A toolkit for numerical simulation of PDE’s I: fundamentals of generic finite-volume simulation

Carl Ollivier-Gooch

Advanced Numerical Simulation Laboratory, Department of Mechanical Engineering, The University of British Columbia, 2324 Main Hall, Vancouver V6T 1Z4, Canada

Received 29 July 2002; received in revised form 19 October 2002

Abstract

To understand underlying physical phenomena or to design better devices or processes, experts in various application areas within science and engineering often need to solve partial differential equations numerically. These experts have tremendous knowledge about the physical phenomena they study, but often are significantly less knowledgeable about scientific computing. A scientific computing toolkit for generic solution of PDE’s would be a great benefit for such workers, requiring them only to specify the physics of their problem, with numerical issues handled by the toolkit.

This paper presents just such a toolkit, based on the finite-volume method, and argues that the finite-volume method is a more user-friendly choice for such a toolkit than the finite-element method. The user specifies problem physics by providing code snippets to compute interior and boundary fluxes, source terms, and initial conditions; and by specifying constraints on the solution at the boundaries. The toolkit addresses all of the strictly numerical issues: it recovers high-order accurate solution and gradient data from the control volume averaged solution; enforces boundary conditions; integrates the user-supplied fluxes and source terms; and performs time advance. Examples are given for advection–diffusion, solid mechanics, and compressible flow problem to demonstrate the flexibility of the advanced numerical simulation library framework.

1. Introduction

Scientists and engineers use numerical solutions of partial differential equations to solve problems of industrial and societal relevance in fields as diverse as aerodynamic analysis of aircraft [1,2], clinical treatment of cancer [3–5] and electron transport in semiconductors [6,7]. Experts in each of these areas know a tremendous amount about the important physical processes in the problems they study. Unfortunately, simulating these problems numerically requires scientific computing expertise that problem-do-
main experts must either acquire or hire. A far better alternative for such people would be to write a small amount of computer code to describe the physics of a problem—which they thoroughly understand—and use a scientific computing toolkit to automatically, correctly, and efficiently handle the numerical aspects of the simulation.

The finite element method has been the method of choice in previous work on generic solvers for PDE’s. One such generic solver, FEMLAB [8], provides an implementation of the finite-element method with Matlab providing the numerical linear algebra back-end. FEMLAB contains several standard simulations as well as a mechanism for discretizing general PDE’s for custom physics simulations. Unfortunately, the general PDE mechanism does not allow users flexibility in choosing element types, often a critical ingredient in correct numerical modeling.

Eyheramendy and Zimmermann describe a system for semiautomatic programming of finite-element simulations [9–11]. Their framework allows the user to guide derivation of new finite-element formulations and generates simulation code in Smalltalk. In subsequent work, the same authors have extended their system to allow derivation of finite-element formulations for non-linear problems [12]. This approach has great merit as a tool for experienced finite-element practitioners interested in exploring new discretization techniques or in attacking new physical problems, but requires more knowledge of the internals of the finite-element method than the typical novice in scientific computing is likely to have.

Perhaps the most comprehensive support framework for finite-element solution of PDE’s is DiffPack [13–15]. DiffPack is a set of tools designed to allow flexibility for end users in specifying physics, discretization methods, and numerical methods. In addition to allowing high-level control of problem physics, DiffPack also allows users to manipulate numerics by defining new finite elements (including integration points, mappings, and elemental matrices), for example, or by selecting different particulars for iterative solution of linear systems. As with the system described by Eyheramendy and Zimmermann, significant knowledge about the finite-element method is required to create a custom simulation in DiffPack. Specifically, users must derive and code the integrand for the weak form of the PDE, as well as providing code to implement boundary conditions.

A fundamental challenge for finite-element-based toolkits for PDE solution is that de-coupling physics and numerics is difficult with the finite-element method. This is because element matrices require integration of the product of test and basis functions, and these functions are often chosen specifically to capture particular physical features of a problem. This paper argues that the finite-volume method allows a cleaner de-coupling between physics and numerics, and introduces a scientific computing toolkit—called the advanced numerical simulation library (ANSLib)—that provides support for PDE simulation using the finite-volume method. 1 Because problem physics enters into the finite-volume method primarily through the calculation of fluxes—which themselves are free from numerical intricacy—detailed numerical knowledge is not required to produce custom simulations in ANSLib. Herein lies the key advantage to using the finite-volume method for a generic PDE solver: users with knowledge of the physics will be able to correctly identify and write code for fluxes and source terms, and to specify boundary conditions, and with a finite-volume approach, this is all that they need to able to do. In contrast, finite-element generic solvers can not provide flexibility in describing physics without embroiling users in numerical details.

ANSLib is designed to maximize flexibility in numerics with no dependence on physics. Our goal is to provide sufficiently robust and general numerical support that prototype numerical simulations will function reasonably well, regardless of the physical problem or the topology of the underlying mesh. Section 2 begins with a more detailed argument that finite-volume discretizations split cleanly between physics and numerics. The remainder of the section describes the current status of numerical infrastructure

---

1 A preliminary version of this work was presented at CFD2K, the Canadian Society for Computational Fluid Dynamics annual conference for 2000 [16].
in ANSLib. At this point, ANSLib supports both unstructured and structured meshes, with spatial discretization accuracy as high as fourth-order for unstructured meshes; currently, structured mesh accuracy is limited to second-order (see Sections 2.1 and 2.3). Time advance currently uses multistage explicit schemes (see Section 2.4). Also included in Section 2.5 are descriptions of several important data structures and low-level routines.

The physics of a problem is encoded for ANSLib by providing functions that describe the interior fluxes, boundary fluxes, solution constraints at the boundary, source terms, and initial conditions. The mechanics of this encoding is described in Section 3. Section 4 describes several examples of problems solved using ANSLib, drawn from both fluid and solid mechanics. In each case, the implementation of the problem physics within the ANSLib framework is described, in addition to showing results and demonstrating the asymptotic accuracy attainable with ANSLib.

1.1. The scope of ANSLib

ANSLib is not necessarily expected to produce simulations that are comparable in efficiency to carefully coded, special-purpose simulations for particular problems. Instead, the goal is to reduce to the extent possible the development time for new simulations, especially for users without scientific computing experience, while still producing simulations of acceptable accuracy and efficiency. For many users and applications, the dramatic reduction in development time will more than compensate for somewhat slower run-time performance.

We should also emphasize that ANSLib is not intended as a system for converting a PDE directly into a numerical simulation. For many application areas, specialized techniques for evaluating physical fluxes are required to accurately simulate problem physics, and producing software that can identify when to apply which particular techniques is beyond the current intent of ANSLib. By drawing a boundary between physics—fluxes, source terms, and boundary conditions—and numerics, we avoid the problems encountered by systems that begin with the PDE and proceed mechanistically, including issues with non-constant coefficients and non-linearity.

2. The anatomy of a generic finite-volume solver

Finite-volume algorithms divide the computational domain into a large number of small control volumes and seek to solve the integral form of a partial differential equation, obtained by integrating the PDE over each control volume and applying Gauss’s theorem. That is, the PDE

\[
\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = S(U,x,y,z)
\]

is transformed, for control volume CV\(_i\), into

\[
\int_{CV_i} \frac{\partial U}{\partial t} \, dv + \int_{\partial CV_i} (F_i + G_j + H_k) \cdot \hat{n} \, dA = \int_{CV_i} S(U,x,y,z) \, dv.
\]

Reversing the order of integration and differentiation for the first term; assuming fixed control volumes; and defining \(\langle \cdot \rangle \equiv \frac{1}{V} \int_{CV_i} (\cdot) \, dv\), we obtain for control volume \(i\)

\[
\frac{dU_i}{dt} + \frac{1}{V_i} \int_{\partial CV_i} (F_{ni} + G_{ni} + H_{ni}) \, dA = S_i.
\]

The key spatial discretization step in the finite-volume method is numerical flux integration. Regardless of the physical problem, any finite-volume scheme must calculate and integrate fluxes and source terms ac-
accurately. In turn, high-order accurate flux calculations require that the solution be approximated more accurately than by control volume averages; this is typically accomplished by reconstructing the solution in each control volume as a low-degree polynomial. Flux and source term quadrature can be accurately performed by using appropriate Gauss quadrature rules. In outline, a typical finite-volume solver can be formulated as:

1. Initialize control volume averaged solution.
2. Until end time or steady state is reached, do:
   (1) Reconstruct piecewise polynomial solution in each control volume from CV average data.
   (2) Enforce boundary conditions.
   (3) For each control volume interface in the mesh (both interior and boundary).
      (a) Compute the flux at Gauss integration points, and
      (b) Add contribution to the flux integrals of the control volumes incident on the interface.
   (4) For each control volume.
      (a) Compute the source term at Gauss integration points, and
      (b) Integrate over the control volume.
   (5) Advance the solution in time.

