



ELSEVIER

Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

Comput. Methods Appl. Mech. Engrg. xxx (2002) xxx–xxx

**Computer methods  
in applied  
mechanics and  
engineering**[www.elsevier.com/locate/cma](http://www.elsevier.com/locate/cma)

# 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.

© 2002 Published by Elsevier Science B.V.

*IDT:* 18; 27; 73

*Keywords:* Finite-volume method; Numerical solution of partial differential equations; Software library; Generic solver

## 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-

*E-mail address:* [cfog@mech.ubc.ca](mailto:cfog@mech.ubc.ca) (C. Ollivier-Gooch).

*URLs:* <http://www.mech.ubc.ca/~cfog>, <http://tetra.mech.ubc.ca/ANSLab>.

31 main experts must either acquire or hire. A far better alternative for such people would be to write a small  
32 amount of computer code to describe the physics of a problem—which they thoroughly understand—and  
33 use a scientific computing toolkit to automatically, correctly, and efficiently handle the numerical aspects of  
34 the simulation.

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

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

48 Perhaps the most comprehensive support framework for finite-element solution of PDE's is DiffPack [13–  
49 15]. DiffPack is a set of tools designed to allow flexibility for end users in specifying physics, discretization  
50 methods, and numerical methods. In addition to allowing high-level control of problem physics, DiffPack  
51 also allows users to manipulate numerics by defining new finite elements (including integration points,  
52 mappings, and elemental matrices), for example, or by selecting different particulars for iterative solution of  
53 linear systems. As with the system described by Eyheramendy and Zimmermann, significant knowledge  
54 about the finite-element method is required to create a custom simulation in DiffPack. Specifically, users  
55 must derive and code the integrand for the weak form of the PDE, as well as providing code to implement  
56 boundary conditions.

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

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

---

<sup>1</sup> A preliminary version of this work was presented at CFD2K, the Canadian Society for Computational Fluid Dynamics annual conference for 2000 [16].

75 in ANSLib. At this point, ANSLib supports both unstructured and structured meshes, with spatial dis-  
 76 cretization accuracy as high as fourth-order for unstructured meshes; currently, structured mesh accuracy is  
 77 limited to second-order (see Sections 2.1 and 2.3). Time advance currently uses multistage explicit schemes  
 78 (see Section 2.4). Also included in Section 2.5 are descriptions of several important data structures and low-  
 79 level routines.

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

### 86 1.1. The scope of ANSLib

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

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

## 100 2. The anatomy of a generic finite-volume solver

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

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = S(U, x, y, z) \quad (1)$$

105 is transformed, for control volume  $CV_i$ , into

$$\int_{CV_i} \frac{\partial U}{\partial t} dV + \oint_{\partial CV_i} (F\hat{i} + G\hat{j} + H\hat{k}) \cdot \hat{n} dA = \int_{CV_i} S(U, x, y, z) dV.$$

107 Reversing the order of integration and differentiation for the first term; assuming fixed control volumes;  
 108 and defining  $\overline{(\cdot)}_i \equiv \frac{1}{V_i} \int_{CV_i} (\cdot) dV$ , we obtain for control volume  $i$

$$\frac{d\overline{U}_i}{dt} + \frac{1}{V_i} \oint_{\partial CV_i} (Fn_x + Gn_y + Hn_z) dA = \overline{S}_i. \quad (2)$$

110 The key spatial discretization step in the finite-volume method is numerical flux integration. Regardless of  
 111 the physical problem, any finite-volume scheme must calculate and integrate fluxes and source terms ac-

112 curately. In turn, high-order accurate flux calculations require that the solution be approximated more  
 113 accurately than by control volume averages; this is typically accomplished by *reconstructing* the solution in  
 114 each control volume as a low-degree polynomial. Flux and source term quadrature can be accurately  
 115 performed by using appropriate Gauss quadrature rules. In outline, a typical finite-volume solver can be  
 116 formulated as:

117 Initialize control volume averaged solution.  
 118 Until end time or steady state is reached, do:

- 119 (1) Reconstruct piecewise polynomial solution in each control volume from CV average data.  
 120 (2) Enforce boundary conditions.  
 121 (3) For each control volume interface in the mesh (both interior and boundary).  
 122 (a) Compute the flux at Gauss integration points, and  
 123 (b) Add contribution to the flux integrals of the control volumes incident on the interface.  
 124 (4) For each control volume.  
 125 (a) Compute the source term at Gauss integration points, and  
 126 (b) Integrate over the control volume.  
 127 (5) Advance the solution in time.

128 In this outline, initialization and steps 2, 3(a), and 4(a) require knowledge of the physics, while steps 1, 3(b),  
 129 4(b), and 5 are purely numerical. Also, the numerical parts of this process typically require more code and  
 130 are more complicated than the physics. These observations are the motivation for ANSLib. The remainder  
 131 of this section describes, following the outline given above, the numerical infrastructure of ANSLib.

## 132 2.1. Solution reconstruction

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

### 143 2.1.1. Least-squares reconstruction on unstructured meshes

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

- 146 (1) *The mean is conserved*: The computed control volume average should match the average of the poly-  
 147 nomial produced by reconstruction. Conservation of the mean acts as a constraint on the coefficients.  
 148 (2) *The reconstruction is accurate for smooth functions*: If some smooth function  $f(\vec{x})$  is averaged over the  
 149 control volumes in a mesh,  $k$ -exact reconstruction aims to reconstruct the function from its control vol-  
 150 ume averages to within  $O(\Delta x^{k+1})$ . Accuracy is achieved by requiring that the polynomial in each control  
 151 volume minimize in a least-squares sense the error in predicting the average function value in nearby  
 152 control volumes.

153 Numerous researcher have described  $k$ -exact reconstruction in detail (for example, see [17–20]); we will  
 154 not repeat the details here. Given the polynomial reconstruction, solution and gradient data can be pro-  
 155 vided as needed for all points in the control volume. Note that two values exist at points on the control  
 156 volume boundaries: one from each neighboring control volume. In general the difference between these  
 157 values is on the order of truncation error.

158 Each unstructured mesh representation in ANSLib (cell-centered and vertex-centered control volumes in  
 159 two dimensions; cell-centered CV's in three dimensions) <sup>2</sup> must provide the following information for each  
 160 control volume in support of reconstruction:

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

$$\overline{(x^l y^m z^n)}_i \equiv \frac{1}{V} \int_{CV_i} (x - x_i)^l (y - y_i)^m (z - z_i)^n dV, \quad (3)$$

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

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

$$\overline{(x^l y^m z^n)}_i \equiv \frac{1}{V} \int_{\partial CV_i} \frac{(x - x_i)^{l+1}}{l+1} (y - y_i)^m (z - z_i)^n \hat{n} dA, \quad (4)$$

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

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

### 185 2.1.2. Data extrapolation on structured meshes

186 For structured meshes, least-squares reconstruction would again be accurate, but is much less efficient  
 187 than the standard approach of direction-by-direction interpolation. At present, ANSLib uses linear ex-  
 188 trapolation to estimate solution data on both sides of each control volume boundary. That is, when esti-  
 189 mating the solution on the left side of the control volume interface at  $i + (1/2), j$ , ANSLib computes  
 190  $(\cdot)_{i+(1/2),j} = (3/2)(\cdot)_{i,j} - (1/2)(\cdot)_{i-1,j}$ ; an analogous formula is used to estimate the value on the right side.  
 191 Near boundaries, the stencils are adjusted as required. Gradients are computed in the usual way: by first

<sup>2</sup> Currently under development.

