Grid algebra in finite difference methods

Michael P. Lamoureux[*] and Heather K. Hardeman[*]

ABSTRACT

Large, sparse matrices often appear in numerical methods for solving partial differential equations, particularly in finite difference solutions to the wave equation in two and three dimensions. We demonstrate a technique to represent these operators directly on a grid and develop linear algebraic methods to simplify or factor the operator into a form that is easy to solve using back-substitution. We apply the technique to implement an implicit solver for a finite difference algorithm applied to the wave equation in two dimensions.

INTRODUCTION

Finding efficient numerical methods for solving systems of differential equations that describe physical processes is a central challenge in scientific computation. In seismic imaging, a key problem is finding numerical solutions for the various wave equation models that describe seismic wave propagation through the earth's subsurface. Several numerical methods for calculating the wave propagation in space involve approximating the wavefield on a grid of nodes or cells in space and modelling its propagation along the grid. A key step is representing the wave operator (or some part of it) as a linear matrix acting on the vector space of functions evaluated at the grid points. For even modest grids used in finite differences, say 100×100 in 2D or $100\times100\times100$ in 3D, the vector spaces that appear have 10,000 or 1,000,000 dimensions and the associated linear operators are represented as matrices of size $10{,}000\times10{,}000$ or $1{,}000{,}000\times1{,}000{,}000$. Fortunately, these matrices are typically sparse, and there exist sparse matrix methods and iterative methods that can manage the high dimensions of the problem. A good reference to such standard methods is in (Press et al., 2007).

We are interested specifically in developing implicit solutions for finite difference methods of modelling wave propagation. A central challenge in the implicit method is that somewhere along the line, a system of linear equations has to be solved. Here, the high dimensions are a particular problem and even general sparse matrix methods only partially alleviate the challenge.

We note, however, that there is often an underlying structure in the linear systems that arises from the fact that the organized grid of points in 2D or 3D gives additional geometric relationships between coefficients in the linear operator. The general sparse matrix techniques do not readily take advantage of these geometric relationships, so there is an opportunity to develop algorithm efficiencies from this extra information. For instance, a linear operator on a 2D grid that has only "nearest neighbour" dependence on terms will be represented by a sparse, banded matrix that has entries spread out all over the span of the matrix. Applying row operations or performing an LU decomposition of the matrix will destroy the underlying nearest neighbour structure, resulting in a matrix that is more

[*]University of Calgary

difficult with which to compute. As an alternative to the sparse matrix methods, we aim to use the underlying geometric structure to perform “matrix operations” directly on the structured linear system to produce more efficient linear algebraic methods.

In this work, we demonstrate a representation of these linear operators via a graph of arrows and nodes. The nodes represent grid points, arrows represent non-zero entries in the associated linear operator connecting one node to another, with weights on the arrows representing the value of the coefficient in the associated linear term. We then show how elementary row operations are performed on this grid representation, and show how back-substitution can quickly solve the linear system for operators with certain restrictions on the arrows that appear. We demonstrate a factorization of linear operators, similar to the LU factorization for regular matrices, but making use of the structure of the grid representation. In factored form, we can then solve the system of linear equations in $O(N)$ steps.

We then apply this factorization to the Laplace operator, and to a linear system that arises in the implicit method for solving the wave equation in 2D which was the original motivation for this work.

GRAPHS TO REPRESENT A LINEAR OPERATOR

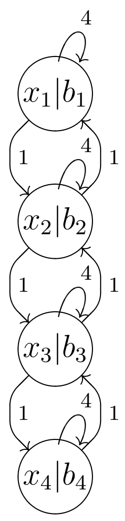
A simple matrix equation $Ax = b$ as in this example

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{1}$$

can be visualized as a bipartite graph. We place inputs x_j on the left and outputs b_k on the right, and arrows with weights connecting the inputs to outputs, as shown in Figure (2):

$$\begin{tikzcd}
x_1 \arrow[r, "4"] \arrow[dr, "1"] & b_1 \\
x_2 \arrow[r, "4"] \arrow[ur, "1"] \arrow[dr, "1"] & b_2 \\
x_3 \arrow[r, "4"] \arrow[ur, "1"] \arrow[dr, "1"] & b_3 \\
x_4 \arrow[r, "4"] \arrow[ur, "1"] & b_4
\end{tikzcd} \tag{2}$$

