Symbolic and Numerical Computation for Artificial Intelligence

edited by

Bruce Randall Donald
Department of Computer Science
Cornell University, USA

Deepak Kapur
Department of Computer Science
State University of New York, USA

Joseph L. Mundy
AI Laboratory
GE Corporate R&D, Schenectady, USA

Academic Press
Harcourt Brace Jovanovich, Publishers
London San Diego New York
Boston Sydney Tokyo Toronto

ACADEMIC PRESS LIMITED
24–28 Oval Road
London NW1

US edition published by
ACADEMIC PRESS INC.
San Diego, CA 92101

This book is printed on acid-free paper

A catalogue record for this book is available from the British Library

ISBN 0-12-220535-9

Printed and Bound in Great Britain by
The University Press, Cambridge

Chapter 14

Symbolic/Numeric Techniques in Modeling and Simulation

Richard Zippel†

Department of Computer Science

Cornell University

Ithaca, NY 14853

Modeling and simulating collections of physical objects that are subject to a wide variety of physical forces and interactions is exceedingly difficult. The construction of a single simulator capable of dealing with all possible physical processes is completely impractical and, it seems to us, wrong-headed. Instead, we propose to build custom simulators designed for a particular collection of physical objects, where a particular set of physical phenomena are involved. For such an approach to be practical, an environment must be provided that facilitates the quick construction of these simulators. In this paper we describe the essential features of such an environment and describe in some detail how a general implementation of the weighted residual method, one of the more general classes of numerical integration techniques, can be used.

1. Introduction

We are interested in building software systems that simulate reality—especially when several different physical phenomena are involved in the simulation. Depending on the nature of the objects in a scene, their behavior may be governed by rigid body dynamics, fluid flow, quantum mechanics or other families of laws. The forces that act on these objects are gravitation and electromagnetism for macroscopic systems, and weak and strong interactions for systems at atomic scales. In addition, many observable properties of physical systems, including superconductivity, semiconductors and chemistry, are manifestations of statistical averages of detailed lower level behavior. These macroscopic phenomena are usually simulated through their own models, it being prohibitively expensive to simulate from first principles.

† This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014–88–K–0591, the National Science Foundation through grant IRI–9006137, the Office of Naval Research through contract N00014–89–J–1946 and in part by the U.S. Army Research Office through the Mathematical Science Institute of Cornell University.

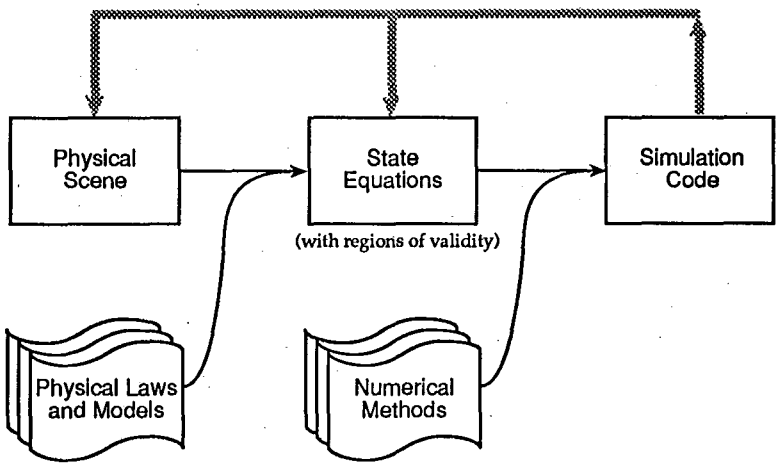

Figure 1. Simulation architecture.

Besides the computational costs, the complexity of dealing with all physical phenomena and mechanisms makes such a simulator ferociously difficult to build. Rather than build such a general purpose simulator we propose a new approach: to build special purpose simulators tuned for a particular configuration of physical objects and where a particular set of physical phenomena are involved. Such a simulator should be less complex than a general purpose simulator, which must be prepared for any eventuality. The specialized simulator will have only to consider a known system of equations with known parameters. It will consist of more straight-line code and will have fewer conditions, and thus should be easier to tune for high performance/parallel computer architectures. However, each new problem configuration requires the creation of a new simulator.

To make this endeavor practical, we are combining a wide array of techniques from artificial intelligence, computer algebra and compiler technology to provide an environment that vastly simplifies the process of building special purpose simulators. In effect, one builds a "simulator generator" that crafts a custom simulator for a particular configuration of physical objects, or *scene*. Such a system generates the particular set of differential equations that model the scene and converts these equations into a piece of code for the explicit equations that apply to the problem, as shown in figure 1. This approach has a number of advantages:

- More sophisticated mathematical techniques can be used to generate the systems of equations to be solved.
  - Conformal mapping techniques can be applied to the non-linear differential equations to simplify and regularize boundary conditions.
  - Averaging and perturbation techniques can be applied to reduce the order of the equations.
- Numerical techniques specialized to the equations being solved can be used.
- Software can be retargeted to different computer architectures relatively easily.

This new approach to simulation and modeling is replete with new problems to be studied and new technologies that need to be developed and applied. In this paper we sketch a general framework for simulation generation and consider some of the com-

ponents in detail. It should be noted that we are sketching a simulation and analysis framework that is to be used not only for Newtonian mechanics but also for problems that are driven by electrodynamics, relativistic mechanics and/or quantum mechanics as well as aggregate models like solid state theory, galactic dynamics, chemical kinetics and fluid dynamics. Thus one should exercise caution when extrapolating from experience in just one simulation domain.

The process of performing a simulation is shown in figure 1. We begin with the *observable scene* to be simulated. An observable scene is those properties of the system that can be observed and are independent of the physics used to model the behavior of the scene. Typically, this is the geometry and material properties of the objects in the scene. By applying the laws of physics to the scene, *state equations* are generated whose solution describes the evolution of the scene with time, within certain regions of validity. The state equations are then converted into code that numerically computes the scene's state changes. As time advances, the state equations may cease to be valid and must be changed. Similarly, the geometric or topological characteristics of the scene itself may change. These effects are indicated by the shaded "feedback" arrows in figure 1.

The state of a physical system is determined by the values of a set of *state variables*, which may include a subset of the observable parameters of the objects in a scene. The result of applying the physical laws to a scene are a set of *state equations* that constrain the state variables over time. For instance, clocked boolean logic circuits have a finite set of state variables, each of which ranges over {`true`,`false`}. Time is modeled by a sequence of discrete events occurring at clock edges and the state equations are boolean equations. For rigid body dynamics, there is a discrete set of state variables that have continuous values, time is modeled as a sequence of irregularly spaced events and the state equations are ordinary differential equations. The state equations of fluid dynamics are partial differential equations. This can be viewed as an infinite number of ordinary differential equations, where there is one state variable for each point in the fluid. Thus there are an infinite number of state variables in this case.

Once the state equations of a scene have been generated (the middle box of figure 1), general mathematical techniques can be used to convert them into a form in which numerical information about their state variables can be effectively determined. Examples include conversions of ordinary differential equations into finite difference formulas by Runge-Kutta methods, or the conversion of partial differential equations into systems of linear equations by finite element methods. We call the process of converting a system of equations into an effective computational form a *discretization*.

These computation structures can then be converted to actual programs (or codes) that numerically simulate the scene. If something is known about the architecture of the computer that will run the program then especially fast codes can be generated by symbolic elimination of variables, unrolling of loops or duplication of code. Each of these options may be appropriate because of cache sizes, vector processing structures or interprocessor communication costs. Other techniques of compiler theory are also appropriate and should be carefully considered at this point. More radical transformations like changing the order of the discretization or the discretization method itself may also be appropriate (for instance, if the cache size is too small). The process of converting state equations into computational structures and then into executable code is indicated in the right half of figure 1.

This paper discusses each of these steps in the simulation process. In section 2 we discuss one approach to representing scenes, their components and the underlying physics. Once the state equations have been generated, they can be directly solved numerically, yielding the trajectory of the scene from a given set of initial conditions. The approach we are pursuing is discussed in some detail in section 4.

However, occasionally some property of the trajectory is of interest—not the trajectory itself. We argue that by using symbolic techniques, the state equations of the system can often be transformed into other equations whose solutions more precisely answer the questions being asked. Solving these transformed equations is often substantially easier than solving the original system. However, substantial (non-numeric computation) is required to produce the transformed equations. In section 3 we illustrate how averaging techniques can be applied to reduce the dimension of the problem being solved and more directly answer certain questions. This technique is classical, but we feel is representative of the type of reduction that will be valuable in the future and is made possible by the general framework being proposed.