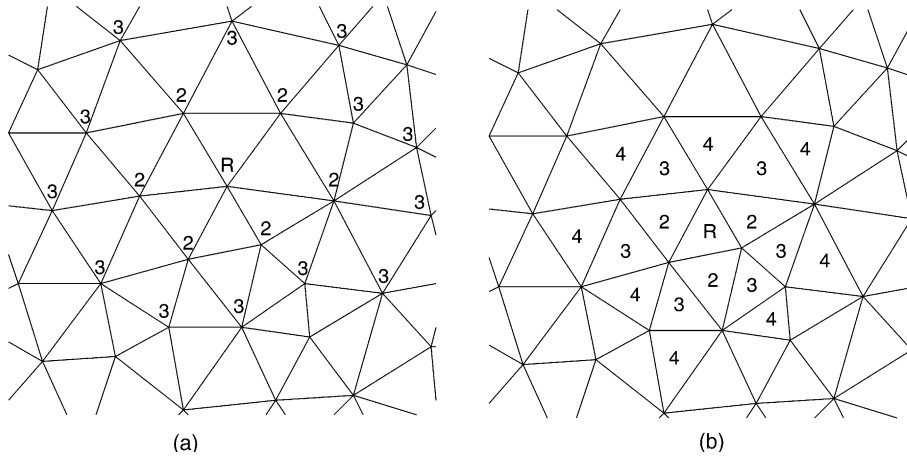


Fig. 1. Sample reconstruction stencils: (a) vertex-centered control volumes, (b) cell-centered control volumes.

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

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

## 199 2.2. Constraining the solution at the boundaries

200 The *specification* of boundary conditions is clearly a part of the physics of a particular problem. Nev-  
 201 ertheless, the *enforcement* of boundary conditions is a numerical issue, and can be approached generically.  
 202 For example, Dirichlet boundary conditions can all be enforced by matching the solution at the boundary  
 203 to some given value, whether the physical boundary condition is fixed temperature in a heat conduction  
 204 problem, zero turbulent kinetic energy for a turbulent flow, or an imposed displacement in a solid me-  
 205 chanics problem. Likewise, Neumann boundary conditions can be expressed either as a constraint on the  
 206 solution gradient at the boundary or as a special boundary flux, whether the physical boundary condition is  
 207 fixed heat flux in a heat conduction problem or an imposed stress in a solid mechanics problem.

208 As with reconstruction, the approach used within ANSLib to enforce boundary conditions is signifi-  
 209 cantly different for unstructured versus structured meshes. Nevertheless, specification of boundary condi-  
 210 tions—the user's only concern—is essentially the same regardless of mesh topology.

### 211 2.2.1. Unstructured mesh boundary constraints

212 One bonus of using least-squares reconstruction for unstructured meshes is that we can enforce Dirichlet,  
 213 Neumann, and mixed Dirichlet–Neumann boundary conditions by constraining the least-squares recon-  
 214 struction in control volumes adjacent to the boundary. This boundary condition enforcement scheme is  
 215 parsimonious, in the sense that boundary conditions are strictly enforced only at points where boundary  
 216 data is actually used—the Gauss integration points along the boundary. Elsewhere, the boundary condi-  
 217 tions are satisfied to within truncation error. This boundary constraint mechanism is quite flexible, in  
 218 principle allowing any linear boundary condition to be enforced directly as a solution constraint (Fig. 2).

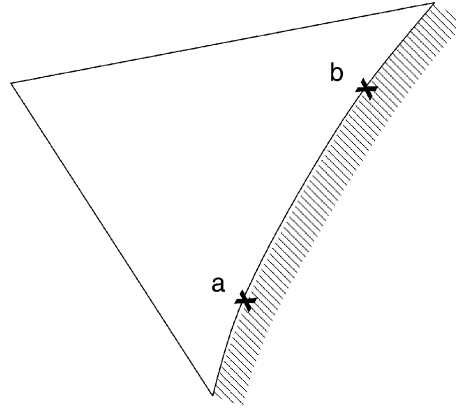


Fig. 2. Cell-centered control volume at the boundary, including Gauss integration points for third- and fourth-order accuracy.

219 More complex boundary conditions (e.g., characteristic boundary conditions for compressible flow) must  
 220 be handled, at least in part, by careful definition of boundary fluxes. Also, in certain cases (e.g., imposed  
 221 stress in solid mechanics), the solution constraints are complex enough that it is easier to enforce the  
 222 boundary conditions weakly, using a special boundary flux, instead of strongly, using a constraint.

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

$$f_1(\vec{x}_g) = \phi_i^R(\vec{x}_g) = \phi|_i + \frac{\partial\phi}{\partial x}\bigg|_i(x_g - x_i) + \frac{\partial\phi}{\partial y}\bigg|_i(y_g - y_i) + \frac{\partial^2\phi}{\partial x^2}\bigg|_i \frac{(x_g - x_i)^2}{2} + \frac{\partial^2\phi}{\partial x\partial y}\bigg|_i(x_g - x_i)(y_g - y_i) + \frac{\partial^2\phi}{\partial y^2}\bigg|_i \frac{(y_g - y_i)^2}{2} + \dots, \quad (5)$$

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

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

240 **2.2.1.1. Boundary constraint descriptions.** The boundary constraint description can be described using a  
 241 context-free LALR(1) grammar, which is compatible with standard parser generators; a summary of the  
 242 grammar (without rule actions) is given in Appendix A.1. This subsection gives a verbal description of the  
 243 file structure, using the sample boundary constraint description file shown in Fig. 3 as an example. This  
 244 particular example is drawn from the artificial compressibility formulation for incompressible fluid flow.

8

C. Ollivier-Gooch / Comput. Methods Appl. Mech. Engrg. xxx (2002) xxx-xxx

```

1  %{
2  double Re;
3  double InflowVel(double x,
                    double y)
4  { return (4*y*(1-y)); }
5  %}
6  Variables u v P
7  Vector u v
8  BC Inflow 1 {
9  u = InflowVel(x, y)
10 v = 0
11 Dn P = -2/Re
12 }
13 BC Wall 2 {
14 u = 0
15 v = 0
16 Dn P - [1/Re] *
17 }
18 BC Outflow 3 {
19 P = 0
20 Dn u = 0
21 Dn v = 0
22 }
23 BC Symmetry 4 {
24 U_Norm = 0
25 Dn U_Tang = 0
26 Dn P = 0
27 }
28 BC Slip 5 {
29 U_Norm = 0
30 }

```

Fig. 3. Sample boundary constraint description file (line numbers added for exposition).

245 The first section of the boundary constraint description file, set apart between `%{` and `%}` in conscious  
246 imitation of yacc grammar specification files, is copied directly into the output C++ file. This section is  
247 used for declaring variables and functions needed for boundary condition enforcement; in this case, the  
248 Reynolds number is declared as a global variable and a function is defined that returns a fully developed  
249 velocity profile for a channel of width one and  $u_{\max} = 1$ .

