Multiplicable Discrete Operators In Finite Volume Solvers

DRAGAN VIDOVIĆ
Institute For The Development
Of Water Resources
"Jaroslav Černi"
Jaroslava Černog 80,
11226 Pinosava, Belgrade
SERBIA
draganvid@gmail.com

MILENKO PUŠIĆ
Department For Hydrogeology
Faculty For Mining And Geology
Đušina 7,
11000 Belgrade
SERBIA
mpusic@ptt.rs

MILAN DIMKIĆ
Institute For The Development
Of Water Resources
"Jaroslav Černi"
Jaroslava Černog 80,
11226 Pinosava, Belgrade
SERBIA
jdjcerni@jcerni.co.rs

Abstract— This paper presents a technique for implicit PDE solver implementation using multiplicable discrete analogues of first-order differential operators and constitutive relations. The technique is intended for mimetic discretizations, but may be used for other methods as well. As a model problem, Laplace equation is solved using this technique and the finite volume method. Several mimetic methods to reconstruct the flux in mesh faces have been proposed, as well as a method to reconstruct the node velocity.

Keywords— Mimetic discretizations, Discrete operators, Finite volumes, Groundwater, Laplace equation

I. INTRODUCTION

Most partial differential equations (PDEs) arising from physics are combinations of first-order differential operators and constitutive relations. Commonly used differential operators are the divergence, the gradient, and the curl, while examples of constitutive relations are Fick's law, Darcy's law, or Hooke's law.

Differential operators satisfy integral identities closely related to conservation laws of continuum mechanics, such as Gauss theorem. Support operator method (SOM) [12], [13], [6], [5], [7] uses such an integral identity to derive a discrete analog of a differential operator that satisfies the same identity. Then an orthogonality relation is used to obtain a dual operator.

Discrete difference operators constructed in this way are sometimes called *mimetic* because they mimic symmetry properties of their continuum counterparts. These symmetry properties allow to prove exact discrete analogues of conservation laws, and the convergence of the resulting numerical schemes [3], [10], [17].

The requirement that discrete difference operators result from integral identities leads to another requirement that, instead of directly discretizing additive PDE terms, which may be combinations of differential operators and constitutive relations, first-order difference operators and discrete versions of constitutive relations are combined by means of operator multiplication to obtain discrete versions of additive terms. Even discrete operators that were not constructed using SOM are often presented as multiples of simpler operators.

While this guideline is often followed when deriving discretization schemes, software tools for solving PDEs do not offer a possibility to combine basic difference operators and constitutive relations by multiplying them. Instead, they allow combining configurable additive terms.

Although this approach is flexible enough for most engineering applications, researchers developing numerical schemes could benefit from multiplicable discrete operators. Assembling a sparse computational matrix of an implicit PDE solver is a complicated task, and this is the reason why explicit Euler method, which does not use a computational matrix, is still used by many researchers, even though it imposes a limit on the time step which is often prohibitive. Writing a code that assembles a computational matrix is simplified by using multiplicable operators, and this way the time needed to develop a pilot code can be decreased.

Discretization of a linear continuous operator leads to a linear discrete operator, but it operates on a discrete field a part of which is fixed by boundary conditions. Seen as an operator on the unknown part of its operand field only, the discrete linear operator becomes affine. Such affine operators can still be combined by multiplication, and this has been investigated in this paper.

While matrix-based multiplicable operators may add some small overhead in the computational time, such overhead is often acceptable in a pilot code, and it is not comparable to the enormous overhead added by explicit Euler time discretization.

In transient problems, assembling a computational matrix typically takes time which is comparable to the computation of one time step. In problems where the matrix is computed only once because it does not change in time, some overhead in the time needed to assemble the matrix is often acceptable.

Memory overhead added by this technique can be significant. In our program, assembling the linear system takes approximately 10% more memory then solving it. Memory requirements can be reduced if matrix-free implementation is used for some of the multiplicable operators.

The SOM method was briefly presented in Section II. Section III analyses the requirements of the assembling process on a model problem. Section IV presents some implementation details, as well as the key parts of the code. Section V presents numerical results. A method to reconstruct the nodal velocity vectors for visualization purposes was presented in Section VI. Accuracy of the scheme was discussed in Section VII. Sections VIII and IX present methods for piecewise constant and piecewise linear flux reconstruction respectively, leading to improved accuracy. A method to solve the reconstruction systems was presented in Section X. Conclusion was made in Section XI.

II. Support operator method

The following integral identity

$$\int_\Omega u\,\mathbf{div}\,\mathbf{w}d\Omega+\int_\Omega \mathbf{w}\cdot\mathbf{grad}\,u d\Omega=\oint_{\partial\Omega}u(\mathbf{w}\cdot\mathbf{n})d\Gamma,\quad(1)$$

where u is a scalar field, $\mathbf{w}$ is a vector field, and $\mathbf{n}$ is the outer normal on the domain boundary $\partial\Omega$, implies that the gradient is the negative adjoint of the divergence if the boundary contribution vanishes:

$$\mathbf{grad}=-\,\mathbf{div}^*\qquad(2)$$

The definition of the divergence operator is extended to $\partial\Omega$ in SOM [5] , and the usual scalar product is replaced with the one that includes a boundary integral. As a result, equation (2) holds even if the boundary integral does not vanish.

