Numerical solution of differential equations through empirical eigenfunction expansions

by

Peter S. Wyckoff

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
Massachusetts Institute of Technology
September 1995

Author .....
Department of Chemical Engineering
August 11, 1995

Certified by .....
Gregory J. McRae
Joseph R. Mares Professor
Thesis Supervisor

Accepted by .....
Robert E. Cohen
Chairman, Committee on Graduate Students

Numerical solution of differential equations
through empirical eigenfunction expansions

by

Peter S. Wyckoff

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree
of Doctor of Philosophy at the
Massachusetts Institute of Technology

August 11, 1995

Abstract

This thesis considers some common chemical engineering problems in a broad framework and applies knowledge from other fields in simplifying their solutions. The general theme is to reduce the *total* time to solution which includes the entire path from initial model development and analysis through actual simulation and verification of the results.

The computational solution of the differential equations in models remains the bottleneck in solution time, especially for models which will be executed multiple times, such as in an optimization loop or process control algorithm. To remedy this situation, a system is presented which generates the optimal representation of a problem given information about its previous solutions and solves the model in the much smaller coefficient space of the chosen expansion instead of in the physical domain.

Thesis supervisor: Gregory J. McRae
Joseph R. Mares Professor of Chemical Engineering

Contents

II Optimal field representation

III Use of optimal bases in integration

1 Introduction

1.1 Thesis statement

The complexity of chemical engineering processes has been steadily increasing over the lifetime of the field due to increased scientific understanding of the underlying physics, more sophisticated demands from consumers of manufactured items, and expansion of the field itself into novel areas of operation. As a result of this continual advancement, the characteristics of basic research in chemical engineering are changing to continue to be able to provide the field with new ideas. These include delving into uncharted areas of work: using new substances, developing revolutionary operating techniques or managing old plants in new ways, and relying more on pre-construction design and modeling.

Chemical engineering has always been considered an interdisciplinary field, drawing on knowledge from chemistry, operations research, mechanical engineering, and others. Appealing to the more pure sciences is quickly becoming the vogue, especially considering the benefits that it often brings. In particular, there is a large literature of research on strictly theoretical mathematical problems which have no obvious physical applications but a knowledge of the foundations of this body of work may yield insights during analysis of an engineering problem.

This thesis considers some common chemical engineering problems under a more general framework in which knowledge from other fields may be readily employed. An overriding theme in the topics to follow is that of reducing the total time to solution. This is defined as the real time required to complete a problem, starting from problem inception through development of a model, validation and verification, execution of that model, plus design iterations which may reformulate the original goal itself, as illustrated in Figure 1. In fact the answer to a given problem may very well be that the question itself should not be asked. Traditional research too often falls into the habit of accepting as given the assumptions in the problem statement and making advancements in only the narrow regime defined by those constraints, when often there is much to gain by considering the larger picture.

Considering the vertical flow in Figure 1, initial considerations of a given problem should start at the highest possible analytic stage, and use any of a wide array of mathematical analysis techniques and modeling capabilities. This first means of attack may provide properties useful in later stages, such as symmetries or qualitative rules of thumb. Analysis of data produced by similar models or real physical processes may enter into the modeling stage as well. Next, since the lifetime of chemical

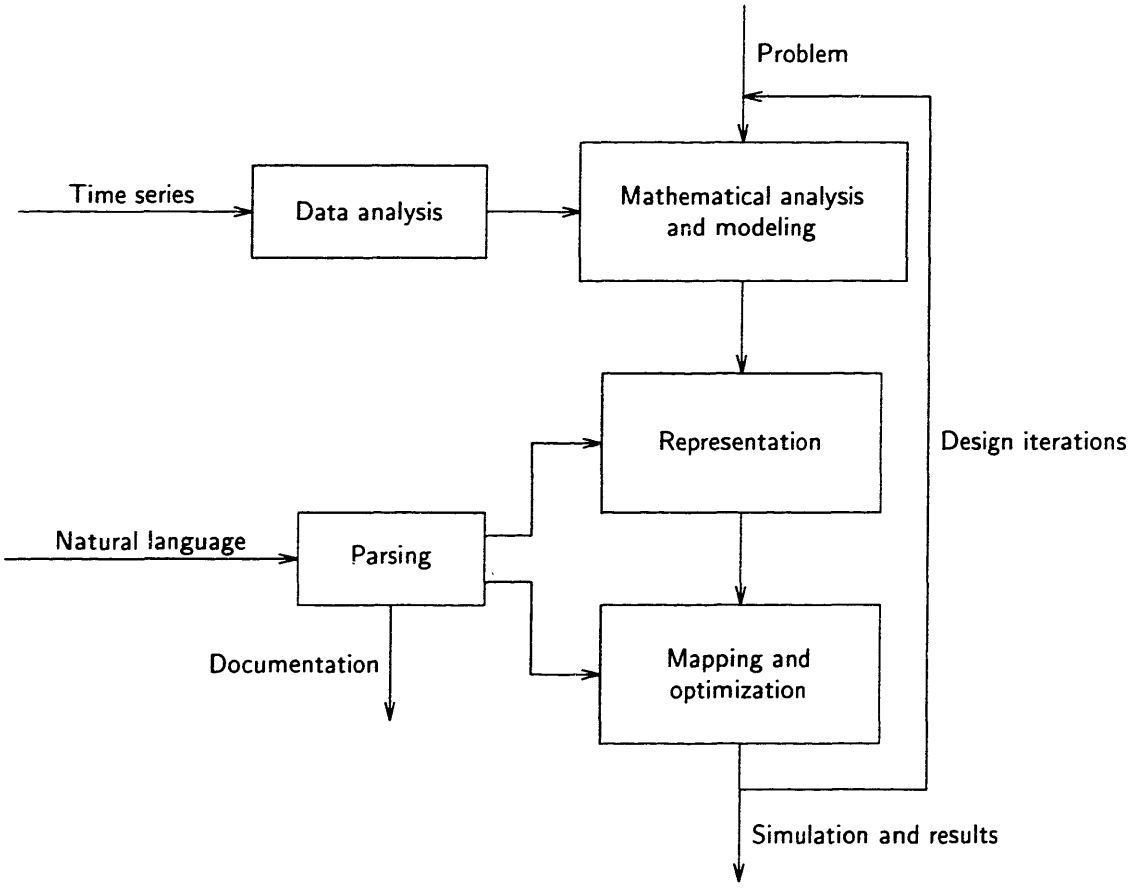


Figure 1. Conceptual illustration of the various facets involved in chemical engineering problem solving.

engineering models tends to be much longer than that of any particular piece of computational hardware, consideration must be given to choosing the best representation of a problem, independent of any particular architectural design. This representation will ideally use information learned from the first stage of analysis, and be optimal for the specific problem. The final stage is to produce actual results by mapping the chosen problem representation onto available hardware, optimizing its method of execution, and solving. The results can be used to guide further iterations of the design. The crucial concepts of parsing and documentation enter into both the representation and mapping stages as additional constraints: choose the language such that it is as natural as possible, and make the documentation of the process easy and complete.

1.2 Chemical engineering modeling

The models used in chemical engineering have enough common features to set them apart from generic modeling efforts. First, due to the all-encompassing nature of the field, a whole range of disparate physical processes often must be considered: convective transport, diffusion, chemical reaction, radiative heating, and many others. The implication of dealing with more than a few of these at once is that common simplifications cannot be applied to the phenomena independently, magnifying the complexities inherent in any single process.

A browse through a few recent issues of *AIChE Journal* shows how far this field has wandered away from standard BSL*-style analysis. The first set can be classified as advanced uses of BSL

* A classic text by Bird, Stewart, and Lightfoot [19].

and include residue curve maps in reactive distillation [139], mathematical solution techniques of diffusion-convection-reaction in multicomponent mixtures [57], and using optimization to synthesize nonequilibrium reactive distillation processes [32]. The second class is that of new techniques and contains two papers using molecular models to derive pure component properties and vapor-liquid equilibria [35, 78]. Next are new materials, namely supercritical water [144]. Finally is a paper which brings together the fields of chemistry, physics, and biology in explaining the leaching of ores [13]. These examples serve to illustrate the necessity of broadening one's background in doing research in chemical engineering, and suggest a more interdisciplinary future.

Chemical engineering models tend to be, in general, rather large. Furthermore the models are not written to be used only once, but instead to be placed inside larger structures, such as an optimization loop or model-predictive controller. Even without a formal optimization procedure, the models will be invoked many times during testing and validation as well as when new data are received. This extensive reuse of any particular model is what makes the use of an empirically derived basis space advantageous, as will be discussed in later chapters.

1.3 Motivation

Problem solving in chemical engineering involves a number of steps: proper identification of the problem, its description in the appropriate mathematical language, analyzing that description to ascertain characteristics which suggest or constrain the choice of solution method, generation of the computational solution method, and finally, an application. That final executable is not the end of the project—output from it will indicate problems or inaccuracies in the model which leads to further design iterations. Overall development time must be the metric for gauging the requirements of a problem solution, not simply the execution speed of the final model. Furthermore, if the model is to be at all useful to those other than its developer, documentation of the model plays a crucial role all the way from problem identification down to the source code itself. Brian Kernighan showed the sizes of the program `awk` and the accompanying documentation [73]: 3 000 lines for the program source code in about eight major sections, and 50 000 lines worth of manuals, change lists, and test problems. In light of this, a major emphasis in this thesis is the hierarchical organization of the various (and numerous) components which are necessary to implement a field-based integration. This organization includes, of course, many user guides and examples, and switches to produce either easily readable and alterable output, or output which is highly optimized for speed, attempting to offer the best of both worlds.

In spite of the above discussion about reducing the total time to solution, however, the bottleneck in the use of current models is frequently in the differential equation solvers. The emphasis lies on the word *use*, suggesting repeated invocations of the model after the initial development stage has been completed. This is the case since many models are used as the inner loop of some larger framework. One example is an atmospheric chemistry model which was used inside an optimization code to determine what the ground-level emissions should be to match observed pollutant concentrations. The model, which takes roughly 80 CPU minutes to solve on a single-processor Cray 90, is invoked for each search position chosen by the line search loop and by necessity must be fast for the optimization to be even feasible. Another example is a model of a chemical vapor deposition reactor including a complex set of chemistry, transport, and heating by both radiation and convection, which is used in a real-time controller to maintain chip temperature by adjusting heat lamp input. The time scale of the entire process is no more than two minutes, greatly constraining the execution time available for the model. In general complex chemical engineering models are often run repeatedly with rather similar parameters, such as initial conditions and reaction rate constants, and this consistent reuse allows for the construction of a solution method which takes into account the knowledge of these previous solutions in future calculations.

1.4 Objectives

The goal of this thesis at the outset was to bring about a radical change in the way differential equations are solved numerically by incorporating all the available knowledge of the problem. Models of interest are those which need to be simulated repeatedly, perhaps inside an optimization loop or model predictive controller. To do this requires three major steps: development of the theory behind the method, implementation of that theory, and examination of numerical results.

The theoretical issues involved begin at the choice of representation of the system. The way in which the equations are written down determine the structure of the problem and set up characteristics of its execution, hence great care must be taken in choosing a representation that is robust across the set of problems considered as well as optimal for any single problem. Numerical issues arise regarding the implementation of a chosen representation, including convergence, stability, and numerical round-off error tolerance, and must be considered in the theoretical stages such that support will be available to the implementation.

Implementation of large numerical solutions of differential equations is a complex process involving the participation of many modules and transformation utilities, and is inherently an iterative procedure. The general system design is worked out before coding begins, but unforseen complications are bound to arise. Overriding themes in the implementation of any section of the modeling process are

1. **Modularity.** Conceptually distinct components of the process must be kept apart, either through the use of multiple small executables or just placing groups of functions in separate source files.
2. **Reusablity.** Any part of the implementation, taken individually, must be self-contained and easily understandable. Furthermore, no section may be implemented in more than one place.
3. **Efficiency.** Although this requirement sometimes seems at odds with the first two, the whole goal is to *speed up* differential equation solution strategies. Crucial sections should be written automatically by an optimizing pre-compiler which handles the per-problem details, guided by more natural high-level language descriptions of the model. Input and output is all through the use of binary files, but utilities to convert between this and an ASCII representation will be available.

These themes also ensure that the results will be easily accessible and usable by others.

Finally, actual numerical results will be generated to verify the success of the method and suggest further avenues for improvement. Example implementations are the best sort of illustration of the method as well.

1.5 Thesis outline

This thesis is organized into three major parts, as designated by the three topics in the backbone of Figure 1. The first part discusses four related ideas of mathematical analysis and modeling. First, given a model and its boundary conditions, symmetries of the domain are shown to limit qualitatively the class of possible flows using topological analysis. The second topic considers numerical probing of a model to classify the various modes in parametric space, and using that classification to suggest improved operating strategies. Next a type of system identification is presented, where the model is generated directly from the data and is constrained by topological parameters of the template underlying the data. The final topic in the first part has two major sections dealing with the problem of choosing the optimal mapping strategy of a simulation onto underlying computational hardware.

The second part is concerned with how to choose the optimal representation for a problem, the second major step in a full problem solution strategy. Previous approaches to differential equation solving are considered, then the focus is narrowed to Galërkin methods in particular, which will be seen to encompass the majority of the commonly employed schemes. The success of any Galërkin method depends strongly on the appropriateness of the choice of test and trial functions used in

the expansion. The final chapter shows how to choose optimal functions given a history of previous solutions of the problem and its constraints.

The actual solution of a complex model always reduces to numerical computation, and accordingly, the final part of this thesis explains how to use the results of earlier sections in implementing integrations of differential equations. The major concerns here are conversion of the system equations into the optimal representation chosen for the problem and the ability to track errors dynamically during a simulation. Numerical results and timing information are presented for a variety of problems.

1.6 Acknowledgments

This work was directly supported by the Computational Science Graduate Fellowship Program of the Office of Scientific Computing in the U. S. Department of Energy. Additional funding was provided by the following: Cray Research, National Science Foundation, U. S. Environmental Protection Agency, Pittsburgh Supercomputing Center, and Digital Equipment Corporation.

Part I

Mathematical Analysis, Modeling, and Implementation

The time evolution of differential equations in general describes a wide class of physical situations, and techniques to follow this evolution are sufficiently mature that there is little novelty in such an analysis. Neither, however, is there much qualitative information to be gained. Other fields of mathematics provide alternative ways of studying the features of a problem and may prove beneficial before invoking the standard time evolution analysis.

The following four chapters present different aspects of mathematical analysis techniques used in modeling. The first is a topological study of a class of fluid flow problems, where the equations themselves are used to restrict qualitatively the behavior of the system. The second is a geometric approach to a bifurcating catalytic system, in which the model equations are probed numerically to find characteristic operating regimes depending on parameters in the model. Next, data from an existing system is used to discover the underlying model, and characterize it topologically, working backward from the usual modeling sequence. Finally, the actual implementation of a given model on high-performance distributed computers is considered, as eventually all but the most qualitative modeling efforts reduce to numerical computation.

2 Topology of a flow

Before embarking on any sort of numerical analysis of a problem, it is often worthwhile to study it from a more abstract perspective. There may be special symmetries to exploit, or the problem may have already been considered by others, only under a different name. Palis [108] says in his book on dynamical systems:

> *The qualitative study of a differential equation consists of a geometric description of its orbit space. Thus it is natural to ask when do two orbit spaces have the same description, the same qualitative features; this means establishing an equivalence relation between differential equations. An equivalence relation that captures the geometric structure of the orbits is what we shall define below as topological equivalence.*

This section describes such a study of a particular class of fluid flow.

Flow geometries in which the fluid flow may be characterized by closed streamlines, that is a constant enclosed volume of fluid, lead to a surprisingly wide variety of problems. In this section, only two-dimensional geometries of unit aspect ratio are considered for ease of presentation but this should not limit the general applicability of the ideas presented. In fact, more complex geometries would be an interesting test of the validity of the results obtained using a square domain.

2.1 Flow geometry

The flow geometry under consideration is presented in Figure 2, where a fluid is contained by two immobile walls on the sides and two movable ones on the top and bottom, all of equal length. V_1 and V_2 may take on any values, and be in the same or opposite directions. The moving walls are taken to be infinitely long such that the system may remain at steady state forever. The depth of the container (into the page) is considered zero for now, although it will be shown later what further solutions are allowed for a three-dimensional configuration. Also it is assumed that the fluid is Newtonian and that temperature effects are unimportant. For the case in which V_1 and V_2 are in opposing directions and of approximately equal magnitudes, one prediction is that the flow might resemble what is presented in the figure, with a stagnation point near the center.

For varying values of V_1 and V_2, one may imagine that other more complicated flow patterns are possible, like that shown in Figure 3. The small circles represent points of flow stagnation. In this case there are three points in the interior of the flow volume where zero fluid velocity will be found in addition to others on the stationary walls where shear stress along the wall changes sign due to the divergence of flow at those points. The four stagnation points in the corners are common

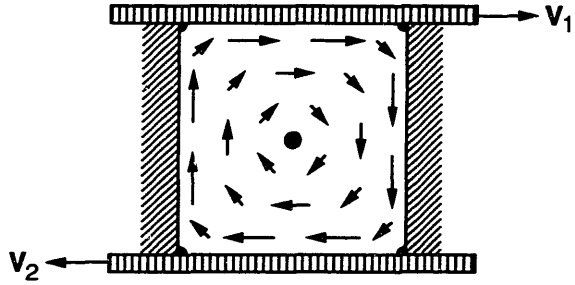


Figure 2. Container geometry with a likely flow.

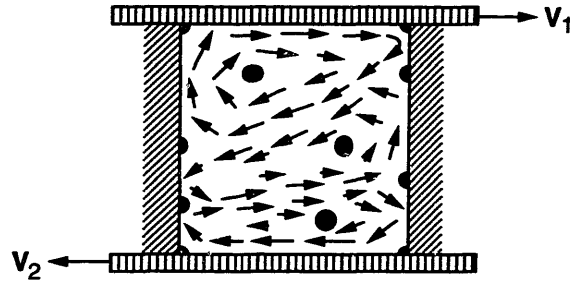


Figure 3. Another flow.

to all flows in this geometry due to the discontinuity of flow there required by the abrupt change of boundary conditions.

2.2 Equations of fluid motion

For an incompressible fluid with constant density and viscosity under the effect of no external forces other than the four walls, and Newtonian behavior, the following equations completely describe the flow:

$$\nabla \cdot \underset{\sim}{v} = 0 \qquad \text{(continuity)}$$

$$\rho \frac{D\underset{\sim}{v}}{Dt} = -\nabla p + \mu \nabla^2 \underset{\sim}{v} \qquad \text{(motion)}$$

with these boundary conditions:

$$\begin{aligned} &\text{at } x = 0, && v_y = 0 \\ &\text{at } x = L, && v_y = 0 \\ &\text{at } y = 0, && v_x = V_2 \\ &\text{at } y = L, && v_x = V_1 \end{aligned}$$

where L is the length of a side of the square and the origin of the coordinates is taken at the lower left corner of the box.

These equations are not solvable analytically although Weiss and Florsheim [140] present solutions for two special cases, using the assumption that vorticity is constant along streamlines neglecting convection of vorticity, $\underset{\sim}{v} \cdot \nabla \underset{\sim}{\omega}$. The resulting linear equation can be solved and predicts a single center as in Figure 2 for the case $V_2 = 0$ or two symmetrically located centers for the case that both plates are moving at the same rate in the same direction (Figure 11).

Due to the nonlinearity in the term on the left side of the equation of motion, a single solution is not guaranteed for the problem. Bozeman and Dalton [22] applied various numerical methods to the unapproximated problem and give some example results, copied in Figure 4. Pan and Acrivos [109] have shown numerically, using the same method as Burggraf [29], and experimentally that multiple solutions do indeed occur (Figure 5).

2.3 Mapping

A convenient method to study this problem involves considering the two-dimensional vector field of velocity at any point in the square as the tangent vector field of some two-dimensional manifold. In order to perform this transformation, it is convenient (and necessary) to replicate the flow geometry first. Let the original square be duplicated by placing its mirror image beside it and adjoining two stationary edges, as in Figure 6 where the moving plates have been designated by a single or double arrowhead and the immobile ones by a single slash each. (Here is one reason for restricting the

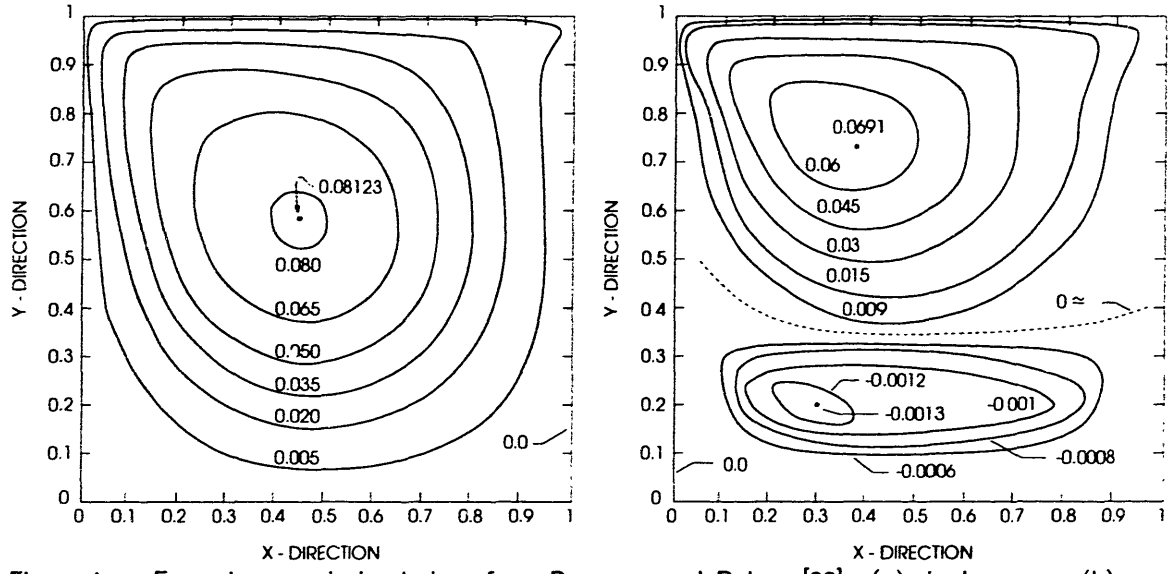


Figure 4. Example numerical solutions from Bozeman and Dalton [22]. (a) single center; (b) two centers with two half-hyperbolas on opposite walls.

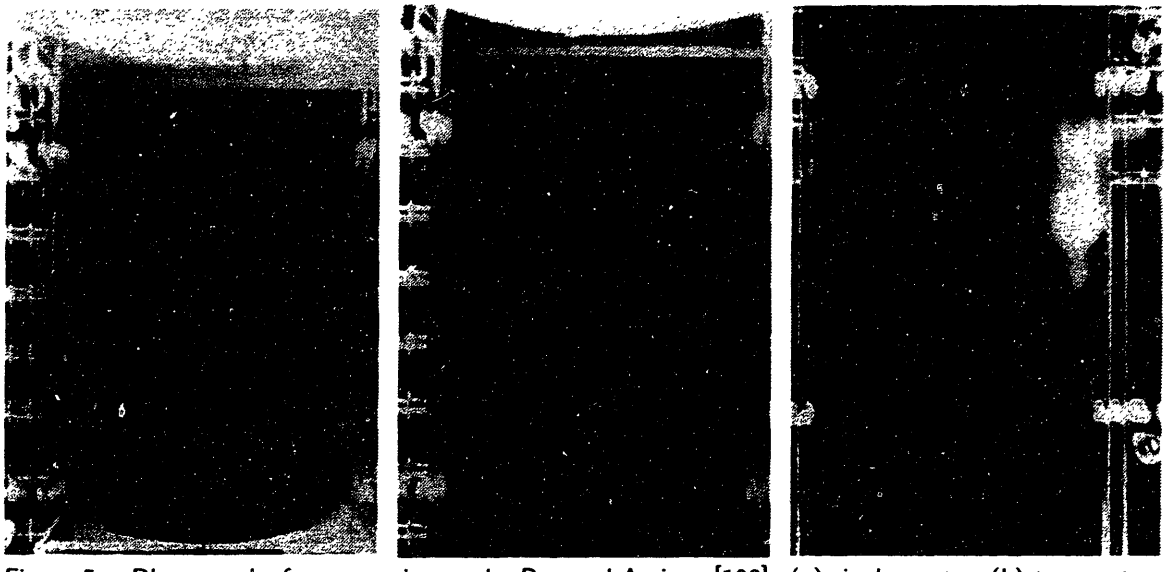


Figure 5. Photographs from experiments by Pan and Acrivos [109]. (a) single center; (b) two centers with two divergences on opposite walls; (c) development of a third vortex with two more divergences at high Reynolds number.

problem to two-dimensional flows: it is not intuitively obvious to the observer trapped in a three-dimensional world how to "fold" the cube out of its volume in order to construct a mirror image.) Repeating this procedure once more results in the final drawing in the figure and conceptually four identical systems have been oriented as shown by rotating them and reversing the signs of V_1 or V_2 where necessary to effect the flips. No changes have been made to whatever flow field may have been inside the containers originally.

From algebraic topology (see Kosniowski [79] in particular), two topological spaces X and Y are *homeomorphic* if there is a continuous invertible function f such that $f : X \to Y$ and $f^{-1} : Y \to X$. That is, if f is applied to every point in X, the space Y is obtained, and applying the continuous

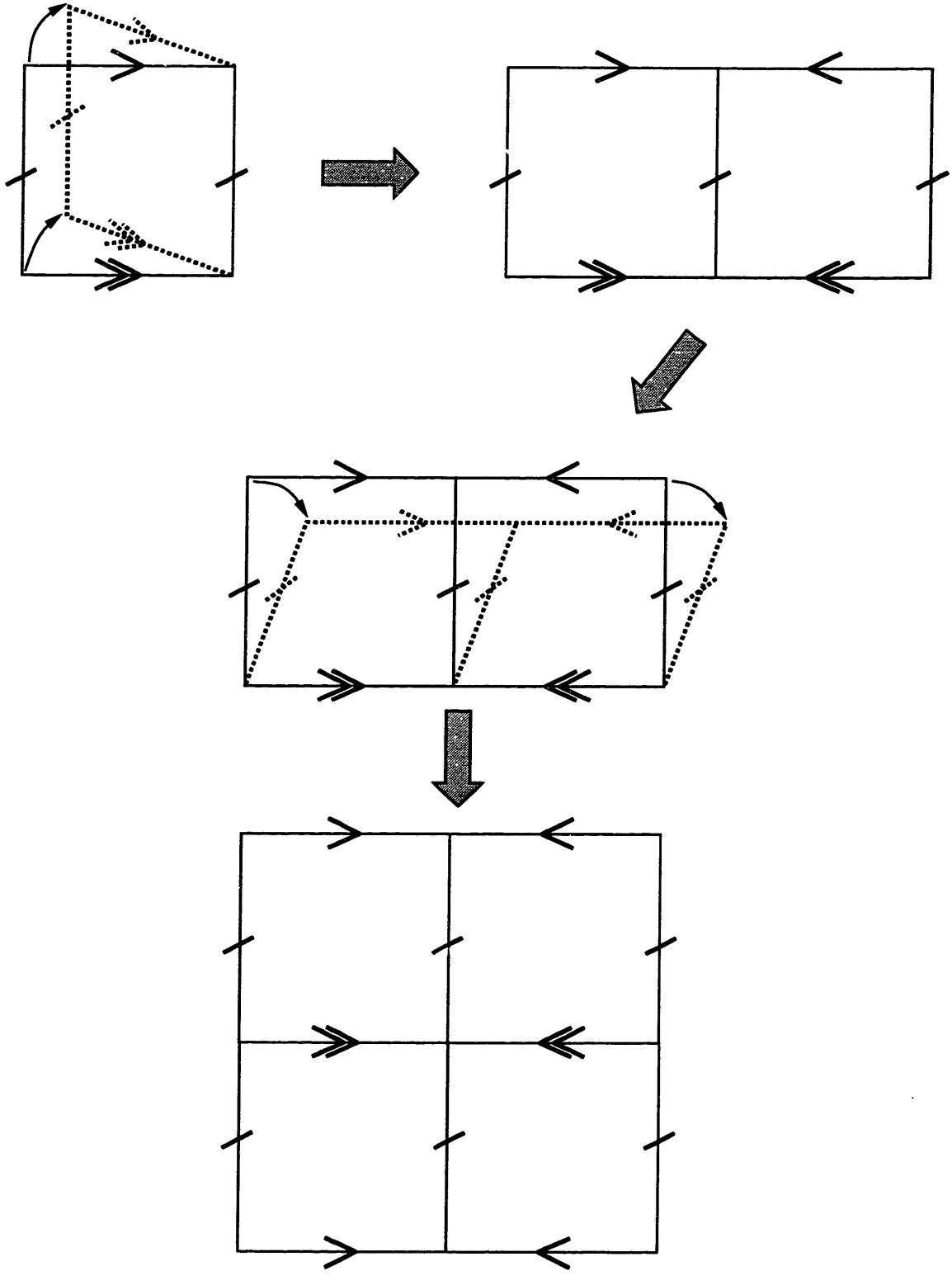


Figure 6. Sequence of flips.

inverse f^{-1} to Y returns the original X. As a simple example, a square $\{(x,y) \mid x = \pm 1, -1 \leq y \leq 1$ and $y = \pm 1, -1 \leq x \leq 1\}$ may be pulled into a rectangle of six times the area by the function $f(x,y) = (2x, 3y)$, and the rectangle may be pushed back into a square by $g(x,y) = (x/2, y/3)$ where $g \equiv f^{-1}$. Notice that "corners" may be added to apparently smooth spaces and vice versa, still fulfilling the requirements for homeomorphism. $f(x,y) = (x/m, y/m)$ where $m = \max(|x|, |y|)$ will flatten out the unit circle into the unit square while $g(x,y) = (x/r, y/r)$ where $r = \sqrt{x^2 + y^2}$ achieves the inverse.

Intuitively two shapes are homeomorphic if one can twist, bend, stretch, or shrink one into the other without joining sections or making cuts in the process. For example, the series of homeomorphisms depicted in Figure 7 fold the final square of Figure 6 into a torus. The first operation is just stretching the replicated systems out of the page to form almost a cylinder (where internal line segments have been omitted for clarity). The second operation involves the use of *topological identification* whereby the equivalence classes of a space are obtained by applying a certain equivalence relation. In this case, the equivalence relation is simply that both the left and right edges of the first drawing represent identical stationary edges so they may be joined together. Next comes a stretching of the cylinder such that the edges almost touch, then an identification of those edges which represent the same moving plate (V_1) with the two seams separating the opposite directions from each other. (The seams are again not shown but would be circles running the long way around the torus.)

To summarize the whole procedure, the square with two moving plates was copied into four adjoining squares then that larger square was stretched in such a way as to form a torus, sewing up the edges by identification of the equivalences of those edges. Now topological properties of the torus may be employed to give solutions of the original problem.

2.4 Euler-Poincaré characteristic

It can be shown that every compact manifold has a decomposition into smaller solids of the same dimension. (See O'Neill [104] for references to a proof.) The most useful decompositions employ regular shapes, as will be seen shortly. In particular, the torus is a two-dimensional compact manifold and can be constructed by gluing together enough sufficiently small rectangles in the appropriate way. Each of these rectangles has one face, four edges, and four vertices, by way of definition. The *Euler-Poincaré characteristic* of a manifold is defined as $v - e + f$ where v is the number of vertices, e is the number of edges, and f is the number of faces of some decomposition of the manifold. This characteristic is an example of a *topological invariant* which is a property preserved by homeomorphisms.

For example, a cube has 8 vertices, 12 edges, and 6 faces so its Euler-Poincaré characteristic $\chi(C) = 8 - 12 + 6 = 2$. A tetrahedron has $\chi(R) = 4 - 6 + 4 = 2$ also. Notice that "inflating" either shape results in a sphere, which must also have $\chi(S) = 2$ due to the nature of the "inflation" homeomorphism.

For the more conceptually difficult case at hand, the torus can be pictured as the revolution of a circle about a non-intersecting line in its plane then three distinct circles which are swept out can be chosen. Replacing them with triangles and connecting similar points forms a three-dimensional solid, effecting a decomposition of the torus into rectangles. This solid has 9 vertices, 18 edges, and 9 faces so a torus must have $\chi(T) = 9 - 18 + 9 = 0$.

2.5 Index of a singularity

A vector field on a manifold is defined as a function that assigns to each point of the manifold one of the tangent vectors at that point, where the tangent vectors are simply the set of tangents to all curves passing through the point. In mapping the flow problem from a flat square into the torus, the velocity vectors at each point in the square may be taken as a vector field on the torus. Stagnation

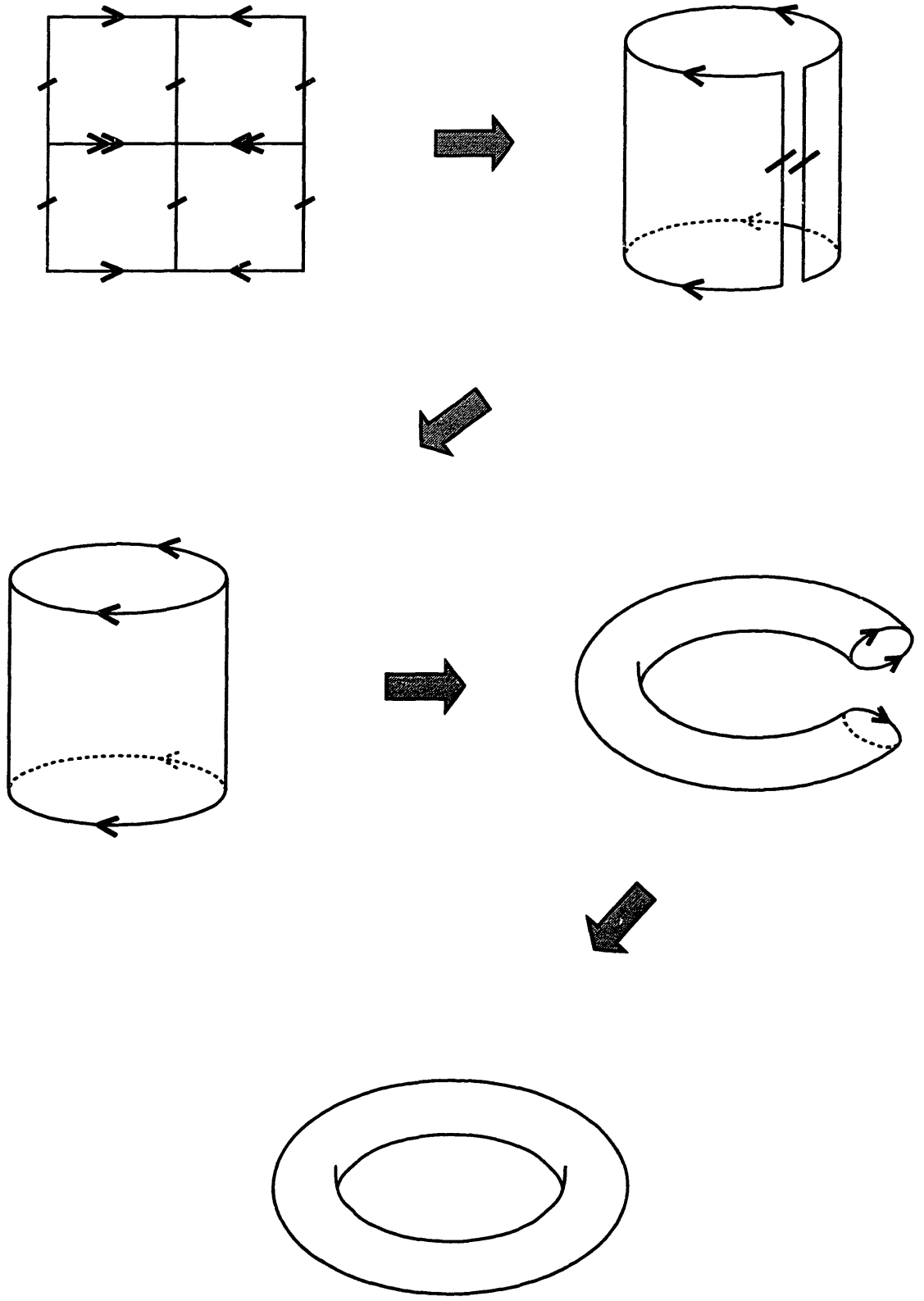


Figure 7. Twisting a square into a torus, with identifications.

points of the flow in the square do not assign a tangent vector to their corresponding points on the torus, and are called *singularities* of the vector field on the torus due to the indeterminate nature of the mapped point. Each of the small circles in Figures 2 and 3 becomes a singularity when mapped to a torus; in complex analysis they are called "zeros."

Poincaré has defined the *index of a singularity.* (Stoker [127] contains a reference.) Consider a simple closed curve which has at most one singularity in its interior and none on its boundary. The angle, which an arrow representing the vector field at each point on the curve, will continuously change as the curve is traced out. On making one circuit in the counterclockwise direction the angle changes by an amount $2\pi j$ where j is the index of the enclosed singularity. In accordance with the definition, the index of singularity about a region which is nonzero everywhere is zero. A "center" or "spiral" has index $+1$, regardless of the direction of rotation. A "source" or "sink" also has index $+1$, the name depending on whether the fluid is entering or leaving the point. A "hyperbola" has index -1. (See Figure 8.)

In fact, Milnor [91] states that the index of the vector field at a non-degenerate zero is ± 1 depending on the sign of the determinant of the matrix of derivatives evaluated at the zero. This gives another way to calculate the index. An example of a degenerate zero is the "clover" given in Figure 8.

2.6 Connection

Lefschetz [81] performs a complex derivation of the fact which will link together the concepts of index of a singularity and characteristic of a manifold. The proof is much too lengthy and dependent on a large background of information about fixed point theory in general to present here. It suffices simply to summarize that the sum of the indices of all the zeros of a vector field on a manifold is equal to the Euler-Poincaré characteristic, $\chi(M)$, of that manifold. Milnor [91] also asserts the same fact without proof.

This apparently simple statement is what will allow the determination of all possible solutions of the flow problem by just counting up all the different types of stagnation points.

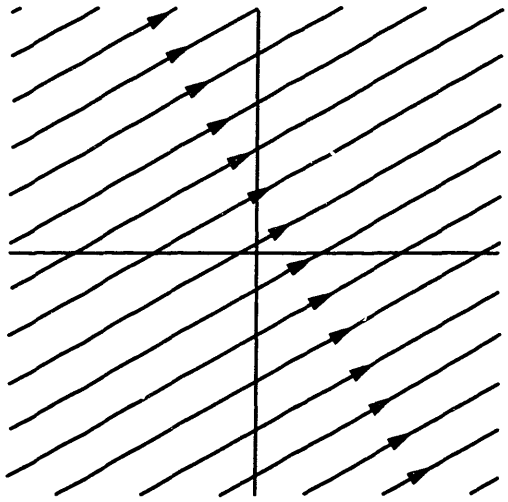
2.7 Acceptable flows

From the previous sections, there is a set of conditions which provide information about flow fields which satisfy the equations of motion. Considering the construction of streamlines on the four-fold system in Figure 6:

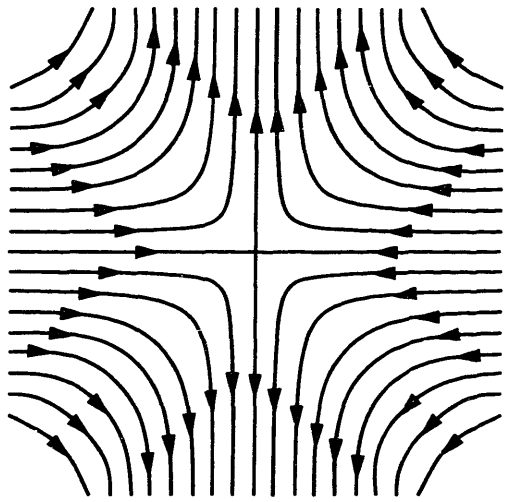
1. No fluid is permitted to flow across the perpendicular lines which represent edges of distinct systems.
2. Vector fields must be symmetric about both internal lines.
3. The sum of the indices of all stagnation points must be equal to zero.

The first and second conditions stem from the method in which copies and flips were performed to obtain the diagram. The third is the crucial result from algebraic topology which applies over the whole torus (represented by the four-square) and over the original system due to the induced symmetry. (If the problem had been mapped to a different manifold with non-zero characteristic, then it would be necessary to divide that value by four to obtain the sum of indices in an individual square.) Note that any solution to the equations **must** satisfy these conditions, but vector fields obtained from the conditions may not necessarily be physically realizable flows. Only the qualitative nature of attainable flows can be gathered from this method, not exact velocities.

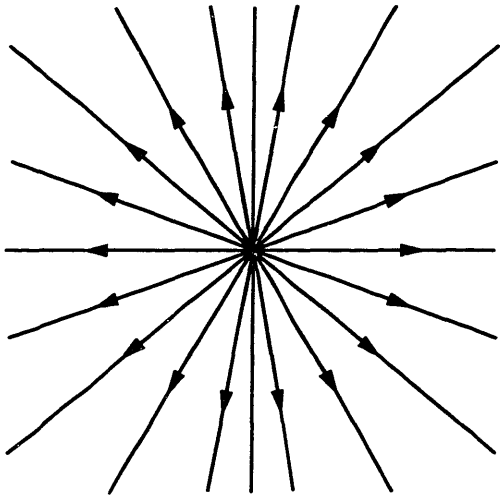
Using such a diagram is simply a convenience to ensure symmetry and aid in visualizing possible flows. Consider the example in Figure 9. First it must be recalled that due to the boundary conditions where the moving walls meet the stationary ones, divergent (hyperbolic) stagnation points must be placed in all four corners of each box. This "empty box" can not be a possible flow since it does not satisfy condition number three. Each divergent point counts for -1 in the summation, but



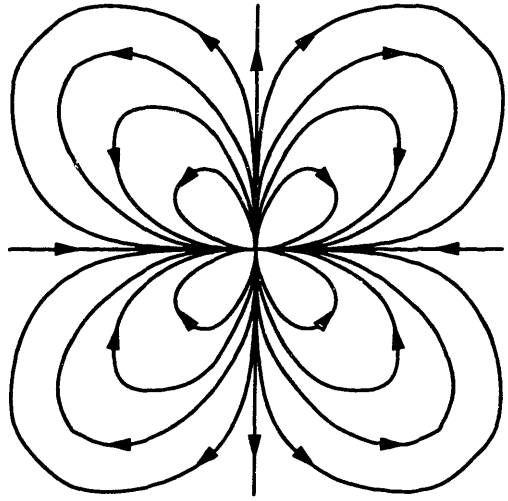
a) Constant vector field, $j = 0$.



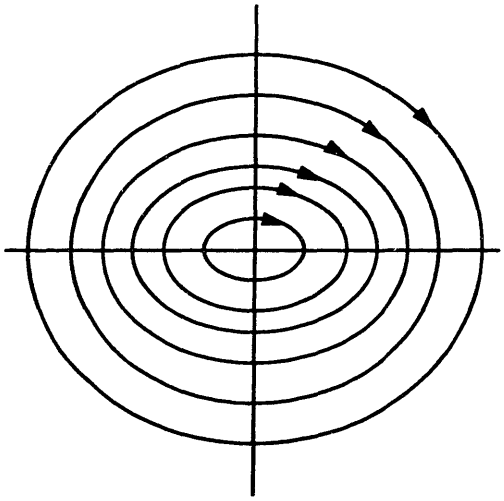
b) Hyperbolic zero (saddle point), $j = -1$.



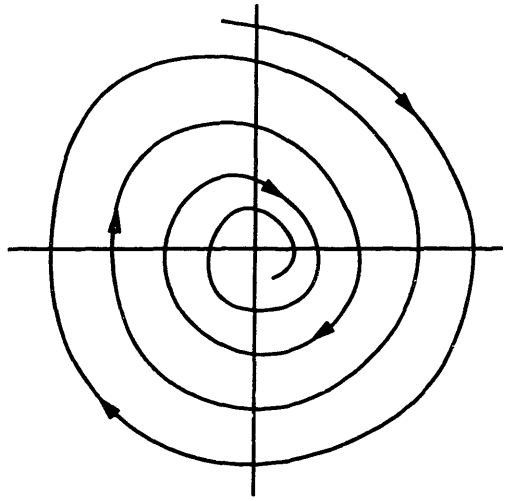
c) Source (or sink) point, $j = +1$.



d) Clover point, $j = +2$.



e) Center zero, $j = +1$.



f) Spiral zero, $j = +1$.

Figure 8. Examples of stagnation point singularities.

the ones on the edges contribute only $-\frac{1}{2}$ and the corners only $-\frac{1}{4}$. This can be seen by replicating the system under discussion four more times to make a sixteen-fold set of boxes. Picking any two adjoining four-boxes and computing their index sums and that of the total system will show on comparison that each half-circle must have the same index, and that two half-circles have an index of -1. This process may be continued to obtain the index number of the quarter circles as well. The characteristic number of one cell of interest is therefore $4 \times -\frac{1}{4} = -1 \neq 0$.

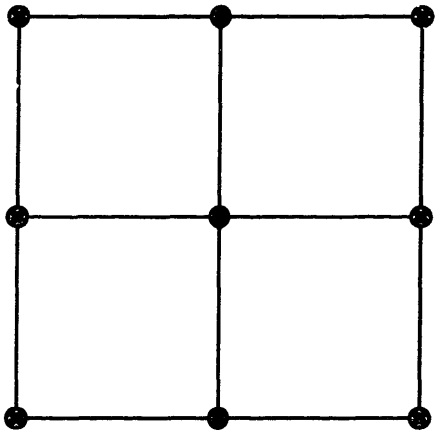


Figure 9. Building a flow, first an empty box.

In order to make this sum zero, it is necessary to add more stagnation points. One obvious choice is a single center of rotation, placed arbitrarily. Due to the symmetry condition, that center must appear in each of the other squares as well (Figure 10). Since a center has index $+1$, Figure 2 is an acceptable flow pattern, and has been observed both numerically (Figure 4a) and experimentally (Figure 5a). In fact, the location of the center depends on physical properties of the fluid, size of the box, and velocities of the plates.

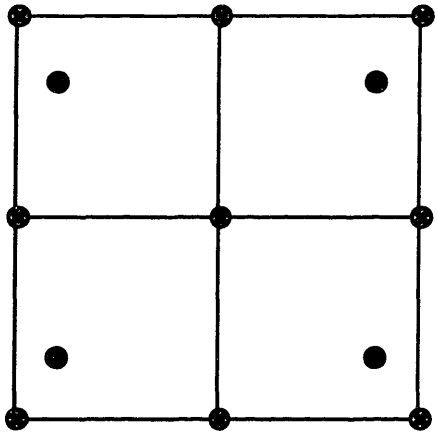


Figure 10. Adding one center zero symmetrically.

It is informative to see what happens when more singularities are added. The necessity of replicating across the internal edges is merely a formality which may be dropped as long as the original reasons for choosing such a geometry are kept in mind. First, another center, placed at random, requires another divergent point, or parts of one. The resulting flow as illustrated by arrows in the direction of motion (Figure 11) may not seem obvious working under the original illustrations with V_1 and V_2 in *opposite* directions, but has been observed when the plates move in

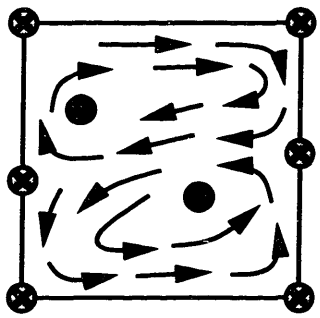


Figure 11. A two-center flow with plates moving in the same direction.

the same direction (Figures 4b and 5b). Adding more centers and hyperbolic zeros in pairs gives rise to further and more complicated flow patterns, justifying Figure 3 and corroborating with Figure 5c.

2.8 Extensions

It is plausible to believe that this method may be applied to three-dimensional geometries in much the same manner. Figure 12 shows a flow which fits the criteria above and has intuitively the correct shape, where the two centers have been replaced with a source point and a sink point. It can be imagined that the fluid is rotating in the same fashion in every plane parallel to the page, with the sources and sinks smoothly varying. A source of magnitude S in the topmost plane becomes a sink of the same magnitude in the bottom plane to conserve the total amount of fluid in the volume.

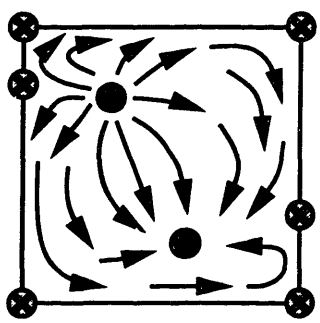


Figure 12. Two-dimensional slice of a possible three-dimensional flow.

No proof is offered to support this suggestion here but it would involve performing a similar type of mapping except that instead of taking a rectangle onto the torus, the new mapping would take the **cube** of volume onto some three-manifold embedded in four-space. Pan and Acrivos [109] support this speculation by observing that the "presence of three dimensional fluid motions ... did not extend into mid-section where to all appearances the flow was indeed two dimensional." They had been worried that their two-dimensional results may have been jeopardized by the non-zero depth of their experimental system and confined all measurements to a plane in the middle.

A spiral center has not been shown in any of the diagrams but it may be considered as just a variation on the source or sink, depending on whether the fluid converges to or diverges from the point, and drawn similarly as in Figure 12 except with rotating fluid elements in the vicinity of the spiral. Other types of zeros are also possible when allowed an extra dimension, except that calling them "stagnation points" is somewhat a misnomer now. Still given any plane containing such a zero, an analog to one of the traditional points may be seen.

2.9 Conclusion

The presentation just completed connects two fields which rarely have any common interest: algebraic topology and incompressible fluid flow, to obtain a description of the restrictions placed on the fluid flow system generated by the mathematical space in which the problem lies. Abstract analyses such as the one demonstrated here provide an extra level of knowledge about a problem beyond that normally given by consideration of the system in its own well-defined regime.

Another application of algebraic topology in determining the qualitative structure of a problem is presented by Doherty [39], who finds conditions under which the presence or absence of ternary azeotropes may be determined by knowing the number and stability type of any binary azeotropes and the pure components in a three-component residue curve map.

Unsteady catalysis

A typical chemical engineering activity is the modeling of complicated systems by a series of mathematical equations which may then be manipulated in this abstract form to gain information about the system at hand. Quite often these equations will be nonlinear, and require the application of various assumptions and approximations before they are able to be solved. Characteristics of the resulting simplified equations may be products of the reduction method itself instead of actual representations of the physical system. In particular, bifurcation theory provides a method of classifying behavior near singular points and will be used to describe a model of CO oxidation on single crystal platinum surfaces.

The literature of chemical engineering abounds with situations where nonlinear analysis has been used to better understand the basic structure of physical processes. For example, the Lorenz equations [83] are a highly simplified form describing atmospheric motion, and exhibit chaotic behavior when analyzed. These model results do not support reality in this case due to exclusion of stabilizing damping terms, and erroneously predict that the presence of a butterfly in Japan, for instance, may influence storm formation in the Atlantic Ocean. Another interesting case is the Belousov-Zhabotinskii [146] reaction sequence, named after those who first noted experimentally this oscillating liquid-phase system. A complete mechanistic examination of the species present and the reactions they might possibly undergo leads to fourteen ordinary differential equations which do **not** reproduce the observed behavior. With reduction of the system down to seven or less equations using knowledge about the rates of reaction [137, 121, 45], systems exhibiting the proper behavior can be constructed.

With the ever-increasing availability of computer-aided modeling tools, the use of nonlinear analysis in chemical engineering systems may be extended to the realm of process design. One example is the development of control strategies to utilize knowledge about the underlying structure of the system itself. A detailed investigation of a catalytic reactive process, including asymptotic stability analysis of the governing differential equations, reveals that an oscillatory control strategy will substantially improve performance.

3.1 Motivating example

The conversion of pollutants to form more acceptable by-products in industrial reactions is a popular pursuit, with the goal being to eliminate harmful species such as carbon monoxide and nitric oxide at the highest possible rate. Fick *et al.* [44] describe the reaction of NO and CO on a Pt(100) catalyst,

and observe damped oscillations for certain sets of operating parameters. They also noticed hysteresis depending on the direction of a temperature ramp applied to the system. To promote their studies to the regime of industrially feasible operations, it is necessary to find out why these phenomena are occurring and how to remove or avoid them. Another question which should be asked is, "Do we really want to avoid these oscillations?"

The experimenters' observations constitute the "real world" which can be coerced into a mathematical form by making suitable approximations. Typical of these are the continuum approximation for the gas phase, and the assumption of a homogeneous substrate, but it should be realized that our everyday philosophical tradition imposes even more structure on the mathematical forms.

A mathematical system of partial differential equations may be written down by considering all the possible effects on the system: concentration and temperature gradients in the catalyst and gas phases and between, and reactions between the participating species in all phases. Knowledge about the system suggests assumptions which permit reduction to coupled ordinary differential equations. For this case, everything except three surface reactions and the gas to solid adsorption and desorption of NO and CO is neglected. Fick *et al.* propose a set of kinetic equations which will be used for later analysis which is illustrated graphically in Figure 13.

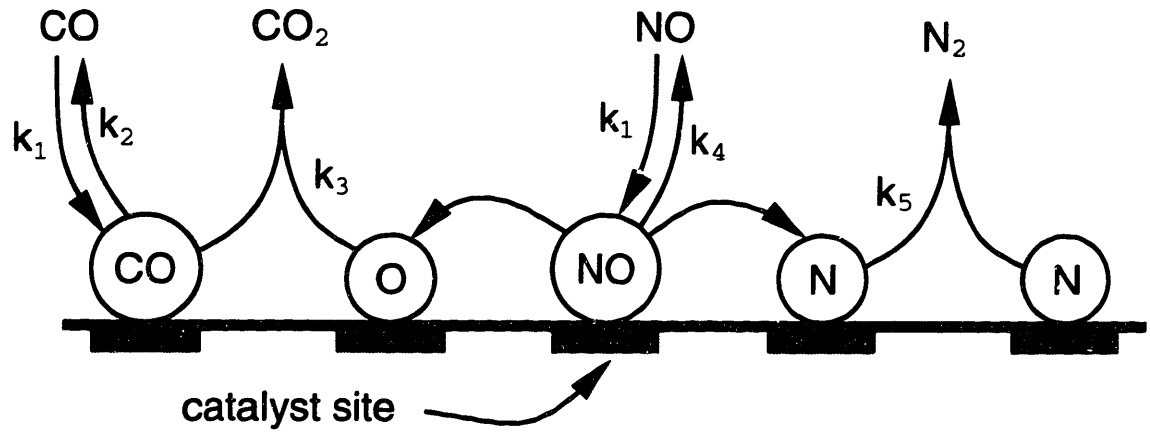


Figure 13. Reaction of NO and CO on Pt(100).

The equations describing this mechanism can be immediately written down using standard mass-action kinetic relations, here in terms of surface coverages:

$$\begin{aligned}
\dot{\theta}_{CO} &= k_1 p_{CO}(1 - \theta_{CO} - \theta_{NO}) - k_2\theta_{CO} - k_3\theta_{CO}\theta_O \\
\dot{\theta}_{NO} &= k_1 p_{NO}(1 - \theta_{CO} - \theta_{NO}) - k_4\theta_{NO} - k_5\theta_{NO}\theta_{empty} \\
\dot{\theta}_O &= k_5\theta_{NO}\theta_{empty} - k_3\theta_{CO}\theta_O
\end{aligned}$$

where

$$\theta_{empty} = \max\left\{1 - \frac{\theta_{NO} + \theta_{CO}}{\theta^{inh}_{CO,NO}} - \frac{\theta_O}{\theta^{inh}_O}, 0\right\}$$

and '$\dot{}$' denotes differentiation with respect to time. The inhibition coverages represent the requirement of auxiliary empty sites for the dissociation of NO and are given in Fick as 0.6 for NO and CO and 0.4 for oxygen. A further assumption is that the presence of adsorbed nitrogen has no effect, since is desorbs practically instantaneously after NO dissociation. Fick *et al.* report values for the k_i in Arrhenius form with activation energies as a function of coverage: $E_i = E_i^0 - 24$ kcal/mol $(\theta_{CO} + \theta_{NO})^2$.

The next natural step to pursue in this sequence of converting the real world to a set of mathematical equations is to linearize or decouple or somehow add further assumptions about the system

to render it analytically solvable. This procedure produces "solutions" which have little relationship to experimental results, similar to Lorenz's atmospheric motion equations mentioned in the introduction.