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

255 The remainder of the file is a series of boundary condition declarations. Each of these begins with a label  
256 (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  
257 for that same boundary condition (see Section 3.2 for information about boundary fluxes). This matching is  
258 dramatically simplified because the boundary constraint compiler adds a preprocessor definition for the  
259 label to its output C++ file. In this particular case, that definition would be

```
#define User_BC_Inflow 1.
```

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

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

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

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

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

279 *2.2.1.2. Taylor series manipulation.* At the basic level, a Taylor series expansion for a solution variable, (e.g.,  
280  $u$ ) can easily be created and stored, with a coefficient stored for each term into an array whose columns are  
281 derivatives in the Taylor series expansion and whose rows are monomials, as shown in Fig. 4. While this  
282 format is not compact in memory usage, it allows easy differentiation of Taylor series, even in arbitrary  
283 combination, because each row of coefficients in the expansion maps onto another row. For example, under  
284 differentiation by  $y$ , all coefficients in the  $y^2$ -row, regardless of the expansion terms to which they are at-  
285 tached, are doubled and placed in the  $y$ -row. This property is especially useful for more complicated ex-  
286 pressions.

287 Many boundary constraints require combinations of more than one variable. For example, the normal  
288 component of velocity (written as `U_norm` in the input file) is shorthand for  $u_n_x + v_n_y$ . To store such  
289 combinations of expansions, we use an extended storage scheme in which separate expansions for each  
290 variable are kept within a single object.

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

	$u$	$u_x$	$u_y$	$u_{xx}$	$u_{xy}$	$u_{yy}$	$u_{xxx}$	$u_{xxy}$	$u_{xyy}$	$u_{yyy}$
1	1									
$x$		1								
$y$			1							
$x^2$				$\frac{1}{2}$						
$xy$					1					
$y^2$						$\frac{1}{2}$				
$x^3$							$\frac{1}{6}$			
$x^2y$								$\frac{1}{2}$		
$xy^2$									$\frac{1}{2}$	
$y^3$										$\frac{1}{6}$

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

300 2.2.2. Structured mesh boundary constraints

301 The goal is the same here as for the unstructured mesh case, although the mechanism is adapted for  
 302 structured meshes, with direction-by-direction data interpolation instead of reconstruction. Three quan-  
 303 tities must be computed at the boundary for each unknown: its value, its normal derivative, and its tan-  
 304 gential derivative.<sup>3</sup> Just as in the unstructured case, boundary constraints require different values for these  
 305 than the unconstrained solution would produce (Fig. 5).

306 Making matters more complicated, the derivatives that are available are not the normal and tangential  
 307 derivatives but those in the computational space. For the general case of meshes not orthogonal to the  
 308 boundary, both computational derivatives contribute to both the normal and tangential derivatives.  
 309 Consider the case of a cell along the western boundary of the domain (minimum  $i$ ). The value of the so-  
 310 lution  $\phi$  at the boundary can be extrapolated, to second-order, as:

$$\phi_{w,j} = \frac{3}{2}\bar{\phi}_{1,j} - \frac{1}{2}\bar{\phi}_{2,j}. \tag{6}$$

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

<sup>3</sup> 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.

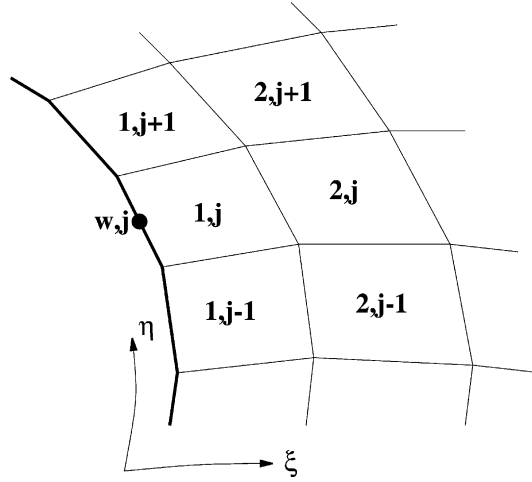


Fig. 5. Stencil for calculation of solution and derivatives at the west boundary of a structured mesh.

$$\left. \frac{\partial \phi}{\partial \xi} \right|_{w,j} = \frac{-6\phi_{w,j} + 7\bar{\phi}_{1,j} - \bar{\phi}_{2,j}}{2}, \quad (7)$$

$$\left. \frac{\partial \phi}{\partial \eta} \right|_{w,j} = \frac{3}{2} \frac{\bar{\phi}_{1,j+1} - \bar{\phi}_{1,j-1}}{2} - \frac{1}{2} \frac{\bar{\phi}_{2,j+1} - \bar{\phi}_{2,j-1}}{2}. \quad (8)$$

315 To get the derivatives in the normal and tangential directions, we apply the chain rule for differentiation  
 316 and replace metrics in physical space with metrics in computational space:

$$\frac{\partial \phi}{\partial n} = + \frac{t_\eta}{J} \frac{\partial \phi}{\partial \xi} - \frac{t_\xi}{J} \frac{\partial \phi}{\partial \eta}, \quad (9)$$

$$\frac{\partial \phi}{\partial t} = - \frac{n_\eta}{J} \frac{\partial \phi}{\partial \xi} + \frac{n_\xi}{J} \frac{\partial \phi}{\partial \eta}, \quad (10)$$

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

$$n_\xi \equiv \frac{\partial n}{\partial \xi} = x_\xi n_x + y_\xi n_y,$$

$$t_\xi \equiv \frac{\partial t}{\partial \xi} = x_\xi t_x + y_\xi t_y = y_\xi n_x - x_\xi n_y,$$

$$n_\eta \equiv \frac{\partial n}{\partial \eta} = x_\eta n_x + y_\eta n_y,$$

$$t_\eta \equiv \frac{\partial t}{\partial \eta} = x_\eta t_x + y_\eta t_y = y_\eta n_x - x_\eta n_y,$$

$$J = n_\xi t_\eta - n_\eta t_\xi = x_\xi y_\eta - x_\eta y_\xi.$$

321 If there is no constraint on the solution at the boundary, then, we can write a small system of equations  
 322 relating the wall value of the solution and its derivative components:

$$\begin{bmatrix} 1 & 0 & \frac{3t_\eta}{J} \\ 0 & 1 & \frac{-3n_\eta}{J} \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \frac{\partial \phi}{\partial n_{w,j}} \\ \frac{\partial \phi}{\partial t_{w,j}} \\ \phi_{w,j} \end{pmatrix} = \begin{pmatrix} R_n \\ R_t \\ R \end{pmatrix}, \quad (11)$$

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

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

$$\begin{aligned} \frac{\partial \phi}{\partial n} + 3a\phi_w &= a \left( \frac{7}{2} \bar{\phi}_{I1} - \frac{1}{2} \bar{\phi}_{I2} \right) + b \frac{\partial \phi}{\partial \chi}, \\ \frac{\partial \phi}{\partial t} + 3c\phi_w &= c \left( \frac{7}{2} \bar{\phi}_{I1} - \frac{1}{2} \bar{\phi}_{I2} \right) + d \frac{\partial \phi}{\partial \chi}, \end{aligned}$$

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

334 When a constraint is imposed—whether Dirichlet, Neumann, or mixed—the relationship between the  
 335 solution and the normal and tangential derivatives at the wall (Eqs. (9) and (10)) are unchanged, and the  
 336 value of the solution at the wall is still required to compute them. The constraint, therefore, replaces the  
 337 extrapolated value of the solution at the boundary, which is the last equation in the system given in Eq.  
 338 (11).