In this way, operator $\mathbf{grad}$ may be defined in terms of operator $\mathbf{div}$, or vice versa.

Similar approach can be applied to the curl operator. For details, see [5].

III. Model problem

As an illustration, a boundary value problem describing stationary flow in a porous medium is considered:

$$\begin{aligned}\mathbf{div}(\mathbf{K}\,\mathbf{grad}\,h)=0 &\quad \text{on } \Omega,\\ h=h_b &\quad \text{on } \Gamma_1,\\ -(\mathbf{K}\,\mathbf{grad}\,h)\cdot\mathbf{n}=u_n &\quad \text{on } \Gamma_2,\end{aligned}\qquad(3)$$

where $\mathbf{K}$ is the conductivity matrix, h is the hydraulic potential, h_b and u_n prescribe respectively the potential and the outflow rate at the boundary, and $\partial\Omega$ is an exclusive union of Γ_1 and Γ_2. Mathematical representation of many other physical processes takes the same form, including heat transfer through a solid medium. More information on numerical simulation of flow in porous medium can be found in [4].

Problem (3) was discretized using one of the standard variants of the finite volume method. Unknown hydraulic potential is represented by it point values in cells centers. The discrete gradient operator $\overline{\mathbf{GRAD}}$ computes the differences of these point values in neighboring cells, subtracting the potential in the cell with the lower cell number from the potential in the cell with the higher cell number. At $\partial\Omega$, gradient operator computes the difference between the potential at the boundary face center and at the boundary cell center. More information on finite volume methods can be found in [16].

The discrete divergence operator $\mathbf{DIV}$ is defined using the divergence theorem:

$$\mathbf{DIV}\,\mathbf{u}_V=\int_V \mathbf{div}\,\mathbf{u}dV=\oint_{\partial V}\mathbf{u}\cdot\mathbf{n}d\Gamma=\sum_i\delta_i u_i,\quad(4)$$

where V is a computational cell, u_i is the flux through face i from the cell with the lower cell number to the cell with the higher cell number, and $\delta_i=1$ if the outer cell has a lower cell number than V or if i is a boundary face, otherwise $\delta_i=-1$.

Following the SOM approach, operator $\mathbf{DIV}$ is extended to the boundary:

$$\overline{\mathbf{DIV}}\mathbf{u}=\begin{cases}\mathbf{DIV}\,\mathbf{u} & \text{inside } \Omega\\ -\int_A u_n d\Gamma & \text{at } \partial\Omega\end{cases},\qquad(5)$$

where A is a boundary face.

The divergence and the gradient defined in this way satisfy an equivalent of (2):

$$\overline{\mathbf{GRAD}}=-\overline{\mathbf{DIV}}^*\qquad(6)$$

This method does not fully follow the SOM approach because $\overline{\mathbf{DIV}}$ and $\overline{\mathbf{GRAD}}$ do not approximate the continuous $\mathbf{div}$ and $\mathbf{grad}$ operators. If $\overline{\mathbf{DIV}}$ is scaled by the volume of the computational cell, and if $\overline{\mathbf{GRAD}}$ is scaled by the distance between the neighboring cell centers, then these two operators approximate their continuous counterparts. However, (6) allows to use the adjoint operation to construct the discrete gradient operator, like this is done in SOM.

Written in the matrix form, operator $\overline{\mathbf{DIV}}$ has the following structure:

$$\overline{\mathbf{DIV}}=\begin{bmatrix}\mathbf{D}_1 & \mathbf{D}_2\\ \mathbf{D}_3 & \mathbf{D}_4\end{bmatrix}.\qquad(7)$$

If $\overline{\mathbf{U}}$ is the vector of fluxes through the mesh faces, then

$$\mathbf{DIV}\,\overline{\mathbf{U}}=\mathbf{D}_1\mathbf{U}+\mathbf{D}_2\mathbf{U}_N,\qquad(8)$$

where $\mathbf{U}_N$ contains the prescribed fluxes through the Γ_2 faces, while $\mathbf{U}$ contains all other (unknown) fluxes. Equation (8) defines an affine operator acting on $\mathbf{U}$ that will be denoted by $\mathbf{Div}$:

$$\mathbf{Div}\,\mathbf{U}\equiv\mathbf{D}_1\mathbf{U}+\mathbf{D}_2\mathbf{U}_N,\qquad(9)$$

Operators $\mathbf{D}_3$ and $\mathbf{D}_4$ compute the divergence at the boundary as defined by the extension in (5).