In this outline, initialization and steps 2, 3(a), and 4(a) require knowledge of the physics, while steps 1, 3(b), 4(b), and 5 are purely numerical. Also, the numerical parts of this process typically require more code and are more complicated than the physics. These observations are the motivation for ANSLib. The remainder of this section describes, following the outline given above, the numerical infrastructure of ANSLib.

2.1. Solution reconstruction

Finite-volume schemes compute the average value of the solution over each control volume in the mesh. The difference between the average solution in adjacent control volumes is first-order in the mesh spacing. As a result, when the fluxes at control volume boundaries are calculated using control volume averaged data, the overall solution is only first-order accurate. To achieve higher accuracy, the control volume averaged solution must be replaced in each control volume by a more accurate approximation, typically a low-degree polynomial. For efficiency reasons, ANSLib uses different approaches for solution reconstruction for unstructured and structured meshes. This is invisible to the user, because the reconstruction is mesh-type sensitive, and so ANSLib automatically selects the correct reconstruction for the mesh supplied at run time. Note also that the reconstruction is independent of problem physics; the number of unknowns is provided to the reconstruction as a parameter, and no information about the problem itself is required.

2.1.1. Least-squares reconstruction on unstructured meshes

For unstructured meshes, ANSLib uses $k$-exact least-squares polynomial reconstruction, a process that computes for each control volume the coefficients for a polynomial so that:

1. The mean is conserved: The computed control volume average should match the average of the polynomial produced by reconstruction. Conservation of the mean acts as a constraint on the coefficients.
2. The reconstruction is accurate for smooth functions: If some smooth function $f(x)$ is averaged over the control volumes in a mesh, $k$-exact reconstruction aims to reconstruct the function from its control volume averages to within $O(\Delta x^{k+1})$. Accuracy is achieved by requiring that the polynomial in each control volume minimize in a least-squares sense the error in predicting the average function value in nearby control volumes.
Numerous researchers have described $k$-exact reconstruction in detail (for example, see [17–20]); we will not repeat the details here. Given the polynomial reconstruction, solution and gradient data can be provided as needed for all points in the control volume. Note that two values exist at points on the control volume boundaries: one from each neighboring control volume. In general, the difference between these values is on the order of truncation error.

Each unstructured mesh representation in ANSLib (cell-centered and vertex-centered control volumes in two dimensions; cell-centered CV’s in three dimensions)² must provide the following information for each control volume in support of reconstruction:

1. **Moments of the control volume:** For reconstruction to degree $k$, moments for all monomials of degree $\leq k$ are required. These moments, which are computed and stored as a pre-processing step, are given by

$$
\frac{1}{V} \int_{CV} (x-x_i)^l (y-y_i)^m (z-z_i)^n \, dV,
$$

where $\bar{x_i}$ is the reference location for control volume $i$. For vertex-centered control volumes, this reference point is taken to be the vertex itself, while for cell-centered control volumes, the cell centroid is used.

In practice, the divergence theorem is used to convert the integral in Eq. (3) to an integral around the boundary of the control volume:

$$
\frac{1}{V} \int_{\partial CV} \frac{(x-x_i)^{l+1}}{l+1} (y-y_i)^m (z-z_i)^n \hat{n} \, dA,
$$

where $\hat{n}$ is the outward unit normal for the surface. This transformation especially simplifies the calculation of moments for control volumes along curved domain boundaries.

2. **Stencil information:** Least-squares reconstruction sets polynomial coefficients to match data in nearby control volumes; a list of nearby control volumes is therefore required. To allow some slack for the least-squares process to filter out noisy data, ANSLib uses about 50% more control volumes than the number of polynomial coefficients to be determined. Specifically, ANSLib uses a minimum of 3 (4) control volumes for linear reconstruction, 8 (15) for quadratic reconstruction, and 14 (30) for cubic reconstruction in 2 (3) dimensions. Control volumes are added to the reconstruction stencil based on their topological proximity to the reconstruction control volume. All neighbors at a given level are added at once. Fig. 1 gives examples of stencils for both vertex- and cell-centered control volumes in the interior of a two-dimensional mesh. Each figure shows the stencil for reconstruction in the control volume labeled R; the numeric labels indicate the order of accuracy at which a given control volume is added to the stencil. Note that the vertex-centered example does not require any additional control volumes in the stencil for fourth-order (cubic) reconstruction that are not already present for third-order (quadratic) reconstruction. At boundaries, stencils are constructed using the same principles, although more layers of neighbors are typically required to get large enough stencils.

2.1.2. Data extrapolation on structured meshes

For structured meshes, least-squares reconstruction would again be accurate, but is much less efficient than the standard approach of direction-by-direction interpolation. At present, ANSLib uses linear extrapolation to estimate solution data on both sides of each control volume boundary. That is, when estimating the solution on the left side of the control volume interface at $i + (1/2), j$, ANSLib computes

$$(\cdot)_{i+(1/2), j} = (3/2)(\cdot)_{i,j} - (1/2)(\cdot)_{i-1,j};$$

an analogous formula is used to estimate the value on the right side. Near boundaries, the stencils are adjusted as required. Gradients are computed in the usual way: by first

² Currently under development.
computing derivatives in a computational space and then converting these to derivatives in physical space by using the mesh metrics (see, for instance, Anderson et al. [21] for a full description). All derivatives used in computing gradients are centered, with the result that the gradients computed for a control volume boundary are the same in both control volumes.

For both structured and unstructured meshes, the only requirement to make solution reconstruction independent of problem physics is that the number of unknowns must be a parameter of the reconstruction process.

2.2. Constraining the solution at the boundaries

The specification of boundary conditions is clearly a part of the physics of a particular problem. Nevertheless, the enforcement of boundary conditions is a numerical issue, and can be approached generically.

For example, Dirichlet boundary conditions can all be enforced by matching the solution at the boundary to some given value, whether the physical boundary condition is fixed temperature in a heat conduction problem, zero turbulent kinetic energy for a turbulent flow, or an imposed displacement in a solid mechanics problem. Likewise, Neumann boundary conditions can be expressed either as a constraint on the solution gradient at the boundary or as a special boundary flux, whether the physical boundary condition is fixed heat flux in a heat conduction problem or an imposed stress in a solid mechanics problem.

As with reconstruction, the approach used within ANSlib to enforce boundary conditions is significantly different for unstructured versus structured meshes. Nevertheless, specification of boundary conditions—the user’s only concern—is essentially the same regardless of mesh topology.

2.2.1. Unstructured mesh boundary constraints

One bonus of using least-squares reconstruction for unstructured meshes is that we can enforce Dirichlet, Neumann, and mixed Dirichlet–Neumann boundary conditions by constraining the least-squares reconstruction in control volumes adjacent to the boundary. This boundary condition enforcement scheme is parsimonious, in the sense that boundary conditions are strictly enforced only at points where boundary data is actually used—the Gauss integration points along the boundary. Elsewhere, the boundary conditions are satisfied to within truncation error. This boundary constraint mechanism is quite flexible, in principle allowing any linear boundary condition to be enforced directly as a solution constraint (Fig. 2).
More complex boundary conditions (e.g., characteristic boundary conditions for compressible flow) must
be handled, at least in part, by careful definition of boundary fluxes. Also, in certain cases (e.g., imposed
stress in solid mechanics), the solution constraints are complex enough that it is easier to enforce the
boundary conditions weakly, using a special boundary flux, instead of strongly, using a constraint.

Suppose that along part of the boundary the solution must satisfy a Dirichlet boundary condition
\( \phi(\bar{x}) = f_1(\bar{x}) \). We enforce this by requiring that the polynomial reconstruction of the solution in the control
volume must match the boundary condition at each Gauss integration point \( \bar{x}_i \):

\[
f_1(\bar{x}_i) = \phi_i^R(\bar{x}_i) = \phi_{i} \bigg|_{x} (x_{x} - x_{i}) + \frac{\partial \phi}{\partial y} \bigg|_{y} (y_{y} - y_{i})
+ \frac{\partial^2 \phi}{\partial x^2} \bigg|_{x} \frac{(x_{x} - x_{i})^2}{2} + \frac{\partial^2 \phi}{\partial x \partial y} \bigg|_{x} (x_{x} - x_{i})(y_{y} - y_{i}) + \frac{\partial^2 \phi}{\partial y^2} \bigg|_{y} \frac{(y_{y} - y_{i})^2}{2} + \cdots,
\]

where \( \phi_{i} \) is the value of the reconstructed solution at the reference point for control volume \( i \), and so on.
Neumann and mixed boundary conditions can be written in a similar mathematical form. For each
boundary Gauss point in a boundary control volume, therefore, we add an equality constraint to the least-
squares reconstruction problem. This constrained least-squares problem is solved by applying Gauss
elimination with pivoting to the constraints, then solving the remainder of the system in a least-squares
sense by using Householder transforms; for a full discussion, see [22].