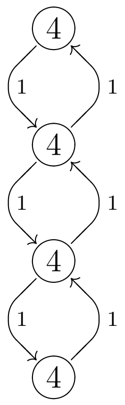
To illustrate more clearly the structure of this linear operator (in particular, the tridiagonal format), it is convenient to collapse the inputs and outputs into single nodes, and sketch the relationship as in the graph in Figure (3):



(3)

We read the diagram in Figure (3) as follows: at any node, the b_k is equal to the sum of all the arrows coming into the node, using the appropriate input x_j at the tail of the arrow, multiplied by the weight on the arrow. This includes the self-looping arrows (here with weight 4) which adds a diagonal term $4x_k$ to b_k.

To save space in the diagram, it is convenient to drop the parameters x_j, b_k and take them to be "understood." The diagonal weight can by written inside the node, for simplicity, and we get the graph shown in Figure (4):



(4)

An advantage of this graph representation is that we can easily see the nearest-neighbour dependence on the outputs, reflecting the tridiagonal structure of the original matrix.

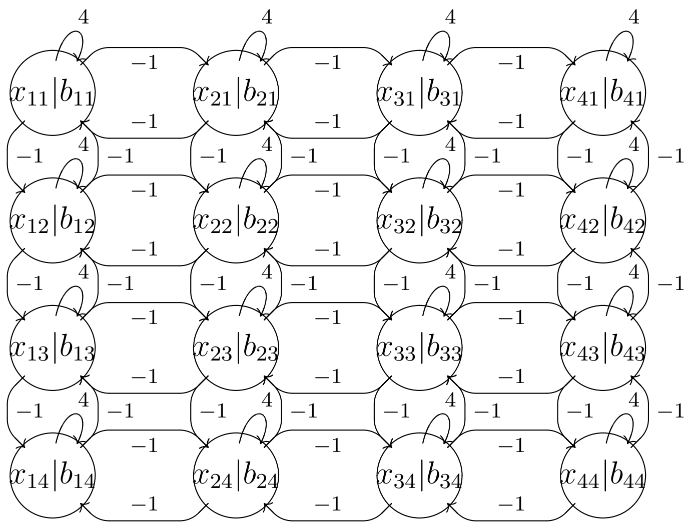
GRID REPRESENTATIONS FOR LINEAR OPERATORS

We will use a similar graphical representation for linear operators that occur in numerical methods for solving PDEs in 2D and 3D in order to develop algorithms that take advantage of the structure of the graph where possible.

In a 2D grid, we want to make sense of a grid diagram such the one shown in Figure (5), which represents a typical linear operator approximating Laplacians ∇^2:

$$\begin{tikzcd}
4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[d, leftrightarrow, "-1"'] \\
4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[d, leftrightarrow, "-1"'] \\
4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[r, leftrightarrow, "-1"] \arrow[d, leftrightarrow, "-1"'] & 4 \arrow[d, leftrightarrow, "-1"'] \\
4 \arrow[r, leftrightarrow, "-1"] & 4 \arrow[r, leftrightarrow, "-1"] & 4 \arrow[r, leftrightarrow, "-1"] & 4
\end{tikzcd} \tag{5}$$

Somewhat more verbosely, this grid diagram can be expanded as shown in Figure (6):



(6)

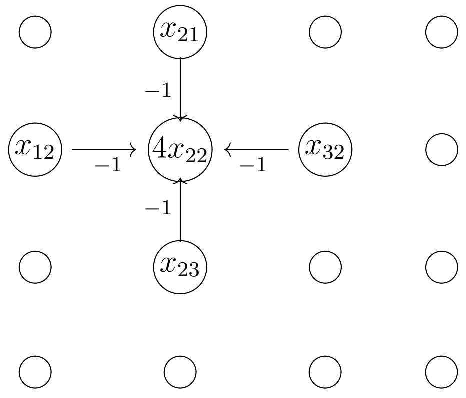
This represents a discrete approximation to the 2D Laplacian in the xy plane, yielding a system of equations in the form

$$b_{ij} = 4x_{ij} - x_{i-1,j} - x_{i,j+1} - x_{i-1,j} - x_{i+1,j}, \qquad \text{for } 1 \leq i, j \leq N.$$

For instance, the $(2,2)$ component of the discrete Laplacian is given by the formula