Equations (6) and (7) define the structure of the $\overline{\mathbf{GRAD}}$ operator in matrix form:

$$\overline{\mathbf{GRAD}}=\begin{bmatrix}-\mathbf{D}_1^* & -\mathbf{D}_3^*\\ -\mathbf{D}_2^* & -\mathbf{D}_4^*\end{bmatrix}.\qquad(10)$$

If $\mathbf{H}$ denotes the vector of potential values in cell centers, $\mathbf{H}_B$ denotes the vector of potential values at boundary faces, and $\overline{\mathbf{H}}$ denotes the concatenation of these two vectors, then

$$\mathbf{GRAD}\,\overline{\mathbf{H}} = -\mathbf{D}_1^*\mathbf{H} - \mathbf{D}_3^*\mathbf{H}_B. \tag{11}$$

Equation (11) defines an affine operator acting on $\mathbf{H}$ that will be denoted by $\mathbf{Grad}$:

$$\mathbf{Grad}\,\mathbf{H} \equiv -\mathbf{D}_1^*\mathbf{H} - \mathbf{D}_3^*\mathbf{H}_B. \tag{12}$$

Even though $\mathbf{H}_B$ also contains the unknown potentials at Γ_2, their corresponding entries in $\mathbf{D}_3^*$ are zero, so $\mathbf{D}_3^*\mathbf{H}_B$ is known.

Operators $-\mathbf{D}_2^*$ and $-\mathbf{D}_4^2$ compute the potential differences along Γ_2. Since the flux was prescribed here, these potential differences are not needed.

Operator $\mathbf{Grad}$ computes the potential differences between cell centers, while $\mathbf{Div}$ takes the fluxes as its argument. It is the task of the constitutive relation, Darcy's law, to map one onto another.

The simplest discretization of the constitutive relation was used. It was assumed that the conductivity is isotropic, in which case $\mathbf{K}$ reduces to a scalar. Operator $\mathbf{Darcy}$ mapping $\mathbf{Grad}\,\mathbf{H}$ to $\mathbf{U}$ is represented by a diagonal matrix with diagonal entries

$$d_{ii} = -\frac{A_i K_i}{l_i}, \tag{13}$$

where A_i is the area of face i, K_i is the conductivity over face i, and l_i is the distance between the centers of cells sharing face i, or between the cell center and face i center for boundary faces. This mapping assumes that the line connecting the neighboring cell centers is normal to the face in between. Therefore it is not very accurate on distorted meshes. However, it leads to a symmetric discrete Laplacian operator.

A discrete version of (3) is

$$(\mathbf{Div} * \mathbf{Darcy} * \mathbf{Grad})(\mathbf{H}) = 0, \tag{14}$$

where $*$ denotes the operator multiplication defined as $(\mathbf{A} * \mathbf{B})(\mathbf{x}) = \mathbf{A}(\mathbf{B}(\mathbf{x}))$. This relation includes the boundary conditions.

IV. Implementation aspects

We have used MTL4 library [2] for sparse matrix and vector operations because most other sparse matrix libraries do not implement matrix-matrix multiplication. Distributed and Unified Numerics Environment (DUNE) [1] was used as a grid interface.

To compute the gradient operator using formula (6), the adjoint function must know $\mathbf{D}_3^*\mathbf{H}_B$, and this information is not contained in $\mathbf{Div}$ operator. Therefore, the implementation of the divergence operator must contain matrix $\mathbf{D}_1$ as well as vectors $\mathbf{D}_2\mathbf{U}_N$ and $\mathbf{D}_3^* H_B$. Then the adjoint function computes the adjoint of $\mathbf{D}_1$ and swaps the vectors.

The following listing shows how the C++ class representing multiplicable operators:

```
 1  template<class matrixtype,
 2                     class vectortype>
 3  class mulop // Class of multiplicable
 4              // operators
 5  {
 6  public:
 7
 8  // Constructors
 9
10     mulop() {}
11     mulop(const matrixtype& A): _A_(A) {}
12     mulop(const matrixtype& A,
13        const vectortype& cons): _A_(A),
14        _cons(cons) {}
15     mulop(const matrixtype& A,
16        const vectortype& cons,
17        const vectortype& cons1) :_A_(A),
18        _cons(cons), _cons1(cons1) {}
19
20  // Other member functions
21
22     matrixtype& matrix()
23     {
24        return _A_;
25     }
26     vectortype& vector()
27     {
28        return _cons;
29     }
30     vectortype& vector1()
31     {
32        return _cons1;
33     }
34     const mulop<
35     mtl::matrix::scaled_view<int,
36        matrixtype>,
37        mtl::vector::scaled_view<int,
38        vectortype>> operator-() const
39     {
40        mulop<mtl::matrix::scaled_view
41                 <int,matrixtype>,
42                 mtl::vector::scaled_view
43                 <int,vectortype>>
44           n(-1*_A_, -1*_cons, -1*_cons1);
45        return n;
46     }
47  private:
48     matrixtype _A_;
49     vectortype _cons, _cons1;
50  };
```

To provide maximum flexibility without compromising performance, the class was implemented as a template. Lines 10–18 show various constructors. The private section (lines 48–49) contains the matrix and the two vectors necessary to represent an affine operator and its adjoint. The access to these private members is provided by functions shown in lines 22–33. Lines 34–46 show the unary minus operator. Instead of multiplying each member of a matrix or a vector with -1, the MTL4 library provides the `scaled_view` templates which represent the result of such multiplication, and vector elements are multiplied with the factor when accessing an element of the view.