In the domain in which we are working (fluid dynamics), the state equations are partial differential equations. A wide variety of different methods are available for their numerical solution. Many of these methods can be subsumed within the general mathematical framework of *weighted residual methods*. Because we have access to the state equations in symbolic form, we can directly apply the weighted residual methodology to the differential equations of the problem to produce a computational structure based on a wide variety of different techniques including finite element, spectral and collocation methods. This approach is discussed in section 4. In section 4.1 we describe the general principles behind the weighted residual method. In section 4.2 we use the weighted residual method to produce a spectral method computational structure for a problem in fluid dynamics. This particular example illustrates the complexity of the codes generated in the study of turbulent fluid dynamics.

In section 4.3 we give another illustration of the weighted residual method in fluid dynamics, but this time the result of the discretization process is not a system of linear equations, but rather a system of ordinary differential equations. This is another example of where symbolic techniques can be used to convert a numerical problem into a simpler one that more directly provides the desired answers.

2. Scenes and Laws of Physics

This section makes more precise what we mean by scenes and physical laws. Section 2.1 discusses scenes while section 2.2 discusses the components of a physics and some of their functions.

2.1. Scenes

When describing a physical system that is to be simulated, we distinguish the observable properties and characteristics of the system from those properties and characteristics that are required by a particular physical model. The former are components of the *observable scene*, while the later belong to the physical laws and models that are to be

applied to the scene. For instance, the charge, mass and position of an electron are components of the scene, but the electric field is a characteristic of a physical model that might be used to determine the effect of the electron on other charged particles. Fields in physics are not themselves directly observable. only through their effect on other objects can their existence can be ascertained. Determining the effect of one object on another is the purpose of a physics. Fields are artifices used to facilitate the physics itself. (Recall that general relativity replaces a gravitational field by bending of the fabric of space itself. For small masses these disparate mechanisms give the same predications.)

Similarly, the "ether" of nineteenth century physics belongs to a set of physical laws, and is not intrinsic to the scene. Ether is posited by nineteenth century physics and is not, itself, observable. A more modern example is the wave function of quantum mechanics. It cannot be observed in the scene but is essential for a particular set of physical laws. In all of these cases the physics used to analyze the system imposes additional parameters (e.g. wave functions) or objects (e.g. fields or ether) as an aid in specifying the physics itself. These new quantities are part of the interpretation scene that is generated by the physics (see section 2.2).

A scene consists of a number of *objects* (rods, resistors, fluids, etc.) and *connections* (hinges, electrical nodes, etc.) between them. The connections constrain the behavior of two or more objects in some fashion. For instance, a hinge between two rods requires that the rods remain connected, while an electrical node connecting the pins of two resistors ensures that the two pins always have the same potential.

In addition, the objects possess a number of "observable" properties, e.g. the position and momentum of a particle. These properties are those aspects of the state of the object that may be observed in the scene, and thus are independent of the physics used to model the behavior of the scene. The observable properties may be redundant and/or related by some equations. For instance, the observable properties of a particle include the particle's mass (m), position ($\mathbf{r}$), velocity ($\mathbf{v}$), momentum ($\mathbf{p}$) and kinetic energy (T), where

$$\mathbf{v} = \frac{d\mathbf{r}}{dt},$$

$$\mathbf{p} = m\mathbf{v},$$

$$T = \frac{m\mathbf{v} \cdot \mathbf{v}}{2} = \frac{\mathbf{p} \cdot \mathbf{p}}{2m}.$$

For some models of physics, like Newtonian mechanics, the observable parameters are the state variables used by the physics. That is, the observable position and momentum are actually the position and momentum of the object in the physics. In other models, e.g. quantum mechanics, the observables are derived from the their correspondents. That is, the quantum mechanical position and momentum of a particle are not interchangeable with the observable position and momentum of the particle.

2.2. Physics

The properties of an observable scene are not necessarily appropriate for simulation. Instead, the physical laws translate the scene into one where the new scene's objects are described using state variables. For instance, a two-dimensional scene that consists

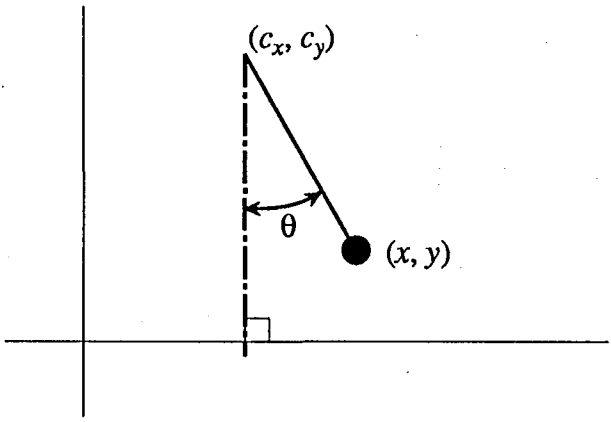

Figure 2. Simple pendulum.

of a heavy bob at the end of a rigid, massless rod whose other end is hinged (i.e. a two-dimensional pendulum) might have constitutive parameters of the length of the rod (ℓ) and the mass of the bob (m)—see figure 2. The observable parameters in the scene might be the position of the bob ($(x, y) \in \mathbf{R}^2$). However, when formulating a simulation, one would probably use the deflection of the rod from vertical ($\theta \in [-\pi, \pi)$) as the state variable of the system. The position of the bob can be derived from θ by

$$(x, y) = (c_x + \ell \sin\theta, c_y + \ell \cos\theta).$$

Each set of physical laws acts similarly. It must construct from an observable scene an *interpretation scene* that consists of objects, state variables that are appropriate to the physical model and the manifold structure on which the state variables lie. A correspondence also needs to be provided between state variables in the interpretation scene and quantities in the observable scene. The combination of the state variables, their manifold, and the correspondence we call an *interpretation scene* or just an *interpretation*. Examples of interpretations are the generalized coordinates of Hamiltonian mechanics (which were used in the pendulum example) and the wave functions of quantum mechanics.

The physical laws that apply to a scene are kept separate from the scene and should be expressed independently of their application to a particular scene. There should be one (or more) descriptions of rigid body dynamics and one (or more) descriptions of electrodynamics. These descriptions include the "laws of physics" (e.g. $\mathbf{F} = m\mathbf{a}$ for rigid body dynamics, or Maxwell's equations), specifications of when the particular laws are applicable and procedural specifications of how to apply the laws to a particular scene.

We call a set of physical laws a *physics*. Each physics has a limited range of applicability (until the Grand Unified Theory is discovered). Among the components of a physics is a specification of how forces and energies of objects in a scene can be computed. There are multiple physics's, some of which are compatible with each other over certain ranges of state variables and some of which are incompatible. For instance, Newtonian mechanics and classical electrodynamics are compatible for small masses and slowly moving macroscopic particles. Electrodynamics merely introduces a new force, which is characterized by Coulomb's law. Quantum mechanics and general relativity seem incompatible.

A description of a physics thus consists of (i) the domains of validity of the physics, (ii) a means of generating an interpretation scene from an observable scene and (iii) how fields and energies are to be derived from the resulting interpretation scene. In order for two different physics to be combined there must be some commonality between the two physics. First and foremost, the interpretation scenes must be somewhat compatible. This is one of the reasons why incorporation of quantum mechanics is so hard. In those cases where this type of mixing can be done the effects of the different physics are usually related through the energies of the objects in the scene.

Though the physics provides complete information about the constraints between the system's state variables, it does not provide a means of generating the state equations. We call such a mechanism which does generate the state equations form the constraints provided by the physics a *formulation*. Examples of formulations are Lagrangian and Hamiltonian mechanics. Both formulations provide a means of combining the fields and energies produced by the physics into a set of differential equations. Furthermore, notice that these formulation techniques can be used with several different physics, e.g. Hamiltonian mechanics can be used to formulate both Newtonian and quantum mechanics once the potential and kinetic energies of the system are properly defined.

The approach we have described in this section ensures that different physical considerations are dealt with separately. For instance, one should be able to simulate an electric motor by applying both rigid body dynamics and electromagnetics to a scene that consists of the rotor and stator of the motor, with the appropriate constitutive properties. We believe that greater modularity will result from this approach, although it places a premium on the symbolic techniques.

3. Harmonic Balance

When setting up a system of differential equations that models some physical situation, it is often easier to generate the equations in terms of state variables that are different from the ones that the user is really interested in. For instance, for a mechanical system it may be easiest to generate equations in terms of cartesian coordinates while the interesting behavior might be best expressed in terms of radial coordinates, or angular momenta or even averaged angular momenta. Each of these conversions can be performed after the numerical solutions are generated. This is the only approach possible when the state equations of the system are treated as a black box. However, when a simulator generator is used, the state equations are known to the simulator *a priori* and symbolic techniques can be used to perform this conversion before the integration process makes generating an accurate solution easier.

To illustrate this approach we will use a more sophisticated type of coordinate change that also facilitates an averaging technique. Thus we will ultimately generate differential equations for the average values of the variables of interest.