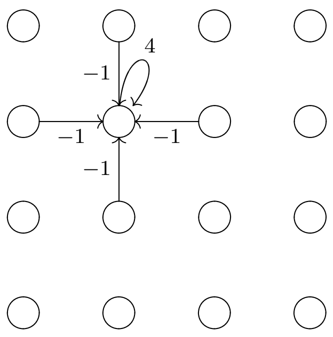
$$b_{22} = 4x_{22} - x_{21} - x_{23} - x_{12} - x_{32}.$$

This one equation is represented by the five coefficients in the graph, with four arrows pointing into the $(2,2)$ node, and the fifth coefficient from the diagonal term. The grid diagram for this one equation is represented by the four arrows in Figure (7):



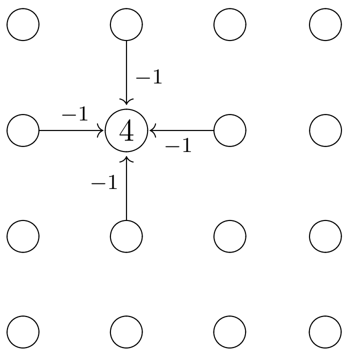
(7)

It is convenient to avoid writing out the nodes x_{ij} explicitly, and just indicate arrows and the associated weights. The one equation above can be represented with weighted arrows as in Figure (8):



(8)

Again, we can indicate the weight of the self-loop by simply marking it inside the node, as in Figure (9):



(9)

Any linear operator A acting on a 2D grid of data points x_{ij} can be represented in this manner, with arrows and weights, provided we allow for arrows that can point between any two nodes in the grid. A similar technique works for 3D grids, although the graphs are harder to sketch. The point is that certain linear operators, such as those with only nearest neighbour dependence, will have very simple grid structures in this representation. By comparison, the corresponding matrix representation will be sparse and structured (banded), but the structure doesn't easily reveal the nearest neighbour property, for instance.

In 3D, again we can see structures in the grid representation which will not be obvious in the usual matrix representation of the linear operator.

ELEMENTARY “ROW” OPERATIONS

In linear algebra, one uses row operations to reduce a matrix or system of linear equations to a simpler, but equivalent form. One can do the same with grid representations – just keep in mind, however, that the rows and columns in the grid have nothing to do with rows and columns in the corresponding matrix representation. A row in a matrix corresponds to all the arrows pointing towards one node in the grid (including the self-loop at that node), and the coefficients in the row of the matrix correspond to the weights on those arrows. Thus, the “row operations" on a matrix correspond to operations on collections of arrows that all point to one node in the grid.

For instance, the matrix row operation of multiplying a row by a constant is equivalent to the grid operation of multiplying (by that constant) all the weights on all arrows pointing into one node. Exchanging two rows in a matrix is equivalent to exchanging all the arrows that point into one node with all the arrows pointing into another node (keeping their tails the same). Adding one row to another in the matrix is equivalent to adding weights of all arrows pointing into one node onto the corresponding weights of arrows pointing into the second node, where two arrows “correspond” if their tails are at the same node.

Some care is needed for augmented matrices – basically, you have to keep track of the b_{ij}’s. In this paper, we won’t be doing anything very complex so we skip those details.

SOLVE $AX = B$ BY BACK-SUBSTITUTION

When the grid operator has certain forms, it is easy to solve the related linear system of equations $Ax = b$ using back-substitution just as is done in the case of lower triangular matrices in the tradition square matrix form. For instance, a grid operator where all the arrows go down and/or to the right is in a form that is quickly solved by back-substitution. Consider an example of the grid operator A represented in Figure (10):

$$\begin{tikzcd}
\textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[d, "-1"'] \\
\textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[d, "-1"'] \\
\textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[r, "-1"] \arrow[d, "-1"'] & \textcircled{2} \arrow[d, "-1"'] \\
\textcircled{2} \arrow[r, "-1"] & \textcircled{2} \arrow[r, "-1"] & \textcircled{2} \arrow[r, "-1"] & \textcircled{2}
\end{tikzcd} \tag{10}$$

Suppose we want to solve $Ax = b$ with output array b given in Eqn (11):

$$b = \begin{bmatrix} 2 & 3 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 3 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