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

347 To handle this situation in general, ANSLib computes the boundary values of the solution and its de-  
 348 rivatives by using a block system. Without boundary constraints, this system is given by

Table 1  
Parameters for structured-mesh boundary constraint equations

Boundary	West (Min $i$ )	East (Max $i$ )	South (Min $j$ )	North (Max $j$ )
$a$	$t_\eta/J$	$-t_\eta/J$	$-t_\xi/J$	$t_\xi/J$
$b$	$-t_\xi/J$	$-t_\xi/J$	$t_\eta/J$	$t_\eta/J$
$c$	$-n_\eta/J$	$n_\eta/J$	$n_\xi/J$	$-n_\xi/J$
$d$	$n_\xi/J$	$n_\xi/J$	$-n_\eta/J$	$-n_\eta/J$
$\partial$	$\partial$	$\partial$	$\partial$	$\partial$
$\frac{\partial}{\partial \chi}$	$\frac{\partial}{\partial \eta}$	$\frac{\partial}{\partial \eta}$	$\frac{\partial}{\partial \xi}$	$\frac{\partial}{\partial \xi}$

$$\begin{bmatrix} I & 0 & \frac{3t_\eta}{J}I \\ 0 & I & \frac{-3t_\xi}{J}I \\ 0 & 0 & I \end{bmatrix} \begin{pmatrix} \frac{\partial \mathbf{U}}{\partial n_{x,j}} \\ \frac{\partial \mathbf{U}}{\partial t_{x,j}} \\ \mathbf{U}_{w,j} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_n \\ \mathbf{R}_t \\ \mathbf{R} \end{pmatrix}, \quad (12)$$

350 where  $\mathbf{U}_{w,j}$  is the solution vector at the boundary, and  $I$  is an identity matrix of the same dimension as  $\mathbf{U}$ .

351 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  
352 system. The structured mesh boundary constraint compiler is more restrictive than its unstructured analog:  
353 coupling between unknowns (other than as components of a vector of unknowns) is not allowed, and only  
354 first derivatives in the boundary coordinate system are allowed. See Appendix A.2 for the LALR(1)  
355 grammar description of structured boundary constraint description files.

356 On return from boundary constraint setup, the system of Eq. (12) looks like:<sup>4</sup>

$$\begin{bmatrix} I & 0 & \frac{3t_\eta}{J}I \\ 0 & I & \frac{-3t_\xi}{J}I \\ B_1 & 0 & B_2 \end{bmatrix} \begin{pmatrix} \frac{\partial \mathbf{U}}{\partial n_{x,j}} \\ \frac{\partial \mathbf{U}}{\partial t_{x,j}} \\ \mathbf{U}_{w,j} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_n \\ \mathbf{R}_t \\ \mathbf{R} \end{pmatrix}, \quad (13)$$

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

$$\begin{bmatrix} I & 0 & \frac{3t_\eta}{J}I \\ 0 & I & \frac{-3t_\xi}{J}I \\ 0 & 0 & B_2 - \frac{3t_\eta}{J}B_1 \end{bmatrix} \begin{pmatrix} \frac{\partial \mathbf{U}}{\partial n_{x,j}} \\ \frac{\partial \mathbf{U}}{\partial t_{x,j}} \\ \mathbf{U}_{w,j} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_n \\ \mathbf{R}_t \\ \mathbf{R} - B_1 \mathbf{R}_n \end{pmatrix}. \quad (14)$$

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

### 366 2.3. Flux integration

367 Once accurate solution and gradient values are available at control volume boundaries, the fluxes can be  
368 computed and integrated. Flux computation is part of the problem physics and is performed by user code.  
369 Flux integration, however, is done within ANSLib and must match the solution reconstruction in accuracy.

370 ANSLib integrates fluxes around each control volume by using Gauss quadrature. Each face in the mesh  
371 has a unique piece of control volume interface associated with it. For cell-centered control volumes, this  
372 interface is the face itself; for vertex-centered control volumes in 2D, each face is crossed by the interface  
373 between a single pair of control volumes. ANSLib follows the standard procedure of iterating over all faces  
374 to compute flux integrals. This is done in two loops, one for interior faces and one for boundary faces.<sup>5</sup>

<sup>4</sup> Note the explicit assumption that there is no constraint applied to the tangential derivative component.

<sup>5</sup> For vertex-centered meshes, control volumes along the boundary share an interface in the interior of the mesh that is associated with the boundary face. ANSLib treats this fragment as part of the interior loop, so boundary faces are actually touched in both contexts for this mesh representation.

375 To ensure that the accuracy of the quadrature matches the accuracy of solution (and therefore flux)  
 376 evaluations, we use one quadrature point per segment with linear reconstruction, and two quadrature  
 377 points per segment for quadratic and cubic reconstruction. Standard Gauss point locations and weights for  
 378 integration along a segment are used. This is shown schematically for interior control volumes in two di-  
 379 mensions in Fig. 6, including surface normals scaled by integration weights. At each quadrature point, the  
 380 flux is calculated by user code based on the solution, solution gradient, location, and control volume in-  
 381 terface normal. The result is then multiplied by a quadrature weight (the size of the control volume interface  
 382 associated with the quadrature point) and accumulated in the flux integrals for the control volumes on  
 383 either side of the interface.

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

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

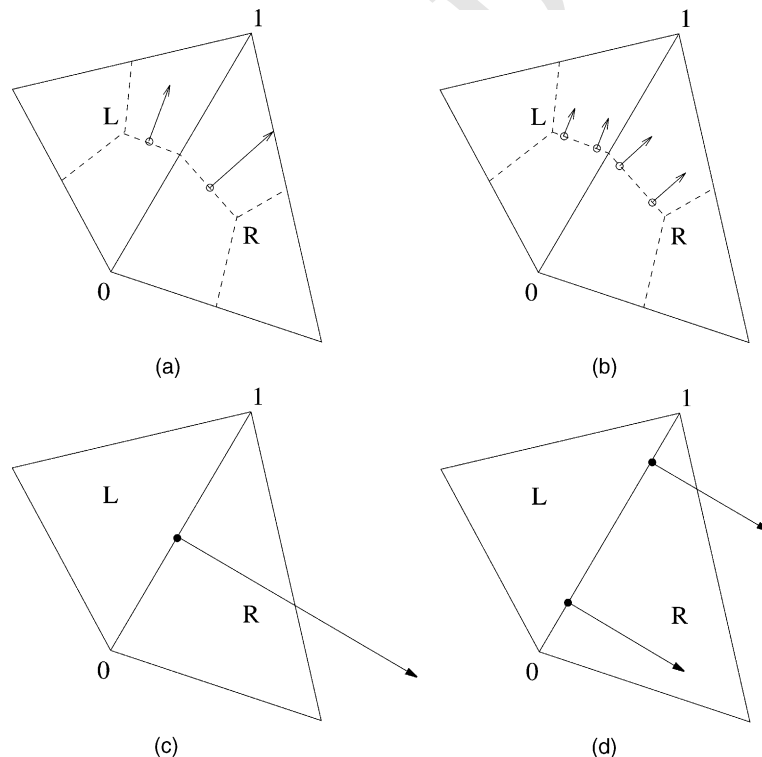


Fig. 6. Gauss quadrature for interior control volumes: (a) second-order accuracy vertex-centred control volumes, (b) third- and fourth-order accuracy vertex-centred control volumes, (c) second-order accuracy cell-centred control volumes, and (d) third- and fourth-order accuracy cell-centred control volumes.