The approach just described is in practice an excellent way to enforce many simple boundary conditions.
To shield the creator of new simulations from the challenging task of setting up the correct Taylor ex-
pansions for each boundary condition, ANSLib uses a boundary constraint compiler and C++ class
(Taylor) that manipulates Taylor series representations. The constraint compiler produces a C++ source
file that contains instructions for Taylor objects to construct the proper solution constraint. The Taylor
class encapsulates the code that stores Taylor expansions, operates on their coefficients, and eventually
communicates a low-level constraint on the Taylor series coefficients to the reconstruction routine.

2.2.1.1. Boundary constraint descriptions. The boundary constraint description can be described using a
context-free LALR(1) grammar, which is compatible with standard parser generators; a summary of the
grammar (without rule actions) is given in Appendix A.1. This subsection gives a verbal description of the
file structure, using the sample boundary constraint description file shown in Fig. 3 as an example. This
particular example is drawn from the artificial compressibility formulation for incompressible fluid flow.
The first section of the boundary constraint description file, set apart between \%\% and \%\% in conscious imitation of yacc grammar specification files, is copied directly into the output C++ file. This section is used for declaring variables and functions needed for boundary condition enforcement; in this case, the Reynolds number is declared as a global variable and a function is defined that returns a fully developed velocity profile for a channel of width one and $u_{\text{max}} = 1$.

Next, the unknowns of the problem are identified (line 6), in order, so that the Taylor series that will eventually be produced will constrain the correct solution variables in the reconstruction. For problems with an unknown vector quantity—such as displacement, velocity, or momentum—the components of that vector can be defined (line 7); this allows constraints to be placed on, for example, the velocity component normal to a boundary.

Fig. 3. Sample boundary constraint description file (line numbers added for exposition).
The remainder of the file is a series of boundary condition declarations. Each of these begins with a label (e.g., Inflow in line 8) and an integer tag (e.g., 1 in line 8). This tag must coincide with the tag for the flux for that same boundary condition (see Section 3.2 for information about boundary fluxes). This matching is dramatically simplified because the boundary constraint compiler adds a preprocessor definition for the label to its output C++ file. In this particular case, that definition would be

```
#define User_BC_Inflow 1
```

Now the boundary flux code case uses `User_BC_Inflow` in a switch to select the appropriate boundary flux to match a given constraint.

After the label and tag, a series of constraints on the solution are given (e.g., lines 9–11). Each constraint must have on its left-hand side a linear combination of solution variables or their derivatives; derivatives are written as $Dw$, where $w$ may be $x$, $y$, or $z$ for derivatives in the Cartesian directions or $n$, $t$, or $c$ for derivatives in the normal, tangential, and cross directions relative to the boundary. For the inflow boundary condition, constraints are given for the velocity components and the normal derivative of pressure. Later examples include use of the special values $U_{\text{Norm}}$ and $U_{\text{Tang}}$, which are respectively the normal and tangential components of the vector quantity defined in line 7.

Any constants that appear on the left-hand side of a constraint must be surrounded by square brackets ([ ]) in the current implementation so that the boundary constraint compiler will pass them through to the output C++ code properly, as in line 16 of the example.

The right-hand side of a constraint must contain only (symbolic or literal) constants. The spatial coordinates $x$, $y$, and $z$ are also recognized as special cases, allowing spatially varying boundary conditions.

At compile time, ANSLib's boundary constraint compiler converts the boundary constraint description file into a C++ input file containing a single routine that sets up (one or more) boundary constraints analogous to Eq. (5) for each boundary condition. This routine is used during reconstruction in boundary control volumes to create constraints on the solution at each boundary Gauss point.

### 2.2.1.2. Taylor series manipulation

At the basic level, a Taylor series expansion for a solution variable, (e.g., $u$) can easily be created and stored, with a coefficient stored for each term into an array whose columns are derivatives in the Taylor series expansion and whose rows are monomials, as shown in Fig. 4. While this format is not compact in memory usage, it allows easy differentiation of Taylor series, even in arbitrary combination, because each row of coefficients in the expansion maps onto another row. For example, under differentiation by $y$, all coefficients in the $y^2$-row, regardless of the expansion terms to which they are attached, are doubled and placed in the $y$-row. This property is especially useful for more complicated expressions.

Many boundary constraints require combinations of more than one variable. For example, the normal component of velocity (written as $U_{\text{Norm}}$ in the input file) is shorthand for $um + vn$. To store such combinations of expansions, we use an extended storage scheme in which separate expansions for each variable are kept within a single object.

When an expansion is complete, each column can be summed at a given Gauss quadrature point by using the particular values of $x$ and $y$ at that point relative to the reference point for the control volume. The result can be written as a single constraint equation analogous to (though perhaps much more complicated than) the one in Eq. (5). Appendix B illustrates the step-by-step process of creating a more complex combined expansion that properly related the pressure and velocities at a wall. The unfortunate consequence of allowing such coupled boundary conditions is that the least-squares reconstruction in boundary control volumes must consider each term in the expansion of each variable separately, as opposed to reconstruction in interior control volumes, for which the reconstruction result is a purely geometric combination of control volume average values, with the same combination for each variable.
2.2.2. Structured mesh boundary constraints

The goal is the same here as for the unstructured mesh case, although the mechanism is adapted for structured meshes, with direction-by-direction data interpolation instead of reconstruction. Three quantities must be computed at the boundary for each unknown: its value, its normal derivative, and its tangential derivative. Just as in the unstructured case, boundary constraints require different values for these than the unconstrained solution would produce (Fig. 5).

Making matters more complicated, the derivatives that are available are not the normal and tangential derivatives but those in the computational space. For the general case of meshes not orthogonal to the boundary, both computational derivatives contribute to both the normal and tangential derivatives.

Consider the case of a cell along the western boundary of the domain (minimum $i$). The value of the solution $\phi$ at the boundary can be extrapolated, to second-order, as:

$$\phi_{i,j} = \frac{1}{2}\phi_{1,j} - \frac{1}{2}\phi_{2,j}.$$  

The derivatives in computational space, to the same accuracy, are given by:

$$\frac{\partial\phi}{\partial x} = \frac{1}{2}\phi_{x,1,j} + \frac{1}{2}\phi_{x,2,j}.$$  

Fig. 4. Storage for Taylor expansion of solution variable $u$.

---

3 The fluxes are likely to use $x$- and $y$-derivatives, but the mapping between these and the normal/tangential derivatives is trivial and the latter are more commonly used in boundary conditions.
To get the derivatives in the normal and tangential directions, we apply the chain rule for differentiation and replace metrics in physical space with metrics in computational space:

\[
\frac{\partial \phi}{\partial \tilde{z}} |_{w,j} = -6\phi_{w,j} + 7\tilde{\phi}_{1,j} - \tilde{\phi}_{2,j}
\]

\[
\frac{\partial \phi}{\partial \eta} |_{w,j} = \frac{3}{2} \frac{\tilde{\phi}_{1,j+1} - \tilde{\phi}_{1,j-1}}{2} - \frac{1}{2} \frac{\tilde{\phi}_{2,j+1} - \tilde{\phi}_{2,j-1}}{2}
\]

where the metrics and Jacobian in surface coordinates are defined in terms of Cartesian metrics as:

\[
n_{\xi} = \frac{\partial n}{\partial \xi} = x_n x_{\xi} + y_n y_{\xi},
\]

\[
t_{\xi} = \frac{\partial t}{\partial \xi} = x_t x_{\xi} + y_t y_{\xi},
\]

\[
n_{\eta} = \frac{\partial n}{\partial \eta} = x_n x_{\eta} + y_n y_{\eta},
\]

\[
t_{\eta} = \frac{\partial t}{\partial \eta} = x_t x_{\eta} + y_t y_{\eta},
\]

\[
J = n_{\xi} t_{\eta} - n_{\eta} t_{\xi} = x_t y_{\eta} - x_n y_{\xi}.
\]

If there is no constraint on the solution at the boundary, then, we can write a small system of equations relating the wall value of the solution and its derivative components:
\[
\begin{bmatrix}
1 & 0 & \frac{3n_i}{J} \\
0 & 1 & -\frac{3n_j}{J} \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\frac{\partial \phi}{\partial n_{w,j}} \\
\frac{\partial \phi}{\partial t_{w,j}} \\
\phi_{w,j}
\end{bmatrix} = \begin{bmatrix}
R_o \\
R_t \\
R
\end{bmatrix},
\]

where data on the right-hand side depends only on control volume average values of \( \phi \) in the interior of the mesh, and the order of the variables has been carefully chosen to give an upper-triangular system, which is equivalent to first calculating \( \phi_{w,j} \), then using that data to compute the normal and tangential derivatives.