To solve, first we augment the grid represtation to include the coefficients for b and write out explicitly the unknowns x_{ij} as well as the loops, shown in Figure (12):

$$\begin{tikzcd}
x_{11}|2 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{21}|3 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{31}|0 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{41}|1 \arrow[d, "-1"'] \arrow[loop above, "2"] \\
x_{12}|1 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{22}|1 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{32}|1 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{42}|1 \arrow[d, "-1"'] \arrow[loop above, "2"] \\
x_{13}|3 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{23}|2 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{33}|3 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & x_{43}|0 \arrow[d, "-1"'] \arrow[loop above, "2"] \\
x_{14}|0 \arrow[r, "-1"] \arrow[loop above, "2"] & x_{24}|0 \arrow[r, "-1"] \arrow[loop above, "2"] & x_{34}|0 \arrow[r, "-1"] \arrow[loop above, "2"] & x_{44}|0 \arrow[loop above, "2"]
\end{tikzcd} \tag{12}$$

The top-left corner of the grid represents the formula $2x_{11} = 2$ which we solve as $x_{11} = 1$. Moving one node to the right, we have $2x_{21} - x_{11} = 3$, and knowing already that $x_{11} = 1$, we find $2x_{21} - 1 = 3$, so $x_{21} = 2$. Moving to the $(1, 2)$ node we have $2x_{12} - x_{11} = 1$ for which we conclude $x_{12} = 1$.

Now at the $(2, 2)$ node we have $2x_{22} - x_{21} - x_{12} = 1$ which we know simplifies to $2x_{22} - 2 - 1 = 1$ and find $x_{22} = 2$. Continuing in this method, moving down and to the right, we can fill in all the unknowns to get the solved system shown in Figure (13):

$$\begin{tikzcd}
1|2 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 2|3 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 1|0 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 1|1 \arrow[d, "-1"'] \arrow[loop above, "2"] \\
1|1 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 2|1 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 2|1 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 2|1 \arrow[d, "-1"'] \arrow[loop above, "2"] \\
2|3 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 3|2 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 4|3 \arrow[r, "-1"] \arrow[d, "-1"'] \arrow[loop above, "2"] & 3|0 \arrow[d, "-1"'] \arrow[loop above, "2"] \\
1|0 \arrow[r, "-1"] \arrow[loop above, "2"] & 2|0 \arrow[r, "-1"] \arrow[loop above, "2"] & 3|0 \arrow[r, "-1"] \arrow[loop above, "2"] & 3|0 \arrow[loop above, "2"]
\end{tikzcd} \tag{13}$$

In this example, the arrows only moved at most by one cell down or to the right. But this back-substitution method will work for any grid whose arrows are of arbitrary length and direction, provided the directions are restricted to a particular half-plane. (Specifically, a half plane of arbitrary orientation.) For instance, if we have a grid array with arrows all going up and to the left, we could just start the back-substitution at the lower right corner, and similarly for other limited direction arrows.

Note this is much more general than in the case of the usual matrix representation where back-substitution only works for lower triangular matrices, or upper triangular matrices. Here, in 2D, we have many more directions to do the back-substitution. Also in 3D, there is much more flexibility, with directions limited by half spaces.

FACTORING GRID ARRAYS

The LU decomposition for matrices provides a method for fast solution of a linear system as the lower triangular matrix L gives a linear system that is quickly solved with back-substitution. The same is true for the upper triangular matrix U. A similar idea holds for grid array systems – if we can factor a grid array A into a product of linear operators $[DR] * [UL] = A$, where $[DR]$ is a grid array with only down-and-left arrows and $[UL]$ is a grid array with only right-and-up arrows, then we can solve the related system $Ax = b$ by two iterations of back substitution. (One with $[DR]$ and one with $[UL]$.)

More generally, one can factor the matrix into the product of several factors $F_1F_2F_3 \cdots F_n$, where each one has arrows restricted to some half plane. Then, each factor can be solved quickly by back substitution. So by n applications of substitution, we solve the original system. Again, note there is much more flexibility in the grid representation as there are many choices of half planes for the arrows.