3.2 Nonlinear analysis

Instead of searching for a method of solution of the equations, an alternative approach to gain information about the solutions is by a qualitative stability analysis. For simplicity, the three expressions for time change of coverages can be written as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t))$$

where $\mathbf{x}(t)$ is a three-vector of coverages and $\mathbf{f}(\cdot)$ is the right hand sides of the equations above. The asymptotic stability of the vector field near a fixed point $\bar{\mathbf{x}}$ is given by Wiggins [143] by examining the eigenvalues of $D\mathbf{f}|_{\bar{\mathbf{x}}}$ *. For example, using the parameter values reported by Fick *et al.* to show oscillatory motion, it is found that the system reaches a global steady state at $\{0.156, 0.391, 0.0209\}$ where the Jacobian has eigenvalues $\{-3.12, 0.119 + 0.0631i, 0.119 - 0.0631i\}$, a double saddle with rotation on the unstable manifold, hence oscillatory as the experiments attest. As another example, if the reaction had happened to be started with large amounts of both NO and CO already adsorbed on the catalyst, a different fixed point is reached, $\bar{\mathbf{x}} = \{0.310, 0.269, 0.00467\}$ where the eigenvalues of the Jacobian are all negative, for asymptotic stability.

Rewriting the operating equation to show explicit dependence on parameters,

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t);\, T, p_{\mathrm{CO}}),$$

a form is obtained which permits analysis of the change in qualitative behavior in response to changes of the parameters. Temperature and CO partial pressure were chosen as two easily adjustable quantities which can be implemented directly in an experimental or industrial setting. NO pressure is assumed to change in response to keep the total pressure in the reactor constant. A global search for fixed points $\bar{\mathbf{x}}$, where $\dot{\mathbf{x}} = \mathbf{0}$, using common root finding procedures from starting points on several sides of the suspected stable region was systematically conducted, and determination of their stability produces a diagram showing where the system jumps from one area of operation to another. The algorithm is sketched in Table 1, and the results for this system are in Figures 14 and 15.

For each point $\mathbf{c}$ in parameter space,
search over $\mathbf{x}$ for fixed points $\bar{\mathbf{x}}$
calculate Jacobian $D\mathbf{f}\|_{\bar{\mathbf{x}}}$
classify stability
Group regions of identical stability type
Identify boundaries

Table 1. Algorithm used to generate bifurcation diagrams.

In the literature, this type of figure is often called a bifurcation diagram since it shows how solutions to the equations differ qualitatively as system parameters are adjusted. It does not show the location in phase space of the fixed points, just the behavior of trajectories near them. The various terms assigned to different regions are intuitive labels used to indicate the sign and presence of imaginary components of the three eigenvalues of $D\mathbf{f}|_{\bar{\mathbf{x}}}$, and are listed in Table 2. The fact that eigenvalues with nonzero imaginary component always appear in complex conjugate pairs reduces

* Solutions which produce any eigenvalues with zero real parts require special treatment by construction of a center manifold at the fixed point.

the number of entries in the table. The fixed points which occur in a stable region have all negative eigenvalues so that the system is at a local minimum. The presence of a single positive eigenvalue indicates that the surface at the fixed point has the shape of a saddle, and a system finding itself at that point will, under a slight perturbation, move away from the fixed point in the direction of the eigenvector corresponding to the positive eigenvalue. A double saddle has two unstable dimensions at the fixed point, and unstable points have all three. The presence of imaginary components has the effect of causing trajectories to spiral towards or away from the fixed point, but does not change the stability class.

$-\ -\ -$	stable
$-\ -\ +$	saddle
$-\ +\ +$	double saddle
$+\ +\ +$	unstable
$-\ -i\ -i$	spiraling stable
$+\ -i\ -i$	spiraling saddle
$-\ +i\ +i$	spiraling double saddle
$+\ +i\ +i$	spiraling unstable

Table 2. Names assigned to the possible eigenvalue combinations. The sign always refers to that of the real component of an eigenvalue, and i indicates the corresponding eigenvalue has a nonzero imaginary component.

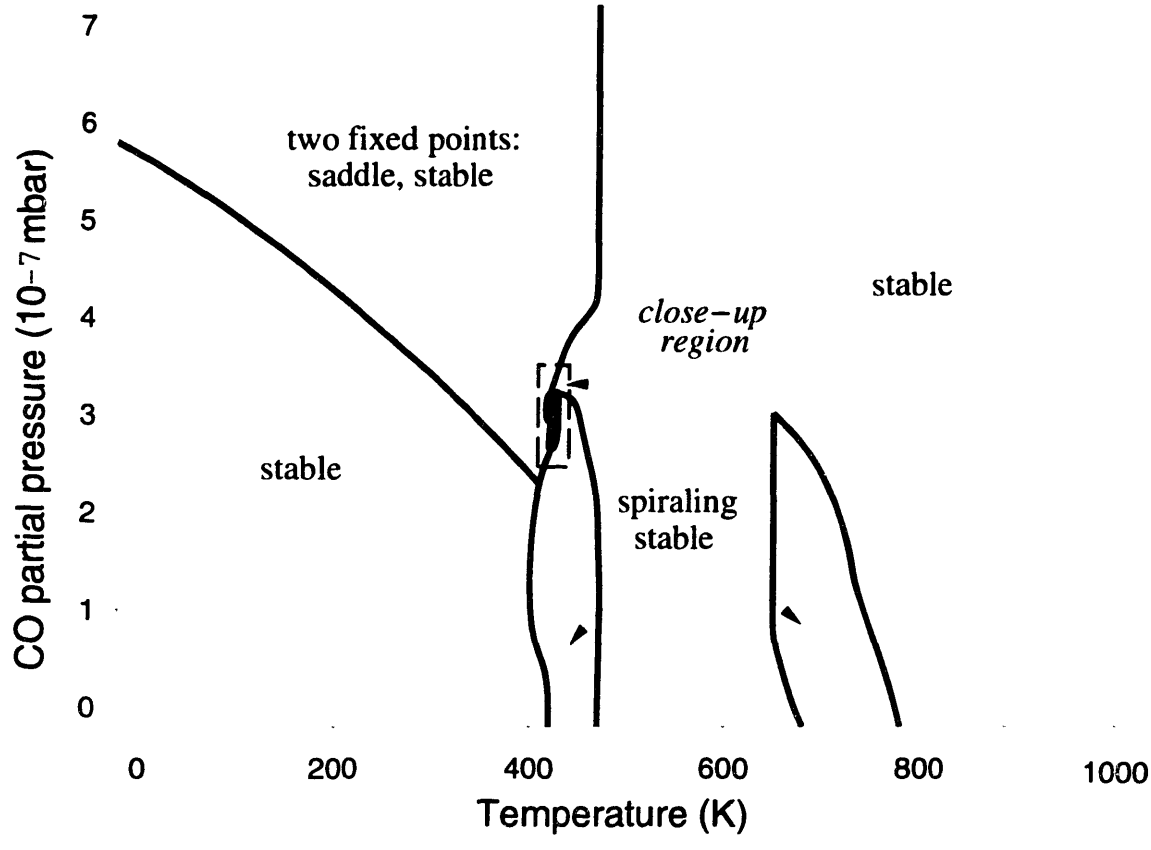


Figure 14. Bifurcation diagram showing regions of changing stability characteristics.

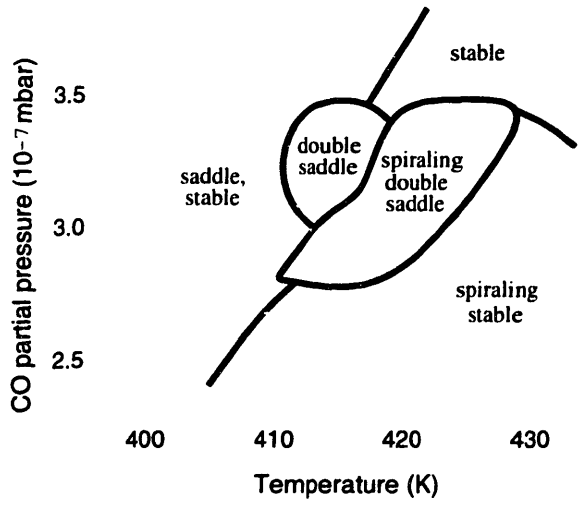


Figure 15. Close-up of bifurcation diagram in Figure 14.

Based on the nonlinear analysis, adjustments may be made to the system to elicit certain behavior. From the plot of Figure 14, it can be seen that there is a wide range of stable operation, and a goal of pure stability is easily met. But industrial application of catalytic reactors such as this often specify that the yield of products or conversion of reactants should be maximized, within certain safety limits. There may not be just one stable location which produces a maximum yield in the system as given, and employing non-steady control strategies could produce a higher conversion rate overall.

For example, the system at hand can be operated across the edge that connects the basins of attraction for two different stable points. The transition from one basin to another can be forced by altering any of the system parameters; here T and p_{CO} are assumed to be the easiest to adjust with a controller. The system can be forced to move from one zone of operation to another completely different one with minute adjustments of temperature and/or feed pressure.

As an example of the profound change in system output, consider a steady flow reactor designed to consume as much NO and CO as possible. Under the model outlined above, instantaneous consumption can be defined as:

$$Y_{CO} = \frac{L \cdot MW_{CO}}{N_{avg}} k_3 \theta_{CO} \theta_O$$
$$Y_{NO} = \frac{L \cdot MW_{NO}}{N_{avg}} k_5 \theta_{NO} \theta_{empty}$$

One particular operating condition in the stable region is at a temperature of 415K and with a CO feed pressure of 3.0×10^{-7} mbar, giving a consumption of 29.4 g/(hr g-cat) for both species. At the parameter values of Fick's experiments, in the nearby oscillatory region, the yields are $Y_{CO} = 26.5$ and $Y_{NO} = 28.3$ g/(hr g-cat), slightly lower. Figure 16 shows graphically the changes in the variables as a function of time to illustrate the nature of this operation.

With a minor controller modification to cause it to apply a step function to the inlet temperature of 40 K over a period of 30 seconds, so that in the reactor

$$T = 400\,\mathrm{K} + \begin{cases} 40, & 13 \le t \bmod 30 < 30 \text{ sec.} \\ 0, & 0 \le t \bmod 30 < 13, \end{cases}$$

one obtains forced oscillations between two stable conditions. This is illustrated graphically in Figure 17 and the yields with their improvements are given in Table 3. By the introduction of a

	consumption g/(hr g-cat)		improvement	
	CO	NO	CO	NO
stable	29.4	29.4	—	—
oscillatory	26.5	28.3	−9%	−4%
driven	34.4	36.8	17%	25%

Table 3. Effect of three possible operating conditions on consumption. "Improvement" is the percent consumption gain over stable operation.

variable control strategy, the NO/CO/Pt(100) process has moved from the domain of experimental science where negligible amounts of reactants were consumed into the world of possible industrial application.

3.3 Conclusion

The model calculations presented in this section are but one possibility for optimizing chemical reaction systems through a non-steady state control strategy. The bifurcation diagram of Figure 14 shows that many more regions of operation are available for exploration in this particular system as well. Further refinement of operating conditions could be aided by the construction of a Poincaré map perpendicular to either orbit of the consumption functions and solving the simpler two-dimensional system just generated to discover the resonant frequencies of the system. This particular application of nonlinear analysis techniques to a chemical engineering problem is but one of a wide range of possibilities for these tools.

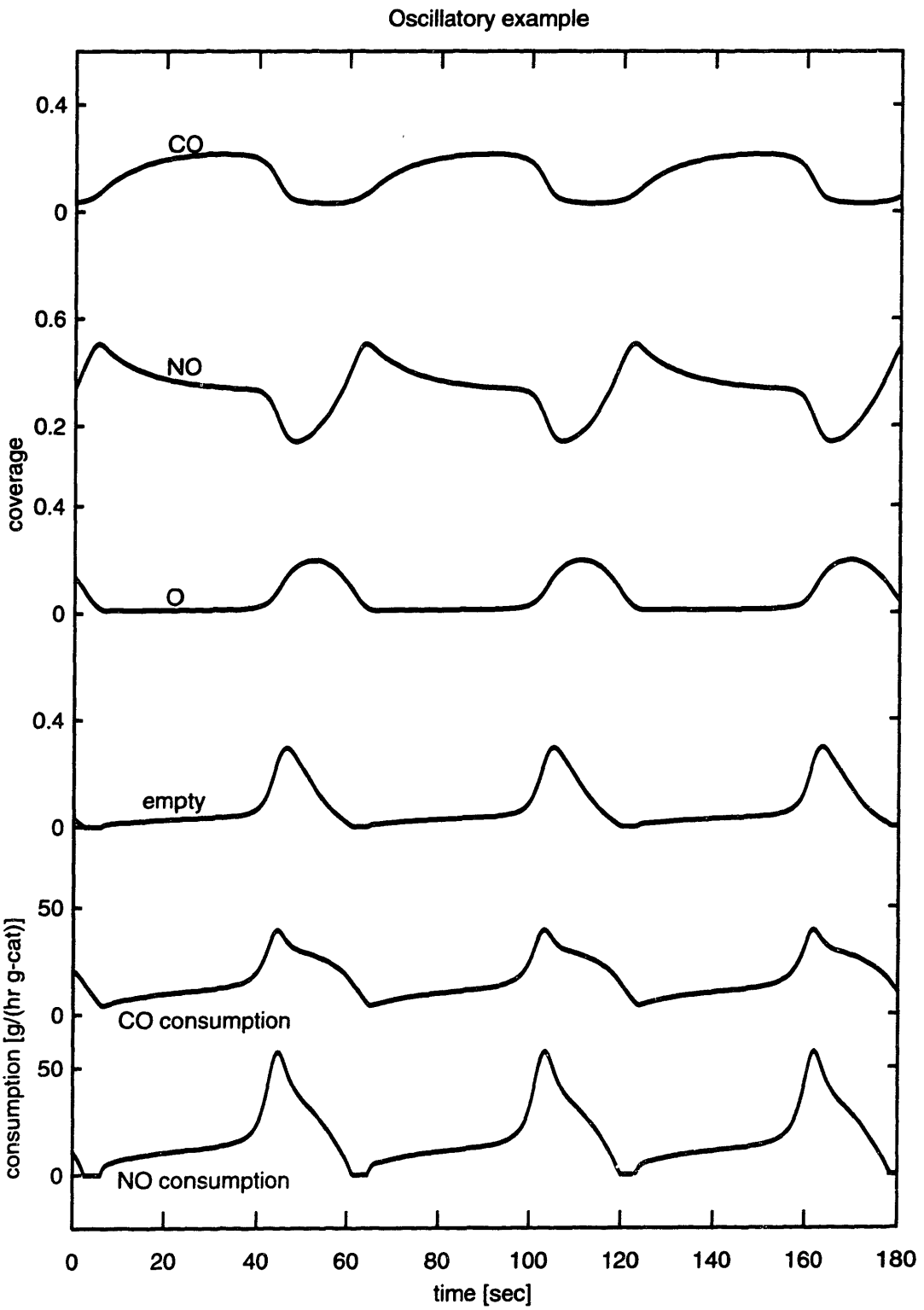


Figure 16. Catalyst surface coverages and consumptions during natural oscillatory operation.

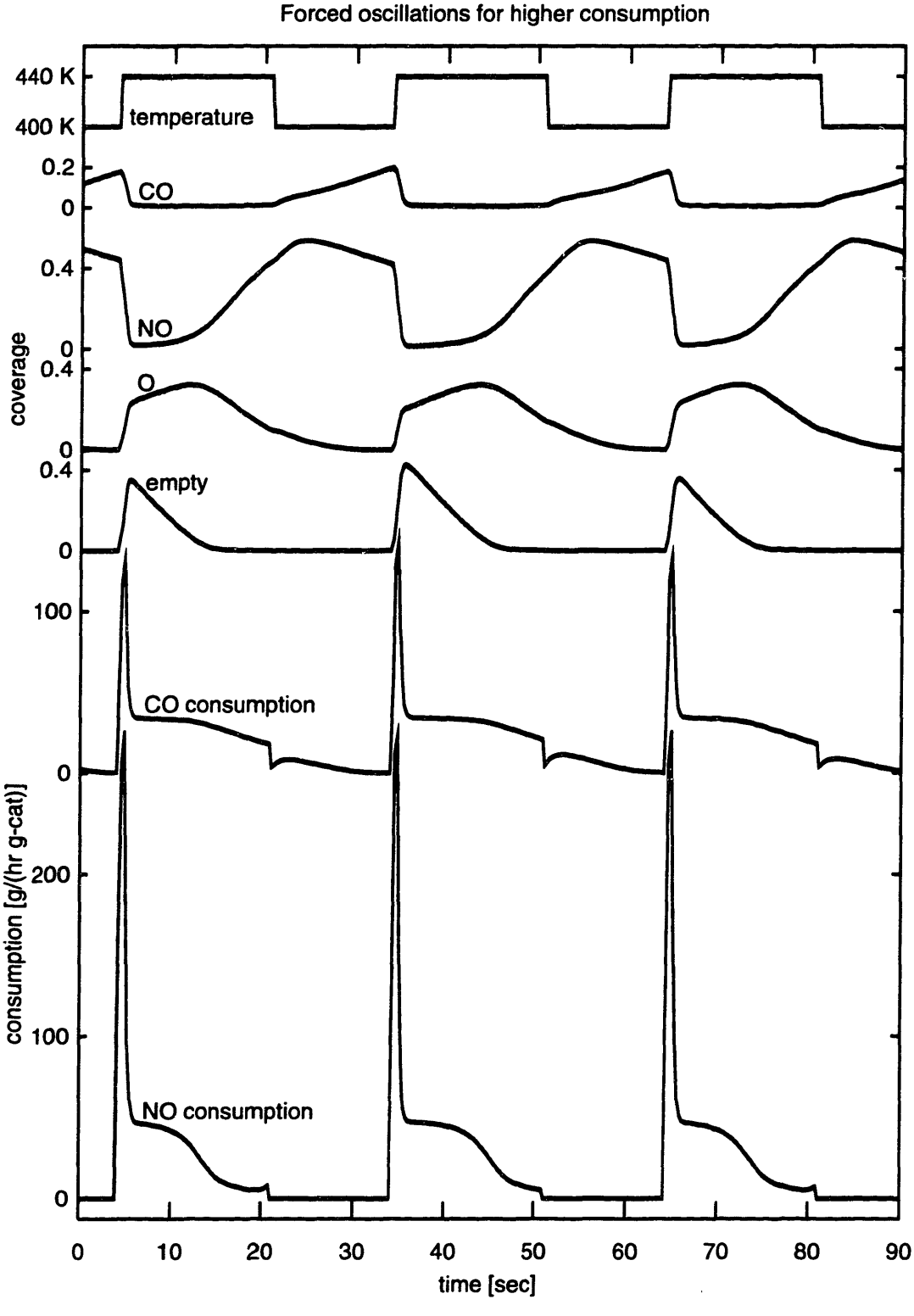


Figure 17. Catalyst surface coverages and consumptions during forced oscillatory operation.

Data analysis

Characteristics of the data generated by a process under study may yield information which can not be derived from any *a priori* source. This information can be used, perhaps, in developing low-dimensional faithful approximations of the system, or to elucidate inherent structures of the system otherwise undiscoverable, as discussed in the sections below.

4.1 Vibrating string

This section discusses topological analysis of a vibrating string, work done in collaboration with Nick Tufillaro at Los Alamos National Laboratory, and uses much of the material from the paper [135] produced by that work. Techniques called "topological time series analysis" are used to obtain the template organizing the chaotic dynamics and topological parameter values of a vibrating string directly from experimental time series. This information shows that the string is governed by a dissipative horseshoe map whose periodic orbit spectrum is well described by a unimodal map. Further, a model can be synchronized to the system allowing the creation of "synthetic data" which topologically agrees with the experimental data and can be used to obtain a finer characterization of the system which would not have been possible using only the original data.

4.1.1 Introduction

Study of the nonlinear and chaotic vibrations of a string is motivated by two intertwining concerns. First, we want to know strings. Or at least what we can know of strings via experiments showing chaotic motions in a vibrating wire. Second, given an experimental times series from a low dimensional chaotic process, we would like to provide a quantitative characterization of the chaotic attractor—or at least those properties of the attractor which are possibly shared by a large class of similar chaotic systems. Such a characterization should be robust under the effects of noise and smooth parameter changes. Such a characterization is necessarily topological.

Interest in a string's motion has a long and illustrious history. In 1883 Kirchoff derived an inherently nonlinear differential equation which properly took into account the simultaneous longitudinal and transverse displacements of a physical string [74]. In 1968 Narashima rederived a scaled version of Kirchoff's equation [101]. Miles in 1984, beginning with Narashima's equation, derived and analyzed a four dimensional model of the averaged oscillations of a string and discovered a Hopf bifurcation [90]. In 1989, after careful numerical studies, Johnson and Bajaj detected the presence of chaotic orbits in this same model [66]. Guided by these theoretical insights, experimental detection

of these chaotic oscillations soon followed [99, 107]. More complete historical references are found in several recent PhD studies on the nonlinear vibrations of a string [106, 131, 97].

Of particular interest to our experimental studies are the theoretical examinations of low dimensional nonlinear string models by Miles [90], Bajaj and Johnson [11], Tufillaro [130], and O'Reilly [107, 105, 89]; and the experimental reports of chaotic vibrations in a string by Molteno and Tufillaro [99], Molteno [97], and O'Reilly and Holmes [107]. As remarked by most of these authors, chaotic motions in a string are not easy to observe. They exist only in a small parameter range and occur not at the forcing frequency, but rather in the slow oscillations of the amplitude envelope. These amplitude modulations are usually only a fraction of the overall transverse displacement. These facts may help to account for the rather late discovery of chaotic vibrations in such a common system.

As recently advocated by several authors [114, 132], the quantitative topological characterization of a low dimensional chaotic invariant set should proceed in two steps. First, the strange set needs to be assigned a good symbolic encoding. We address this problem by identifying the template organizing the stretching, twisting, and folding motion of the attractor in phase space [136]. Second, rules need to be developed specifying the presence or absence of subsets of symbol sequences when parameters are changed. The second problem is sometimes called pruning [36, 62]. We present evidence that the "pruning problem" can be solved with predictions calculated by unimodal map theory—at least in the parameter regime considered and to within the resolution of the experimental measurements. Two types of evidence are used in support of these findings. The "symbol plane" is reconstructed for an experimental chaotic trajectory and it reveals that the system is well approximated by a vertical pruning front with no steps. Additionally, periodic orbits are extracted by the method of close recurrence [9, 134] and their spectrum is ordered by unimodal theory. Since the data appears to be well described by unimodal theory, we also estimate the value of the kneading sequence, the "topological parameter" which fixes the spectrum of periodic orbits.

In addition, we construct an empirical global model of the dynamics that produced the time series by determining a vector which best fits the data [26, 24]. The method used to fit the vector field is based on the minimum description length (MDL) criterion of Rissanen [118]. Two distinct methods are used in an attempt to address the question of how closely the empirical model fits the data. First, we show that we can synchronize the model to the data. Synchronizing these models to the data set provides the first piece of evidence that the empirical model is "close" to the flow observed experimentally [27]. Second, we show that the experimental data and a time series from the model share the same template and are close in terms of topological parameters.

This paper is organized as follows. Subsection 2 describes the string experiment and examines data sets taken from a parameter regime which leads to the onset of a crisis. Subsection 3 briefly describes how to construct a global empirical model from a data set. The Lyapunov spectra of the model and the data set are also calculated and it is suggested that calculating such system characteristics indirectly from "synthetic data" is a desirable and sensible procedure. Subsection 4 describes how periodic orbit spectra are extracted by the method of close recurrence for data sets at two distinct parameter values. It also describes how braid invariants are used to identify the template organizing the periodic and chaotic orbits of the strange attractor. A topological analysis of data generated from both the experiment and the empirical model is then presented. We call these data processing techniques "topological time series analysis," since they seek to ascertain topological form and topological parameter values directly from an experimental or simulated time series. Some concluding remarks are offered in subsection 5.

Lastly, we would like to remark that our experimental results show good qualitative agreement with the numerical simulations on nonlinear and chaotic vibrations in strings by Bajaj and Johnson [11]. This correspondence is discussed more fully in Molteno's PhD thesis [97].

4.1.2 Experimental setup and data set description

A schematic of the experimental apparatus used to excite and detect forced vibrations in a thin wire is shown in Figure 18. This apparatus is a considerably improved version of that described in the

first reference by Molteno and Tufillaro [99]. A non-magnetic thoriated tungsten wire, 0.15 mm in diameter, is fastened in a rigid mount. The wire sits in a permanent magnetic field created with rare earth ceramic magnets. An alternating current is passed through the wire to excite vibrations. Optoelectronic motion detectors are used to measure the transverse wire displacement simultaneously in two orthogonal directions. The detectors consist of an infrared photodiode coupled to a matched phototransistor positioned to detect the partial occlusion of the beam by the wire.

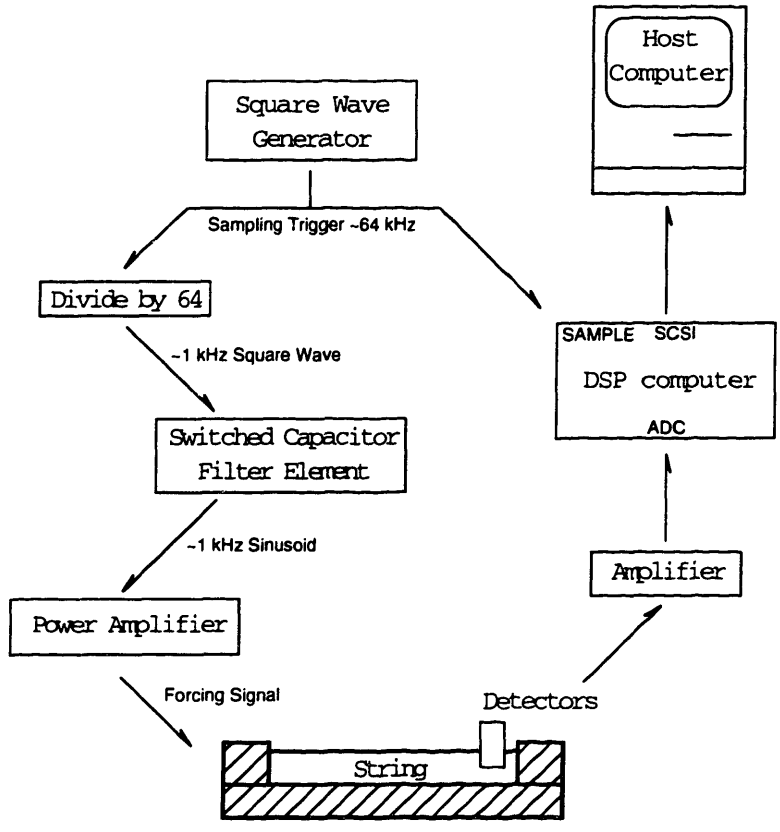


Figure 18. Schematic of experimental apparatus used to excite and detect forced vibrations in a thin wire.

Both the forcing current and detected signal are handled with a custom built controller/detector system designed around an Analog Devices ADSP 2105 Digital Signal Processor (DSP). It is a 16-bit device with a pipelined Harvard Architecture capable of performing 30 million operations per second. Direct Digital Synthesis (DDS) is used to generate the signal for forcing the wire. This technique is used because fine control over the forcing frequency is required to characterize the constant amplitude response of the wire. This digital control and detection scheme also allows the development of real-time nonlinear signal processing and identification tools such as variable (in phase and amplitude) Poincaré sections, Fast Fourier Transforms, and embedding plots with variable delays. All these real-time nonlinear signal processing tools greatly aid in the detection, identification, and optimization of the measurements reported here.

Typical parameter values for the wire and apparatus are: wire length, 0.07 m; mass per unit length, 3.39×10^{-4} kg/m; density 2.1×10^4 kg/m^3; Young's modulus, 197778×10^6 N/m; magnetic field strength, 0.2 Tesla; and a free frequency of vibration in the neighborhood of 1350 Hz. The damping of the system is measured from the decay rate of free vibrations and can be modified by the application of a silicone coating to the wire [107].

The transverse amplitude displacement of the wire is usually sampled once each forcing cycle at a fixed phase ($f_d \approx 1300$ Hz). The response frequency of the amplitude modulation is about 10 Hz. Thus, we have about 130 samples per cycle—the system is over sampled. Such over sampling is necessary, however, in order to calculate accurately some of the braid (extracted periodic orbit) invariants described in subsection 4. Since the interesting dynamics occurs so slowly, it is possible to save large data sets directly to disk; data sets of 10^5 measurements are easy to obtain. The amplitude displacement measurements are usually low-noise (less than 1% of the RMS amplitude) and make good use of the wide dynamic range available. However, two measurement problems are present. First the system is subject to a parametric drift (typically manifested as a 1% variation of the RMS amplitude over 200 seconds) which we believe is due to a temperature instability in the drive system. Second, there are some systematic measurement errors due to phase jitter in the digital detecting electronics. Overall, though, the measurements are of high quality: large, low noise, data sets making good use of the wide dynamic range made available from the 16-bit digitizers.

After being recorded the raw data is "dynamically" cleaned by the nonlinear noise reduction technique described in references [56, 69, 80, 122]. The scalar data is embedded using eleven dimensional delay coordinates. For each data point a small neighborhood is formed in which the two dimensional submanifold spanned by the attractor is approximated by a linear subspace. Projections onto this subspace yield an improved time series. The procedure is iterated six times and curvature corrections are applied.

In addition, dynamical cleaning provides an estimate of the noise level. For the data we have examined, the amplitude of the correction, which can be taken as an estimate for the amplitude of the noise in the data, is 55 units (0.44%). We also estimate the amount of noise in the data independently by the method given in [123]. The functional form of the increase of the correlation integral with embedding dimension is known for Gaussian measurement noise, so that the width of the Gaussian distribution can be determined by a simple function fit. We found that indeed the errors are compatible with a Gaussian distribution of width 65 units (0.5%). After nonlinear noise reduction the errors are no longer expected to be Gaussian but the remaining noise level is found to be at most 25 units (0.2%).

As the first step toward extracting a template from the time series, a three dimensional embedding of a chaotic trajectory is created out of the scalar amplitude measurements made by a single detector. This is accomplished via a time delayed embedding of the original scalar data set [3]. The offset for the delay is determined by the mutual information criterion [49]. Values for the offset range from 25 to 39 for the data sets examined, or, not unexpectedly, about one quarter of the samples per chaotic cycle. A false nearest neighbor test also indicates that the time series is embeddable in three dimensions [3]. Figure 19 shows three trajectories that are realized after a torus-doubling route to chaos [99]. Figures 19(a) and 19(b) show chaotic attractors, and Figure 19(c) shows a trajectory after the attractor has been extinguished by a crisis with a remote fixed point. These experimental results are in good qualitative agreement with the numerical simulations of Bajaj and Johnson [11]. Fractal dimension calculations of these chaotic time series indicate a dimension of 2.1 [98]. Perhaps the strongest evidence, though, of the low dimensionality of these data sets is obtained by constructing a first return map of a two dimensional Poincaré surface of section: these are shown in the insets of Figures 19(a) and 19(b).

4.1.3 A synchronized empirical model

Our goal in this section is to create an empirical (global) model of the time series data. We do this by generating a vector field for a three dimensional system of ordinary differential equations whose dynamics "mimics" the dynamics of the experimental time series. The ability to "learn the evolution rules" only from *a priori* information is a powerful technique which opens the door to further applications such as short-term prediction and control. Also, as suggested here, the empirical model can be used to generate large, low noise, "synthetic" data sets. Such a synthetic data set, or the analytic knowledge supplied by the model, often permits a more refined calculation of a system's

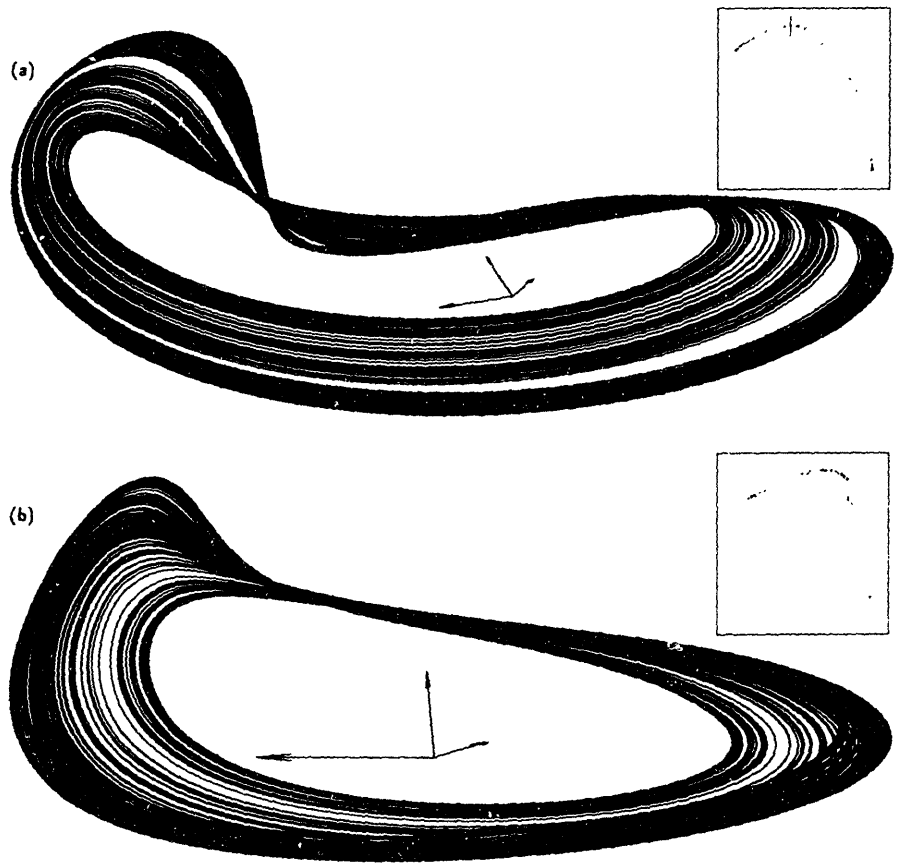


Figure 19. Three dimensional embedded time series from the string experiment at three different parameter values: (a) chaotic data set (a); (b) chaotic data set (b); (c) crisis with a remote fixed point. The insets show the low dimensional character of the next return maps constructed from a surface of section of the embedded attractors.

characteristics, such as its Lyapunov spectrum. However, before putting too much trust in the model, we must address the question of how faithfully the empirical model represents the dynamics that generated the data. We approach this question by two distinct methods. In this section we show that the model can be synchronized to the data [50, 113, 112]. In the next section we compare topological properties extracted from the experimental time series to the properties extracted from a time series generated by the model.

The method used to generate an empirical model directly from a data set is described in detail in a series of papers by Brown and co-workers [26, 24]. The method assumes that the dynamics that

produced the data vectors, $\mathbf{y}$, can be written as a set of ordinary differential equations

$$\frac{d\mathbf{y}}{dt} = \mathbf{F}(\mathbf{y}).$$

The vector field, $\mathbf{F}$, is written as a expansion in terms of polynomials that are orthonormal on the attractor represented by the experimental data

$$\mathbf{F}(\mathbf{y}) = \sum_{\mathbf{I}=\mathbf{0}}^{\mathbf{N_P}} \mathbf{p}^{(\mathbf{I})} \pi^{(\mathbf{I})}(\mathbf{y}).$$

These polynomials, $\pi^{(\mathbf{I})}$ form a basis set in which to write a model. The expansion coefficients, $\mathbf{p}^{(\mathbf{I})}$ are trained by modeling the system's dynamics as an implicit Adams integration scheme

$$\mathbf{y}(t+\tau) = \mathbf{y}(t) + \tau \sum_{j=0}^{M} a_j^{(M)} \mathbf{F}\big[\mathbf{y}(t-(j-1)\tau)\big],$$

where the $a_j^{(M)}$'s are the implicit Adams coefficients of order M.

In the traditional numerical integration scheme the coefficients, $\mathbf{p}^{(\mathbf{I})}$'s, are viewed as being fixed. In contrast, in the modeling context, we view these coefficients as being variable, their values being chosen to ensure a good correspondence between the experimental data and the empirical model. The model is trained by minimizing an MDL-type functional [118] which includes both a least squares minimization term and terms associated with the size and order of the vector field used to fit the data. The advantage of using an MDL-type functional is that an *optimal* model, from the class of polynomial models, can be found. Heuristically, the parameterization chosen is optimal in that it best fits the data with fewest number of terms. Higher order models may provide more accuracy, but the cost of the additional terms in the MDL functional outweighs the profit gained by the increase in accuracy.

To illustrate this modeling technique we consider the data set shown in Figure 19(b). It consists of 128×10^3 scalar 16-bit measurements of the vertical transverse string displacement. The forcing frequency in this particular example is 1.384 kHz and the characteristic response frequency of the amplitude modulation is about 9 Hz. The average mutual information and false near neighbor methods indicates that the optimal time delay and embedding dimension are $T = 39$ and $d = 3$ [3]. The model is trained by using a subsection of 10^4 points of the entire data set. More details about the modeling of this particular data set can be found in reference [27].

As suggested in references [26, 24] after the model is constructed we then use the experimental time series to *drive* the empirical model. We find that it the experimental time series is used to drive the model via dissipative coupling then the model synchronizes to the experimental time series. A detailed study of the quality of synchronization between the model and the data in the presence of additive noise and drift in the dynamics of the driving signal is presented in reference [27]. Synchronization between model and data provides the first piece of evidence that the empirical model and the experimental system are, in some respects, close.

We have also calculated the spectrum of Lyapunov exponents for the model as well as the data. Since it is the easier of the two calculations we first calculate the Lyapunov spectrum of the empirical model using a modified QR method [42, 111, 2, 1]. We only report the two nonzero exponents of the three dimensional model. The results are shown in Figures 20(a) and 20(b) where K represents the total number of Jacobians used for the calculations (the total evolution time is $K\tau$ where $\tau = 0.05$ is the normalized time between the data points). The figures indicate that for $K > 5000$ the values of λ_1 and λ_3 are essentially constant. We have used a small ensemble of ten different initial conditions taken from disparate regions of the attractor for our calculations. The error bars in the figures indicate the maximum and minimum values of the Lyapunov exponents

calculated from this ensemble. The ensemble is small, however the error bars indicate that the variance of the calculated values of the Lyapunov exponents over the initial conditions is also small. Furthermore, if the initial conditions for our calculations are within the basin of attraction of the attractor then the Lyapunov exponents are independent of the initial conditions in the $K\tau \rightarrow \infty$ limit [42, 2]. The relative independence of the calculated values of the Lyapunov exponents for $K > 5000$ and different initial conditions provide circumstantial evidence that our calculations have reached the asymptotic regime. We report the following values for the Lyapunov exponents of the empirical model: $\lambda_1 = 4.31 \times 10^{-2}$, and $\lambda_3 = -0.576$. These values are obtained by averaging over the ten initial conditions for $K = 15\,000$.

Calculating the full spectrum of Lyapunov exponents from an experimental data set is a notoriously difficult procedure. This is particularly true with regard to the negative exponents. In general, the best that one can hope for is to calculate a spectrum that is consistent with itself and whatever additional facts are known about the dynamics. The method we use to calculate the Lyapunov exponents from the experimental data is a minor variation of the one reported in reference [25]. This method uses polynomials to map local neighborhoods on the attractor into their time evolved images.

The first variation we employ involves using the modified QR method reported in references [2, 1] instead of the one reported in reference [25]. The second modification is more complex and involves the data used to form the attractor. This second modification is required because the original data set is highly over sampled and needs to be "thinned out" before a reliable estimate of the spectrum can be obtained. The initial calculations use every other data vector to form the attractor. Next, every third data vector is used to form the attractor. Finally, every fourth data vector is used to form the attractor.

The second modification also results in a change in the evolution time (step) associated with the polynomial maps from one local neighborhood to its image. We find that as the step increases the influence of experimental noise is reduced and consistent values are obtained for the Lyapunov exponents. In addition, the number of possible initial conditions increases. When we use every second data vector we have two different initial conditions, $\mathbf{y}(1)$ and $\mathbf{y}(2)$. Similarly when we use every third data vector we have three different initial conditions, etc. We use the different initial conditions as another consistency check for our calculations. For each case the number of Jacobians used in the calculation is $K \simeq N$.

In all cases the total number of vectors used to form the attractor is $N \simeq 10\,000$. Therefore, the size of the local neighborhoods used for the polynomial fitting is essentially the same for all tests. This insures consistency over our various tests. The value of $N \simeq 10\,000$ is chosen on the *ad hoc* belief that larger values of N will result in local neighborhoods that are so small that their dynamics will be unduly influenced by noise in the data [145, 23].

The results of these calculations are presented in Figures 21(a) and 21(b). When the order of the fit is greater than two the calculated values of the Lyapunov exponents become independent of the order of the fit. The only exception occurs for λ_3 and the attractor obtained by using every other data vector, step = 2. Since these values are associated with the negative Lyapunov exponent, and are well outside the range of the other results, we will ignore them. As one would expect, there is some spread in the calculated values of the Lyapunov exponents depending on the initial condition used and the different evolution times (step = 2, 3, or 4). In general however, the spread is small for both the positive *and* the negative Lyapunov exponent. As a final matter we note that the total evolution time for our calculation is $2K\tau$ when we use every other data point, $3K\tau$ when we use every third data point, etc. The independence of our results over step = 2, 3, and 4 indicates that $K \simeq 10\,000$ corresponds to the time asymptotic regime. The consistency of our results over changes in initial conditions, order of the fitting, and the evolution time leads us to conclude that the Lyapunov exponents calculated from the data set using this method are close to the true exponents. To obtain numerical values of the Lyapunov exponents we first averaged the calculated values of λ over the different initial conditions and then averaged that value over the order of the fit for all fits

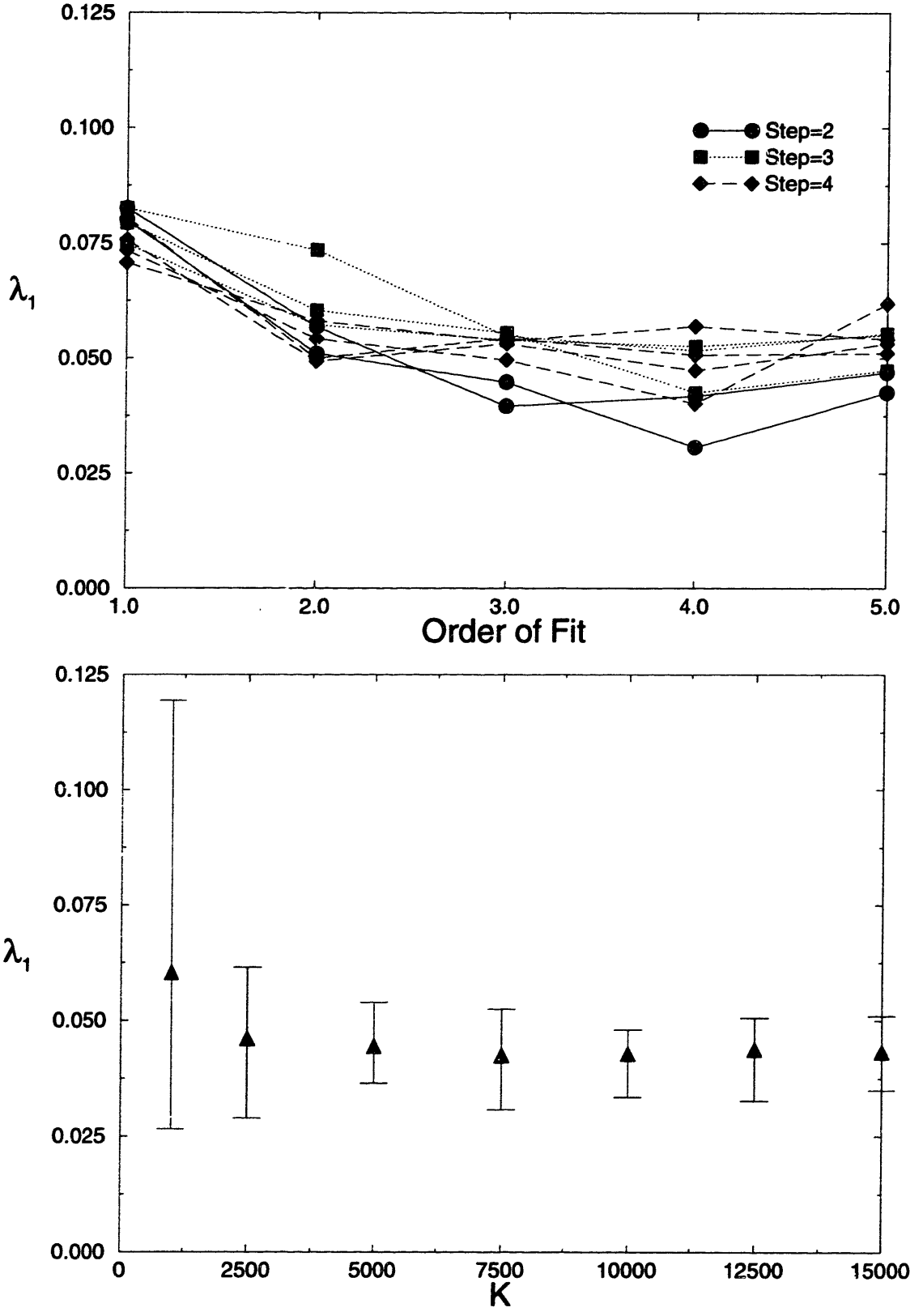


Figure 20. Lyapunov spectrum for empirical model.

greater than 2. We find that $\lambda_1 = 4.97 \times 10^{-2}$ and $\lambda_3 = -0.702$. These values differ from those reported for the model by approximately 15% for the positive exponent and 20% for the negative exponent.

4.1.4 Template identification

As the first step in a "topological time series analysis," we attempt to identify a "template" [136, 96, 93, 94, 110] organizing the periodic orbit structure of the flow. A template is nothing more or less than a cartoon of the stretching and folding structure of the flow written in a canonical form [88].

Before constructing the template, we note that the single-hump form of the first return map shown in the insets of Figures 19(a) and 19(b) suggests that a "once-fold" or "horseshoe" like map [36, 62] is organizing the global dynamics. Hence, one does not need the template to ascertain this topological information. However, the template contains an additional piece of topological information, namely, the global torsion, which can not be found by simply examining the first return map.

To ascertain the form of the template we proceed in a straightforward way. First, we examine the data in the three dimensional embedding by rotating it, and viewing it from several viewpoints with the aid of a graphics program we have written. This graphical visualization suggests that the sheeted structure shown in Figure 22(a) properly portrays the stretching, twisting, and folding of phase space which is organizing the orbit structure. The (negative) half-twist at the top of Figure 22 is followed by a small (positive) fold shown at the bottom (compare with Figure 22(b)). This small fold is responsible for the stretching (causing sensitive dependence) and folding (causing the periodic and aperiodic recurrence) that produces the chaos. Both of these features of the flow are themselves organized by basic topological properties of the flow, namely, the flow's fixed points and their homoclinic and heteroclinic connections.

To arrive at the canonical form of the template we separate the sheet shown in Figure 22(a) into two branches by splitting along the trajectory of the fold point (in the language of references [36, 62], we are creating a symbolic partition by considering the outermost "primary tangency"). Next we pull the small fold all the way to the bottom (thus going from a pruned to an unpruned system). The (negative) half twist in Figure 22(a) results in a half twist in each branch, with both branches crossing as shown in the top of Figure 22(b). The bottom of Figure 22(b) shows how the small fold results in a horseshoe template in standard form [88]. Now we push all the branch twists in Figure 22(b) to the top of the diagram and note that the twists in the left branch cancel, so that we arrive at the template shown in Figure 22(c). We note that this is itself a horseshoe template with a global torsion of zero. Our construction thus shows that a standard horseshoe template with a global negative half twist is again a horseshoe template.

In subsection 6 we will extract periodic orbits from chaotic time series and discuss how each orbit is given a symbolic itinerary. Given these periodic orbits, we then attempt to verify (or falsify) the conjecture that a horseshoe template with zero global torsion is organizing the dynamics. Specifically, we check two types of invariants: linking numbers of periodic orbits (a knot invariant), and the exponent sum (an invariant on *positive* braids) calculated from the "natural" braid associated to each periodic orbit* [88, 134]. As noted by Hall [60, 58], the *exponent sum is a complete braid invariant for all horseshoe braids up to period eight*—thus, this single invariant tells us the symbolic name (up to saddle-node pairs) of the low period orbits in a horseshoe without the need for an empirical symbolic partition. This is an effective procedure on orbits all of whose crossings are clearly resolved. (These are all the orbits examined up to period eleven in the model, but only orbits up to period six in the experimental data.) We have found no discrepancies between the invariants

* If a periodic orbit of period N is plotted in a Cartesian embedding space, then the "natural" braid on N strands associated to this periodic orbit is a plot of the trajectory in cylindrical coordinates in Cartesian space $(x, y, z) = (\theta \bmod 2\pi, r, z)$. Heuristically, this is the plot that results when the surface of section $\Sigma_{\theta(t)}$—whose z axis is perpendicular to the center of mass of the trajectory—sweeps through 2π.

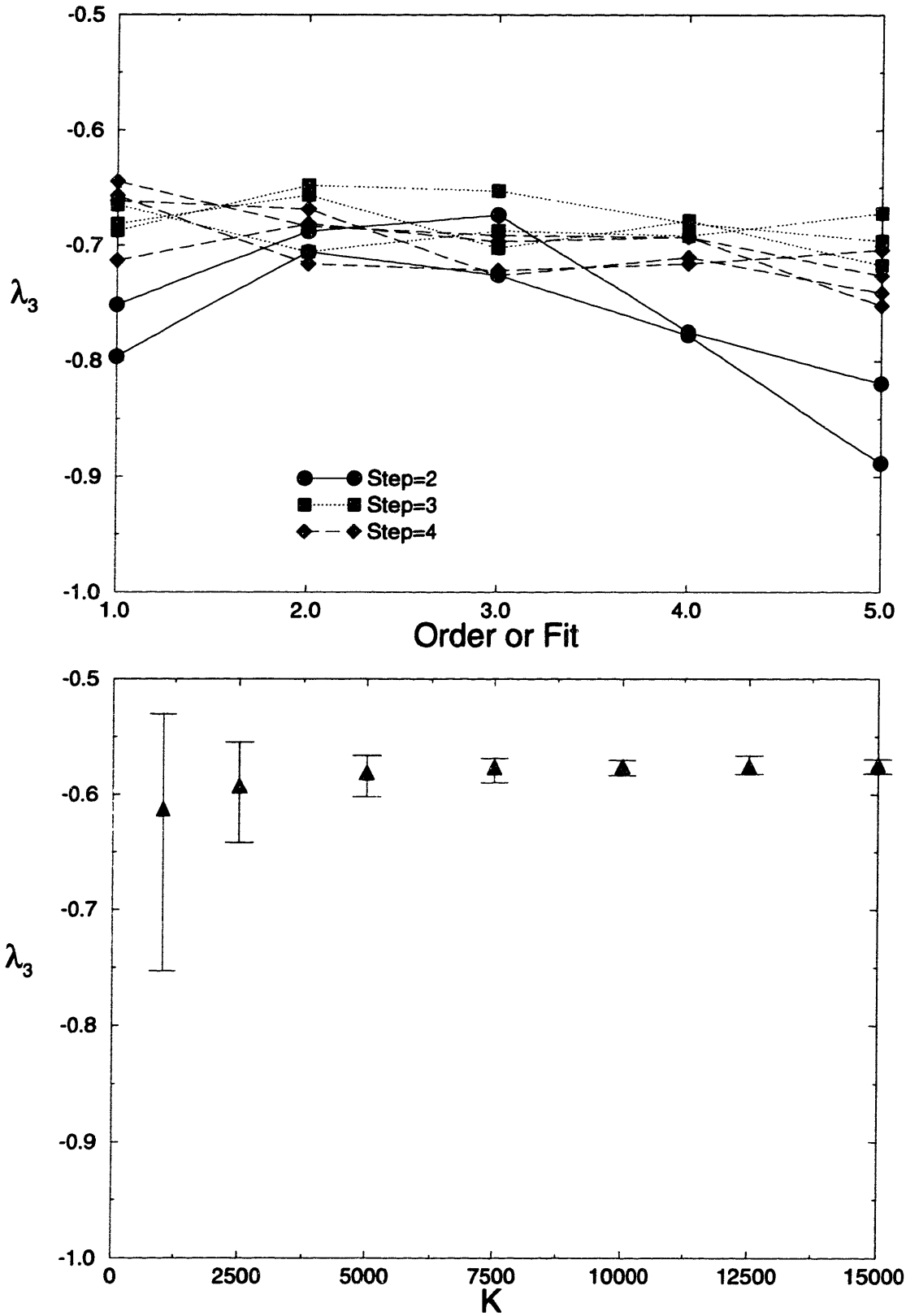


Figure 21. Lyapunov spectrum from experimental data.

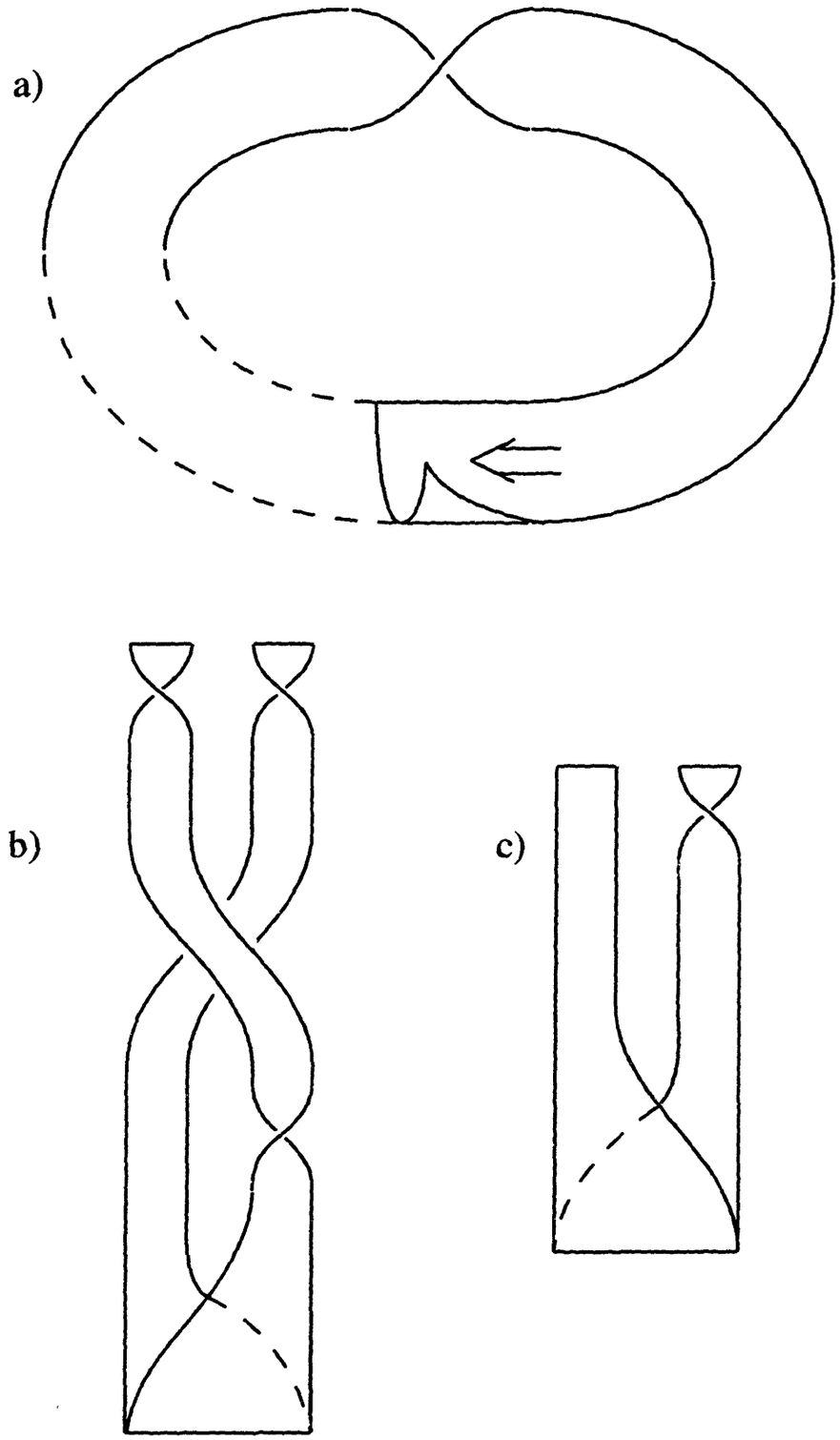


Figure 22. Transformations on a horseshoe template: (a) Schematic of sheeted structure in the experimental data; (b) The template obtained from (a); (c) Horseshoe template equivalent to (a).

predicted by a horseshoe template, those calculated from the data sets shown in Figure 19, and data from the model constructed in subsection 3. Some of these periodic orbits are shown in Figure 23. These results provide good evidence that—to within the resolution of the measurements and in the parameter regime considered—the orbits of the string and the model are governed by a horseshoe template with zero global torsion. It is encouraging that the model agrees in this first topological check. Evidently, the analytic properties of the model class, along with the initial conditions provided by the experimental data, are sufficient to determine (at least approximately) the fixed points which are in the vicinity of the sampled flow.