The following listing shows the adjoint function:

```
template<class matrixtype,
         class vectortype>
mulop<mtl::matrix::transposed_view
      <matrixtype>, vectortype>
   adj(mulop<matrixtype,vectortype>& m)
{
   mtl::matrix::transposed_view
         <matrixtype> tr(m.matrix());
   mulop<mtl::matrix::transposed_view
         <matrixtype>, vectortype>
      n(tr,m.vector1());
      // To conserve the memory, do
      // not copy Neumann condition
   return n;
}
```

Multiplication of two operators is provided below:

```
template<class matrixtype,
   class matrixtype1, class vectortype,
   class vectortype1>
mulop<matrixtype, vectortype>
   operator*(const mulop<matrixtype,
   vectortype>& m1, const mulop
   <matrixtype1, vectortype1>& m2)
{
   mulop<matrixtype, vectortype> m;
   m.matrix() = m1.matrix()*m2.matrix();
   if (m2.vector().size() != 0)
   {
      m.vector() =
            m1.matrix()*m2.vector();
   }
   if (m1.vector().size() != 0)
   {
      m.vector() += m1.vector();
   }
   return m;
}
```

Multiplication of an operator and a vector is provided below:

```
template<class matrixtype,
   class vectortype, class vectortype1>
vectortype1 operator*
   (const mulop<matrixtype,
   vectortype>& m, const vectortype1& v)
{
   vectortype1 w;
   w = m.matrix()*v;
   if (m.vector().size()) != 0)
   {
      w += m.vector();
   }
   return w;
}
```

Linear system (14) is assembled in C++ using the following notation:

```
linsys = div*darcy*(-adj(div));
```

Variables `div`, `darcy`, and `linsys` are instances of the same `mulop` class. The code looping trough mesh elements in order to initialize operators `div` and `darcy` was not shown.

To reduce memory requirements, it is possible to implement an affine operator that does not store a matrix. Instead, it computes the non-zero matrix entries on the fly. In this case a specialized `operator*` that multiplies such affine operator with other affine operators must be provided.

A method to optimize the performance of a C++ finite volume code was presented in [18].

V. Example

For the computational domain we take the unit cube (Fig. 1). Outflow rate $u_n \equiv 1$ was prescribed on the right hand boundary face, while the potential $h = 1 - x$ was prescribed on all other boundaries. We take $K \equiv 1$. The resulting linear system was solved using BiCGStab method with ILU preconditioner, provided by MTL4 library. Fig. 1 shows the distribution of the calculated potential over a symmetry intersection plane.

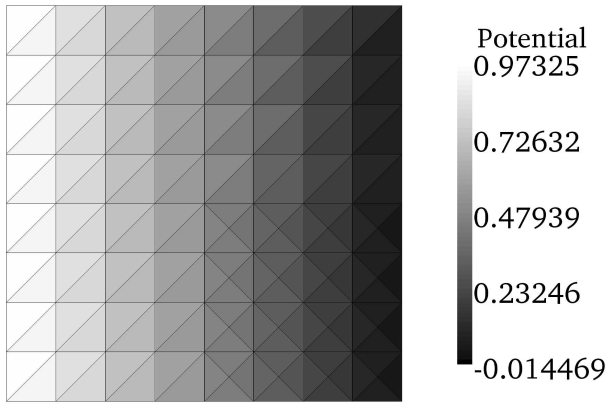

Fig. 1. The potential.

VI. Velocity reconstruction

For the sake of visualization, it is necessary to compute the velocity vectors in mesh nodes. Once the potential is computed, the fluxes trough mesh faces can be obtained as (see (14))

$$\mathbf{F} = (\mathbf{Darcy} * \mathbf{Grad})(\mathbf{H}). \tag{15}$$

This part requires no additional programming because all operators are already available.

Obtained flux can be seen as a staggered velocity field. Reconstruction of staggered vector fields has been studied in [11], [14], and [15]. For visualization purposes, a simple first order accurate reconstruction method suffices. The velocity vector $\mathbf{u}_j$ is reconstructed in each vertex j of each volume i by solving the linear system

$$\mathbf{u}_j \cdot \mathbf{n}_k = F_k/A_k, \qquad k \in \{1, 2, 3\}, \tag{16}$$

where n_k is the normal to face k of volume i sharing node j pointing in the direction of the cell with the higher cell number, and F_k and A_k are the corresponding flux and the face area, respectively.

On faces where Neumann boundary condition was prescribed, equation (16) is replaced by

$$\mathbf{u}_j \cdot \mathbf{n}_k = u_k, \tag{17}$$

where u_k is the prescribed normal velocity in face k.

The velocity vectors obtained in this way for each cell meeting in a node are averaged.

VII. Accuracy of the scheme

While the potential field shown in 1 looks reasonable, the velocity field obtained from this potential by the reconstruction procedure described in Section VI is way off, as shown in Fig. 2. The reason is the inaccurate discretization of the constitutive relation. The exact potential in the model problem takes values between zero and one, so this inaccuracy can also be seen in the legend of Fig. 1, which shows values below zero. This does not mean that the scheme is unbounded: due to the presence of the Neumann boundary condition, the maximum principle cannot be applied to this case.