EXAMPLE: APPROXIMATELY FACTORING THE LAPLACIAN

The Laplacian in 2D can be factored as follows:

$$\nabla^2 = (\frac{\partial}{\partial x})^2 + (\frac{\partial}{\partial y})^2 = (\frac{\partial}{\partial x} - i\frac{\partial}{\partial y})(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}). \tag{14}$$

Replacing the partial derivatives with finite differences, this suggests a discrete, factorized version for the Laplacian as

$$\nabla^2 = [\frac{(R-I)}{\Delta x} - i\frac{(U-I)}{\Delta y}][\frac{(I-L)}{\Delta x} + i\frac{(I-D)}{\Delta y}] \tag{15}$$

where R is the operator that shifts the grid to the right, U shifts up, L shifts left and D shifts down. The operator I is just the identity operator, and of course $UD = I, LR = I$ since the up and down shifts are inverses of each other as are the left and right shifts.

In this factored form, the first grid array operator has only up and right arrows so the related linear system can be solved by back-substitution. Similarly, the second operator has only down and left arrows so we can again solve by back-substitution. In our grid representations, the weights on the left/right arrows are $\pm 1/\Delta x$ while the weights on the up/down arrows are $\pm i/\Delta y$. The weights on the diagonal (self-loops) are $\pm 1/\Delta x + i/\Delta y$.

Via this back-substitution, with this method we can solve a discrete approximation to the Laplace equation

$$\nabla^2 u = f \tag{16}$$

with an order $O(N)$ method where N is the total number of grid points in the discrete approximation.

It is worth noting that we can expand the above product as

$$[\frac{(R-I)}{\Delta x} - i\frac{(U-1)}{\Delta y}][\frac{(I-L)}{\Delta x} - i\frac{(I-D)}{\Delta y}] = \tag{17}$$

$$[\frac{R-2I+L}{\Delta x^2} + \frac{U-2I+D}{\Delta y^2}] + i[\frac{R-I}{\Delta x}\frac{I-D}{\Delta y} - \frac{U-I}{\Delta y}\frac{I-L}{\Delta x}].$$

The real part is the central difference formula for the second partials $(\frac{\partial}{\partial x})^2 + (\frac{\partial}{\partial y})^2$ which is the discrete Laplacian we want. The imaginary part is the finite difference approximation to the difference of operators $\frac{\partial^2}{\partial x \partial y} - \frac{\partial^2}{\partial y \partial x}$ which will go to zero as the grid spacing reduces to zero since the mixed partials will be equal in the limit.

EXAMPLE: FACTORING THE THE LAPLACIAN, WITH BOUNDARY

The astute reader will have noticed that, in the case of a finite grid, we do not really have the identity $RL = I$ as a left shift followed by a right shift introduces some zeros at the boundary. In the 1D case, the product $(I-R)(I-L) \approx -R + 2I - L$ represents the central difference approximation to the second derivative which, in matrix form, is simply a tridiagonal matrix with 2's on the diagonal and -1's on the sub- and super-diagonal. Numerically, it is convenient to use a Cholesky factorization for this difference matrix.

Recall the Cholesky factorization for a symmetric matrix A expresses the matrix in the form

$$A = Q^T Q \tag{18}$$

where Q is an upper triangular matrix and Q^T is its transpose. For the central difference matrix, with 2's on the diagonal and -1's on the off-diagonal, the matrix Q has non-zero coefficients only on the diagonal and super-diagonal, with values $\sqrt{(j+1)/j}$ on the diagonal $(1 \leq j \leq n)$ and entries $-\sqrt{j/(j+1)}$ on the superdiagonal (($1 \leq j \leq n-1$).

We can use this factorization in both the horizontal and vertical directions in the 2D grid case. We write

$$\nabla^2 = -[\frac{Q_x^T}{\Delta x} - i\frac{Q_y^T}{\Delta y}][\frac{Q_x}{\Delta x} + i\frac{Q_y}{\Delta y}], \tag{19}$$

where the Q_x represents the Cholesky factor in the x-direction, and Q_y the factor in the y-direction. As in the last example, the first factor only has up and right arrows in the grid representation, and the second factor has only down and left arrows. Thus, back-substitution, done once on each factor, quickly solves the Laplace equation in $O(N)$ steps where N is the total number of grid points.

To be a bit more explicit in the grid algebra representation, the horizontal arrows have weights $\pm\sqrt{j/(j+1)}/\Delta x$ while the vertical arrows have weights $-i\sqrt{k/(k+1)}/\Delta y$.

(Here, j is the grid index in the x-direction and k is the index in the y-direction.) The diagonal weights (self-loops at nodes) will have weights $\pm\sqrt{(j+1)/j}/\Delta x + i\sqrt{(k+1)/k}/\Delta y$.