4.1.5 Symbol plane

To get a topological "road map" of the data we use an empirical procedure to construct the symbol plane generated by a single chaotic trajectory [36, 62]. We construct two symbol planes, one for data from the experiment (Figure 19(b)), consisting of 725 points in the return map, and one for the model, consisting of 4500 points in the return map. To construct the symbol plane we first convert the chaotic trajectory to a symbolic string of 0's and 1's, depending on whether the orbit passes to the left or right of the maximum point of the first return map shown in the inset of Figure 19(b). As shown in the next subsection, this empirical symbolic partition is fine enough to distinguish orbits at least up to period eleven. Now, since we know the topological form (the template) of the chaotic set, we can use kneading theory to determine the "well ordered" [36, 62, 92] symbol sequences needed to construct the symbolic plane.

After conversion the scalar data set has become a symbol string of the form

$$\mathbf{s} = \ldots s_{-3}s_{-2}s_{-1}s_0.s_1s_2s_3\ldots$$

where symbols to the left and right of s_0 are the past and future symbols respectively. The coordinates of the symbol plane are calculated from the well ordered past (c_i) and future (b_i) symbols as follows:

$$x(\mathbf{s}) = \sum_{i=1}^{D} \frac{b_i}{2^i}, \qquad b_i = \sum_{j=1}^{i} s_j \bmod 2$$

and

$$y(\mathbf{s}) = \sum_{i=0}^{D-1} \frac{c_i}{2^i}. \qquad c_i = \sum_{j=0}^{i+1} s_{-j} \bmod 2.$$

If $\mathbf{s}$ is an infinite symbol string generated by a chaotic orbit, then D is infinity in the above sums. However, since we are dealing with finite data sets, we approximate the symbol plane coordinates of a point by taking $D = 16$. In this way we can use a finite symbol string from a chaotic trajectory to generate a sequence of points on the symbol plane.

The resulting plots for both the experimental and model data are shown in Figure 24. These plots suggest several nontrivial observations about the topological properties of the flows producing these data sets. First, it appears that a good approximation to the "pruning front" [36, 62] is a single vertical line (a schematic for this pruning front is illustrated in Figure 24), at least to within the resolution of our experimental measurements and data processing procedures. This indicates that, at the topological level, the symbolic dynamics of the system are well-described by unimodal map theory, with the pruning determined by a single topological parameter, the kneading invariant [36, 62, 92], which we can attempt to approximate.* Second, the close similarity between

* The one-dimensional nature of the data set came as a surprise to us. We were expecting, indeed hoping, for a more two-dimensional data set—many stepped pruning front—in order to test the data processing techniques described here. Using these techniques to approximate a many stepped pruning front is discussed by N. Tufillaro in reference [133].

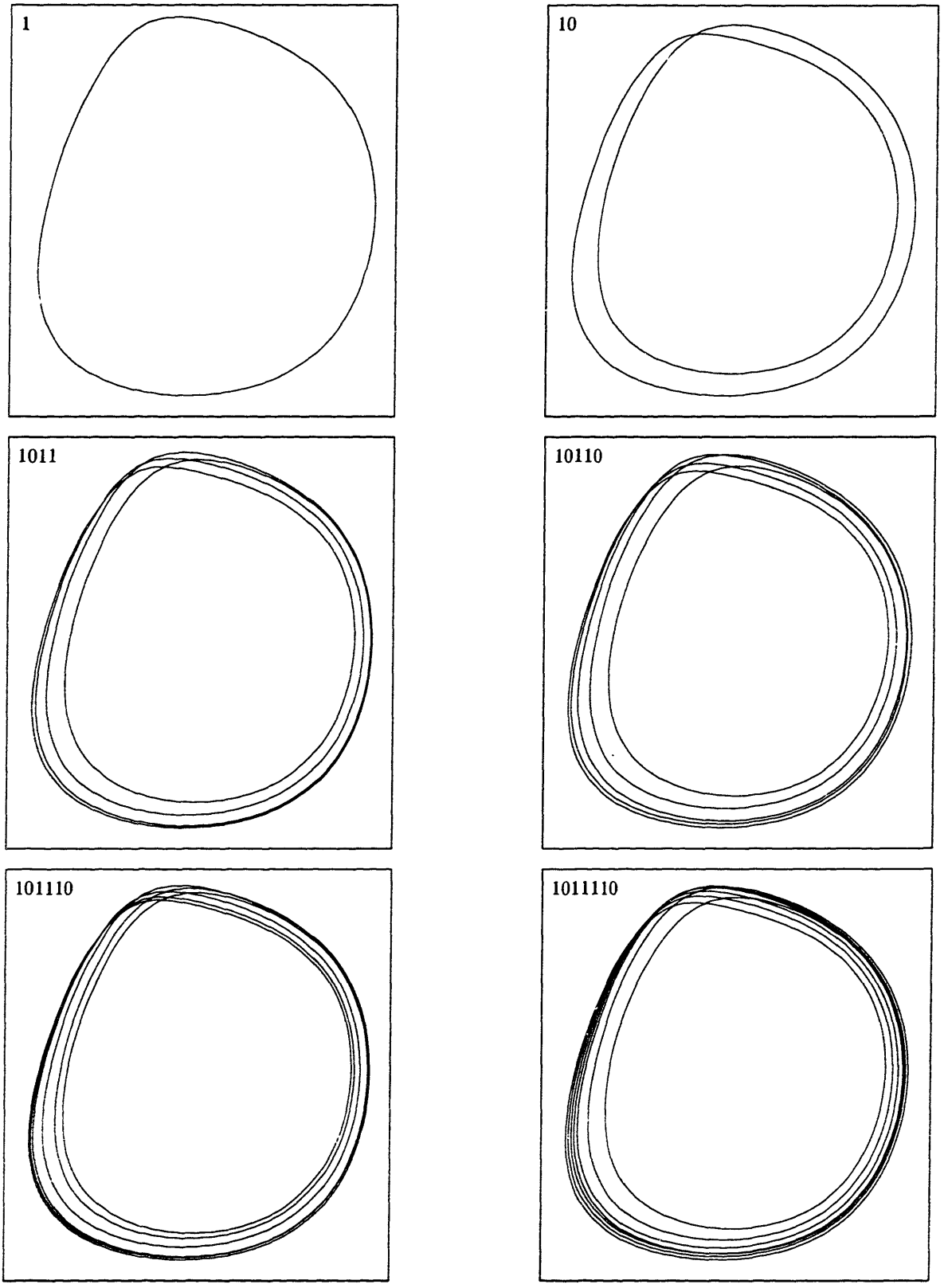


Figure 23. Some periodic orbits extracted from a chaotic time series from the string experiment.

the plot generated by experimental data, and model data, again provides striking evidence that the empirical model correctly captures the topological properties of the flow. Moreover, we can estimate the closeness of this topological fit by comparing estimates of the kneading invariant from the experimental data and model data. Third, the simple form of these plots, and the conjectured (approximate) pruning front, provides yet another powerful self-consistency check that the template (and hence the formula for well-ordering the symbol sequences) is identified correctly. In the next section we extract periodic orbits from the chaotic time series and show how they can be used in this example to systematically approximate the vertical pruning front suggested by Figure 24.

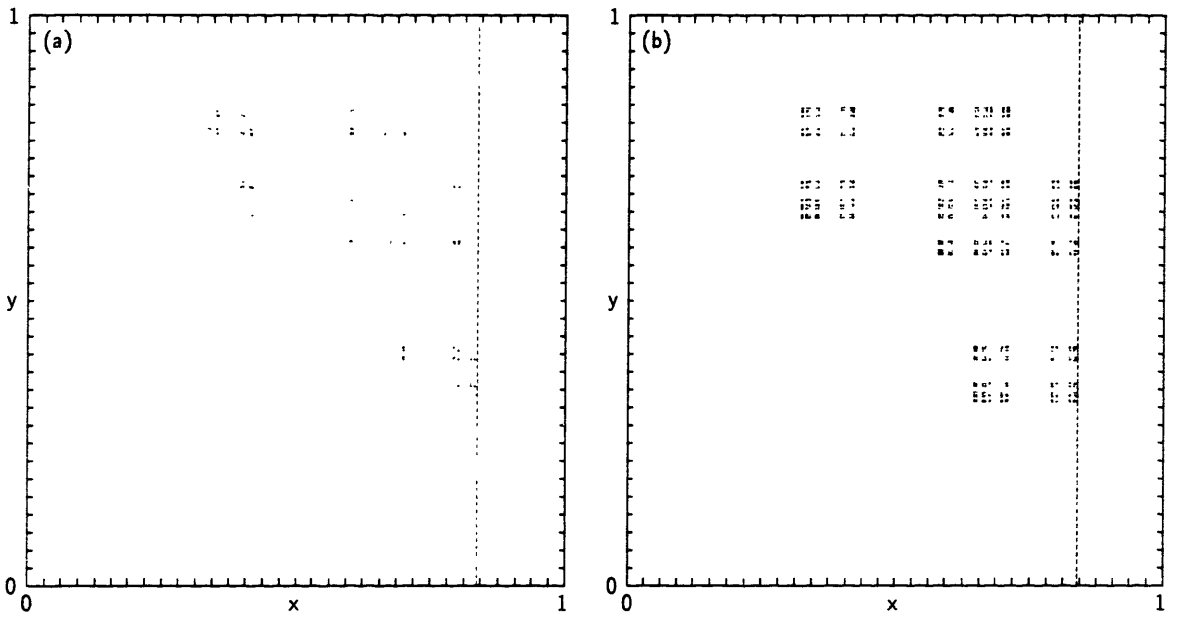


Figure 24. Symbol planes generated by chaotic time series from a string: (a) Experimental data; (b) Model (synthetic data).

4.1.6 Periodic orbit spectra

To extract the (approximate) periodic orbits by the method of close recurrence we first convert the next impact map from the coordinate values directly into a symbol sequence. In this particular instance, we found that an adequate symbolic description (at least up to period eleven orbits, or approximately one part in 2^{11}) is obtained by choosing the maximum value of the next impact shown in the insets of in Figure 19. Orbits passing to the left of the maximum are labeled '0', and those to the right are labeled '1'. Next we search this symbolic encoding for each and every periodic symbol string. Every time a periodic symbol string is found we calculate its (normalized) recurrence and then save the instance of the orbit with the best recurrence. The advantage of this procedure of periodic orbit extraction is that it is exhaustive. We search for every possible orbit up to a given period. In these studies we searched for all orbits of period one through eleven.

The resulting spectrum of periodic orbits for both experimental and model data sets is shown in Table 4. The orbits which are present in (the full shift) complete hyperbolic system, and not present in the Table 4 are said to be *pruned*. Two topological invariants (the linking number and the exponent sum of positive braids) of the extracted orbits are calculated and compared with those of a horseshoe (see Table 4. There are no discrepancies on the orbits in which the crossings can be unambiguously resolved. This indicates that, at least to this level of resolution, the template is correctly identified and the symbolic partition is adequate. Moreover, the template for the string experiment remains unchanged for the two different experimental parameter settings examined,

though the pruning is much more severe in data set (a) than it is in data set (c). The symbolic label (up to braid type) can also be determined—and used as yet another self-consistency check—by considering a simple and easily computable braid invariants. As pointed out by Hall [60, 58], the exponent sum (simply the sum of braid crossings in our example) is a complete invariant for horseshoe braids up to period eight. Also, an inspection of the exponent sums as a function of period reveals that the exponent sum manages to distinguish most of the pseudo-Anosov horseshoe braids of periods nine, ten, and eleven as well [133]. The symbolics determined by the braid type, and that determined by the empirical partition are consistent for all the orbits listed in Table 4. Our goal now is to predict as best as possible the pruned spectrum from the chaotic time series.

P	c_P	es	h_1	(a)	(b)	(c)	(d)
s_1^1	1	0	0	–	0.02192	0.01383	0.01880
s_2^1	10	1	0	0.01796	0.01459	0.01773	0.02929
s_4^1	1011	5	0	0.00840	0.01287	0.00993	0.02386
s_5^1	10110	8	0.414	–	–	–	0.02678
s_5^1	10111	8	0.414	–	–	–	0.00863
s_6^1	101110	13	0.241	0.00780	0.00433	0.00849	0.01106
s_6^1	101111	13	0.241	0.00675	0.00381	0.01350	0.01202
s_7^1	1011110	18	0.382	–	0.00634	0.00917	0.00994
s_7^1	1011111	18	0.382	–	–	–	0.01260
s_8^1	10111010	23	0	0.00750	0.00299	0.00789	0.01903
s_8^2	10111110	25	0.305	–	0.02241	0.01770	0.02087
s_8^2	10111111	25	0.305	–	–	–	0.01264
s_9^1	101111110	32	0.366	–	0.00893	0.01421	0.00662
s_9^1	101111111	32	0.366	–	–	–	0.02458
s_9^2	101111010	30	0.397	–	0.01864	0.02638	0.02201
s_9^2	101111011	30	0.397	–	–	–	0.02336
s_{10}^1	1011101010	37	0.207	0.00752	0.00225	0.00635	0.01260
s_{10}^1	1011101011	37	0.207	0.00511	0.01560	0.01826	0.02052
s_{10}^2	1011111010	39	0.272	–	0.01694	0.00859	0.00452
s_{10}^2	1011111011	39	0.272	0.02766	0.01312	0.01875	0.01438
s_{10}^3	1011111110	41	0.328	–	0.00622	0.02037	0.01512
s_{10}^3	1011111111	41	0.328	–	–	–	0.00772
s_{11}^1	10111111110	50	0.357	–	0.02365	0.01339	0.02657
s_{11}^1	10111111111	50	0.357	–	–	–	0.00974
s_{11}^2	10111111010	48	0.374	–	–	–	0.00960
s_{11}^2	10111111011	48	0.374	–	–	–	0.01491
s_{11}^3	10111101010	46	0.390	–	–	0.02305	0.01199
s_{11}^3	10111101011	46	0.390	–	–	–	0.01811
s_{11}^4	10111101110	46	0.403	–	–	0.01483	0.01780
s_{11}^4	10111101111	46	0.403	–	–	–	0.01600

Table 4. Spectrum of low period orbits extracted from chaotic time series (all orbits with $\varepsilon < 0.03$ are shown). Extracted orbits, their exponent sum, one-dimensional topological entropy, and their (best) normalized recurrence are recorded. Data set (a): experimental with 500 points in return map; Data set (b): dynamically cleaned version of data set (c); Data set (c): experimental with 725 points in return map; Data set (d): empirical model of data set (c)—the "synthetic data" (4500 points in return map).

Before we discuss pruning, though, it is useful to consider the number of distinct periodic orbits extracted as a function of the number of points in the return map. This is shown for the model data in Table 5. As expected, the number of orbits that can be extracted increases with the number of points in the return map. More importantly, Table 5 strongly suggests that using the method

of close recurrence it is possible to obtain all the low period orbits embedded within the strange attractor. For instance, after 10^5 points are examined, we see that no new periodic orbits are found below period seven. Similarly, after examining 5×10^5 points we believe we have found all orbits up to period nine. These results also caution us when working with smaller samples such as those in data set (a) with 500 points in the return map, and data set (b) with 725 points in the return map—the extracted orbit spectrum is expected to miss orbits either because the orbit is pruned (it is not in the strange set) or because the sample of the strange set we are examining fails to provide a close enough coverage over the whole attractor.

As suggested by the symbol plane diagram (Figure 24), unimodal map theory should be sufficient to explain these periodic orbit spectra. To test this hypothesis, we first examine the periodic orbit spectra from the model data set and locate the maximal (rightmost) periodic point in terms of unimodal theory. It is a period five orbit with symbolic name 10110, and indicated by the label (up to saddle-node pairs) of s_5^1. In unimodal theory, the s_5^1 orbit *forces* [59] the orbits s_{11}^4, s_9^2, s_{11}^3, s_7^1, s_{11}^2, s_9^1, s_{11}^1, s_{10}^3, s_8^2, s_{10}^2, s_6^1, s_{10}^1, s_8^1, s_4^1, s_2^1, and s_1^1. The notation identifies (up to saddle-node pairs) the period of each orbit in the subscript, and the value in terms of unimodal ordering (within each period) in the superscript. This theoretically determined spectrum agrees with that found in Table 4. In fact, both partners of all the saddle-node pairs are present except for the period one orbit pair. In this case, though, it is known that the period one orbit with symbolic label '0' is not present in the strange attractor, though it is present in the flow. Moreover, we can now use the maximal periodic orbit to estimate the itinerary of the kneading sequence in the model data set, $\kappa_m \approx \overline{10110}$, or in two dimensions by the (vertical) pruning front specified by $\overline{0}_1^0.\overline{10110}$. The horizontal coordinate of the (maximal) point of this periodic orbit on the symbol plane is $x(\mathbf{s}) \approx 0.8485$. An estimate of the kneading invariant also provides an estimate of the (one-dimensional) topological entropy [21, 95]: $h_1(s_5^1) \approx 0.4140$.

P	5	10	50	100	500	816
1	0	0	1	1	1	1
2	0	0	0	0	1	1
3	0	0	0	0	0	0
4	0	1	1	1	1	1
5	1	1	1	2	2	2
6	0	0	2	2	2	2
7	0	1	2	2	2	2
8	0	1	1	2	3	3
9	0	0	1	3	4	4
10	0	1	2	4	6	6
11	2	2	2	4	8	8

Table 5. Number of distinct periodic orbits of a given period extracted as a function of the sample size ($\times 10^3$).

A similar analysis of the experimental data set (c) shows that the maximal periodic orbit in the unimodal ordering is s_{11}^4 which forces s_9^2, s_{11}^3, s_7^1, s_{11}^2, s_9^1, s_{11}^1, s_{10}^3, s_8^2, s_{10}^2, s_6^1, s_{10}^1, s_8^1, s_4^1, s_2^1, and s_1^1. Up to period six, all the orbits forced by s_{11}^4 are present. Nine orbits are missing between periods seven through eleven. But we believe that these higher period orbits are missing due to the small sample size. We use the maximal orbit to approximate the itinerary of the kneading sequence in the experimental data set as $\kappa_b \approx \overline{10111101111}$, or in two dimensions by the (vertical) pruning front specified by $\overline{0}_1^0.\overline{10111101111}$. The horizontal coordinate of the maximal point of this periodic orbit on the symbol plane is $x(\mathbf{s}) \approx 0.8385$. An estimate of the one-dimensional topological entropy derived from the experimental data is: $h_1(s_{11}^4) \approx 0.4032$.

Time series from the model and experiment are close in both their topological form and topological parameter value(s). The model, though, does appear to determine parameter values which

over estimate the topological entropy. This difference in entropy could be due to several sources. First, uncertainties due to noise and parametric drift in the experimental data set place inherent limits on the modeling procedure's accuracy. Second, low-period orbits, perhaps of higher entropy than those extracted, may be present, but are not found because of the limited sample size of the experimental data. Third, the estimation in the entropy can not be any finer than that allowed by the order of the periodic orbit approximation. As an examination of Table 4 indicates, the difference in entropy between the two periodic orbits considered (up to period eleven), which are close in the unimodal ordering, is roughly 0.01. This last point suggests that the estimated difference in entropy between the experimental data and model data is an upper bound on the difference. The entropy between the two process may, in fact, be closer than this estimate indicates.

4.1.7 Concluding remarks

In this paper we create an empirical global model of a data set from a string experiment and show that the model captures the dynamics implicit within the data set in at least two significant respects: the model and data can be made to synchronize [26, 24, 50, 113, 112], and they share quantifiable common topological properties. We also suggest that it is a sensible and desirable procedure to use synthetic data from the model to characterize the system in ways which may not be readily accessible from the data set alone. In particular we discuss how to characterize the system at the topological level by its periodic orbit spectrum, and at the metric level by its Lyapunov spectrum. More generally, we illustrate how to characterize a chaotic invariant set both by its general form (a template) and specific topological parameter values (kneading sequences) estimated from the spectrum of periodic orbits of the chaotic attractor, or from an empirically constructed "pruning front" [36, 62].

Global, empirically constructed, analytic models open the door to a host of practical applications in systems analysis, identification, and control [26, 24]. In particular, we emphasize how synthetic data sets permit the calculation of quantities requiring data with a larger sample size, or lower noise, than may be directly available from the experimental data alone [23]. In addition, we would also like to emphasize how a synchronized model of a system can be coupled directly to a data stream permitting a kind of model-based real-time dynamic filter [30].

4.2 Further applications

The techniques discussed above for the particular case of a vibrating string have possibly wide applicability. Standard time series analysis of arbitrary systems is fairly well known now, but efforts to model these systems are thwarted by the fact that each type of nonlinearity has its own character so general techniques applicable to all systems do not exist, as opposed to in the linear case. The use of time series data in the modeling process has the obvious advantage of tailoring the model to the specific problem, and the algorithm discussed above has the advantage of generating optimal models in the sense that they are the simplest ones that capture all the information in the data. Also the ability to synchronize the model with a running experiment is important as it opens up the possibility of using the model as a feedback control system to adjust continually the physical system. There are many applications which seem to be candidates for this type of analysis, two of which are described here.

Fluid flows. Systems operating in the transitional Reynolds number regime between laminar and turbulent flow exhibit highly nonlinear coherent structures, such as the shedding of vortices for flow past a cylinder or near a surface. These systems can be described by only a few degrees of freedom when considered in the appropriate space. That space can be found by analyzing images of the experimental system using a Karhunen-Loève transform, for instance, to extract out the important structures, then applying the full mechanics described above on the coefficients of that expansion.

System identification. Topological characteristics of a well-understood system (or at least a given one) may be extracted from its time series output and used in analysis of novel systems. If the unknown system will synchronize to the ordinary differential equations prescribed by a known system, then the dynamics of both systems must be identical. Libraries of understood systems may be amassed and new systems tested systematically against the collection. This procedure of system identification is superior in its robust response in the face of process noise, as demonstrated above.

Representation

This chapter discusses the idea of choosing the proper language in which to formulate a given problem. High school algebra teaches us the most rudimentary method of moving to a different representation, in the form of a story problem whose words must be translated into the language of algebra to permit the manipulations leading to the solution (*e.g.*, yards of fence needed to enclose Uncle Jacob's field). More advanced, but still mathematical, forms of this include all the convenient transformations such as Fourier and Laplace. These transformations were motivated by particular problems: in the case of Fourier, a bounded domain with periodic boundary conditions, and for Laplace, functions of a positive variable like time in initial value problems.

Another class of representational transformations operates at a more linguistic or structural level. One example is a group of multilingual speakers agreeing on one common language in which to conduct their discussion. Languages include those with which we communicate with computers as well, and the design of many of these was motivated by the expected application. Early modes of interacting with the machines could hardly be called languages, as the programmer was reduced to hand-coding instructions in hexadecimal. The first true languages were built around constraints of the machines themselves, which led to some strange artifacts such as the difference in C between pre- and post-increment. (VAX architecture made it fast and convenient to *post*-increment or *pre*-decrement, but slower to do the opposite operations.) FORTRAN has complex number mathematics built into the language, making it popular with the scientific applications community. COBOL [100] was designed for business transactions and has constructs to make easy the printing of accounting reports using fixed-width monetary notation.

The introduction of multiprocessor supercomputers has added another dimension to the design of efficient data structures. Variations in languages to accommodate the need to choose explicit data distribution and transport schemes are not widely accepted, as most programmers prefer to handle scheduling and message passing using libraries of explicit calls, again sinking to the lowest possible level of operation. FORTRAN 90 has been around for some time, and certain compilers will implement full array operations in parallel, but data layout issues are only considered in the as yet standardless HPF, high performance FORTRAN. Truly machine-independent multiprocessor languages must by necessity of their compilation be somewhat cryptic to the standard programmer, and their development suffers due to this small group of users [7].

The two sections presented here are at an even higher level of abstraction, specific to very particular problems. The first section below considers the problem of mapping a set of reacting chemical species across multiple processors attempting to minimize the total execution time. Transforming

the problem statement into a variation on the common set covering problem allows utilization of previous research in combinatorics to achieve the optimal solution to this problem.

Paradigms have come about regarding the language of data structures internal to a computer program, such as linked lists or binary trees, because of the ease with which certain operations may be performed on certain structures. The second section below discusses particular uses of these structures in transforming human-written lists of chemical reactions into machine-generated FORTRAN instructions to perform time integration of the reactions, in an optimal way with respect to subexpression reutilization.

5.1 Optimal functional mapping of reactive systems

The motivation for the work in this section was the need to parallelize an extensive atmospheric chemistry model to decrease the total execution time of an optimization loop over the model. The model does grid-based calculations and is split into several distinct steps inside the loop for each hour. The framework for the algorithm is shown it Table 6. Execution time for a standard 24 hour run on a single-processor Cray C-90 is about 80 minutes. Almost three quarters of the entire runtime is devoted to calculating the reaction contribution to the changing concentrations at each grid cell, and even though it is a trivial calculation, doing it once for each of the 12 000 grid cells adds up. What "reaction contribution" refers to is just the right-hand sides of the differential equations for time integration of the species concentrations. For example, for the pair of reactions

$$A + B \longrightarrow C$$
$$B + D \longrightarrow E$$

are derived the reaction contributions

$$\dot{A} = -k_1 AB$$
$$\dot{B} = -k_1 AB - k_2 BD$$
$$\dot{C} = k_1 AB$$
$$\dot{D} = -k_2 BD$$
$$\dot{E} = k_2 BD,$$

just a simple set of multiplicated and added expressions of the concentrations and reaction rates (the latter calculated elsewhere).

Initialize	.43%
Loop over simulation time (24 hours)	
Hour initialization	1.48%
Vertical advection	.01%
Integration	
Rate constants	.16%
ODE solver	17.66%
Reaction contribution	71.87%
Sources, diffusive coupling, BCs	6.21%
Horizontal transport	2.17%
Output	.01%
End	

Table 6. Skeleton of the AIRSHED algorithm with fractional execution times.

A typical approach to parallelization would be to split the domain of integration across the available processors, and each machine would do the same set of chemistry, only across a smaller domain. Alternately, one could envision *functional* decomposition, where each processor works on the entire domain, but implements a subset of the chemistry. Of course combinations of the two are possible as well. For the **AIRSHED** problem in particular, data-domain decomposition was easy enough to implement and produced an almost-linear speedup with each additional processor. At some point however, the effects of granularity start to swamp out any potential gain by adding another processor because the domain sizes are so small, or possibly there may be more machines available than grid cells in the domain (in the not too distant future). Then the utility of functional decomposition becomes apparent.

Choosing appropriate partitions of the chemistry is not simple. Due to the coupling of the various chemical species through all the reactions, information about many species must be available at each processor. Eventually each chemical species will be assigned to exactly one of the processors to avoid doing extra work. The schematic in Figure 25 shows the coupling for this set of reactions:

$$\begin{aligned} NO_2 + h\nu &\xrightarrow{k_1} NO + O \\ O &\xrightarrow{k_2} O_3 \\ NO + O_3 &\xrightarrow{k_3} NO_2 \end{aligned}$$

The figure demonstrates that to calculate the change in concentration of NO_2, the processor will need the current concentration of NO_2 as well as those of O_3 and NO, to calculate the increase due to the third reaction. The other species have similar requirements. Any scheme to partition the chemistry across multiple processors will divide this graph, cutting across the arcs which represent the reactions. Each such cut implies one piece of information which must be communicated between the processors at each time step, and thus the cuts are to be minimized.

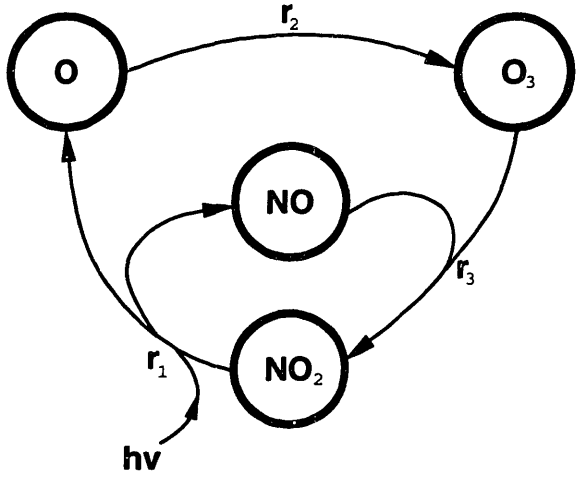


Figure 25. Illustration of tight coupling of chemical species in a simple set of three reactions.

The complete system of **AIRSHED** chemistry is too large to permit the drawing of such a graph, but its complexity may be represented in a different way, by the incidence matrix of the graph. In Figure 26, a blackened square in a row of the matrix indicates that to calculate the species of that row, current information about the species in the corresponding column is required. Methyl *tert*-butyl ether, ethanol, methanol, and others are affected by few reactions so have quite empty rows. Very reactive species, such as OH, are involved in many reactions, and information about them must be known by many of the processors. Two species in this mechanism, TBF and SO_3, never react and are only produced, so their columns are empty. Note that the diagonal is full because

calculating $c(t+\delta t)$ requires $c(t)$ as well as $\left.\frac{dc}{dt}\right|_t$. Another observation about this matrix is that it is non-symmetric and relatively sparse: only about 20% of the entries are non-zero.

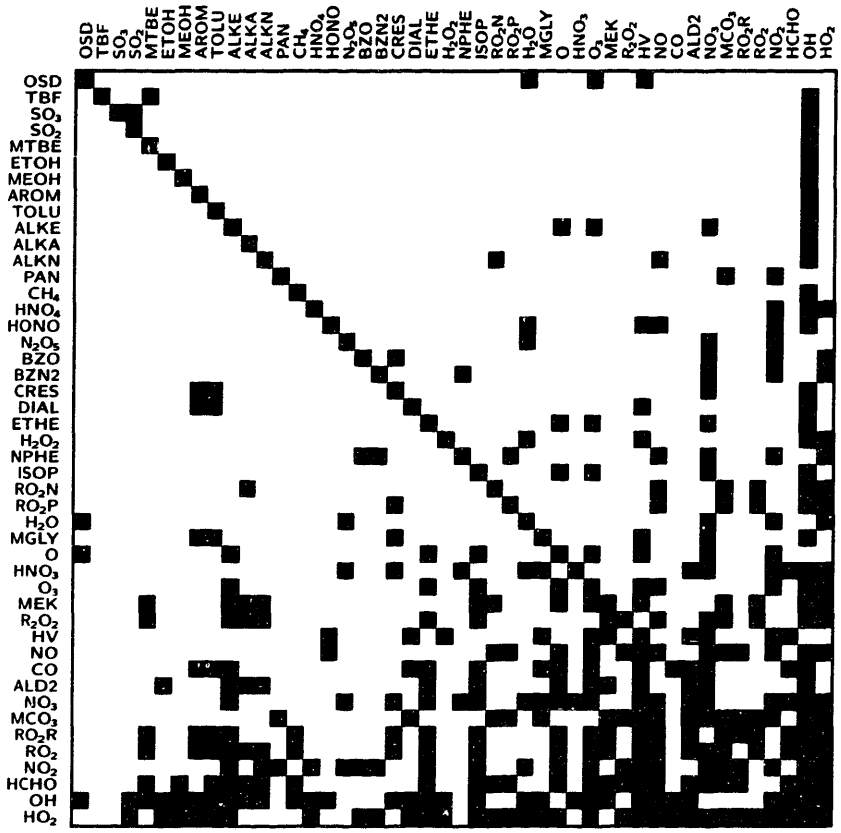


Figure 26. Graphical illustration of coupling between AIRSHED species.

A first attempt at dividing up this system of chemistry would be to rearrange the matrix in Figure 26 into Jordan canonical blocks. However, since OH is implicated in the reactions of 37 species (including itself), there must be at least one block of that size, and the resulting decomposition is guaranteed not to be very even. Placing the problem in a mathematical framework will allow an advantageous comparison to be made. With the definitions

S	set of species participating in the reactions
B	set of all subsets of S
A	incidence matrix of B, $a_{ij} = 1$ if $s_i \in b_j$
P	number of processors
X	optimized variable, $x_j = 1 \iff$ some processor will calculate block b_j,

and constraints

$x_i \in \{0,1\}$	blocks are either calculated or not
$A \cdot X = \mathbf{1}$	every species must be calculated exactly once
$\sum_{i=1}^{\lvert B\rvert} x_i = P$	each processor is allocated one block,

the optimization is to find the vector X such that some function $f(X)$ is minimized. This function will depend on the arrangement of processors to address issues of intercommunication and programming model, but reflects the total time to solve the entire set of chemistry. The length of X is quite large; it is the size of B, or $2^{|S|}$. For the 46 species considered in the **AIRSHED** mechanism, this is 7.0×10^{13} which is a prohibitively large number of variables. With the representation above, the problem is seen to be equivalent to the generalized set-partitioning problem, which is just the set-covering problem with equality constraints, and with weights implemented in the function $f(X)$. Numerous references discuss various forms of the problem [102, 119, 53] and offer good solution techniques for special cases [52], but the problem has been proven to be of the class NP-complete, hence no polynomial time algorithm exists for its solution.

For the purpose of generating the functional form of $f(X)$, the entire **AIRSHED** program may be considered to be divided into a single controlling "front end" and P slave processors. The master processor is responsible for reading inputs and writing output, doing vertical advection and horizontal transport, and general housekeeping while each slave will implement only its portion of the chemistry. Communication is restricted to sending initial concentrations to the slave processors by the master, and returning the results, as illustrated in Figure 27.

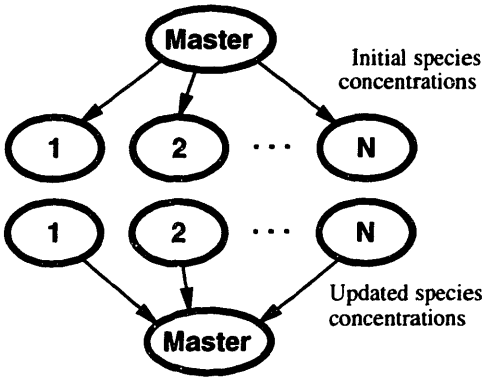


Figure 27. Communication between front end processor and chemistry subtasks.

The main aspect of a parallel machine which will influence the form of f is its type of memory access. There are two basic classes of this. Shared memory machines have essentially constant access time to reference a chunk of memory as access is usually regulated by some piece of hardware which sits between the processors and the memory bus. Common examples of this architecture are the Cray Y-MP and C-90. Since all processors will see the same time delay (which is rather small, on the order of a few hundred cycles perhaps), it can be subtracted out giving

$$f(X) = \max_{\{i \mid x_i = 1\}} T(b_i)$$

where $T(b_i)$ is the time to calculate one block, or one specific subset of the chemical species. For the purposes of the branch and bound algorithm to be discussed later, the obvious lower bound on f will be reported here:

$$f(x) \geq T(b_i) \quad \text{for each} \quad \{i \mid x_i = 1\}.$$

The second major class of memory access occurs when each processor has a chunk of memory associated with it, and for another to access it requires some form of request and reply protocol to have the owning processor send the information out. This is becoming a common architecture for supercomputers, such as the CM-5, and is realized by the virtual machine when using a network of workstations, perhaps under PVM [15]. The access time for any piece of memory depends on the "distance" between the requesting processor and the one which owns the memory, and in the general case may not be known ahead of time. Since this particular problem has restricted the types

of memory access due to the communication protocol illustrated in Figure 27, the value of $f(X)$ can be worked out for each combination by determining the optimal schedule of sending and receiving. An example of a schedule is shown in Figure 28 to illustrate these issues. At the beginning of an iteration, the master processor prepares the initial concentration values and sends them off one set at a time to the slaves, which are idle. As each slave receives the necessary concentrations it starts calculating, independently of all the others, then sends back the results and waits for the next set. Assumed in the figure and throughout this discussion is that the master is able to communicate with only one slave at a time, otherwise the problem reduces to the case of the previous paragraph. There are 2^P different schedules for each partition of the species, and no easy way to decide which is the fastest without testing all the alternatives. In fact, the scheduling problem described here is another well-known optimization problem and is also of class NP-complete. However for the purpose of the upper-level optimizer, there are the bounds

$$\min_{\{i \mid x_i=1\}} T(b_i) - \sum_{\{i \mid x_i=1\}} S(b_i) + R(b_i) \;\le\; f(X) \;\le\; \max_{\{i \mid x_i=1\}} T(b_i) - \sum_{\{i \mid x_i=1\}} S(b_i) + R(b_i),$$

where S and R give the times to send and receive a block of species. One example of the relative widths of the bars in Figure 28 is given by timing obtained using PVM between DEC workstations across a single 10baseT network at MIT, where the send time for one species is 0.204 seconds, and the calculation time is 1.003, on average. For more well-connected processors, it may be that the calculation times overwhelm the sending and receiving times in which case a shared-memory approximation may be made. The rest of this section will not address the more complex situation, and solve the problem only for shared memory architectures. On non-dedicated machines, exact scheduling is further complicated by the runtime decisions of the operating system of which task actually executes.

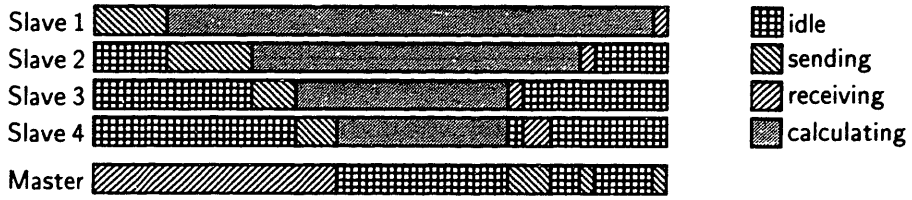


Figure 28. Example communication schedule for four slave processors.

The time to calculate each species is the sum of the times to calculate the extent of each reaction in which that species participates. The time to calculate a single block, $T(b_i)$, in general would be calculated by performing the calculation for each species and summing over the species in the block using the incidence matrix A, then subtracting out reactions common to the block. The reactions in the AIRSHED chemical mechanism are all uni- or bi-molecular and first order in the reacting species, so with the observation that array accesses to get current concentrations are much slower than floating point calculations, all the reactions take about the same amount of time. This constant factors out and makes the algorithm a bit simpler by removing the need to keep a table of the reaction times.

Even with these simplifications, the problem as it stands has 46 species, so 7×10^{13} variables, which leaves it essentially unsolvable. One reduction, which is common to both cases of memory access, uses the tight coupling to reduce the effective number of species. The set of species S can be split into two groups: $\overline{S}$, the "essential" species, and $\underline{S}$, the "covered" species, so that

$$\underline{S} = \left\{ s_i \in S \mid \exists\, s_j \in \overline{S} \text{ such that } N(s_i) \subseteq N(s_j) \right\}.$$

N depends on the memory type. For the shared memory case, $N(s_i)$ is the set of reactions which affects the species s_i, while for exclusive memory machines it is necessary to add to N the set of

species which participate in those reactions, as their concentrations must be present at the working processor. This augmenting set can be read off the appropriate row of the matrix in Figure 26. An example of one covered species is given in Table 7 where the 17 reactions involving lumped aldehydes are listed, and are seen to include all four of the reactions involving 3-carbon and longer alkenes. So for a processor which has already been assigned to calculate aldehydes (as some processor must), it makes no sense to assign alkenes to any other processor as the work to calculate the update has already been performed as part of calculating the update to aldehydes. Applying this simplification reduces the number of essential species (those which are not covered in the calculation of any other) down to 31, resulting in a problem of 2×10^9 variables, still a huge number.

	ALD + OH	$\longrightarrow MCO_3$
	ALD + HV	$\longrightarrow CO + HCHO + RO_2R + HO_2 + RO_2$
	ALD + NO_3	$\longrightarrow HNO_3 + MCO_3$
	MEK + HV	$\longrightarrow MCO_3 + ALD + RO_2R + RO_2$
	MEK + OH	$\longrightarrow 1.2\ R_2O_2 + 1.2\ RO_2 + MCO_3 + .5\ ALD + 0.5\ HCHO$
	ALKA + OH	$\longrightarrow b_1\ HCHO + b_2\ ALD + b_3\ MEK + b_4\ RO_2N + b_6\ R_2O_2 + b_7\ RO_2$
	ALKN + OH	$\longrightarrow NO_2 + 0.15\ MEK + 1.53\ ALD + 0.16\ HCHO + 1.39\ R_2O_2 + 1.39\ RO_2$
	ETHE + OH	$\longrightarrow RO_2R + RO_2 + 1.56\ HCHO + 0.22\ ALD$
*	ALKE + OH	$\longrightarrow RO_2 + RO_2 + b_8\ HCHO + b_9\ ALD$
*	ALKE + O_3	$\longrightarrow b_{10}\ HCHO + b_{11}\ ALD + b_{12}\ RO_2R + b_{12}\ RO_2 + b_{13}\ HO_2 + b_{14}\ OH + b_{15}\ CO$
*	ALKE + O	$\longrightarrow b_{16}\ CO + b_{17}\ MEK + b_{18}\ HCHO + b_{19}\ ALD + b_{20}\ HO_2 + b_{21}\ RO_2R + b_{21}\ RO_2$
*	ALKE + NO_3	$\longrightarrow NO_2 + b_8\ HCHO + b_9\ ALD + R_2O_2 + RO_2$
	ISOP + OH	$\longrightarrow HCHO + ALD + RO_2R + RO_2$
	ISOP + O_3	$\longrightarrow 0.5\ HCHO + 0.65\ ALD + 0.21\ MEK + 0.16\ HO_2 + 0.29\ CO + 0.06\ OH + 0.14\ RO_2R + 0.14\ RO_2$
	ISOP + O	$\longrightarrow 0.4\ HO_2 + 0.5\ MEK + 0.5\ ALD$
	ISOP + NO_3	$\longrightarrow NO_2 + HCHO + ALD + R_2O_2 + RO_2$
	ETOH + OH	$\longrightarrow ALD + HO_2$

Table 7. Reactions required to calculate changes in concentration of lumped aldehyde (ALD). Those marked with an asterisk (*) are the full set of reactions needed to calculate 3-carbon and longer alkenes.

In an attempt to reduce the size of this optimization problem, heuristic initialization is used to prune away some of the branches at the outset. A good low value for total time to execute was found essentially by guessing, although in an algorithmic way, in a program that took two hours to complete, which permits the exclusion of all but 6.5×10^6 blocks. The code to implement this integer programming problem does a standard branch and bound search but with quite a few modifications. Even with this great reduction in the number of variables, it was clear from running an initial version that it would not complete for many years. Many purely computational improvements were brought to bear, the main one using the fact that 31 species could be represented as bits in a standard memory word (32 bits), which allows replacing array indexing and addition with bit shifts and masks. Also arrays of times for words containing three or fewer '1' bits were pre-generated and stored on disk to speed up timing calculations; the same was done for pair lists of manageable sizes. Counting the number of '1' bits in a word, a frequently performed operation, was replaced with looking up the word in a table, replacing a loop of length 32 over 3 operations with a single array reference. Furthermore the most critical routine was hand-coded in assembly language, for about a factor of four speedup.

It must be stressed that this whole procedure needs to be performed only one time once the choice of chemistry is made, and is not part of the production system which will do the actual calculations of changing concentrations. It is reasonable to spend a fair amount of time up front

if it will be gained back by decreasing the runtime of a calculation to be executed thousands of times later. The optimization for the AIRSHED system of chemistry took five assorted machines (one DEC MIPS 5000/240, two DEC Alphas, an SGI, and an HP) about two weeks working full time to solve the problem for the case of four processors. The execution times for partitioning the chemistry across four and eight processors are given relative to the time to solve all the chemistry on a single processor in Table 8. For four processors, the speedup is only a factor of two since the procedure is limited by one large block. In the case of eight processors, the time to calculate the entire set of chemistry is again limited by the largest chunk, which requires a third of the total single-processor time. In spite of the apparent poor performance gain, the figures reported in the table are indeed optimal for the chosen parallelization strategy and set of chemical reactions.

processors	execution time
1	1.00
4	0.52 = max(0.52, 0.37, 0.30, 0.30)
8	0.33 = max(0.33, ..., 0.07)

Table 8. Relative execution times of functional parallelization of chemistry, for different numbers of processors.

The conclusion one may reach is that this type of parallelism should never be employed due to the extremely sublinear scaling of calculation time as a function of number of processors, and one would be correct. This problem is quite amenable to the more standard technique of data parallelization, where a chunk of the total spatial domain is given to each processor, and each processor implements the full set of chemistry (the full "function") on each grid cell in its assigned chunk. For the case where the number of processors is much smaller than the number of grid cells, data parallelization is the obvious choice. When, as may be seen in current trends, the number of processors becomes comparable to or exceeds the number of grid cells in a problem, two possibilities may happen. First, as is often relied on by parallel algorithm designers [142], programmers may increase the resolution of their codes until the cell-to-processor ratio becomes large again. Failing that, it becomes necessary to use functional parallelization if the full power of the machine is to be exploited, and the method described here is the optimal way to do that.

One other advantage of this method of parallelization which is simply mentioned is in the reduction of per-processor algorithmic complexity. The extent of the more than 100 reactions involved in the example described here is too large to allow use of the vectorization registers on the Cray C-90, however the smaller sets of chemistry generated by this procedure may allow their utilization, realizing an unexpected benefit of functional parallelization.

5.2 Subexpression superoptimizer

The second use of a representation tailored to fit the problem is also related to speeding up the right-hand side calculations described in the previous section. This time there is no consideration of parallelization, just a maximization of the raw scalar speed through optimal preservation of subexpressions. As an example, consider (once again) the reaction

$$A + B \longrightarrow C.$$

The differential equations for the three species are, trivially,

$$\frac{dA}{dt} = -kAB$$
$$\frac{dB}{dt} = -kAB$$
$$\frac{dC}{dt} = kAB.$$

Inside any time integration package, there will eventually be statements which compute the right-hand sides of each of these equations, to be used in updating current concentration values according to whatever time integration scheme is used. This code may have the following form:

```
Adot = -k * A * B;
Bdot = -k * A * B;
Cdot =  k * A * B;
```

A naïve code compiler might produce machine instructions which resemble the following:

```
load  $f1, k
neg   $f1
load  $f2, A
mul   $f3, $f1, $f2
load  $f4, B
mul   $f5, $f3, $f4
store $f5, Adot

load  $f1, k
neg   $f1
load  $f2, A
mul   $f3, $f1, $f2
load  $f4, B
mul   $f5, $f3, $f4
store $f5, Bdot

load  $f1, k
load  $f2, A
mul   $f3, $f1, $f2
load  $f4, B
mul   $f5, $f3, $f4
store $f5, Cdot
```

where three nearly-identical sections are needed to produce the results. A more clever compiler might notice that the expression $k * A * B$ is used frequently, and preserve it for many calculations:

```
load  $f1, k
load  $f2, A
mul   $f3, $f1, $f2
load  $f4, B
mul   $f5, $f3, $f4
store $f5, Cdot
neg   $f5
store $f6, Adot
store $f6, Bdot
```

Since the number of loads and stores is absolutely fixed by the structure of the problem, it is clear that the second code listing calculates the results with the minimum number of operations. The goal of this section is to do the same analysis, only for much larger systems of chemistry.

There are many levels of optimization available to a code writer, be it a compiler or a person hand-crafting the instructions. The analysis above stresses the "top" level, that of re-ordering operations to minimize redundant work. Optimizing at deeper levels requires specific knowledge about operating characteristics of the machine. For instance the choice between storing a subexpression in memory for later use then retrieving it when needed, or simply recalculating the subexpression depends on many factors: number of times the subexpression is reused, speed of accessing memory, time to calculate the subexpression, and presence of other stored subexpressions. The last item

is complicated by the existence of different levels of memory in most machines: registers, on-chip cache, off-chip cache, main memory, swap. And the optimal code under one operating condition for the machine may not be optimal when other processes are competing for machine resources as well. Thus the philosophy taken here is that it may be worth the effort to superoptimize this very special structure of operations (namely that generated from mass-action chemistry rate equations), but that there is no value in trying to complete the compiler's tasks of register allocation, instruction reordering, and pipelining.

The code which does this analysis is based on a common reaction mechanism parsing framework, which is discussed elsewhere in this thesis and fully described in the appendices. Taking the lists of reactions, species, and constants produced by the parser, the optimization code performs the following steps:

1. **Build node trees.** Construct reaction rate expressions from the left hand sides of the reactions and express for each species its combined sum of reactions, with pre-multiplying stoichiometric coefficients. This creates a forest of trees, one for each species, which share links only at reaction rate nodes.
2. **Combine multiplications.** Further interlink the forest by sharing all multiplication nodes at the highest level. This is done by first creating a linear list of all multiplication nodes in any tree, sorting the list according to a canonical strict ordering on nodes, and linking all identical nodes to a single reference. A further sort places the shared multiplication nodes in most-used order.
3. **Combine additions.** Analogous to multiplication, addition nodes are sorted in most-used order and shared.
4. **Output.** A `FORTRAN` subroutine is created by dumping all the rate expressions, with temporary expressions declared only just before their use, hopefully to help the compiler in making wise register allocations.

One complexity which need not be handled is that of additions inside multiplication expressions because the special structure of reaction rates does not allow that situation to occur. Otherwise the above four steps would be complicated by the choice of when to distribute multiplication across addition and when not to do so.

The results for the `AIRSHED` chemistry of 44 species, 9 of which are taken to be in steady-state conditions, and 106 reactions are presented in Table 9. The base for computing the fractions is the number of operations in the original source code for calculating the species updates, from which the number of times and plus signs were simply tallied up. The original author was clever enough to save some operations over a "naïve" code by first computing the extent of each reaction, then summing up products of stoichiometric coefficients and those extents; the naïve code simply recalculates the extents of reaction at each point needed.

	multiplication		addition		total	
	count	fraction	count	fraction	count	fraction
Naïve code	721	1.325	340	1.046	1061	1.221
Source code	544	1.000	325	1.000	869	1.000
MIPS compiler	423	0.778	329	1.012	752	0.865
CRAY compiler	415	0.763	342	1.052	757	0.871
Optimized	335	0.616	286	0.880	621	0.714

Table 9. Operation counts for various compilation schemes.

The counts reported in the third and fourth rows of the table were produced by a DEC MIPS fortran compiler, and by CRAY's `cf77` compiler, both set to use the highest level of optimization available. The assembly code outputs were searched through, counting up the floating point operations, to generate the numbers in the table. The final row is the result of the super-optimizer

described above. Both operation counts are clearly much lower, and since the two machines considered here both have constant operation time for addition and multiplication in this code, the subroutine takes only about 82% of the time to execute compared to the default optimized versions. One reassuring result is that the compiler is unable to optimize away any more instructions once the superoptimizing compiler has done its work. That is, the operation counts reported in the last row of Table 9 are computed by counting up *'s and +'s in `FORTRAN` source code, and are identical to those that would be calculated by counting `add` and `mul` instructions in the object code produced by the MIPS or CRAY compilers using full optimization.

Here again, then, is stressed the important theme of choosing the proper representation for a given problem. The superoptimizer does its work not on raw source code or even on the reaction equations themselves, but rather transforms the reactions into a forest of species-rooted trees which are then intertwined using natural binary tree operations. Intermediate steps require sorting of the expressions, for which the more natural linear list representation is derived, and the results of sorts are returned to their proper locations in the binary tree. So five major representations were employed on this single problem:

1. Reaction equations, the "natural" language for human chemists and engineers.
2. Reaction-rooted binary trees, straightforward structures into which to parse reaction equations.
3. Species-rooted binary trees, necessary basis for sharing nodes, and the order in which output must be generated.
4. Linear lists, for intermediate sorting steps.
5. `FORTRAN` source code, as required for machine translation.

All the operations performed on these structures are simple; the thought goes into deciding when to change representations, and performing the transformations.

The concept of representation discussed in this chapter is the cornerstone of the development of numerical differential equation solving algorithms by finite differences or finite elements. A functional form is chosen to represent the dependent variables, and the problem is reformulated in that new space. Later chapters discuss this in more detail, and in particular, for the case of non-analytic functions.

Part II

Optimal Field Representation

The following three chapters consider the search for the optimal representation for a given problem. In particular, for the case of differential equation solving, it is very simple now to take an off-the-shelf integration package, write a derivatives (and perhaps Jacobian) subroutine, and let it produce the answer. This is a fine approach for small problems or even big problems which will not need to be solved more than a few times, as these schemes guarantee error control to a user-specified tolerance and in general have been thoroughly tested and debugged. In using these packaged solution modules, one accepts the representation prescribed by the author and must tailor his problem to fit that representation which may not be the obvious or best one for the particular problem.

Programmers of large complex codes with significant differential equation solving costs spend a fair amount of time fine tuning the solution algorithm to the application. Global weather prediction codes generally use spectral methods, the clear choice when the domain is a sphere, and use many assumptions which are quite valid for atmospheric conditions. Ocean flow models must use a very different set of assumptions, given the different fluid properties and the presence of horizontal boundaries. The regional scale chemical species reaction and transport model, AIRSHED, uses a hybrid integration technique with seven numerical constants to control stepsize selection, tailored specifically to the problem and the domain.

What is suggested in this thesis is that this degree of tuning the solution strategy to the problem at hand should go one step further. By using all the information available about the equations, including likely solutions, it will be shown that a scheme can be developed which is the optimal method of solution. This scheme depends explicitly on the assumption that the code is intended to be executed many times under similar conditions to provide a good set of example fields from which to create the basis. The scheme is based on a Galërkin method using empirically derived basis functions, generated by a Karhunen-Loève expansion.

Different approaches to differential equation solving are first presented, followed by a general formulation of a solution technique from which can be derived finite elements, finite differences, and spectral methods. The last chapter derives the optimal basis for a given set of previous solutions.

6 Differential equation solving: previous approaches

Departures from standard solution strategies based on finite differences or finite elements are uncommon in practical use. The following sections describe some of the more popular methods of reducing differential equation solution time.

6.1 Parameterization

The preeminent strategy used to speed up the solution of a model is to reduce its complexity—adjust the physics to ease the strain on the numerics. This is commonly done by an outright neglect of certain effects, such as the neglect of non-Newtonian effects, entrance and exit conditions, and compressibility in the use of the Hagen-Poiseuille law to relate volumetric fluid flow to pressure drop. Weather prediction models frequently ignore the dynamics of the interaction of the atmosphere with land and sea, such as heating and moisture changes, and assume that short-wave solar radiation is not absorbed by clouds. Recent studies are suggesting both of these assumptions are not valid.

When the physics is too important to be omitted, yet too complex to model or solve, parameterized approximations can be substituted for the actual effects. The modeling of turbulent fluid flow is a classic example of this, with the Blasius formula

$$f = \frac{0.0791}{\mathrm{Re}^{1/4}}$$

which gives decent estimates of the friction factor for flow in tubes for Reynolds number in the range 10^3–10^5. Different functional dependencies arise for flow past spheres or over flat plates, all derived by fitting experimental data to plausible functional forms. In models of cloud formation used in weather prediction, the microscale physics of particle coagulation and water vapor condensation is commonly parameterized with values which give believable results. The effects of different aerosol properties is ignored.

Models featuring reactive chemistry make approximations by omitting certain reactions, or assuming some reactions are so fast as to be always at steady state (ignoring the kinetics), or lumping many similar chemical species into one aggregate species. The concentration of this aggregate species is adjusted based on fractional instead of mass action kinetics, with the fractions adjusted so that the answer looks correct, or based on an experimental study.

Molecular dynamics codes attempt to predict properties of single molecules or a small collection by using force laws based on the individual atoms. One popular assumption is that since these forces fall off with increasing interatomic distance, not all pairs of atoms require force calculation, just those proximal to each other. Also the force laws are not explicitly known, but potentials such as the Lennard-Jones provide good approximations, it seems.