A large variety of simple oscillatory type systems can be modeled by differential equations of the form

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= -x + \epsilon h(x, y). \end{aligned} \tag{3.1}$$

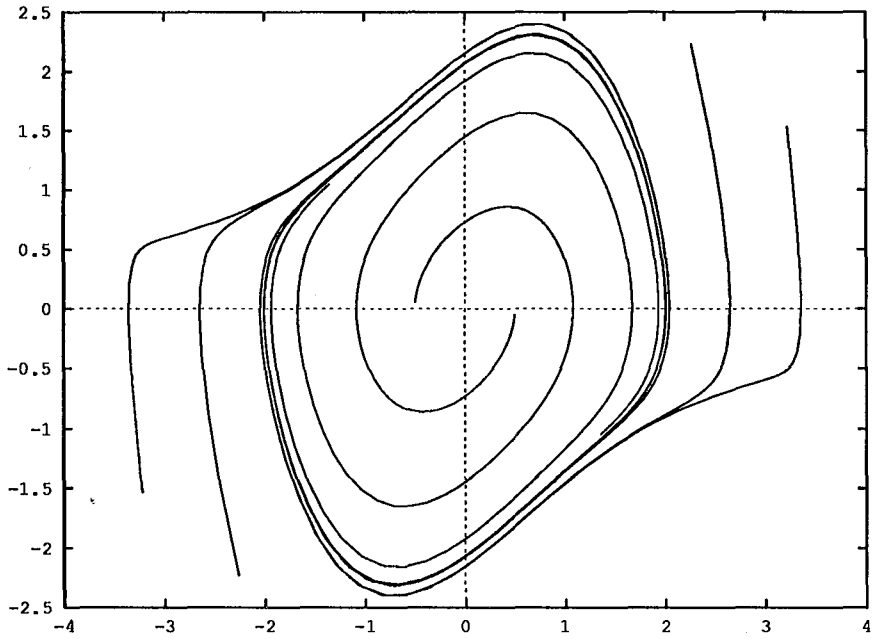

Figure 3. van der Pol oscillator.

For $h(x,y) = (1-x^2)y$ we have the van der Pol equation (van der Pol, 1926), for $h(x,y) = (1-y^2)y$ the Rayleigh equation (Rayleigh, 1883) etc. When $\epsilon = 0$, (3.1) reduces to a simple harmonic oscillator, whose solution is:

$$x(t) = r_0 \cos(t + \phi_0), \qquad y(t) = \dot{x}(x) = -r_0 \sin(t + \phi_0), \tag{3.2}$$

where r_0 and ϕ_0 are constants set by the initial conditions. In the x-y plane (the phase plane), the solutions are circles centered at the origin. The term $\epsilon h(x,y)$ of (3.1) acts as a perturbing non-linear damping factor on the solution to the harmonic oscillator. An example of the behavior of this damping factor can be seen from the van der Pol equation where $h(x,y) = (1-x^2)y$:

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= -x + \epsilon(1-x^2)y. \end{aligned} \tag{3.3}$$

The phase plot of the van der Pol equation, for $\epsilon = 0.6$ and various initial conditions, is shown in figure 3.

In the phase plane, (3.3) has a stable *limit cycle* of radius approximately 2. If the initial state of the system lies outside the limit cycle, the system will cycle inwards, asymptotically approaching the limit cycle. If the starting point is inside the limit cycle the system will oscillate outwards towards the limit cycle. From a physical point of view we might have two basic questions:

- What is the average amplitude of the limit cycle?
- How quickly does the system converge to the limit cycle?

We can study the behavior of the non-linear oscillator by assuming the solution is of the form (3.2) but allow the constants to be time varying functions, i.e.

$$x = r(t)\cos(t + \phi(t)),$$
$$y = r(t)\sin(t + \phi(t)).$$

Substituting these expressions into (3.1) gives the following system of equations

$$\dot{r}\cos(t+\phi) - \sin(t+\phi)(1+\dot{\phi}) = r\sin(t+\phi),$$
$$\dot{r}\sin(t+\phi) + \cos(t+\phi)(1+\dot{\phi}) = \epsilon h(r\cos(t+\phi), r\sin(t+\phi)).$$

When solved for $\dot{r}$ and $\dot{\phi}$, which must be done symbolically, we have

$$\dot{r} = \epsilon h(r\cos(t+\phi), r\sin(t+\phi))\sin(t+\phi)$$
$$\dot{\phi} = -\frac{\epsilon}{r} h(r\cos(t+\phi), r\sin(t+\phi))\cos(t+\phi).$$

The r component of the solution to this system of equations is the amplitude of the oscillator, which is closer to what we are looking for. In a physical system we probably don't care about the phase information, i.e. the ϕ component. We are more interested in the asymptotic behavior of the system. This can be obtained by *averaging* these equations over one oscillation, i.e. t to $t + 2\pi$:

$$\frac{d\langle r\rangle}{dt} = \frac{\epsilon}{2\pi}\int_0^{2\pi} h(r\cos(t+\phi), r\sin(t+\phi))\sin(t+\phi)\,dt,$$
$$\frac{d\langle \phi\rangle}{dt} = -\frac{\epsilon}{2\pi r}\int_0^{2\pi} h(r\cos(t+\phi), r\sin(t+\phi))\cos(t+\phi)\,dt.$$

In the r-ϕ coordinate system, the van der Pol equation becomes

$$\begin{aligned} \dot{r} &= \epsilon r(1 - r^2\cos^2(t+\phi))\sin^2(t+\phi) \\ \dot{\phi} &= \epsilon(1 - r^2\cos^2(t+\phi))\sin(t+\phi)\cos(t+\phi) \end{aligned} \tag{3.4}$$

When, averaged, the equation for $\dot{r}$ becomes

$$\frac{d\langle r\rangle}{dt} = \frac{\epsilon}{2\pi}\int_0^{2\pi}(r^2\cos^2(t+\phi) - 1)r\sin^2(t+\phi)\,dt = -\epsilon\left(\frac{\langle r\rangle^3}{8} - \frac{\langle r\rangle}{2}\right). \tag{3.5}$$

The solution of this equation is precisely the evolution of the "average" amplitude of the oscillation without any additional information. Notice that by averaging out the phase information we have been able to reduce the order of the equation by one. In figure 4 we have shown the evolution of a solution of (3.4) for two starting points, one inside and one outside the limit cycle, using a solid line. The dotted lines indicate solutions of the averaged equation (3.5) from the same two starting points. The averaged equation (3.5) has two advantages over (3.4). First, the averaged equation is of dimension 1 while the original system was of dimension 2, so the numerical integration will be computationally easier. Second, notice from figure 4 that the averaged solution is much smoother than than the original equation, so larger time steps can be taken in the numerical integration process, which again speeds up the numerical computation.

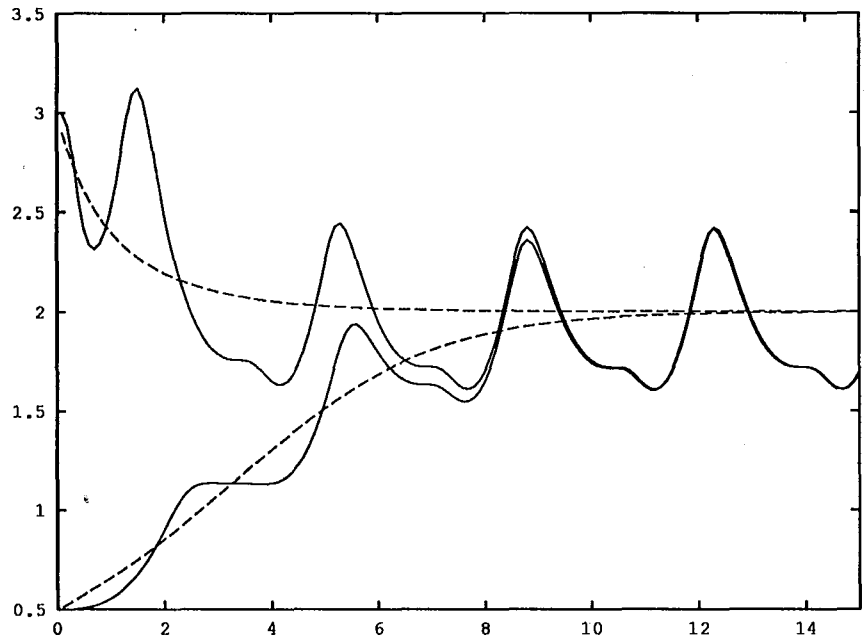

Figure 4. Amplitude of van der Pol oscillator: raw and averaged.

