$$B = (C \oplus C) \hat{B} (C \oplus C).$$

So we get the following algorithm for the solution of the Helmholtz equation with Robbins boundary conditions.

algorithm 3 (Robbins problem) *This algorithm solves the system of equations*

$$M_{RR}\mathbf{u} = \mathbf{b}$$

where M_{RR} is the matrix (16) coming from a finite difference approximation of the Helmholtz equation on a rectangular grid with Robbins boundary conditions.

1. Find the LU-factorization of \hat{B} with the previous algorithm (Algorithm 2).
2. Calculate $\mathbf{z} = M_{NN}^{-1}\mathbf{b}$ with M_{NN} from (18) by using a fast direct method for the Neumann-Neumann problem.
3. Calculate \mathbf{u}_0 and \mathbf{u}_{n_2} .
 - (a) Calculate $\hat{\mathbf{z}}_0 = C\mathbf{z}_0$ and $\hat{\mathbf{z}}_{n_2} = C\mathbf{z}_{n_2}$ with a fast cosine transform.
 - (b) Solve the system of equations

$$\hat{B} \begin{bmatrix} \hat{\mathbf{u}}_0 \\ \hat{\mathbf{u}}_{n_2} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{z}}_0 \\ \hat{\mathbf{z}}_{n_2} \end{bmatrix}$$

by using the LU-factorization of \hat{B} .

(c) Calculate $\mathbf{u}_0 = C\hat{\mathbf{u}}_0$ and $\mathbf{u}_{n_2} = C\hat{\mathbf{u}}_{n_2}$ with a fast cosine transform.

4. Calculate

$$\mathbf{y} = M_{NN}^{-1} \begin{bmatrix} \gamma_0 \mathbf{u}_0 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \gamma_{n_2} \mathbf{u}_{n_2} \end{bmatrix}$$

analogous as in step 2.

5. Calculate $\mathbf{u} = \mathbf{y} - \mathbf{z}$

Because we had to solve two systems of equations with M_{NN} as matrix the computational complexity of the algorithm will be $10n_1n_2 \log n_2 + O(n_1^2 + n_1n_2)$ flops.

5 Numerical Examples

5.1 Timings

We implemented the algorithm (with pivoting) in Fortran90, using the NAG95-Fortran compiler on a linux platform. The tests were done on a Pentium III 860 Mhz processor, with 512 MByte of memory. In order to test its complexity, we generated a fixed example and changed the variables n_1 and n_2 . By using the least squares approximation we obtained the following complexity:

$$0.6n_1n_2 \log(n_2) + 3.6n_1^2 - 3.6n_1n_2$$

in (sec/Mflops). It can be seen in figure 1 that this is a good approximation: the surface is generated using the formula above, and the dots are the measured timings. Making n_1 larger, slows down the algorithm more than enlarging the factor n_2 . However this is no problem, because, when n_1 is larger than n_2 , we can just swap the roles of the variables x and y in (1). Then also n_1 and n_2 are interchanged.