394 for the north  $(i, j + \frac{1}{2})$  faces. Likewise, boundary faces are integrated in two groups, one for the east and  
 395 west boundaries and one for the north and south boundaries.

#### 396 2.4. Solution update

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

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

$$\Delta t_i = \frac{V_i}{\oint_{CV_i} C_{\max} ds}, \quad (15)$$

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

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

$$\frac{(\vec{x}_j - \vec{x}_i) \cdot \hat{n}_{ij}}{|\vec{x}_j - \vec{x}_i|^2},$$

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

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

#### 427 2.5. Other ANSLib internals

##### 428 2.5.1. Integration over control volumes

429 ANSLib uses integration over control volumes to set initial conditions, evaluate source terms, and  
 430 compute solution error. In each of these cases, Gauss quadrature is used, with the quadrature rule chosen  
 431 appropriately for the context. Initial conditions are evaluated only once, and solution error is typically  
 432 evaluated only at the end of a simulation, so efficiency is not an issue, and ANSLib uses its most accurate

433 available quadrature rule (sixth-order accurate). Source terms are evaluated using the same order of ac-  
434 curacy as the solution reconstruction.

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

#### 441 2.5.2. Storage of solutions and other field data

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

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

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

#### 456 2.5.3. Solution and error output

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

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

### 464 3. Encoding problem physics for ANSLib

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

471 This section describes how user code for ANSLib is written. Throughout the discussion, examples will  
472 focus around implementation of the two-dimensional unsteady heat conduction equation with internal heat  
473 generation—including flux functions, boundary constraints, source terms, and initial condition. The  
474 problem and its boundary conditions are given by

$$\frac{\partial T}{\partial t} = \alpha \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + S(T, x, y) \text{ for } (x, y) \in \Omega, \quad (16)$$

$$S(T, x, y) = \frac{1}{\sqrt{x^2 + y^2}} + \frac{T}{100} \text{ (source term),} \quad (17)$$

$$T(x, y) = f(x, y) \text{ on } \partial\Omega_1 \text{ (Dirichlet),} \quad (18)$$

$$\frac{\partial T(x, y)}{\partial n} = g(x, y) \text{ on } \partial\Omega_2 \text{ (Neumann),} \quad (19)$$

$$AT + B \frac{\partial T(x, y)}{\partial n} = h(x, y) \text{ on } \partial\Omega_3 \text{ (mixed),} \quad (20)$$

$$T(x, y, t = 0) = T_0(x, y) \text{ (initial condition).} \quad (21)$$

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

$$\frac{d\bar{T}_i}{dt} - \frac{1}{A_i} \oint_{CV_i} \alpha \nabla T \cdot \hat{n} ds = \bar{S}_i,$$

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

### 486 3.1. Interior fluxes

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

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

$$\text{Flux} = -\alpha((\text{Grad\_Left}[\text{Var\_T}][x] + \text{Grad\_Right}[\text{Var\_T}][x])\hat{n}_x \\ + (\text{Grad\_Left}[\text{Var\_T}][y] + \text{Grad\_Right}[\text{Var\_T}][y]) \times \hat{n}_y)/2,$$

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

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

$$C_{\max} = \alpha D^{-1},$$

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

503 Implementation information about the interface and communications between ANSLib and flux func-  
504 tions may be found in [24,25].

## 505 3.2. Source terms

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

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

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

513 where  $\text{Loc}$  is a vector containing the coordinates of the integration point, and  $\text{Soln}$  is a vector containing  
 514 the unknowns of the problem at the integration point. Because these integration points are associated with  
 515 a specific control volume, the solution is single-valued.

## 516 3.3. Boundary conditions

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

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

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

## 529 3.3.1. Enforcement through boundary constraints

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

```
533 Variables T
534 BC Dirichlet 1 {
535     T = f(x, y)
536 }
537 BC Neumann 2 {
538     Dn T = g(x, y)
539 }
540 BC Mixed 3 {
541     [A] * T + [B] * Dn T = h(x, y)
542 }
```

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

```

545 case BC_Dirichlet:
546 case BC_Neumann:
547 case BC_Mixed:
548     Flux =  $-\alpha(\text{Grad}[\text{Var}_T][x] \times \hat{n}_x + \text{Grad}[\text{Var}_T][y] \times \hat{n}_y)$ 

```

### 549 3.3.2. Enforcement through boundary fluxes

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

```

554 Variables T
555 BC Dirichlet 1 {
556     T = f(x,y)
557 }
558 BC Neumann 2 {
559 }
560 BC Mixed 3 {
561 }

```

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

```

564 case BC_Dirichlet:
565     Flux =  $-\alpha(\text{Grad}[\text{Var}_T][x] \times \hat{n}_x + \text{Grad}[\text{Var}_T][y] \times \hat{n}_y)$ 
566 case BC_Neumann:
567     Norm_deriv = g(x,y)
568     Flux =  $-\alpha \times \text{Norm\_deriv}$ 
569 case BC_Mixed:
570     Norm_deriv =  $(h(x,y) - A \times f(x,y))/B$ 
571     Flux =  $-\alpha \times \text{Norm\_deriv}$ 

```

### 572 3.4. Initial conditions

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

### 577 3.5. Change of variables

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

586 **4. Examples**

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

590 **4.1. Advection–diffusion equation**

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

$$\frac{\partial T}{\partial t} + \frac{\partial uT}{\partial x} + \frac{\partial vT}{\partial y} = \alpha \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right),$$

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

$$\frac{d\bar{T}_i}{dt} + \frac{1}{A_i} \oint_{CV_i} \begin{pmatrix} uT - \alpha \frac{\partial T}{\partial x} \\ vT - \alpha \frac{\partial T}{\partial y} \end{pmatrix} \cdot \hat{n} ds = 0. \quad (22)$$

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

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

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

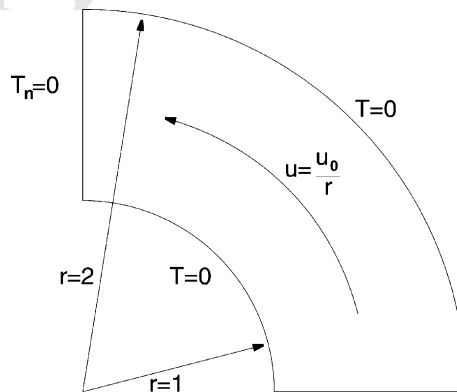


Fig. 7. Geometry and boundary conditions for annular test case. The inflow boundary condition is given in the text.

609 Once the `Physics` object containing these functions is defined, the only difference between the ANSLib  
 610 simulations for unstructured meshes (with either cell-centered or vertex-centered control volumes) and for  
 611 structured meshes is the input file read at run time. All variations in discretization, flux integration,  
 612 boundary condition enforcement, time advance, et cetera, are handled automatically by ANSLib.

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

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

616 where

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

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

620 A range of cases were run for this problem, covering all three supported two-dimensional mesh types;  
 621 order of accuracy from second- to fourth-order accurate for the unstructured cases; and several mesh  
 622 resolutions. Results were uniformly good, and are shown in Fig. 8. The convergence of the error norms for  
 623 the four cases illustrated in the figure are all unremarkable—advertised orders of accuracy were achieved.  
 624 For a given number of control volumes, the structured mesh solution is somewhat more accurate than an  
 625 unstructured mesh, second-order accurate, cell-centered control volume solution. Also shown in Fig. 8 are  
 626 the measured convergence rates for the  $L_2$  error norm for all families of solutions to this problem; clearly,  
 627 ANSLib’s discretization scheme is successful in obtaining solutions of the expected order of accuracy for  
 628 this problem.