Regardless of the topological boundary of the mesh at which the boundary condition is applied, the general form for the first two equations of this system, including the right-hand sides, is

\[
\frac{\partial \phi}{\partial n} + 3a\phi_w = a\left(\frac{7}{2}\phi_{I1} - \frac{1}{2}\phi_{I2}\right) + b \frac{\partial \phi}{\partial \chi},
\]

\[
\frac{\partial \phi}{\partial t} + 3c\phi_w = c\left(\frac{7}{2}\phi_{I1} - \frac{1}{2}\phi_{I2}\right) + d \frac{\partial \phi}{\partial \chi},
\]

where \( I1 \) represents the interior cell adjacent to the boundary face in question, and \( I2 \) the next interior cell away from the boundary. The derivative on the right is evaluated at the wall by extrapolation from the interior, and so can be computed without imposing boundary constraints. The values of \( a, b, c, d, \) and \( \chi \) differ depending on which boundary the constraint is imposed at. The correct values are shown in Table 1.

When a constraint is imposed—whether Dirichlet, Neumann, or mixed—the relationship between the solution and the normal and tangential derivatives at the wall (Eqs. (9) and (10)) are unchanged, and the value of the solution at the wall is still required to compute them. The constraint, therefore, replaces the extrapolated value of the solution at the boundary, which is the last equation in the system given in Eq. (11).

For a single unknown, then, the worst-case scenario is solution of a 3-by-3 system. In practice, many problems have unknowns that collectively form a physical vector quantity; examples include the momentum vector in fluid mechanics and the displacement vector in solid mechanics. In many cases, boundary conditions are specified in terms of these vector components. For example, a symmetry boundary condition in solid mechanics would require both zero normal displacement and zero normal derivative of the tangential displacement at the boundary. Such boundary conditions couple unknowns together. To make matters even more interesting, a boundary condition may not constrain all components of the vector quantity at the boundary.

To handle this situation in general, ANS Lib computes the boundary values of the solution and its derivatives by using a block system. Without boundary constraints, this system is given by

<table>
<thead>
<tr>
<th>Boundary</th>
<th>West (Min ( i ))</th>
<th>East (Max ( i ))</th>
<th>South (Min ( j ))</th>
<th>North (Max ( j ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( t_i/J )</td>
<td>( -t_i/J )</td>
<td>( -t_j/J )</td>
<td>( t_j/J )</td>
</tr>
<tr>
<td>( b )</td>
<td>( -t_i/J )</td>
<td>( -t_i/J )</td>
<td>( t_j/J )</td>
<td>( t_j/J )</td>
</tr>
<tr>
<td>( c )</td>
<td>( -n_i/J )</td>
<td>( n_i/J )</td>
<td>( n_j/J )</td>
<td>( -n_j/J )</td>
</tr>
<tr>
<td>( d )</td>
<td>( n_i/J )</td>
<td>( n_i/J )</td>
<td>( -n_j/J )</td>
<td>( -n_j/J )</td>
</tr>
<tr>
<td>( \frac{\partial \phi}{\partial \chi} )</td>
<td>( \frac{\partial \phi}{\partial \eta} )</td>
<td>( \frac{\partial \phi}{\partial \eta} )</td>
<td>( \frac{\partial \phi}{\partial \zeta} )</td>
<td>( \frac{\partial \phi}{\partial \zeta} )</td>
</tr>
</tbody>
</table>
where $U_{w,j}$ is the solution vector at the boundary, and $I$ is an identity matrix of the same dimension as $U$.

As for the unstructured mesh case, a boundary constraint compiler reads boundary constraint descriptions from a file and generates code to set boundary constraints, which replace the last row in the block system. The structured mesh boundary constraint compiler is more restrictive than its unstructured analog: coupling between unknowns (other than as components of a vector of unknowns) is not allowed, and only first derivatives in the boundary coordinate system are allowed. See Appendix A.2 for the LALR(1) grammar description of structured boundary constraint description files.

On return from boundary constraint setup, the system of Eq. (12) looks like:

$$
\begin{bmatrix}
I & 0 & \frac{3t_b}{J} I \\
0 & I & \frac{-3t_b}{J} I \\
B_1 & 0 & B_2
\end{bmatrix}
\begin{bmatrix}
\frac{\partial U}{\partial n_{x,j}} \\
\frac{\partial U}{\partial U_{x,j}} \\
U_{w,j}
\end{bmatrix} = \begin{bmatrix} R_n \\ R_t \\ R \end{bmatrix},
$$

(12)

where the contents of $B_1$ and $B_2$ depend on the constraints applied. A simple block elimination yields

$$
\begin{bmatrix}
I & 0 & \frac{3t_b}{J} I \\
0 & I & \frac{-3t_b}{J} I \\
0 & 0 & B_2 - \frac{3t_b}{J} B_1
\end{bmatrix}
\begin{bmatrix}
\frac{\partial U}{\partial n_{x,j}} \\
\frac{\partial U}{\partial U_{x,j}} \\
U_{w,j}
\end{bmatrix} = \begin{bmatrix} R_n \\ R_t \\ R - B_1 R_n \end{bmatrix}.
$$

(13)

The last diagonal block is itself diagonal unless a constraint was applied to one or more components of a vector variable, in which case (for 2D), the block is tri-diagonal. Once this block is inverted, the solution at the boundary is known, and the gradient in boundary coordinates can be found by backsubstitution. The final step in the process is to rotate this gradient into Cartesian coordinates, because these are the coordinates in which the flux is specified.

2.3. Flux integration

Once accurate solution and gradient values are available at control volume boundaries, the fluxes can be computed and integrated. Flux computation is part of the problem physics and is performed by user code. Flux integration, however, is done within ANSLib and must match the solution reconstruction in accuracy. ANSLib integrates fluxes around each control volume by using Gauss quadrature. Each face in the mesh has a unique piece of control volume interface associated with it. For cell-centered control volumes, this interface is the face itself; for vertex-centered control volumes in 2D, each face is crossed by the interface between a single pair of control volumes. ANSLib follows the standard procedure of iterating over all faces to compute flux integrals. This is done in two loops, one for interior faces and one for boundary faces.\footnote{Note the explicit assumption that there is no constraint applied to the tangential derivative component.}
To ensure that the accuracy of the quadrature matches the accuracy of solution (and therefore flux) evaluations, we use one quadrature point per segment with linear reconstruction, and two quadrature points per segment for quadratic and cubic reconstruction. Standard Gauss point locations and weights for integration along a segment are used. This is shown schematically for interior control volumes in two dimensions in Fig. 6, including surface normals scaled by integration weights. At each quadrature point, the flux is calculated by user code based on the solution, solution gradient, location, and control volume interface normal. The result is then multiplied by a quadrature weight (the size of the control volume interface associated with the quadrature point) and accumulated in the flux integrals for the control volumes on either side of the interface.

Flux integration on the domain boundary is done similarly. For curved boundaries, Gauss quadrature points must be placed precisely on the boundary and spaced according to arc length rather than straight-line distance to achieve the nominal accuracy for schemes of order higher than two, as the distance between points on the curved boundary and on the straight line segment between two boundary vertices is $O(\Delta s^2)$ for a segment of length $\Delta s$. In general, the boundary representation must be at least as accurate as the desired flux integral accuracy. For example, for a fourth-order accurate solution, a curved boundary must be represented by at least a cubic curve.

For structured meshes, ANSLib presently supports only second-order accuracy, and so flux quadrature is quite simple: the flux is calculated at the midside of each edge, and the quadrature weight is simply the edge length. In this case, the interior faces are integrated in two groups, one for the east $(i + \frac{1}{2}, j)$ faces and one

![Fig. 6. Gauss quadrature for interior control volumes: (a) second-order accuracy vertex-centred control volumes, (b) third- and fourth-order accuracy vertex-centred control volumes, (c) second-order accuracy cell-centred control volumes, and (d) third- and fourth-order accuracy cell-centred control volumes.](image)
for the north \((i, j + \frac{1}{2})\) faces. Likewise, boundary faces are integrated in two groups, one for the east and west boundaries and one for the north and south boundaries.

### 2.4. Solution update

At present, ANSLib supports solution update by using multi-stage explicit local time advance. Subsets of this functionality also permit global time advance, with either user-provided or automatically computed time step. For explicit Euler time advance with a user-supplied global time step, solution update is trivial: the old solution is incremented by the flux integral times the time step. High-order time accuracy through use of Runge–Kutta schemes is equally easy. The remainder of this section addresses the more complicated task of estimating local and global time steps using feedback from user flux functions.

For local time stepping or global time stepping with an automatically computed time step, ANSLib uses wave speed estimates provided by the flux functions to estimate time step in parallel with flux integration. The general approach follows Barth [18], using integration of wave speeds around each control volume:

\[
\Delta t_i = \frac{V_i}{\int_{CV_i} C_{\text{max}} \, ds},
\]

where \(C_{\text{max}}\) is the speed of the fastest wave entering the control volume, and must be provided by the flux function as feedback to the flux integration routine. For advection-dominated problems on structured meshes, this formulation leads to the traditional CFL condition; on unstructured meshes, Eq. (15) also produces reasonable time step estimates.