In many cases these approximations or parameterizations are entirely appropriate, and it would be senseless to include the calculation of a small effect which will have zero contribution to the final result, or whose contribution falls within the uncertainty of the result. In other cases the physics is clearly being held back by the inability to perform the calculations quickly enough, and the onus falls upon the numerical algorithms.

The true goal of modeling is to find the best trade off between model complexity and solution feasibility, bounded between a useless result when the model is too simple, and an uncalculable result when it is too complex. The dream, however, is that solving the model is a trivial detail, providing for complete freedom in prescribing the physics. (If this were the case, all models would probably be atomic scale.)

6.2 Multigrid and adaptive grid methods

Often to generate sufficiently accurate solutions to differential equations over a spatial domain, it is necessary to increase the resolution in a grid-based method to such an extent that the problem becomes computationally intractable. One remedy to this problem is to use multiple grids at different resolution levels, or a grid which changes dynamically during the course of an integration. Both these techniques still rely on a grid-based solution method, just modify it such that fine resolution is used only where necessary. In the case of multigrid methods, Bank [12] has implemented a solution strategy for elliptic equations with good results. Berger [16, 17] uses adaptive grids to solve hyperbolic problems occurring in shock hydrodynamics to solve previously intractable problems.

While this approach seems like the obvious thing to do, there are some associated problems. Code complexity is increased due to the added overhead of checking some metric to decide if the grid should be locally refined, in adaptive grid schemes. Multigrid algorithms usually prescribe the locations of the extra grids before runtime, locking their use to specific conditions (*e.g.*, a non-moving shock front). Conceptually the worst problem is deciding how to transfer information between the multiple grids or along the boundaries of refined adaptive grids. Linear interpolation and extrapolation is commonly used but has no physical basis. Biswas *et al.* [20] discuss variations on an adaptive finite element mesh which allow the conservation of various components which would otherwise not be conserved across linear interpolation between grids.

Implementation of dynamic adaptive grid methods on parallel machines suffers in that the work load for any given section of the grid changes during the simulation. Processor loads may initially be balanced, but quickly change to the case where processors calculating a quiescent region become idle while those working near shock fronts keep the rest of the system from advancing to the next time step. Ideally load should be balanced dynamically, as Wheat *et al.* [141] attempt. Unfortunately the overhead for this sort of load balancing often is too expensive.

6.3 Wavelet finite elements

Another promising multiresolution algorithm is presented by Bacry *et al.* [10] who use wavelets as the orthonormal bases in a finite element scheme. Wavelets have the nice properties of scaling and translation invariance, allowing for easy grid resolution, and locality. In their algorithm, multiple sets of finite element grids are maintained, ranging from spatially large elements covering the whole domain down to very small locally supported elements only where the error from the larger ones is significant. All the layers are superimposed to give the actual value at a point. Adaption in time works by checking the error at each point, and adding the next smaller (by a factor of two

always, from the definition) layer of wavelets in the region of that point. Conversely when small-scale wavelets are contributing little to the solution, they can be removed from the calculated set. A similar approach is used by Engquist [43].

This approach seems quite promising for partial differential equations which generate solutions with localized structures in space that evolve quickly in time, such as flows with moving shock fronts or the nonlinear Schrödinger equation. However the numerical complexity grows quicker than would be expected as it is linear in the number of nonzero wavelet coefficients (not scaled down by the extent of the wavelet), and the constants are not small [18, 84]. Another problem is that the boundary conditions must be described on cubes in $\mathbb{R}^n$; arbitrary domains must somehow be mapped into cubes, or approximated.

6.4 Faster machines

Finally, the method most relied upon for decreasing model execution time is to port the code to a faster machine. In doing an optimization over the emission fields using `AIRSHED` as the model to calculate pollutant concentrations, it was found that a runtime of 5 hours per function evaluation would make the problem completely infeasible. Porting the code to the Cray C-90 reduced this time by a factor of 60 which allowed the optimization to converge in only a few days.

The algorithmic changes involved in using a faster machine are often trivial, but in the optimization just described and many other models, some rudimentary data parallelization must be performed. This technique of speeding up a code must certainly be considered for working models, but provides no fundamental improvement in the field of numerical solution of differential equations.

7 Galërkin methods

Particular computational methods have assumed prominent positions in certain areas of applications, such as finite element methods in structural problems, finite difference methods in flow around aircraft wings, and spectral methods for global atmosphere modeling and weather prediction. These apparently unrelated classes of algorithms are actually all specializations of one general method, with the artificial division arising just due to the segregation into different modeling arenas.

Solution techniques in this general class are called Galërkin methods. Problems consisting of ordinary or partial differential equations and integral equations have been simplified and solved using Galërkin formulations for many years. The capstone paper was published by Galërkin [51] in 1915 on the elastic equilibrium of rods, and gained attention in the western world in papers by Duncan [40, 41]. The attractiveness of such methods at that time when calculations were done by hand is that they could provide significant accuracy with minimal manual effort. Widespread availability of computers led to more emphasis on Galërkin methods, as solutions of greater accuracy with minimal execution time could be achieved.

A good introduction including connections to other numerical equation-solving methods through appropriate choices of the expansion functions and the test functions can be found in a book by Fletcher [48]. One special point to notice that is often used in developing a Galërkin method is that the expansion fields are deliberately chosen to satisfy the boundary conditions. The work in this thesis takes that observation one step further and also requires that the expansion fields be likely candidates of the actual problem solution. The use of non-analytic functions does not alter the Galërkin nature of this procedure and is consistent with the historical trend to use the advantage of computational power to simplify total problem solution times, reducing the analytic component of the solution process.

7.1 Background

A wide class of solution techniques, including finite differences, finite elements, and spectral methods, are derivable using the Galërkin method framework. The key features of a Galërkin method are a differential equation

$$Lu = 0$$

in a domain D with boundary conditions

$$Su = 0$$

on ∂D. Now the Galërkin method assumes that the solution u can be well approximated by a series of known functions $\{\phi_i\}_{i=1}^N$:

$$u = u_0(\mathbf{x}) + \sum_{i=1}^{N} a_i \phi_i(\mathbf{x}).$$

Normally the ϕ_i are taken to be analytic functions, but that need not be the case, as will be shown by the empirical nature of the ones exploited in this thesis. Frequently the trial functions, ϕ_i, are deliberately chosen to satisfy the boundary conditions. Substituting the expansion for u into the linear governing equation produces a nonzero residual given by

$$R = Lu = Lu_0 + \sum_{i=1}^{N} a_i L(\phi_i).$$

In the Galërkin formulation, one demands that the residual be orthogonal to the same functions used in the expansion which allows the coefficients a_i to be determined from a matrix equation.

The leading term u_0 is included to satisfy the boundary conditions and in general can be subtracted off the dependent variable u to produce a problem with homogeneous boundary conditions. Otherwise the boundary condition must be handled through the trial functions which effectively uses up a basis member. Handling the boundary conditions in this way allows for local error reduction (on ∂D) while the whole Galërkin procedure is constructed to achieve global error reduction.

7.2 Method of weighted residuals

An extension to Galërkin methods, which encompasses them, is sometimes called the method of weighted residuals [14]. In this general case, the residual is forced to be orthogonal to some other class of functions, not necessarily those used in the expansion. The general formulation comes from Collatz [34] in which one attempts to find a solution u which is close to the exact solution u_e:

$$u_e \simeq u(x_1, \ldots, x_n, a_1, \ldots, a_p)$$

by requiring one of the three conditions:

1. The differential equations are satisfied exactly ("boundary method").
2. The boundary conditions are satisfied exactly ("interior method").
3. Neither the differential equations nor the boundary conditions are satisfied exactly ("mixed method").

Then parameters $a_1, \ldots, a_p$ are chosen such that, respectively,

1. The error in satisfying the boundary conditions is minimized.
2. The error in satisfying the differential equations is minimized.
3. Errors in satisfying both the differential equations and the boundary conditions are simultaneously minimized.

Solution techniques of ordinary differential equations mostly use interior methods since if the solution to the differential equations is not known exactly, it is still necessary to solve a nonlinear system of equations to get the boundary conditions. Partial differential equation solution techniques use both boundary and interior methods, but boundary methods are to be preferred since integration along the boundary is computationally less expensive than integration across the entire domain. Mixed methods are used when both the differential equations and boundary conditions are especially complex.

Considering just interior methods, the method of weighted residuals starts in the same manner as the Galërkin method, expressing an approximate solution in terms of known analytic trial functions,

$$u(\mathbf{x}, t) = u_0(\mathbf{x}, t) + \sum_{i=1}^{N} a_i(t) \phi_i(\mathbf{x}),$$

and again u_0 is chosen to satisfy the boundary conditions if possible. This formulation reduces partial differential equations in $\mathbf{x}$ and t to ordinary differential equations in t. If instead the functions $\phi_i(t)$ and $a_i(\mathbf{x})$ are used, it reduces to a partial differential equation in $\mathbf{x}$. For trial functions $\phi(\mathbf{x}, t)$ of both space and time, or for non-time varying problems, this expansion reduces the equations to algebraic. Where the method of weighted residuals is more general than the Galërkin method is that the test functions need not be the same as the trial functions, so the requirement

$$\langle R, w_i(\mathbf{x}) \rangle = 0, \qquad i = 1, \ldots, N$$

is used to generate equations for the coefficients a_i. In order to produce N independent equations, the test functions w_i must be independent (orthogonal), but that is the only constraint.

Some common choices for the weight functions have developed.

1. Subdomain. The domain D is divided into possibly overlapping chunks and the test functions are the characteristic functions on the division:

   $$w_i(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in D_i \\ 0 & \text{otherwise.} \end{cases}$$

   This method is similar to the finite volume concept in fluid mechanics and heat transfer, which uses the concept of a control volume to calculate, for instance, accumulation of a passive scalar:

   $$\int_S \rho u_n \, ds = -\frac{\partial}{\partial t} \int_V \rho \, dv$$

   as the flux through the surface of the volume. By Green's theorem, this becomes

   $$\int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) \right) \, dv = 0.$$

   Each control volume is, then, its own subdomain.
2. Collocation. The choice

   $$w_i(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_i)$$

   is made by most finite difference models, forcing agreement of the equations at a finite number of points inside the domain. Extremal point collocation and orthogonal collocation sample values at the zeroes of polynomials, as discussed at length in Finlayson [46], in which examples are chosen from common chemical engineering problems. Using the collocation method, Schetz [120] solved three different viscous flow problems using as trial functions the analytic solutions from related problems. For instance, the momentum-only equation for flow across a semi-infinite flat plate was approximated using solutions from the case for an infinite plate:

   $$u_\infty \operatorname{erf}\left(\frac{y}{2a_1 x} \mathrm{Re}_x^{1/2} \right), \quad u_\infty \operatorname{erf}\left(\frac{y}{2a_2 x^2} \mathrm{Re}_x^{1/2} \right), \quad u_\infty \operatorname{erf}\left(\frac{y}{2a_3 x^3} \mathrm{Re}_x^{1/2} \right), \quad \ldots$$

   essentially parameterizing the solution using a polynomial expansion. The other two problems considered were natural convection near a vertical flat plate and forced convection over a plate with a step change in temperature. This work is relevant since it seems to be the first instance of choosing the trial functions not only to fit the boundary conditions, but also to use information about the expected solution, namely that it be close to the solution of a related problem.
3. Least squares. Writing

   $$w_i(\mathbf{x}) = \frac{\partial R}{\partial a_i}$$

where a_i are the coefficients in the expansion, is equivalent to requiring that

$$\langle R, R \rangle$$

is minimized. This is perhaps the most popular method, especially for steady problems. For unsteady problems, the inner product must be augmented to consider both the spatial and temporal domains.

4. Method of moments. The test functions

$$w_i(x) = x^i$$

have some appealing statistical properties.

5. Galërkin method. As already discussed, the test functions are exactly the trial functions

$$w_i(\mathbf{x}) = \phi_i(\mathbf{x})$$

although in this case the ϕ need not be orthogonal.

6. Generalized Galërkin method. The test functions are approximately the trial functions, but extra terms may be added to improve numerical conditioning and stability.

A variation on the method of weighted residuals, called the discrete method of weighted residuals, uses an inner product such as

$$\langle f, g \rangle = \sum_i f_i g_i$$

where all the variables are discrete and the underlying equations are finite difference equations, either ordinary or partial. Neuman [103] gives an overview of the uses of this method, using a "modal" expansion which is still based on underlying analytic functions, in spite of the inherent discrete nature.

7.3 Variational formulation

The general problem $Au - f = 0$ can be considered in a variational framework by noticing that this equation is the Euler equation of the minimization problem

$$Fu = \int_X (Au - 2f)u \, d\mathbf{x}.$$

The method of Rayleigh-Ritz seeks to minimize such an integral by choosing a family of functions

$$u(\mathbf{x}) = \phi(\mathbf{x}, a_1, \ldots, a_n)$$

which satisfy the given boundary conditions, then substituting them into the integral and finding the values of a_i for which

$$\frac{\partial F(u)}{\partial a_i} = 0, \qquad i = 1, \ldots, n.$$

In the cases most important practically [68], ϕ is taken to be a polynomial which is linear in the coefficients a_i, so may be written

$$u(\mathbf{x}) = \sum_{i=1}^{n} a_i \phi_i(\mathbf{x})$$

which gives (using orthogonality of the ϕ_i)

$$F(u) = \sum_i \left(\int_X a_i^2 A\phi_i(\mathbf{x})\phi_i(\mathbf{x})\, d\mathbf{x} + \sum_{j \neq i} \int_X 2a_i a_j (A\phi_j - 2f)\phi_i\, d\mathbf{x} - \int_X 2a_i \phi_i f\, d\mathbf{x} \right)$$

and setting the derivatives with respect to a_i equal to zero gives

$$0 = 2\sum_{j=1}^{n} \int_X a_j A\phi_j \phi_i\, d\mathbf{x} - 2\int_X \phi_i f\, d\mathbf{x}, \quad i = 1, \ldots, n$$

or more succinctly

$$\sum_{j=1}^{n} a_j \langle A\phi_j, \phi_i \rangle = \langle \phi_i, f \rangle, \quad i = 1, \ldots, n$$

which is the same result as would have been obtained using a Galërkin formulation. This connection to the Rayleigh-Ritz method allows the strong bounds on convergence given in Kantorovich [68] to be applied to the Galërkin method as well. In general, convergence estimates must be written on a per-problem basis as the structure of the exact solution is not known. The traditional error metric,

$$\|u - u_e\|,$$

where u_e is the exact solution, is not generally available. However the value of the residual is always easily obtained, and can be related to the exact error.

7.4 Choice of basis

The trial functions used in a Galërkin expansion are critical in determining the success of the method, in terms of rate of error reduction and calculation time required. Certain choices of trial functions are popular enough to have developed their own names as "methods," specializing from the general class of Galërkin methods, as in the finite difference method, finite element method, and spectral method discussed below.

First, however, the use of orthogonal trial functions is seen in many cases since this property allows for easy calculations. In particular, a derivation using an arbitrary set of trial functions leads to a system of equations which must be solved simultaneously to obtain values for all the coefficients, and increasing the number of trial functions requires another complete solution of the (now larger) set of coefficients. For the case of orthogonal functions, the set of equations are independent and can be solved one at a time. Adding another orthogonal element to the basis requires only a single derivation and the solution of only one more equation since the values of previously solved coefficients do not change. The magnitudes of higher-order coefficients give an indication of the current error in the solution. Various uses of these orthogonality properties are given below.

7.4.1 Finite elements

The use of piecewise linear test functions of local support is the basis of the finite element method. Here the coefficients of each finite element are calculated in an inherently local way, but the global error is used to drive the values. Reduction of the system of equations by a Galërkin method produces a linear system of the form

$$LA = MA$$

where A is the matrix of coefficient values, one for each test function, and L and M are in general fully populated due to the non-zero inner products of the test functions. In the finite element case, for example using the element shown in Figure 29, each element has only two neighbors, leading to

L and M matrices which are non-zero only on the diagonal and elements adjacent to the diagonal. This sparse banded nature allows for very fast solutions of the linear set of equations, but the local nature of the test functions generally leads to less accuracy than would functions of global support.

Most of the finite elements are orthogonal simply due to the fact that they are compactly supported on different non-intersecting domains, still leading to a full system of equations, but of a special structure which can be easily solved. Various choices of the finite elements will lead to different degrees of overlap, of course. Here a tradeoff is made between the ease of solution that full orthogonality provides and global error control that complete non-orthogonality gives.

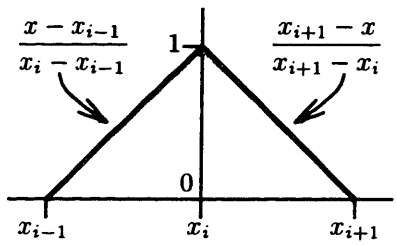


Figure 29. Piecewise linear finite element basis function.

7.4.2 Spectral methods

In contrast with finite elements which rely on the low-order, local nature of the trial functions, spectral methods exploit global, orthogonal trial functions, trading off ease of computation for rapid convergence. In spectral methods the choice of the functions is of utmost importance. Some of the most popular sets of functions are now described.

1. An eigenfunction basis can be formed by calculating the eigenfunctions of a related evolution operator, then the idea of small perturbations and relying on the fact that the structure of the related problem is similar allows its use in solution of the problem at hand.
2. A Fourier transform expands an operator in terms of the trigonometric functions sine and cosine and is well-suited for problems with periodic boundary conditions. This basis has the nice property of infinite differentiability as well, and has been shown to exhibit the best convergence rate for smooth functions, but performs very poorly near discontinuities (including at non-periodic boundaries).
3. A Taylor series of the monomials x^n for n increasing from zero does not see wide use in practice due to the wild oscillations which occur when trying to fit functions using a large number of basis elements.
4. Legendre polynomials fix this problem to some degree and offer good wavelength resolution as well, and are non-periodic, but converge very slowly near discontinuity boundaries.
5. Chebyshev polynomials are perhaps the most robust available in terms of being able to fit most functions, and exhibit good convergence rates near discontinuities, however they are not strictly orthogonal which adds to the solution time. These polynomials satisfy the criterion of minimum possible maximum error as well [61].

The list above is ordered by increasing robustness, but decreasing accuracy, the standard trade-off faced when choosing global trial functions. One nice property of using global orthogonal functions is that the evolution equations reduce to explicit uncoupled ordinary differential equations.

7.4.3 Finite differences

Finite difference schemes result from approximating derivatives with finite expansions based on some underlying grid [128], but are strongly connected with finite element schemes. Low-order finite element shape functions on a regular grid produce finite difference expressions. For example, the general convection-diffusion equation in one spatial dimension,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2} = 0,$$

reduces on a regular grid with linear elements to

$$\frac{1}{6}\frac{du_{k+1}}{dt} + \frac{2}{3}\frac{du_k}{dt} + \frac{1}{6}\frac{du_{k-1}}{dt} + c\frac{u_{k+1} - u_{k-1}}{2\Delta x} + D\frac{u_{k+1} - 2u_k + u_{k-1}}{(\Delta x)^2} = 0.$$

The last two terms are immediately seen to be identical to those produced by a three-point centered finite difference approach. The major difference in the two expressions above is that the finite difference formula is explicit, while the finite element formula is implicit. The latter is more complex to solve but tends to require fewer iterations due to the relaxation over three points instead of one.

For the term $\frac{\partial u}{\partial x}$, a quadratic finite element scheme on a regular grid produces

$$\frac{u_{k-2} - 4u_{k-1} + 4u_{k+1} - u_{k+2}}{4\Delta x}$$

while a five-point finite difference scheme generates

$$\frac{u_{k-2} - 8u_{k-1} + 8u_{k+1} - u_{k+2}}{12\Delta x}.$$

For the second-order term $\frac{\partial^2 u}{\partial x^2}$, finite elements gives

$$\frac{-u_{k-2} + 8u_{k-1} - 14u_k + 8u_{k+1} - u_{k+2}}{4\,(\Delta x)^2}$$

and finite differences gives

$$\frac{-u_{k-2} + 16u_{k-1} - 30u_k + 16u_{k+1} - u_{k+2}}{12\,(\Delta x)^2}.$$

The two schemes generate similar formulas which involve the same terms, but with different weights. The different weights arise in the finite element scheme from the shape of the trial functions, which can be adjusted to give any weights desired and hence any finite difference expression.

7.4.4 Wavelet bases

The use of a wavelet basis exhibits the highest degree of exploitation of orthogonality. Wavelets are generated from a single function with two parameters, scaling and translation, and are constructed so that every distinct choice of the two parameters yields a function which is orthogonal to all the others. Wavelets are by necessity local in nature, but the scaling allows this degree of locality to change. Use of a wavelet basis in a partial differential equation solution scheme is shown by Bacry and Mallat [10].

7.4.5 Problem-specific basis functions

The use of analytical, as opposed to empirical, functions has a long history because the first applications of Galërkin methods occurred before the widespread acceptance of computers, when it was necessary to perform all calculations by hand or pocket calculator. All the previous subsections explained various popular choices of trial functions, the choice of which is motivated by many different factors. Clearly one desires to have the ultimate set of basis functions for each particular problem, but it is rarely obvious what this set would be. Instead of restricting the choice to analytic functions only, widening the possibilities to include all functions allows the derivation of the single optimal basis for a given problem. This derivation is explained in the next chapter, and uses knowledge of previous solutions of the problem and constraints on the physical system.

8 Basis computation

Central to the whole procedure of basis space integration is the idea that the set of states the system will visit over the course of an integration is much smaller than the set of all possible states, allowing a representation of the state of the system in terms of a basis of this smaller set. This section describes how to generate that basis given information about the system: previous solutions and physical constraints. Also to be discussed are suggestions about what to do when enough information is not available. This chapter concludes with the related topic of determining how good a selected basis is, based on different criteria.

8.1 Previous solutions

Given a set of previous simulations of a model, the problem is to generate a basis for future executions which is somehow representative of the previous solutions so that an expression of any of the solutions in the basis is as compact as possible. Standard bases are various polynomials or transcendental functions, which are chosen depending on expectations of the form of the solution, such as periodicity. Here instead of writing down a set of analytic basis functions, numerical fields are generated to capture as much information of the input fields in the least number of basis fields. The term "basis" usually implies orthogonality of its elements, and that will be the case studied here, although non-orthogonal dictionaries may be advantageous in other applications [37, 38, 85]. It will be clear in the next part of this thesis how orthogonality is required here.

The set of previous solutions and the basis set to be generated will both be taken to be subsets of the same Hilbert space so that an inner product will be available, and that only one will be required. Eventually it will be necessary to use discrete spaces, but for now a continuous representation will be used to make the presentation more intuitive. The experimentally gathered previous solutions are

$$u(\mathbf{x}, t)$$

defined on $X \times T$ which can be thought of as the product of a physical spatial domain and a parameter by which the fields change (*e.g.*, time), or as a discrete index to count the fields. The basis to be generated is

$$\{\phi_i(\mathbf{x})\}$$

where i must be a discrete index, and can be taken to be an integer (although the most general case has i as a net). It is guaranteed that the Hilbert space X has an orthonormal basis [117], but

the choice of such a basis is left open. What is desired is that the basis be aligned such that the first element is "closest" to the data, the second element is as close as it can be while remaining orthogonal to the first, and so on. This concept of closeness is made concrete with an inner product on X,

$$\langle u, v \rangle \triangleq \int_X u(\mathbf{x})v(\mathbf{x})\, d\mathbf{x},$$

here chosen without any weight function, but applications in some coordinates would require a weight. As long as the inner product remains well-defined* the addition of a weight function makes no difference.

In trying to find basis elements, it is desired that the element being searched be as close as possible to the example fields, or that

$$|\langle \phi_i(\mathbf{x}), u(\mathbf{x}, t) \rangle|$$

be as large as possible for each t. Now a norm on the parameter space must be chosen to expand this maximization for a single example field into one over a collection of examples. For the obvious case of L^1, the value which should be maximized becomes

$$\int_T |\langle \phi_i(\mathbf{x}), u(\mathbf{x}, t) \rangle|\, dt$$

and in general for L^p,

$$\int_T |\langle \phi_i(\mathbf{x}), u(\mathbf{x}, t) \rangle|^p\, dt.$$

Division by the total number of example fields and the taking of a $1/p$ root have been dropped, as these lead to the same answer since the problem is one of maximization. The problem must be constrained by requiring that the basis elements have unit norm, otherwise the solution would be to pick a very large ϕ_i, and that they are orthogonal to each other, otherwise all ϕ_i would be the same.

Solving this problem, for each i one at a time, can be considered intuitively as in Figure 30 where in three dimensions with zero-mean data and a least-squares norm, the line ϕ_1 is chosen to fit the data as closely as possible, then ϕ_2 is chosen to be perpendicular to ϕ_1 and capture as much of the remaining variation. ϕ_3 is then the only remaining vector orthogonal to the first two, under the constraint that all the axes have unit norm.

The choice of which norm to use in the maximization problem presented above may be dictated by the structure of a problem or its physical foundations. For example, if the data is concentrations of atmospheric pollutants, it may make sense to use the infinity (or max) norm, as regulations regarding the pollutants are often concerned about the maximum concentration over a whole day, not some average. Many applications prefer the 2-norm, perhaps because it is simple and intuitive, and fortunately for that case the maximization problem reduces using calculus of variation to an eigenvalue problem. For other norms, such a closed form solution may not exist.

8.1.1 L^2-optimal basis generation

For the case when the norm on the example field index parameter ("time") is a 2-norm, an equation which must be satisfied by all the basis elements leads to an easy method of calculation. This is the procedure of Karhunen-Loève, although more generalized than what is presented by the original developers [70, 82].

---

* An inner product will be taken to be a real-valued function from $X \times X$ to $\mathbb{R}$, and must satisfy the four conditions:

1. $\langle u, u \rangle \geq 0$, and $\langle u, u \rangle = 0$ if and only if $u = 0$
2. $\langle u, v + w \rangle = \langle u, v \rangle + \langle u, w \rangle$
3. $\langle u, av \rangle = a\langle u, v \rangle$, a a real constant
4. $\langle u, v \rangle = \langle v, u \rangle$

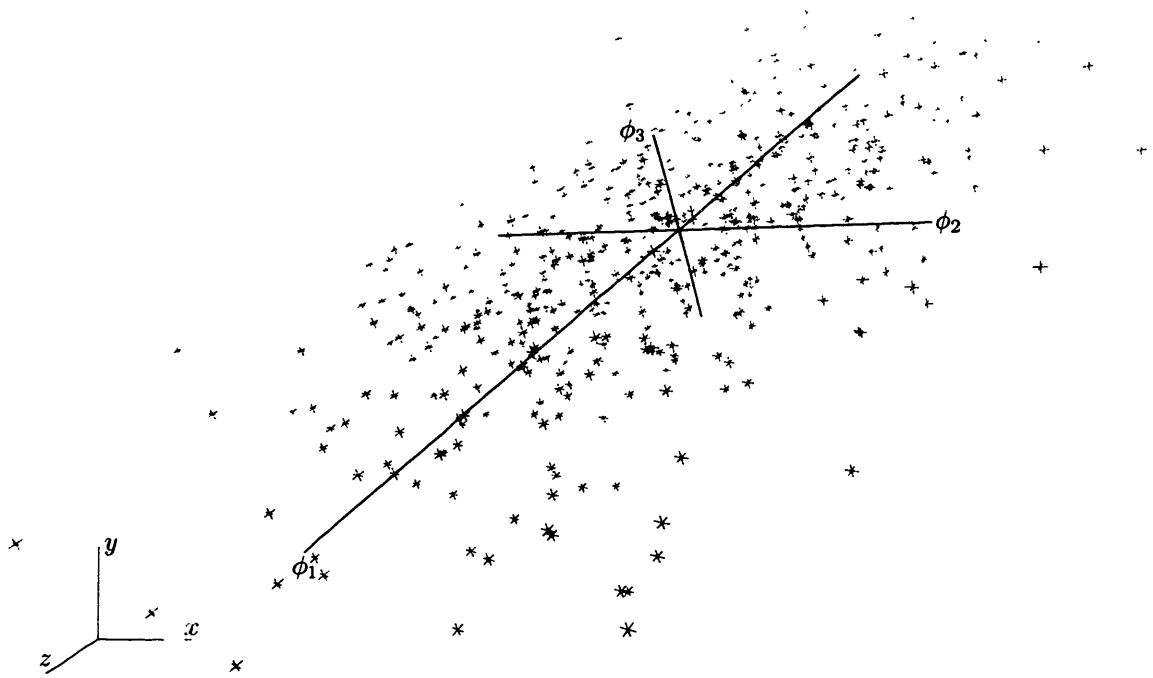


Figure 30. Optimal basis of a set of points, intuitively. The lengths of the axes are proportional to the amount of variance captured.

Assign to λ the value of the maximized function at the optimal ϕ,

$$\lambda = \frac{\int_T \langle \phi(\mathbf{x}), u(\mathbf{x},t)\rangle^2 \, dt}{\langle \phi(\mathbf{x}), \phi(\mathbf{x})\rangle},$$

where the constraint that ϕ be of unit norm is explicitly included in the equation as the dividing term $\sqrt{\langle \phi, \phi\rangle}$, which is squared in calculating the two-norm. Calculus of variations permits maximization by supposing that ϕ maximizes λ, then substituting any other function $\phi + \alpha\phi'$, ϕ' arbitrary, and seeing what happens to λ for small α.

$$\lambda' = \frac{\int_T \int_X \int_X u(\mathbf{x},t)u(\mathbf{y},t)\Big(\phi(\mathbf{x}) + \alpha\phi'(\mathbf{x})\Big)\Big(\phi(\mathbf{y}) + \alpha\phi'(\mathbf{y})\Big)\, d\mathbf{x}\, d\mathbf{y}\, dt}{\int_X \Big(\phi(\mathbf{x}) + \alpha\phi'(\mathbf{x})\Big)^2 \, d\mathbf{x}}$$

$$= \frac{\begin{array}{r}\int_T \int_X \int_X u(\mathbf{x},t)u(\mathbf{y},t)\phi(\mathbf{x})\phi(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y}\, dt + 2\alpha \int_T \int_X \int_X u(\mathbf{x},t)u(\mathbf{y},t)\phi(\mathbf{x})\phi'(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y}\, dt \\ +\alpha^2 \int_T \int_X \int_X u(\mathbf{x},t)u(\mathbf{y},t)\phi'(\mathbf{x})\phi'(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y}\, dt\end{array}}{\int_X \phi^2(\mathbf{x})\, d\mathbf{x} + 2\alpha \int_X \phi(\mathbf{x})\phi'(\mathbf{x})\, d\mathbf{x} + \alpha^2 \int_X \phi'^2(\mathbf{x})\, d\mathbf{x}}$$

Taking the derivative of λ' with respect to α, setting α and the derivative equal to zero gives

$$\frac{\int_T \int_X \int_X u(\mathbf{x},t)u(\mathbf{y},t)\phi(\mathbf{x})\phi(\mathbf{y})\ d\mathbf{x}\ d\mathbf{y}\ dt}{\int_X \phi^2(\mathbf{x})\ d\mathbf{x}} = \frac{\int_T \int_X \int_X u(\mathbf{x},t)u(\mathbf{y},t)\phi(\mathbf{x})\phi'(\mathbf{y})\ d\mathbf{x}\ d\mathbf{y}\ dt}{\int_X \phi(\mathbf{x})\phi'(\mathbf{x})\ d\mathbf{x}}$$

where the left hand side is just the equation for λ, which permits the following:

$$\int_T \int_X u(\mathbf{x},t)\phi'(\mathbf{y}) \int_X u(\mathbf{y},t)\phi(\mathbf{y})\ d\mathbf{y}\ d\mathbf{x}\ dt = \lambda \int_X \phi(\mathbf{x})\phi'(\mathbf{x})\ d\mathbf{x}$$

$$\int_T \int_X u(\mathbf{x},t)\phi'(\mathbf{x})\langle u(\mathbf{y},t),\phi(\mathbf{y})\rangle\ d\mathbf{x}\ dt = \lambda\langle\phi,\phi'\rangle$$

$$\left\langle \int_T \langle u(\mathbf{y},t),\phi(\mathbf{y})\rangle u(\mathbf{x},t)\ dt, \phi'(\mathbf{x})\right\rangle - \langle\lambda\phi,\phi'\rangle = 0$$

$$\left\langle \int_T \langle u(\mathbf{y},t),\phi(\mathbf{y})\rangle u(\mathbf{x},t) - \lambda\phi(\mathbf{x}), \phi'(\mathbf{x})\right\rangle = 0$$

Since ϕ' is arbitrary, this gives an equation for ϕ and λ, which can be written in integral form:

$$\int_T \int_X u(\mathbf{x},t)u(\mathbf{y},t)\phi(\mathbf{y})\ d\mathbf{y}\ dt = \lambda\phi(\mathbf{x})$$

or, if the definition of the spatial covariance matrix, $K(\mathbf{x},\mathbf{y}) = \int_T u(\mathbf{x},t)u(\mathbf{y},t)\ dt$, is used,

$$\int_X K(\mathbf{x},\mathbf{y})\phi(\mathbf{y})\ d\mathbf{y} = \lambda\phi(\mathbf{x}).$$

Thus the problem of determining the optimal basis using an L^2 norm on the input fields reduces to a standard eigenvalue problem.

Similarly, it may be postulated that there is a set of functions $\{\beta_i(t)\}$ such that for each point $\mathbf{x}$ in the field, the value

$$\langle\beta(t),u(\mathbf{x},t)\rangle_T^2 \triangleq \int_T \beta(t)u(\mathbf{x},t)\ dt$$

is maximal. Requiring the same to hold for all points $\mathbf{x}$ in the field gives the maximization problem

$$\lambda = \frac{\int_X \langle\beta(t),u(\mathbf{x},t)\rangle_T^2\ d\mathbf{x}}{\langle\beta(t),\beta(t)\rangle_T}.$$

This gives the solution, in analogy with that for ϕ,

$$\int_T C(s,t)\beta(s)\ ds = \lambda\beta(t),$$

where the temporal (or parametric) covariance, $C(s,t) = \int_X u(\mathbf{x},t)u(\mathbf{x},s)\ d\mathbf{x}$, is used. This gives the optimal fit in parameter space at a single point in the field:

$$u(\mathbf{x},t) = \sum_i \alpha_i\beta_i(t).$$

Augmenting the above to cover the entire spatial domain,

$$u(\mathbf{x},t) = \sum_i \alpha_i \beta_i(t)\phi(\mathbf{x}),$$

a procedure to calculate the ϕ is obtained by taking the inner product of both sides with $\beta_j(t)$ and using orthonormality of $\{\beta_i(t)\}$:

$$\langle \beta_j(t), u(\mathbf{x},t)\rangle_T = \alpha_j \phi(\mathbf{x}),$$

or

$$\phi_j(\mathbf{x}) = \frac{1}{\alpha_j}\int_T \beta_j(t)u(\mathbf{x},t),$$

one for each β_j. Furthermore, the newly generated set $\{\phi_j(\mathbf{x})\}$ is orthogonal in X since

$$\begin{aligned}
\langle \phi_i, \phi_j\rangle &= \frac{1}{\alpha_i\alpha_j}\int_X\int_T\int_T \beta_i(t)\beta_j(s)u(\mathbf{x},t)u(\mathbf{x},s)\,ds\,dt\,d\mathbf{x} \\
&= \frac{1}{\alpha_i\alpha_j}\int_T\int_T \beta_i(t)\beta_j(s)C(s,t)\,ds\,dt \\
&= \frac{1}{\alpha_i\alpha_j}\int_T \beta_i(t)\lambda_j\beta_j(t)\,dt \\
&= \frac{\lambda_i}{\alpha_i^2}\delta_{ij}
\end{aligned}$$

and can be normalized by setting $\alpha_i = \sqrt{\lambda_i}$ in the expansion.

The set $\{\phi_i(\mathbf{x})\}$ generated by requiring an L^2-optimal in parameter space set $\{\beta_i(t)\}$ can be seen to be identical to those generated directly by substituting the procedure for deriving $\phi_i(\mathbf{x})$ into the eigenvalue problem for calculating it directly, using $\widehat{\lambda}$ for the temporal eigenvalues:

$$\begin{aligned}
\int_X K(\mathbf{x},\mathbf{y})\phi_i(\mathbf{y})\,d\mathbf{y} &= \lambda_i\phi_i(\mathbf{x}) \\
\int_X\int_T u(\mathbf{x},t)u(\mathbf{y},t)\phi_i(\mathbf{y})\,d\mathbf{y}\,dt &= \lambda_i\phi_i(\mathbf{x}) \\
\int_X\int_T\int_T u(\mathbf{x},t)u(\mathbf{y},t)\frac{1}{\sqrt{\widehat{\lambda}_i}}\beta_i(s)u(\mathbf{y},s)\,d\mathbf{y}\,dt\,ds &= \lambda_i\int_T \frac{1}{\sqrt{\widehat{\lambda}_i}}\beta_i(t)u(\mathbf{x},t)\,dt \\
\int_T\int_T C(s,t)\beta_i(s)u(\mathbf{x},t)\,dt\,ds &= \lambda_i\int_T \beta_i(t)u(\mathbf{x},t)\,dt \\
\widehat{\lambda}_i\int_T \beta_i(t)u(\mathbf{x},t)\,dt &= \lambda_i\int_T \beta_i(t)u(\mathbf{x},t)\,dt \\
\widehat{\lambda}_i &= \lambda_i
\end{aligned}$$

Hence the optimization criterion is satisfied, and it is seen that the eigenvalues agree. In the discrete cases used in practice, the number of fields is often much smaller than the number of grid points, making the calculation of $C(s,t)$ much easier than that of $K(\mathbf{x},\mathbf{y})$, so the procedure of generating the temporal eigenfunctions first and from it deriving the spatial ones is favored.

8.1.2 General overview

Now that the details for the case of an L^2 norm have been discussed, it is worth stepping back from the problem and considering other ways it may be presented (in arbitrary norm). There are four different viewpoints in which one can frame the Karhunen-Loève expansion, which may add some intuitive feel to this procedure.

1. As an extension to Gram-Schmidt orthogonalization, Karhunen-Loève can be seen as an iterative procedure to generate a basis. First define $\phi_1(\mathbf{x}, t)$ to be the principal axis in the sense that

$$\phi_1 = \max_{\|\phi\|=1} \langle \phi, u \rangle_{X \times T}.$$

   Then successive fields are defined by

$$\phi_i = \max_{\|\phi\|=1} \langle \phi, u \rangle_{X \times T} \quad \text{such that} \quad \langle \phi_i, \phi_j \rangle_{X \times T} = 0 \quad \text{for} \quad j < i.$$

   This is the conceptual approach used in the field of statistics for deriving the principal components of a data set [67], called "principal component analysis" there, and illustrated in Figure 30.
2. Supposing that the experimental fields can be approximated by a series expansion of separated basis functions,

$$u(\mathbf{x}, t) = \sum_{i=1}^{N} \beta_i(t) \phi_i(\mathbf{x}),$$

   where $\beta_i(t) = \int_X u(\mathbf{x}, t) \phi_i(\mathbf{x})\, d\mathbf{x}$, require that the time-dependent coefficients be uncorrelated:

$$\int_T \beta_i(t) \beta_j(t)\, dt = \lambda_i \delta_{ij}.$$

   This will ensure a unique representation and generate the constants λ_i which can be considered the *energy* of a particular state. The whole system has an energy approximated by:

$$\|u\|^2 = \left\langle \sum_{i=1}^{N} \beta_i(t) \phi_i(\mathbf{x}), \sum_{j=1}^{N} \beta_j(t) \phi_j(\mathbf{x}) \right\rangle_{X \times T} = \sum_{i=1}^{N} \lambda_i.$$

   These coefficients are precisely the λ_i mentioned above, after sorting into a decreasing sequence.
3. The spatial covariance matrix of the given fields can be constructed:

$$K(\mathbf{x}, \mathbf{y}) = \int_T u(\mathbf{x}, t) u(\mathbf{y}, t)\, dt,$$

   a function of time, and it is symmetric and positive. Again, it will have a uniformly convergent spectral decomposition

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} \beta_i(t) \phi_i(\mathbf{x}) \phi_i(\mathbf{y})$$

   which satisfies

$$\int_X K(\mathbf{x}, \mathbf{y}) \phi_i(\mathbf{y})\, d\mathbf{y} = \beta_i(t) \phi_i(\mathbf{x}).$$

   The resulting decomposition can be shown equivalent to the earlier ones when the coefficients are averaged over the spatial domain. This point of view is preferable when members of the ensemble are just snapshots of the system as it evolves in time, so that in the discrete case the operations of ensemble average and time average are identical.
4. Equivalently the covariance in time, which may be a more tractable quantity if the space dimensions are large in comparison,

$$C(t, s) = \int_X u(\mathbf{x}, t), u(\mathbf{x}, s)\, d\mathbf{x}$$

can be spectrally decomposed as before to generate expansions of the form

$$u(\mathbf{x},t) = \sum_{i=1}^{N} \beta_i(t)\phi_i(\mathbf{x})$$

in which

$$\int_T C(t,s)\beta_i(s)\,ds = \phi_i(\mathbf{x})\beta_i(t)$$

Requiring that $\int_T \alpha_i(t)\alpha_j(t)\,dt = \delta_{ij}$ provides a method to generate the spatial fields:

$$\int_T u(\mathbf{x},t)\beta_i(t)\,dt = \sum_{j=1}^{N} \phi_j(\mathbf{x})\delta_{ij} = \phi_i(\mathbf{x}).$$

The normalization constant for the spatial fields recovers the energy as defined in item two.

Incidentally, the Karhunen-Loève expansion and its variations are called by many other names: "proper orthogonal decomposition" in operator theory [8], "principal component analysis" in statistics [115], and "empirical orthogonal functions" in meteorology [125]. The snapshot method of basis generation is explained further in Sirovich [126]. Since the basis fields so generated are complete (with respect to the space spanned by the original ensemble), any function can be expanded using them:

$$u = \sum_{i=1}^{\infty} \langle u, \phi_i \rangle \phi_i$$

Also, the coefficients of such an expansion are uncorrelated.

Computationally the eigenvalue problems derived above are usually solved using singular value decomposition since the number of significant modes in the input data (numerical rank of the covariance matrix) is frequently small compared to the total number of inputs (dimension of the covariance matrix), hence calculating just the significant fields, as determined by the size of the eigenvalue, is a good scheme. It is possible, however, to use other eigenvalue solvers to extract all or the first few eigenfunctions, or a range dependent on eigenvalue magnitude. Power methods may allow for quicker eigenfunction extractions when needed, but the next part of this thesis will show basis computation to be an "out of the loop" calculation so the time it takes to solve the eigenvalue problem is irrelevant, as it need be done so infrequently. Also present is the choice of using a correlation matrix instead of the covariance matrix discussed above. This has the advantage of producing numerically well-scaled values, although a slightly different maximization criterion is implicitly being used. More discussion on the numerical aspects can be found in the appendices, but a general outline of the algorithm is given in Table 10.

Read input, gathering field dimensions	$u_i(\mathbf{x})$
Optionally subtract off the mean	$\bar{u}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} u_i(\mathbf{x}), \quad u_i(\mathbf{x}) = u_i(\mathbf{x}) - \bar{u}(\mathbf{x})$
Compute the covariance matrix	$C(i,j) = \int_X u_i(\mathbf{x})u_j(\mathbf{x})\,d\mathbf{x}$
Call SVD to decompose	$C = T \cdot W \cdot S^{\mathsf{T}}$
Generate spatial eigenfunctions	$\phi_i(\mathbf{x}) = \frac{1}{\sqrt{W_{ii}}}\sum_{j=1}^{N} u_j(\mathbf{x})T_{ij}$

Table 10. Algorithm used to generate a Karhunen-Loève basis, with equations from the text.

8.2 Coping with a lack of information

When there is simply not enough known about the model which is under study, and the construction of a good basis for the problem is not possible, there are methods by which an artificial basis may be used. One way is to choose a basis that is complete, run the simulations using it, then prune it down by extracting another basis from the solutions thus generated. For instance, on a grid consisting of points $\{\mathbf{x}_n\}_{n=1}^{N}$, the basis fields defined as

$$\phi_n(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_n)$$

are orthonormal as required and span the space of all possible functions on the grid. Of course the number of basis elements is precisely the number of grid cells, which is considered huge when used for an integration of this sort, but feasible for a few test runs. Solutions using this complete basis can be superimposed and used as input fields to generate a smaller, more practical basis. From this point, basis augmentation techniques to be described in later sections may be used.

Another possibility for generating a preliminary basis is to use knowledge about characteristics of the problem. If it is suspected that the output will be generally smooth or periodic, for instance, orthogonal sine fields can be generated. If a moving front is suspected, localized bumps may be constructed. Any construction method of the suspected fields is acceptable, as they can be made orthogonal by many different ways. The direct method is to create fields one at a time, forcing them to be orthogonal to all previously created fields by direct subtraction of their projection components, much like Gram-Schmidt orthogonalization. Indirectly, many candidate fields may be constructed, multiply so according to their presupposed "importance" to force a stronger weighting during basis generation. The basis generated from these inputs can be used directly for a few rough simulations, and augmented as usual.

8.3 Static error determination

Information must by necessity be lost when expressing large-dimensional grid-based fields in terms of a smaller dimension functional expansion. This section discusses some ways of computing and reducing that error.

8.3.1 System-generated error

Before choosing and constructing a basis to be used in an integration, it is possible to analyze the system under study to get upper bounds on the error at any point in time given the initial condition errors and system evolution equations. The simplest chemical reaction will first be considered

$$A \xrightarrow{k} B$$

with expansions for the species to the indicated levels

$$\mathbf{A} = \sum_{i=1}^{N_A} a_i \mathbf{A}_i, \qquad \mathbf{B} = \sum_{i=1}^{N_B} b_i \mathbf{B}_i$$

and a domain size of N_G grid cells. ($\mathbf{A}_i$ stands for the i-th orthonormal basis function in an expansion of $A(\mathbf{x}, t)$, and $a_i(t)$ is the scalar coefficient at time t.) This value N_G represents the maximum possible expansion since exactly N_G independent fields are needed to span the complete space.

It is straightforward to write down the L^2 error of the initial expansions, as the difference between the exact expansion $\tilde{\mathbf{A}}$ and the approximate one

$$\begin{aligned}
\mathrm{err}_A(0) &= \|\tilde{\mathbf{A}} - \mathbf{A}\|^2 \\
&= \left\| \sum_{i=1}^{N_G} a_i(0)\mathbf{A}_i - \sum_{i=1}^{N_A} a_i(0)\mathbf{A}_i \right\|^2 \\
&= \left\langle \sum_{i=N_A+1}^{N_G} a_i(0)\mathbf{A}_i, \sum_{i=N_A+1}^{N_G} a_i(0)\mathbf{A}_i \right\rangle \\
&= \sum_{i,j=N_A+1}^{N_G} a_i(0)a_j(0)\langle \mathbf{A}_i, \mathbf{A}_j \rangle \\
&= \sum_{i,j=N_A+1}^{N_G} a_i(0)a_j(0)\delta_{ij} \\
&= \sum_{i=N_A+1}^{N_G} a_i^2(0)
\end{aligned}$$

Similarly,

$$\mathrm{err}_B(0) = \sum_{i=N_B+1}^{N_G} b_i^2(0)$$

where the coefficients are taken to be functions of time, reflecting the various values taken during the integration. Error in the representation of $\mathbf{A}$ will propagate into $\mathbf{B}$, but not the other way around, as will be seen in the derivation of system equations in the next part of the thesis, accepted for now on faith. Comparing the exact formula for $\mathbf{A}$'s coefficients changing through time and the truncated one:

$$\begin{aligned}
\tilde{\dot{a}}_i &= -ka_i \qquad i \in 1..N_G \\
\dot{a}_i &= -ka_i \qquad i \in 1..N_A
\end{aligned}$$

yields, in a manner similar to the calculation of $\mathrm{err}_A(0)$.

$$\begin{aligned}
\dot{\mathrm{err}}_A(t) &= \left\| \sum_{i=N_A+1}^{N_G} -ka_i(t)\mathbf{A}_i \right\|^2 \\
&= k^2 \sum_{i=N_A+1}^{N_G} a_i^2(t).
\end{aligned}$$

The factor $\mathbf{A}_i$ is included in determining the error, but it is understood that this basis element, for $i > N_A$, was never constructed. Every Hilbert space has a basis so $\mathbf{A}_i$ is known to exist, however. The expression for $\mathbf{B}$ is complicated by the fact that it is affected through $\mathbf{A}$; the first line below is the evolution equation for $\mathbf{B}$ as determined by the next part.

$$\tilde{\dot{b}}_i = k\sum_{j=1}^{N_G} a_j\langle \mathbf{B}_i, \mathbf{A}_j \rangle \qquad i \in 1..N_G$$

$$\dot{b}_i = \begin{cases} k\sum_{j=1}^{N_m} a_j\langle \mathbf{B}_i, \mathbf{A}_j \rangle & i \in 1..N_B \\ 0 & \text{otherwise} \end{cases}$$

where $N_m = \min(N_A, N_B)$. Hence it is correct to write

$$\dot{\text{err}}_B(t) = \left\| k \sum_{i=N_B+1}^{N_G} \sum_{j=1}^{N_m} a_j \langle \mathbf{B}_i, \mathbf{A}_j \rangle \mathbf{B}_i + k \sum_{i=1}^{N_G} \sum_{j=N_m+1}^{N_G} a_j \langle \mathbf{B}_i, \mathbf{A}_j \rangle \mathbf{B}_i \right\|^2$$

which after much manipulation becomes

$$\dot{\text{err}}_B(t) = k^2 \sum_{i=N_B+1}^{N_G} \sum_{j=1}^{N_G} \sum_{k=1}^{N_G} a_j a_k \langle \mathbf{A}_j, \mathbf{B}_i \rangle \langle \mathbf{A}_k, \mathbf{B}_i \rangle + k^2 \sum_{i=1}^{N_B} \sum_{j=N_m+1}^{N_G} \sum_{k=N_m+1}^{N_G} a_j a_k \langle \mathbf{A}_j, \mathbf{B}_i \rangle \langle \mathbf{A}_k, \mathbf{B}_i \rangle$$

It is not hard to see (although messy to write) that expressions of this sort may be formed for arbitrary systems of chemistry and basis functions.

The analytic solution to the chemical system may be written down directly

$$a_i(t) = a_i(0) e^{-kt}$$
$$b_i(t) = b_i(0) + a_i(0) \left(1 - e^{-kt}\right)$$

and substituted into the equations for $\dot{\text{err}}_A(t)$ and $\dot{\text{err}}_B(t)$, which can be integrated to find the total error at any point in the integration.

$$\begin{aligned} \text{err}_A(t) &= k^2 \sum_{i=N_A+1}^{N_G} a_i^2(0) \int_0^t e^{-2kt'} \, dt' \\ &= \frac{k}{2} \left(1 - e^{-2kt}\right) \sum_{i=N_A+1}^{N_G} a_i^2(0) \end{aligned}$$

Similarly for $\mathbf{B}$,

$$\text{err}_B(t) = \frac{k}{2} \left(1 - e^{-2kt}\right) \left(\sum_{i=N_B+1}^{N_G} \sum_{j=1}^{N_G} \sum_{k=1}^{N_G} a_j(0) a_k(0) \langle \mathbf{A}_j, \mathbf{B}_i \rangle \langle \mathbf{A}_k, \mathbf{B}_i \rangle \right.$$
$$\left. + \sum_{i=1}^{N_B} \sum_{j=N_m+1}^{N_G} \sum_{k=N_m+1}^{N_G} a_j(0) a_k(0) \langle \mathbf{A}_j, \mathbf{B}_i \rangle \langle \mathbf{A}_k, \mathbf{B}_i \rangle \right)$$

The only knowledge which can be used to simplify this last expression is that all the basis fields are normalized, so $\langle \mathbf{A}_j, \mathbf{B}_k \rangle$ is always smaller than 1. Hence,

$$\text{err}_B(t) \leq \frac{k}{2} \left(1 - e^{-2kt}\right) \left((N_G - N_B - 1) \sum_{i,j=1}^{N_G} a_j(0) a_k(0) + N_B \sum_{i,j=N_m+1}^{N_G} a_j(0) a_k(0) \right)$$

regardless of the particular basis used in the expansions.

As an example of these last results, notice that the expression $1 - e^{-2kt} \rightarrow 1$ as t gets large, with larger values of k causing quicker approach to the asymptote. Requiring that the upper bound of error stay below some value becomes quite simple. Consider a 10×10 grid with 7 fields needed to expand the initial condition of $\mathbf{A}$, and 4 for $\mathbf{B}$, both so that the unused coefficients are no larger than 10^{-3}. Then

$$\text{err}_A(\infty) \leq 4.7 \times 10^{-5} \, k$$
$$\text{err}_B(\infty) \leq 4.9 \times 10^{-1} \, k$$

It is seen that the chemistry directly gives an upper bound on the error even before choosing the basis fields, which can only decrease the upper bound due to the eigenvalue ordering chosen by the solver (singular value decomposition, for example). Working backwards to find N_A and N_B given k and the error tolerances is also possible, using the above formulas.

8.3.2 Basis-generated error

The obvious way to generate a basis for the participating species is to perform a simulation at the specified conditions using a standard grid-based integration method, then extract from those fields a complete basis. In this section the goal is to construct a basis which is complete, that is, such that an integration in the space of the complete basis will accumulate no errors due to any projection coefficients falling outside the span of the space. In this way, errors which may arise due to a poorly computed basis may be removed.

In attempting to form as large a basis as possible, it is necessary to ensure that each successive field in the basis is orthogonal to all the others as well as normalized itself. For any particular basis $\{\phi_i\}$, the matrix with elements $\langle \phi_i, \phi_j \rangle$, the inner product of two of the basis members, can be constructed. For a perfectly orthonormal set this inner product should be the discrete delta function δ_{ij} resulting in an identity matrix. The "diagonal deviation" is defined as the largest absolute value difference of any diagonal element from one, and indicates normalization error. The "body deviation" is the largest off-diagonal element, all of which should be zero, and indicates a more fundamental problem in the construction of the fields: lack of orthogonality. These criteria are useful in gauging how much accuracy is being lost due to an improper basis.

species	size	diagonal deviation	body deviation
NO	10	8.42×10^{-6}	2.23×10^{-4}
NO_2	11	7.32×10^{-8}	1.86×10^{-2}
O	10	5.64×10^{-6}	9.30×10^{-4}
O_3	10	4.86×10^{-6}	9.08×10^{-7}

Table 11. Deviations obtained by one-pass SVD method.

It is instructive to compare two different methods for generating a basis. An example problem involving four reacting chemical species with no transport will be considered for illustrative purposes. First, the standard one-pass singular value decomposition eigenvalue problem solution method works by reading in the input fields, computing the decomposition, and writing the output. Accumulated errors in extracting all the eigenfunctions at once account for the magnitude of the errors reported in Table 11. The example fields were generated by a grid-based simulator from each of eleven time steps including the initial condition, but only one species started out at nonzero concentration, hence the disparity in the sizes of the bases.

species	size	diagonal deviation	body deviation
NO	10	2.44×10^{-15}	9.72×10^{-11}
NO_2	11	1.89×10^{-15}	1.59×10^{-7}
O	10	1.22×10^{-15}	9.70×10^{-10}
O_3	10	8.88×10^{-16}	4.80×10^{-11}

Table 12. Deviations obtained by single-eigenvalue extraction method.

The results of a second way, a renormalizing single-eigenvalue extraction procedure, are reported in Table 12. It works by taking only the largest eigenvalue from the covariance matrix as the first basis element, then replacing the input fields with their orthogonal complements with respect to that basis element, normalizing each input to one, then constructing a new covariance matrix and repeating until the entire basis has been constructed. This method is considerably slower, but as

can be seen, the deviations from a perfect basis are much smaller. The deviations reported here seem quite acceptable for most applications.

It can be understood why the basis produced by singular value decomposition is so much worse than that produced by the renormalizing single-eigenvalue extraction method by examining a plot of the eigenvalues of one of the covariance matrices. Shown in Figure 31 are the logarithms of the eigenvalues of the covariance matrix for species NO_2. The way that the magnitude of the eigenvalues falls off quickly is indicative of an input set of small numerical rank. Considering the eigenvalues as the squared lengths of the principal axes of the ellipse described by the covariance matrix, each successive axis is an order of magnitude smaller than the previous one. This is what leads to the inaccuracies in the generated basis, but only because it was demanded that a complete set be produced. In situations when the data is not so one-dimensional and only the first few (important) eigenvectors are required, singular value decomposition becomes the preferred technique due to its increased speed.