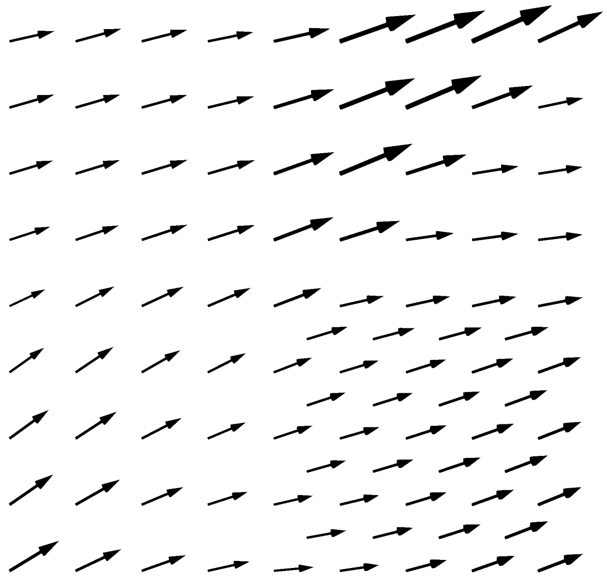
Fig. 2. Reconstructed velocity field.

In practical applications, the flux obtained from (3) is often needed to compute some sort of transport. Fluxes leading to the velocity field shown in Fig. 2 are clearly not accurate enough for such applications.

A method to improve the accuracy has been presented in [8]. Vector normal to a face is represented as a linear combination of vector connecting the centers of the two cells sharing that face, and a vector parallel to that face. Accordingly, the flux is split into two components. The parallel component is computed by interpolating cell-centered gradients to the face in between, while the other component is computed as in (13). However, the formula used to compute the cell-centered gradients does not comply with the orthogonality relation (2) and may cause the unboundness reported in [8].

VIII. Piecewise constant reconstruction

To obtain a more accurate discretization of the constitutive relation, we deploy a reconstruction procedure similar to the ones presented in [11], [14], [15], and in Section VI: we are looking for vector $\mathbf{u}$ such that

$$\mathbf{u} \cdot \mathbf{l} = -K_l \Delta h, \tag{18}$$

for some set of faces, where $\mathbf{l}$ is the vector connecting the cell centers on the two sides of such a face, directed towards the cell with the higher cell number, K_l is the conductivity in the direction of $\mathbf{l}$, and Δh is the difference in the potential between these two cells. Equation (18) is the Darcy law applied between the two cell centers.

In boundary faces where Dirichlet condition was prescribed, for $\mathbf{l}$ we take vectors connecting the boundary cell center with the boundary face center pointing outside, and Δh is the difference between the potential prescribed in the boundary face and the potential in the boundary cell center.

If Neumann boundary condition was prescribed in some face, it is imposed instead of (18), like in (17).

Methods presented in [11], [14], [15], and in Section VI reconstruct the velocity field from its face normal components. The fact that in (18) we want to reconstruct the velocity vector from components which are not in the direction of face normals has an important consequence: it is not guaranteed that a 3x3 system (17,18) made of equations corresponding to three faces meeting in one node is not singular. For example, Fig. 3 shows three neighboring tetrahedral cells with a common horizontal base plane and a common top. We assume that the front face of the middle cell (face ABD) is a boundary face in which Neumann boundary condition was prescribed. In order to reconstruct the velocity vector in vertex D of cell ABCD, we form a linear system containing two equations of type (18) corresponding to cell pairs ABCD-ACDE and ABCD-BCDF, and one equation of type (17) corresponding to face ABD. However, lines connecting the centers of the presented cells are parallel to the base plane, which is also true for the normal vector in face ABD if this face is normal to the base plane, and therefore our reconstruction system is singular.

To avoid this situation, the linear system is made up of equations (17,18) corresponding to all faces of a cell. This system is overdetermined because it has at least four equations (for tetrahedral cells, otherwise more), and we use the least squares method to solve it. This gives a velocity vector u_c for each cell c.

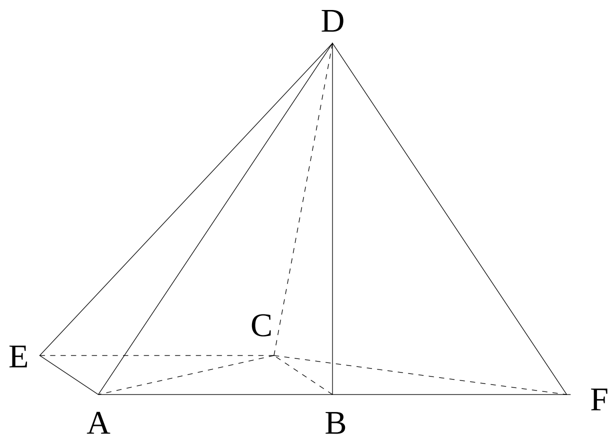

Fig. 3. An example reconstruction stencil giving a singular reconstruction system.

This reconstruction technique is exact for any linear potential field leading to a constant velocity field, such as the one presented in Section V. It leads to the same stencil as the unbounded technique presented in [8], but, since it does not violate (2), it does not cause unboundness. The obtained potential field shown in Fig. 4 and the corresponding velocity field shown in Fig. 5 are exact.