In a variation on this theme, a pseudo-wave speed can be devised for diffusive terms; here the intent is to compute a pseudo-wave speed that results in a local time step for the one-dimensional heat conduction equation that scales correctly for explicit time advance schemes. The desired local time step is given by \(\Delta t_{\text{max}} \propto \Delta x^2\), so a “wave speed” of \(c \propto 1/\Delta x\) gives the right asymptotic behavior. For unstructured meshes, a generalized inverse distance measure is needed in place of \(1/\Delta x\). ANSLib uses the inverse of the distance between control volume reference points, projected onto the normal to the control volume boundary. That is, \(1/\Delta x\) is replaced by

\[
\frac{\left(\vec{x}_j - \vec{x}_i\right) \cdot \vec{n}_{ij}}{|\vec{x}_j - \vec{x}_i|^2},
\]

where \(\vec{x}_j - \vec{x}_i\) is the vector from the control volume reference location in control volume \(i\) to that in control volume \(j\), and \(\vec{n}_{ij}\) is the unit normal to the interface between the two control volumes, pointing into control volume \(j\). This diffusive wave speed is used successfully in combination with the advective wave speed in the example of Section 4.1 and alone in the example of Section 4.2.

Once the maximum permissible time step for each control volume is known, time advance can be performed either by local time stepping or by global time stepping. For global time stepping, the smallest of the local time steps is used as the global time step. In either case, the time step is estimated for the first stage of the Runge–Kutta time advance and frozen for the remaining stages.

### 2.5. Other ANSLib internals

#### 2.5.1. Integration over control volumes

ANSLib uses integration over control volumes to set initial conditions, evaluate source terms, and compute solution error. In each of these cases, Gauss quadrature is used, with the quadrature rule chosen appropriately for the context. Initial conditions are evaluated only once, and solution error is typically evaluated only at the end of a simulation, so efficiency is not an issue, and ANSLib uses its most accurate
available quadrature rule (sixth-order accurate). Source terms are evaluated using the same order of accuracy as the solution reconstruction.

Integration over control volumes is performed by iterating over all cells in the mesh. For cell-centered control volumes (both structured and unstructured), the function evaluations at each Gauss point are accumulated directly into the result for that cell. For vertex-centered control volumes for two-dimensional unstructured meshes, each cell overlaps three control volumes. In this case, the quadrature is performed over six smaller triangles within each cell (each triangle connects a vertex, an edge midside, and the triangle centroid) and the result is added to the integral for the appropriate vertex-centered control volume.

2.5.2. Storage of solutions and other field data

Solving any problem cast in finite-volume form requires storage of field variables—quantities defined on a per-control volume basis—including the solution, flux integral, etc. For scalar problems, these field variables generally have only one component, whereas for systems of PDE's, field variables will have multiple components per control volume. In addition, the preferred access methods for field variables will be different for unstructured versus structured meshes.

ANSLib addresses these issues through a FieldQuant class, which acts as a convenient wrapper around a single large one-dimensional array. This class acts much like an array of flexible dimension; this is possible because each FieldQuant stores internally the mesh type and dimensions of the mesh for which it stores data as well as the number of unknowns per control volume. (ANSLib stores all components of a field variable for a given control volume contiguously for efficient memory access.) Unstructured mesh data access uses the index of the control volume, while structured mesh data access uses the usual $i, j$ indexing, with ANSLib computing the storage index of the control volume internally.

Operations on FieldQuant's, including scalar multiplication, addition, subtraction, etc., are defined as member functions, acting directly on the internal one-dimensional array for efficiency.

2.5.3. Solution and error output

ANSLib also provides output capabilities for all mesh types. Any field variable can be written to a graphics file for further post-processing (currently, the VU [23] file format is used, but others will eventually be supported as well). Where appropriate, reconstruction can be performed to give accurate local values at mesh vertices instead of control volume average values. This is especially useful for cell-centered unstructured meshes and for structured meshes, because the solution is not associated with mesh vertices.

In addition, if the user provides a function describing the exact solution to a problem, ANSLib automatically computes the error in the steady-solution and outputs that as well.

3. Encoding problem physics for ANSLib

ANSLib interfaces with code describing problem physics uniformly, regardless of the problem in question. In C++, this is easy to accomplish by declaring a Physics class which defines the interface, then implementing each problem as an instance or a specialization of the Physics base class. As such, this class contains and provides a uniform external interface for flux and source term computation; boundary condition enforcement; initial condition setup; and calculation of auxiliary variables from the conserved quantities.

This section describes how user code for ANSLib is written. Throughout the discussion, examples will focus around implementation of the two-dimensional unsteady heat conduction equation with internal heat generation—including flux functions, boundary constraints, source terms, and initial condition. The problem and its boundary conditions are given by
\[
\frac{\partial T}{\partial t} = \alpha \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + S(T, x, y) \text{ for } (x, y) \in \Omega, \\
S(T, x, y) = \frac{1}{\sqrt{x^2 + y^2}} + \frac{T}{100} \text{ (source term)}, \\
T(x, y) = f(x, y) \text{ on } \partial \Omega_1 \text{ (Dirichlet)}, \\
\frac{\partial T(x, y)}{\partial n} = g(x, y) \text{ on } \partial \Omega_2 \text{ (Neumann)}, \\
AT + B \frac{\partial T(x, y)}{\partial n} = h(x, y) \text{ on } \partial \Omega_3 \text{ (mixed)}, \\
T(x, y, t = 0) = T_0(x, y) \text{ (initial condition)}. 
\]

Integrating over control volumes and applying Gauss’ theorem yields the control volume form of the heat conduction equation:

\[
\frac{dT_i}{dt} - \frac{1}{A_i} \oint_{CV_i} \alpha \nabla T \cdot \hat{n} ds = S_i,
\]

where again \( CV_i \) denotes the \( i \)th control volume with area \( A_i \), \( T_i \equiv \frac{1}{A_i} \int_{CV_i} T dA \), and \( \hat{n} \) is the outward-pointing unit normal vector.

3.1. Interior fluxes

When calculating the flux integral in Eq. (2), ANSLib must evaluate the flux at all Gauss quadrature points along the control volume boundary. The functions that provide this flux information are written by the user, and can calculate the flux using whatever method the user feels is appropriate. In the most general case, the flux will depend on location, on the direction normal to the control volume boundary, and on the solution and its derivatives. ANSLib automatically packages the information required by a particular flux function, whether that flux function is intended for interior fluxes or boundary fluxes.

For the heat conduction equation (Eq. (16)), the flux components in the \( x \)- and \( y \)-directions are proportional to the gradient of the solution, and the interior flux can be written schematically as

\[
\text{Flux} = -\alpha \left( \text{Grad}_\text{Left}[\text{Var}_T][x] + \text{Grad}_\text{Right}[\text{Var}_T][x]\right) \hat{n}_x \\
+ \left( \text{Grad}_\text{Left}[\text{Var}_T][y] + \text{Grad}_\text{Right}[\text{Var}_T][y]\right) \times \hat{n}_y)/2,
\]

where \( \alpha \), the gradient of the solution on both sides of the control volume boundary (Grad_Left and Grad_Right), and the unit normal \( \hat{n} \) are all known within the flux function. In this case, there is only one unknown, but the gradients are two-dimensional arrays with the general case in mind.

As described in Section 2.4, a pseudo-wave speed can be computed for the heat conduction equation to allow automatic selection of a stable time step for explicit time advance. This speed \( C_{\text{max}} \) is computed as

\[
C_{\text{max}} = \alpha D^{-1},
\]

where \( D^{-1} \) is the inverse distance referred to in Section 2.4, and is provided to the flux function by ANSLib.

Implementation information about the interface and communications between ANSLib and flux functions may be found in [24,25].
3.2. Source terms

To evaluate the average source term in Eq. (2), ANSLib must evaluate the source term at appropriate Gauss quadrature points in each control volume, as described in Section 2.5.1. Again, the function that evaluates the source term must be provided by the user, and the source can depend on location and on the solution and its derivatives. Communication between ANSLib and source functions is done in the same way as communication with flux functions.

In this particular case, the source term $S(T,x,y)$ can be written schematically as

$$\text{Source} = \frac{1}{(\text{Loc}[X] * \text{Loc}[X] + \text{Loc}[Y] * \text{Loc}[Y]) + \text{Soln}[\text{Var}.T]} / 100,$$

where Loc is a vector containing the coordinates of the integration point, and Soln is a vector containing the unknowns of the problem at the integration point. Because these integration points are associated with a specific control volume, the solution is single-valued.

3.3. Boundary conditions

If any boundary constraints must be enforced, the user must provide a boundary constraint description file, as described in Section 2.2. The Physics class interface includes calls for enforcing boundary constraints for both unstructured and structured mesh cases.

Whether constraints are specified or not, a boundary flux function must be provided, capable of computing the correct boundary flux for each possible boundary condition. For boundary conditions that are imposed solely as constraints, the boundary flux can often be simply an analytic calculation of the flux based on the known solution and gradient. For other cases, the code may be much more complex.