MATLAB VERIFICATION

We used MATLAB to verify that we can solve the equation

$$\nabla^2 u = f \tag{20}$$

quickly, in $O(N)$ steps, using the factorization and back-substitution mentioned above. Of course the linear operator has an inverse with large norm so the numerical behaviour is challenging.

IMPLICIT FD SOLUTION TO THE WAVE EQUATION

This section is the central point of the paper as it forms the motivation for looking at grid algebra.

We know that solving the implicit finite difference equations for the wave equation involves solving some linear system, as seen in any book on numerical solutions of PDEs, such as (Myint-U and Debnath, 2007). In the 1D wave equation, we get a tridiagonal matrix so the solution is fast. Indeed, the matrix can be factored in an LU decomposition where the matrices are in fact bi-diagonal. So the solution is $O(N)$ and thus, very fast to compute.

In the 2D wave equation (and 3D), one obtains a sparse, structured matrix that has to be solved. Using grid algebra is our attempt to do this in a fast way.

We follow the development in (Myint-U and Debnath, 2007) for a finite difference implicit method for the wave equation, adjusting their 1D problem to the 2D problem here. Start with the acoustic wave equation, constant velocity c for simplicity, write

$$\frac{\partial^2 u}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{21}$$

and place the problem on a grid with fixed spacings $\Delta t, \Delta x, \Delta y$. Evaluate u on the grid to get elements u_{ij}^k where k is the time index, and ij is the spatial index. Use central difference formulas to approximate the second derivatives. In order to get an implicit method, we compute the spatial derivatives at times $k+1, k-1$ and take their average.

This gives the time stepping formulas

$$\begin{aligned} \frac{1}{\Delta t^2}(u_{ij}^{k+1} - 2u_{ij}^k + u_{ij}^{k-1}) = \\ \frac{c^2}{2}\left[\frac{1}{\Delta x^2}(u_{i+1,j}^{k+1} - 2u_{ij}^{k+1} + u_{i-1,j}^{k+1}) + \frac{1}{\Delta x^2}(u_{i+1,j}^{k-1} - 2u_{ij}^{k-1} + u_{i-1,j}^{k-1})\right] \quad + \\ \frac{c^2}{2}\left[\frac{1}{\Delta y^2}(u_{i,j+1}^{k+1} - 2u_{ij}^{k+1} + u_{i,j-1}^{k+1}) + \frac{1}{\Delta y^2}(u_{i,j+1}^{k-1} - 2u_{ij}^{k-1} + u_{i,j-1}^{k-1})\right]. \end{aligned} \tag{22}$$

Grouping all the elements at time $k+1$ onto the left-hand side, we get the formula for implicit stepping as

$$2(1+\epsilon_x^2+\epsilon_y^2)u_{ij}^{k+1} - \epsilon_x^2(u_{i+1,j}^{k+1}+u_{i-1,j}^{k+1}) - \epsilon_y^2(u_{i,j+1}^{k+1}+u_{i,j-1}^{k+1}) = \\ 4u_{ij}^k + \epsilon_x^2(u_{i+1,j}^{k-1}+u_{i-1,j}^{k-1}) + \epsilon_y^2(u_{i,j+1}^{k-1}+u_{i,j-1}^{k-1}) - 2(1+\epsilon_x^2+\epsilon_y^2)u_{ij}^{k-1}, \tag{23}$$

where we define the small parameters $\epsilon_x^2 = (c\Delta t/\Delta x)^2$ and $\epsilon_y^2 = (c\Delta t/\Delta y)^2$ as usual.

It is the left hand side of this equation that forms the sparse linear system that needs to be solved; so we will factor it using the grid algebra. We recognize in this expression the left and right shift operators, as well as the up and down operators, so we attempt to solve

$$2(1+\epsilon_x^2+\epsilon_y^2)I - \epsilon_x^2(R+L) - \epsilon_y^2(U+D) = [(a+bR)+i(c+dU)][(a+bL)-i(c+dD)]. \tag{24}$$

Multiply out the right hand side, ignore the imaginary part (for now) and we get the system of equations

$$\begin{aligned} a^2+b^2 &= 1+2\epsilon_x^2 \\ ab &= -\epsilon_x^2 \\ c^2+d^2 &= 1+2\epsilon_y^2 \\ cd &= -\epsilon_y^2. \end{aligned} \tag{25}$$

Letting $a = \sqrt{1+2\epsilon_x^2}\cos\theta, b = \sqrt{1+2\epsilon_x^2}\sin\theta$, we find $\sin 2\theta = -2\epsilon_x^2/(1+2\epsilon_x^2)$. Hence, an arcsine computation gives us a, b and a similar computation works for c, d.