The two questions raised about the system, that of the amplitude of the limit cycle and of the rate of approach are easily answered from the averaged equation. On the stable limit cycle of the system, $\langle r \rangle$ will vanish, so by solving

$$0 = -\epsilon \left(\frac{\langle r \rangle^3}{8} - \frac{\langle r \rangle}{2} \right)$$

we see that the average radius of the limit cycle is 2, which is independent of the initial conditions. This can also be observed from figure 4.

The rate at which a solution approaches the limit cycle can be determined by solving (3.5):

$$r(t) = \left(\frac{4c_1 e^{\epsilon t}}{c_1 e^{\epsilon t} - 1} \right)^{1/2}.$$

This type of perturbation analysis has been used in celestial mechanics since the time of Laplace and Lagrange. The particular problem we consider here, the behavior of solutions of equations of the form (3.1), was discussed in some detail in Poincaré (1892). More recently Krylov and Bogoliubov (1947) have demonstrated the use of averaging techniques in a wide variety of problems. Rigorous results on the validity of averaging techniques are discussed in Sanders and Verhulst (1985).

4. Weighted Residual Methods

For a large number of physical simulation problems, the state equations are partial differential equations. Though the number of techniques for solving these systems can be bewildering, the most important techniques can be divided into two major classes: finite difference algorithms and weighted residual methods. We have decided to focus on weighted residual methods because most of the techniques of interest in our application area are of the weighted residual type.

There are a vast number of implementations of numerical algorithms based on particular weighted residual methods, most often for particular partial differential equations, but to our knowledge there have been no previous attempts to build a system that generates a numerical solver for a wide class of weighted residual methods.

We describe the basic principles behind the weighted residual method in section 4.1. In section 4.2 we give a brief illustration of how the weighted residual method is used to generate particular numerical codes in fluid mechanics. Finally, in section 4.3 we use the weighted residual method, along with a number of other ideas, to reduce some questions about the boundary layer of a fluid flow to questions about a system of ordinary differential equations.

4.1. General Approach

Let

$$Lu = f \tag{4.1}$$

be a partial differential equation, where L is a partial differential operator, f is a known function (often representing the boundary conditions) and u is a function of $\{x_1, \ldots, x_m\}$. The weighted residual method assumes there exists a (possibly infinite) set of trial functions $\{\phi_i\}$ such that, for some choice a_i,

$$\hat{u} = \sum_{0 \le i < N} a_i \phi_i \tag{4.2}$$

is an approximation to u, the solution of (4.1). The ϕ_i are functions of some subset of $\{x_1, \ldots, x_m\}$ while the a_j are functions of the remaining variables. Substituting (4.2) into (4.1) we have residual error

$$R_E(\hat{u}) = L\left(\sum_{0 \le i < N} a_i \phi_i\right) - f.$$

The goal of a weighted residual method is to choose the a_i in a fashion that minimizes $R_E(\hat{u})$ in some global sense.

A set of equations for the a_i can be deduced by choosing a set of weighting functions, w_j and requiring the inner product of $R_E(\hat{u})$ with the weights to vanish, i.e.

$$\int w_j R_E(\hat{u})\, dV = 0 \tag{4.3}$$

If the ϕ_i are functions of all of the variables $\{x_1, \ldots, x_m\}$, then the resulting equations are algebraic in the a_j. When L is a linear differential operator, the resulting equations are linear. Applying L to the components of the expansion and rewriting (4.3) we have

$$\sum_{0 \le i < N} a_i \int w_j \, L\phi_i \, dV = \int w_j f \, dV$$

For certain operators L and known ϕ_i and w_j the integrals above can be tabulated. Thus the bulk of the symbolic computation inherent in the reduction of (4.3) to systems of equations in the a_i can be performed *a priori*. However, if the ϕ_i and w_j are supplied by the user and, especially if L is non-linear, then symbolic computation is unavoidable in the application of the weighted residual method. This typically limits the applicability of the weighted residual method, but see Wang *et al.* (1984) and Wang (1988) for examples of how this can be automated for the finite element method.

A wide variety of different integration schemes fall into this general framework. If the w_j are chosen to be the same as the ϕ_i we get a Galerkin projection. This is especially convenient if the ϕ_i are orthogonal and eigenfunctions of L. The system of linear equations are then diagonal. The resulting technique is called a spectral method. The most common spectral method chooses $\phi_k = e^{i\mathbf{k}\cdot\mathbf{x}}$.

The finite element method discretizes the computational domain Ω into a number of elements, $\Omega_1, \ldots, \Omega_N$. It then chooses the ϕ_i to be continuous functions that are zero everywhere except within Ω_i. A Galerkin projection then gives the equations for the a_i.

In general, determining the linear equations or ordinary differential equations that need to be solved from (4.3) is a rather painful process that must be performed by hand. By taking advantage of methods from symbolic computation we can largely automate this process.

When the ϕ_i involve a subset of $\{x_1, \ldots, x_m\}$, (4.3) is a system of ordinary differential equations. Often the ϕ_i are functions only of the spatial variables and the a_j are functions of time. This is the situation in the two examples considered here. In section 4.2 the partial differential equation is first discretized in time and then the weighted residual method is applied, producing a system of linear equations that need to be solved. In section 4.3, the weighted residual method is applied directly to the spatial variables resulting in a system of ordinary differential equations for the a_i.

4.2. Numerical Example

In this section we illustrate how the weighted residual method is used to produce a numerical code for a problem that arises in the study of turbulent channel flow. This example illustrates the complexities that arise in practical applications of the weighted residual method. A simplification of one of the equations that arose in the study of turbulence in a channel flows by Kim *et al.* (1987) is

$$\frac{\partial g(x,y,t)}{\partial t} = h(g) + \frac{1}{\mathrm{Re}} \nabla^2 g, \tag{4.4}$$

where x and y are the spatial dimensions of the problem (only two are used in this illustration, although three are used in the real problem). h is a known non-linear function

$$\frac{x^{n+1}-x^n}{\Delta t} = \begin{cases} f(x^n) & \text{Explicit Euler} \\ f(x^{n+1}) & \text{Implicit Euler} \\ \dfrac{f(x^{n+1})+f(x^n)}{2} & \text{Crank-Nicolson} \\ \dfrac{1}{2}\left[3f(x^n)-f(x^{n-1})\right] & 2^{nd} \text{ order Adams-Bashforth} \\ \dfrac{1}{12}\left[23f(x^n)-16f(x^{n-1})+5f(x^{n-2})\right] & 3^{rd} \text{ order Adams-Bashforth} \end{cases}$$

Figure 5. Discretization techniques for $\dot{x}(t) = f(x)$.

of g and other functions that occur in the problem. In practice it can be a fairly complex expression and more than one partial differential equation be involved.

In solving this problem, discretization must occur in three different dimensions—time and the two spatial dimensions. Three different schemes will be used: An implicit finite difference scheme for time (t), a spectral method for the x dimension and a Galerkin type method using Chebyshev polynomials for the y dimension.

These three schemes are used in three successive steps. First, time is discretized and the value of $g(x,y,t)$ at the n^{th} time step is denoted by $g^n(x,y) = g(x,y,n\,\Delta t)$. Second, $g^n(x,y)$ is discretized in the x dimension using Fourier expansion with coefficients $\tilde{g}^n_k(y)$. Finally, $\tilde{g}^n_k(y)$ is discretized using Chebyshev polynomials in the y dimension with coefficient $\tilde{g}^n_{k,j}$. That is,

$$\begin{aligned} g^n(x,y) &= \sum_{0\le k<N_x} \tilde{g}^n_k(y) e^{2\pi i k x/L_x}, \\ &= \sum_{0\le k<N_x} \sum_{0\le j<N_y} \tilde{g}^n_{k,j} T_j(y) e^{2\pi i k x/L_x}. \end{aligned}$$

At this point the coefficients are numbers, and if done properly they are solutions of linear equations. Once these linear equations have been solved we can reconstruct $g(x,y,t)$ by summing the series.

Each of these transformations can be automated using symbolic techniques. In practice, their application is not completely straightforward. The following paragraphs illustrate this with some comments on the implementation of these techniques using symbolic computation.

The first step is to perform the time-wise discretization. We denote by $g^n = g^n(x,y)$ the function that corresponds to g at the n^{th} time step. The most straightforward discretization would be the explicit formula

$$\frac{g^{n+1}-g^n}{\Delta t} = h(g^n) + \frac{1}{\mathrm{Re}}\nabla^2 g^n$$

But this is known to be relatively unstable.

Figure 5 gives a number of different discretization techniques that can be used for ordinary differential equations. For this particular problem none of them is completely satisfactory. For instance, the explicit methods (Bashforth and Adams, 1883; Gear, 1971) are not sufficiently stable when applied to the entire equation. The implicit methods

require the solution of a non-linear equation at each time step (because of the nonlinearity of h) and are thus too costly.

