

Marching on the BL equations

Harvey S. H. Lam

February 10, 2004

Abstract

The assumption is that you are competent in Matlab or Mathematica. White's §4-7 starting on page 275 shows us that all generic boundary layer problems can easily be solved numerically.

1 Introduction

Everybody knows that there are three kinds of second order partial differential equations (PDEs): elliptic (e.g. Laplacian), hyperbolic (e.g. wave equation), and parabolic (e.g. diffusion equation). The mathematical properties of these three kinds of PDEs are distinct, and so are their numerical solution methodologies.

The original Navier-Stokes equations is elliptic, and the Prandtl's boundary layer equations is parabolic.

Ordinary differential equations (ODEs) are very straightforward. Given any system of ODEs (with minor caveats), it is straightforward to go back to the definition of a derivative and approximate it by a finite difference formula, and march onward one small step after another to obtain the numerical solution.

Parabolic PDEs can be solved numerically essentially by the same generic idea.

2 The BL equations

We will restrict attention to steady, two-dimensional laminar boundary layers with constant density and viscosity. For a change, we will use dimensional variables.

The momentum and continuity equations are (in terms of conventional notations):

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = U_\infty \frac{dU_\infty}{dx} + \nu \frac{\partial^2 u}{\partial y^2}, \quad (1)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \quad (2)$$

The initial and boundary conditions are:

$$x = 0 : u(0, y) = U_i(y), \quad v(0, y) = V_i(y) \quad (3)$$

$$y = 0 : u(x > 0, 0) = U_w(x), \quad v(x > 0, 0) = V_w(x), \quad (4)$$

$$y \rightarrow \infty : u(x > 0, \infty) \rightarrow U_\infty(x). \quad (5)$$

The functions $U_\infty(x), U_i(y), V_i(y), U_w(x), V_w(x)$ and the values of ρ and ν are assumed given.¹

Our job is to find $u(x > 0, y)$ and $v(x > 0, y)$ as a function of the given functions and parameters.

3 Rewriting the BL equations

Eqs.(1),(2) can be rewritten as follows:

$$\frac{\partial u}{\partial x} = \mathcal{N}(y; u, v), \quad (6)$$

$$\frac{\partial v}{\partial y} = -\mathcal{N}(y; u, v), \quad (7)$$

where \mathcal{N} is shorthand for:

$$\mathcal{N}(y; u, v) \equiv \frac{1}{u} \left(-v \frac{\partial u}{\partial y} + \frac{1}{2} \frac{dU_\infty^2}{dx} + \nu \frac{\partial^2 u}{\partial y^2} \right). \quad (8)$$

Note that at any streamwise x -station (such as the initial station), $\mathcal{N}(y; u, v)$ —which involves only partial derivatives with respect to y —is explicitly known. Note that in eq.(6)—which is an ODE with respect

¹Actually, $V_i(y)$ is not free for you to specify—since it can be computed from the given $U_i(y)$ (by eliminating $\partial u/\partial x$ in the momentum equation in favor of $\partial v/\partial y$ using the continuity equation—see eq.(7)). In order for this computed initial v to be “reasonable,” $U_i(y)$ is also not totally arbitrary—for example, for a no-slip wall the viscous term evaluated at the wall at the initial station must precisely cancel the pressure gradient term at the wall. Such “sloppiness” here are not expected to cause major problems. I will discuss this in class.

to $x-y$ is merely a “parameter,” while it is the independent variable in eq.(7)—which is an ODE for v .

Formally, you can use eq.(6) to march u in the positive x -direction, updating your v at any new station using eq.(7). The two equations are to be solved “simultaneously” during the march.

4 One thing at a time

Instead of solving eqs.(6-7) simultaneously, one may chose to

1. First use eq.(6) along with the known $v(x, y)$ to march forward in x one tiny step Δx , and compute $u(x + \Delta x, y)$,
2. Then use eq.(7) to solve for the new $v(x + \Delta x, y)$ which is consistent with the newly computed $u(x + \Delta x, y)$. Ready to repeat the cycle again!

The idea is that, when Δx is sufficiently small, this one-thing-at-a-time procedure is a good approximation to doing things simultaneously.

We can see that if we had imposed $V_i(0, y)$ in addition to $U_i(0, y)$ as initial conditions, the mathematics is *wrong*, but the damage is small.

5 Numerical issues

Obviously, once numerical solutions has been chosen as the methodology, we need to approximate derivatives by some finite difference formulas, and adopt some numerical integration schemes.