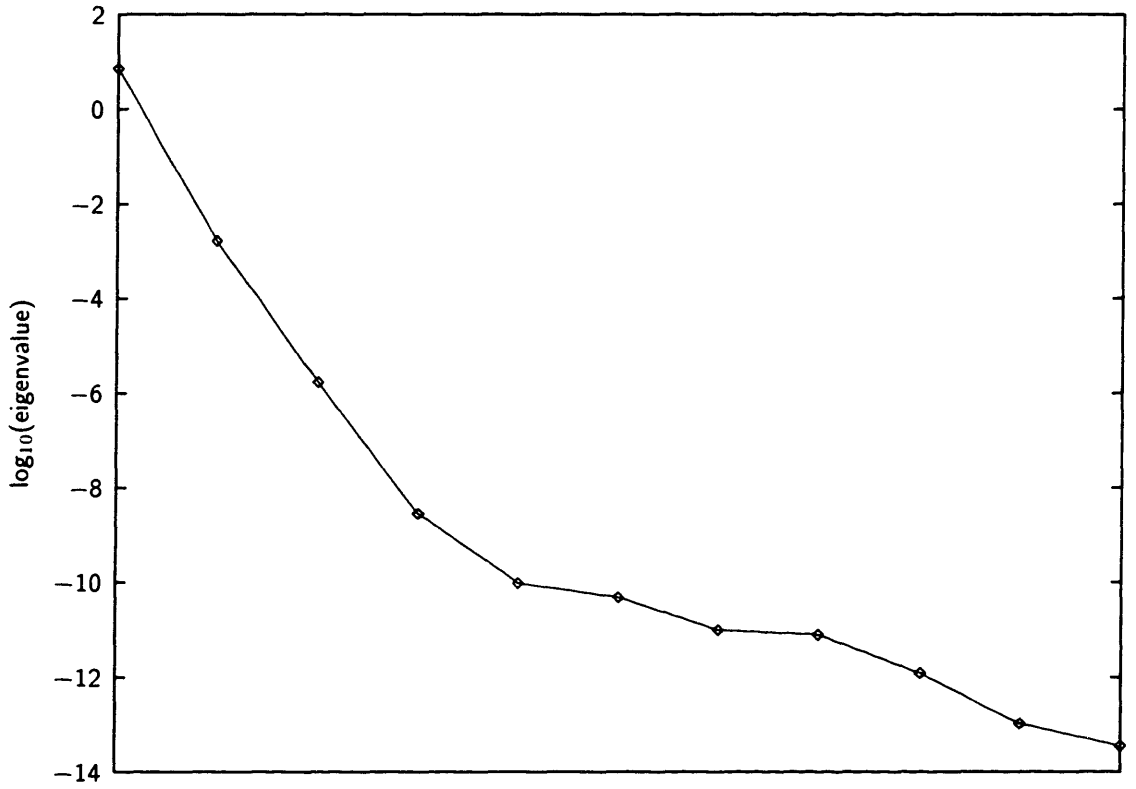


Figure 31. Eigenvalue spectrum for species NO_2.

Further iterative improvements on any generated basis are possible through other means. One utility program is `orthbasis` which attempts to make the basis orthonormal, that is, $\langle \phi_i, \phi_j \rangle = \delta_{ij}$ is satisfied as closely as numerically possible. This effectively always reduces the maximum deviations reported above down to machine precision (about 10^{-16} on the particular workstation used in this thesis), although it does not consider the original inputs in any way.

Another way to generate new basis members which might help in reducing integration error is to examine just where the projection assumption is not satisfied. Expressing the system of differential equations in terms of a basis presupposes that each projected field lies completely in the span of the space on which it is to be projected, which is not necessarily the case for all points during the integration. As a concrete example, the calculation of the concentration time derivatives for species

O is:

$$\dot{c}_{\mathrm{O},i} = -k_2\,\mathrm{M}\,\mathrm{O}_2\,c_{\mathrm{O},i} + k_1 \sum_{j=1}^{N_{\mathrm{NO}}} c_{\mathrm{NO},j}\langle \mathbf{NO_2}_j, \mathbf{O}_i\rangle.$$

The first term presents no problem, but the second one relies on the fact that for each basis field j of NO_2,

$$\sum_{i=1}^{N_{\mathrm{O}}} \langle \mathbf{NO_2}_j, \mathbf{O}_i\rangle = 1,$$

which is equivalent to saying that field $\mathbf{NO_2}_j$ lies completely within the span of $\{\mathbf{O}_i\}_{i=1}^{N_{\mathrm{O}}}$ and that the latter set is orthonormal. One way of testing this assumption is simply to project each NO field onto the O basis and see how close the sum above is to one. This is implemented by a simple shell script **proj_goodness**, which actually prints the distance of the result from one to allow higher accuracy. A plot of the common logarithm of the results shows which fields are poorly represented in the respective basis. This and other utilities are described fully in the appendices. The simple fix when projection errors arise is to include members of other bases in the input set of the failing one, or in this case O input fields should include the basis for NO_2. More examples of this appear in the next part of the thesis.

When the evolution equations involve nonpolynomial nonlinearities, other higher products or transcendental functions of basis elements must be considered, but this is an easy extension to the basic technique described here.

Part III

Use of Optimal Bases in Integration

Having considered general mathematical analysis and modeling techniques including machine mapping, and having shown a method to construct an optimal basis for each particular problem, it remains to show how all this can be used to perform actual simulations. The following chapters cover conversion of the evolution operators and dynamic error tracking during an integration, and conclude with some timing results.

9 System conversion

Previous chapters have described the approach to be taken in solving differential equations using an empirically derived basis, and have shown techniques and considerations of constructing an appropriate basis. Transformation of the equations into the basis thus generated and implementation of the transformed equations will be described in the current chapter, starting with the simplest case, ordinary differential equations, then adding the framework for handling terms involving partial derivatives. Special transformations when the system can be characterized by a global motion such as a rotation or translation are considered also. Finally a short description of machine-generated system conversion is presented.

9.1 Ordinary differential equations

First, the simplest sort of problem is considered. A linear homogenous ordinary differential equation of one dependent variable is to be solved as an initial value problem. The number of independent variables is arbitrary, but one is taken to be "time" to allow a forward integration. For simplicity of presentation, $\mathbf{A}$ will be some scalar-valued field over two spatial dimensions and t the independent variable. The general form is

$$\frac{d\mathbf{A}(x,y)}{dt} = \alpha(t)\mathbf{A}(x,y)$$

with α is an arbitrary function of t, but not of $\mathbf{A}$. For now this equation indicates the simulation of a field which changes only pointwise, that is, there are no interactions between neighboring points of the spatial field such as diffusive or convective effects. While not arising directly from any physical system, this type of equation is often encountered during the reduction of a more complex differential equation using operator splitting methods. The full evolution operator is broken up into conceptually separate chunks, and executed iteratively to obtain the result of the entire operation.

Using the Galërkin assumption that $\mathbf{A}$ can be well-approximated by a set of basis functions, this equation becomes

$$\frac{d}{dt}\left(\sum_i a_i\mathbf{A}_i(x,y)\right) = \alpha(t)\sum_i a_i\mathbf{A}_i(x,y),$$

where $\mathbf{A}_i$ is the i-th basis field, which has the same dimensionality of $\mathbf{A}$, and a_i is a scalar coefficient scaling the contribution of that basis element. There is no loss in generality in choosing an orthogonal

basis and normalizing each element to one. Since the basis does not change during the integration, the equation becomes

$$\sum_i \frac{da_i}{dt}\mathbf{A}_i(x,y) = \alpha(t)\sum_i a_i\mathbf{A}_i(x,y).$$

Now orthogonality will allow the separation of this equation into one for each basis coefficient, by taking the inner product of both sides:

$$\sum_j \frac{da_j}{dt}\langle \mathbf{A}_i(x,y), \mathbf{A}_j(x,y)\rangle = \alpha(t)\sum_j a_j\langle \mathbf{A}_i(x,y), \mathbf{A}_j(x,y)\rangle$$

and simplifying the inner product to a Kronecker delta function:

$$\sum_j \frac{da_j}{dt}\delta_{ij} = \alpha(t)\sum_j a_j\delta_{ij}$$

to give the evolution equation for each coefficient:

$$\frac{da_i}{dt} = \alpha(t)a_i.$$

The result is trivial, but illustrates the basic principles behind differential equation conversion.

Extending to multiple dependent variables or adding nonlinearities both require the introduction of a method to handle product species which arise due to polynomial terms. Consider a chemical reaction where two species react to form a third:

$$A + B \xrightarrow{k} C$$

The differential equation for C is

$$\frac{dC}{dt} = kAB.$$

One way to think about handling this equation is to introduce another species to the set A and B which represents the product of their values and changes in time during the integration, named $\mathbf{AB}$ here. Initially the field $\mathbf{AB}$ is generated by pointwise multiplication of the fields of its components, simply $\mathbf{AB}(x,y) = \mathbf{A}(x,y)\mathbf{B}(x,y)$ in scalar terms. A good basis is generated for $\mathbf{AB}$ just as for the other two, using the products of the inputs to the basis generator used for $\mathbf{A}$ and $\mathbf{B}$. Then the equation for C reduces as in the single-component linear case above to

$$\dot{c}_i = k\sum_j \dot{ab}_j\langle \mathbf{AB}_j, \mathbf{C}_i\rangle$$

(with '$\cdot$' representing time differentiation to save space). What is missing now is an equation to change the values of $\{ab_j\}$ over the course of the integration. The problem is, then, given $\{a_i\}$, $\{b_i\}$, $\{\dot{a}_i\}$, and $\{\dot{b}_i\}$, at a current point in time, compute $\{\dot{ab}_i\}$. There are two ways to do this.

First consider the "exact" method, where the derivatives $\{\dot{ab}_i\}$ will be used in the inner loop of **LSODE** or whatever solver to update the corresponding values of ab_i, treating the latter as just another reacting species. The Liebniz rule separates the product into its components

$$\frac{d\mathbf{AB}}{dt} = \mathbf{A}\frac{d\mathbf{B}}{dt} + \mathbf{B}\frac{d\mathbf{A}}{dt}$$

which can be expressed in terms of their basis elements

$$\frac{d\mathbf{AB}}{dt} = \left(\sum_i a_i\mathbf{A}_i\right)\left(\sum_j \dot{b}_j\mathbf{B}_j\right) + \left(\sum_j b_j\mathbf{B}_j\right)\left(\sum_i \dot{a}_i\mathbf{A}_i\right)$$

and combined

$$\frac{d\mathbf{AB}}{dt} = \sum_{ij}(a_i\dot{b}_j + \dot{a}_i b_j)(\mathbf{A}_i * \mathbf{B}_j)$$

where $\mathbf{A}_i * \mathbf{B}_j$ is the field which is the pointwise product of $\mathbf{A}_i$ and $\mathbf{B}_j$—the same operation that was used to create the initial product field **AB**. These product fields can be computed ahead of time, but will not need to be stored during the calculation, since their values will never be directly needed. To extract a certain coefficient $\dot{ab}_k$, the inner product of both sides is performed as usual, giving

$$\dot{ab}_k = \sum_{ij}(a_i\dot{b}_j + \dot{a}_i b_j)\ \langle \mathbf{AB}_k, \mathbf{A}_i * \mathbf{B}_j \rangle .$$

It is only this latter set of real-valued inner products which needs to be stored in memory at runtime, adding N^3 more words of memory for each product species, where N is the number of fields required in an expansion (assuming all species use the same expansion order, for convenience in presentation).

This exact method seems nice because it is exact, in the sense that there is no more error involved in a product species calculation than there is for a true reacting species. However there are many drawbacks, the worst being the addition of essentially another reactant to the system which will be integrated by LSODE. Also if expansions of various length are employed, there is additional overhead involved in figuring out the limits for the summations, and if dynamic expansion lengths are implemented, members of the inner product matrix may have to be recalculated, although one may suspect that this condition may be shared by other update methods. Memory requirements go up significantly only because of the lengthening of LSODE's list of variables, but not so much due to the tensor of inner products.

If the exact method is deemed too time-consuming or overly accurate in the face of slowly changing component fields for the product, one may consider moving the product species out of the inner loop. As with any approximation, care will have to be given to bounding the error somehow. The method is essentially the same as above except performed in a discrete way. Before an integration step, the initial values of a_i, b_i, and ab_i are saved, then afterwards the primed values are computed using

$$\mathbf{AB}' = \mathbf{A}' * \mathbf{B}'$$
$$\mathbf{AB} = \mathbf{A} * \mathbf{B}$$

since the goal is to keep **AB** reflective of the current values of its components. Subtracting and expanding gives:

$$\begin{aligned}
\mathbf{AB}' - \mathbf{AB} &= \mathbf{A}' * \mathbf{B}' - \mathbf{A} * \mathbf{B} \\
\sum_k \delta ab_k \mathbf{AB}_k &= \sum_{ij} a'_i b'_j \mathbf{A}_i * \mathbf{B}_j - \sum_{ij} a_i b_j \mathbf{A}_i * \mathbf{B}_j \\
\delta ab_k &= \sum_{ij} a'_i b'_j\ \langle \mathbf{AB}_k, \mathbf{A}_i * \mathbf{B}_j \rangle - \sum_{ij} a_i b_j\ \langle \mathbf{AB}_k, \mathbf{A}_i * \mathbf{B}_j \rangle \\
&= \sum_{ij}(a'_i b'_j - a_i b_j)\,\langle \mathbf{AB}_k, \mathbf{A}_i * \mathbf{B}_j \rangle
\end{aligned}$$

Again, the same extra coefficients are stored, with the same problems as before but without the extra overhead of another active species.

Note also with this second method that other species may lose accuracy if they depend on values of **AB** during their integration, since the product species is updated to reflect changes in its components only outside LSODE's integration loop. Forcing the integrator to take smaller time steps, *i.e.*, not integrating over a wide domain and returning intermediate values as requested, could alleviate this.

Clearly the tradeoff between these two methods is the classic one: speed versus accuracy. Since they are so closely related, though, one may be switched for the other at any time during the calculation. Once accuracy is lost, however, the only way to get it back is to do a complete re-multiplication of the two component fields plus another basis expansion. It is conceivable that the new product field could be projected onto the original basis fields to extract a reasonable improvement to the current coefficients, but as the rest of this entire procedure, deciding which technique to use and when to use it amounts to most of the theoretical work.

The two methods described above can be considered as variations on one common, and as it will be shown, preferred method for handling products of active species. First, notice again that the second method is simply a discretization of the "exact" method. Remembering the problem as stated,

$$A + B \xrightarrow{k} C$$

where bases have been generated for all three species as usual, to derive carefully this third way of handling products it is useful to consider the effect on only one point in the field.

$$\dot{\mathbf{C}}(x,y) = k\mathbf{A}(x,y)\ \mathbf{B}(x,y)$$

$$\left(\sum_i \dot{c}_i \mathbf{C}_i\right)(x,y) = k\left(\sum_j a_j \mathbf{A}_j\right)(x,y)\ \left(\sum_k b_k \mathbf{B}_k\right)(x,y)$$

$$= k\left(\sum_j a_j \mathbf{A}_j(x,y)\right)\left(\sum_k b_k \mathbf{B}_k(x,y)\right)$$

$$= k\sum_{jk} a_j b_k \mathbf{A}_j(x,y)\ \mathbf{B}_k(x,y)$$

$$= k\sum_{jk} a_j b_k (\mathbf{A}_j * \mathbf{B}_k)(x,y)$$

$$\sum_i \dot{c}_i \mathbf{C}_i = k\sum_{jk} a_j b_k \mathbf{A}_j * \mathbf{B}_k$$

$$\dot{c}_i = k\sum_{jk} a_j b_k \langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{C}_i \rangle$$

where the symbol '$*$' still indicates the pointwise product of two fields, *i.e.*,

$$(\mathbf{A} * \mathbf{B})(x,y) = \mathbf{A}(x,y)\ \mathbf{B}(x,y),$$

and the last two lines come from generalizing from scalars to fields, then using orthogonality to reduce to a simple expression for the derivative of each coefficient.

Now it will be shown that the first method is completely equivalent to the one just described, given a certain common condition on the fields. To recall, the first scheme is a two-step one involving an intermediate product species:

$$\dot{c}_i = k\sum_j \dot{ab}_j \langle \mathbf{AB}_j, \mathbf{C}_i \rangle$$

$$\dot{ab}_i = \sum_{jk} \left(a_j \dot{b}_k + \dot{a}_j b_k\right) \langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_i \rangle$$

Expressing the value of the coefficient ab_i at a time t requires an integration:

$$ab_i(t) = \int_0^t \sum_{jk} \left(a_j \dot{b}_k + \dot{a}_j b_k\right) \langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_i \rangle\ dt'$$

$$= \sum_{jk} \left(\int_0^t a_j \dot{b}_k\ dt' + \int_0^t \dot{a}_j b_k\ dt'\right) \langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_i \rangle$$

An integration by parts yields

$$\int_0^t a_j \dot{b}_k \, dt' = a_j(t')b_k(t') \Big|_0^t - \int_0^t \dot{a}_j b_k \, dt'$$

which cancels with the other term inside parentheses to give

$$\mathcal{a}b_i(t) = \sum_{jk} \big(a_j(t)b_k(t) - a_j(0)b_k(0)\big)\langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_i \rangle + \mathcal{a}b_i(0)$$

Notice that the constant of integration above may be expressed in a simpler form by considering how the initial condition of **AB** is related to those of **A** and **B**:

$$\mathbf{AB}(0) = \mathbf{A}(0) * \mathbf{B}(0)$$

$$\sum_i \mathcal{a}b_i(0)\mathbf{AB}_i = \sum_{jk} a_j(0)b_k(0)\mathbf{A}_j * \mathbf{B}_k$$

$$\mathcal{a}b_i(0) = \sum_{jk} a_j(0)b_k(0)\langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_i \rangle$$

This cancels with another term to yield

$$\mathcal{a}b_i(t) = \sum_{jk} a_j(t)b_k(t)\langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_i \rangle$$

Substituting this into the expression for $\dot{c}_i$ reduces the pair to the single equation

$$\dot{c}_i(t) = k \sum_{jkl} a_j(t)b_k(t)\langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_l \rangle \langle \mathbf{AB}_l, \mathbf{C}_i \rangle$$

The trick now will be to show that this expression is equivalent to the one given at the end of the previous paragraph.

The task is easily done by expressing most fields in terms of their expansions in **AB**'s fields. The accuracy of the results to follow hinges on the degree to which those fields span the space occupied by **A**, **B**, and **C**. This requirements arises often in the theoretical part of this work and is satisfied by the majority of models to which the whole technique is to be applied. To begin, both $\mathbf{A}_j * \mathbf{B}_k$ and $\mathbf{C}_i$ are written in terms of their expansions in **AB** fields:

$$\mathbf{A}_j * \mathbf{B}_k = \sum_m \langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_m \rangle \mathbf{AB}_m$$

$$\mathbf{C}_i = \sum_n \langle \mathbf{C}_i, \mathbf{AB}_n \rangle \mathbf{AB}_n$$

Notice that the limits to the sums are not given. This is meant to demonstrate the fact that higher accuracy will usually be achieved with more terms in the expansion, with equality only in the case that the expanded field lies completely within the space spanned by the expansion fields. Expansion of the second method for product species given above will lead to an equivalence between the two.

$$\begin{aligned}
\dot{c}_i(t) &= \sum_{jk} a_j(t)b_k(t)\langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{C}_i \rangle \\
&= \sum_{jk} a_j(t)b_k(t) \left\langle \sum_m \langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_m \rangle \mathbf{AB}_m, \sum_n \langle \mathbf{C}_i, \mathbf{AB}_n \rangle \mathbf{AB}_n \right\rangle \\
&= \sum_{jkl} a_j(t)b_k(t)\langle \mathbf{A}_j * \mathbf{B}_k, \mathbf{AB}_l \rangle \langle \mathbf{AB}_l, \mathbf{C}_i \rangle
\end{aligned}$$

The new method just derived is to be preferred over the original one since it reduces the number of active species in the inner integration loop, where most of the time and memory is required. The only drawback is that more projections must be calculated prior to the start of a simulation, but the original statement of the problem allows this added time to be disregarded when compared with the time consumed by expected multiple executions. Further, the accuracy of the two methods is identical as shown in the derivation.

Now that the proper method for handling quadratic nonlinearities has been presented, extensions to higher-order polynomials are given. For the arbitrary polynomial equation consisting of terms in the right-hand side of the form:

$$\frac{d\mathbf{A}}{dt} = \mathbf{A}^{\alpha}\mathbf{B}^{\beta}\mathbf{C}^{\gamma}\cdots,$$

each term is expanded separately to give the equations for time evolution of the coefficients of field **A**. The expansion proceeds as before, starting with substitution of the basis for each field.

$$\begin{aligned}
\sum_i \dot{a}_i \mathbf{A}_i &= \mathbf{A}^{\alpha}\mathbf{B}^{\beta}\mathbf{C}^{\gamma}\cdots \\
&= \left(\sum_j a_j \mathbf{A}_j\right)^{\alpha} * \left(\sum_k b_k \mathbf{B}_k\right)^{\beta} * \left(\sum_l b_l \mathbf{C}_l\right)^{\gamma} * \cdots \\
&= \left(\sum_{j_1} a_{j_1}\mathbf{A}_{j_1}\right) * \cdots * \left(\sum_{j_\alpha} a_{j_\alpha}\mathbf{A}_{j_\alpha}\right) * \left(\sum_{k_1} b_{k_1}\mathbf{B}_{k_1}\right) * \cdots * \left(\sum_{k_\beta} b_{k_\beta}\mathbf{B}_{k_\beta}\right) * \cdots \\
&= \sum_{j_1\cdots j_\alpha k_1\cdots k_\beta\cdots} a_{j_1}\cdots a_{j_\alpha} b_{k_1}\cdots b_{k_\beta}\cdots \mathbf{A}_{j_1} * \cdots * \mathbf{A}_{j_\alpha} * \mathbf{B}_{k_1} * \cdots * \mathbf{B}_{k_\beta} * \cdots \\
\dot{a}_i &= \sum_{j_1\cdots j_\alpha k_1\cdots k_\beta\cdots} a_{j_1}\cdots a_{j_\alpha} b_{k_1}\cdots b_{k_\beta}\cdots \langle \mathbf{A}_{j_1} * \cdots * \mathbf{A}_{j_\alpha} * \mathbf{B}_{k_1} * \cdots * \mathbf{B}_{k_\beta} * \cdots, \mathbf{A}_i\rangle
\end{aligned}$$

Clearly as the order of the polynomial term grows, this summation quickly becomes unmanageable. The depth of the summation is the order of the polynomial, and the extent of each limit is

$$\{\underbrace{N_A,\ldots,N_A}_{\alpha},\underbrace{N_B,\ldots,N_B}_{\beta},\ldots\}$$

for the example above, where N_A is the number of fields used for species A in the integration, and so on. However the number of scalar inner product terms that need to be stored is less than the expected

$$N_A N_A^{\alpha} N_B^{\beta}\cdots$$

due to symmetry of the inner product, and is only

$$N_A\binom{N_A+\alpha-1}{\alpha}\binom{N_B+\beta-1}{\beta}\cdots *.$$

Presumably, though, there are not so many polynomial terms of such complexity in actual numerical models. Regardless, such details as multiple species and complicated polynomial terms will be handled automatically by the mechanism parser described in a future section.

---

* A nice way to see this is to view an entry of a d-dimensional square matrix of size N as a list of numbers from 1 to N. An S_d equivalence class of entries is labeled by d non-decreasing numbers from 1 to N. A specification of such an entry can be done using a finite state machine which has one element of memory, one state, and two operations: "Increase the value" and "Output the value," with the initial state of no output and a value of 1. For the case $d = 3$ and $N \geq 8$, some valid sequences of operation

9.1.1 Non-polynomial nonlinearities

Arbitrary polynomials are easily handled by the product mechanism described above. Other, non-polynomial functions present a further level of difficulty. For illustrative purposes, the equation

$$\frac{d\mathbf{A}}{dt} = e^{\mathbf{A}}$$

is considered, which is again an abbreviation for the set of equations over a spatial domain with a single scalar dependent variable. Certainly no real application would ever demand the integration of this unstable problem, but the simple form will be useful in describing the general approach. Attempting to expand out $\mathbf{A}$ in terms of its basis gives

$$\sum_i \dot{a}_i \mathbf{A}_i = \exp\left(\sum_j a_j \mathbf{A}_j\right).$$

At this point, previous derivations have relied on the inner product to separate the spatially varying field terms away from the time-variant scalar coefficients to give evolution equations for the coefficients, thus removing any full-field calculations from the temporal evolution procedure. Due to the presence of the exponential, the inner product operation applies not only to the field term and the equation does not separate. One approach, which will work for all types of non-linearities, is to expand into polynomials via a Taylor series. For this example, the series gives

$$\sum_i \dot{a}_i \mathbf{A}_i = \mathbf{I} + \sum_j a_j \mathbf{A}_j + \frac{1}{2!}\left(\sum_j a_j \mathbf{A}_j\right)^2 + \frac{1}{3!}\left(\sum_j a_j \mathbf{A}_j\right)^3 + \cdots$$

which reduces after expanding out the summations to

$$\sum_i \dot{a}_i \mathbf{A}_i = \mathbf{I} + \sum_j a_j \mathbf{A}_j + \frac{1}{2!}\sum_{jk} a_j a_k \mathbf{A}_j \mathbf{A}_k + \frac{1}{3!}\sum_{jkl} a_j a_k a_l \mathbf{A}_j \mathbf{A}_k \mathbf{A}_l + \cdots$$

giving the coefficient evolution equations for $\{a_i\}$:

$$\dot{a}_i = \langle \mathbf{I}, \mathbf{A}_i \rangle + \sum_j a_j \langle \mathbf{A}_j, \mathbf{A}_i \rangle + \frac{1}{2!}\sum_{jk} a_j a_k \langle \mathbf{A}_j \mathbf{A}_k, \mathbf{A}_i \rangle + \frac{1}{3!}\sum_{jkl} a_j a_k a_l \langle \mathbf{A}_j \mathbf{A}_k \mathbf{A}_l, \mathbf{A}_i \rangle + \cdots.$$

This final equation marks the introduction of the term $\langle \mathbf{I}, \mathbf{A}_i \rangle$ which is the solution to the differential equation $\frac{dA}{dt} = 1$, the leading term in the exponential.

While satisfying in the presentation, the above may be completely useless in an actual calculation due to the grand errors which may be introduced by such an expansion. The final equation assumes implicitly that $\sum_j a_j \mathbf{A}_j$ is near zero for all time, or equivalently the $\{a_j\}$ are near zero. One small improvement is to center the Taylor expansion about the actual values which will be encountered in

---

with the resulting matrix entries are:

$$\begin{array}{ll} \text{OOO} & \Longrightarrow \{1,1,1\} \\ \text{IIIOOO} & \Longrightarrow \{4,4,4\} \\ \text{IOIIIOIIO} & \Longrightarrow \{2,6,8\} \end{array}$$

It can be seen that operation O is invoked d times, and operation I no more than $N-1$ times, leading to $\binom{N-1+d}{d}$ possible entries under the equivalence class. This is called a "stars and bars" [4] argument.

the integration. Suppose that these central values have been discovered, and written as $\{a_j^0\}$. Then the expansion becomes

$$\sum_i \dot{a}_i \mathbf{A}_i = \exp\left(\sum_j a_j^0 \mathbf{A}_j\right)\left[\mathbf{I} + \sum_j (a_j - a_j^0)\mathbf{A}_j + \frac{1}{2!}\left(\sum_j (a_j - a_j^0)\mathbf{A}_j\right)^2 + \cdots\right]$$

which yields the evolution equations:

$$\dot{a}_i = \exp\left(\sum_j a_j^0 \mathbf{A}_j\right)\left[\langle \mathbf{I}, \mathbf{A}_i\rangle + \sum_j (a_j - a_j^0)\langle \mathbf{A}_j, \mathbf{A}_i\rangle + \frac{1}{2!}\sum_{jk}(a_j - a_j^0)(a_k - a_k^0)\langle \mathbf{A}_j\mathbf{A}_k, \mathbf{A}_i\rangle + \cdots\right].$$

The first factor on the right hand side is a constant, as the central values of the coefficients are chosen to be fixed over the time domain of the integration, and can be precalculated along with the inner products. If the integration is characterized by a few distinct ranges in time, multiple sets of $\{a_j^0\}$ can be used, choosing the appropriate set depending on the time. Of course wide changes of parameter values across an integration are not uncommon and may render this variation as useless as the first zero-centered expansion.

A more clever, but less generally applicable, method of handling such an equation is to transform the variables so as to coerce the nonlinearity into a different form. For the example above, pick a new field $\mathbf{B}$ to represent the term $\exp(\mathbf{A})$, then

$$\frac{d\mathbf{B}}{dt} = \frac{de^{\mathbf{A}}}{dt} = e^{\mathbf{A}}\frac{d\mathbf{A}}{dt} = e^{\mathbf{A}}e^{\mathbf{A}} = \mathbf{B}^2$$

which reduces easily (and exactly) to

$$\dot{b}_i = \sum_{jk} b_j b_k \langle \mathbf{B}_j\mathbf{B}_k, \mathbf{B}_i\rangle.$$

The whole problem is converted to work in this exponential space by first setting the initial condition and example fields pointwise by $\mathbf{B}(x,y) = \exp\big(\mathbf{A}(x,y)\big)$, then printing out the results as $\mathbf{A}(x,y) = \log \mathbf{B}(x,y)$. Unfortunately, for a problem such as

$$\frac{d\mathbf{A}}{dt} = e^{\mathbf{A}} + \mathbf{A},$$

this transformation may introduce more inaccuracy than it removes as now some sort of expansion for $\log \mathbf{B}$ must be included. Section 9.4 discusses more examples of this sort of transformation.

Other types of nonlinearities face the same problems as the exponential discussed above. The common trigonometric transcendentals may be reduced to functions of exponentials and treated that way, or considered in their own right. One special functional form are the rational functions. For example, the equation

$$\frac{d\mathbf{A}}{dt} = \frac{1}{\mathbf{A}}$$

does not reduce without expansion of the right-hand side, but transformation by logarithms works, as does sending $\mathbf{A} \to \mathbf{A}^2$ in which case the equation reduces to

$$\frac{d\mathbf{A}^2}{dt} = 2\mathbf{A}\frac{d\mathbf{A}}{dt} = 2\mathbf{A}\frac{1}{\mathbf{A}} = 2.$$

General rational functions will require an expansion into polynomials when no such trivial transformation of the problem exists.

Considering the focus of this thesis on differential equations generated by mass-action chemistry and the usual transport terms generated by convection and diffusion, the inability to handle these non-polynomial functions is not too relevant. Hopefully most other models will have only limited use of such functions as well.

In some cases, the evaluation of such nonlinearities in the coefficient space takes entirely too long to allow for feasible execution times. It is possible in these circumstances to expand out the coefficients and basis functions to obtain the values of the system in physical space, evaluate the nonlinear terms there, then transform the results back into coefficient space using a projection, as illustrated by:

$$
\begin{array}{ll}
\{a_i\} & \\
\Big\downarrow & \text{basis superposition} \\
\mathbf{A}(x,y) & \\
\Big\downarrow & \text{physical space evaluation using original equations} \\
\mathbf{A}'(x,y) & \\
\Big\downarrow & \text{basis projection} \\
\{a_i'\} &
\end{array}
$$

Then the coefficients are updated by summing the results of the above along with the coefficient adjustments produced by evaluating any other terms in coefficient space. The major drawback to this operation is that it too will be quite slow due to the superposition and projections required at every function evaluation.

9.1.2 Inhomogeneities

The presence of scalar terms in the differential equations adds another small complication. First, a modification of the differential equation which starts this chapter will illustrate the procedure:

$$\frac{d\mathbf{A}(x,y)}{dt} = \alpha(t)\mathbf{A}(x,y) + \beta(t).$$

The equation converts as before with substitution of the field expansion for $\mathbf{A}$ into

$$\sum_i \frac{da_i}{dt}\mathbf{A}_i(x,y) = \alpha(t)\sum_i a_i\mathbf{A}_i(x,y) + \beta(t)$$

only applying the inner product of both sides with $\mathbf{A}_i$ gives

$$\sum_j \frac{da_j}{dt}\langle \mathbf{A}_i(x,y), \mathbf{A}_j(x,y)\rangle = \alpha(t)\sum_j a_j\langle \mathbf{A}_i(x,y), \mathbf{A}_j(x,y)\rangle + \beta(t)\langle \mathbf{I}, \mathbf{A}_i(x,y)\rangle$$

and finally

$$\frac{da_i}{dt} = \alpha(t)a_i + \beta(t)\langle \mathbf{I}, \mathbf{A}_i(x,y)\rangle.$$

The new term includes just another scalar constant which can be generated before simulation time along with the product field scalars, and execution of this equation with a non-homogenous term is not too much more costly than that of the original.

9.2 Partial differential equations

To enable the solution of partial differential equations, it is necessary to generate a difference scheme which is a good approximation to the continuous system. There are two ways to consider this

difference scheme. First, a particular finite difference scheme is constructed, then examined to determine which new fields will be necessary. This discussion will be more concrete if a specific problem is considered: one-dimensional diffusion. The governing equation is

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

which can be discretized using the standard forward-time central-space scheme [128],

$$\frac{u_m^{n+1} - u_m^n}{\Delta t} = D\frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{(\Delta x)^2}.$$

This formula, used directly inside a field-based integration, would require pointwise adjustments of the current concentration field itself, instead of just changes in the coefficients of the expansion, an unacceptable result. However, there is an equivalent way to express this scheme which is as accurate than the difference formula above. First, consider the individual concentration values as a vector which is updated by applying a matrix to the current values. Thus,

$$\underset{\sim}{u}^{n+1} = \underset{\sim}{u}^n + \alpha \begin{pmatrix} -2 & & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & & \ddots & & & \\ & & & 1 & -2 & 1 & \\ & & & & & -2 & \end{pmatrix} \underset{\sim}{u}^n, \qquad \left(\text{where } \alpha = \frac{D\Delta t}{(\Delta x)^2}\right)$$

depending on the choice of boundary conditions. M is used to denote the matrix above. As usual, writing the original and updated concentration fields in terms of their expansions yields a formula for updating only the coefficients:

$$\sum_i u_i^{n+1}\mathbf{U}_i = \sum_j u_j^n(I + \alpha M)\mathbf{U}_j$$

$$u_i^{n+1} = \sum_j u_j^n \langle (I + \alpha M)\mathbf{U}_j, \mathbf{U}_i \rangle$$

The last term in angle brackets is just another projection which can be calculated before the simulation starts, by forming the matrix $I + \alpha M$ and doing a matrix-vector multiply for each basis element (column vector) $\mathbf{U}_j$ to form $(I + \alpha M)\mathbf{U}_j$ then calculating the projection. Other methods are implemented as easily as the explicit form. For instance, a fully implicit scheme reduces to $\underset{\sim}{u}^{n+1} = (I - \alpha M)^{-1}\underset{\sim}{u}^n$, and Crank-Nicholson is just the two combined: $\underset{\sim}{u}^{n+1} = (I - \frac{\alpha}{2}M)^{-1}(I + \frac{\alpha}{2}M)\underset{\sim}{u}^n$.

Alternatively, the coefficient evolution equation can be developed directly, before a choice of discretization of the differential fields is performed. Falling back on more common notation, this can be shown using the simplest partial differential equation:

$$\frac{\partial \mathbf{A}}{\partial t} = \frac{\partial \mathbf{A}}{\partial x}.$$

Writing $\mathbf{A}$ in terms of its basis as always and substituting into the equation gives

$$\frac{\partial}{\partial t}\left(\sum_i a_i(t)\mathbf{A}_i(x)\right) = \frac{\partial}{\partial x}\left(\sum_j a_j(t)\mathbf{A}_j(x)\right)$$

where the functional dependence of the variables has been included to emphasize the point. The basis is fixed for all time across an integration while the coefficients do not change over the spatial

domain since the basis functions have, as always, global support. Using this observation to move the differentiation operators onto only the affected functions gives

$$\sum_i \frac{da_i}{dt}(t)\mathbf{A}_i(x) = \sum_j a_j(t)\frac{d\mathbf{A}_j}{dx}(x)$$

and performing an inner product with $\mathbf{A}_i$ produces the coefficient evolution equation:

$$\dot{a}_i = \sum_j a_j \left\langle \frac{d\mathbf{A}_j}{dx}, \mathbf{A}_i \right\rangle .$$

The only complication, then, is additional storage of inner product terms involving spatial differentiation. Here also is where the choice of a differentiation scheme comes into play as in the previous scheme. Conceptually these fields can be treated just like additional independent variables for which no evolution equations need be generated, though. In practice a basis for $\mathbf{A}$ is generated, then each field in that basis is numerically differentiated to generate the set $\left\{ \frac{d\mathbf{A}_j}{dx} \right\}$, which is used in calculating the inner products.

Although this second view is algebraically cleaner, it has the possibly unfortunate side effect of decoupling the temporal and spatial discretization schemes, making any combined convergence or stability arguments impossible. As with any discretized scheme, all the standard considerations about these issues carry over into basis space integration, and are widely discussed, for example in [128]. In simple advection, for instance, care must be taken that the Courant limit, which relates the time step and grid size through advection velocity, remain satisfied. Hence knowledge of the differentiation scheme used to generate the fields $\left\{ \frac{d\mathbf{A}_i}{dx} \right\}$ must be available when deciding the time step to use during an integration. It is not obvious now how the choice of a difference scheme will affect the results of a basis space integration scheme in general, and if the choice dictated for the grid-based simulation is also the correct one for a field-based simulation.

9.3 Boundary conditions

Up to this point, boundary conditions have not been explicitly considered. The major reason is that for many problems of interest the boundaries remain fixed at some value so that all the basis functions automatically satisfy the boundary conditions. For the case of either Neumann or mixed conditions at the boundary, or where the boundary conditions vary as a function of simulation time, it will be necessary to augment the solution procedure at least, and possibly the fields. The use of a tau method in spectral methods is one likely candidate scheme for handling this situation, but will not be considered in this thesis.

9.4 Bulk motion transforms

Frequently systems involving fluid flow can be characterized by a simple bulk motion. Simple flow in a tube may feature a flat or parabolic velocity profile on top of which is diffusion and other smaller scale transport effects. Combustion in a whirling reactor starts with a carrier gas rotating around the central axis into which fuel is added. A common test case for the transport calculation in atmospheric chemistry models is the rotating plume problem, as traditional solvers are usually based in an x-y coordinate system in which motion in the θ direction leads to the introduction of much numerically generated diffusion. A sharp plume will spread out as it is rotated by standard solvers. This section describes this problem and the general case of how to handle systems which

are governed by large scale motion, through the transformation into a coordinate system in which that motion does not exist.

The derivation of the transport of a passive scalar in a purely rotational flow field follows from a shell balance in rectangular coordinates giving

$$\frac{\partial c}{\partial t} + \nabla \cdot (c\mathbf{v}) = 0.$$

In radial coordinates, ∇ takes the form

$$\nabla \cdot \mathbf{f} = \frac{1}{r}\frac{\partial}{\partial r}(rf_r) + \frac{1}{r}\frac{\partial f_\theta}{\partial \theta} + \frac{\partial f_z}{\partial z}.$$

In two dimensions the radial convection equation is then

$$\frac{\partial c}{\partial t} + \frac{\partial (cv_r)}{\partial r} + \frac{cv_r}{r} + \frac{1}{r}\frac{\partial (cv_\theta)}{\partial \theta} = 0.$$

Making the assumptions suitable for this problem, that $\mathbf{v}$ is not a function of θ and that $v_r = 0$ so that there is no mixing between the different lanes (radial positions), leaves only a choice for the form of $v_\theta(r)$. Letting that be Mr will keep initial conditions unchanged except for a global rotation at rate M in time. Now the equation becomes

$$\frac{\partial c}{\partial t} + M\frac{\partial c}{\partial \theta} = 0$$

with initial condition

$$c(r, \theta, 0) = c_0(r, \theta).$$

A boundary condition will not be needed if θ is taken to be in $\mathbb{S}^1$ instead of $[0, 2\pi) \subset \mathbb{R}^1$. The solution (by inspection) to this partial differential equation is

$$c(r, \theta, t) = c_0(r, \theta - Mt).$$

A nice function to use for testing the various integrators of this system is the Gaussian bump

$$c_0(r, \theta) = h \exp\left[-\left(r^2 + r_0^2 - 2rr_0\cos(\theta - \theta_0)\right)/\sigma^2\right],$$

centered at (r_0, θ_0) with spread controlled by σ and maximum height h. Another possible function to consider is the cosine bump, reported in the paper evaluating advection performance of the CALGRID [87] model:

$$c_0(x, y) = \begin{cases} \frac{1}{2}\left(1 + \cos\frac{\pi R}{R_b}\right) & \text{for } R \le R_b \\ 0 & \text{for } R \ge R_b \end{cases}$$

where

$$R = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

and (x_0, y_0) is the initial location of the peak. Note the puff has a radius at its base of R_b.

To transform the equation in preparation for a basis space integration, use of the structure of this equation will be employed by first moving to a frame of reference in which there is no rotation. Let

$$\hat{c}(r, \gamma, t) = c(r, \theta - Mt, t)$$

then

$$\frac{\partial c}{\partial t} = \frac{\partial \hat{c}}{\partial \gamma}\frac{\partial \gamma}{\partial t} + \frac{\partial \hat{c}}{\partial t}\frac{\partial t}{\partial t} = -M\frac{\partial \hat{c}}{\partial \gamma} + \frac{\partial \hat{c}}{\partial t}$$

$$\frac{\partial c}{\partial \theta} = \frac{\partial \hat{c}}{\partial \gamma}\frac{\partial \gamma}{\partial \theta} + \frac{\partial \hat{c}}{\partial t}\frac{\partial t}{\partial \theta} = \frac{\partial \hat{c}}{\partial \gamma}$$

giving the equation for $\hat{c}$

$$\frac{\partial \hat{c}}{\partial t} = 0$$

which has trivial solution (satisfying the initial condition)

$$\hat{c}(r, \gamma, t) = c_0(r, \theta - Mt, t).$$

Nevertheless, transformation of this system via substitution of an expansion of the concentration field in terms of its eigenfunctions yields another integrable system which will give *exactly* the correct result.

Since transformation of the simple rotating plume convection problem results in such a trivial solution, another term is added to bring it closer to more practical integration problems. The equation

$$\frac{\partial c}{\partial t} + M\frac{\partial c}{\partial \theta} = \alpha\beta\cos(\alpha t)\frac{\partial c}{\partial r}$$

is transformed via $\gamma = \theta - Mt$ since it is suspected that the structure of the solution will again have bulk properties of a rotating field (for proper ratios of α, β, and γ) giving

$$\frac{\partial \hat{c}}{\partial t} = \alpha\beta\cos(\alpha t)\frac{\partial \hat{c}}{\partial r}.$$

The analytic solution is

$$\hat{c}(r, t) = \hat{c}_0(r + \beta\sin(\alpha t), t).$$

A numerical solution by finite differences in cylindrical coordinates was implemented first to provide timing information for an exact solution, using upwind differencing in both the radial and angular coordinates. (Implementing this for the radial coordinate requires changing the integration scheme depending on the sign of $\cos(\alpha t)$, and may not be worth the effort if the amplitude β is small.) The scheme is

$$\frac{c^{n+1}_{l,m} - c^n_{l,m}}{\Delta t} = -M\frac{c^n_{l,m} - c^n_{l,m-1}}{\Delta\theta} + \alpha\beta\cos(\alpha t)\frac{c^n_{l,m} - c^n_{l-1,m}}{\Delta r}$$

where l, m and n index the r, θ and t coordinates respectively. Finding the stepsize parameters which give stability as detailed in Strikwerda [128] involves replacing $c^n_{l,m}$ with $g^n e^{il\psi} e^{im\phi}$ in the difference formula, which rearranged gives:

$$g = 1 - \frac{M\Delta t}{\Delta\theta}\left(1 - e^{-i\phi}\right) + \frac{\alpha\beta\cos(\alpha t)\Delta t}{\Delta r}\left(1 - e^{-i\psi}\right).$$

For the scheme to be stable, this amplification factor must have absolute value smaller than one, which means

$$\left|\frac{M\Delta t}{\Delta\theta}\left(1 - e^{-i\phi}\right)\right| > \left|\frac{\alpha\beta\cos(\alpha t)\Delta t}{\Delta r}\left(1 - e^{-i\psi}\right)\right|.$$

Simplification by eliminating the exponential and cosine factors (which are absolutely bounded by one) and noticing that all other parameters are strictly positive gives

$$\frac{\Delta r}{\Delta\theta} > \frac{\alpha\beta}{M}$$

which must be satisfied for stability to be *achievable*. This condition says nothing about further restrictions on the stepsize Δt.

In Cartesian coordinates, the pure rotational convection problem is

$$\frac{\partial c}{\partial t} + \frac{\partial(cv_x)}{\partial x} + \frac{\partial(cv_y)}{\partial y} = 0$$

and the wind field $v_\theta(r) = Mr$ becomes

$$v_x = -\sin\theta\, v_\theta = -My, \qquad v_y = \cos\theta\, v_\theta = Mx,$$

giving

$$\frac{\partial c}{\partial t} - My\frac{\partial c}{\partial x} + Mx\frac{\partial c}{\partial y} = 0.$$

The generalized form requires transformation of the term $\dfrac{\partial c}{\partial r}$ as follows:

$$\begin{aligned}\frac{\partial c}{\partial r} &= \frac{\partial \hat{c}}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial \hat{c}}{\partial y}\frac{\partial y}{\partial r} \\ &= \frac{\partial \hat{c}}{\partial x}\cos\theta + \frac{\partial \hat{c}}{\partial y}\sin\theta \\ &= \frac{x}{r}\frac{\partial \hat{c}}{\partial x} + \frac{y}{r}\frac{\partial \hat{c}}{\partial y}\end{aligned}$$

leading to the complete form

$$\frac{\partial c}{\partial t} - My\frac{\partial c}{\partial x} + Mx\frac{\partial c}{\partial y} = \frac{\alpha\beta\cos(\alpha t)}{\sqrt{x^2+y^2}}\left(x\frac{\partial \hat{c}}{\partial x} + y\frac{\partial \hat{c}}{\partial y}\right).$$

(At the grid origin, of course, the right hand side must be taken as zero.)

Conversion of rotation in Cartesian coordinates into rotating space, that is, applying a rotation of $-Mt$ to the above equation involves finding

$$\hat{c}(\hat{x}, \hat{y}, t) = c(x, y, t)$$

where

$$(\hat{x}, \hat{y}) = \mathrm{Rot}(-Mt) \circ (x, y)^T \quad \text{and} \quad (x, y) = \mathrm{Rot}(Mt) \circ (\hat{x}, \hat{y})^T.$$

Generalized two-dimensional rotation of an angle α about the origin (counterclockwise motion is positive as always) is

$$\mathrm{Rot}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

giving

$$\mathrm{Rot}(-Mt) = \begin{pmatrix} \cos Mt & \sin Mt \\ -\sin Mt & \cos Mt \end{pmatrix} \quad \text{and} \quad \mathrm{Rot}(Mt) = \begin{pmatrix} \cos Mt & -\sin Mt \\ \sin Mt & \cos Mt \end{pmatrix}.$$

So the variables transform as

$$\begin{array}{l|l} \hat{x} = x\cos Mt + y\sin Mt & x = \hat{x}\cos Mt - \hat{y}\sin Mt \\ \hat{y} = -x\sin Mt + y\cos Mt & y = \hat{x}\sin Mt + \hat{y}\cos Mt \end{array}$$

As before, transform each variable individually.

$$\begin{aligned}
\frac{\partial c}{\partial t} &= \frac{\partial \hat{c}}{\partial \hat{t}}\frac{\partial \hat{t}}{\partial t} + \frac{\partial \hat{c}}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial t} + \frac{\partial \hat{c}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial t} \\
&= \frac{\partial \hat{c}}{\partial \hat{t}} + \left(-Mx\sin Mt + My\cos Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} + \left(-Mx\cos Mt - My\sin Mt\right)\frac{\partial \hat{c}}{\partial \hat{y}} \\
&= \frac{\partial \hat{c}}{\partial \hat{t}} + \left(-M\left(\hat{x}\cos Mt - \hat{y}\sin Mt\right)\sin Mt + M\left(\hat{x}\sin Mt + \hat{y}\cos Mt\right)\cos Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} \\
&\quad + \left(-M\left(\hat{x}\cos Mt - \hat{y}\sin Mt\right)\cos Mt - M\left(\hat{x}\sin Mt + \hat{y}\cos Mt\right)\sin Mt\right)\frac{\partial \hat{c}}{\partial \hat{y}} \\
&= \frac{\partial \hat{c}}{\partial \hat{t}} + \left(-M\hat{x}\sin Mt\cos Mt + M\hat{y}\sin^2 Mt + M\hat{x}\sin Mt\cos Mt + M\hat{y}\cos^2 Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} \\
&\quad + \left(-M\hat{x}\cos^2 Mt + M\hat{y}\sin Mt\cos Mt - M\hat{x}\sin^2 Mt - M\hat{y}\sin Mt\cos Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} \\
&= \frac{\partial \hat{c}}{\partial \hat{t}} + \left(M\hat{y}\sin^2 Mt + M\hat{y}\cos^2 Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} + \left(-M\hat{x}\cos^2 Mt - M\hat{x}\sin^2 Mt\right)\frac{\partial \hat{c}}{\partial \hat{y}} \\
&= \frac{\partial \hat{c}}{\partial \hat{t}} + M\hat{y}\frac{\partial \hat{c}}{\partial \hat{x}} - M\hat{x}\frac{\partial \hat{c}}{\partial \hat{y}}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial c}{\partial x} &= \frac{\partial \hat{c}}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + \frac{\partial \hat{c}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial x} \\
&= \cos Mt\frac{\partial \hat{c}}{\partial \hat{x}} - \sin Mt\frac{\partial \hat{c}}{\partial \hat{y}} \\
\frac{\partial c}{\partial y} &= \frac{\partial \hat{c}}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial y} + \frac{\partial \hat{c}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y} \\
&= \sin Mt\frac{\partial \hat{c}}{\partial \hat{x}} + \cos Mt\frac{\partial \hat{c}}{\partial \hat{y}}
\end{aligned}$$

$$\begin{aligned}
My\frac{\partial c}{\partial x} &= M\left(\hat{x}\sin Mt + \hat{y}\cos Mt\right)\left(\cos Mt\frac{\partial \hat{c}}{\partial \hat{x}} - \sin Mt\frac{\partial \hat{c}}{\partial \hat{y}}\right) \\
&= M\hat{x}\sin Mt\cos Mt\frac{\partial \hat{c}}{\partial \hat{x}} - M\hat{x}\sin^2 Mt\frac{\partial \hat{c}}{\partial \hat{y}} + M\hat{y}\cos^2 Mt\frac{\partial \hat{c}}{\partial \hat{x}} - M\hat{y}\sin Mt\cos Mt\frac{\partial \hat{c}}{\partial \hat{y}} \\
Mx\frac{\partial c}{\partial y} &= M\left(\hat{x}\cos Mt - \hat{y}\sin Mt\right)\left(\sin Mt\frac{\partial \hat{c}}{\partial \hat{x}} + \cos Mt\frac{\partial \hat{c}}{\partial \hat{y}}\right) \\
&= M\hat{x}\sin Mt\cos Mt\frac{\partial \hat{c}}{\partial \hat{x}} + M\hat{x}\cos^2 Mt\frac{\partial \hat{c}}{\partial \hat{y}} - M\hat{y}\sin^2 Mt\frac{\partial \hat{c}}{\partial \hat{x}} - M\hat{y}\sin Mt\cos Mt\frac{\partial \hat{c}}{\partial \hat{y}}
\end{aligned}$$

$$\begin{aligned}
-My\frac{\partial c}{\partial x} + Mx\frac{\partial c}{\partial y} &= \left(-M\hat{x}\sin Mt\cos Mt - M\hat{y}\cos^2 Mt + M\hat{x}\sin Mt\cos Mt - M\hat{y}\sin^2 Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} \\
&\quad + \left(M\hat{x}\sin^2 Mt + M\hat{y}\sin Mt\cos Mt + M\hat{x}\cos^2 Mt - M\hat{y}\sin Mt\cos Mt\right)\frac{\partial \hat{c}}{\partial \hat{y}} \\
&= \left(-M\hat{y}\cos^2 Mt - M\hat{y}\sin^2 Mt\right)\frac{\partial \hat{c}}{\partial \hat{x}} + \left(M\hat{x}\sin^2 Mt + M\hat{x}\cos^2 Mt\right)\frac{\partial \hat{c}}{\partial \hat{y}} \\
&= -M\hat{y}\frac{\partial \hat{c}}{\partial \hat{x}} + M\hat{x}\frac{\partial \hat{c}}{\partial \hat{y}}
\end{aligned}$$

It is obvious that the right-hand side, since it has essentially only r dependence, is invariant under this transformation, but it may be worked out in a similar manner as above. Plugging all the above into the original equation in Cartesian coordinates gives

$$\frac{\partial \hat{c}}{\partial t} + M\hat{y}\frac{\partial \hat{c}}{\partial \hat{x}} - M\hat{x}\frac{\partial \hat{c}}{\partial \hat{y}} - M\hat{y}\frac{\partial \hat{c}}{\partial \hat{x}} + M\hat{x}\frac{\partial \hat{c}}{\partial \hat{y}} = \frac{\alpha\beta\cos(\alpha t)}{\sqrt{\hat{x}^2+\hat{y}^2}}\left(\hat{x}\frac{\partial \hat{c}}{\partial \hat{x}} + \hat{y}\frac{\partial \hat{c}}{\partial \hat{y}}\right)$$

or

$$\frac{\partial \hat{c}}{\partial t} = \frac{\alpha\beta\cos(\alpha t)}{\sqrt{\hat{x}^2+\hat{y}^2}}\left(\hat{x}\frac{\partial \hat{c}}{\partial \hat{x}} + \hat{y}\frac{\partial \hat{c}}{\partial \hat{y}}\right),$$

as it should.

The different schemes for integrating the convection equation were tried and compared under changes of the value of parameter α, the frequency of oscillatory motion in the radial direction expected in the solution. As has been done in the past [87], peak height retention after one full rotation is used as the metric for evaluating the integrators, but other features are also reported below. The five different schemes are listed in Table 13. The schemes **quad** and **pdetwo** differ in that the former is a naïve implementation, using a constant time-step Euler algorithm, while the latter uses the full GEARB [64] package to solve the derived ODEs.

Name	Coordinate system	Space
rotate	Cylindrical	Grid
quad	Cartesian	Grid
pdetwo	Cartesian	Grid
brotate	Cylindrical	Empirical basis
bquad	Cartesian	Empirical basis

Table 13. Names and characteristics of the five different integration schemes used to study the rotating plume convection problem.

In general for all cases, **rotate** can be made to produce the most accurate results with an appropriately small grid and time step. The word "accurate" is used loosely here since the goal of this section is not to analyze exactly the errors of any one of these methods, but just to show the practicality of the transformation. **quad** suffers from the first common problem of convective solvers, the introduction of artificial diffusion, which reduces a Gaussian to less than 10% of its initial height after only one rotation, a problem which is exacerbated with decreasing step size. **pdetwo** accounts for this diffusion and can maintain a peak quite well, but the GEAR solver within has no knowledge of the Courant stability limit for this particular problem and loses information by taking steps in time which allow parcels of air completely to cross grid cells, leading to the characteristic trailing ripples. An example of the development of ripples after one rotation is shown in the top of Figure 32, with the corresponding correct version as produced by **rotate** or either of the basis space methods below it.

A straightforward basis space implementation was written, and works with as small an error tolerance as desired, with the errors resulting from an incomplete basis as is usual, and as opposed to from any approximations in the algorithm. One minor variation that was necessary for this section is that for a cylindrical grid space a different definition of inner product must be used to account for the variation of grid cell size with radial position. The definition

$$\langle f, g\rangle = \int_0^1 \int_0^{2\pi} f(r,\theta)g(r,\theta)\, r\, dr\, d\theta$$

is required. The process for actually implementing a full basis space integration run involves several non-trivial steps: generating previous runs, creating the basis in cylindrical coordinates, and estimating final errors; a flowchart of the processes used is shown in Figure 33.