The solution is to use an explicit scheme on the nonlinear terms and an implicit scheme on the linear terms. Using the second order Adams-Bashforth formula for the linear terms and the formula of Crank and Nicolson (1947) for the linear terms yields

$$\frac{g^{n+1}-g^n}{\Delta t} = \frac{1}{2}\left(3h(g^n) - h(g^{n-1})\right) + \frac{1}{2\mathrm{Re}}\left(\nabla^2 g^{n+1} + \nabla^2 g^n\right).$$

In a symbolic manipulation system this process is quite simple. The differential equation is first converted to a sum of terms form. Each term is then examined to see if it is linear in g. If so, an implicit formula is applied to each term, otherwise an explicit one. The results of these replacements are then added together and simplified.

The terms that involve g^{n+1} can be isolated on the left hand side to give

$$g^{n+1} - \frac{\Delta t}{2\mathrm{Re}}\nabla^2 g^{n+1} = \frac{\Delta t}{2}\left(3h(g^n) - h(g^{n-1})\right) + g^n + \frac{\Delta t}{2\mathrm{Re}}\nabla^2 g^n. \tag{4.5}$$

Again the symbolic processing involved is straightforward, each term is examined and placed on one side of the equation or the other based on its dependence on g^{n+1}.

At a given time step, each of the terms on the right hand side of (4.5) is known and can be computed directly. The next step is to compute the Fourier transform of this equation, eliminating the functional dependence on x.

$$g^n(x,y) = \sum_{0\le k<N_x} \tilde{g}_k^n(y) e^{2\pi i k x/L_x}.$$

Thus the k^{th} mode the Fourier transform of the left hand side of (4.5) is

$$\mathcal{F}_k\left\{\left(1 - \frac{\Delta t}{2\mathrm{Re}}\nabla^2\right) g^{n+1}\right\} = \tilde{g}_k^{n+1} + \frac{\Delta t}{2\mathrm{Re}}\left(4\pi^2 \tilde{g}_k^{n+1}\left(\frac{k}{L_x}\right)^2 - \frac{\partial^2 \tilde{g}_k^{n+1}}{\partial y^2}\right)$$

So for each k we need to solve the equation:

$$\begin{aligned}\tilde{g}_k^{n+1} + \frac{\Delta t}{2\mathrm{Re}}\left(4\pi^2\left(\frac{k}{L_x}\right)^2 - \frac{\partial^2}{\partial y^2}\right)\tilde{g}_k^{n+1} \\ = \mathcal{F}_k\left\{\frac{\Delta t}{2}\left(3h(g^n) - h(g^{n-1})\right) + g^n + \frac{\Delta t}{2\mathrm{Re}}\nabla^2 g^n\right\}.\end{aligned} \tag{4.6}$$

This equation is finally discretized in y by expanding $\tilde{g}_k^n(y)$ in terms of Chebyshev polynomials:

$$\tilde{g}_k^n(y) = \sum_{0\le j<N_y} \tilde{g}_{k,j}^n T_j(y),$$

where $\tilde{g}_{k,j}^n$ are numbers. The Chebyshev expansion of the left hand side of (4.6) is

$$\tilde{g}_k^{n+1} + \frac{\Delta t}{2\mathrm{Re}}\left(4\pi^2\left(\frac{k}{L_x}\right)^2 - \frac{\partial^2}{\partial y^2}\right)\tilde{g}_k^{n+1}$$

$$= \sum_{0\le j<N_y} \tilde{g}_{k,j}^{n+1} T_j(y) + \frac{\Delta t}{2\mathrm{Re}}\left(4\pi^2\left(\frac{k}{L_x}\right)^2 T_j(y)\tilde{g}_{k,j}^{n+1} + \frac{d^2T_j(y)}{dy^2}\tilde{g}_{k,j}^{n+1}\right)$$

$$= \sum_{0\le j<N_y}\left[\left(1 + \frac{2\pi^2\Delta t}{\mathrm{Re}}\left(\frac{k}{L_x}\right)^2\right)\tilde{g}_{k,j}^{n+1}\right] T_j(y) - \frac{d^2T_j(y)}{dy^2}\tilde{g}_{k,j}^{n+1}$$

The last term in this sum causes some problems because it is not expressed as a sum of Chebyshev polynomials, but as a sum of their derivatives. However, derivatives of Chebyshev polynomials can be expressed as a sum of Chebyshev polynomials by repeated application of the formula

$$\frac{T''_{n+2}(x)}{(n+1)(n+2)} - \frac{T''_n(x)}{(n^2-1)} + \frac{T''_{n-2}(x)}{(n-1)(n-2)} = 4T_n(x),$$

or by solving the tridiagonal system it implies. At this point, we have converted the problem of advancing time in (4.4) to solving systems of linear equations and computing Fourier and Chebyshev transforms.

For other basis and weight functions, and for other differential equations, very similar approaches are used. Simple symbolic methods (arithmetic operations and some simplification) are used to reduce the projection process to a sequence of integral. In the case discussed here, all of the integrals could be performed by table lookup. In the next section the integrals will have to be performed numerically.

4.3. Proper Orthogonal Decomposition

By discretizing the spatial dimensions but not the time dimension, we can reduce the Navier-Stokes equations to a system of ordinary differential equations. If the proper basis functions are chosen and sufficient terms are used the dynamical behavior of the ODEs should closely approximate that of the Navier-Stokes equations.

Lumley (1967) has suggested using this approach to study the behavior of the turbulent boundary layer of a fluid moving over a flat plate. Within the boundary layer bursts can be observed that are spatially and temporally somewhat periodic (Kline *et al.*, 1967). It would be interesting to know if these periodic phenomena manifest themselves in the ordinary differential equations where more powerful mathematical techniques can be used to analyze their behavior.

This reduction and detailed study of the resulting dynamical systems was originally performed by Aubry *et al.* (1988). As we shall see, the ordinary differential equations that result are extremely complex and are best generated by symbolic techniques.

Fluid flow is governed by the Navier-Stokes equations. In the absence of external forces

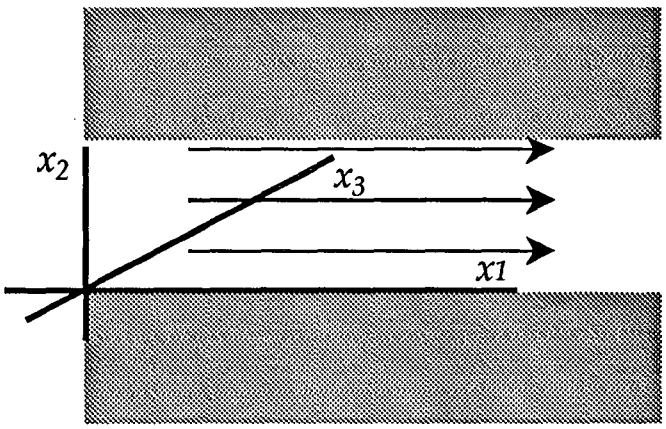

Figure 6. Coordinate system for a channel.

the dimensionless form of these equations is

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla\pi + \frac{1}{\mathrm{Re}}\nabla^2\mathbf{v}, \qquad \nabla \cdot \mathbf{v} = 0 \tag{4.7}$$

where $\mathbf{v}$ denotes the velocity field of the fluid and Re is the Reynolds number of the fluid. Flows with small Reynolds numbers tend to be rather steady, while flows with large Reynolds numbers (typically greater than 2300) are generally turbulent. In order to write the equations in a dimensionless form, characteristic lengths need to be defined in each of the three dimensions. We denote these different characteristic lengths by L_1, L_2 and L_3.

If $f(x_1, x_2, x_3)$ is function of position in the channel, we will denote by $\langle f \rangle$ its spatial average in a plane parallel to the walls of the channel:

$$\langle f(x_1, x_2, x_3) \rangle = \frac{1}{L_1 L_3} \int f(x_1, x_2, x_3)\, dx_1 dx_3,$$

where L_1 and L_3 are characteristic dimensions in the x_1 and x_3 directions respectively. Within this plane the turbulent flow is relatively homogeneous. Variations occur in the orthogonal direction. Thus $\langle f(x_1, x_2, x_3) \rangle$ is a function of only the distance from the wall, x_2.

The streak structure that we are interested in is not a function of the mean velocity of the fluid, only its fluctuations. Denoting

$$\langle \mathbf{v} \rangle = (U(x_2), 0, 0) = \mathbf{U},$$

we can determine $U(x_2)$ exactly:

$$U(x_2) = \mathrm{Re} \int_0^{x_2} \langle u_1 u_2 \rangle\, dx_2' + \mathrm{Re} u_T^2 \left(x_2 - \frac{x_2^2}{H} \right), \tag{4.8}$$

where u_T is the dimensionless wall shear velocity and H is the half height of the channel.