5.1 Explicit and Implicit Schemes

Consider the simple ODE problem:

$$\frac{du}{dt} = g(u) \tag{9}$$

where $g(u)$ is some differentiable function of u . For the moment, consider the linear case $g = -\lambda y$ where λ is a positive constant (what is its physical unit?). The exact solution to this linear problem is known to all:

$$u(t \geq 0) = u_o \exp(-t/\lambda). \tag{10}$$

It is seen that $u(t \rightarrow \infty) \rightarrow 0$ —in other words, u decays monotonically toward zero.

Now, let us approximate the derivative by:

$$\frac{du}{dt} = \frac{u(t + \Delta t) - u(t)}{\Delta t}, \quad (11)$$

and propose to solve the original ODE (for the linear case) by:²

$$\frac{u(t + \Delta t) - u(t)}{\Delta t} = -\lambda u(t). \quad (12)$$

The original ODE now becomes a finite difference equation:

$$u(t + \Delta t) = (1 - \lambda\Delta t)u(t). \quad (13)$$

After m steps of Δt , we have:

$$u(t + m\Delta t) = (1 - \lambda\Delta t)^m u(t). \quad (14)$$

If we keep the product $m\Delta t$ fixed while we increase m (or decrease Δt , it is easy to show that we recover the exact solution in the limit.³ But what happens when we use eq.(14) with a finite Δt . How small must Δt be before you feel somewhat more comfortable?

OK. What is the physical unit of Δt ? What dimensional parameters in the given problem do we have to cook up something that has the physical dimension of Δt ? One answer is obvious.

Looking at eq.(14), it is clear that it requires

$$|\lambda\Delta t| \ll 1 \quad (15)$$

to have a ghost of a chance to be good. What happens if you insist on using $\lambda\Delta t > 1$? You can easily see that your computed “solution” is going haywire.

Suppose, instead of eq.(12), we now choose:

$$\frac{u(t + \Delta t) - u(t)}{\Delta t} = -\lambda u(t + \Delta t). \quad (16)$$

²This is called the Euler algorithm, and is not highly regarded. Most people use Runge Kutta nowadays.

³It is useful to recall that the definition of e (2.718...) is

$$e \equiv \lim_{\epsilon \rightarrow 0} (1 + \epsilon)^{1/\epsilon}$$

Solving for $u(t + \Delta t)$, we have:

$$u(t + \Delta t) = \frac{u(t)}{1 + \lambda\Delta t}. \quad (17)$$

Similarly, after m steps, we have:

$$u(t + m\Delta t) = \frac{u(t)}{(1 + \lambda\Delta t)^m}. \quad (18)$$

So long as $\lambda > 0$, we see that this way of doing things (called the *implicit method*) avoids the unpleasant finite Δt restriction of the previous way (called the *explicit method*) of doing things.

Note: *both* explicit and implicit methods needs $\lambda\Delta t$ to be small in order to produce a good solution. Often one is interested in the “steady state” (fixed point) of an ODE system, and not particularly interested in the precise trajectories of how the fixed point is reached. For such problems the implicit method has advantages (but must pay dearly when $g(u)$ is highly nonlinear). From the programming point of view, the explicit method is often easier to code (couldn’t care less how nonlinear $g(u)$ may be). Yes, some people do advocate taking some average of the explicit and implicit algorithms.

What happens if you are dealing with a vector \mathbf{u} governed by a system of ODEs:

$$\frac{d\mathbf{u}}{dt} = \mathbf{g}(\mathbf{u}), \quad (19)$$

where $\mathbf{g}(\mathbf{u})$ is a vector of some complicated nonlinear function? You go and find the eigenvalues of the Jacobian of $\mathbf{g}(\mathbf{u})$ —they are your λ ’s (what is the physical dimension of the λ ’s?).

It is important to remark that the above exposition is meant only to high light the distinction between explicit and implicit schemes. For numerical integration of ordinary differential equations, neither schemes as discussed above are used in practice—there are much better schemes available in the literature.

5.2 PDE issues

When we discretize our y -space into N grid points Δy apart, then at each x we represent $u(x, y)$ by N unknowns: $u(x, n\Delta y)$, $n = 1, 2, \dots, N$. The values of u for $n = 0$ and $n = N + 1$ (and the value of v at $n = 0$) are known from the boundary conditions.

Once discretized, the diffusion PDE emerges as a system of N coupled first order ODEs. It is a big system. If you need $N = 256$ to resolve the solution you are looking for, you will have 256 unknowns (and ODEs) to work on.

Consider the classical diffusion equation:

$$u_* \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial y^2}, \quad (20)$$

where u_* is a positive constant (with physical unit of velocity).