So we have factored the sparse linear system in the time stepping to a product of two grid operators which are solved quickly using back-substitution. The result is an $O(N)$ computation for each time step where N is the total number of spatial grid points. Note this is a 2D array; so for a grid size of $N_1 \times N_2$ points, the method is $O(N_1 * N_2)$.

MATLAB VERIFICATION

This is a work in progress. Our first attempt to implement the implicit solution method using a grid algebra results in a wave propagation example as shown in Figure 26. Clearly there are problems in this result – the solution is not a symmetric waveform even though the initial conditions were symmetric. And the waveform has grown in amplitude.

This is a bit discouraging, but the speed is verified to be linear in number of grid points, and perhaps with a bit more care in the coding, we may achieve something that works. We hope to report on this by the time of the CREWES meeting in December.

CONCLUSIONS

Grid representation of linear operators provides a technique to manipulate sparse linear operators that appear in numerical methods for partial differential equations that allow us to take advantage of the spatial structure within these operators. Many of the usual methods of linear algebra (row reduction, back-substitution, LU decomposition) can be implemented in the grid representation form. We can generate efficient algorithms for solving these numerical problems by making use of the grid structure directly. We are demonstrating these methods specifically in creating implicit methods for finite difference solutions to the wave equation in two and three dimensions.

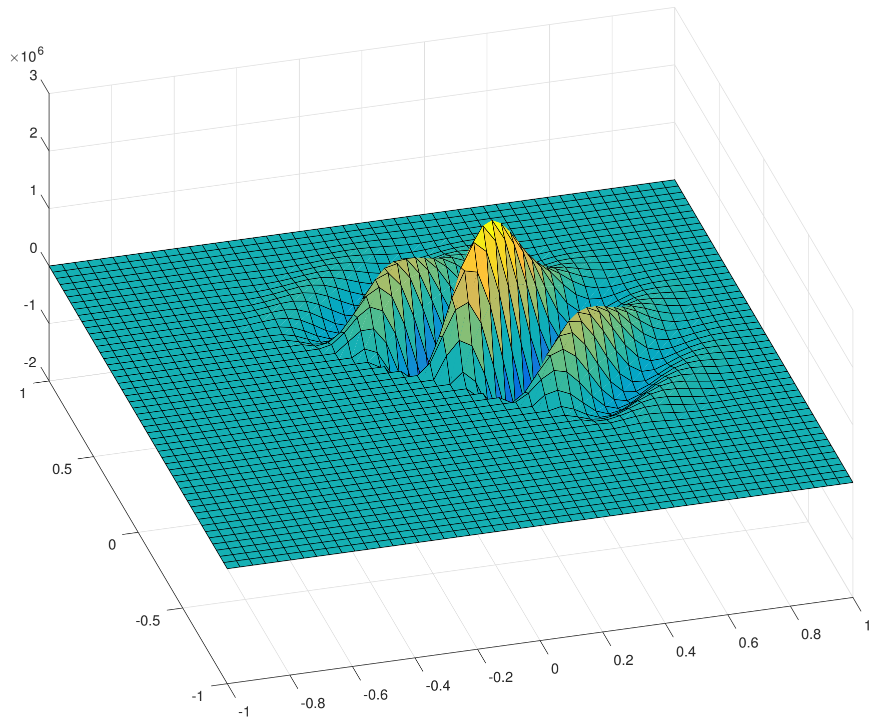


FIG. 26. A two dimension wave using the grid algebra method.

ACKNOWLEDGEMENTS

We thank the sponsors of CREWES for their support. We also gratefully acknowledge support from NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13 and through a Discovery Grant for the first author.

REFERENCES

Myint-U, T., and Debnath, L., 2007, Linear Partial Differental Equations for Scientists and Engineers: Birkhäuser.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., 2007, Numerical Recipes: The Art of Scientific Computing: Cambridge University Press.