Applying this to (4.7) we get the *Reynolds averaged* Navier-Stokes equations:

$$\frac{du_i}{dt} + \mathrm{Re}\,\frac{\partial u_i}{\partial x_1}\left[\int_0^{x_0} \langle u_1 u_2\rangle\, dx_2 + u_T^2\left(x_2 - \frac{x_2^2}{H}\right)\right] + \mathrm{Re}\,u_2\,\delta_{i1}\left[\langle u_1 u_2\rangle + u_T^2\left(1 - \frac{2x_2}{H}\right)\right] + \sum_{1\le j\le 3} u_j\left(\frac{\partial u_i}{\partial x_j} - \left\langle u_j \frac{\partial u_i}{\partial x_j}\right\rangle\right) \tag{4.9}$$

$$= -p_{,i} + \frac{1}{\mathrm{Re}}\nabla^2 u_i.$$

These are the equations to which we will apply the weighted residual method.

Notice that, while the Navier-Stokes equations are quadratic, these equations for the velocity fluctuation are cubic due to the quadratic behavior of the mean velocity in (4.8).

4.3.1. Eigenfunction Projections

Because the flow is homogeneous in the plane parallel to the wall, we can use a Fourier expansion in the x_1 and x_3 directions (parallel to the wall). We assume we are given a set of basis functions in the inhomogeneous direction. These basis will not be known numerically, but rather will be provided numerically. These basis functions are called the "empirical eigenfunctions". Expanding the velocity fluctuation $\mathbf{u}$ just along x_1 and x_3 dimensions we have

$$\mathbf{u}(x_1, x_2, x_3, t) = \frac{1}{\sqrt{L_1 L_3}} \sum_{\substack{k_1=-\infty \\ k_3=-\infty}}^{\infty} \hat{\mathbf{u}}(x_2, t; k_1, k_3) e^{2\pi i\left(\frac{k_1}{L_1}x_1 + \frac{k_3}{L_3}x_3\right)}.$$

Each of the $\hat{\mathbf{u}}(x_2, t; k_1, k_3)$ can be expanded in series based on the empirical eigenfunctions:

$$\hat{\mathbf{u}}(x_2, t; k_1, k_3) = \sum_{n=1}^{\infty} a_{k_1 k_3}^{(n)}(t)\boldsymbol{\phi}_{k_1 k_3}^{(n)}(x_2).$$

Combining these two expansions gives the following representation of the velocity fluctuation field:

$$\mathbf{u}(x_1, x_2, x_3, t) = \frac{1}{\sqrt{L_1 L_3}} \sum_{n=1}^{\infty} \sum_{\substack{k_1=-\infty \\ k_3=-\infty}}^{\infty} a_{k_1 k_3}^{(n)}(t)\, e^{2\pi i\left(\frac{k_1}{L_1}x_1 + \frac{k_3}{L_3}x_3\right)} \boldsymbol{\phi}_{k_1 k_3}^{(n)}(x_2). \tag{4.10}$$

Notice that (4.10) is actually three equations, one for each component of $\mathbf{u}$. We denote the components of $\boldsymbol{\phi}_{k_1 k_3}^{(n)}$ by

$$\boldsymbol{\phi}_{k_1 k_3}^{(n)} = \left\langle \phi_{1_{k_1 k_3}}^{(n)}, \phi_{2_{k_1 k_3}}^{(n)}, \phi_{3_{k_1 k_3}}^{(n)} \right\rangle.$$

In addition we use the following identity, which can be computed by almost any symbolic system.

$$\int_0^{L_3}\int_0^{L_1} e^{2\pi i\left(\frac{p_1}{L_1}x_1 + \frac{p_3}{L_3}x_3\right)} dx_1 dx_3 = \begin{cases} L_1 L_3 & \text{if } p_1 = p_3 = 0. \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

Rather than compute the projection of the entire differential equation, we will illustrate the technique using the following term from (4.9),

$$\mathrm{Re}\, u_T^2 \frac{\partial u_i}{\partial x_1}\left(x_2 - \frac{x_2^2}{H}\right).$$

We can ignore the $\mathrm{Re}\, u_T^2$ term since it is a constant.

Our goal is to compute $f^{(n)}_{k_1k_2}$ such that

$$\frac{\partial u_i}{\partial x_1}\left(x_2 - \frac{x_2^2}{H}\right) = \frac{1}{\sqrt{L_1 L_3}} \sum_{n=1}^{\infty} \sum_{\substack{k_1=-\infty \\ k_3=-\infty}}^{\infty} f^{(n)}_{k_1k_3}\, e^{2\pi i(\frac{k_1}{L_1}x_1 + \frac{k_3}{L_3}x_3)} \phi^{(n)}_{k_1k_3}(x_2). \tag{4.12}$$

The $f^{(n)}_{k_1k_3}$ are functions of the $a^{(n)}_{k_1k_3}$. They are obtained by taking inner products (integrals) of (4.12) with the orthonormal basis functions. The first two inner products are Fourier transforms, that is

$$\tilde{f}_{k_1k_3} = \frac{1}{\sqrt{L_3}} \int_0^{L_3} \left(\frac{1}{\sqrt{L_1}} \int_0^{L_1} \frac{\partial u_i}{\partial x_1}\left(x_2 - \frac{x_2^2}{H}\right) e^{-2\pi k_1 x_1/L_1}\, dx_1 \right) e^{-2\pi k_3 x_3/L_3}\, dx_3.$$

The final inner product takes the form

$$f^{(n)}_{k_1k_3} = \int_0^{L_2} \tilde{f}_{k_1k_3} \phi^{(n)*}_{k_1k_3}\, dx_2.$$

The final term of U is a polynomial in x_2 and is easily dealt with. The Fourier transform gives

$$\mathcal{F}_{k_1k_2}\left\{ u_{i,1} u_T^2 \left(x_2 - \frac{x_2^2}{H}\right)\right\} = \left(\sum_{l=1}^{\infty} a^{(l)}_{k_1k_3} \frac{2\pi i k_1}{L_1} \phi^{(l)}_{i_{k_1k_3}} \right) u_T^2 \left(x_2 - \frac{x_2^2}{H}\right).$$

The Galerkin projection is just a simple integral, so

$$\mathcal{G}_\phi\left\{ u_{i,1} u_T^2 \left(x_2 - \frac{x_2^2}{H}\right)\right\} = u_T^2 \frac{2\pi i k_1}{L_1} \sum_{l=1}^{\infty} a^{(l)}_{k_1k_3} \int_0^{L_2} \left(x_2 - \frac{x_2^2}{H}\right) \phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}}\, dx_2.$$

The result of many similar symbolic computations is the system of ordinary differential equations shown in the appendix.

4.3.2. Numerical Computation

Having produced a symbolic system of ordinary differential equations like that shown in the appendix we must still compute each of the coefficients. This can itself be a rather complex undertaking without the proper abstractions. Consider, for instance, a piece of the sample term computed in the last section:

$$a^{(l)}_{k_1k_3} \int_0^{L_2} \left(x_2 - \frac{x_2^2}{H}\right) \phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}}\, dx_2.$$

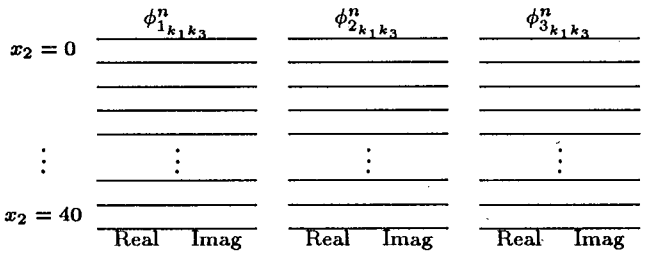

Figure 7. Memory structure of an empirical eigenvector.

The product of the two eigenvectors $\phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}}$ really means the dot product:

$$\phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}} = \phi^{(l)}_{1_{k_1k_3}} \phi^{(n)*}_{1_{k_1k_3}} + \phi^{(l)}_{2_{k_1k_3}} \phi^{(n)*}_{2_{k_1k_3}} + \phi^{(l)}_{3_{k_1k_3}} \phi^{(n)*}_{3_{k_1k_3}}.$$

Furthermore, each of the components of the eigenvectors are complex valued functions that are only known by their numerical values at selected points.

To understand the complexity of this computation, consider its implementation in a conventional programming language like C. The sampled functions that make up each eigenvector would be represented as arrays of the $N_{\mathtt{samp}}$ values of the sampled functions. Since complex numbers are not primitive in C, complex valued functions are represented by pairs of sampled functions. Each eigenvector would be a triple of complex valued sampled functions. This is illustrated in figure 7.