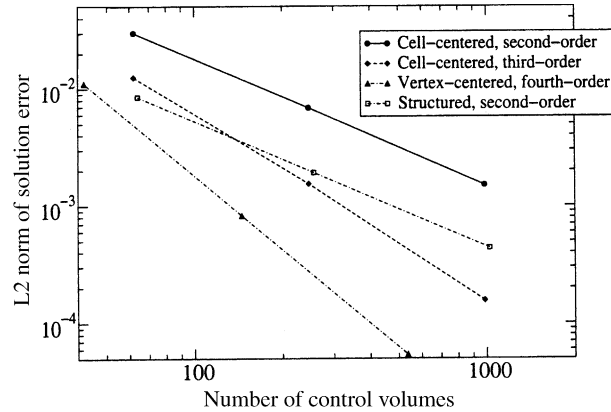
#### 629 4.2. Solid mechanics: a plate with a hole

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

$$\begin{aligned} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} &= 0, \\ \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= 0 \end{aligned}$$

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

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{pmatrix} \frac{\partial u}{\partial x} + \nu \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial y} + \nu \frac{\partial u}{\partial x} \\ \left(\frac{1+\nu}{4}\right) \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \end{pmatrix},$$



Nominal Order	Order of $L_2$ -norm		
	Cell	Vertex	Struct
2	2.18	2.08	2.14
3	3.31	3.27	—
4	3.74	4.17	—

Fig. 8. Results for advection–diffusion test case.

640 where  $E$  is the modulus of elasticity and  $\nu$  is Poisson's ratio. This problem can easily be cast as a steady-state  
641 problem in finite-volume form:

$$\frac{E}{1-\nu^2} \oint_{CV_i} \left\{ \left( \begin{array}{c} \frac{\partial u}{\partial x} + \nu \frac{\partial v}{\partial y} \\ \frac{1+\nu}{4} \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \end{array} \right) \hat{n}_x + \left( \begin{array}{c} \frac{1+\nu}{4} \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \\ \frac{\partial v}{\partial y} + \nu \frac{\partial u}{\partial x} \end{array} \right) \hat{n}_y \right\} ds = 0. \quad (23)$$

643 The interior fluxes are implemented as a direct translation of this, with gradients evaluated using the av-  
644 erage of the values in the two control volumes at the Gauss point. Once again, a pseudo-wave speed is  
645 calculated to allow time step estimation. A time step is needed because ANSLib solves this not as a steady  
646 problem but as a relaxation problem, with the flux integral used to update the solution in a pseudo-time  
647 marching approach.<sup>6</sup>

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

<sup>6</sup> 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.

Table 2  
Results for stress analysis of a plate with a hole

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

650 tensile stress boundary condition is applied by setting  $\sigma_{xx}$  to one when evaluating the boundary flux there.  
 651 At the two symmetry lines (the left and bottom of the quarter plate), the normal displacement is constrained  
 652 to be zero, as is the normal derivative of the tangential displacement. The boundary flux at the symmetry  
 653 boundaries is calculated by using the analytic flux from Eq. (23).

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

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

#### 661 4.3. Compressible flow: Prandtl–Meyer expansion

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

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

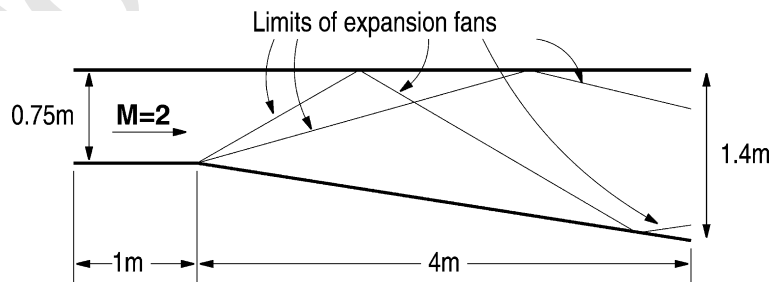


Fig. 9. Schematic of a Prandtl–Meyer expansion fan.

$$\frac{\partial}{\partial t} \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E + P) \end{pmatrix} + \frac{\partial}{\partial y} \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ v(E + P) \end{pmatrix} = 0,$$

674 where  $(\rho \rho u \rho v E)^T$  are the densities of mass,  $x$ -momentum,  $y$ -momentum, and energy, respectively. The  
 675 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  $\gamma$   
 676 the ratio of specific heats for the gas. The finite-volume formulation for this problem is

$$\frac{d}{dt} \begin{pmatrix} \bar{\rho}_i \\ \bar{\rho u}_i \\ \bar{\rho v}_i \\ \bar{E}_i \end{pmatrix} + \frac{1}{A_i} \oint_{\partial CV_i} \left\{ \begin{pmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E + P) \end{pmatrix} \hat{n}_x + \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ v(E + P) \end{pmatrix} \hat{n}_y \right\} ds = 0.$$

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

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

691 For this problem, data translation routines are provided so that ANSLib reconstructs the primitive  
 692 variables  $(\rho uv P)^T$  (temperature is never needed in this calculation, unlike the compressible viscous case  
 693 mentioned in Section 3.5). As a bonus, this allows specification of initial conditions in primitive variables  
 694 with ANSLib converting to conserved variables internally.

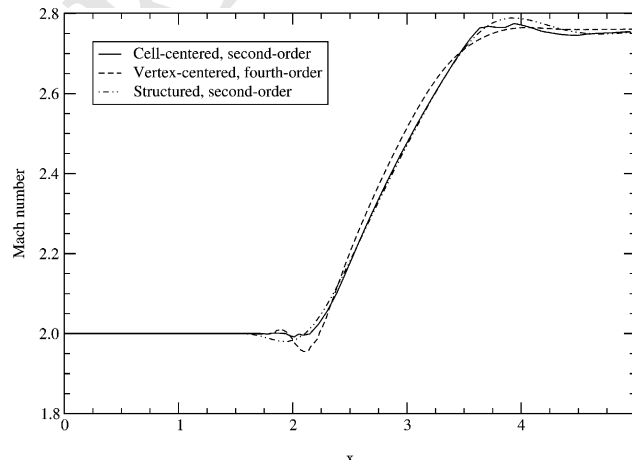


Fig. 10. Mach number along top wall for several cases for the Prandtl-Meyer expansion fan.

695 The flow through the geometry of Fig. 9 was calculated with an inflow Mach number of 2 using both  
696 unstructured and structured meshes of varying densities; all meshes were approximately uniform, despite the  
697 fact that adaptive refinement is of significant benefit for this problem. Unstructured simulations included  
698 both cell-centered and vertex-centered cases from second- to fourth-order accuracy. For this test case, the  
699 first expansion wave should reach the top wall at about  $x = 2.3$ , and the outflow Mach number at the upper  
700 wall should be about 2.76. Fig. 10 compares the Mach number along the upper wall for selected cases. In each  
701 case, agreement is good both for the location of the expansion wave and the outflow Mach number. Dif-  
702 ferences among solutions are largely attributable to differences in handling of the singular point in the flow at  
703 the expansion corner; errors in the flow at this corner are propagated along characteristics to the upper wall.