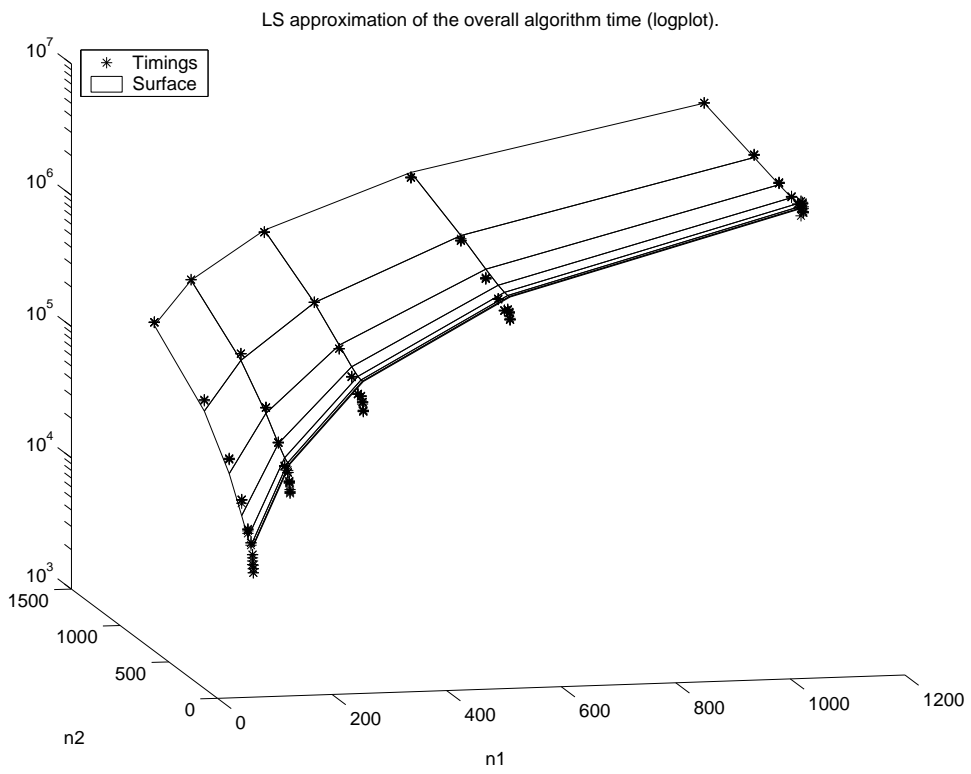


Figure 1: Timing results for the overall algorithm.

5.2 Stability

First of all, we changed the parameter λ , to make the system of equations very ill-conditioned, we did this for parameter values $n_1 = 16$ and $n_2 = 16$. Note that if we fill in for λ the eigenvalues of M_{RR} (with $\lambda=0$), then the corresponding M_{RR} matrix becomes nonsingular. We approximated the largest eigenvalue in modulus using the power method, and then changed it slightly, to obtain some different condition numbers for the matrix M_{RR} . The largest eigenvalue obtained by the power method: $\lambda = 1.06175777127e + 02$, in the following table the $\Delta\lambda$, and the different condition numbers are shown:

n^o	$\Delta\lambda$	Condition(M_{RR})	Condition(M_{NN})	Condition(B)
1	6.2500000e-02	1.9730530e+03	1.0973156e+02	9.8528442e+01
2	7.7160499e-04	1.5987623e+05	1.1662279e+02	7.8476879e+03
3	1.1972999e-05	1.0303353e+07	1.1671302e+02	5.0564809e+05
4	1.1800000e-07	1.0494080e+09	1.1671443e+02	5.1500656e+07
5	9.9998942e-10	1.0242767e+11	1.1671445e+02	5.0267124e+09
6	0	1.0270375e+13	1.1671445e+02	5.0399766e+11

It can be seen, that the bad conditioning of M_{NN} is completely incorporated in the matrix B . In the following table, it can be seen that the loss of significant numbers can completely be explained by the ill conditioning of the matrix. When using the notation $M_{RR}\mathbf{u} = \mathbf{b}$, where \mathbf{u} denotes the exact solution and \mathbf{x} is the solution calculated by the algorithm, then the relative residual norm is calculated as follows: $\|M_{RR}\mathbf{x} - \mathbf{b}\|_2 / \|\mathbf{b}\|_2$.

n^o	Relative residual norm	$\ \mathbf{x}\ _2$	$\ M_{RR}\ _2$
1	0.269875541131E-15	0.137537825554E+02	2.124455918105E+00
2	0.151802077218E-13	0.750048621793E+03	2.125753734878E+00
3	0.775739556738E-12	0.483388831755E+05	2.125769724492E+00
4	0.787803224555E-10	0.492337955471E+07	2.125769974033E+00
5	0.745524749715E-08	0.480537415275E+09	2.125769976496E+00
6	0.938779346662E-06	0.482384800059E+11	2.125769976517E+00

In the second part we first took fixed parameters, for the system M_{RR} , except for λ and the right-hand side $f_{i,j}$, the λ 's here are the same as those above, to have

an ill conditioned system. For every λ we searched the right-hand side using a fixed solution \mathbf{u}_e (with $\|\mathbf{u}_e\|_2 \approx 500$). On these problems we ran the algorithm, and calculated the relative residual norm and the relative error norm (which is calculated as follows: $\|\mathbf{x} - \mathbf{u}_e\|_2 / \|\mathbf{u}_e\|_2$), as shown in the next table:

n^o	Relative residual norm	Relative error norm
1	0.247972802994E-15	0.273041336350E-13
2	0.302336805921E-15	0.107901396342E-11
3	0.236340232958E-15	0.163183467236E-09
4	0.308345718370E-15	0.269762204052E-08
5	0.251476127414E-15	0.530808215848E-07
6	0.269757268817E-15	0.707838423187E-04

In the last test we did, we tried some random examples. We generated random numbers in Fortran and printed out the relative residual norm. We took $n_1 = n_2$ and did ten random tests for each choice of $n_1 = 2^k$ where $k = 3, 4, \dots, 10$. In figure 2 the relative residual norms are shown.