Using this representation, the expression $\phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}}$, which only involves an inner (dot) product, and the complex conjugation of the vector $\phi^{(n)}_{i_{k_1k_3}}$ would expand into the following pseudo-C code:

```
double Ans[2][Nsamp];

for (j = 0; j < Nsamp; j++) {
    for (k = 0; k < 3; k++) {
        Ans[0][j] = Ans[0][j] + φ^(ℓ)_{k1k3}[k][0][j] × φ^n_{k1k3}[k][0][j]
                              + φ^(ℓ)_{k1k3}[k][1][j] × φ^n_{k1k3}[k][1][j];
        Ans[1][j] = Ans[1][j] − φ^(ℓ)_{k1k3}[k][0][j] × φ^n_{k1k3}[k][1][j]
                              + φ^(ℓ)_{k1k3}[k][1][j] × φ^n_{k1k3}[k][0][j];
    }
}
```

The most straightforward approach is to expand the equations of the appendix into similar code. This is extremely time-consuming, and produces an enormous mass of code that is difficult to understand, hard to verify and almost impossible to modify if different turbulence models are introduced or if the original partial differential equations are modified.

The usual solution to this problem is to divide this computation into subroutines that deal with the different pieces of the differential equations, e.g. routines for arithmetic with complex valued sampled functions, arithmetic with vectors of complex valued sampled functions and integration of complex valued sampled functions.

We have taken a different approach. Over the past decade advanced computer algebra systems have been constructed using a "functorial" or "categorical" approach, as discussed in Jenks and Trager (1984). One of the tents of this approach is that mathematical objects like polynomials should be implemented as parameterized objects over the relevant ground structures (the coefficient domains). This approach is greatly simplified by the use of object-oriented programming languages like the Common Lisp Object System (Bobrow *et al.*, 1988), and C++ (Ellis and Stroustrup, 1990). Weyl is an example of these object-oriented languages that can be used in this manner. Zippel (1990) describes some of the issues that arise from this approach.

Among the functorial structures that are built using this approach are vectors, matrices, polynomials and series expansions. The base structures typically include integers, hardware-supported floating point numbers and arbitrary precision floating point numbers. To deal with this problem we have extended Weyl to include a new type of a functorial structure, which we call a "sampled function".

A sampled function is a function from an interval of $\mathbf{R}$ to a field K that is represented by its value at certain points in the interval. At other points the function's value is automatically interpolated (or extrapolated) from the values at which it is known. This structure is implemented functorially so the field K can be any field structure dealt with by Weyl including the real and complex numbers. Furthermore, the code required for the interpolation process and for performing arithmetic with sampled functions needs to be written only once.

Like all objects in Weyl, arithmetic with sampled functions can be performed using the usual Lisp operators, including conjugation. The eigenvectors $\phi^{(n)}_{k_1 k_3}$ are just (Weyl) vectors of sampled functions. Being vectors, the `dot-product` operator can be used to multiply them. The whole expression can thus be computed as shown below:

```
(* (var l k1 k3)
   (integral (* (make-sampled-function
                  (lambda (x) (- x (/ (* x x) H)))
                (dot-product (eigen l k1 k3)
                             (conjugate (eigen n k1 k3))))
             :lower-bound 0
             :upper-bound X2))
```

Notice that the Weyl code is a direct translation of the more mathematical form given above. With this approach, the code to generate the equations in the appendix requires only a couple of pages and is written in a manner that allows a fluid dynamicist to verify it.

However, we are still not quite finished. One of these ordinary differential equations, when using only a single eigenfunction, has the form

$$\begin{aligned}\dot{a}_1 = {} & 6.1a_1 + 2.1a_1a_2^* + 1.1a_2a_3^* + 0.4a_3a_4^* + 0.3a_4a_5^* \\ & - (3.0a_1a_1^* + 3.7a_2a_2^* + 2.4a_3a_3^* + 1.3a_4a_4^* + 0.6a_5a_5^*)a_1,\end{aligned} \tag{4.13}$$

where the coefficients are specified to only one decimal place for conciseness. The a_i are complex valued functions so, to numerically integrate the equations, each a_i must be converted to a pair of real valued functions. For (4.13) this gives the following pair of

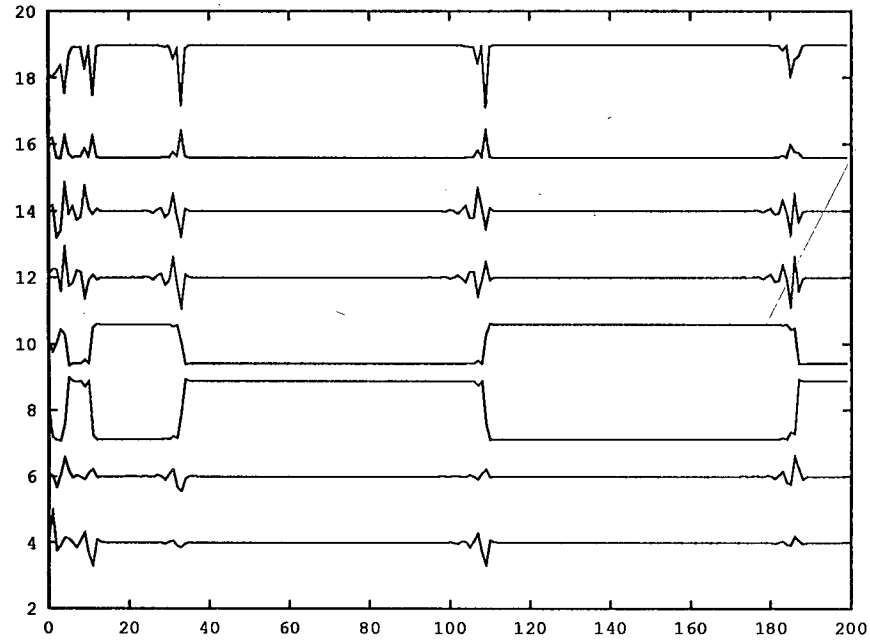

Figure 8. Typical amplitudes.

equations.

$$\begin{aligned}
\dot{x}_1 &= 6.1x_1 + 2.1(x_2x_1 + y_2y_1) + 1.1(x_3x_2 + y_3y_2) + 0.4(x_4x_3 + y_4y_3) \\
&\quad + 0.3(x_5x_4 + y_5y_4) - 3.0(x_1^2 + y_1^2)x_1 + 3.7(x_2^2 + y_2^2)x_1 \\
&\quad + 2.4(x_3^2 + y_3^2)x_1 + 1.3(x_4^2 + y_4^2)x_1 + 0.6(x_5^2 + y_5^2)x_1, \\
\dot{y}_1 &= 6.1y_1 - 2.1(x_2y_1 - y_2x_1) - 1.1(x_3y_2 - y_3x_2) - 0.4(x_4y_3 - y_4x_3) \\
&\quad - 0.3(x_5y_4 - y_5x_4) - 3.0(x_1^2 + y_1^2)y_1 + 3.7(x_2^2 + y_2^2)y_1 \\
&\quad + 2.4(x_3^2 + y_3^2)y_1 + 1.3(x_4^2 + y_4^2)y_1 + 0.6(x_5^2 + y_5^2)y_1.
\end{aligned}$$

Currently, these equations are integrated using the LSODE package (Hindmarsh, 1983; Petzold, 1983). A typical integration is shown in figure 8. There are periodic bursts of behavior, where the equations become very stiff. The Jacobians of the equations are currently generated symbolically to speed the calculation during the stiff regimes. Since the right-hand sides of these equations are polynomials, symbolic differentiation does not cause the expressions to grow and minimizes some of the benefits that accrue by using automatic differentiation (Kedem, 1980; Rall, 1981; Griewank, 1989).

The bursts of activity in figure 8, when converted into a velocity fluctuation, correspond to the periodic formation of the counter-rotating vortices. Thus the reduced system of ordinary differential equations has the same qualitative behavior as the far more complex Navier-Stokes equations. We are currently studying how to make this correlation more

quantitative and how the correlation with physical behavior depends upon the number of empirical eigenfunctions used.

One should note that for a modest number of empirical eigenfunctions, the size of the system of ordinary differential equations becomes very large and their formation and manipulation without symbolic techniques would be impractical.

5. Conclusions

In this paper we have advocated the construction of special purpose simulators for particular scenes, rather than building a general purpose simulator. Towards this end, we have discussed one possible approach to the construction of an environment which would enable the construction of such simulators. We have particularly focused on the use of symbolic techniques to transform differential equations into executable code.

We have outlined two major areas in which symbolic computation can be effectively used in numerical computations: (i) transforming differential equations into equations that more accurately address the questions being asked of the system under study, and (ii) the formation of the numerical integration code itself from libraries of technique fragments. Both of these techniques suggest different organizations of symbolic computation systems than we currently have available.

Acknowledgments

This work has greatly benefited from discussions with other members of the Modeling and Simulation group at Cornell, in particular Paul Chew, Jim Cremer, John Hopcroft, Anne Nierynck and Rick Palmer. The technique of proper orthogonal decomposition and the application of it to the determination of coherent structures was brought to my attention by Gal Berkooz, who is a student of John Lumley. Steve Rapkin has helped develop the code that actually generates differential equation code for the boundary layer problem. I am most grateful for their assistance.