## 704 5. Conclusions

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

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

### 718 5.1. Future plans for ANSLib

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

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

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

## 731 Acknowledgements

732 This work has been supported by the Canadian Natural Science and Engineering Research Council  
733 under grant OPG-0194467. The `Physics` object for the computational solid mechanics problem shown  
734 here was implemented by Charles Boivin.

735 **Appendix A. Grammars for boundary constraint description files**

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

741 LITERAL\_CODE this is code that is transferred directly to the output C++ file  
 742 VAR\_NAME the symbolic name of a variable (P, u, T, etc.)  
 743 VEC\_NAME identifies a variable that is a component of a physical vector quantity; the number of com-  
 744 ponents varies with spatial dimension of the problem  
 745 BC the symbolic name of a boundary condition (Wall, Inflow, etc.)  
 746 BC\_INDEX an integer tag to associate the boundary condition with an appropriate part of the mesh  
 747 boundary  
 748 NUM anything that is not identifiable in a constraint as a VARIABLE or DERIV is labeled as a pure  
 749 number  
 750 VARIABLE the use of one of the VAR\_NAME's in a constraint  
 751 DERIV a derivative operator

752 *A.1. Unstructured mesh boundary constraints*

```

753 file: literal_code var_name_list bc_list
754 literal_code: /* Might not be any literal code */
755 | literal_code LITERAL_CODE
756 var_name_list: var_names vector_vars
757 var_names: VAR_NAME | var_names VAR_NAME
758 vector_vars: /* Might have no vector */
759 | VEC_NAME VEC_NAME
760 | VEC_NAME VEC_NAME VEC_NAME
761 bc_list: /* Might not need any boundary constraints */
762 | bc_list bc
763 bc: BC BC_INDEX {constraint_list}
764 constraint_list: /* Might have a BC with no constraints */
765 | constraint_list constraint
766 constraint: constraint_expr '=' NUM
767 constraint_expr:
768 VARIABLE
769 | NUM '*' constraint_expr
770 | '-' constraint_expr
771 | constraint_expr '+' constraint_expr
772 | constraint_expr '-' constraint_expr
773 | DERIV constraint_expr
  
```

774 *A.2. Structured mesh boundary conditions*

775 This grammar is nearly identical in rules (though not in actions) to the unstructured case. The only  
 776 difference at the grammar specification level is that constraints are much less general in the structured mesh

777 case. Because of this, ironically, the rules defining a legal constraint expression are *longer* than in the  
 778 unstructured cases, so that all legal possibilities can be listed explicitly. In addition to the visible restrictions,  
 779 only normal derivatives are allowed and a variable can only be combined with normal derivatives of itself;  
 780 these restrictions are enforced by the rule action.

```

781 file: literal_code var_name_list bc_list
782 literal_code: /* Might not be any literal code */
783 | literal_code LITERAL_CODE
784 var_name_list: var_names vector_vars
785 var_names: VAR_NAME | var_names VAR_NAME
786 vector_vars: /* Might have no vector */
787 | VEC_NAME VEC_NAME
788 | VEC_NAME VEC_NAME VEC_NAME
789 bc_list: /* Might not need any boundary constraints */
790 | bc_list bc
791 bc: BC BC_INDEX {constraint_list}
792 constraint_list: /* Might have a BC with no constraints */
793 | constraint_list constraint
794 constraint: constraint_expr '=' NUM
795 constraint_expr:
796 value_expr
797 | deriv_expr
798 | value_expr '+' deriv_expr
799 | deriv_expr '+' value_expr
800 | value_expr '-' deriv_expr
801 | deriv_expr '-' value_expr
802 value_expr:
803 VARIABLE
804 | NUM VARIABLE
805 | '-' VARIABLE
806 | '-' NUM VARIABLE
807 deriv_expr:
808 DERIV VARIABLE
809 | NUM DERIV VARIABLE
810 | '-' DERIV VARIABLE
811 | '-' NUM DERIV VARIABLE

```

## 812 Appendix B. Complex boundary constraint example

813 This appendix shows, step-by-step, how a complex boundary constraint is converted from differential  
 814 form into a Taylor expansion constraint for a particular boundary Gauss integration point. The boundary  
 815 constraint used in this example is the normal momentum constraint for a wall in incompressible flow (line  
 816 16 in Fig. 3). In all the Taylor tables shown below, cubic monomials and third derivative terms omitted for  
 817 brevity. All blank entries in the tables are zero.

818 *Step 1:* Create an expansion for the normal component of velocity,  $U_{\text{Norm}}$ , from  $u$  and  $v$  and the given  
 819 normal components  $n_x$  and  $n_y$ . Specifically,  $u_n = un_x + vn_y$ .

	$u$	$u_x$	$u_y$	$u_{xx}$	$u_{xy}$	$u_{yy}$	$v$	$v_x$	$v_y$	$v_{xx}$	$v_{xy}$	$v_{yy}$
1	$n_x$						$n_y$					
$x$		$n_x$						$n_y$				
$y$			$n_x$						$n_y$			
$x^2$				$\frac{n_x}{2}$						$\frac{n_y}{2}$		
$xy$					$n_x$						$n_y$	
$y^2$						$\frac{n_x}{2}$						$\frac{n_y}{2}$

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

	$u$	$u_x$	$u_y$	$u_{xx}$	$u_{xy}$	$u_{yy}$	$v$	$v_x$	$v_y$	$v_{xx}$	$v_{xy}$	$v_{yy}$
1		$n_x^2$	$n_x n_y$					$n_x n_y$	$n_y^2$			
$x$				$n_x^2$	$n_x n_y$					$n_x n_y$	$n_y^2$	
$y$					$n_x^2$	$n_x n_y$					$n_x n_y$	$n_y^2$
$x^2$												
$xy$												
$y^2$												

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

	$u$	$u_x$	$u_y$	$u_{xx}$	$u_{xy}$	$u_{yy}$	$v$	$v_x$	$v_y$	$v_{xx}$	$v_{xy}$	$v_{yy}$
1				$\frac{n_x^3}{Re}$	$\frac{2n_x^2 n_y}{Re}$	$\frac{n_x n_y^2}{Re}$				$\frac{n_x^2 n_y}{Re}$	$\frac{2n_x n_y^2}{Re}$	$\frac{n_y^3}{Re}$
$x$												
$y$												
$x^2$												
$xy$												
$y^2$												

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

	$P$	$P_x$	$P_y$	$P_{xx}$	$P_{xy}$	$P_{yy}$
1		$n_x$	$n_y$			
$x$				$n_x$		
$y$					$n_x$	
$x^2$						$n_y$
$xy$						
$y^2$						

830 *Step 5:* Subtract the result of Step 3 from the result of Step 4. In this table, rows and columns with all  
831 zero entries are omitted.