For our heat conduction example, we will show two different ways to implement the boundary conditions, one using boundary constraints for all boundary conditions, and the other using boundary constraints only for the Dirichlet boundary condition, while the Neumann and mixed conditions are enforced using special boundary fluxes. The two approaches produce the same solution for a given problem (within truncation error), although convergence rates may differ.

3.3.1. Enforcement through boundary constraints

In this case, all the boundary conditions are imposed by constraining the reconstruction at the domain boundary. The ANSLib boundary constraint description file can be written (using the format described in Section 2.2) as:

Variables T
BC Dirichlet 1 {
  T = f(x,y)
}
BC Neumann 2 {
  Dn T = g(x,y)
}
BC Mixed 3 {
  [A] * T + [B] * Dn T = h(x,y)
}

The boundary fluxes are extremely simple (mirroring the internal flux with single-valued data for the gradient) and are the same for each case:
3.3.2. Enforcement through boundary fluxes

For this problem, the Neumann and mixed boundary conditions can be enforced by computing the flux in specific ways. The Dirichlet condition must still be enforced as a solution constraint. Note that the Neumann and mixed boundary conditions are still listed in the boundary condition description file so that the integer tags identifying the boundary condition will be known to ANSLib.

For the Dirichlet condition, the same simple boundary flux can be used as before; for the other two boundary conditions, special fluxes are required:

\[
\text{case BC.Dirichlet:} \\
\text{Flux} = -\alpha (\text{Grad}[\text{Var}_T][x] \times \hat{n}_x + \text{Grad}[\text{Var}_T][y] \times \hat{n}_y)
\]

\[
\text{case BC.Neumann:} \\
\text{Norm}_\text{deriv} = g(x,y) \\
\text{Flux} = -\alpha \times \text{Norm}_\text{deriv} \\
\text{case BC.Mixed:} \\
\text{Norm}_\text{deriv} = (h(x,y) - A \times f(x,y))/B \\
\text{Flux} = -\alpha \times \text{Norm}_\text{deriv}
\]

3.4. Initial conditions

Whether the goal of the simulation is a time-accurate solution or a steady-state solution, some initial condition must be given, as no general guess will work for all problems. The initial condition must provide the solution as a function of the location, so that it can be integrated over control volumes. For this case, the initial condition is an encoding of \(T_0(x,y)\).

3.5. Change of variables

Finally, for some problems, it is advantageous to reconstruct the solution using a different set of variables than the conserved quantities. For example, for viscous compressible flow, reconstruction of the conserved variables \((\rho u \rho v \rho e)T\) is typically less convenient than reconstruction of the primitive variables \((PwT)\), because the latter set allows enforcement of positivity of pressure and temperature during reconstruction as well as producing all of the derivatives required for viscous flux calculation. To accommodate such problem-specific conversions, ANSLib allows user-defined conversion routines to map conserved variables onto reconstruction variables and vice versa. If these routines are not defined in the Physics object for a simulation, ANSLib makes no attempt to execute them.
4. Examples

The examples in this section are intended to demonstrate the flexibility and accuracy of ANSLib by exhibiting solutions for problems of graded computational difficulty, ranging up to a non-linear system of PDE’s.

4.1. Advection–diffusion equation

The first and simplest ANSLib application example is the advection–diffusion equation, which can be written as a PDE in cartesian coordinates as:

\[ \frac{\partial T}{\partial t} + \frac{\partial u T}{\partial x} + \frac{\partial v T}{\partial y} = \alpha \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right), \]

where the solution \( T \) may be temperature or any other passive scalar quantity and the velocity vector \( (u, v) \) is given. Transforming this to control volume form gives

\[ \frac{dT_i}{dt} + \frac{1}{A_i} \int_{CV_i} \left( uT - \alpha \frac{\partial T}{\partial x} \right) \cdot \hat{n} \, ds = 0. \] (22)

The interior flux function computes the integrand in Eq. (22). The advective fluxes are computed using data from the upwind side of the control volume boundary only, while the derivative terms are averaged at the CV boundary. Feedback from the flux function to ANSLib about time step is in the form of wave speeds, as described in Section 2.4, with the diffusive pseudo-wave speed augmenting the advective velocity.

We will solve Eq. (22) this problem for the specific case illustrated in Fig. 7 for \( \alpha = 0.01 \). At the inflow, the boundary condition is given by \( T(r, \theta = 0) = \sin \left( \frac{\pi \ln r}{\ln 2} \right) \). For this problem the boundary conditions were all imposed by using solution constraints. At the inflow (bottom) and walls (inner and outer arcs), these are Dirichlet constraints, whereas the outflow has a Neumann constraint. Boundary fluxes are computed using the same flux formula as the interior fluxes, but with single-valued data.

The initial condition is simply the solution to the pure advection problem with the same inflow boundary condition: \( T(r, \theta) = \sin \left( \frac{\pi \ln r}{\ln 2} \right) \) for all \( \theta \). For this problem, neither a source term nor conversion function was needed.

Fig. 7. Geometry and boundary conditions for annular test case. The inflow boundary condition is given in the text.
Once the Physics object containing these functions is defined, the only difference between the ANSLib simulations for unstructured meshes (with either cell-centered or vertex-centered control volumes) and for structured meshes is the input file read at run time. All variations in discretization, flux integration, boundary condition enforcement, time advance, et cetera, are handled automatically by ANSLib.

For a velocity field of \((v_r, v_0) = (0, u_0/r)\) and the given boundary conditions, the advection–diffusion equation in cylindrical coordinates is separable, and the solution is given by

\[
T(r, \theta) = \sin \left( \frac{\pi \ln r}{\ln 2} \right) s_1 \exp \left( s_1 \frac{\pi}{2} + s_2 \theta \right) - s_2 \exp \left( s_2 \frac{\pi}{2} + s_1 \theta \right),
\]

where

\[
s_{1,2} = \pm \sqrt{u_0^2 + \left( \frac{2\pi}{\ln 2} \right)^2}.\]

Note that, although the solution is given in terms of \(r\) and \(\theta\), we solved the problem numerically by using the Cartesian advection–diffusion equation.

A range of cases were run for this problem, covering all three supported two-dimensional mesh types; order of accuracy from second- to fourth-order accurate for the unstructured cases; and several mesh resolutions. Results were uniformly good, and are shown in Fig. 8. The convergence of the error norms for the four cases illustrated in the figure are all unremarkable—advertised orders of accuracy were achieved.

For a given number of control volumes, the structured mesh solution is somewhat more accurate than an unstructured mesh, second-order accurate, cell-centered control volume solution. Also shown in Fig. 8 are the measured convergence rates for the \(L_2\) error norm for all families of solutions to this problem; clearly, ANSLib’s discretization scheme is successful in obtaining solutions of the expected order of accuracy for this problem.

4.2. Solid mechanics: a plate with a hole

As a second example of the capabilities of ANSLib, we will compute the stress concentration factor for a square plate with a small hole under plane stress assumptions. The problem is symmetric about both the \(x\) and \(y\) axes, so only the upper right quadrant of the physical plate is simulated. This is a problem that is typically solved by using finite-element methods, but as we shall see, the formulation of the problem is also amenable to finite-volume computation. For this problem, we solve the equilibrium equations

\[
\begin{align*}
\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} &= 0, \\
\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= 0
\end{align*}
\]

for the displacements \((u, v)\). Under the plane stress assumption, we can write the stresses in terms of the displacements as:

\[
\begin{pmatrix}
\sigma_{xx} \\
\sigma_{yy} \\
\sigma_{xy}
\end{pmatrix} = \frac{E}{1 - v^2} \begin{pmatrix}
\frac{\partial u}{\partial x} + v \frac{\partial v}{\partial y} \\
\frac{\partial v}{\partial y} + v \frac{\partial u}{\partial x} \\
\frac{1 + v}{4} \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)
\end{pmatrix}.
\]
where $E$ is the modulus of elasticity and $v$ is Poisson’s ratio. This problem can easily be cast as a steady-state problem in finite-volume form:

$$
E \frac{1 - v^2}{1 + v} \int_{CV} \left\{ \left( \frac{\partial u}{\partial x} + v \frac{\partial v}{\partial y} \right) \mathbf{n}_x + \left( \frac{1 + v}{4} \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right) \mathbf{n}_y \right\} ds = 0.
$$

(23)

The interior fluxes are implemented as a direct translation of this, with gradients evaluated using the average of the values in the two control volumes at the Gauss point. Once again, a pseudo-wave speed is calculated to allow time step estimation. A time step is needed because ANSLib solves this not as a steady problem but as a relaxation problem, with the flux integral used to update the solution in a pseudo-time marching approach.\(^6\)

The top of the plate and the curved inside of the hole are both free boundaries: no stress is applied there. This is enforced by explicitly setting the flux (i.e., stress) at these boundaries to zero. At the right side, the

---

\(^6\) This approach is, not surprisingly, an efficiency problem, not least because for this case the physical unsteady problem contains the acceleration (second derivative of displacement) rather than the velocity (first derivative). As will be discussed later, efficiency improvements for ANSLib are currently being aggressively pursued.
tensile stress boundary condition is applied by setting $\sigma_{xx}$ to one when evaluating the boundary flux there. At the two symmetry lines (the left and bottom of the quarter plate), the normal displacement is constrained to be zero, as is the normal derivative of the tangential displacement. The boundary flux at the symmetry boundaries is calculated by using the analytic flux from Eq. (23).