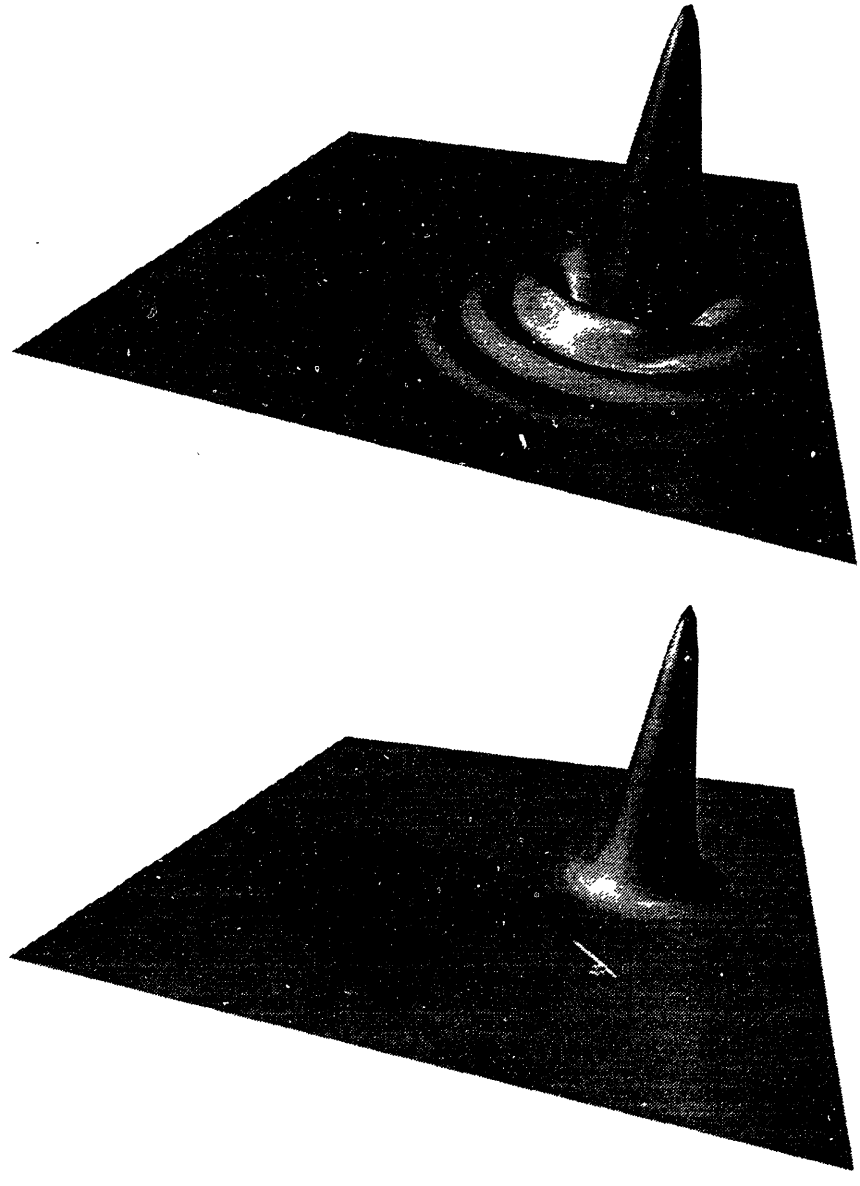


Figure 32. Formation of trailing ripples after one counter-clockwise rotation by pdetwo, above, with the ideal solution below.

The top central column creates the `derivs()` routine which evaluates the right-hand side of the convection equation being solved by generating a basis, evaluating the spatial differences, and constructing projection coefficients:

$$\left\{\left\langle \mathbf{C}_i, \frac{\partial \mathbf{C}_j}{\partial r} \right\rangle, \left\langle \mathbf{C}_i, \frac{\partial \mathbf{C}_j}{\partial \theta} \right\rangle\right\}_{i,j=1}^{N}.$$

Centered differences were used in the radial direction, as it seems to provide better accuracy, while backward differences were used in the θ-direction to get the benefits of upwinding. The input fields to the basis generator were not actually created from previous runs, as is the *modus operandus* of basis space integration, but rather since the equation has an analytic solution, it was used. Probable

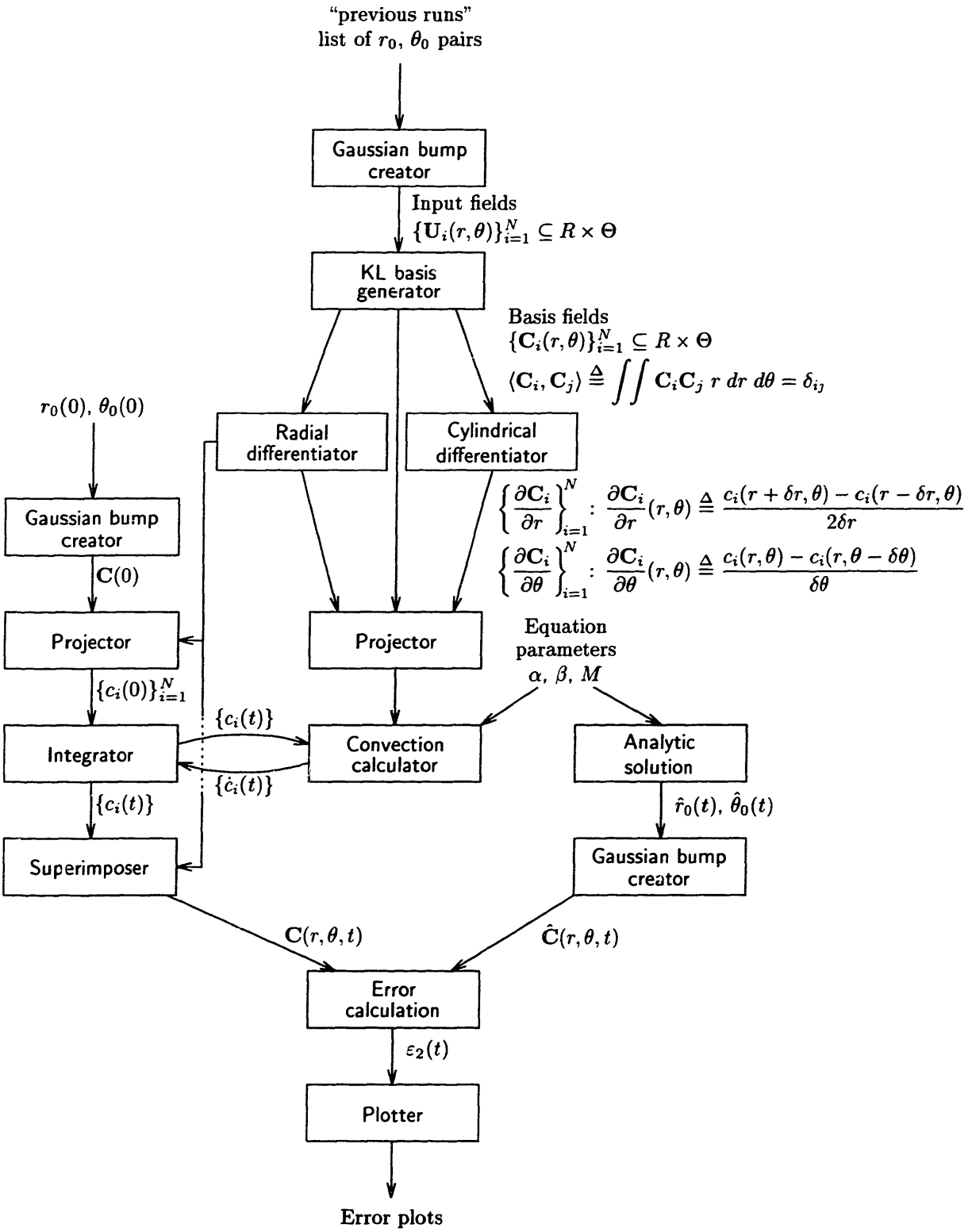


Figure 33. Flowchart of steps involved in performing and evaluating a basis space integration in cylindrical coordinates for the rotating plume problem.

values for the location of the non-diffusing bump were listed and fed to a utility which creates (r, θ) fields suitable for basis generation. This has the advantage of using the best solution available, and is equivalent to writing a traditional integrator which solves the problem very well and using that to provide inputs for basis generation.

The column on the left of Figure 33 is the main integration program, which is closely tied to the `derivs()` routine as they pass current coefficient values and their time derivatives back and forth. The integrator is initialized by a bump at $(r_0(0), \theta_0(0))$ projected onto the basis. Output at the requested time steps is superimposed with the basis to produce fields again. The equation parameters enter only into the two convection equation solvers, analytic and basis space. The analytic solution is used to determine the location of the bump as a function of time, which gives the true fields to use in error calculation. Generally an L^2 norm over the domain is used to evaluate the solutions.

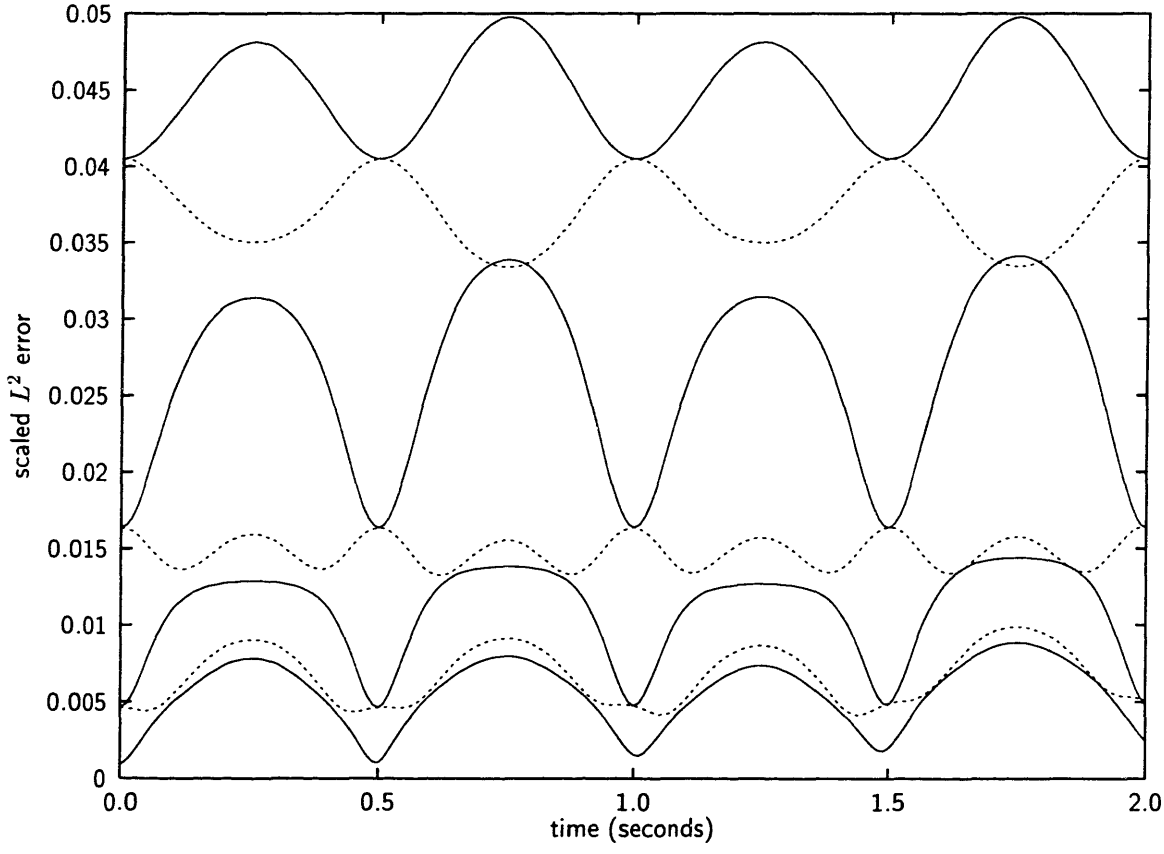


Figure 34. L^2 error as a function of time over an integration of two seconds, for different basis sizes used in the integration: 1–7, which monotonically decrease.

The first obvious error analysis to construct is an L^2 error norm as a function of time and number of basis fields. For an analytically constructed set of input fields of eleven even steps in r_0 from 0.25 to 0.75 (The radial domain is taken to be $[0, 1]$ for convenience.), using the Gaussian bump of unit height and spread 0.1, and integration parameters $\alpha = 2\pi$ (radial oscillation frequency) and $\beta = 0.1$ (radial oscillation amplitude), the error is shown plotted in Figure 34. The scaled L^2 norm which is plotted is defined similarly as in Cartesian coordinates:

$$\|f\|_2^2 \triangleq \frac{1}{\pi}\langle f, f\rangle^2 = \frac{1}{\pi}\int_0^1 \int_0^{2\pi} f(r,\theta)\, r\, dr\, d\theta$$

where the factor $\frac{1}{\pi}$ is used just to ensure that $\|1\|_2 = 1$. This norm, when discretized, is independent of the size of the radial grid and thus is well suited for comparisons among different schemes. The first aspect of the error plot to notice is that the norm steadily decreases as the number of basis fields used in the calculation is increased, but that the error is oscillatory in time, with a period that is some small multiple of that prescribed by α in the governing equation. Also, the troughs in the error plot for a single basis just exactly touch the peaks of the plot when two basis fields are used, and similarly for three and four. These characteristics can be understood by looking at the basis generated by the example fields.

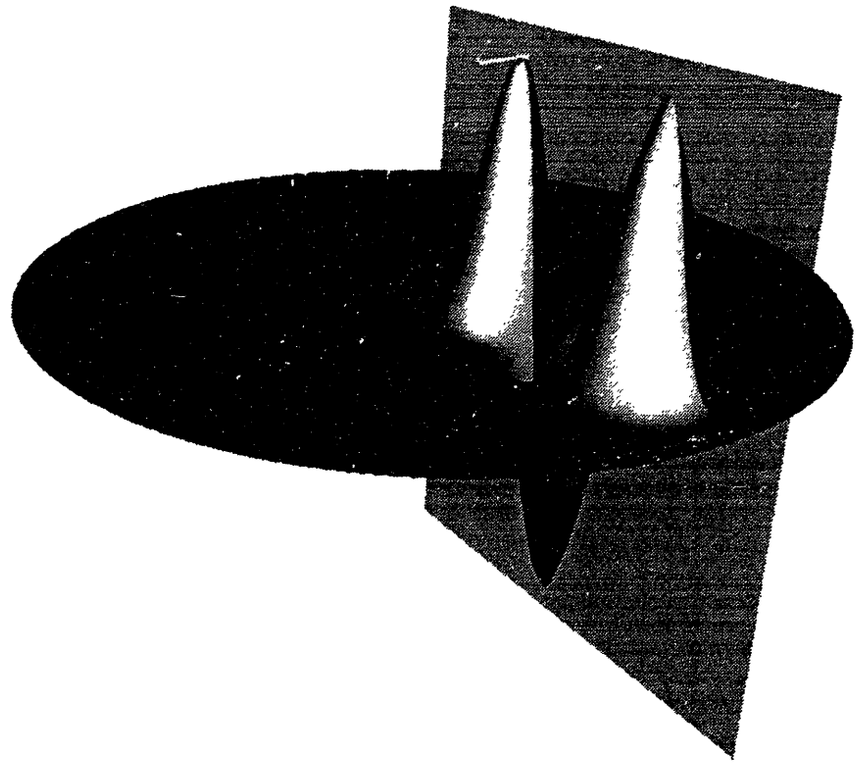


Figure 35. Example slice through a cylindrical basis field showing the source of data plotted in Figure 36.

Since the bases were generated using an analytic Gaussian bump creator, they show bilateral symmetry in the cylindrical coordinate about $\theta = 0$. Thus it suffices to plot only a single radial slice instead of the whole data field, which greatly reduces the amount of data needed to examine and compare the fields. The slice is illustrated using the third basis member in Figure 35, and the radial slices of all the basis elements are plotted in Figure 36. In the error plot of Figure 34, the error using only one field in the integration (the average of all the inputs essentially), is reduced by adding a second field during most of the integration, but at times 0.0, 0.5, 1.0, 1.5, and 2.0, the presence of the second field has no effect whatsoever on the error. These times are precisely when the analytic solution predicts the center of the bump to be at $r = 0.5$, in the middle of the field. The second basis very much resembles an odd function about $r = 0.5$, while the initial condition bump centered at $r = 0.5$ is an even function, so the inner product of the two is zero leading to no contribution by the second basis field. The same argument applies to the error curves in Figure 34 corresponding to 3 and 4 fields, and 5 and 6 fields in the basis. Beyond that the comparison becomes muddled due, perhaps, to the radial asymmetry in the inner product (the integration with respect

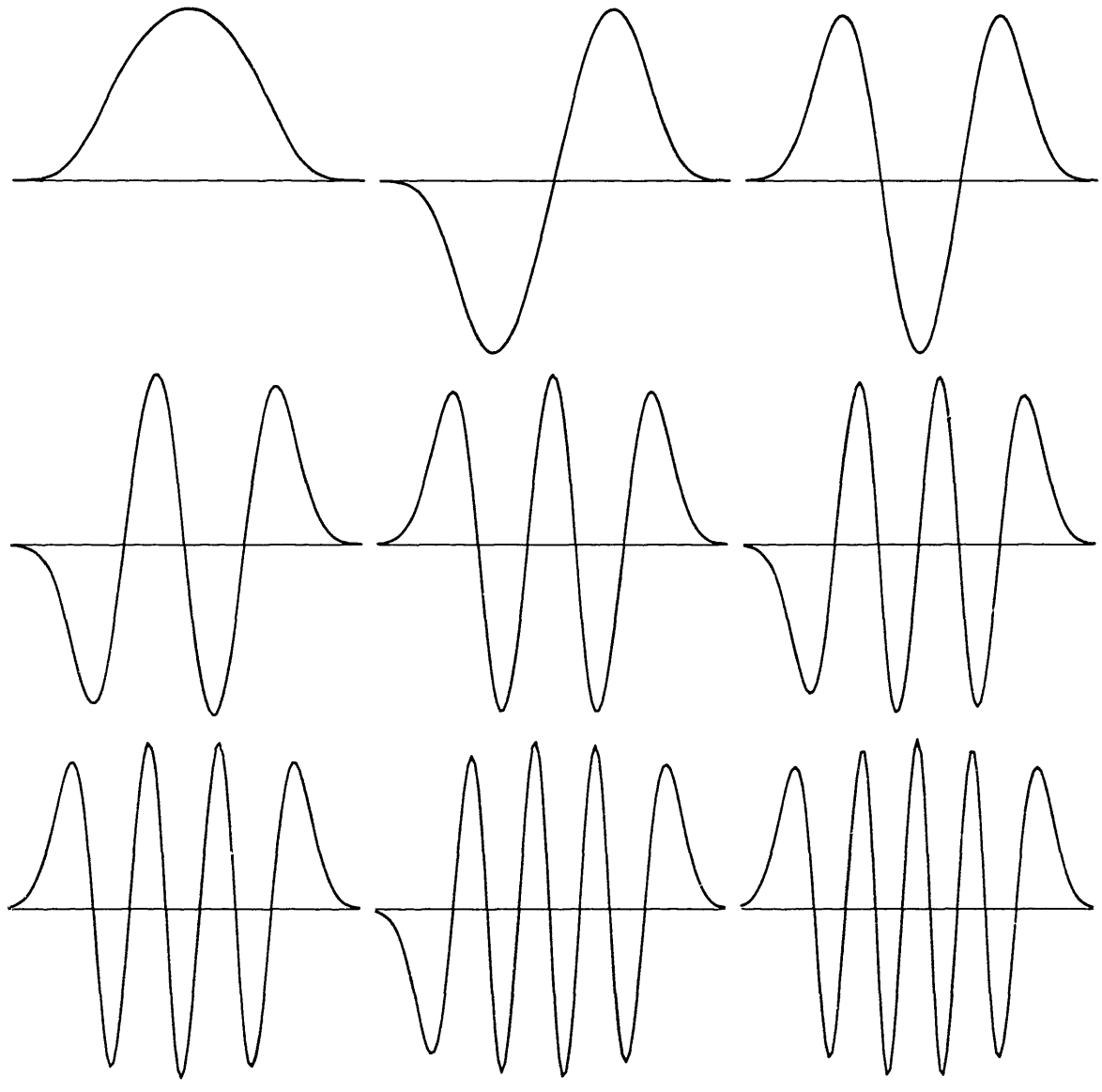


Figure 36. First nine cylindrical basis functions (from left to right and top to bottom) for transformed rotating plume problem with radial oscillations.

to the measure $r\ dr\ d\theta$). Another observation is that the error using a basis of four fields has an apparent frequency of oscillations which is twice those of the previous three error plots, just as the fourth basis field has twice as many maxima as does the second. The doubled frequency arises from the reduced contributions made by the fourth field at $r = 0.5$ as well as at $r = 0.25$ and $r = 0.75$.

As has been emphasized before in using basis space integration, the parameters used during integration should be close to those used in deriving the basis fields for the most accurate results. This is illustrated in Figure 37 which shows the error increasing with larger radial oscillation amplitude. In the case of $\beta > 0.25$, the error becomes unbounded since the input fields only account for cases satisfying $\beta \leq 0.25$; part of the information in the solution is able to escape the space spanned by the basis function fields, permanently degrading the solution. Repeating the integration for this β, but using a basis augmented with the orthogonal components of Gaussian bumps centered at $r = 0.2$

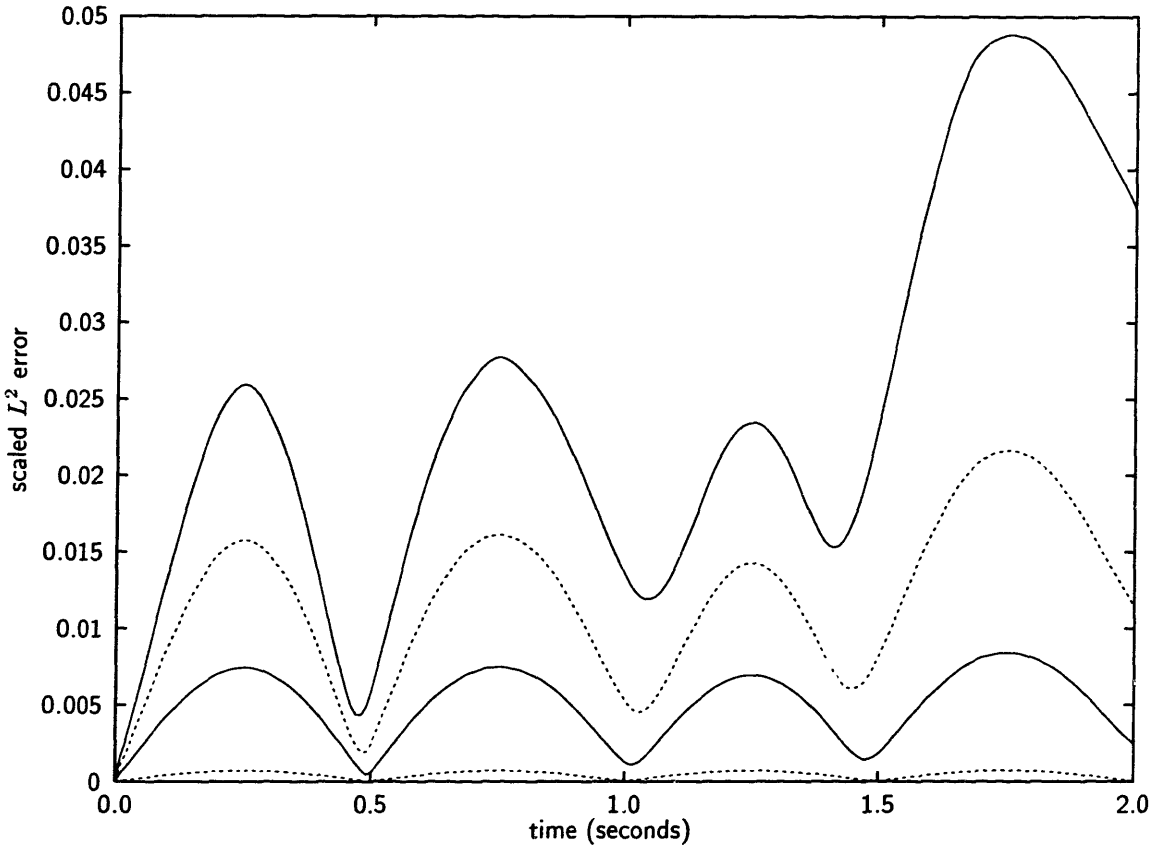


Figure 37. L^2 error as a function of time for different values of the parameter β: 0.3, 0.2, 0.1, and 0.01, all using the full set of eleven basis fields.

and $r = 0.8$ brings the error back under control. Changing the radial oscillation frequency α has no effect on the quality of the solution, as long as the integration time has been chosen small enough to capture the time scale of variation. The effect of the grid size used to generate the basis is also minimal as long as the grid spacing is fine enough to resolve the bump at any location, since the grid is completely irrelevant once integration begins.

A slight generalization of the above situation involves the same equation but without constraints on the ratio of velocities in the radial and circular directions. There are two limiting conditions which prescribe the transformation more likely to be effective in reducing the dimensionality of the resulting system through basis space integration. The first, as considered above, is that when the ratio α/M is small, corresponding to fast advection around the central point, with slow (or small) variations in the radial direction. Transforming into a coordinate system which rotates at the same rate effectively causes one whole dimension of the motion to drop out of the equations. The other extreme is the case when the time scale of oscillations in the radial direction is much shorter than that of the global rotation. For integration over short time scales it is more advantageous to make the transformation into a constant-r space. When the problem is considered on a 2-torus ($\mathbb{S}^1 \times \mathbb{S}^1$) instead of on a disc ($\mathbb{S}^1 \times [0,1]$), the bound on r is lifted, making the two directions equivalent. The ratio of average r-velocity to θ-velocity directs which transformation should be made. Both extremes are illustrated in Figure 38. Other problem considerations may dictate the appropriate transformation for more complicated assemblies, naturally. An implementation of empirical basis space integration for this particular example *without* using the θ transform requires far more basis elements to capture the motion faithfully, and is more prone to diffusive spreading than the transformed case.

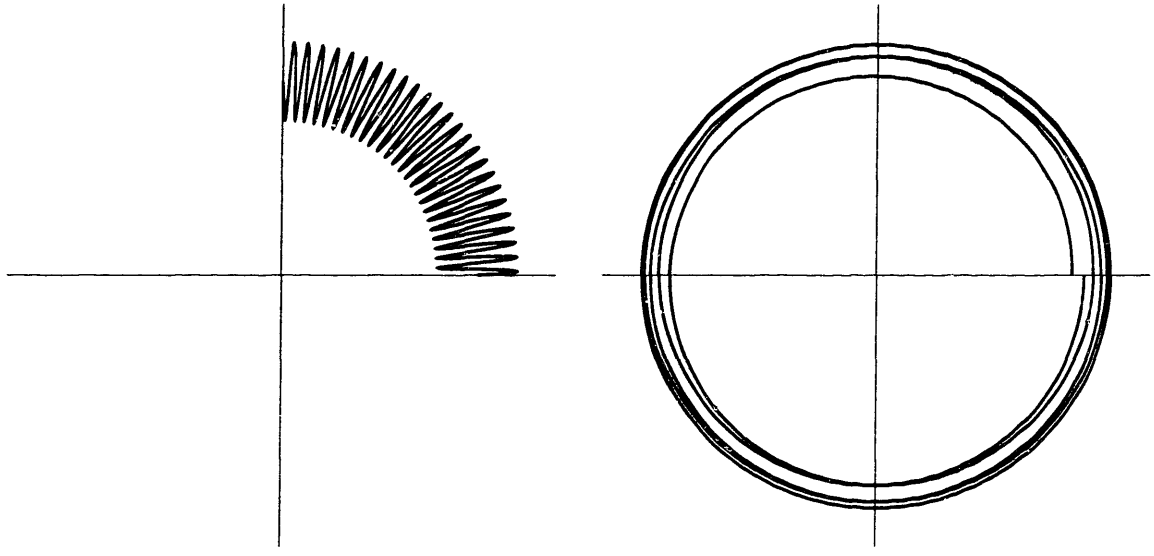


Figure 38. Limits of oscillation frequency ratios: large $\frac{\alpha}{M}$ corresponding to fast radial motion, and small $\frac{\alpha}{M}$ for dominant cylindrical motion.

9.5 Automatic conversion

This chapter has described methods of transforming a set of differential equations to employ integration in the space of an empirically derived basis. For applications involving more than just a few dependent variables, such as a system of chemical reactions, doing the transformation by hand is tedious and error-prone. A system has been written to automate this process and the details of its operation are described fully in the appendices.

Input to the automatic system consists of a chemical mechanism, in an unconstrained textual format including species, constants, reactions, and a section for specifying extra differential fields that may be needed to implement convection or other cross-grid motion. Another input file specifies details of the calculation: where to find the input fields to generate the basis (including derivative fields) for each species, the specific file formats of the inputs, and how many fields to use in the integration, for each species. These two input files are parsed to create a `FORTRAN` subroutine implementing the system plus two shell scripts for re-generating the basis fields and running the integration. The subroutine may be linked together with a template integrator package, or used in more complex codes.

10 Dynamic error tracking

Using information about static error in each of the basis fields used in an integration, as derived in Section 8.3, it is possible to track dynamically the extent to which the solution is not correct. This chapter considers the various sources of error during an integration and discusses means of quantifying the error from each source as well as possible remedies.

10.1 Initial condition error

First, before any time derivatives are generated, the initial condition field for each dependent variable must be projected onto its basis. The error for each component is simply

$$\text{err}_A = \left\| \mathbf{A} - \sum_{i=1}^{N} a_i \mathbf{A}_i \right\|, \qquad a_i = \langle \mathbf{A}, \mathbf{A}_i \rangle$$

and can be eliminated by adding the orthogonal component into the basis, as discussed in Section 8.3 and the appendices, that is, by computing the field inside double vertical bars of the above expression, normalizing it to unity, and appending it to the original basis of size N to create a larger one which will capture the initial condition with zero error. One detail to notice is that the initial coefficient values chosen above assume that the basis is perfect, that is, that $\langle \mathbf{A}_i, \mathbf{A}_j \rangle = \delta_{ij}$ for all combinations of i and j. If this is not the case, as is often seen, the coefficients may be calculated iteratively, performing the two steps

$$a_i = \langle \mathbf{A}, \mathbf{A}_i \rangle$$
$$\mathbf{A} = \mathbf{A} - a_i \mathbf{A}_i$$

for each i, one at a time. This guarantees that each component in $\mathbf{A}$ which falls into the span of two or more basis elements gets counted only once since the component is immediately subtracted from the field when it is encountered. The problem with using this sort of projection scheme is that numerical roundoff errors accumulate quickly with the repeated subtractions from $\mathbf{A}$.

This initial error is particularly harmful to the total accuracy of a solution scheme as it accumulates according to the dynamics of the physical system, as shown in Section 8.3. For example, consider the set of differential equations

$$\frac{d\mathbf{A}}{dt} = k_A \mathbf{A}, \qquad \frac{d\mathbf{B}}{dt} = k_B \mathbf{A}$$

for which the initial condition of field **A** is captured entirely by two basis elements. So that projection errors are not a factor, **B** will need at least two basis elements, as will be shown in the next section. The solution for **A**'s coefficients is easily seen to be

$$a_1(t) = a_1(0)e^{k_A t}$$
$$a_2(t) = (a_2(0) - \varepsilon)e^{k_A t}$$

except for the small positive parameter ε representing an error in the projection of the initial condition of **A** onto its second basis element. This error propagates through into **B**, first by writing the differential equation for the coefficients,

$$\dot{b}_j = k_B a_1(0)\langle \mathbf{A}_1, \mathbf{B}_j\rangle e^{k_A t} + k_B(a_2(0) - \varepsilon)\langle \mathbf{A}_2, \mathbf{B}_j\rangle e^{k_A t}$$

and solving to obtain

$$b_j(t) = b_j(0) + \Big(a_1(0)\langle \mathbf{A}_1, \mathbf{B}_j\rangle + (a_2(0) - \varepsilon)\langle \mathbf{A}_2, \mathbf{B}_j\rangle\Big)\frac{k_B}{k_A}\Big(e^{k_A t} - 1\Big).$$

Now the exact solution is just the above with ε set to zero, and the difference between this and the exact is

$$b_{\mathrm{err},j}(t) = \varepsilon\langle \mathbf{A}_2, \mathbf{B}_j\rangle\frac{k_B}{k_A}\Big(e^{k_A t} - 1\Big).$$

And the total error in the whole field is

$$\begin{aligned}
\|\mathbf{B}_{\mathrm{err}}(t)\| &= \left\|\Big(\langle \mathbf{A}_2, \mathbf{B}_1\rangle\mathbf{B}_1 + \langle \mathbf{A}_2, \mathbf{B}_2\rangle\mathbf{B}_2\Big)\varepsilon\frac{k_B}{k_A}\Big(e^{k_A t} - 1\Big)\right\| \\
&= \left\|\Big(\langle \mathbf{A}_2, \mathbf{B}_1\rangle\mathbf{B}_1 + \langle \mathbf{A}_2, \mathbf{B}_2\rangle\mathbf{B}_2\Big)\right\|\varepsilon\frac{k_B}{k_A}\Big(e^{k_A t} - 1\Big) \\
&\le \Big(\|\langle \mathbf{A}_2, \mathbf{B}_1\rangle\mathbf{B}_1\| + \|\langle \mathbf{A}_2, \mathbf{B}_2\rangle\mathbf{B}_2\|\Big)\varepsilon\frac{k_B}{k_A}\Big(e^{k_A t} - 1\Big) \\
&\le \Big(|\langle \mathbf{A}_2, \mathbf{B}_1\rangle| + |\langle \mathbf{A}_2, \mathbf{B}_2\rangle|\Big)\varepsilon\frac{k_B}{k_A}\Big(e^{k_A t} - 1\Big)
\end{aligned}$$

using orthogonality of **B**'s basis, and the claim that there is no projection error from **A**'s fields. Thus a miniscule error in the initial projection of **A** leads to exponentially increasing error in **B**, for the case of this unstable set of equations. Realistic systems are often characterized by some damping which will tend to wash away any initial condition errors.

When this error can be written analytically, and using ε as reported from the diagonal elements of `testbasis`, a decision about when the integration must be halted can be made, based on the given error tolerance. When no analytic derivation is available, this error can be tracked by the addition of another coefficient in the integration scheme. The evolution equation is the sum of that for all the other coefficients of **B**, with ε_j substituted for a_j wherever it appears. In this case the equation is

$$\dot{b}_{\mathrm{err}} = \sum_{ij} k_B\,\varepsilon_j\langle A_j, B_i\rangle$$

with an initial condition of zero. In the example, ε_1 was taken to be zero. Of course, since this error can be tracked, it can also be eliminated by improving the supposed orthonormality of the basis of **A** so as to reduce all the ε_j such that the integrated error in **B** is minimal over all time. And the exponentially increasing dependence on time is not common in realistic systems of equations, serving further to reduce the importance of this sort of error.

10.2 Projection error

The largest source of error in integration schemes described in this thesis comes from projection errors—the projection of elements of one basis upon those of another. Evolution equations for the coefficients of a field expansion feature interaction terms with other fields of the form

$$\langle \mathbf{B}_j, \mathbf{A}_i \rangle$$
$$\langle \mathbf{B}_j * \mathbf{C}_k, \mathbf{A}_i \rangle$$

and so forth as discussed in Chapter 9. These terms represent, pointwise, contribution to $A(\mathbf{x})$ from the presence of $B(\mathbf{x})$ and from the product $B(\mathbf{x})C(\mathbf{x})$, respectively, but transformed into an empirical basis become sums of coefficients and inner products of fields. It is precisely these inner products which cause all the problems.

Essentially what is required for every term of the form given above is that each of the fields $\mathbf{B}_j$, be fully realizable in terms of the basis of $\mathbf{A}$, or that the product of two basis elements $\mathbf{B}_j * \mathbf{C}_k$ be. Each basis element or product must lie fully in the span of $\mathbf{A}$ if all its energy is to be captured in such an inner product. Mathematically, the error in such a projection is easily quantifiable:

$$\begin{aligned} \left\|\mathrm{err}(\mathbf{B}_j, \mathbf{A})\right\|^2 &= \left\|\mathbf{B}_j - \pi(\mathbf{B}_j, \mathbf{A})\right\|^2 \\ &= \left\|\mathbf{B}_j - \sum_i \langle \mathbf{B}_j, \mathbf{A}_i \rangle \mathbf{A}_i\right\|^2 \\ &= \|\mathbf{B}_j\|^2 - \sum_i \langle \mathbf{B}_j, \mathbf{A}_i \rangle^2 \\ &= 1 - \sum_i \langle \mathbf{B}_j, \mathbf{A}_i \rangle^2 \end{aligned}$$

using the Pythagorean theorem and orthonormality of $\{\mathbf{A}_i\}$ and $\{\mathbf{B}_j\}$. This value is easily calculated for each set of projection fields required, and clearly the criterion for no projection error is that the last sum be equal to one, which is equivalent to saying that the individual field $\mathbf{B}_j$ lies completely in the span of the $\{\mathbf{A}_i\}$.

One obvious method of eliminating this source of error is to use the same basis for every dependent variable (chemical species) in the system. Then it is guaranteed that every single-field projection error is by definition zero. However terms of the form $\mathbf{B}_j * \mathbf{C}_k$ still introduce errors

$$\left\|\mathrm{err}(\mathbf{B}_j * \mathbf{C}_k, \mathbf{A})\right\|^2 = 1 - \sum_i \langle \mathbf{B}_j * \mathbf{C}_k, \mathbf{A}_i \rangle^2,$$

suggesting the addition of the orthogonal components of the product fields $\mathbf{B}_j * \mathbf{C}_k$ into the common basis. This method of reducing projection error is certainly not to be favored, as it introduces many more seemingly unnecessary basis elements into the expansions, and does not mesh with the dogma of choosing the basis using only information about the problem and its constraints. These projection correction basis members seem artificial, but in practice it is often necessary to include them if the interacting species have significantly different spatial structures.

The amount of error introduced by projection losses can be determined statically, using the script `proj_goodness` which essentially calculates the formulas given above for each projection needed by an integration scheme, directly from the user-generated input files. However, large projection losses will not occur if the involved coefficients turn out to be small. Continuing with the example, the evolution equation for the coefficients of component $\mathbf{A}$ may include a term

$$\frac{da_i}{dt} = \cdots + \sum_{jk} b_j c_k \langle \mathbf{B}_j * \mathbf{C}_k, \mathbf{A}_i \rangle + \cdots$$

and for a particular choice of j and k will introduce an error of magnitude

$$\|b_j c_k \operatorname{err}(\mathbf{B}_j * \mathbf{C}_k, \mathbf{A})\|$$

into the values of field **A**. It could result that values of j and k where the error term is large will only occur for small (in relation to the other coefficients) b_j or c_k or both for all time. Thus the results of static error determination alone do not necessarily warrant the introduction of augmenting fields. The use of dynamic error tracking possibly requires a few more design iterations, but may save overall execution time in the long run.

10.3 Integrator error

The final source of error is introduced during the actual process of integration—stepping the coefficient values through time. An off-the-shelf ordinary differential equation solver such as LSODE [63] is sufficient for performing the simple forward time integration, as this type of code has been in existence for some time and the algorithms are in general stable and efficient. Furthermore, provisions for controlling the absolute and relative error tolerances are given, and the codes guarantee robust operation and satisfaction of these bounds.* Hence for the intents of this thesis, possible errors which may be introduced by the low-level integration scheme are considered to be completely controlled and will not be considered.

Choosing the values for the error tolerances which are specified to the integrator requires some care, however. Most integrators will supply a method of setting relative and absolute error tolerances (R and A) such that

$$\|\varepsilon\| \leq R\|\phi\| + A$$

where ε is the difference between the true solution and the current value of ϕ, as estimated by the integration package. The expression above is written for the case of error of a whole field, which will be expanded in terms of a basis $\{\phi_i\}$. The question is, what should be the values R_i and A_i of error control on the coefficients of the expansion such that a certain integrator error given by R and A is satisfied:

$$|\varepsilon_i| \leq R_i|c_i| + A_i \qquad \text{in} \quad \phi = \sum_i^N c_i\phi_i.$$

The equation for the full field error reduces through definitions:

$$\begin{aligned}\|\varepsilon\| &= \|\phi - \phi^*\| \\ &= \left\| \sum_i^N c_i\phi_i - \sum_i^N c_i^*\phi_i \right\| \\ &= \left\| \sum_i^N (c_i - c_i^*)\phi_i \right\|\end{aligned}$$

where $\{c_i^*\}$ are the coefficients of expansion if the integration is performed exactly. This expansion is assumed to exist, since the determination of projection errors as discussed above would quantify the deviations in this expansion. Specializing now to the case where the norm is a possibly weighted L^2 norm, which encompasses many of the practical applications of this error determination, the term

$$\left\| \sum_i^N a_i\phi_i(\mathbf{x}) \right\|$$

* This claim by the writers of the differential equation solving algorithms is disputed, however, by Russell Allgor and Paul Barton.

becomes

$$\left(\int_X w(\mathbf{x})\left(\sum_i^N a_i\phi_i(\mathbf{x})\right)^2 d\mathbf{x}\right)^{\frac{1}{2}}$$

which reduces using the generalized binomial series to

$$\left(\int_X w(\mathbf{x})\left(a_1^2\phi_1^2(\mathbf{x}) + \cdots + a_N^2\phi_N^2(\mathbf{x}) + 2a_1\phi_1(\mathbf{x})a_2\phi_2(\mathbf{x}) + \text{other cross terms}\right)\right)^{\frac{1}{2}}.$$

The "other cross terms" in the last expression will be seen to vanish and need not be considered in detail. Pushing the integration onto each term in the sum gives

$$\left(\int_X w(\mathbf{x})a_1^2\phi_i^2(\mathbf{x})\,d\mathbf{x} + \cdots + \int_X w(\mathbf{x})a_N^2\phi_N^2(\mathbf{x})\,d\mathbf{x} + 2\int_X w(\mathbf{x})a_1a_2\phi_1(\mathbf{x})\phi_2(\mathbf{x})\,d\mathbf{x} + \text{cross terms}\right)^{\frac{1}{2}}$$

which reduces, using the definitions of norm and inner product, to

$$\left(\sum_i^N \|a_i\phi_i\|^2 + 2\langle a_1\phi_1, a_2\phi_2\rangle + \text{cross terms}\right)^{\frac{1}{2}}$$

All the "cross terms" involve inner products of the form $\langle\phi_i, \phi_j\rangle$ where $i \neq j$, and vanish due to orthogonality of the basis. This leaves

$$\begin{aligned}\left\|\sum_i^N a_i\phi_i(\mathbf{x})\right\| &= \left(\sum_i^N \|a_i\phi_i\|^2\right)^{\frac{1}{2}} \\ &= \left(\sum_i^N a_i^2\|\phi_i\|^2\right)^{\frac{1}{2}} \\ &= \left(\sum_i^N a_i^2\right)^{\frac{1}{2}}\end{aligned}$$

using normality of the basis elements. The original expansion of the full field error becomes, using this result,

$$\|\varepsilon\| = \left(\sum_i^N (c_i - c_i^*)^2\right)^{\frac{1}{2}}.$$

Defining the per-coefficient error $\varepsilon_i = |c_i - c_i^*|$ shows this gives a restriction on the l_2 norm of the coefficients:

$$\begin{aligned}\sum_i^N \varepsilon_i^2 &\leq R^2\|\phi\|^2 + A^2 + 2RA\|\phi\| \\ &= R^2\sum_i^N c_i^2 + A^2 + 2RA\left(\sum_i^N c_i^2\right)^{\frac{1}{2}}\end{aligned}$$

but what is desired is a restriction on each of the ε_i individually,

$$|\varepsilon_i| \leq R_i|c_i| + A_i.$$

Inserting this into the l_2 norm on the set ε_i gives

$$\begin{aligned}\sum_i^N \varepsilon_i^2 &\leq \sum_i^N (R_i|c_i| + A_i)^2 \\ &= \sum_i^N \left(R_i^2 c_i^2 + A_i^2 + 2R_i A_i |c_i|\right) \\ &= \sum_i^N R_i^2 c_i^2 + \sum_i^N A_i^2 + 2\sum_i^N R_i A_i |c_i|\end{aligned}$$

By comparison with the expression involving the desired error criteria parameters R and A suggests choosing the values

$$\begin{aligned}R_i &= R \\ A_i &= \frac{1}{\sqrt{N}} A\end{aligned}$$

for all i, which gives

$$\sum_i^N \varepsilon_i^2 = R^2 \sum_i^N c_i^2 + A^2 + 2RA \sum_i^N |c_i|.$$

Comparing this expression with the restriction derived directly from the full field error criterion shows that this bound will be at least as good as the desired one if

$$R^2 \sum_i^N c_i^2 + A^2 + 2RA \sum_i^N |c_i| \quad \leq \quad R^2 \sum_i^N c_i^2 + A^2 + 2RA \left(\sum_i^N c_i^2\right)^{\frac{1}{2}}$$

or, subtracting and dividing out common elements,

$$\sum_i^N |c_i| \leq \left(\sum_i^N c_i^2\right)^{\frac{1}{2}}.$$

The term on the left side of the inequality is the l_1 norm of the coefficients, while the one on the right is the l_2 norm. Using the Holder inequality* shows this is indeed satisfied, and the assignments to R_i and A_i will satisfy the original error requirements.

10.4 Conclusion

Three distinct sources of error have been analyzed in this chapter: initial condition error, projection error, and integrator error. The first type has the ability to cause discrepancies of an unbounded nature, but is easy to detect and correct. The third source of error is that which is common to all integration schemes due to discretization in time, and the only result specific to basis space

* Let p, q, and r be positive numbers satisfying $p, q, r \geq 1$ and $p^{-1} + q^{-1} = r^{-1}$. Suppose $f \in L^p(X, d\mu)$, $g \in L^q(X, d\mu)$. Then $fg \in L^r(X, d\mu)$ and

$$\|fg\|_r \leq \|f\|_p \|g\|_q.$$

Here the l_p spaces used are just $L^p(\mathbf{R}, d\mu)$ where μ is the measure with mass one at each positive integer and zero elsewhere, $p = q = 2$, $r = 1$, and $g(x) = 1$ everywhere, using $f = c$.

integration is the conversion of error tolerances on the fields into tolerances on the coefficients of field expansions. Projection errors, caused by the inability of a basis to capture contributions from products of other basis elements, is common to all implementations of a Galërkin method, but in cases where analytic bases are used it sometimes is easy to write analytic expressions for the projection error as a function of time. Here, however, the best that can be offered is a method for tracking that error during an actual simulation. While not an *a priori* way to discover the magnitudes of projection errors, dynamic tracking can be used iteratively to improve a simulation. Examples of this are given in the next chapter.

11 Numerical results

A theoretical discussion of an integration algorithm is crucial for determining the important properties of convergence, expected memory requirements and so on, but the true test comes from real applications. This chapter discusses some specific problems in detail, reporting execution times and errors.

11.1 Three-reaction ozone

This example is not interesting for the numerical results obtained, as the system approaches steady state quickly and uniformly, rendering plots of convergence as a function of simulation time as straight lines. The ozone chemistry is used here as a framework for explaining some of the choices that go into choosing a good basis for simulation and different ways to evaluate the worthiness of such a basis numerically. Properties of various eigenvalue solver schemes are detailed as well.

The basic photochemical cycle of the oxides of nitrogen and ozone is [124]:

$$\begin{aligned} NO_2 + h\nu &\xrightarrow{1} NO + O \\ O + O_2 + M &\xrightarrow{2} O_3 + M \\ O_3 + NO &\xrightarrow{3} NO_2 + O_2 \end{aligned}$$

where M represents any third body which absorbs excess vibrational energy to stabilize the forming ozone. In the atmosphere, M can be treated as a constant with concentration 10^6 ppm, and oxygen will have the constant value 2.1×10^5 ppm. As a convenience in writing the mechanism file, the term $h\nu$ is retained in the equation but set as a constant equal to one so that it is effectively ignored in the construction of rate equations. Actual variations in light intensity would be recorded in the rate constant for that reaction. This small set of equations is quite stiff due to the wide range in the reaction rates:

$$\begin{aligned} k_1 &= 0.555 \text{ min}^{-1} \\ k_2 &= 2.183 \times 10^{-5} \text{ ppm}^{-2} \text{ min}^{-1} \\ k_3 &= 26.59 \text{ ppm}^{-1} \text{ min}^{-1} \end{aligned}$$

leading to different time scales for the reactions.

Effects of the integrator used to do the time stepping have a non-negligible influence on the structure of errors in the results. For this example the common and robust integrator LSODE [63] was used. It offers two basic methods, implicit Adams and backward differentiation, each with six variations to the corrector iteration method, as shown in Table 14. From analyzing the Jacobian it is clear that this problem will not realize any advantage by using a banded structure, hence the last two possibilities above will be ignored. To illustrate the effects of the other LSODE parameter settings, the results for a basis of 30 fields, the same one for all four species, are given in Table 15, where `mf` = 10+`miter` for implicit Adams, and `mf` = 20+`miter` for backward differentiation (Gear's convention). In Table 15, "f-" and "j-eval" record the number of function and Jacobian evaluations, respectively; the internal number of iterations is listed under "steps;" values for "time" are given as a sum of user and system CPU seconds. As this simulation starts from an initial condition which is quite far from the steady-state values, it is infeasible to use no Jacobian at all during the region of stiffness, and LSODE succumbs after trying for some time. The quickest runs by far are those using a user-supplied Jacobian with which, due to its analyticity, there is no need to generate a table using repeated function evaluations at small steps away from the state of interest. Also it is clear that the problem becomes diagonally dominant as LSODE is able to integrate using a diagonal Jacobian in about half the time as a full internally generated Jacobian, and the backward differentiation method is consistently faster than implicit Adams. It should be mentioned that for all the cases the comparison of the output against a grid-based run under the same conditions was nearly identical (within one part in a million), with the odd exception that `mf` = 13 was about one percent *better* than all the others, perhaps only since the relative error tolerance was spread over a smaller number of coefficients, forcing them to better convergence.

`miter`	iteration method	Jacobian structure	Jacobian source
0	functional	(no Jacobian is involved)	
1	chord	full	user-supplied
2	chord	full	internally generated
3	chord	diagonal	internally generated
4	chord	banded	user-supplied
5	chord	banded	internally generated

Table 14. LSODE integration methods.

`mf`	time	f-eval	j-eval	steps
10	393.34+0.24	(too much work)		
11	28.53+0.08	310	26	267
12	247.70+0.16	3437	26	275
13	157.72+0.11	2207	472	876
20	395.71+3.77	(too much work)		
21	11.93+0.08	108	9	85
22	170.91+0.13	2388	19	85
23	94.75+0.10	1321	291	551

Table 15. Statistics for the various LSODE integration methods.

Now that the timing and convergence errors of using different integrator setups has been considered, the actual path of the integration is examined more closely. For this, debugging lines are inserted into the basis space integration code to output the input coefficients at each call to the `derivs` routine (function evaluation), as well as the output time derivatives of the coefficients. An auxiliary program reads these coefficients, calculating error norms in various ways. The natural

choice of this norm, given that the Karhunen-Loève procedure is being used to generate optimal bases, is the mean-square norm, which is calculated in the following steps:

1. Read the input coefficients and expand into fields.
2. Perform the zero-dimensional chemistry in grid space.
3. Read the output coefficients and expand into fields.
4. Compute the L^2 difference between the results of steps (2) and (3).

This produces a value for each time the `derivs()` routine is called, then, and makes a plot similar to that given in Figure 39.

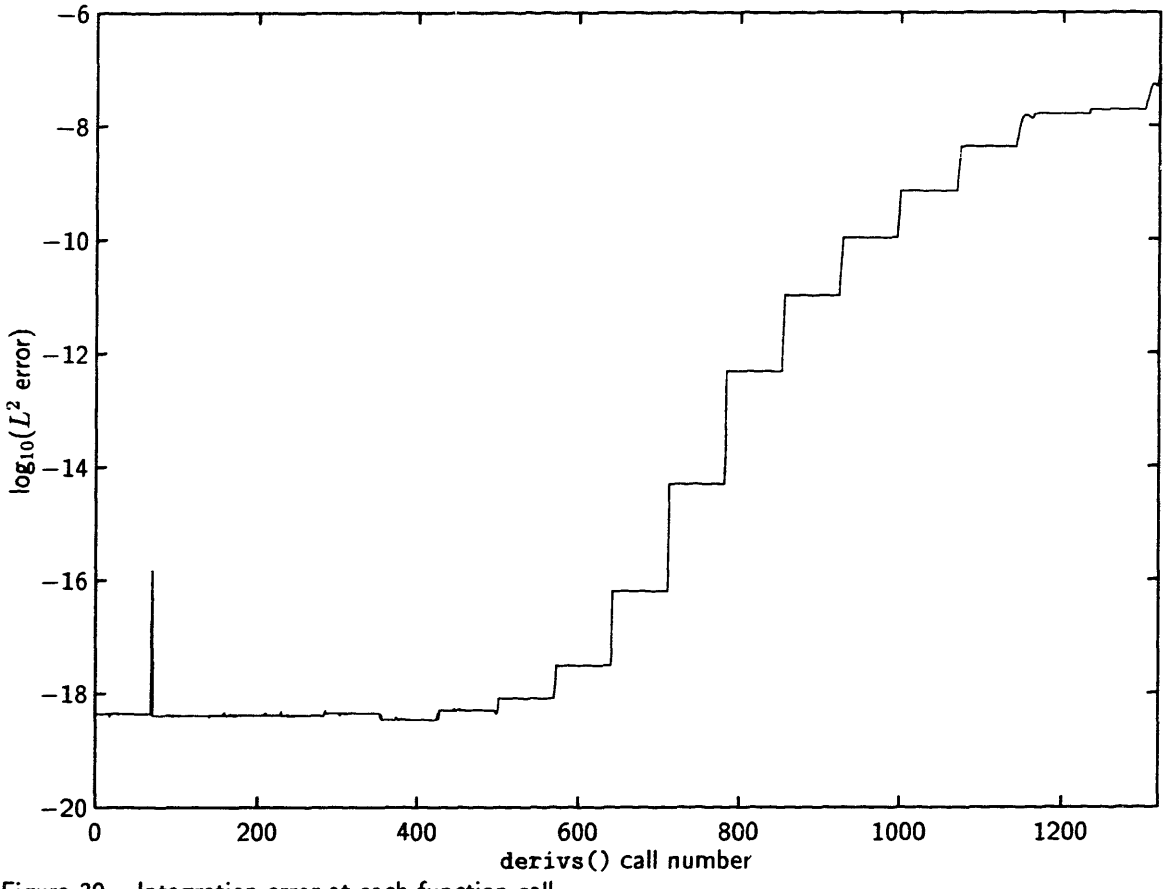


Figure 39. Integration error at each function call.

The horizontal axis in Figure 39 is integral, corresponding to the number of the call being made to `derivs()`, and says nothing about the simulation time (although they both grow starting from zero). Initially the error is on the order of machine precision and small fluctuations arise from roundoff problems as the integrator develops its numerical Jacobian matrix. Since the coefficients were exact (within basis errors) at the initial condition, they remain exact during the initial Jacobian derivation. However, as forward steps are made in time the error rises, although the final value in the plot was small enough given the accuracy desired for the integration. The stepwise nature in the plot happens when the integrator makes some function calls to gather derivative information, which requires many function calls at nearby coefficient values. Large jumps between these plateaus represent the integrator making a decision to apply the iteration matrix.

Replacing step four above with an L^∞ calculation produces a similar picture, but this time showing the maximum absolute value deviation, which may be useful if that criterion is important. One further metric on these coefficients is to find the projection error. This procedure ignores step

(3) above and asks instead, "Just how well can the results of the grid-based chemistry be projected onto the given bases?" and may guide understanding of what part of the space is missing from the provided bases. This information can be used to guide the process of adding more elements to the basis, as will be discussed in detail in the next section.