	$P_x$	$P_y$	$P_{xx}$	$P_{xy}$	$P_{yy}$	$u_{xx}$	$u_{xy}$	$u_{yy}$	$v_{xx}$	$v_{xy}$	$v_{yy}$
1	$n_x$	$n_y$				$-\frac{n_x^3}{Re}$	$-\frac{2n_x^2 n_y}{Re}$	$-\frac{n_x n_y^2}{Re}$	$-\frac{n_x^2 n_y}{Re}$	$-\frac{2n_x n_y^2}{Re}$	$-\frac{n_y^3}{Re}$
x			$n_x$	$n_y$							
y				$n_x$	$n_y$						

833 *Step 6:* This Taylor table must be transcribed term-by-term as an expansion to be applied as a constraint.  
834 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:

$$P_x n_x + P_y n_y + P_{xx} n_x (x_G - x_i) + P_{yy} n_y (y_G - y_i) + P_{xy} [n_y (x_G - x_i) + n_x (y_G - y_i)] - \frac{n_x}{Re} (u_{xx} n_x^2 + 2u_{xy} n_x n_y + u_{yy} n_y^2) - \frac{n_y}{Re} (v_{xx} n_x^2 + 2v_{xy} n_x n_y + v_{yy} n_y^2) = 0.$$

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

## 838 References

- 839 [1] A. Jameson, Artificial diffusion, upwind biasing, limiters and their effect on accuracy and multigrid convergence in transonic and  
840 hypersonic flows, AIAA paper 93-3359-CP, July 1993.
- 841 [2] D.L. Marcum, N.P. Weatherill, Aerospace applications of solution adaptive finite element analysis, *Comput. Aided Geom. Des.*  
842 12 (7) (1995) 709–731.
- 843 [3] M.D. Sherar, F.F. Liu, D.J. Newcombe, B. Cooper, W. Levin, W.B. Taylor, J.W. Hunt, Beam shaping for microwave wave-guide  
844 hyperthermia applicators, *Int. J. Radiat. Oncol. Biol. Phys.* 25 (5) (1993) 849–857.
- 845 [4] M.D. Sherar, H. Clark, B. Cooper, J. Kumaradas, F.F. Liu, A variable microwave array attenuator for use with single-element  
846 wave-guide applicators, *Int. J. Hypertherm.* 10 (5) (1994) 723–731.
- 847 [5] C.C. Vernon, J.W. Hand, S. Field, D. Machin, J.B. Whaley, J. van der Zee, W.L.J. van Putten, G.C. van Rhooon, J.D.P. van Dijk,  
848 D.G. Gonzalez, F.F. Liu, P. Goodman, M. Sherar, Radiotherapy with or without hyperthermia in the treatment of superficial  
849 localized breast cancer: results from five randomized controlled trials, *Int. J. Radiat. Oncol. Biol. Phys.* 35 (4) (1996) 731–744.
- 850 [6] R. Sacco, F. Saleri, Mixed finite volume methods for semiconductor device simulation, *Numer. Methods Part. Differential*  
851 *Equations* 13 (3) (1997) 215–236.
- 852 [7] J.T. Trattles, C.M. Johnson, Comparative study of finite element formulations for the semiconductor drift-diffusion equations,  
853 *Int. J. Numer. Methods Engrg.* 40 (18) (1997) 3405–3419.
- 854 [8] FEMLAB: An Introductory Course, 2002. Available from the FEMLAB website, <<http://www.femlab.com>>.
- 855 [9] T. Zimmermann, D. Eyheramendy, Object-oriented finite elements: I. Principles of symbolic derivations and automatic  
856 programming, *Comput. Methods Appl. Mech. Engrg.* 132 (1996) 259–276.
- 857 [10] D. Eyheramendy, T. Zimmermann, Object-oriented finite elements: II. A symbolic environment for automatic programming,  
858 *Comput. Methods Appl. Mech. Engrg.* 132 (1996) 277–304.
- 859 [11] D. Eyheramendy, T. Zimmermann, Object-oriented finite elements: III. Theory and application of automatic programming,  
860 *Comput. Methods Appl. Mech. Engrg.* 154 (1998) 41–68.
- 861 [12] D. Eyheramendy, T. Zimmermann, Object-oriented finite elements: IV. Symbolic derivations and automatic programming of  
862 nonlinear formulations, *Comput. Methods Appl. Mech. Engrg.* 190 (2001) 2729–2751.
- 863 [13] A.M. Bruaset, H.P. Langtangen, A comprehensive set of tools for solving partial differential equations: Diffpack, in: M. Dæhlen,  
864 A. Tveit (Eds.), *Numerical Methods and Software Tools in Industrial Mathematics*, Birkhäuser, 1997, pp. 63–92.
- 865 [14] A.M. Bruaset, E.J. Holm, H.P. Langtangen, Increasing the efficiency and reliability of software development for systems of PDEs,  
866 in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), *Modern Software Tools for Scientific Computing*, Birkhäuser, 1997, pp. 247–  
867 268.
- 868 [15] H.P. Langtangen, *Computational Partial Differential Equations—Numerical Methods and Diffpack Programming*, Lecture Notes  
869 in Computational Science and Engineering, Springer-Verlag, 1999.

- 870 [16] C.F. Ollivier-Gooch, ANSLib: A scientific computing toolkit supporting rapid numerical solution of practically any PDE, in:  
871 Proceedings of the Eighth Annual Conference of the Computational Fluid Dynamics Society of Canada, 2000, pp. 21–28.
- 872 [17] T.J. Barth, P.O. Frederickson, Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction,  
873 AIAA paper 90-0013, January 1990.
- 874 [18] T.J. Barth, Aspects of unstructured grids and finite-volume solvers for the Euler and Navier–Stokes equations, in: Unstructured  
875 Grid Methods for Advection-Dominated Flows, AGARD, Neuilly sur Seine France, 1992, pp. 6-1–6-61, AGARD-R-787.
- 876 [19] C.F. Ollivier-Gooch, Quasi-ENO schemes for unstructured meshes based on unlimited data-dependent least-squares reconstruc-  
877 tion, J. Comput. Phys. 133 (1997) 6–17.
- 878 [20] C.F. Ollivier-Gooch, High-order ENO schemes for unstructured meshes based on least-squares reconstruction, AIAA paper 97-  
879 0540, January 1997.
- 880 [21] D.A. Anderson, J.C. Tannehill, R.H. Pletcher, Computational Fluid Mechanics and Heat Transfer, Series in Computational  
881 Methods in Mechanics and Thermal Sciences, Hemisphere Publishing Corporation, New York, 1984.
- 882 [22] C.F. Ollivier-Gooch, M. Van Altena, A high-order accurate unstructured mesh finite-volume scheme for the advection–diffusion  
883 equation, J. Comput. Phys. 181 (2) (2002) 729–752.
- 884 [23] B. Ozell, C. Pic, VU: A scientific visualization program, 1998–2002. Available from <[http://www.cerca.umontreal.ca/vu/  
885 welcome.html](http://www.cerca.umontreal.ca/vu/welcome.html)>.
- 886 [24] C. Boivin, C. Ollivier-Gooch, A generic finite-volume solver for multiphysics problems I: Field coupling, in: Proceedings of CFD  
887 2002, CFD Society of Canada, 2002, pp. 49–54.
- 888 [25] C. Boivin, C. Ollivier-Gooch, A generic finite-volume solver for multiphysics problems II: Interface coupling, in: Proceedings of  
889 CFD 2002, CFD Society of Canada, 2002, pp. 55–60.
- 890 [26] P.L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys. 43 (1981) 357–372.
- 891 [27] S. Balay, W. Gropp, L.C. McInnes, B. Smith, PETSc, the Portable, Extensible Toolkit for Scientific Computation, 1998–2002.  
892 Available from <<http://www.mcs.anl.gov/petsc>>.