This problem requires no source term or variable translation. The initial condition is set to zero displacement everywhere; convergence would doubtless be faster if the initial condition were the displacement in a similar plate with no hole.

Results for several combinations of accuracy and mesh type are summarized in Table 2. For both second-order accurate cell-centered and third-order accurate vertex-centered cases, the solution approaches the correct solution (a stress concentration factor of three) as the mesh is refined. Not surprisingly, the third-order accurate solution is significantly better for a given mesh resolution than the second-order accurate solution.

### Table 2

<table>
<thead>
<tr>
<th>Mesh type</th>
<th>Order</th>
<th>Resolution (points on hole)</th>
<th>Max $\sigma_{xx}$</th>
<th>Stress concentration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell-centered</td>
<td>2</td>
<td>9</td>
<td>2.27</td>
<td>2.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>2.70</td>
<td>2.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24</td>
<td>2.87</td>
<td>2.80</td>
</tr>
<tr>
<td>Vertex-centered</td>
<td>3</td>
<td>9</td>
<td>2.51</td>
<td>2.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>2.90</td>
<td>2.83</td>
</tr>
</tbody>
</table>

4.3. Compressible flow: Prandtl–Meyer expansion

Non-linear problems require major additional effort in the design for many generic PDE solution systems; see, for example, Eyheramendy and Zimmermann’s description of the challenges in extending their finite-element system to non-linear problems [12]. Our system does not suffer from this difficulty, because the non-linearity is contained within the flux function, and therefore lies solely in the domain of the application programmer, rather than within ANSLib. The underlying numerical processes—solution reconstruction, flux integration, time advance, etc.—are all unaffected by the non-linearity of the physical problem. As such, no modifications to the ANSLib core were required to allow solution of non-linear problems.

As an example, consider the compressible flow past an expansion angle, shown in Fig. 9; this is the well-known Prandtl–Meyer expansion fan. The physics of the flow is described by the Euler equations, which are written in two-dimensional conservation-law form as:

![Fig. 9. Schematic of a Prandtl–Meyer expansion fan.](image-url)
\[ \frac{\partial}{\partial t} \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} \rho \frac{\partial u}{\partial x} + \frac{\partial P}{\partial x} \\ \rho u \frac{\partial u}{\partial x} + P \frac{\partial u}{\partial x} \\ \rho v \frac{\partial v}{\partial x} + P \frac{\partial v}{\partial x} \\ u(E + P) \end{pmatrix} + \frac{\partial}{\partial y} \begin{pmatrix} \rho \frac{\partial v}{\partial y} + \frac{\partial P}{\partial y} \\ \rho u \frac{\partial v}{\partial y} + P \frac{\partial v}{\partial y} \\ \rho v \frac{\partial u}{\partial y} + P \frac{\partial u}{\partial y} \\ v(E + P) \end{pmatrix} = 0, \]

where \((\rho \rho u \rho v E)^T\) are the densities of mass, x-momentum, y-momentum, and energy, respectively. The energy is related to the pressure \(P\) by the perfect-gas equation of state: \(E = P/(\gamma - 1) + \rho (u^2 + v^2)/2\), with \(\gamma\) the ratio of specific heats for the gas. The finite-volume formulation for this problem is

\[
\frac{d}{dt} \begin{pmatrix} \bar{p}_i \\ \bar{p}u_i \\ \bar{p}v_i \\ \bar{E}_i \end{pmatrix} + \frac{1}{A_i} \int_{CV_i} \left\{ \begin{pmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E + P) \end{pmatrix} \hat{n}_x + \begin{pmatrix} \rho v \\ \rho v^2 + P \\ \rho uv \\ v(E + P) \end{pmatrix} \hat{n}_y \right\} ds = 0.
\]

For this example, interior fluxes have been calculated by using Roe's numerical flux [26] because of its superior properties for flows with shocks and contact discontinuities. This is an example of a problem in which a physically inspired flux calculation is much better than simply calculating the analytic flux, and therefore a case where giving users the flexibility to easily include domain-specific knowledge in a simulation is imperative.

Implementation of the boundary conditions also requires significant physical insight. Wall boundaries are reasonably straightforward: a symmetry boundary condition is applied, with the normal component of velocity; the normal pressure gradient; and the normal derivative of tangential velocity all equal to zero. After constraint enforcement, the analytic flux is used as a boundary flux. Inflow and outflow boundary conditions are enforced using characteristic theory. In all cases—subsonic or supersonic, inflow or outflow—the solution data at the boundary is determined by the physically correct combination of data from the interior of the flow and from the external boundary conditions. Once again, the analytic flux is used after boundary data is determined.

For this problem, data translation routines are provided so that ANSLib reconstructs the primitive variables \((\rho \rho u \rho v P)^T\) (temperature is never needed in this calculation, unlike the compressible viscous case mentioned in Section 3.5). As a bonus, this allows specification of initial conditions in primitive variables with ANSLib converting to conserved variables internally.
The flow through the geometry of Fig. 9 was calculated with an inflow Mach number of 2 using both unstructured and structured meshes of varying densities; all meshes were approximately uniform, despite the fact that adaptive refinement is of significant benefit for this problem. Unstructured simulations included both cell-centered and vertex-centered cases from second- to fourth-order accuracy. For this test case, the first expansion wave should reach the top wall at about \( x = 2.3 \), and the outflow Mach number at the upper wall should be about 2.76. Fig. 10 compares the Mach number along the upper wall for selected cases. In each case, agreement is good both for the location of the expansion wave and the outflow Mach number. Differences among solutions are largely attributable to differences in handling of the singular point in the flow at the expansion corner; errors in the flow at this corner are propagated along characteristics to the upper wall.

5. Conclusions

This paper has presented a new framework for numerical simulation of general PDE's using the finite-volume method. ANSLib contains the numerical infrastructure for finite-volume simulations, with the physics of particular problems defined by users. In particular, ANSLib handles internally the numerical issues surrounding recovery of high-order accurate solution and gradient data from the control volume averaged solution; boundary condition enforcement; integration of user-supplied fluxes and source terms; explicit multistage time advance; and solution output. The user specifies physics of their problem by providing code snippets to compute interior and boundary fluxes, source terms, and initial conditions; and by specifying constraints on the solution at the boundaries. The interface between ANSLib and user physics modules is through a C++ base class, providing uniformity regardless of the problem.

ANSLib has been tested for several problems, including the advection, advection–diffusion, Poisson, plane-stress solid mechanics, incompressible laminar Navier–Stokes, and compressible Euler equations. Examples were presented here for several of these problems to demonstrate the flexibility of the ANSLib framework.

5.1. Future plans for ANSLib

The work presented here is intended only as a first step towards a general-purpose finite-volume solver for PDE's. Several issues which we are currently working to address include:

Multi-physics problems: ANSLib support for coupled problems, which will dramatically simplify development of coupled simulations, is nearly complete. This includes support for multiple physical phenomena within a single subdomain (for example, computational solid mechanics with thermal stresses coupled to a heat conduction simulation) and for coupling between different physical problems in adjacent subdomains (such as conjugate heat transfer problems. Preliminary results from this work may be found in \([24,25]\)).

Computational efficiency: Clearly, multistage explicit schemes, no matter how cunningly designed, are inadequate for solving stiff PDE's. Work is underway to implement multigrid convergence acceleration within the ANSLib framework. Support for implicit time advance, in the form of matrix-free Krylov methods, is also under development. A further goal in this area is parallelization of ANSLib. Both implicit time advance and parallelization will likely use PETSc [27] for lower-level support.

Acknowledgements

This work has been supported by the Canadian Natural Science and Engineering Research Council under grant OPG-0194467. The Physics object for the computational solid mechanics problem shown here was implemented by Charles Boivin.
Appendix A. Grammars for boundary constraint description files

In the following grammar specifications, identifiers in ALL CAPITAL letters are tokens returned by the file lexer, which is common to both the structured and unstructured boundary constraint compilers. Each token is accompanied by a string value, which is manipulated in the action associated with a rule. Precedence rules for arithmetic operators are as expected, with derivatives having the highest priority.