One other conclusion which may be reached is that instead of increasing the size of a given basis until error bounds are satisfied, it may be advantageous to have the simulation switch to a partially or entirely new basis. In models governed by a small number of very distinct operating stages this seems like the best solution. The problem is started using the first basis, as always, but either when the dynamic error has grown too large or at a preset transition point, the current values of the coefficients are transformed using projections of each basis element in the first set onto the second, a fairly cheap operation for all but the simplest models. Any internal state of the integrator regarding step sizes and numerically-derived Jacobian information should probably be destroyed at the transition however. Then the simulation resumes and continues to the next transition point or to termination.

11.2 AIRSHED chemistry

The atmospheric chemistry solver **AIRSHED**, as discussed in many different contexts throughout this thesis, will serve as the ultimate test of the proposed method. As a first approach, only the chemistry integration is considered, which requires the bulk of the work of a traditional grid-based solver (around 75%). For completeness, all details of the mechanism will be listed. First, the species participating in the mechanism are shown in Table 16. It includes 33 normal reactive species, and 9 other species which are usually taken to be at steady state due to the short time scales governing the reactions in which they participate. In this integration the "steady state" species are considered to be fully reactive just like all the other species. The final section in the table lists the six "species" which are actually constant, so they always have the same value and no differential equations are generated for them. It is convenient to include such artificial species in writing reactions instead of including the constant in the rate equation as would otherwise be necessary.

There are 106 reactions involving the 48 species given in Table 16, which are listed in Table 17. The mechanism is an adaptation of that constructed by Carter [31], and uses many lumped reactions to approximate the true system of chemistry. Stoichiometric coefficients for lumped reactions are calculated using carbon splitting factors, which approximate the contribution of the reactions to the various categories of carbon-containing compounds. The three fractions considered in the formulas below are:

$xc = 0.4286$	Fraction of C_4 and C_5 alkanes of all 3-carbon or longer alkanes
$yc = 0.6000$	Fraction of terminal alkenes of all 3-carbon or longer alkenes
$zc = 0.6000$	Fraction of dialkyl benzenes in all di- and tri-alkyl benzenes

To convert from carbon fractions to mole fractions, the following equations are used:

$$x = \frac{(xc/4.5)}{xc/4.5 + (1 - xc)/7.0}$$

$$y = \frac{(yc/3.0)}{yc/3.0 + (1 - yc)/4.0}$$

$$z = \frac{(zc/8.0)}{zc/8.0 + (1 - zc)/9.0}$$

The stoichiometric coefficients are given using the splitting factors just derived, from formulas in [31], for alkanes at 300°C:

NO	Nitric oxide
NO_2	Nitrogen dioxide
O_3	Ozone
HONO	Nitrous acid
HNO_3	Nitric acid
HNO_4	Pernitric acid
N_2O_5	Nitrogen pentoxide
NO_3	Nitrate radical
HO_2	Hydroperoxy radical
CO	Carbon monoxide
HCHO	Formaldehyde
ALD2	Lumped aldehydes
MEK	Methyl ethyl ketone
MGLY	Methyl glyoxal
PAN	Peroxyl acyl nitrate
RO_2	Total RO_2 radicals
MCO_3	CH_3CO_3 radical
ALKN	Alkyl nitrate
ALKA	Alkanes, 4-carbon or longer
ETHE	Ethene
ALKE	Alkenes, 3-carbon or longer
TOLU	Toluene
AROM	Lumped aromatics
DIAL	Other dicarbonyls
CRES	Cresol
NPHE	Nitrophenols
H_2O_2	Hydrogen peroxide
MEOH	Methanol
ISOP	Isoprene
ETOH	Ethanol
MTBE	Methyl *tert*-butyl ether
SO_2	Sulfur dioxide
SO_3	Sulfur trioxide
"Steady state" species	
BZO	Phenoxy radical
BZN2	Benzaldehyde *n*-RO_2
RO_2P	Phenol RO_2
RO_2N	Alkyl nitrate RO_2
RO_2R	Lumped RO_2 number 1
R_2O_2	Lumped RO_2 number 2
OH	Hydroxyl radical
O	O atom, $O(^3P)$
OSD	O atom, $O(^1D)$
"Constant" species	
HV	Light, 1.0 for full sunlight
H_2O	Water vapor, 17500 ppm
CH_4	Methane, 2.2 ppm
M	Third body, 10^6 ppm
O_2	Oxygen, 210000 ppm
TBF	Tertiary butyl formate, unreactive

Table 16. Species involved in the AIRSHED mechanism.

1	NO_2	+ $h\nu$	⟶ NO + O	11	N_2O_5	+ H_2O	⟶ $2\,HNO_3$
2	O		⟶ O_3	12	NO_2	+ NO_3	⟶ NO + NO_2
3	O	+ NO_2	⟶ NO	13	NO_3	+ $h\nu$	⟶ NO
4	O	+ NO_2	⟶ NO_3	14	NO_3	+ $h\nu$	⟶ NO_2 + O
5	NO	+ O_3	⟶ NO_2	15	O_3	+ $h\nu$	⟶ O
6	NO_2	+ O_3	⟶ NO_3	16	O_3	+ $h\nu$	⟶ OSD
7	NO	+ NO_3	⟶ $2\,NO_2$	17	OSD	+ H_2O	⟶ 2 OH
8	NO	+ NO	⟶ $2\,NO_2$	18	OSD		⟶ O
9	NO_2	+ NO_3	⟶ N_2O_5	19	NO	+ OH	⟶ HONO
10	N_2O_5		⟶ NO_2 + NO_3	20	HONO	+ $h\nu$	⟶ NO + OH
21	NO_2	+ H_2O	⟶ HONO + $-1\,NO_2$ + HNO_3				
22	NO_2	+ OH	⟶ HNO_3	32	HO_2	+ HO_2	⟶ H_2O_2
23	HNO_3	+ OH	⟶ NO_3	33	NO_3	+ HO_2	⟶ HNO_3
24	CO	+ OH	⟶ HO_2	34	NO_3	+ HO_2	⟶ HNO_3
25	O_3	+ OH	⟶ HO_2	35	RO_2	+ NO	⟶ NO
26	NO	+ HO_2	⟶ NO_2 + OH	36	RO_2	+ HO_2	⟶ HO_2
27	NO_2	+ HO_2	⟶ HNO_4	37	RO_2	+ RO_2	⟶
28	HNO_4		⟶ NO_2 + HO_2	38	RO_2	+ MCO_3	⟶
29	HNO_4	+ OH	⟶ NO_2	39	HCHO	+ $h\nu$	⟶ $2\,HO_2$ + CO
30	O_3	+ HO_2	⟶ OH	40	HCHO	+ $h\nu$	⟶ CO
31	HO_2	+ HO_2	⟶ H_2O_2	41	HCHO	+ OH	⟶ HO_2 + CO
42	HCHO	+ NO_3	⟶ HNO_3 + HO_2 + CO				
43	HCHO	+ HO_2	⟶ RO_2R + RO_2				
44	ALD2	+ OH	⟶ MCO_3				
45	ALD2	+ $h\nu$	⟶ CO + HCHO + RO_2R + HO_2 + RO_2				
46	ALD2	+ NO_3	⟶ HNO_3 + MCO_3				
47	MCO_3	+ NO	⟶ NO_2 + HCHO + RO_2R + RO_2				
48	MCO_3	+ NO_2	⟶ PAN				
49	MCO_3	+ HO_2	⟶ HCHO				
50	MCO_3	+ MCO_3	⟶ $2\,HO_2$ + 2 HCHO				
51	PAN		⟶ MCO_3 + NO_2				
52	MEK	+ $h\nu$	⟶ MCO_3 + ALD2 + RO_2R + RO_2				
53	MEK	+ OH	⟶ $1.2\,R_2O_2$ + $1.2\,RO_2$ + MCO_3 + 0.5 ALD2 + 0.5 HCHO				
54	MGLY	+ $h\nu$	⟶ MCO_3 + HO_2 + CO				
55	MGLY	+ OH	⟶ MCO_3 + CO				
56	MGLY	+ NO_3	⟶ HNO_3 + MCO_3 + CO				
57	ALKA	+ OH	⟶ b_1 HCHO + b_2 ALD2 + b_3 MEK + b_4 RO_2N + b_5 RO_2R + b_6 R_2O_2 + b_7 RO_2				
58	ALKN	+ OH	⟶ NO_2 + 0.15 MEK + 1.53 ALD2 + 0.16 HCHO + $1.39\,R_2O_2$ + $1.39\,RO_2$				
59	RO_2N	+ NO	⟶ ALKN				
60	RO_2N	+ HO_2	⟶ MEK				
61	RO_2N	+ RO_2	⟶ RO_2 + HO_2 + MEK				
62	RO_2N	+ MCO_3	⟶ HCHO + HO_2 + MEK				
63	R_2O_2	+ NO	⟶ NO_2				
64	R_2O_2	+ HO_2	⟶				
65	R_2O_2	+ RO_2	⟶ RO_2				
66	R_2O_2	+ MCO_3	⟶ HCHO + HO_2				
67	RO_2R	+ NO	⟶ NO_2 + HO_2				
68	RO_2R	+ HO_2	⟶				

69	RO_2R	+ RO_2	$\longrightarrow$	$0.5\,HO_2 + RO_2$
70	RO_2R	+ MCO_3	$\longrightarrow$	HO_2 + HCHO
71	ETHE	+ OH	$\longrightarrow$	$RO_2R + RO_2 + 1.56\,HCHO + 0.22\,ALD2$
72	ETHE	+ O_3	$\longrightarrow$	$HCHO + 0.12\,HO_2 + 0.42\,CO$
73	ETHE	+ O	$\longrightarrow$	$RO_2R + RO_2 + CO + HCHO + HO_2$
74	ETHE	+ NO_3	$\longrightarrow$	$RO_2 + NO_2 + 2\,HCHO + R_2O_2$
75	ALKE	+ OH	$\longrightarrow$	$RO_2R + RO_2 + b_8\,HCHO + b_9\,ALD2$
76	ALKE	+ O3	$\longrightarrow$	$b_{10}\,HCHO + b_{11}\,ALD2 + b_{12}\,RO_2R + b_{12}\,RO_2 + b_{13}\,HO_2 + b_{14}\,OH + b_{15}\,CO$
77	ALKE	+ O	$\longrightarrow$	$b_{16}\,CO + b_{17}\,MEK + b_{18}\,HCHO + b_{19}\,ALD2 + b_{20}\,HO_2 + b_{21}\,RO_2R + b_{21}\,RO_2$
78	ALKE	+ NO_3	$\longrightarrow$	$NO_2 + b_8\,HCHO + b_9\,ALD2 + R_2O_2 + RO_2$
79	TOLU	+ OH	$\longrightarrow$	$0.16\,CRES + 0.16\,HO_2 + 0.84\,RO_2R + 0.4\,DIAL + 0.84\,RO_2 + 0.144\,MGLY + .11\,HCHO + .114\,CO$
80	AROM	+ OH	$\longrightarrow$	$0.17\,CRES + 0.17\,HO_2 + 0.83\,RO_2R + 0.83\,RO_2 + b_{22}\,DIAL + b_{23}\,MGLY + b_{24}\,CO$
81	DIAL	+ OH	$\longrightarrow$	MCO_3
82	DIAL	+ $h\nu$	$\longrightarrow$	$HO_2 + CO + MCO_3$
83	CRES	+ OH	$\longrightarrow$	$0.2\,MGLY + 0.15\,RO_2P + 0.85\,RO_2R + RO_2$
84	CRES	+ NO_3	$\longrightarrow$	HNO_3 + BZO
85	RO_2P	+ NO	$\longrightarrow$	NPHE
86	RO_2P	+ HO_2	$\longrightarrow$	
87	RO_2P	+ RO_2	$\longrightarrow$	$0.5\,HO_2 + RO_2$
88	RO_2P	+ MCO_3	$\longrightarrow$	HCHO + HO_2
89	BZO	+ NO_2	$\longrightarrow$	NPHE
90	BZO	+ HO_2	$\longrightarrow$	
91	BZO		$\longrightarrow$	
92	NPHE	+ NO_3	$\longrightarrow$	HNO_3 + BZN2
93	BZN2	+ NO_2	$\longrightarrow$	
94	BZN2	+ HO_2	$\longrightarrow$	NPHE
95	BZN2		$\longrightarrow$	NPHE
96	H_2O_2	+ $h\nu$	$\longrightarrow$	2 OH
97	H_2O_2	+ OH	$\longrightarrow$	HO_2
98	MEOH	+ OH	$\longrightarrow$	HCHO + HO_2
99	CH_4	+ OH	$\longrightarrow$	HCHO + RO_2 + RO_2R
100	ISOP	+ OH	$\longrightarrow$	HCHO + ALD2 + RO_2R + RO_2
101	ISOP	+ O_3	$\longrightarrow$	$0.5\,HCHO + 0.65\,ALD2 + 0.21\,MEK + 0.16\,HO_2 + 0.29\,CO + 0.06\,OH + 0.14\,RO_2R + 0.14\,RO_2$
102	ISOP	+ O	$\longrightarrow$	$0.4\,HO_2 + 0.5\,MEK + 0.5\,ALD2$
103	ISOP	+ NO_3	$\longrightarrow$	NO_2 + HCHO + ALD2 + R_2O_2 + RO_2
104	ETOH	+ OH	$\longrightarrow$	ALD2 + HO_2
105	MTBE	+ 1.4 OH	$\longrightarrow$	$0.6\,TBF + 0.4\,HCHO + 0.4\,MEK + 1.4\,RO_2R + 0.4\,R_2O_2 + 1.8\,RO_2$
106	SO_2	+ OH	$\longrightarrow$	SO_3 + HO_2

Table 17. Chemical reactions simulated in the **AIRSHED** mechanism.

$$b_1 = 0.189\,x + 0.023\,(1 - x)$$
$$b_2 = 0.481\,x + 0.281\,(1 - x)$$
$$b_3 = 0.442\,x + 0.882\,(1 - x)$$
$$b_4 = 0.073\,x + 0.190\,(1 - x)$$

$$b_5 = 0.927\,x + 0.810\,(1 - x)$$
$$b_6 = 0.599\,x + 0.837\,(1 - x)$$
$$b_7 = b_5 + b_6$$

alkenes:

$$b_8 = y$$
$$b_9 = y + 2.00\,(1 - y)$$
$$b_{10} = 0.64\,y$$
$$b_{11} = 0.50\,y + 1.00\,(1 - y)$$
$$b_{12} = 0.13\,y + 0.27\,(1 - y)$$
$$b_{13} = 0.17\,y + 0.21\,(1 - y)$$
$$b_{14} = 0.06\,y + 0.12\,(1 - y)$$
$$b_{15} = 0.28\,y$$
$$b_{16} = 0.40\,y$$
$$b_{17} = (1 - y)$$
$$b_{18} = 0.40\,y$$
$$b_{19} = 0.20\,y$$
$$b_{20} = 0.20\,y + 0.40\,(1 - y)$$
$$b_{21} = 0.60\,y$$

and higher aromatics:

$$b_{22} = 0.650\,z + 0.490\,(1 - z)$$
$$b_{23} = 0.316\,z + 0.860\,(1 - z)$$
$$b_{24} = 0.095\,z$$

Photolytic reaction rates are taken from [86] at their maximum daily values:

$$\phi_1 = 4.974 \times 10^{-1}$$
$$\phi_2 = 1.122 \times 10^{0}$$
$$\phi_3 = 1.022 \times 10^{+1}$$
$$\phi_4 = 2.742 \times 10^{-2}$$
$$\phi_5 = 2.268 \times 10^{-3}$$
$$\phi_6 = 9.780 \times 10^{-2}$$
$$\phi_7 = 1.812 \times 10^{-3}$$
$$\phi_8 = 2.778 \times 10^{-3}$$
$$\phi_9 = 4.410 \times 10^{-4}$$
$$\phi_{10} = 9.480 \times 10^{-5}$$
$$\phi_{11} = 8.520 \times 10^{-3}$$
$$\phi_{12} = 3.174 \times 10^{-2}$$
$$\phi_{13} = 4.518 \times 10^{-4}$$

For ease in writing down the rate constants, the following three functions are used:

$$\alpha(a,b) = \frac{a}{T}\exp\frac{b}{T}$$
$$\alpha_2(a,b) = \frac{a}{T^2}\exp\frac{b}{T}$$
$$\beta(a,b) = a\exp\frac{b}{T}$$

along with the six variables

$$\gamma_1 = 4.40\times10^{17}/T\,(1.053\times10^{-11}\exp(\ -354/T)x + 1.62\times10^{-11}\exp(\ -289/T)(1-x))$$
$$\gamma_2 = 4.40\times10^{17}/T\,(4.85\ \times10^{-12}\exp(\ \ 504/T)y + 1.01\times10^{-11}\exp(\ \ 549/T)(1-y))$$
$$\gamma_3 = 4.40\times10^{17}/T\,(1.32\ \times10^{-14}\exp(-2105/T)y + 9.08\times10^{-15}\exp(-1137/T)(1-y))$$
$$\gamma_4 = 4.40\times10^{17}/T\,(1.18\ \times10^{-11}\exp(\ -324/T)y + 2.26\times10^{-11}\exp(\ \ 10/T)(1-y))$$
$$\gamma_5 = 4.40\times10^{17}/T\,(5.00\ \times10^{-12}\exp(-1935/T)y + 1.00\times10^{-11}\exp(\ -975/T)(1-y))$$
$$\gamma_6 = 4.40\times10^{17}/T\,(1.66\ \times10^{-11}\exp(\ \ 116/T)z + 6.20\times10^{-11}(1-z))$$

Using these definitions, the actual reaction rate constants used are shown in Table 18. All the reaction rates are derived using the standard mass-action formulation, by multiplying the concentrations of the species appearing on the left-hand side raised to the power specified by any stoichiometric coefficient, except for reaction 104, which ignores the factor 1.4 on OH. Reaction 21 is odd in that NO_2 is involved in determining the rate of the reaction, but is not actually consumed. Both reactions 32 and 34 are catalyzed by the presence of water, and their rates are multiplied by the constant concentration of H_2O.

Initial conditions were constructed from the input files used by the original grid-based solver, and example fields for basis construction were taken from previous runs of the **AIRSHED** model. The first requirement in choosing the appropriate number of fields in an expansion of each species is that the initial conditions are all well-satisfied. Any initial errors will grow unchecked over the course of the solution, except for special cases such as when the concentration will head toward zero. The initial expansion lengths were chosen using the plots in Figure 40, which show the logarithm base ten of the error generated by using a certain number of fields in the expansion, from one to 24 in this case. As is expected, the error drops to zero monotonically as the number of fields is increased, except for a few small glitches due to machine precision losses.

The magnitude of error is a product of the initial magnitude of the field and the errors plotted in the figures. Hence, it is necessary to consider the L^2 norms of the initial condition fields. These are shown in Table 19, along with the number of fields that would be required to satisfy a 1% error tolerance. This is only an estimate, as the future behavior of the fields (large or small growth or disappearance) will vary the amount of impact each initial condition error will have.

Before attempting to run the automatically generated code to implement the **AIRSHED** system of chemical reactions it is worth doing a small check that the fields selected using the initial conditions are sufficient to capture most of the required projections. For a smaller example, the reaction

$$A + B \xrightarrow{k_1} C,$$

where each species has a value over a spatial domain, gives the equation

$$\frac{dC(\mathbf{x})}{dt} = k_1 A(\mathbf{x})B(\mathbf{x})$$

1	ϕ_1	37	$441.0/T$	72	$\alpha(5.28 \times 10^3, -2634)$
2	$\alpha(6.30 \times 10^4, 1282)$	38	$1.32 \times 10^6/T$	73	$\alpha(4.58 \times 10^6, -792)$
3	$4.083 \times 10^6/T$	39	ϕ_7	74	$\alpha(8.81 \times 10^5, -2925)$
4	$\alpha(48890, 894)$	40	ϕ_8	75	γ_2
5	$\alpha(7.93 \times 10^5, -1370)$	41	$3.93 \times 10^6/T$	76	γ_3
6	$\alpha(5.285 \times 10^4, -2450)$	42	$\alpha(2.64 \times 10^5, -2060)$	77	γ_4
7	$\alpha(3.52 \times 10^6, 252)$	43	$4.41 \times 10^3/T$	78	γ_5
8	$\alpha(7.22 \times 10^{-3}, 529)$	44	$\alpha(3.039 \times 10^6, 250)$	79	$\alpha(9.25 \times 10^5, 322)$
9	$\alpha(2.035 \times 10^5, 273)$	45	ϕ_9	80	γ_6
10	$\beta(7.98 \times 10^{16}, -11379)$	46	$\alpha(1.32 \times 10^5, -1427)$	81	$1.32 \times 10^7/T$
11	$4.41 \times 10^{-4}/T$	47	$\alpha(1.85 \times 10^6, 180)$	82	ϕ_{12}
12	$\alpha(1.10 \times 10^4, -1229)$	48	$\alpha(1.23 \times 10^6, 180)$	83	$1.76 \times 10^7/T$
13	ϕ_2	49	$1.32 \times 10^6/T$	84	$9.66 \times 10^6/T$
14	ϕ_3	50	$1.1 \times 10^6/T$	85	$\alpha(1.85 \times 10^6, 180)$
15	ϕ_4	51	$\beta(1.2 \times 10^{18}, -13542)$	86	$1.32 \times 10^6/T$
16	ϕ_5	52	ϕ_{10}	87	$441.0/T$
17	$9.69 \times 10^7/T$	53	$\alpha(5.285 \times 10^6, -745)$	88	$1.32 \times 10^6/T$
18	4.32×10^{10}	54	ϕ_{11}	89	$6.62 \times 10^6/T$
19	$\alpha(1.78 \times 10^5, 833)$	55	$7.48 \times 10^6/T$	90	$1.32 \times 10^6/T$
20	ϕ_6	56	$\alpha(1.321 \times 10^5, -1427)$	91	0.06
21	$1.76 \times 10^{-6}/T$	57	γ_1	92	$1.68 \times 10^6/T$
22	$\alpha(4.22 \times 10^5, 737)$	58	$\alpha(9.65 \times 10^6, -709)$	93	$6.62 \times 10^6/T$
23	$\alpha(4.14 \times 10^3, 778)$	59	$\alpha(1.85 \times 10^6, 180)$	94	$1.32 \times 10^6/T$
24	$9.60 \times 10^4/T$	60	$1.32 \times 10^6/T$	95	0.06
25	$\alpha(7.05 \times 10^5, -942)$	61	$441.0/T$	96	ϕ_{13}
26	$\alpha(1.63 \times 10^6, 240)$	62	$1.32 \times 10^6/T$	97	$\alpha(1.36 \times 10^6, -187)$
27	$\alpha(4.493 \times 10^4, 773)$	63	$\alpha(1.85 \times 10^6, 180)$	98	$2.81\,T\exp(148/T)$
28	$\beta(2.61 \times 10^{15}, -10103)$	64	$1.32 \times 10^6/T$	99	$3.06\,T\exp(-1282/T)$
29	$1.76 \times 10^6/T$	65	$441.0/T$	100	$\alpha(1.12 \times 10^7, 410)$
30	$\alpha(6.17 \times 10^3, -579)$	66	$1.32 \times 10^6/T$	101	$5.41 \times 10^3, -2013)$
31	$\alpha(1.00 \times 10^5, 771)$	67	$\alpha(1.85 \times 10^6, 180)$	102	$2.64 \times 10^7/T$
32	$\alpha_2(1.054, 2971)$	68	$1.32 \times 10^6/T$	103	$\alpha(1.12 \times 10^7, -1121)$
33	$\alpha(1.00 \times 10^5, 771)$	69	$441.0/T$	104	$2.72\,T\exp(532/T)$
34	$\alpha_2(1.054, 2971)$	70	$1.32 \times 10^6/T$	105	$3.00\,T\exp(460/T)$
35	$\alpha(1.85 \times 10^6, 180)$	71	$\alpha(9.47 \times 10^5, 411)$	106	$4.0 \times 10^5/T$
36	$1.32 \times 10^6/T$				

Table 18. Reaction rate constants corresponding to the reactions in Table 17.

which, when expanding each species in terms of basis functions over the domain and scalar coefficients, becomes

$$\frac{dc_i}{dt} = k_1 \sum_{jk} a_j b_k \langle \mathbf{A}_j(\mathbf{x})\mathbf{B}_k(\mathbf{x}), \mathbf{C}_i(\mathbf{x}) \rangle.$$

There are two ways in which this equation may produce inaccurate results. First, it may just be wrong, through some error in derivation or a misalignment in a data structure or other problem. Due to the high degree of complexity required in deriving an integration scheme using a basis space expansion, this problem may arise more frequently than expected. The more insidious source of error comes from projection losses. The term

$$\sum_{jk} a_j b_k \mathbf{A}_j(\mathbf{x})\mathbf{B}_k(\mathbf{x})$$

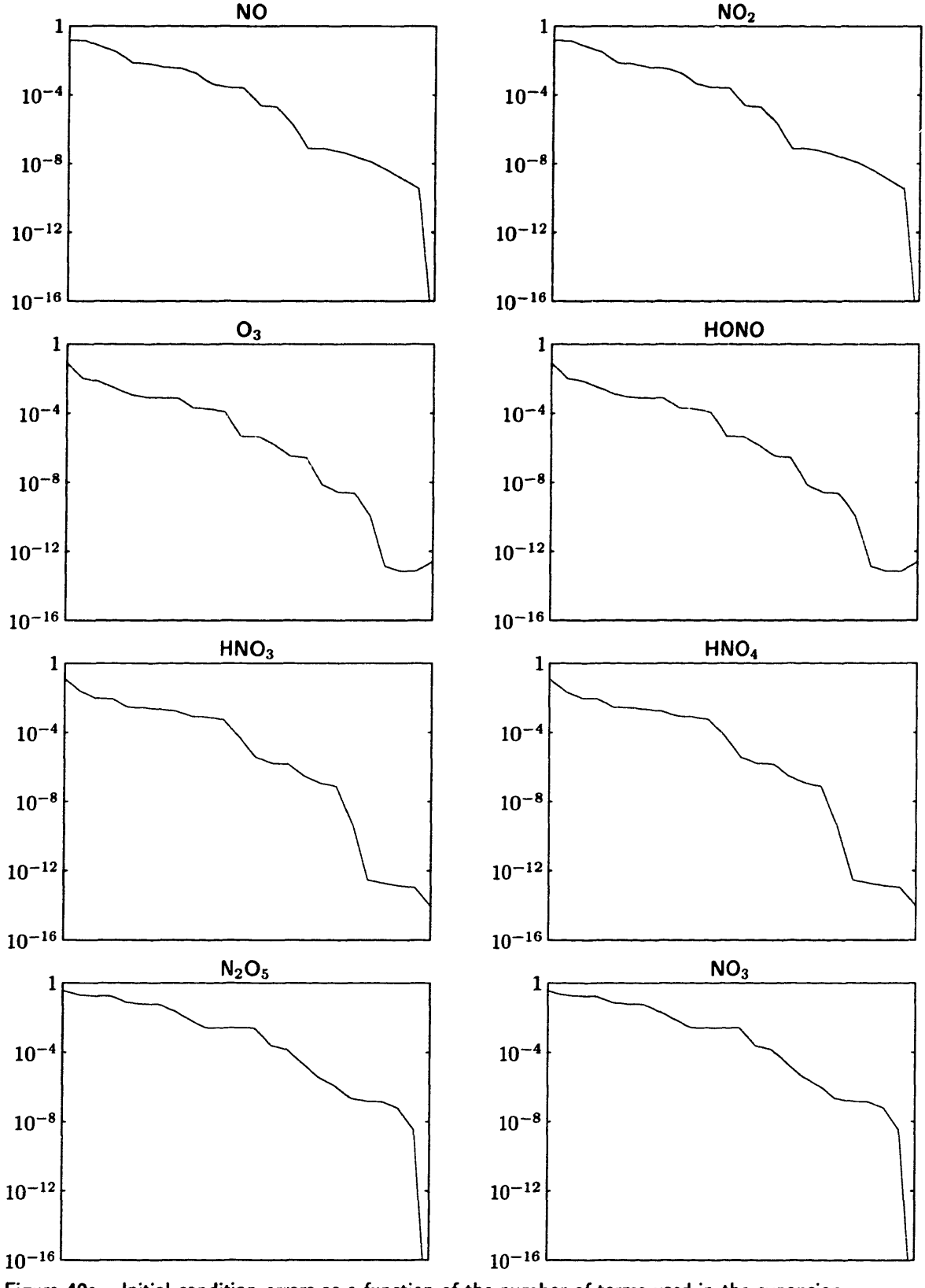


Figure 40a. Initial condition errors as a function of the number of terms used in the expansion.

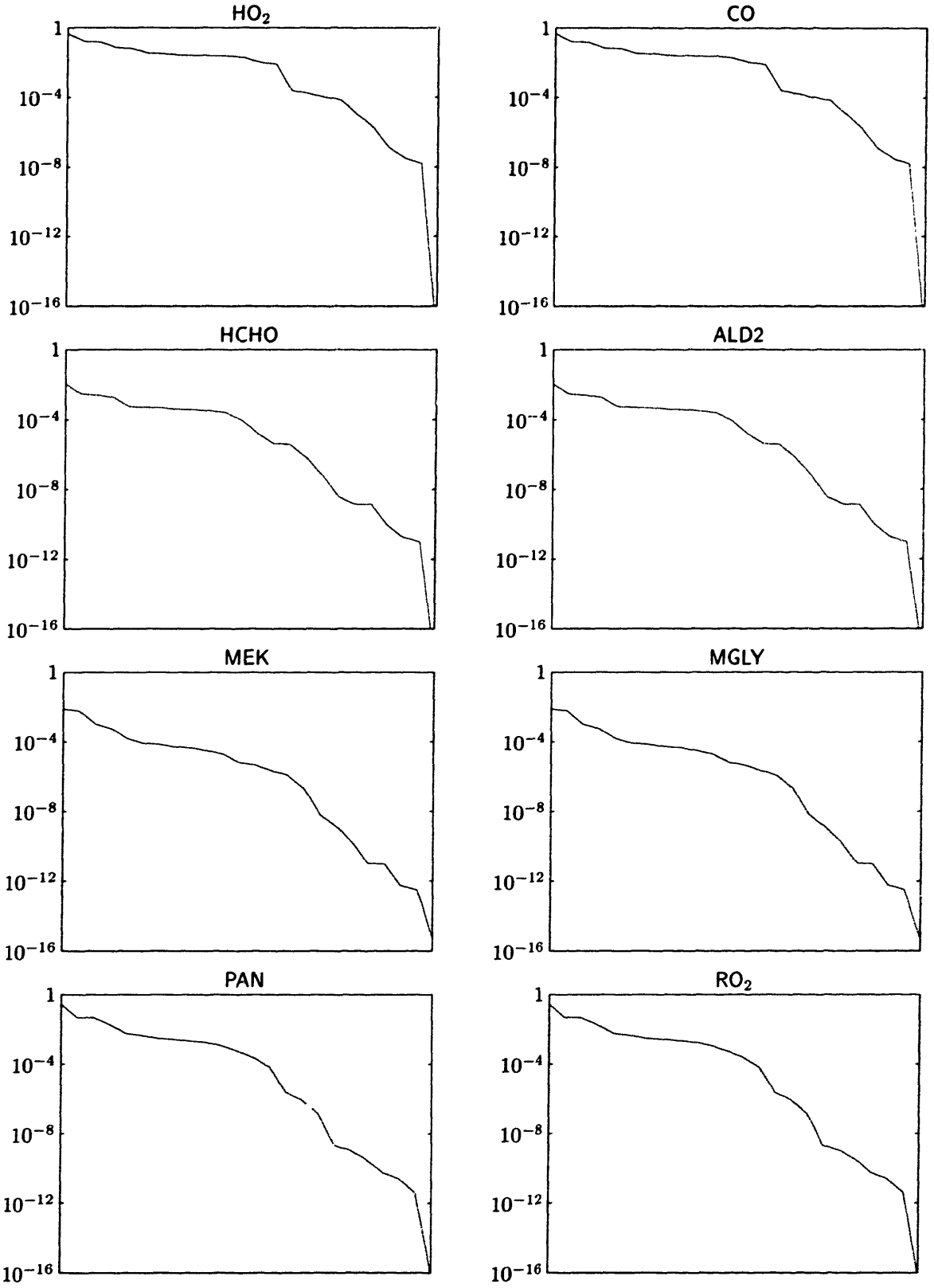


Figure 40b. Initial condition errors as a function of the number of terms used in the expansion.

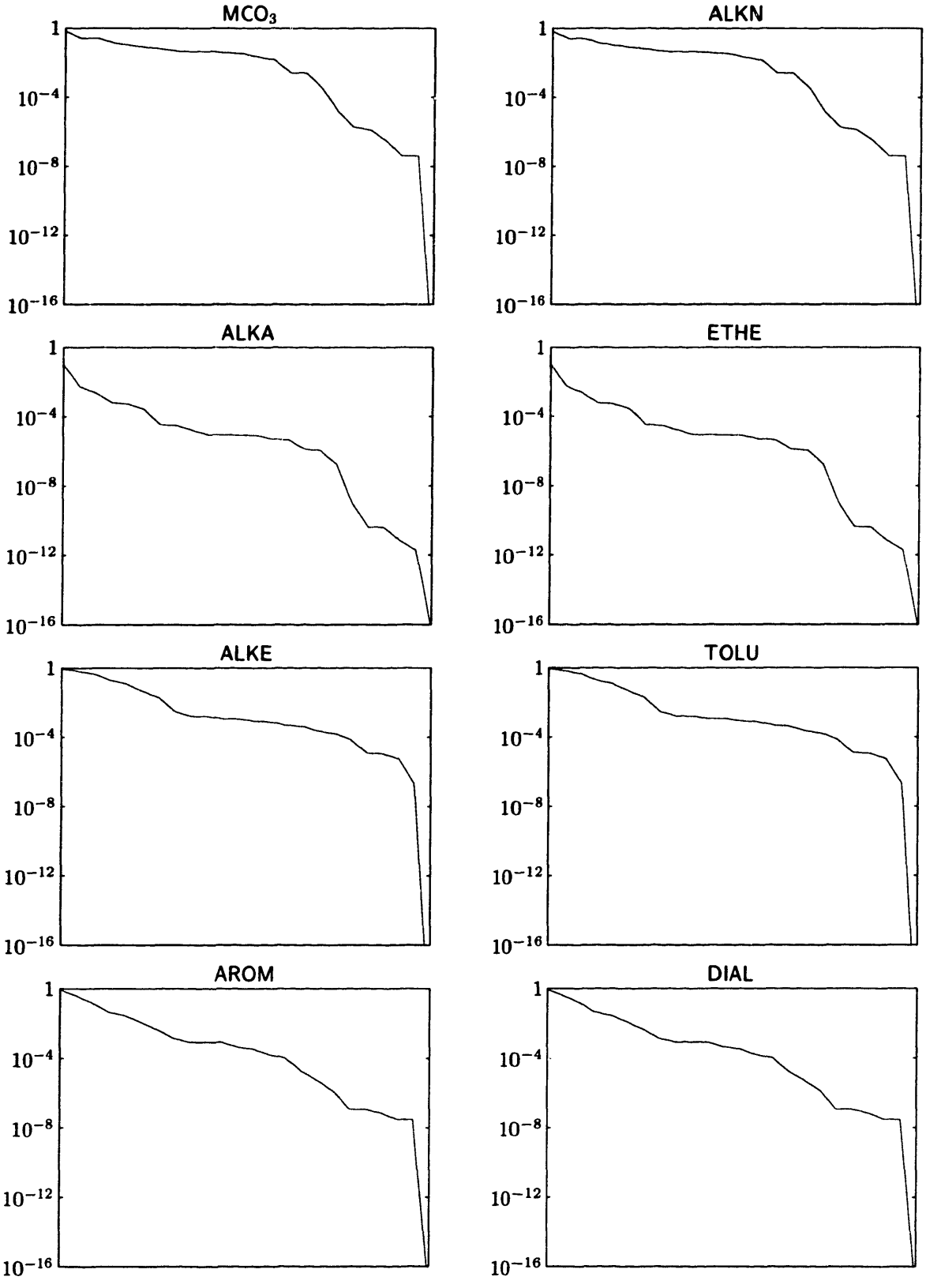


Figure 40c. Initial condition errors as a function of the number of terms used in the expansion.

CO	1.6×10^{3}	3
O_3	1.0×10^{1}	6
ALKA	2.2×10^{0}	3
NO_2	8.9×10^{-1}	3
ALD2	2.5×10^{-1}	2
HCHO	2.3×10^{-1}	1
MEK	2.2×10^{-1}	1
NO	1.4×10^{-1}	3
ETHE	6.2×10^{-2}	2
PAN	6.0×10^{-2}	2
HNO_3	5.5×10^{-2}	1
TOLU	5.0×10^{-2}	1
AROM	1.6×10^{-2}	2
ALKE	9.8×10^{-3}	1
MGLY	2.8×10^{-3}	1
DIAL	1.1×10^{-3}	1
ALKN	3.4×10^{-4}	1
HONO	6.6×10^{-5}	1
HNO4	7.5×10^{-6}	1
HO_2	2.2×10^{-6}	1
RO_2	1.6×10^{-6}	1
MCO_3	1.7×10^{-7}	1
N_2O_5	7.3×10^{-8}	1
NO_3	5.1×10^{-10}	1

Table 19. L^2 norms of the initial condition fields for the nonzero species, sorted by magnitude, with initial expansion length.

may not be completely representable in terms of the basis functions for C, that is, the term

$$\sum_i \alpha_i \mathbf{C}_i(\mathbf{x})$$

will not be the same as that in terms of A and B fields, where the $\{\alpha_i\}$ are the optimally chosen Fourier coefficients:

$$\alpha_i = \left\langle \sum_{jk} a_j b_k \mathbf{A}_j(\mathbf{x}) \mathbf{B}_k(\mathbf{x}), \mathbf{C}_i(\mathbf{x}) \right\rangle .$$

The numerical implementation of these two tests is illustrated in Figure 41.

Starting from an initial set of coefficients $\{a_i\}$, and given the basis which is used in calculating the value of $y(\mathbf{x})$ at any point, the left vertical branch in the figure illustrates the indirect method of calculating the update pointwise using the underlying grid, while the right branch applies the direct method which is under test. The two routines `grid-ydot` and `bsi-ydot` are generated automatically given the mechanism file (by calling `parse` with the `-d` option). The former uses the evolution operator at each point $\mathbf{x}$ in turn to calculate $\frac{dy}{dt}$, while the latter has no concept of physical space and generates the change in coefficient values $\frac{da_i}{dt}$ directly. The input field to `grid-ydot` is calculated by expanding the initial coefficients using the basis, and the output is reduced to coefficients by projecting the resulting time derivative field back onto the basis. The two sets of coefficients thus calculated can be compared to yield "coefficient errors," which will be within machine precision tolerance of zero if there are no structural errors in the equations. This test is

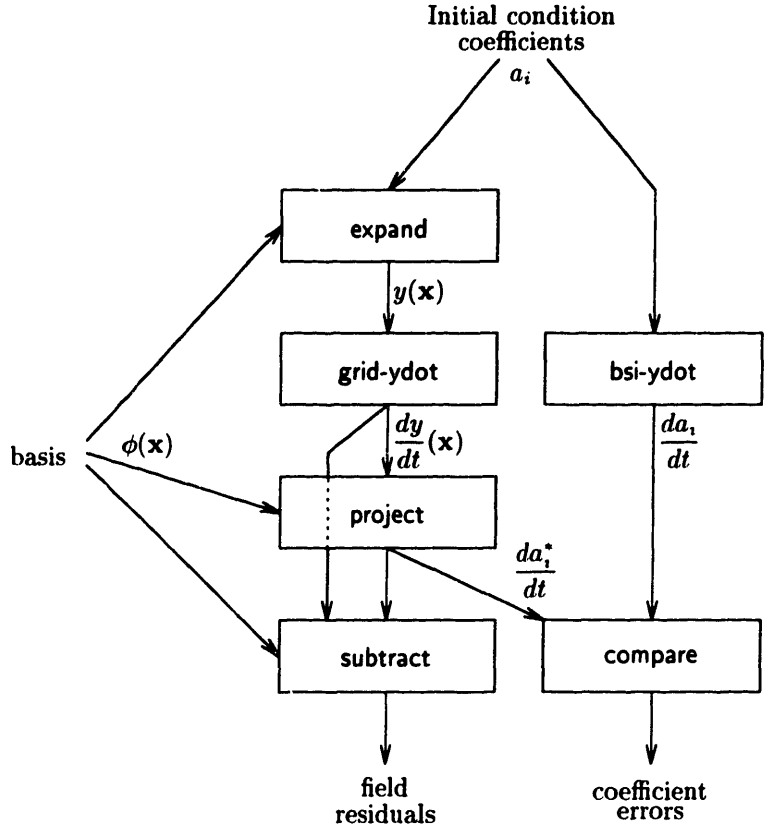


Figure 41. Schematic diagram of steps performed by the debug code in quantifying projection errors.

the first source of error described above, and fixes to problems discovered by this test are not incremental in nature, but will involve various checks to discover the gross algorithmic or data handling mismatch.

The second source of error is caused by projection disagreements and does not stem from a major problem in selecting the fields, but is inherent in the basis space integration procedure itself. Projection errors are calculated by

$$\text{field-residual}(\mathbf{x}) = \frac{dy}{dt}(\mathbf{x}) - \sum_i \frac{da_i^*}{dt}\phi(\mathbf{x}),$$

where the coefficients $\frac{da_i}{dt}$ could have been used to produce the same result once the first source of error has been eliminated. The energy norm of this residual field gives an indication as to the magnitude of error.

The figure shows the procedure for testing the errors for one particular species and one reaction which contributes to that species, but in reality output must be produced for each species, and for each reaction which affects the contribution of that species. The program **debug**, combined from a controlling file for initialization and output, and the two automatically generated evolution operator functions from **parse -d**, performs a loop over all the species and all the reactions which affect the species one at a time. For the **AIRSHED** model it produced 115 lines indicating projection errors on a per-component, per-reaction basis. Errors below a certain cutoff magnitude (10^{-10} in this example) were ignored. As an illustration, after sorting over the error field, the first few lines of the output are as follows:

```
O     0 9.032015e-05
NO    0 6.140088e-05
NO    2 4.607575e-05
O3    1 3.776725e-05
NO2   3 2.886396e-05
HO2   2 2.063415e-05
```

The first line indicates that the result for the contribution of the first reaction (of those which affect O) to species O is low by the amount 9.0×10^{-5}, an L^2 norm over the residual field. No output was generated due to the coefficient errors test, once the algorithmic bugs were eliminated.

Choosing which of the reported field errors to eliminate and the choice of initial condition coefficients requires a physical understanding of the problem, but the numerical work is simple once a source of error is chosen. In this case, the error to species O from its zeroth reaction was initially picked for elimination. The field (not just its residual) of this error was dumped to a file, scaled to have unit norm, and appended to the list of basis elements for O, which is allowable since the residual field is orthogonal to the original basis of O by definition. Re-running the **debug** code shows that the first line has vanished from the list of output errors confirming that particular source had been eliminated. Since NO and O_3 are important observed variables as well as for illustrative purposes, the errors in NO due to its zeroth reaction and in O_3 due to its first were also eliminated using the same procedure.

The stopping criterion for this error elimination procedure is not easily written down. Adding basis elements *ad infinitum* is guaranteed to reduce the error, only at a sacrifice of execution speed. The resolution to this classic tradeoff relies on one's initial error tolerance, and the errors actually generated during a simulation, which correlate with the field residuals but cannot be determined exactly from them. Thus it is necessary to sample coefficient values representative of the entire simulation. One good way to do this is to generate the coefficients of the original fields in terms of the chosen bases and to run **debug** on each set, looking for patterns and unusually large values. Alternatively, a brute force method of simply running the simulation in both a grid-based and field-based domain and comparing will give a larger amount of information. That is the approach chosen for this example, where the scaled L^2 norms of the field differences between the "exact" (grid-based) and "approximate" (field-based) simulations are plotted at each time step for all the species. The norm of each error field is scaled by the average value of the fields so that it can be interpreted as a percentage mismatch. Figure 42 shows the plot for only those species which violate a 5% absolute error tolerance at any time step.

From looking at the figure, the error in lumped aromatics appears to grow unboundedly, but intimate knowledge of the underlying model shows this to be irrelevant. The only reaction in which AROM participates is number 80, in which it reacts with OH to form a variety of products: CRES, HO_2, RO_2R, RO_2, DIAL, MGLY, and CO; however, the maximum concentration of AROM over the entire course of integration is less than .01 ppm and that of OH is less than 10^{-6} ppm, so the contribution of that reaction under the chosen conditions is negligible. Furthermore, it is understood that AROM is not an interesting observed variable. The constant 10% error in ALKE also holds no significance. That species is never created, just reacts away from an initial average concentration of 10^{-3} ppm down to 10^{-12} at the end. Ten percent of nothing is still nothing, as the old saying goes. Both these discrepancies could be eliminated by the addition of another one or more basis elements to each species, but the result is not worth the cost in calculation time in light of the previous discussion. The species to be worried about is NO, which shows a large rise and a strange abrupt drop in error at hour 18. Running **debug** using the conditions at hour 17 reveals NO to be at the top of the list, as a result of inaccuracies from its second reaction, number 5. Adding the residual field to the basis reduces the error in NO to below the tolerance level, although the errors in nitrophenol and benzaldehydes shoot up due to their close coupling with NO. These errors are dismissed for the same reasons presented earlier in the paragraph.

The resulting basis-space implementation of the **AIRSHED** chemistry produces results that are

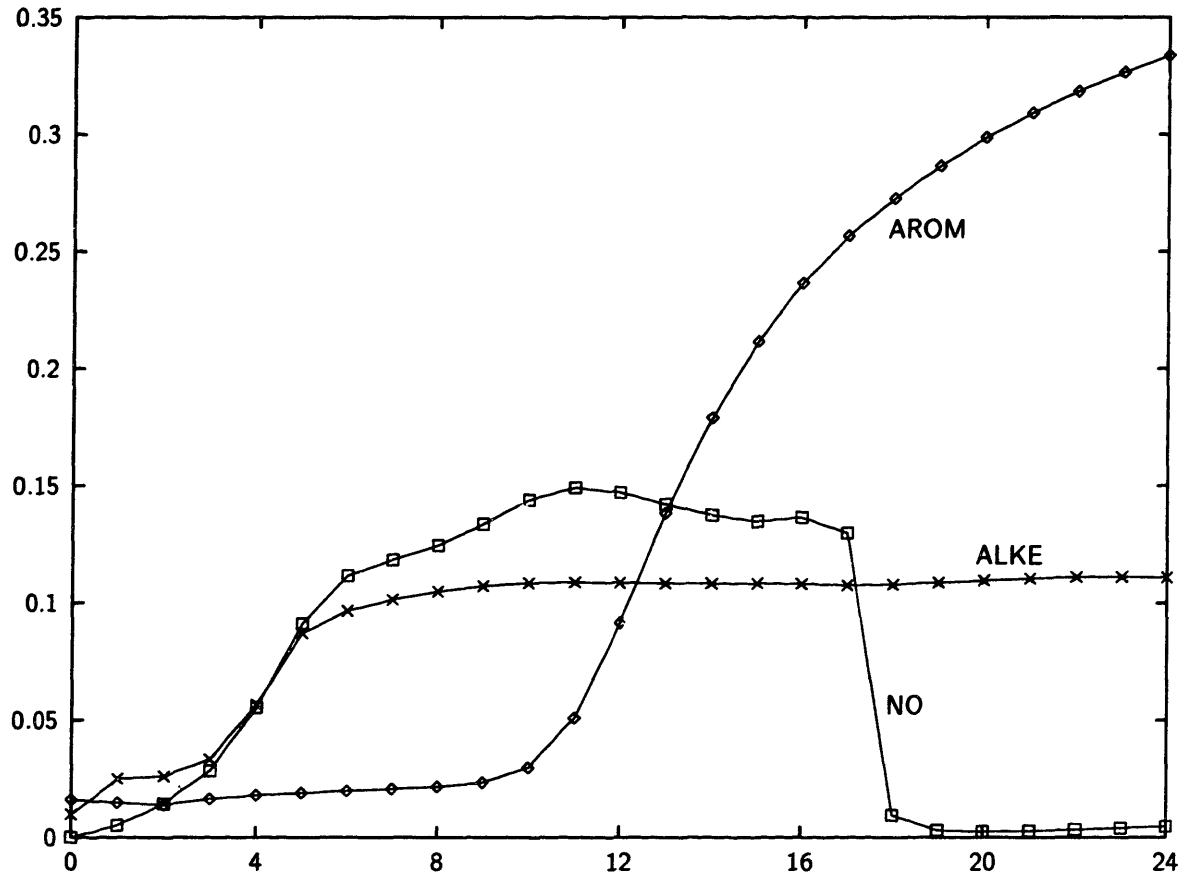


Figure 42. Selected scaled L^2 field differences between grid- and field-based simulations.

within the prescribed tolerances, and solves the 106 reactions of 42 variables at 2400 grid points using only 65 coefficients. The vital statistics are shown in Table 20. The solution strategies in the first two columns of the table both used LSODE as the underlying integrator, with a numerically derived Jacobian, and identical error tolerances. The third column shows statistics for the stripped-down high-performance solver in the actual AIRSHED code itself, for which there are no robust error tolerances. One interesting ratio to compute is the approximate time required for one function call. For the grid-based method, this is 8.3×10^{-5} second, but the field-based method requires 1.5×10^{-3} second, so much more work is being done per call in the field-based simulation, but many fewer calls need to be made. The ratio for the optimized AIRSHED solver is about a third that of the LSODE solver, but more function calls are required, perhaps since no Jacobian is calculated. An interesting extension to this comparison would be to write down symbolic Jacobian matrices for the grid- and field-based integration schemes to use. This usually has the effect of reducing the number of function calls required to solve the problem, and would yield more improvement to the simulation scheme which has the more expensive function calls.

	Field-based	Grid-based	AIRSHED grid-based
Dimension	65	100 800	79 200
Function evaluations	8 835	5 026 671	12 867 590
Memory (kbytes)	38.1	41 156.3	1 237.5
CPU time (seconds)	13.1	418.8	392.7

Table 20. Comparison of three different solution methodologies applied to the AIRSHED chemistry.

11.3 Burgers' equation

The canonical example of nonlinear convection-diffusion is Burgers' equation, which challenges numerical solvers due to the formation of steep fronts leading to large nonphysical oscillations. Originally the equation was posed as a model for statistical theory of turbulent fluid motion, but Burgers noted that, "phenomena pictured by the solutions of this equation are far removed from hydrodynamic turbulence." [28] Cole [33] saw that Burgers' equation modeled flow through a shock wave in a compressible viscous fluid since it exhibits a non-linear term tending to steepen the wave fronts and produce complete dissipation as well as a viscous term of higher order which prevents formation of actual discontinuities and which tends to diffuse any differences in velocity. Fletcher [47] reports extensive solutions to Burgers' equation using many different techniques, while the initial credit for an analytic solution goes to both Cole [33] and Hopf [65].

The equation is

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \varepsilon\frac{\partial^2 u}{\partial x^2}, \qquad (x \in (0,1), t > 0)$$

with initial condition

$$u(x,0) = u_0(x)$$

and homogenous boundary conditions

$$u(0,t) = u(1,t) = 0 \qquad (t \geq 0)$$

where ε serves the role of an inverse Reynolds number, modulating the flow from purely convective ($\varepsilon \searrow 0$) to purely diffusive ($\varepsilon \nearrow \infty$). The parameter ε also changes the character of the differential equation from officially parabolic when the diffusive term is fairly large to practically hyperbolic when its effect is small.

To transform this differential equation into the space of basis functions generated above, the independent variable u is first written in terms of the expansion,

$$\frac{\partial}{\partial t}\left(\sum_i u_i \mathbf{U}_i\right) = -\left(\sum_j u_i \mathbf{U}_i\right)\frac{\partial}{\partial x}\left(\sum_k u_i \mathbf{U}_i\right) + \varepsilon\frac{\partial^2}{\partial x^2}\left(\sum_j u_i \mathbf{U}_i\right)$$

Now, noticing that the coefficients u_i are functions only of time, and the fields $\mathbf{U}_i$ do not change during the time integration, differentiation can be moved inside the summations,

$$\sum_i \dot{u}_i \mathbf{U}_i = -\sum_{jk} u_j u_k \mathbf{U}_j * \frac{d\mathbf{U}_k}{dx} + \varepsilon \sum_j u_j \frac{d^2\mathbf{U}_j}{dx^2},$$

where '$*$' represents pointwise multiplication of two spatial fields. Taking the inner product of both sides with $\mathbf{U}_i$ and using orthonormality gives the expression for the time derivative of each coefficient,

$$\dot{u}_i = -\sum_{jk} u_j u_k \left\langle \mathbf{U}_j * \frac{d\mathbf{U}_k}{dx}, \mathbf{U}_i \right\rangle + \varepsilon \sum_j u_j \left\langle \frac{d^2\mathbf{U}_j}{dx^2}, \mathbf{U}_i \right\rangle.$$

The expressions inside angle brackets are scalars which are calculated offline, before the start of an actual simulation, using discrete formulas for the spatial derivatives of the fields.

A finite difference integrator was developed for this system to be used for comparisons with the results of field-based integration using the formula above. Ames [6] provides a method for generating bounds on the temporal and spatial step sizes for explicit difference equations to ensure convergence and stability, but in practice these bounds are too severe, and appropriate step sizes were discovered experimentally so that the finite difference integrator would not appear to be impossibly slow. The step sizes in the field-based integration could be taken much larger as the fundamental nature of

ε	dx	method	function evaluations	memory (kbytes)	time (seconds)
1.	.1	grid	838	34.8	1.8
		field	144	.5	0.1
.1	.1	grid	497	34.8	0.9
		field	122	.5	0.1
1.	.01	grid	7001	3128.0	1075.0
		field	54	.5	0.5
.1	.01	grid	5723	3128.0	799.8
		field	84	.5	0.5
.01	.01	grid	6405	3128.0	1051.4
		field	60	.5	0.5

Table 21. Burgers' equation execution statistics.

the system of differential equations has changed, in a computationally advantageous way. As an example of the different execution times, Table 21 was calculated using LSODE as the time integrator in both techniques to provide the same basis for comparison.

The statistics for the field method presented in Table 21 were generated by using the outputs from the corresponding grid-based integration, guaranteeing that the generated basis used for calculation contained all the information necessary for that particular value of ε. Only three basis fields (of the 62 or 628 available, depending on dx) were needed in all the field-based integration runs to achieve accuracy within 0.1%, suggesting that the basic spatial structure of the independent variable is not shifting much over time. This is indeed the case, as can be seen in Figure 45, which show $u(x)$ as a function of time for two different parameter values.

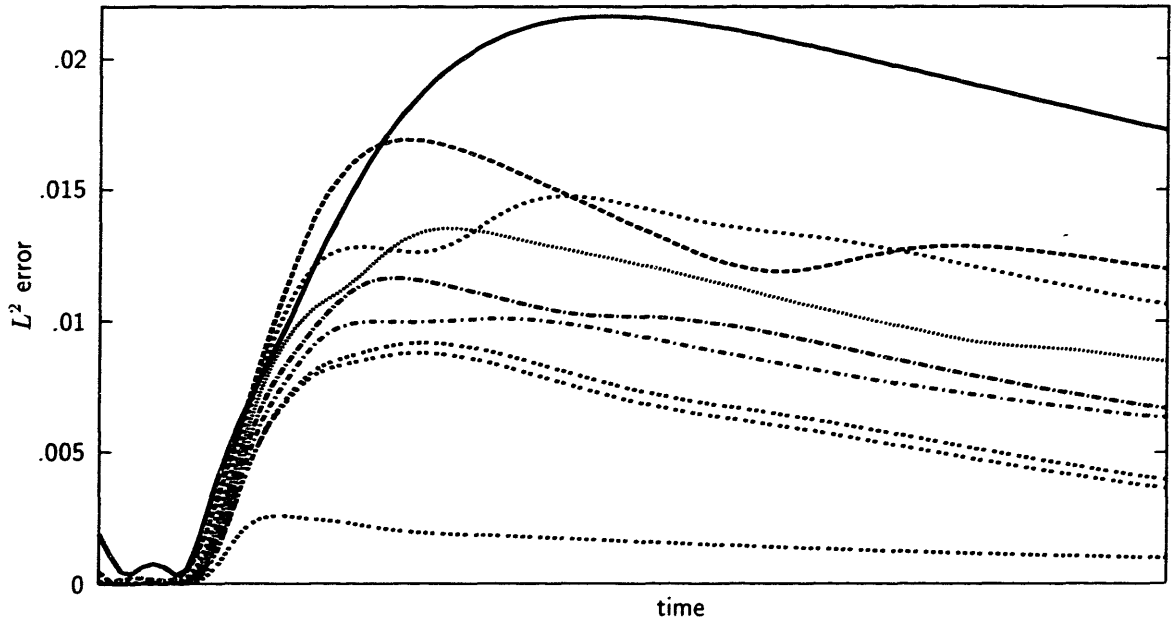


Figure 43. Multiple simulations of Burgers' equation with $\varepsilon = .01$ using fields generated at $\varepsilon = .1$, at different expansion lengths.

