On Software Support for Finite Difference Schemes Based on Index Notation

Krister Åhlander and Kurt Otto

Department of Scientific Computing, Uppsala University,
Box 120, 751 04 Uppsala, Sweden,
{krister,kurt}@tdb.uu.se

Abstract. A formulation of finite difference schemes based on the index notation of tensor algebra is advocated. Finite difference operators on regular grids may be described as sparse, banded, "tensors". Especially for 3D, it is claimed that index notation better corresponds to the inherent problem structure than does conventional matrix notation.
The transition from mathematical index notation to implementation is discussed. Software support for index notation that obeys the Einstein summation convention has been implemented in the C++ package EinSum. The extension of EinSum to support typical data structures of finite difference schemes is outlined. A combination of general index notation software and special-purpose routines for instance for fast transforms is envisioned.

1 Introduction

Basic finite difference schemes for partial differential equations on structured grids are, in principle, simple to implement. If we oversimplify, the process is as follows. The derivative at a certain point of the grid is approximated with a particular finite difference stencil. For example, if we consider the Laplace equation in 2D, a second-order finite difference scheme yields the well-known five-point stencil. If curvilinear grids are used, the stencil weights vary at every point of the grid. A relation between grid functions (discrete approximations of fields) is thus obtained. This relation is usually expressed as a matrix vector multiplication, where each row of the matrix corresponds to a stencil at a specific grid point. The resulting matrix is a band matrix, whose structure can be exploited in order to develop efficient algorithms. Since linear algebra is such a well-developed area of research, both with respect to mathematical aspects as well as to efficient computer programs, the restructuring of the original formulation into a matrix vector multiplication has several advantages.

However, the original formulation relates two grid functions through a stencil at each point of the grid. This data structure can be described as a "grid stencil". Such a grid stencil may of course be implemented as a sparse band matrix, but in this process the conceptual picture of the grid stencil is obscured. This potential risk of missing information makes it more difficult to express finite

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2331, pp. 711–718, 2002.

difference schemes mathematically, particularly in 2D and 3D. It also makes the implementation of finite difference schemes more awkward.

In this paper, we discuss an alternative approach to finite difference schemes, which better takes into account the inherent structure of the problem. This formulation is based on index notation, including the Einstein summation convention. We also study software design based on this formulation. Particularly, we discuss the extension of EinSum to support typical data structures of finite differences.

EinSum is a C++ package for index notation. It allows index notation expressions to be written as plain code, thereby avoiding a notational gap between index notation and implementation code. The EinSum package is well suited for tensor algebra. It has been presented in more detail in [1], and its support of tensors with general symmetries is described in [2]. In the present paper, we focus on an extension to general sub and super diagonals (bands) in multi-dimensional arrays (tensors).

The remainder of this paper is outlined as follows. In Section 2, we exemplify index notation for finite differences. In Section 3, we discuss software support for index notation. Finally, in Section 4, we discuss future work and the relation between different software implementation strategies. We find that general index notation support is useful, for instance for expressing the finite difference operator. To address performance, it should be combined with special-purpose software for performance critical tensor operations.

2 Index notation and finite differences

2.1 Introductory examples

Band matrices arise in finite difference schemes, for instance for solving PDEs in one space dimension. In higher dimensions, a more appropriate data structure is a "band tensor". For instance, the well known five-point stencil D can be interpreted as a $(2;2)$ "tensor", i.e., a multi-dimensional array with 2 upper and 2 lower indices. It operates on a gridfunction x with two upper indices, i.e., a $(2;0)$ tensor, to produce another $(2;0)$ tensor y. In index notation, where the Einstein summation convention is adopted, this operation reads

$$y^{i,j} \leftarrow D^{i,j}_{k,\ell} x^{k,\ell}. \tag{1}$$

Here, the summation convention implies summation over k and ℓ, since they are repeated indices (see Section 3).

The nonzero elements of D are:

$$\begin{aligned} D^{i,j}_{k,\ell} &= -4,\ i = k \text{ and } j = \ell, \\ D^{i,j}_{k,\ell} &= 1,\ i = k \text{ and } j = \ell \pm 1, \\ D^{i,j}_{k,\ell} &= 1,\ i = k \pm 1 \text{ and } j = \ell. \end{aligned}$$

Using the conventions (see Section 3) that summation is always suppressed on the left-hand side in assignments, and that free indices vary over their range, we may assign the nonzero elements of D as

$$\begin{aligned} &D^{i,j}_{i,j} \leftarrow -4, \\ &D^{i,j}_{i,j+1} \leftarrow 1, \quad D^{i,j}_{i,j-1} \leftarrow 1, \\ &D^{i,j}_{i+1,j} \leftarrow 1, \quad D^{i,j}_{i-1,j} \leftarrow 1. \end{aligned} \tag{2}$$