Discretizing, we have:

$$u_* \frac{u(x + \Delta x, y) - u(x, y)}{\Delta x} = \nu \frac{u(x?, y + \Delta y) - 2u(x?, y) + u(x?, y - \Delta y)}{(\Delta y)^2} \quad (21)$$

where $x?$ indicates that we have not yet made up our mind whether to go explicit or implicit yet. We define the dimensionless α_* by:

$$\alpha_* = \frac{\nu(\Delta x/u_*)}{(\Delta y)^2} \quad (22)$$

and duly note the physical meaning of $\Delta x/u_*$.

If we go explicit, the marching formula (using Euler) is:

$$u(x + \Delta x, y) = \alpha_* u(x, y + \Delta y) - (2\alpha_* - 1)u(x, y) + \alpha_* u(x, y - \Delta y) \quad (23)$$

The “next” u is easily obtained by evaluating the right hand side.

If we go implicit, the marching formula (also using Euler) is:

$$-u(x, y) = \alpha_* u(x + \Delta, y + \Delta y) - (2\alpha_* + 1)u(x + \Delta x, y) + \alpha_* u(x + \Delta x, y - \Delta y) \quad (24)$$

This set of algebraic equations can be rewritten in term of matrix notation:

$$\mathbf{A}\mathbf{u} = \mathbf{r} \quad (25)$$

where \mathbf{A} is a $N \times N$ “tridiagonal matrix”:

$$A = \begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 & 0 \\ 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & 0 & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & 0 & 0 & a_N & b_N \end{pmatrix} \quad (26)$$

\mathbf{u} and \mathbf{r} are a N -dimensional column vectors. The matrix \mathbf{u} is completely specified by the three diagonals of $(\alpha_*$ dependent) numbers as shown. The “solution” \mathbf{u} is to be solved from this system of algebraic equations.

In *Numerical Recipes* (Cambridge University Press), you can find codes for the solving this class of problems which can be readily implemented in *Matlab*. This is posted on the course web page. In *Mathematica*, there is a “TridiagonalSolve” routine provided.

5.3 Specifics for the boundary layer equations

For the boundary layer equations, it is straightforward to work out the finite-difference equations for both the explicit and the implicit case. For the implicit formulation, the three diagonals of the \mathbf{A} are not constants and must be appropriately updated as one marches on.

White gave a good presentation on the numerics of the boundary layer equations (§4.7, starting on page 276). He showed that two dimensionless parameters, α and β —both dependent on x and y , need to be specified in order to proceed with the computation:

$$\alpha \equiv \frac{\nu \Delta x}{u (\Delta y)^2} \quad (27)$$

$$\beta = \frac{v \Delta x}{2u \Delta y} \quad (28)$$

For the explicit scheme, he gave the following constraints for their values:

$$\alpha < \frac{1}{2}, \quad \beta < \alpha. \quad (29)$$

For the implicit scheme, he indicated that the α restriction (on Δx) is no longer critical, while Δy needs to be small enough to resolve the boundary layer. Messing around a bit, you will see that the second constraint in his eq.(4-147) should continued to be respected.

6 The bottomline

The bottom line is: getting a numerical solution to the boundary layer equation is now not a big deal. Granted that the Euler scheme used in the discourse here is not the way to go (no respectable engineer would numerically integrate an ODE with Euler—for fear of contempt from

his colleagues). But refining the integration procedure (by tweaking the code to emulate Runge Kutta, for example) is no big deal.

As far as writing the code is concerned, there is no need to do any non-dimensionalization. If you have a practical problem looking for answers in terms of dimensional numbers, you write the code using dimensional variables, run the code, and print and graph the answers as they come out of the computer. If someone asked: "hey, what happens if I switched from water to air?" or "what if I build a contraction twice as large as the case we just ran?" or "what happens if the characteristic flow velocity is doubled?", a less than fully educated engineer would run the boundary layer code again and again. Under the assumption that the flow remained laminar, an educated engineer is able to answer these and other questions with precision after just one code run.

7 Homework

Use Matlab, and write a Matlab program to compute boundary layer solutions. Use the implicit scheme (via the **tridiagonal-solver** which is now posted on the course web page) along with Euler (you are welcome to try to do something better if you want to). Do the linearly *retarded* flow of Howarth (§4-5.1), and find the separation point (if any) applying the no-slip condition on the wall. Choose $U_i(y)=\text{constant}$ and $V_i(y) = 0$. Begin each computation cycle by marching $u(x, y)$ forward first, then compute the new $v(x, y)$. Submit

1. A print out of your Matlab (or Mathematica) code. Explain what you did with some words.
2. A discussion of your choice of your grid size (Δx and Δy), and the value of ν actually used in the code.
3. Show at least one plot of the velocity profile. How would you label your vertical coordinate?
4. After the code is debugged, do one additional problem of your choice, and show the computed velocity profiles. (This should take very little effort on your part).

If you prefer to use Mathematica, fine.