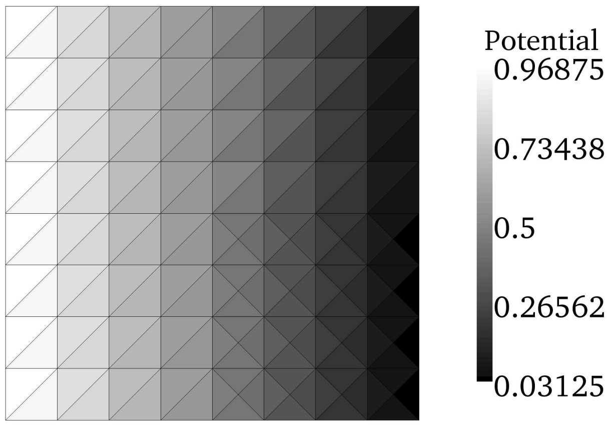

Fig. 4. Potential field - improved accuracy.

IX. Piecewise linear reconstruction

To obtain a reconstruction technique exact for any linear velocity field following from a quadratic potential, the velocity vector $\mathbf{u}$ is represented by a linear function

$$\mathbf{u}(x) = \mathbf{A}\mathbf{x} + \mathbf{b}, \tag{19}$$

where $\mathbf{A}$ is a 3×3 matrix, $\mathbf{x}$ is a position vector, and $\mathbf{b}$ is a constant vector. To determine $\mathbf{A}$ and $\mathbf{b}$, it is required that for some set of faces f

$$\int_l \mathbf{u}(\mathbf{x}) \cdot d\mathbf{l} = \int_l -\mathbf{K}\nabla h \cdot d\mathbf{l}, \tag{20}$$

where the integrals are taken over a *dual edge* of face f, i.e. over the line connecting the centers of cells sharing

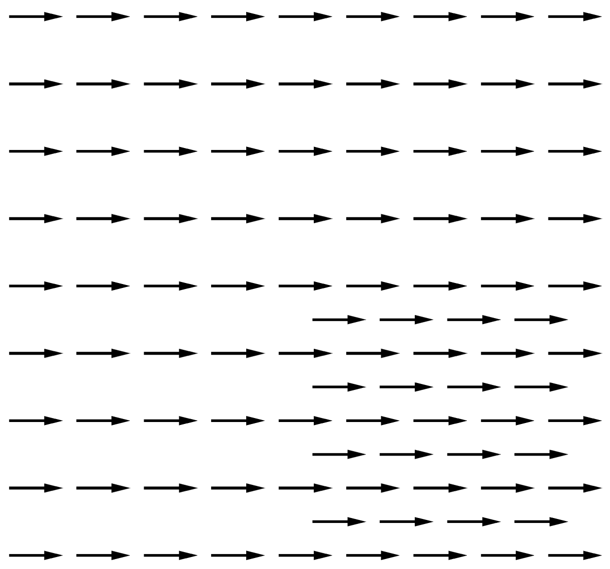

Fig. 5. Reconstructed velocity field - improved accuracy.

face f, in the direction of the cell with the higher cell number. Since $\mathbf{u}$ is linear, assuming that $\mathbf{K}$ is constant along l reduces (20) to

$$\mathbf{u}(\mathbf{x}_f) = -K_l(\Delta h)_f, \tag{21}$$

where $\mathbf{x}_f$ is the coordinate vector of face f center, and $(\Delta h)_f$ is the change in the potential along the dual edge.

In analogy with (17), equation

$$\mathbf{u}(\mathbf{x}_f) \cdot \mathbf{n}_f = u_f \tag{22}$$

is used instead of (21) for faces where Neumann boundary condition was prescribed.

Expression (19) has 12 degrees of freedom. Therefore, linear system (21), (22) needs to include at least 12 equations in order to determine a linear velocity field. It has been shown in [15] that, in the case when dual edges are normal to the faces, due to the divergence over-determination the rank of this system is at most $n_f - n_c + 1$, where n_f and n_c are respectively the number of faces and the number of cells entering the reconstruction stencil. It is said that a cell enters the reconstruction stencil if all its faces enter the reconstruction stencil. In addition, it has been shown in Section VIII that the rank can be lost due to the linear dependence of dual edge vectors. It is therefore difficult to determine the sufficient reconstruction stencils in advance. To reconstruct the velocity in a face center, one can start with a stencil containing the faces of two neighboring cells and of their neighboring cells, and extend it dynamically if the obtained linear system becomes singular or ill-conditioned.

X. Solving the reconstruction system

In the similar manner piecewise quadratic or piecewise cubic reconstruction methods can be constructed, requiring larger stencils. As the reconstruction stencil grows, in order to preserve the diagonal dominance of the resulting computational matrix it is necessary to give more importance to the faces near the center of the reconstruction stencil than to the faces at the edge of it. The classical least squares with constraints [9] cannot be used because the equations corresponding to the central faces may be linearly dependent and contradictory.

One way to solve this problem is to introduce weights. The equations that should be most closely matched are multiplied with weights which may be 100 or 1000 times larger than the weights used to multiply other equations.

However, as the reconstruction stencil grows this technique becomes inadequate. An alternative is to use the *hierarchical least squares* method.