Another way of forming the five-point stencil is as follows. Let the nonzero elements of Δ_+ and Δ_- be

$$\begin{aligned} [\Delta_+]^i_{i+1} \leftarrow 1, \quad [\Delta_+]^i_i \leftarrow -1, \\ [\Delta_-]^i_i \leftarrow 1, \quad [\Delta_-]^i_{i-1} \leftarrow -1. \end{aligned}$$

The five-point stencil is then given by

$$D^{i,j}_{k,\ell} \leftarrow [\Delta_+]^i_m [\Delta_-]^m_k + [\Delta_+]^j_n [\Delta_-]^n_\ell. \tag{3}$$

For curvilinear grids, the band tensors are no longer constant along the diagonals. As an illustration, consider a two-dimensional transformation $(x, y) = (x(\xi, \eta), y(\xi, \eta))$ and let ξ_x denote $\partial\xi/\partial x$ etc. By the chain rule,

$$\frac{\partial}{\partial x} = \xi_x \frac{\partial}{\partial \xi} + \eta_x \frac{\partial}{\partial \eta}.$$

Using the centred difference $\Delta_0 = \frac{1}{2}(\Delta_+ + \Delta_-)$ the corresponding differentiation operator Δ_x may be expressed as:

$$[\Delta_x]^{i,j}_{k,\ell} \leftarrow {\xi_x}^{i,j} [\Delta_0]^i_k + {\eta_x}^{i,j} [\Delta_0]^j_\ell. \tag{4}$$

In summary, finite difference operators may often conveniently be expressed as band tensors. Moreover, index notation provides a useful way to manipulate with the operators as well as the "grid functions".

2.2 Band tensors for the Helmholtz equation

In this subsection, we illustrate the use of index notation as an efficient tool for realistic problems. This section is based mainly upon results in [8–10].

Consider the Helmholtz equation,

$$u_{xx} + u_{yy} + \kappa^2 u = g,$$

for a 2D duct, see Figure 1. Using an orthogonal transformation, the equations may be rewritten for computational coordinates (ξ, η):

$$(a u_\xi)_\xi + (a^{-1} u_\eta)_\eta + e u = g.$$

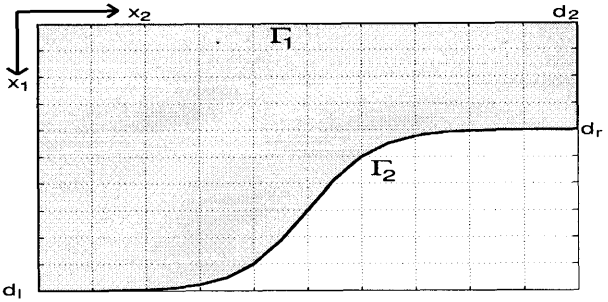

Fig. 1. The Helmholtz equation for the following duct is an application of interest in underwater acoustics.

Here, a and e are coefficients that depend on the transformation. A fourth-order Numerov scheme may be used (see [9]), to express the finite difference scheme in index notation as

$$B^{i,j}_{k,\ell} u^{k,\ell} = g^{i,j}. \tag{5}$$

In the interior, B is a band tensor with nine diagonals.

The system may be solved iteratively using a Krylov subspace method with a normal block preconditioner. Here, we sketch the process of forming the preconditioner, using index notation. The theory is developed in [8], using the considerably more complex matrix notation.

Choose a unitary $(1;1)$ tensor Q with components Q^i_j. Consider partially diagonal $(2;2)$ tensors Λ such that

$$\Lambda^{i,j}_{k,\ell} = 0, \ i \neq k.$$

Compose a $(2;2)$ preconditioner $M^{i,j}_{k,\ell} \leftarrow Q^i_\alpha \Lambda^{\alpha,j}_{\alpha,\ell} [Q^*]^\alpha_k$.

The choice of Λ that minimizes the Frobenius norm of $(B - M)$ is given by

$$\Lambda^{i,j}_{i,\ell} \leftarrow [Q^*]^i_\alpha B^{\alpha,j}_{\beta,\ell} Q^\beta_i. \tag{6}$$

(Note that summation over i is suppressed due to the range and the assignment conventions, see Section 3.)

Equation (5) may now be solved efficiently using M as a preconditioner. The preconditioning step can be performed cheaply, using fast transforms [10].