5.3 The software.

For the reader who is interested in testing or evaluating our software, we implemented the routine in Fortran as well as in Matlab¹. You can download it at: <http://www.cs.kuleuven.ac.be/??????/????????????>

References

- [1] G. Heinig, A. Bojanczyk. *Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices. I. Transformations*. Linear Algebra Appl., 254 (1997), pp. 193-226.
- [2] G. Heinig, A. Bojanczyk. *Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices. II. Algorithms*. Linear Algebra Appl., 278 (1998), pp. 11-36.

¹Matlab is a registered trademark of The Mathworks, Inc.

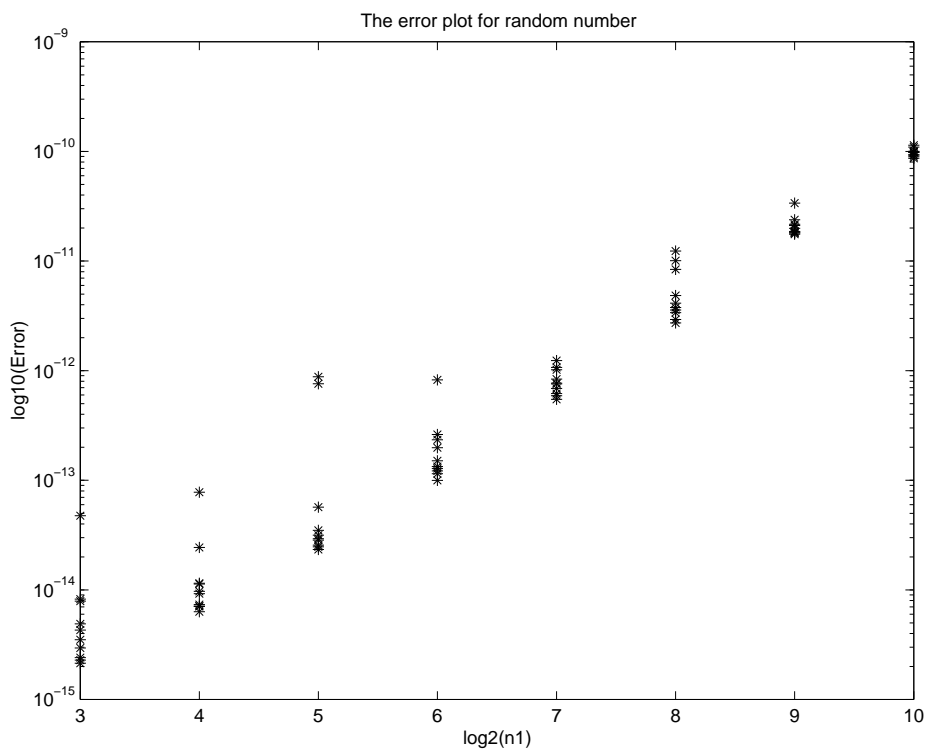


Figure 2: Relative residual norms for the overall algorithm.

- [3] P.N. Swarztrauber. *Symmetric FFTs*. Math. Comput., 47 (1986), pp. 323-346.
- [4] C. Van Loan. *Computational frameworks for the Fast Fourier Transform*. SIAM, Philadelphia, 1992.
- [5] I. Gohberg, T. Kailath, V. Olshevsky. *Fast Gaussian elimination with partial pivoting for matrices with displacement structure*. Math. Comput. 64 (1995), pp. 1557-1576.
- [6] L. Mertens. *S_k -matrices en hun toepassingen bij de eindige differentiebenaderingen van differentiaalvergelijkingen*. PhdThesis, K.U. Leuven, 1990.
- [7] J. Hendrickx. *Veralgemeende S -matrices en hun toepassingen bij de eindigedifferentiebenaderingen van differentiaalvergelijkingen en het oplossen van symmetrische band-Toeplitzstelsels*. PhdThesis, K.U. Leuven, 2000.
- [8] A. Graham. *Kronecker products and matrix calculations with applications*. John Wiley, New York, 1981, Chapter 2-3.
- [9] M. Pickering *An introduction to Fast Fourier Transform Methods for Partial Differential Equations, with applications*. John Wiley, New York, 1986.
- [10] M.N. Ozisik *Heat Conduction*. John Wiley, New York, 1980.