References

N. Aubry, P. Holmes, J.L. Lumley and E. Stone (1988), "The dynamics of coherent structures in the wall region of a turbulent boundary layer", *J. Fluid Mechanics*, **192**, 115–173.

F. Bashforth and J.C. Adams (1883), *Theories of Capillary Action*, Cambridge University Press, Cambridge.

D.G. Bobrow, L.G. DeMichiel, R.P. Gabriel, K. Kahn, S.E. Keene, G. Kiczales, L. Masinter, D.A. Moon, M. Stefik and D.L. Weinreb (1988), *Common Lisp Object System Specification*, Technical Report 88–002, X3J13, ANSI Common Lisp Standardization Committee.

J. Crank and P. Nicolson (1947), "A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type", *Proc. Cambridge Philosophical Soc.*, **43**(1), 50–67.

M.A. Ellis and B. Stroustrup (1990), *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, MA.

G.W. Gear (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Series in Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ.

A. Griewank (1989), "On Automatic Differentiation", *Mathematical Programming: Recent Developments and Applications*, Kluwer Academic Publishers, Boston, MA, 83–108.

A.C. Hindmarsh (1983), "Odepack, a systematized collection of ODE solvers", *Scientific Computing*, R.S. Stepleman *et al.*, eds., North-Holland, Amsterdam, 55–64.

R.D. Jenks and B.M. Trager (1984), "11 keys to new Scratchpad", *EUROSAM '84*, Lecture Notes in Comput. Sci. 174, J.P. Fitch, ed., Springer-Verlag, Berlin-Heidelberg-NY, 123–147.

G. Kedem (1980), "Automatic differentiation of computer programs", *ACM Trans. Math. Software*, **6(2)**, 150–165.

J. Kim, P. Moin and R.T. Moser (1987), "Turbulence statistics in fully developed channel flow at low Reynolds number", *J. Fluid Dynamics*, **177**, 133–166.

S.J. Kline, W.C. Reynolds, F.A. Schraub and P.W. Rundstadler (1967), "The structure of turbulent boundary layers", *J. Fluid Mechanics*, **30**, 741–773.

N. Krylov and N. Bogoliubov (1947), *Introduction to Non-Linear Mechanics*, Annals of Math. Studies, **11**, Princeton University Press, Princeton, NJ.

J.L. Lumley (1967), "The structure of inhomogeneous turbulent flows", *Atmospheric Turbulence and Radio Wave Propagation*, A.M. Yaglom and V.I. Tatarski, eds., Akademi ianauk SSR, Moscow, 166–178.

L.R. Petzold (1983), "Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations", *SIAM J. Scientific and Statistical Comput.*, **4**, 135–148.

H. Poincaré (1892), *Les Méthodes Nouvelles de la Mécanique Céleste*, Gauthier-Villars, Paris.

L.B. Rall (1981), *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Comput. Sci. 120, Springer-Verlag, NY.

Rayleigh (1883), "On maintained vibrations", *Philosophical Magazine*, **15**, Series 5.

J.A. Sanders and F. Verhulst (1985), *Averaging Methods in Nonlinear Dynamical Systems*, Appl. Math. Sci., **59**, Springer-Verlag, NY.

B. van der Pol (1926), "On relaxation-oscillations", *The London, Edinburgh and Dublin Philosophical Magazine and J. Sci.*, **2**.

P.S. Wang, T.Y.P. Chang and K.A. van Hulzen (1984), "Code generation and optimization for finite element analysis", *EUROSAM '84*, Lecture Notes in Comput. Sci. 174, J.P. Fitch, ed., Springer-Verlag, NY, 237–247.

P.S. Wang (1988), "Integrating symbolic, numeric and graphics computing techniques", *Math. Aspects of Scientific Software*, J.R. Rice, ed., Springer-Verlag, NY, 197–208.

R.E. Zippel (1990), *The Weyl Computer Algebra Dubstrate*, Technical Report 90–1077, Dept. of Comput. Sci., Cornell University, Ithaca, NY.

Appendix. Final System of ODEs

$$\dot{a}^{(n)}_{k_1k_3} = \sum_{l=1}^{\infty} a^{(l)}_{k_1k_3} \left\{ \begin{array}{l} -u_\tau^2 \mathrm{Re} \dfrac{2\pi i k_1}{L_1} \displaystyle\int_0^{L_2} (x_2 - \frac{x_2^2}{H}) \phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}} \, dx_2 \\ \qquad -u_\tau^2 \mathrm{Re} \displaystyle\int_0^{L_2} \phi^{(l)}_{2_{k_1k_3}} \phi^{(n)*}_{1_{k_1k_3}} (1 - \frac{2x_2}{H}) dx_2 \\ + \dfrac{1+\alpha_1\nu_T}{\mathrm{Re}} \left[\left(\left(\dfrac{2\pi i k_1}{L_1}\right)^2 + \left(\dfrac{2\pi i k_3}{L_1}\right)^2 \right) \delta_{ln} \right. \\ \qquad \left. + \displaystyle\int_0^{L_2} \frac{d^2}{dx_2^2} \phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}} \, dx_2 \right] \end{array} \right\}$$

$$+ \frac{1}{\sqrt{L_1 L_3}} \left(1 - \delta_{\substack{k_1=0 \\ k_3=0}}\right) \sum_{\substack{k_1'=-\infty \\ k_3'=-\infty \\ m=1,q=1}}^{\infty} a^{(m)}_{k_1'k_3'} a^{(q)}_{k_1-k_1'k_3-k_3'} \times$$

$$\left[\int_0^{L_2} \phi^{(m)}_{l_{k_1'k_3'}} \Omega_l' \phi^{(q)}_{i_{k_1-k_1'\,k_3-k_3'}} \phi^{(n)*}_{i_{k_1k_3}} \, dx_2 \right.$$

$$\left. - \frac{2}{3} l_>^2 \alpha_2 \left(\begin{array}{l} \Omega_l \phi^{(m)}_{k_1'k_3'}(X_2) \Omega_l \phi^{(q)}_{k_1-k_1'\,k_3-k_3'}(X_2) \\ \quad + \Omega_l \phi^{(m)}_{k_1'k_3'}(X_2) \Omega_k \phi^{(q)}_{l_{k_1-k_1'\,k_3-k_3'}}(X_2) \end{array} \right) \phi^{(n)*}_{2_{k_1k_3}}(X_2) \right]$$

$$- \frac{\mathrm{Re}}{L_1 L_3} \sum_{\substack{k_1'=0, k_3'=0 \\ l=1, m=1, q=1}}^{\infty} a^{(l)}_{k_1k_3} \times$$

$$\int_0^{L_2} \left[\begin{array}{l} \dfrac{2\pi i k_1}{L_1} \phi^{(l)}_{i_{k_1k_3}} \phi^{(n)*}_{i_{k_1k_3}} \Lambda_{k_1'k_3'} \left(a^{(q)}_{k_1'k_3'} a^{(m)*}_{k_1'k_3'} \displaystyle\int_0^{x_2} \phi^{(q)}_{1_{k_1'k_3'}} \phi^{(m)*}_{2_{k_1'k_3'}} dx_2' \right) \\ \qquad + \phi^{(l)}_{2_{k_1k_3}} \phi^{(n)*}_{1_{k_1k_3}} \Lambda_{k_1'k_3'} \left(a^{(q)}_{k_1'k_3'} a^{(m)*}_{k_1'k_3'} \phi^{(q)}_{1_{k_1'k_3'}} \phi^{(m)*}_{2_{k_1'k_3'}} \right) \end{array} \right] dx_2$$

$$\Omega_l = \left\{ \begin{array}{ll} \frac{2\pi i k_1}{L_1} & \text{if } l = 1 \\ \dfrac{d}{dx_2} & \text{if } l = 2 \\ \frac{2\pi i k_3}{L_1} & \text{if } l = 3 \end{array} \right. \qquad \Omega_l' = \left\{ \begin{array}{ll} \frac{2\pi i (k_1 - k_1')}{L_1} & \text{if } l = 1 \\ \dfrac{d}{dx_2} & \text{if } l = 2 \\ \frac{2\pi i (k_3 - k_3')}{L_3} & \text{if } l = 3 \end{array} \right.$$

$$\Lambda_{k_1k_3}(f) = \left\{ \begin{array}{ll} f & \text{if } k_1 = k_3 = 0 \\ 2f & \text{if } k_1 = 0, k_3 > 0 \\ 2\Re(f) & \text{if } k_1 > 0, k_3 = 0 \\ 4\Re(f) & \text{if } k_1 > 0, k_3 > 0 \end{array} \right.$$