3 Software support

The conclusion of the previous section is that index notation and tensor-like data structures are convenient for expressing finite difference methods. Consequently, we investigate software support for such operations. This allows for a smooth transition from the mathematical formulation to the implementation of finite difference methods. In this section, we focus on index notation support in the EinSum package. The design of EinSum has been presented in more detail in [1, 2].

Firstly, EinSum grammar is based on the following well-known conventions (see, e.g., [11]):

1. The summation convention: If an index is repeated, both as an upper index and as a lower index, summation is understood.
2. The range convention: A free index is understood as a loop index, ranging over all its values.

EinSum is originally developed for supporting common tensor algebra operations. For example, a contraction of a $(3;1)$ tensor D may be assigned to a $(2;0)$ tensor C as

$$C^{i,j} \leftarrow D_k^{i,k,j}.$$

In EinSum, the corresponding assignment is straightforward. First, some indices are declared, with an approriate range N according to the dimension.

```
EinIndex I(1,N), J(1,N), K(1,N), L(1,N);
```

Next, the tensors are declared with references to appropriate index spaces. Index spaces (without symmetry properties) may be described by "juxtaposing" indices, using the `operator|`. See [2] for more details on index space construction. Below, `D` is associated with an index space of rank 4, whose elements range from $(1,1,1,1)\ldots(N,N,N,N)$ and `C` with an index space of rank 2, whose elements range from $(1,1)\ldots(N,N)$. Furthermore, `D` and `C` are declared as $(3;1)$ and $(2;0)$ tensors, respectively.

```
EinTensor<double> D(I|J|K|L,3,1), C(I|J,2,0);
```

After data initialization, the assignment may now be carried out:

```
C(I|J) = D(I|K|J,K);
```

EinSum requires tensors to be indexed according to their rank. For `C`, as a tensor with no lower indices, `operator()` accepts one argument. `D` must be indexed with two arguments, "multi-indices" of rank three and one, respectively. Again, `operator|` is used for juxtaposition. When the assignment is carried out, EinSum interprets the expression and recognizes that there is an implicit loop over indices `I` and `J`, as well as an implicit summation over `L`. See [1] for more details on the interpretation of the EinSum notation. This includes more complex cases, and also the introduction of a third convention:

3. The assignment convention: Summation is suppressed in the left-hand side of assignments.

This convention yields a natural and intuitive notation for assignments along diagonals. Consider, for instance, the assignments

$$A_i^i \leftarrow B_i^i,\ A_i^i \leftarrow B_j^j,\ A_j^i \leftarrow B_j^i.$$

According to the conventions, all these assignments have different meaning. The first statement assigns the diagonal values of B to the corresponding diagonal elements of A. The second statement assigns the trace of B, i.e., B^j_j. The third statement assigns all elements of B to all elements of A, a "block assignment". All three statements are readily coded in EinSum:

```
A(I,I) = B(I,I);  A(I,I) = B(J,J);  A(I,J) = B(I,J);
```

Compare with for instance Matlab or Fortran notation, or other libraries with some kind of index abstraction, for instance [5, 6, 12]. In these languages or libraries, only the block assignment is directly supported. We argue that the notation supported in EinSum is more powerful, in particular concerning the ability to address diagonals for tensors of higher order.

In the present contribution, we extend these ideas so that we are also able to treat generalized sub and super diagonals, as motivated for instance by equation (2). A crucial point in the implementation is the identification of indices. To this end, each index has a unique ID, automatically generated. In order to handle sub and super diagonals, we need to slightly modify this technique. For instance, i and $i+1$ are two expressions which refer to the same ID, but whose ranges differ. It is achieved by overloading `operator+` to take an integer value and return a new index object with the same ID as the original, but with the range properly adjusted. This makes it possible to initiate the five-point stencil according to equation (2) as follows:

```
EinTensor D(I|J|K|L,2,2);

D(   I|J, I|J ) = -4;
D( I+1|J, I|J ) =  1;  D( I-1|J, I|J ) =  1;
D( I|J+1, I|J ) =  1;  D( I|J-1, I|J ) =  1;
```

Of course, equation (3) provides another way of creating D (assuming that Δ_+ and Δ_- are properly initiated):

```
D(I|J,K|L) = DeltaPlus(I,M)*DeltaMinus(M,K) +
             DeltaPlus(J,N)*DeltaMinus(N,L);
```

Application of the difference operator, see equation (1), may be carried out using the already implemented interpretation mechanisms of EinSum. It reads:

```
Y(I|J) = D(I|J,K|L)*X(K|L);
```

Generalized tensor diagonals are also present when simulating the Helmholtz equation. The difference operator B is a banded tensor, and the computation of Λ, see equation (6), can be written as one line of EinSum notation:

```
Lambda(I|J,I|L) = Qstar(I,Alpha)*B(Alpha|J,Beta|L)*Q(Beta,I);
```

The summation is over α and β, in accordance with the conventions.