Linear system $Ay = b$ is decomposed into a set of linear subsystems:

$$A_i y = b_i, \qquad i = 1, \ldots, s, \tag{23}$$

where $y \in \mathbf{R}^n$, $b_i \in \mathbf{R}^{m_i}$ and $\mathrm{rank}(A) \geq n$. We want to find a family of sets $\mathbf{Y}_1 \supseteq \mathbf{Y}_2 \supseteq \cdots \supseteq \mathbf{Y}_s$ such that $\mathbf{Y}_i = \{y \in \mathbf{Y}_{i-1} | \|A_i y - b_i\| \text{ is minimal } \}$ and $\mathbf{Y}_0 = \mathbf{R}^n$.

We calculate the SVD of matrix A_1 and get $A_1 = U_1 S_1 V_1^T$, where U_1 and V_1 are unitary matrices and S_1 is a diagonal matrix containing singular values of matrix A_1. We replace elements of S_1 by their inverses if they are larger than some tolerance, set others to zero, and denote the obtained matrix by S_1^{-1}.

The singular values below the tolerance are indicators that some of the linear equations are close to being linearly dependent. By setting these singular values to zero we treat these equations as if they were linearly dependent, excluding their weak independence which may lead to computational instabilities.

Solution $y^1_{\mathrm{pr}} \in \mathbf{Y}_1$ such that $\|y^1_{\mathrm{pr}}\|$ is minimal is given by

$$y^1_{\mathrm{pr}} = V_1 S_1^{-1} U_1^T b_1. \tag{24}$$

Strictly speaking, if some of the non-zero singular values were set to zero, then y^1_{pr} does not really belong to $\mathbf{Y}_1$ because it does not minimize $\|A_i y - b_i\|$. It is possible to make this norm even smaller by taking into account these small singular values. This would, however, lead to instabilities. Therefore we should understand the minimum in the definition of $\mathbf{Y}_i$ as the minimum over all solutions that do not lead to trouble.

Columns of V_1 related to the zero (or below the tolerance) singular values form the basis of the null space $null(A_1) = \{y | A_1 y = 0\}$. We shall denote the matrix consisting only of these columns by V_1^{null}. Now

$$\mathbf{Y}^1 = \{y^1_{\mathrm{pr}} + V_1^{\mathrm{null}} y^1 | y^1 \in \mathbf{R}^{n_1}\}, \tag{25}$$

where $n_1 = n - \mathrm{rank}(A_1)$. This is because $\{V_1^{\mathrm{null}} y^1 | y^1 \in \mathbf{R}^{n_1}\} = null(A_1)$ and so $\{y^1_{\mathrm{pr}} + V_1^{\mathrm{null}} y^1 | y^1 \in \mathbf{R}^{n_1}\} = \{y \in \mathbf{R}^n | A_1 y = A_1 y^1_{\mathrm{pr}}\}$.

We proceed by induction. Suppose that we found some $y_0^i \in \mathbf{Y}^i$

$$y_0^i = E_i \begin{bmatrix} b_1 \\ \vdots \\ b_i \end{bmatrix}, \tag{26}$$

and that $\mathbf{Y}^i$ is represented by

$$\mathbf{Y}^i = \{y_0^i + F_i y^i | y^i \in \mathbf{R}^{n_i}\}, \tag{27}$$

where $n_i > 0$. By substituting $y = y_0^i + F_i y^i$ into equation $A_{i+1} y = b_{i+1}$ we obtain

$$A_{i+1} F_i y^i = b_{i+1} - A_{i+1} y_0^i. \tag{28}$$

We calculate the SVD of $A_{i+1} F_i = U_{i+1} S_{i+1} V_{i+1}^T$ and find

$$y^{i+1}_{\mathrm{pr}} = V_{i+1} S_{i+1}^{-1} U_{i+1}^T \left(b_{i+1} - A_{i+1} y_0^i\right) \tag{29}$$

If $A_{i+1} F_i$ has a nontrivial null space then columns of V_{i+1} related to the zero singular values form the basis of this null space. We shall denote the matrix consisting only of these columns by V_{i+1}^{null}.

If $y^i = y^{i+1}_{\mathrm{pr}} + V_{i+1}^{\mathrm{null}} y^{i+1}$ where $y^{i+1} \in \mathbf{R}^{n_{i+1}}$, then $A_{i+1} F_i y^i = A_{i+1} F_i y^{i+1}_{\mathrm{pr}}$ and so

$$\mathbf{Y}^{i+1} = \{y_0^i + F_i(y^{i+1}_{\mathrm{pr}} + V_{i+1}^{\mathrm{null}} y^{i+1}) | y^{i+1} \in \mathbf{R}^{n_{i+1}}\} \tag{30}$$

or

$$\mathbf{Y}^{i+1} = \{y_0^{i+1} + F_{i+1} y^{i+1} | y^{i+1} \in \mathbf{R}^{n_{i+1}}\}, \tag{31}$$

where

$$y_0^{i+1} = E_{i+1} \begin{bmatrix} b_1 \\ \vdots \\ b_{i+1} \end{bmatrix}, \qquad F_{i+1} = F_i V_{i+1}^{\mathrm{null}},$$

$$E_{i+1} = \tag{32}$$

$$\begin{bmatrix} E_i - F_i V_{i+1} S_{i+1}^{-1} U_{i+1}^t A_{i+1} E_i & F_i V_{i+1} S_{i+1}^{-1} U_{i+1}^T \end{bmatrix}.$$

If $A_{i+1} F_i$ does not have a nontrivial null space, then $\mathbf{Y}^{i+1}$ contains only y_0^{i+1}, which is the final solution to the set of systems (23). In this case it should be $i + 1 = n$, otherwise the subsystems $i + 2, \ldots, n$ do not influence the solution. If $i + 1 = n$ and $A_{i+1} F_i$ has a nontrivial null space, this means that $\mathrm{rank}(A) < n$. In this case we can add an additional subsystem and continue without restarting the algorithm. Therefore the reconstruction stencil does not have to be determined a priori.