Tokens returned by the lexer include the following:

- **LITERAL**_CODE_ this is code that is transferred directly to the output C++ file
- **VAR**_NAME_ the symbolic name of a variable (P, u, T, etc.)
- **VEC**_NAME_ identifies a variable that is a component of a physical vector quantity; the number of components varies with spatial dimension of the problem
- **BC** the symbolic name of a boundary condition (Wall, Inflow, etc.)
- **BC_INDEX** an integer tag to associate the boundary condition with an appropriate part of the mesh boundary
- **NUM** anything that is not identifiable in a constraint as a VARIABLE or DERIV is labeled as a pure number
- **VARIABLE** the use of one of the **VAR**_NAME_’s in a constraint
- **DERIV** a derivative operator

### A.1. Unstructured mesh boundary constraints

```
file: literal_code var_name_list bc_list
literal_code: /* Might not be any literal code */
    | literal_code LITERAL_CODE
var_name_list: var_names vector_vars
var_names: VAR_NAME | var_names VAR_NAME
vector_vars: /* Might have no vector */
    | VEC_NAME VEC_NAME
    | VEC_NAME VEC_NAME VEC_NAME
bc_list: /* Might not need any boundary constraints */
    | bc_list bc
bc: BC BC_INDEX {constraint_list}
constraint_list: /* Might have a BC with no constraints */
    | constraint_list constraint
constraint: constraint_expr '=' NUM
constraint_expr:
    VARIABLE
    | NUM '+' constraint_expr
    | '-' constraint_expr
    | constraint_expr '+' constraint_expr
    | constraint_expr '-' constraint_expr
    | DERIV constraint_expr
```

### A.2. Structured mesh boundary conditions

This grammar is nearly identical in rules (though not in actions) to the unstructured case. The only difference at the grammar specification level is that constraints are much less general in the structured mesh.
Because of this, ironically, the rules defining a legal constraint expression are longer than in the unstructured cases, so that all legal possibilities can be listed explicitly. In addition to the visible restrictions, only normal derivatives are allowed and a variable can only be combined with normal derivatives of itself; these restrictions are enforced by the rule action.

```plaintext
file:literal_code var_name_list bc_list
  literal_code: /* Might not be any literal code */
  | literal_code LITERAL_CODE
  | var_name_list: var_names vector_vars
  | var_names: VAR_NAME | var_names VAR_NAME
  | vector_vars: /* Might have no vector */
  | VEC_NAME VEC_NAME
  | VEC_NAME VEC_NAME VEC_NAME
  | bc_list: /* Might not need any boundary constraints */
  | bc_list bc
  bc: BC BC_INDEX {constraint_list}
  constraint_list: /* Might have a BC with no constraints */
  | constraint_list constraint
  constraint: constraint_expr = NUM
  constraint_expr:
  | value_expr
  | deriv_expr
  | value_expr '+' deriv_expr
  | deriv_expr '+' value_expr
  | value_expr '-' deriv_expr
  | deriv_expr '-' value_expr
  value_expr:
  | VARIABLE
  | NUM VARIABLE
  | '-' VARIABLE
  | '-' NUM VARIABLE
  deriv_expr:
  | DERIV VARIABLE
  | NUM DERIV VARIABLE
  | '-' DERIV VARIABLE
  | '-' NUM DERIV VARIABLE
```

**Appendix B. Complex boundary constraint example**

This appendix shows, step-by-step, how a complex boundary constraint is converted from differential form into a Taylor expansion constraint for a particular boundary Gauss integration point. The boundary constraint used in this example is the normal momentum constraint for a wall in incompressible flow (line 16 in Fig. 3). In all the Taylor tables shown below, cubic monomials and third derivative terms omitted for brevity. All blank entries in the tables are zero.
Step 1: Create an expansion for the normal component of velocity, $U_{\text{Norm}}$, from $u$ and $v$ and the given normal components $n_x$ and $n_y$. Specifically, $u_x = u n_x + v n_y$.

\[
\begin{array}{cccccccccccc}
  & u & u_x & u_y & u_{xx} & u_{xy} & v & v_x & v_y & v_{xx} & v_{xy} & v_{yy} \\
1 & n_x & & & & & n_y & & & & & \\
x & n_x & & & & & n_y & & & & & \\
y & & n_x & & & & n_y & & & & & \\
x^2 & & & n_x & & & n_y & & & & & \\
xy & & & & n_x & & & n_y & & & & \\
y^2 & & & & & n_x & & & n_y & & & \\
\end{array}
\]

Step 2: Take the normal derivative of this quantity, where the normal derivative is given by $(\partial / \partial n) = (n_x (\partial / \partial x) + n_y (\partial / \partial y))$. The rearrangement of coefficients is straightforward if one thinks of the table as a representation of a polynomial, which is differentiated and its coefficients distributed into a new table. This process is quite easy to automate.

\[
\begin{array}{cccccccccccc}
  & u & u_x & u_y & u_{xx} & u_{xy} & v & v_x & v_y & v_{xx} & v_{xy} & v_{yy} \\
1 & n_x^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_y^2 \\
x & n_x^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_y^2 \\
y & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 \\
x^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 \\
xy & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 \\
y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 & n_x n_y & n_y^2 \\
\end{array}
\]

Step 3: Take the normal derivative again and divide by the Reynolds number.

\[
\begin{array}{cccccccccccc}
  & u & u_x & u_y & u_{xx} & u_{xy} & v & v_x & v_y & v_{xx} & v_{xy} & v_{yy} \\
1 & n_x^3 / Re & 2 n_x^2 n_y / Re & n_x n_y^2 / Re & n_x n_y & n_y^2 / Re & 2 n_x n_y / Re & n_y^2 / Re & n_x / Re & n_y / Re & n_y^2 / Re \\
x & n_x^3 / Re & 2 n_x^2 n_y / Re & n_x n_y^2 / Re & n_x n_y & n_y^2 / Re & 2 n_x n_y / Re & n_y^2 / Re & n_x / Re & n_y / Re & n_y^2 / Re \\
y & n_x^3 / Re & 2 n_x^2 n_y / Re & n_x n_y^2 / Re & n_x n_y & n_y^2 / Re & 2 n_x n_y / Re & n_y^2 / Re & n_x / Re & n_y / Re & n_y^2 / Re \\
x^2 & n_x^3 / Re & 2 n_x^2 n_y / Re & n_x n_y^2 / Re & n_x n_y & n_y^2 / Re & 2 n_x n_y / Re & n_y^2 / Re & n_x / Re & n_y / Re & n_y^2 / Re \\
xy & n_x^3 / Re & 2 n_x^2 n_y / Re & n_x n_y^2 / Re & n_x n_y & n_y^2 / Re & 2 n_x n_y / Re & n_y^2 / Re & n_x / Re & n_y / Re & n_y^2 / Re \\
y^2 & n_x^3 / Re & 2 n_x^2 n_y / Re & n_x n_y^2 / Re & n_x n_y & n_y^2 / Re & 2 n_x n_y / Re & n_y^2 / Re & n_x / Re & n_y / Re & n_y^2 / Re \\
\end{array}
\]

Step 4: Now we turn our attention to the pressure term, computing the normal derivative of the pressure.

\[
\begin{array}{cccccccc}
  & P & P_x & P_y & P_{xx} & P_{xy} & P_{yy} \\
1 & n_x & n_y & n_x & n_y & n_x & n_y \\
x & n_x & n_y & n_x & n_y & n_x & n_y \\
y & & & & & & & & \\
x^2 & & & & & & & & \\
xy & & & & & & & & \\
y^2 & & & & & & & & \\
\end{array}
\]
Step 5: Subtract the result of Step 3 from the result of Step 4. In this table, rows and columns with all zero entries are omitted.

<table>
<thead>
<tr>
<th></th>
<th>$P_x$</th>
<th>$P_y$</th>
<th>$P_{xx}$</th>
<th>$P_{yy}$</th>
<th>$u_{xx}$</th>
<th>$u_{xy}$</th>
<th>$u_{yy}$</th>
<th>$v_{xx}$</th>
<th>$v_{xy}$</th>
<th>$v_{yy}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$n_x$</td>
<td>$n_y$</td>
<td>$-n_x^3/Re$</td>
<td>$-2n_y^2n_x/Re$</td>
<td>$-n_xn_y^2/Re$</td>
<td>$-2n_n^2n_y/Re$</td>
<td>$n_x^2n_y/Re$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td>$n_x$</td>
<td>$n_y$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 6: This Taylor table must be transcribed term-by-term as an expansion to be applied as a constraint. If $\Delta x \equiv \bar{x}_G - \bar{x}_i$ for a given Gauss point $G$ in control volume $i$, then we can write the constraint as:

$$P_n x + P_n y + P_{xx} n_x (x_G - x_i) + P_{yy} n_y (y_G - y_i) + P_{xy} [n_x (x_G - x_i) + n_y (y_G - y_i)]$$

$$- \frac{n_x}{Re} (u_{xx} n_x^2 + 2u_{xy} n_x n_y + u_{yy} n_y^2) - \frac{n_y}{Re} (v_{xx} n_x^2 + 2v_{xy} n_x n_y + v_{yy} n_y^2) = 0.$$ 

This equality constrains the values of derivatives of $P$, $u$, and $v$ that the reconstruction can compute for control volume $i$.

References