4 Discussions and future work

We have discussed index notation as an alternative to matrix formulation in order to express finite difference algorithms for simulating PDEs. For structured grids, we find that index notation better corresponds to the data structures of the problem. These data structures are multi-dimensional arrays and may be compared with tensors. For finite differences, the sparsity pattern of typical stencils generates band tensors.

Support of tensor notation has already been developed in the EinSum package. In the present paper, we stress that it can be used also for expressing finite difference schemes. In particular, the support for an intuitive notation for generalized sub and super diagonals has been presented. The current implementation does not utilize the sparsity patterns of the band tensors, though. This may be addressed firstly by extending the index space hierarchy of EinSum. As explained in [2], index spaces contain information for instance about structurally implied zero elements. It should not be difficult to develop a new index space with relevant information for banded tensors. The idea of banded index spaces is actually similar to the stencil structures found in for instance PETSc, [4]. An advantage with banded index spaces is the potential for tailoring them according to the particular stencil.

It is, however, more difficult to exploit the sparsity pattern of band tensors with respect to algorithms, when a notation as general as index notation is used. In addition, for some operations other performance improving techniques may be available, which are difficult to cater for in a general package. For example, as part of the preconditioning step for the Helmholtz equation, the following unitary transformations are carried out (see [10]):

$$v^{i,j} \leftarrow Q^i_k y^{k,j}.$$

This is computed according to a fast transform algorithm, using special-purpose software.

As often is the case in scientific computing, there is a trade-off between general packages and specialized routines. Our aim is to utilize the best of both worlds. We believe that the simulation of the Helmholtz equation (5), for example, can benefit from a general index notation library, particularly in the assembly process of the finite difference operator B. Also, as was noted in [7], preconditioners for this and similar problems would gain from a general library with index notation support. However, regarding the number-crunching, highly optimized routines should be used. Even though these may be less general, their design should still be based upon the data structures of the problem, see [3]. For finite differences, we believe that a general band tensor abstraction is such a data structure.

References

1. K. Åhlander. Einstein summation for multi-dimensional arrays. In E. Munthe-Kaas et al., editors, *Norsk Informatikk Konferanse – NIK'2000*, pages 67–78. Tapir,

Norway, 2000.

2. K. Åhlander. Supporting tensor symmetries in EinSum. Technical Report 212, Dept. of Informatics, University of Bergen, Bergen, Norway, June 2001. To appear in Computers and Mathematics with Applications.
3. K. Åhlander, M. Haveraaen, and H. Munthe-Kaas. On the role of mathematical abstractions for scientific computing. In R. Boisvert and P. Tang, editors, *The Architecture of Scientific Software*, pages 145–158. Kluwer Academic Publishers, Boston, 2001.
4. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object-oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 163–202. Birkhäuser, 1997.
5. J.C. Cummings et al. Rapid application development and enhanced code interoperability using the POOMA framework. In S. L. L. M. E. Henderson and C. R. Anderson, editors, *Object-oriented Methods for Interoperable Scientific and Engineering Computing*, Philadelphia, 1999. SIAM. ch. 29.
6. M. Lemke and D. Quinlan. P++, a parallel C++ array class library for architecture-independent development of structured grid applications. *ACM SIGPLAN Notices*, 28(1):21–23, 1993.
7. E. Mossberg, K. Otto, and M. Thuné. Object-oriented software tools for the construction of preconditioners. *Sci. Programming*, 6:285–295, 1997.
8. K. Otto. A unifying framework for preconditioners based on fast transforms. Report 187, Dept. of Scientific Computing, Uppsala Univ., Uppsala, Sweden, 1996.
9. K. Otto. Iterative solution of the Helmholtz equation by a fourth-order method. *Boll. Geof. Teor. Appl.*, 40 suppl.:104–105, 1999.
10. K. Otto and E. Larsson. Iterative solution of the Helmholtz equation by a second-order method. *SIAM J. Matrix Anal. Appl.*, 21:209–229, 1999.
11. J.G. Papastavridis. *Tensor calculus and analytical dynamics.* Library of engineering mathematics. CRC Press LLC, 1999.
12. T. Veldhuizen. Arrays in Blitz++. In *Proceedings of the 2nd International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, Lecture Notes in Computer Science. Springer-Verlag, 1998.