XI. Conclusion

A technique for implicit PDE solver implementation using multiplicable operators was presented. Due to boundary conditions which are incorporated in the discretization, a discrete version of a linear operator is not always linear in the unknown part of its operand, but it may be affine. Such affine operators are combined by matrix-matrix and matrix-vector multiplication.

This technique has been applied to several variants of the finite volume method. The first variant is accurate only if the lines connecting the cell centers are orthogonal or almost orthogonal to the corresponding face. Instead of applying the unbounded non-orthogonality correction found in the literature, we presented a piecewise constant mimetic discretization of the constitutive relation which does not suffer from unboundness and does not lead to a larger stencil, while being exact for piecewise constant velocity. We also presented a similar piecewise linear reconstruction of the velocity field, together with a robust method to solve the resulting reconstruction systems.

Multiplicable operators allow programmers to write solvers in an intuitive manner which resembles mathematical notation. As this technique is based on the discretization of individual operators, it facilitates the design of schemes with favorable properties, such as mimetic methods.

REFERENCES

[1] Distributed and unified numerics environment. http://www.dune-project.org/.

[2] The matrix template library 4. http://www.osl.iu.edu/research/mtl/mtl4/.

[3] N.V. Ardelyan. Method of investigating the convergence of nonlinear finite-difference schemes. *Diff. Eqns.*, 23(7):737–745, 1987.

[4] Aurel Chirita, Horia Ene, Bogdan Nicolescu, and Ion Carstea. Numerical simulation of flow in porous medium. In *Proceedings of the 1st WSEAS International Conference on Finite Differences - Finite Elements - Finite Volumes - Boundary Elements*, pages 34–38, Malta, 2008.

[5] James M. Hyman and Mikhail Shashkov. Adjoint operators for the natural discretizations of the divergence gradient and curl on logically rectangular grids. *Applied numerical mathematics*, 25(4):413–442, 1997.

[6] James M. Hyman and Mikhail Shashkov. Natural discretizations for the divergence, gradient, and curl on logically rectangular grids. *International journal of computers & mathematics with applications*, 33:81–104, 1997.

[7] James M. Hyman and Mikhail Shashkov. Approximation of boundary conditions for mimetic finite-difference methods. *International Journal of Computers & Mathematics with Applications*, 36:79–99, 1998.

[8] Hrvoje Jasak. *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. PhD thesis, Imperial College, UK, June 1996.

[9] T. Liszka, C. Duarte, and W. Tworzydlo. hp-Meshless cloud method. *Computer Methods in Applied Mechanics and Engineering*, 139:263–288, 1996.

[10] J.B. Perot. Conservation properties of unstructured staggered mesh schemes. *J. Comp. Phys.*, 159:58–89, 2000.

[11] J.B. Perot, D. Vidović, and P. Wesseling. Mimetic reconstruction of vectors. In *Compatible Spatial Discretizations*, New York, 2006. Springer.

[12] A.A. Samarskii, V.F. Tishkin, A.P. Favorskii, and M.Yu Shashkov. Operational finite-difference schemes. *Diff. Eqns.*, 17:854–862, 1981.

[13] A.A. Samarskii, V.F. Tishkin, A.P. Favorskii, and M.Yu Shashkov. Employment of the reference-operator method in the construction of finite difference analogs of tensor operations. *Diff. Eqns.*, 18:881–885, 1982.

[14] M. Shashkov, B. Swartz, and B. Wendroff. Local reconstruction of a vector field from its normal components on the faces of grid cells. *J. Comp. Phys.*, 139:406–409, 1998.

[15] D. Vidović, A. Segal, and P. Wesseling. A superlinearly convergent finite volume method for the incompressible Navier-Stokes equations on staggered unstructured grids. *Journal of Computational Physics*, 198:159–177, 2004.

[16] B. A. Youssef and E.H. Atta. Digital image inpainting using finite volume approach and the navier- stokes equations. In *Proc. of the 9th WSEAS Int. Conf. on Mathematical and Computational Methods in Science and Engineering*, pages 305–311, Trinidad and Tobago, 2007.

[17] X. Zhang, D. Schmidt, and J.B. Perot. Accuracy and conservation properties of a three-dimensional unstructured staggered mesh scheme for fluid dynamics. *Journal of Computational Physics*, 175:764–791, 2002.

[18] H. Zindlere and R. Leithner. Optimization of the finite volume method source code by using polymorphism. In *Proceedings of the 5th IASME/WSEAS Int. Conference on Heat Transfer, Thermal Engineering and Environment*, pages 243–248, Athens, 2007.