An obvious question to ask is: how does the choice of basis fields affect the accuracy of an integration? What is seen in practice for this equation is that basis fields produced using a given value of ε work quite well when simulating equal or higher values of ε. This can be explained by noticing that large values of ε correspond to a high degree of diffusion and a correspondingly smaller amount of

"information" in the solution. Small ε implies almost pure convection, with solutions featuring sharp gradients. The basis fields generated by these sharp gradients are quite sufficient in approximating the smoother solutions generated under conditions of high dissipation. Not unexpectedly, the other way around—using smooth basis fields to integrate low-diffusion equations—results in a large loss of accuracy, as shown in Figure 43. The different curves in the figure represent simulations using a different number of fields, from two through ten, with higher numbers producing lower errors. It is clear that the actual solution is significantly outside the space spanned by the first few basis fields, but adding more decreases the overall error since each new member of the basis is orthogonal to all the others. To illustrate the differences, Figure 44 shows the first three basis fields generated for the two values of ε under discussion, showing how the sharp front present at $\varepsilon = .01$ is not captured in the $\varepsilon = .1$ fields.

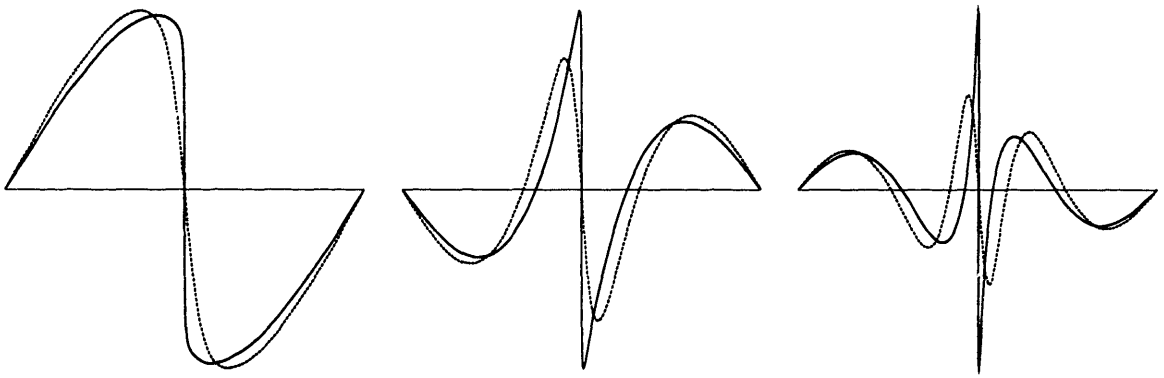


Figure 44. Comparison of the first three basis fields generated at $\varepsilon = .01$ (sharp) and $\varepsilon = .1$ (smooth).

Incidentally, one may wonder if the formulation presented above is the only possible on by which to integrate Burgers' equation in a field-based system. The answer, of course, is no since there are any number of equivalent ways to write the differential equation, leading to different integration techniques. For instance, many theorists are fond of writing Burgers' equation in the form

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial u^2}{\partial x} = \varepsilon \frac{\partial^2 u}{\partial x^2}$$

to emphasize the first derivative of an "energy" term. This transforms into basis space as

$$\dot{u}_i = -\frac{1}{2}\sum_{jk} u_j u_k \left\langle \frac{d\mathbf{U}_j \mathbf{U}_k}{dx}, \mathbf{U}_i \right\rangle + \varepsilon \sum_j u_j \left\langle \frac{d^2\mathbf{U}_j}{dx^2}, \mathbf{U}_i \right\rangle$$

which is completely equivalent to the earlier form *only in the limit of an infinite number of basis fields.* In practice, using finite differences as approximations to the first derivatives necessarily introduces some error into the calculation. Referring to the simulations run for Table 21, this alternate formulation is less accurate for the cases where $dx = .1$ (only 62 grid points) perhaps because multiplying the fields together before differentiating tends to soften the peaks unlike doing the multiplication afterwards. For the cases where $dx = .01$ (628 grid points), the two methods are indistinguishable in their results.

Another detail to notice is that no mention has been made as to the procedure used to generate the numerical first derivatives. The whole host of standard possibilities may be applied: forward-, backward-, centered-space explicit, implicit, Crank-Nicholson, and so on. The numbers presented above were generated using a centered-space explicit method, in fact, but others gave similar results. The usual considerations which govern the choice of a finite difference scheme still apply here and the interested reader is heartily referred to Strikwerda's little book [128] for a complete survey.

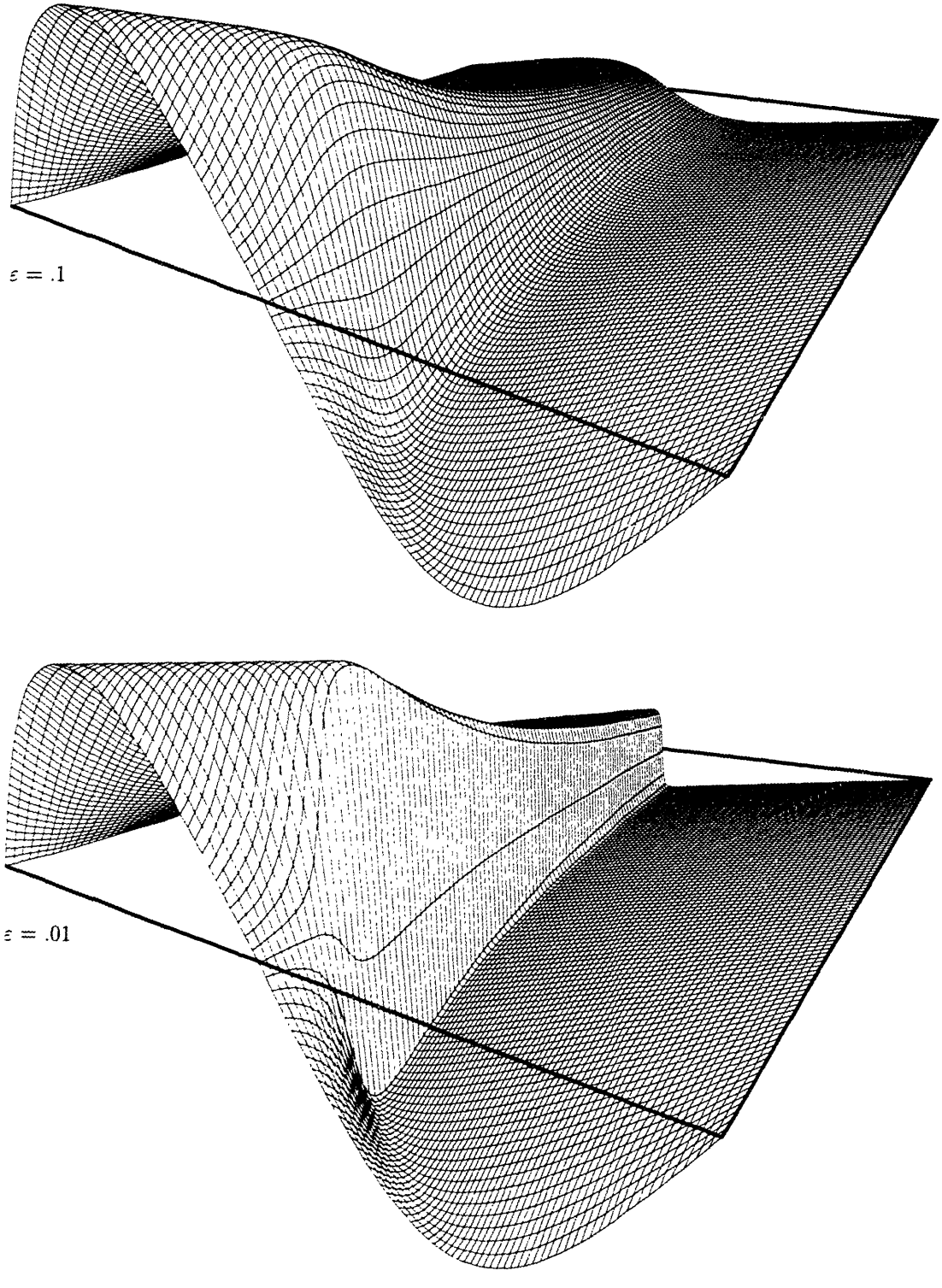


Figure 45. Plots of $u(x,t)$ for two different parameter values. The initial condition is identical in both cases, and is the curve closest to the viewer. As time progresses, diffusion reduces the height in both plots, but the lower ε shows a sharper front.

11.4 Bootstrap Burgers' equation

Considered in this section is the same Burgers' equation as in the previous section, only with different initial and boundary conditions, the presentation of which would amount to a fairly trivial extension would it not be for the fact that no previous simulations will be used in generating the initial basis. All the other examples start with the execution of a similar, grid-based model from which example fields are collected and presented to the Karhunen-Loève basis generator to produce the initial basis for simulation. That initial basis was then refined by locating imprecise regions of operation. Here, however, only engineering knowledge and intuition will be used to judge the validity of the results and instead of augmenting an initial basis, it will be pruned to speed the execution time while still providing reasonable results.

The equation, again for reference, is

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \varepsilon\frac{\partial^2 u}{\partial x^2}, \qquad (x \in (0,1),\, t > 0)$$

with the new initial condition

$$u(x,0) = \begin{cases} 1 & x < 0.5 \\ 0 & x \geq 0.5 \end{cases}$$

and constant boundary conditions

$$\begin{aligned} u(0,t) &= 1 \\ u(1,t) &= 0 \end{aligned} \qquad (t > 0)$$

which transforms as before into

$$\dot{u}_i = -\sum_{jk} u_j u_k \left\langle \mathbf{U}_j * \frac{d\mathbf{U}_k}{dx}, \mathbf{U}_i \right\rangle + \varepsilon \sum_j u_j \left\langle \frac{d^2\mathbf{U}_j}{dx^2}, \mathbf{U}_i \right\rangle.$$

The value $\varepsilon = 0.05$ will represent the parameter range of interest, at which the equation will be expected to be largely convective, but still exhibiting some amount of diffusive character. If it were purely convective, in the limit $\varepsilon \to 0$, the initial vertical shock front would propagate to higher x values without losing its sharpness. With this mild amount of diffusion, it will still move to the right, only the shape should soften as time progresses. Also, since the front is fed by the non-zero boundary condition on the left, the maximum value must remain constant at one, and the process of diffusion will never entirely be able to consume the initial bump. Instead at long times a balance between the translating front and the constant effect of diffusion will be reached which still satisifies the boundary conditions.

With no previous runs of this planned simulation, a basis must be generated from scratch. Given the above discussion, a few possible plots of $u(x)$ can be hand-drawn to help drive the process of basis selection, as shown in Figure 46. The steep initial condition must be satisfied, of course, and the other three curves resemble what is expected at intermediate times during the simulation: diffusion may cause some decrease in values to the left of $x = 0.5$, and the distance to which the diffusion invades depends on the exact value of ε. Also the front will be moving to the right and can be expected to occupy every position between the initial halfway starting position and the posited final equilibrium curve.

At this point considerations from the previous discussions could be used to guide basis generation, by adding the proposed fields one at a time while maintaining orthonormality of the basis. Instead, a basis which is complete in some underlying grid space will be chosen, and results from simulation in that basis will be used to generate a working basis which is more appropriate, that is, smaller. The discrete basis is defined as

$$\{\delta_i(x)\}_{i=1}^{N_x}, \qquad \delta_i(x) = \begin{cases} 1 & i = x \\ 0 & \text{otherwise,} \end{cases}$$

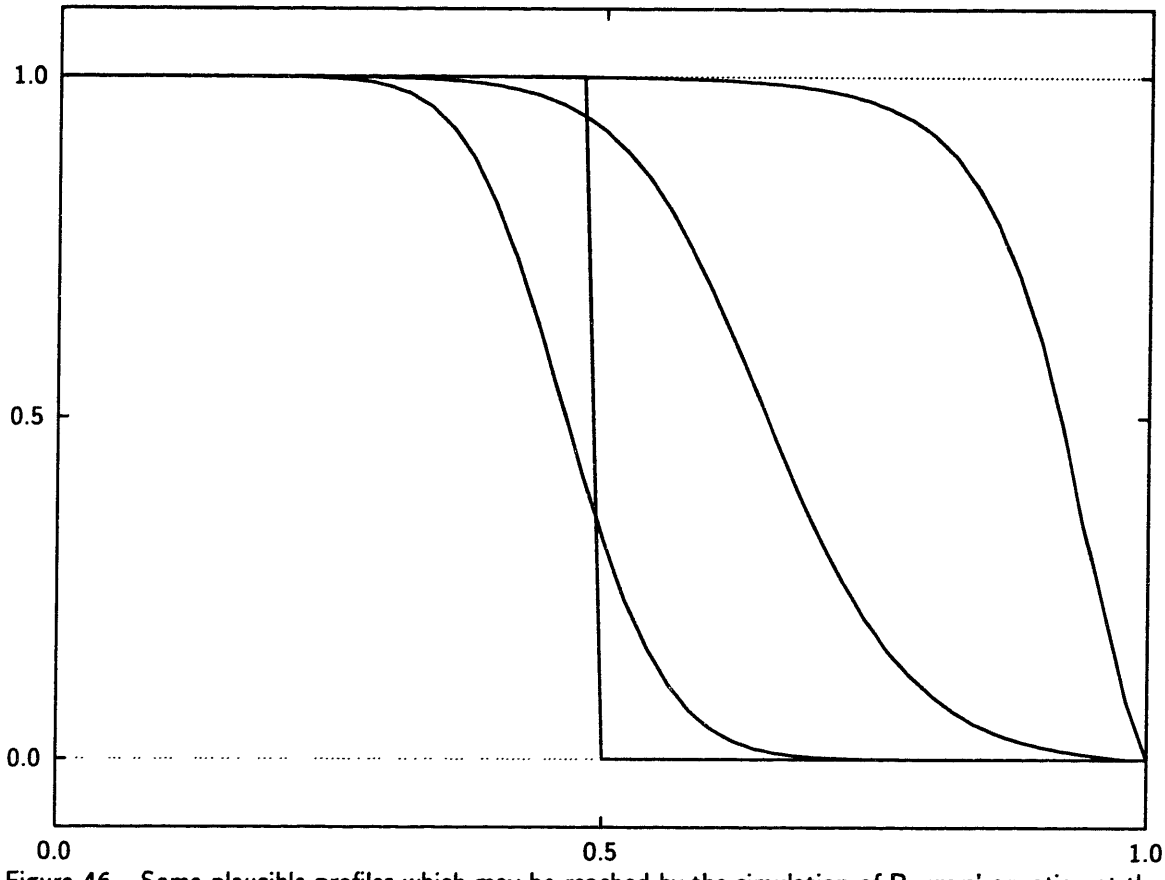


Figure 46. Some plausible profiles which may be reached by the simulation of Burgers' equation at the given conditions.

one function for each point in the grid. As in the previous example using Burgers' equation, finite differences will be used to construct the derivative bases given this discrete one. As a first attempt, centered finite differences will be used for the first and second spatial derivatives, defined by

$$\frac{d\delta_i}{dx}(x) = \frac{\delta_{i+1}(x) - \delta_{i-1}(x)}{2\Delta x}$$
$$\frac{d^2\delta_i}{dx^2}(x) = \frac{\delta_{i+1}(x) - 2\delta_i(x) + \delta_{i-1}(x)}{(\Delta x)^2},$$

where derivatives which are undefined due to being near the edges of the grid are set to zero, enforcing the boundary conditions. These are actually calculated automatically, by performing the numerical differences, as are the needed projection coefficients.

Basis space simulation using this first-cut complete discrete basis gives a surprising result however, which is shown in Figure 47. All runs used a grid consisting of 50 evenly spaced points in the single dimension, an ε of 0.05, and a time step of 10^{-4} over a temporal domain of 2 seconds. While it may initially appear that the shapes of these curves are as expected—smoothing out and moving to the right—they are actually the result of simulating a rather different equation. This can be seen by noticing that the integration of Burgers' equation using a complete set of discrete fields is in fact equivalent to the integration of a related equation on a grid, by substituting the equations for the discrete fields into the basis space integration update formula for $\dot{u}_i$. Using, for now, a centered-space finite difference approximation for the first spatial derivative, and considering only points away from the edges of the domain, the basis fields are (from above)

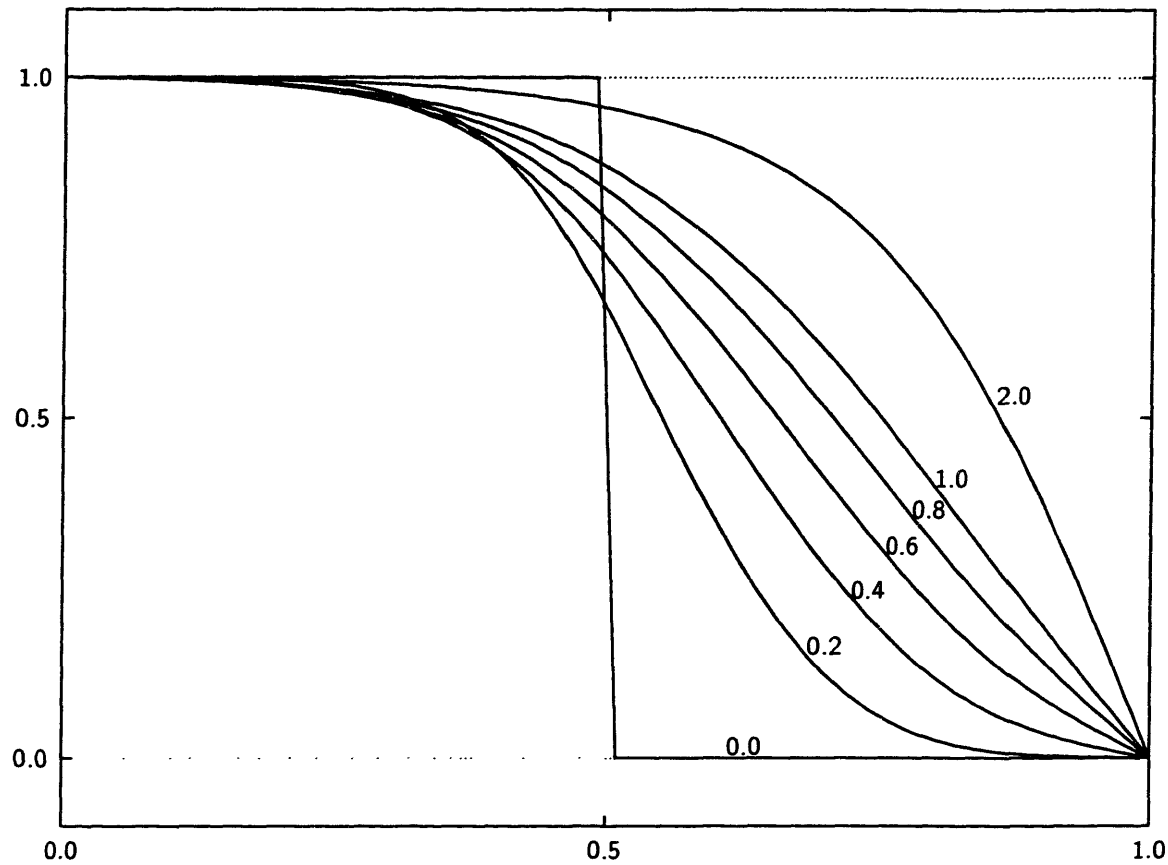


Figure 47. Initial (incorrect) attempt at basis space simulation using a discrete basis: $u(x,t)$ at seven different times.

$$\begin{aligned}
\mathbf{U}_i(x) &= \delta_i(x) \\
\frac{d\mathbf{U}_i}{dx}(x) &= \frac{1}{2\Delta x}\big(\delta_{i+1}(x) - \delta_{i-1}(x)\big) \\
\frac{d^2\mathbf{U}_i}{dx^2}(x) &= \frac{1}{(\Delta x)^2}\big(\delta_{i+1}(x) - 2\delta_i(x) + \delta_{i-1}(x)\big).
\end{aligned}$$

Deriving the required inner product terms involves integrating over the above functions, and uses the reduction from Dirac to Kronecker δ defined by

$$\int_X \delta_i(x)\delta_j(x)\,dx = \delta_{ij}$$

in the following:

$$\begin{aligned}
\frac{d\mathbf{U}_j}{dx} * \mathbf{U}_k &= \frac{1}{2\Delta x}\big(\delta_{j+1}(x)\delta_k(x) - \delta_{j-1}(x)\delta_k(x)\big) \\
\left\langle \frac{d\mathbf{U}_j}{dx} * \mathbf{U}_k, \mathbf{U}_i \right\rangle &= \frac{1}{2\Delta x}(\delta_{ik}\delta_{i,j+1} - \delta_{ik}\delta_{i,j-1}) \\
\left\langle \frac{d^2\mathbf{U}_j}{dx^2}, \mathbf{U}_i \right\rangle &= \frac{1}{(\Delta x)^2}(\delta_{i,j+1} - 2\delta_{ij} + \delta_{i,j-1}).
\end{aligned}$$

Substituting these into the evolution equation for a single coefficient u_i gives

$$\begin{aligned}
\frac{du_i}{dt} &= -\frac{1}{2\Delta x}\sum_{jk} u_j u_k(\delta_{ik}\delta_{i,j+1} - \delta_{ik}\delta_{i,j-1}) + \varepsilon\frac{1}{(\Delta x)^2}\sum_j u_j(\delta_{i,j+1} - 2\delta_{ij} + \delta_{i,j-1}) \\
&= -\frac{1}{2\Delta x}\sum_j u_j u_i(\delta_{i,j+1} - \delta_{i,j-1}) + \varepsilon\frac{1}{(\Delta x)^2}\sum_j u_j(\delta_{i,j+1} - 2\delta_{ij} + \delta_{i,j-1}) \\
&= -\frac{1}{2\Delta x}\sum_j u_i(u_j\delta_{i-1,j} - u_j\delta_{i+1,j}) + \varepsilon\frac{1}{(\Delta x)^2}\sum_j(u_j\delta_{i-1,j} - 2u_j\delta_{ij} + u_j\delta_{i+1,j}) \\
&= -\frac{1}{2\Delta x}u_i(u_{i-1} - u_{i+1}) + \varepsilon\frac{1}{(\Delta x)^2}(u_{i-1} - 2u_j + u_{i+1})
\end{aligned}$$

which is the centered-space finite difference equation for the grid-based simulation of the equation

$$\frac{\partial u}{\partial t} - u\frac{\partial u}{\partial x} = \varepsilon\frac{\partial^2 u}{\partial x^2},$$

just Burgers' equation but with the opposite sign on the convection term. This explains the shapes of the curves in Figure 47. The convective term of this related equation causes a negative contribution everywhere and attempts to reduce the value of the dependent variable, but it is balanced by the diffusion and the boundary conditions. Simulations of the true equation would also not be expected to show much reduction of the values near the zero boundary.

A similar derivation can be performed using a "downwind" first derivative,

$$\frac{du_i}{dx} = \frac{u_i - u_{i-1}}{\Delta x},$$

in the discrete basis space integration, which results in an "upwind" first derivative,

$$\frac{du_i}{dx} = \frac{u_{i+1} - u_i}{\Delta x},$$

simulation in grid space. The reverse is also true. The result of the downwind simulation is shown in Figure 48, and is exactly the result which would be obtained by using upwind differentiation in a gridded domain. These results are then used as a starting point for generating a smaller, but just as accurate, basis for calculation, by presenting the curves generated by simulation using the discrete basis to the Karhunen-Loève expansion generator.

The basis functions have the standard intuitive look to them, with higher numbered basis members exhibiting an increasing number of zero crossings, as shown in Figure 49. They also reflect the simulation results by maintaining a plateau in the left part of the domain. Simulations using various numbers of these new basis functions give increasingly better results, with six elements being sufficient to eliminate any obvious visual differences with the initial input fields. Methods of improving the basis which were discussed in previous sections may now be applied to expand it slightly if necessary to reduce any inaccuracies. Also, as was encountered in the previous section on Burgers' equation, different parameter values in neighborhood of the original one can also be simulated to produce satisfactory results. For large changes in parameter value it may be necessary to return to the use of discrete basis functions, the results of which may be used to augment or replace the basis generated here at $\varepsilon = 0.05$.

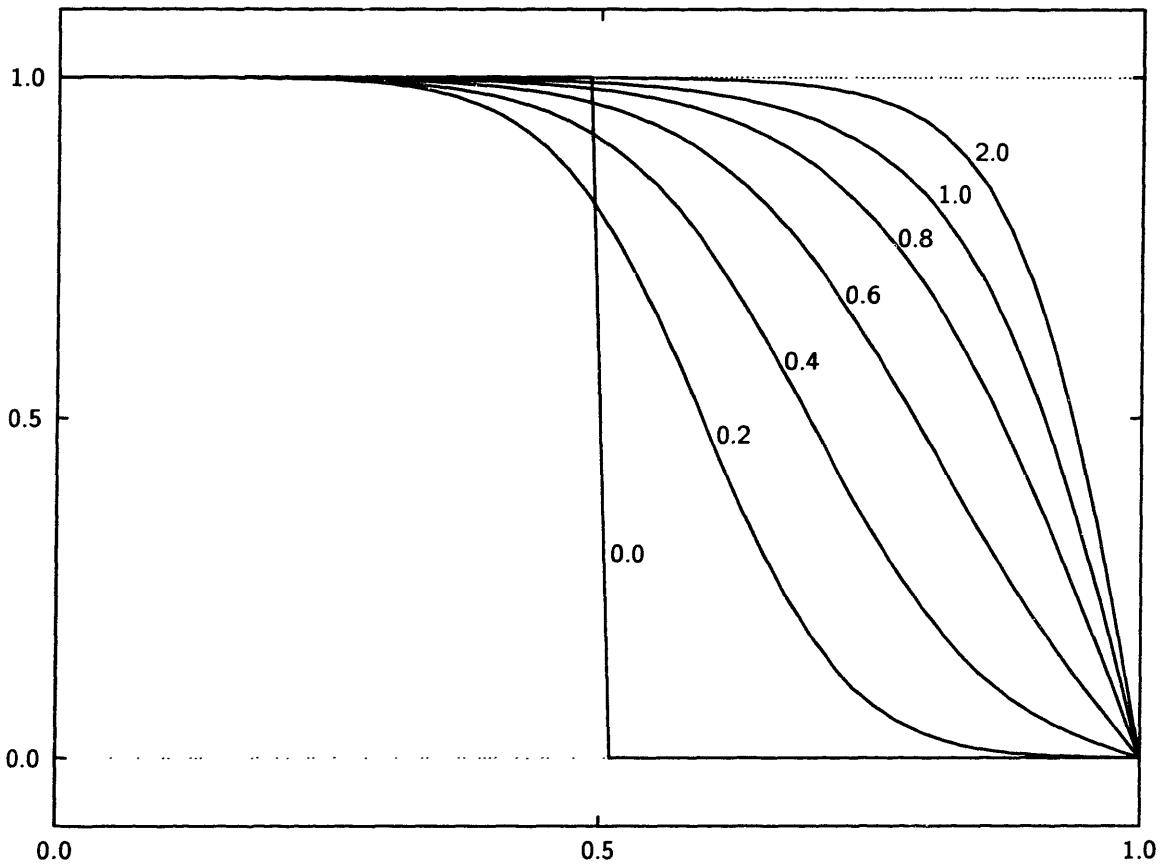


Figure 48. Downwind derivative solution of Burgers' equation in discrete basis space at seven points in time.

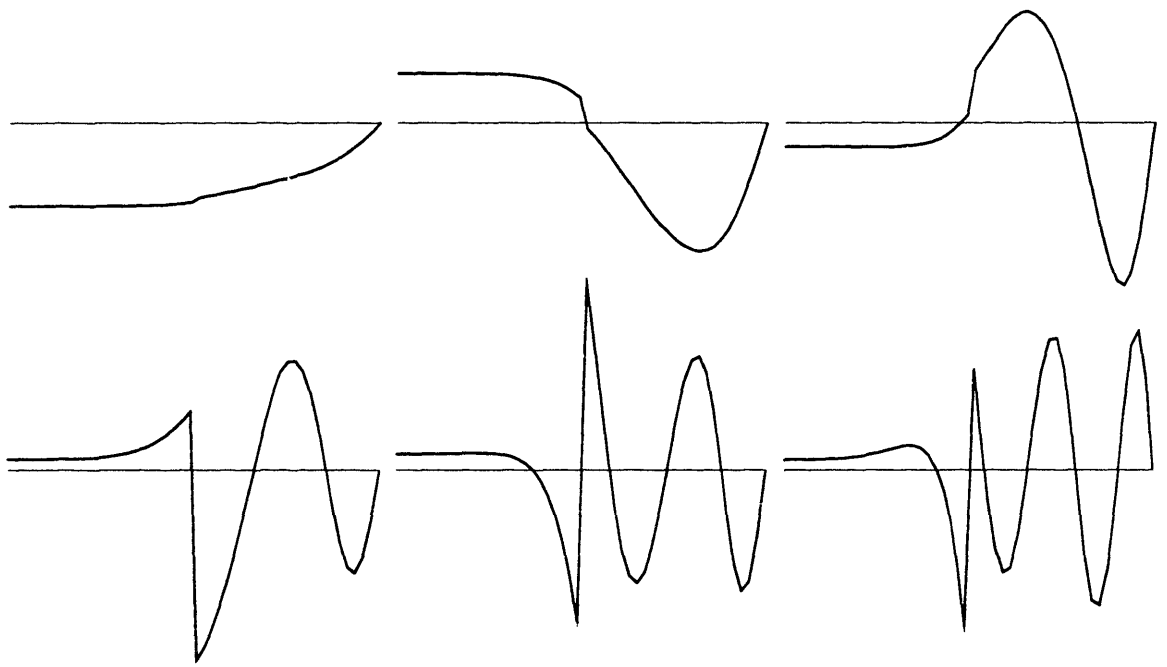


Figure 49. First six basis functions generated from discrete basis simulation, in order from left to right and top to bottom.

11.5 Iterative matrix solutions

This section presents some possible applications for empirical eigenfunction expansions in solving the standard problem

$$A\mathbf{x} = \mathbf{b}$$

using iterative methods, although the conclusion about the use of such methods may be surprising. Direct methods, such as Gaussian elimanition, are frequently used in practice with iterative methods which converge to the solution being the usual choice. Two common iterative methods are based on splitting A into parts, one of which is easily invertible. The Jacobi method writes

$$A = L + D + U,$$

decomposing A into a diagonal part, and lower and upper triangular matrices with zeros on the diagonal, then advancing the solution using

$$D\mathbf{x}^{m+1} = -(L+U)\mathbf{x}^m + \mathbf{b}.$$

The Gauss-Seidel method is a variation on this which does updating in place instead of determining an entire new $\mathbf{x}$ at each step. The matrix equation is

$$(L+D)\mathbf{x}^{m+1} = -U\mathbf{x}^m + \mathbf{b}.$$

Both these methods have been proven to converge to the correct answer, but the number of steps required is prohibitively large. One way to reduce the number of iterations is to predict future corrections by overcorrecting by some amount at each step. Writing the Gauss-Seidel method as

$$\mathbf{x}^{m+1} - \mathbf{x}^m = -(L+D)^{-1}(U\mathbf{x}^m + \mathbf{b}) - \mathbf{x}^m$$

or equivalently,

$$\mathbf{x}^{m+1} = \mathbf{x}^m - \omega(L+D)^{-1}((L+U+D)\mathbf{x}^m + \mathbf{b}),$$

where a scalar parameter ω has been introduced to represent the overcorrection. This method is called simultaneous overrelaxation. The search for an optimal ω per problem and per iteration number is a widely studied probelm, but it has been proven that the method is convergent only for $0 < \omega < 2$ [138].

These sorts of relaxation methods can be generalized to operate on fields which are expressed as basis function expansions, as will be demonstrated here using the Poisson equation as a model.* The Poisson equation

$$\nabla^2 u = f$$

or when $f = 0$, the Laplace eqation, arises in a wide variety of physical models such as compressible fluid flow and static electric charge calculations. Here it will be assumed that the equation must be solved many times but for possibly different source terms f, in analogy to the assumption of many invocations with similar initial conditions and parameters which has been made throughout this thesis. A five-point discretization of the Laplacian on a square grid gives the update formula for u:

$$u^{m+1}(x,y) = u^m(x,y) - \frac{\omega}{-4}R(x,y)$$

* Actually for equations such as this one which reduce to symmetric, tridiagonal matrix problems, there are optimal methods, Fourier analysis or cyclic reduction or a combination, FACR(l) [129], which have been shown to operate in $O(\log_2 \log_2 MN)$ time, instead of $O(MN)$ or worse for the relaxation methods, where the matrix is of size $M \times N$.

where the residual is calculated as

$$R(x,y) = u^m(x-1,y) + u^m(x+1,y) + u^m(x,y-1) + u^m(x,y+1) - 4u^m(x,y) - f(x,y).$$

The spacing between grid points is taken to be unity as it does not affect the result. This can be implemented using in-place updates by replacing values of u on the grid as soon as they are calculated. Expanding u in terms of unchanging basis elements starts with the definition

$$u(x,y) = \sum_i u_i \mathbf{U}_i(x,y)$$

and the discretization of the Laplacian

$$(\nabla^2 \mathbf{U}_i)(x,y) = \mathbf{U}_i(x-1,y) + \mathbf{U}_i(x+1,y) + \mathbf{U}_i(x,y-1) + \mathbf{U}_i(x,y+1) - 4\mathbf{U}_i(x,y)$$

so that the update formula for the coefficients is

$$u_i^{m+1} = u_i^m - \frac{\omega}{-4} R_i$$

with a residual term for each coefficient

$$R_i = \sum_j u_j \langle \nabla^2 \mathbf{U}_j, \mathbf{U}_i \rangle - \langle f, \mathbf{U}_i \rangle.$$

The projection terms in the summation are constants and are calculated once only, after the basis for u has been chosen. The term $\langle f, \mathbf{U}_i \rangle$ is calculated at the start of each simulation when f is initially provided. When using an eigenfunction expansion, there is no concept similar to that of in-place updates, and all the residuals must be calculated before changes to the coefficients are made. The algorithms implementing these equations are shown in Table 22.

Loop over iterations
  Loop over grid points
  calculate $R(x,y)$
  update $u(x,y)$
  Exit if $\sum_{x,y} \lvert R(x,y) \rvert$ within tolerance
	Loop over iterations
	  Loop over coefficients
	  calculate R_i
	  Exit if $\sum_i \lvert R_i \rvert$ within tolerance
	  Loop over coefficients
	  update u_i

Table 22. Grid-based (left) and field-based (right) simultaneous overrelaxation algorithms.

There are some very different problems in this situation related to choosing a basis for the dependent variable u. Given an idea about the class of source terms f for which the Poisson equation is to be solved, the basis for u must be representive of their solutions. It is not necessary, or even desirable, that the basis also be representative of the *path* (in the space of fields $u^m(x,y)$) which a grid-based solver would take, since there is no valid comparison of intermediate states as there has been for the forward-time integration case. As always, the best way to create a basis will be to solve the problem traditionally to generate a good set of possible solutions; for the case of the linear Laplacian, small changes in the input will effect only small changes in the output allowing the accurate solution of a source f which can be interpolated between others which were used to generate the basis.

Considering the problem from a different angle will show that it never makes good sense to implement this scheme. Given a set of $\{f_i\}$ and the corresponding set of $\{\phi_i\}$, each of which satisfies

$$\nabla^2 \phi_i = f_i,$$

what is the best solution to a new source field f? And the solution may only be a linear combination of the given $\{\phi_i\}$. Since the problem is linear, though, it suffices to express f as a linear combination of the given $\{f_i\}$, then use those coeffcients to generate the solution using the $\{\phi_i\}$. The projection of f onto the space spanned by $\{f_i\}$ is unique, however the choice of coefficients in the expansion

$$f = \sum_i \alpha_i f_i$$

is not since the $\{f_i\}$ were not required to be orthogonal, but the same solution will be obtained. This new algorithm for solving an arbitrary Poisson problem using a dictionary of previous solutions is not iterative, and produces exactly the same solution that the suggested iterative method outlined above does. Hence, for this particular operator, it is not worth generating any results.

For linear operators such as the Poisson equation, the solution method through eigenfunction expansions reduces to projecting the source term on a dictionary of previous solutions and superimposing them to obtain the result. There is no need for iterations in this method as there is when using a gridded domain. Applications for this sort of lookup may include pre-conditioning the initial choice of u for use in a traditional algorithm, performing the work of the bulk of the iterations in a much smaller space, and fine tuning that result using traditional grid-based iterative methods.

11.6 Conclusion

Five examples have been presented for five different reasons. First a small set of reactions to illustrate the method and some of the choices in basis selection and underlying integration schemes. Second a large system of chemistry to show the practical steps involved in generating a basis space integration method, including how to augment insufficient bases. Third a standard partial differential equation was solved for comparison with many previous solution techniques and to validate the extension to equations involving partial derivative terms. Fourth, the same partial differential equation was resolved using a different set of boundary conditions, and without the aid of previous solutions to guide the choice of basis. Finally an example of an inappropriate use of the raw application of basis space techniques was given. The computational advantages of basis space integration are clearly shown in the second two sections, and the procedures necessary to adapt a new system are described throughout.

12 Summary

This thesis has considered some common chemical engineering problems in a broad framework which allows for the importation of knowledge from other fields. The general theme has been to work to reduce the total time to solution, defined in terms of initial model development and analysis through actual simulation and verification of the results. This is a broader message than that usually considered, which is just to speed up a given computational piece. There are three large stages in a design which were considered in the three parts of this thesis: initial mathematical analysis and modeling, choice of a representation, and numerical implementation.

Often before embarking on a full simulation of a complex model it is worth investigating to see if it has any special mathematical properties, such as symmetries or conserved quantities. Part one of this thesis presented four different examples of this: a topological study where qualitative restrictions were developed directly from the model equations, a numerical search over parameter space to find an improved operating strategy of a catalytic process, the system identification of an unknown model using experimental data, and two novel ways of implementing standard models numerically on high-performance machines.

It is frequently seen that chemical engineering models are developed and used on a time scale which is much longer than the lifetime of any particular computational architecture. For this reason it is important to develop the representation of any particular model without consideration of the actual processor on which it might be executed. The choice of representation will hopefully be ideal for the particular problem at hand and include any knowledge from initial analyses and previous simulations or results. An approach to constructing the optimal basis in the spatial domain for a given simulation using such knowledge was presented in the chapters of part two. Criteria for determining the error inherent in a basis and a field to be expanded were presented as well.

Finally, the necessary implementation and results were presented. This included the automatic conversion of a natural language representation of the system into computer language codes, and a system for maintenance of large data sets produced in the process. A system for dynamically tracking the error during an integration has been generated also. The speed and space advantages of the method were made clear through numerical examples. Perhaps most importantly, during development of the entire system the need of making the results accessible to other users was understood.

13 Appendices

13.1 All the tools

The following 33 subsections verbosely explain the plethora of utilities developed in the course of this project, and their use. Inner workings of the tools is deferred to the next section, keeping the approach in this one more like that of a user's guide instead of a programmer's reference. The first four topics are crucial for an understanding of the structure of the package, while the remaining tools are certainly useful, but not directly connected to the method of basis space integration.

13.1.1 Preparing for a basis space integration

An executable integrator in empirical basis space is conceptually divided into many individual components. The decision to produce something in this modular form instead of one single unit was motivated by the widely varying projected uses of such an integration package. First is the relatively simple case where a user wants only to simulate some zero-dimensional set of reactions, in which there is no added complexity of having to deal with spatial interactions. At the other extreme is the person who would like to incorporate this method of solving differential equations into some extensive, venerated model. Then, the whole process must by necessity be completely visible so that it can be meshed with the existing simulator. The partitioned scheme is designed to make that joining as easy as possible by keeping conceptually separate components physically separated.

13.1.2 Mechanism parser: `parse`

Resting at the highest level of this process and collection of pieces is the mechanism file. This lists participating species, specifies rate (and other) constants, and reactions, in that order. The example mechanism file below should help elucidate the forthcoming discussion of its complete structure.

```
#
# Test input file to mechanism analyzer
#
Option
    Language: C

Species
```

```
    NO   # Nitric Oxide
    NO2  # NO2
    O    # O-dot
    O3   # ozone

Constants
    O2  = 2.1e5   # oxygen
    M   = 1.0e6   # inert third body
    hv  = 1       # ignore in rate expressions
    k0 = 0.555    # 1/min
    k1 = 2.183e-5 # all in ppm-min units at 298 K
    k2 = 26.59

Reactions
   NO2 + hv     --> NO + O   # a reaction
   O + O2 + M  ---> O3 + M
      rate = k O^2 M  # this rate is for the reaction on the line above
   NO + O3      --> NO2 + O2
```

This is a simple ozone chemistry mechanism, with the species `O2` and `M` being treated as constant. The complete structure of a mechanism file is given below. Comments may appear anywhere in the file, delimited by a hash (#) and extending to the end of the line. Keywords may be in either case but user-defined names are case-sensitive.

mechanism-file:
    option-section$_{opt}$ *species-section constant-section*$_{opt}$
        reaction-section derivative-section$_{opt}$

The subscript *opt* indicates that an item is optional so that each possible construction need not be listed. Only one option is provided now, but other useful switches and commands would later be added here.

option-section:
    Options *option-list*
option-list:
    option
    option option-list
option:
    Language *language*
language:
    C
    FORTRAN

Species are listed one on each line, hopefully with some appropriate comment with each. Optionally the species may be numbered with non-negative integers to indicate their positions in the concentration array. These numbers need not be contiguous or even fill the whole array, but all the species must be numbered if at least one is. This facility is provided to allow basis space integration for only some of the species in the reactions, but allow them to exist in the same data structure as the rest of the species. The template integrator package respects the user-specified numbering scheme, and access methods for the concentration arrays are indexed as specified. The default numbering, when no explicit ordering is given, will start at zero and continue down the list of species.

species-section:
    Species *species-list*

species-list:
    species
    species species-list
species:
    species-name array-position
    species-name

Similar to the list of species is the optional list of constants, in which identifiers can explicitly be given a value to be used in substitution when generating output files, or passed through in symbolic form, to be defined elsewhere in the integrator package. There is a set of special constants of the form **k**⟨*number*⟩ where ⟨*number*⟩ is between zero and the number of reactions in the mechanism. This constant then provides the reaction rate coefficient for that reaction.

constant-section:
    Constants *constant-list*
constant-list:
    constant
    constant constant-list
constant:
    constant-name = *value*
    constant-name

Arbitrary reactions are allowed and stoichiometry is not verified to allow species lumping and the omission of near-constant reactants. If no reaction rate is specified by placing a **rate** expression immediately following the reaction, the default mass-action rate will be constructed, with **k**⟨*number*⟩ used as the coefficient. If a rate is explicitly specified and contains the constant **k**, it will be changed to **k**⟨*number*⟩ for that reaction's number. If the user-given rate contains **k**⟨*number*⟩ the ⟨*number*⟩ must match the reaction's number. No user-defined numbering of reactions is permitted, as the output is based on species, and the order in which participating reactions are performed does not affect the results. This automatic promotion of the term **k** is intended to provide the convenience of being able to change the order of the reactions without having to rewrite the rates. Note, however, that the values of **k**⟨*number*⟩, if specified in the constants section, correspond to the current ordering of reactions. In practice it is convenient to write each reaction followed by its reaction rate coefficient, but allow for automatic generation of the full rate expression, such as:

```
O + NO2 --> NO
k = 4.083e+06/T

NO + O3 --> NO2
k = 7.93e+05/T * exp(-1370/T)
⋮
```

The parser uses the given expression in generating the reaction rate, and there is no need to specify a reaction number when assigning the **k** value using this convention.

reaction-section:
    Reactions *reaction-list*
reaction-list:
    reaction
    reaction reaction-list
reaction:
    reactants --> *products* $rate_{opt}$ $k\text{-}expression_{opt}$
    reactants --> $rate_{opt}$ $k\text{-}expression_{opt}$
        --> *products* $rate_{opt}$ $k\text{-}expression_{opt}$

Reactants and products are really the same thing, just on different sides of the arrow. The three possible forms for a rectant are a single species name, or a numeric or constant stoichometric coefficient before a species name.

reactants:
    reactant-list
products:
    reactant-list
reactant-list:
    reactant
    reactant + *reactant-list*
reactant:
    species-name
    constant-name species-name
    float species-name
rate:
    rate = *expression*
k-expression:
    k = *expression*

Rate expressions can be arbitrary mathematical expressions using standard infix notation, the full structure of which is shown below. Only one built-in function, **exp**, is understood, but the extension to incorporate other built-in functions is straightforward. Implicit multiplication is not provided; explicit * signs must be included. The basic numerical element in expressions is the floating point constant, an optionally-signed string of digits with perhaps one decimal point and an optional exponent using **e** notation as in the examples above.

expression:
    term
    expression + *term*
    expression - *term*
term:
    power
    term * *power*
    term / *power*
power:
    factor
    non-unitary-factor ^ *power*
non-unitary-factor:
    parenthesized-expression
    exp-function
    species-name
    constant-name
    float
factor:
    non-unitary-factor
    + *factor*
    - *power*
parenthesized-expression:
    (*expression*)
exp-function:
    exp *parenthesized-expression*

The optional derivatives section is used for specifying spatial derivative fields which although not part of the *chemical* mechanism, may be necessary for other parts of the algorithm and are generated

and maintained with the other fields. The *xy* notation below is to allow strings such as "A x" to specify $\frac{\partial A}{\partial x}$, or "B xxyyy" for $\frac{\partial A^5}{\partial x^2 \partial y^3}$. The spatial dimension is limited to two, but extensions of this would again be easy to implement.

derivative-section:
    Derivatives *derivative-list*
derivative-list:
    derivative
    derivative derivative-list
derivative:
    species-name xy
xy:
    X
    Y
    X *xy*
    Y *xy*

The output `derivs.c` file from the mechanism parser can be constructed in one of two major ways. First, in the species-based method the time derivatives of the coefficients of expansion for each species are generated as a function of the coefficients of all the reactions in which it participates, one at a time. This is the obvious way to think about the problem, and is how examples have been presented all along in this thesis. However certain values which might be reusable must be recalculated. The mechanism

$$A + B \xrightarrow{1} C + D$$

will generate, for all four species, evolution equations which contain the term

$$k_1 a_j b_k$$

in the right hand side, inside a loop over the coefficients of A and B. The default output will look like the following:

```
/* species A */
for (i=0; i<N_A; i++) {
    adot[i] = 0.;
    for (j=0; j<N_A; j++)
        for (k=0; k<N_B; k++)
            adot[i] -= k1 * a[j] * b[k] * PROJ(P_A_A_B,i,j,k);
}
/* species B */
for (i=0; i<N_B; i++) {
    bdot[i] = 0.;
    for (j=0; j<N_A; j++)
        for (k=0; k<N_B; k++)
            bdot[i] -= k1 * a[j] * b[k] * PROJ(P_B_A_B,i,j,k);
}
/* species C */
for (i=0; i<N_C; i++) {
    cdot[i] = 0.;
    for (j=0; j<N_A; j++)
        for (k=0; k<N_B; k++)
            cdot[i] += k1 * a[j] * b[k] * PROJ(P_C_A_B,i,j,k);
}
/* species D */
```

```
for (i=0; i<N_D; i++) {
    ddot[i] = 0.;
    for (j=0; j<N_A; j++)
        for (k=0; k<N_B; k++)
            ddot[i] += k1 * a[j] * b[k] * PROJ(P_D_A_B,i,j,k);
}
```

where the value `k1 * a[j] * b[k]` is being calculated `N_A + N_B + N_C + N_D` times.

The alternate representation takes into consideration this replication of the same value and is reaction-based instead of species-based. So for each reaction in the mechanism, a block to calculate its effect on the time derivatives of the participating coefficients is generated. The output of a mechanism containing the example reaction would look like:

```
/* zero all time derivatives */
for (i=0; i<N_A; i++)  adot[i] = 0.;
for (i=0; i<N_B; i++)  bdot[i] = 0.;
for (i=0; i<N_C; i++)  cdot[i] = 0.;
for (i=0; i<N_D; i++)  ddot[i] = 0.;
⋮
/* reaction 1 */
for (j=0; j<N_A; j++)
    for (k=0; k<N_B; k++) {
        t = k1 * a[j] * b[k];
        for (i=0; i<N_A; i++)
            adot[i] -= t * PROJ(P_A_A_B,i,j,k);
        for (i=0; i<N_B; i++)
            bdot[i] -= t * PROJ(P_B_A_B,i,j,k);
        for (i=0; i<N_C; i++)
            cdot[i] += t * PROJ(P_C_A_B,i,j,k);
        for (i=0; i<N_D; i++)
            ddot[i] += t * PROJ(P_D_A_B,i,j,k);
    }
/* reaction 2 */
⋮
```

Now a temporary variable is used to store the reused constant value during each of the `j` and `k` loops. The effect is that this alternate way of writing the equations is numerically faster due to the common expression reuse, but not by much because the bulk of the time seems to come from memory accesses and array offset calculations. The numerical results of the two methods are identical, within machine roundoff tolerance.

For future reference, a graphical description of the mechanism parser is shown in Figure 50. The mechanism name is just a sample for the ozone chemistry given above. One output is a shell script—both a file and an executable—and is used to calculate the projection coefficients necessary to simulate the reactions. The other two output files are computer source code defining the species and implementing the reactions.

13.1.3 Fields control file: `fields`

A mechanism specification is a generic description of a set of chemical reactions; there is no inherent link to basis space integration, except perhaps for the listing of derivative species. Information about where to find the data to generate a basis for each species, the number of fields to use during the integration, and initial conditions is all specified in a file which will be used to control generation of fields and execution of the integration itself. The structure of a fields control file is much like that

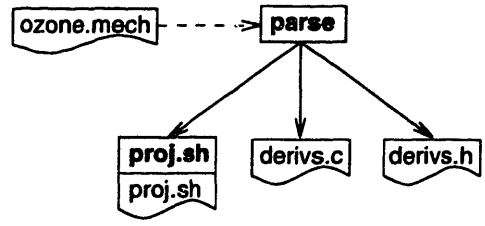


Figure 50. Input and outputs of **parse**.

of the mechanism file, with separate sections marked by header lines and the same conventions on case sensitivity and comments.

An example fields control file is presented to aid the more detailed description below.

```
#
# Fields control file, three-reaction ozone test case
#
Mechanism ozone.mech  # specify the mechanism file

Variables
    ozone = /net/immoral/gig3/pw/thesis/Code/parser/ozone
    xsize = 10      # these two are required
    ysize = $xsize  #
    NUM = 12

Fields NO
    $ozone/NO/NO.all
    $ozone/NO/NO.all skip=5 num=1 ic

Fields NO2
    /dev/null cat="discrete $xsize $ysize"
    $ozone/NO2/NO2.ic ic

Fields O
    $ozone/O/O.all.Z
        num = $NUM skip = 5
        ascii  # binary double is the default data format
        cat = "zcat | awk '{print $2}'"  # uncompress this file
    # no IC specified --> zero IC

Fields O3
    /dev/null cat="discrete $xsize $ysize"

Numfields
    NO  7
    NO2 3
    O3  3
    O   5
```

This example first specifies which mechanism it supports, and the named mechanism file will be used to extract the species declaration section. Next some variables are defined just for convenience, except for **xsize** and **ysize** which are necessary for generating the bases. (Note that **ysize=1** would specify a one-dimensional model.) The next four sections each describe the fields which will be used to generate a basis for the named species, with each entry consisting of a filename and optional

key/value pairs. For instance, all the example fields for NO are in a single binary file of blocks of `xsize * ysize` doubles, with an unspecified number of fields which will be discovered during basis generation. The initial condition is taken to be the sixth field in that file. Twelve fields for O are kept as the second column of a compressed file, with the first five field entries in that file ignored before the twelve are read off. The final section in a fields control file is the number of basis fields to generate for each species. Every species must be mentioned and the specified number must be no bigger than the number of input fields provided.

The full description of a fields control file follows.

fields-control-file:
    mechanism-section option-section$_{opt}$ *variable-section*$_{opt}$
        field-section-list numfield-section
mechanism-section:
    Mechanism *mechanism-name*

The *mechanism-name* could be a full pathname to the mechanism file.

option-section:
    Options *option-list*
option-list:
    option
    option option-list
option:
    Language *language*
language:
    C
    FORTRAN
variable-section:
    Variables *variable-list*
variable-list:
    variable-name = *substitution*

Variables can be used as in shell programming to hold more complex strings which are specified once at the start of the file. The *variable name* is a proper identifier string, but the substitution can contain other special characters and may include previously defined variable names, of the form **$**(*name*), which will be replaced with their substitutions as the new variable is defined. One special variable **$$** is predefined to expand into **$**, permitting dollar signs to make their way into the output, which is useful especially for lines in the *Special* section.

field-section-list:
    field-section
    field-section field-section-list
field-section:
    Fields *field-name field-specification-list*
field-specification-list:
    field-specification
    field-specification field-specification-list
field-specification:
    filename key-list$_{opt}$
key-list:
    key
    key key-list

key:
    num = *variable-or-integer*
    skip = *variable-or-integer*
    type-specification
    cat-specification
variable-or-integer:
    $*variable-name*
    integer
type-specification:
    ascii
    binary
    binary double
    binary float
cat-specification:
    cat = *string*
    cat = "*string*"

The default type is binary double and a specification of **binary** is equivalent to the default. The first form of *cat-specification* will be subject to variable substitution and can contain no embedded spaces while the quoted form will not be checked for variable substitutions and may contain anything except a newline. The next section simply declares the length of the field expansion to use for each species, and the final, optional, *Special* section is intended for use by those who know exactly what is happening inside the outputs. The special lines will be copied with variable substitution into a section of the **basis.sh** output file and interpreted by the shell.

numfield-section:
    Numfields *numfield-list*
numfield-list:
    species-name integer
    species-name integer numfield-list
special-section:
    Special *special-line-list*
special-line-list
    special-line
    special-line special-line-list

Again for reference, a graphical description of the fields control parser is shown in Figure 51. It calls on **parse** with a special flag to have it generate the list of species instead of reparsing the mechanism file directly. The two output shell scripts are used to generate the basis fields, and to provide input to the integrator executable.

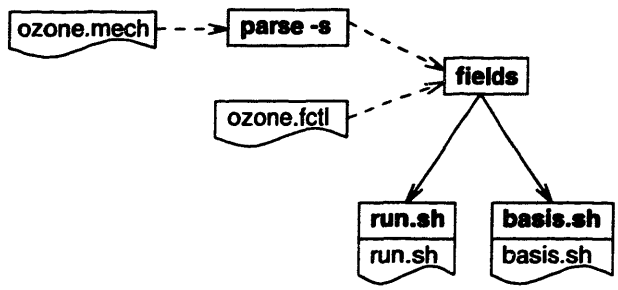


Figure 51. Inputs and outputs of `fields`.

13.1.4 Integrator executable

The main integration program works as a stand-alone integrator, or can be used as an example of how to implement the pieces in a different integrator. It requires as input a list of the expansion length used for each species, initial conditions, and a matrix of projection coefficients needed for the integration. It has no knowledge whatsoever as to the spatial structure of the bases chosen for each species, and is designed to be very fast at what it does. Needless to say, there is a large amount of external support required to facilitate this. Figure 52 shows the relationships among the pieces of the integrator. The solid horizontal arrows denote that the item on the left calls the item on the right. The program starts with general initialization: reading the required input, which is usually provided via the `run.sh` script generated by the fields control file parser, then loops over time, integrating and printing out results. The time integrator used in this example application is LSODE [63], and it is decoupled from the main routines by using a set of interface routines to do initialization and time stepping. Replacing the integrator should not require changes to the main program, just to this interface layer. The `derivs()` routine generated by the mechanism parser is called as needed by LSODE to generate temporal derivatives. Output coefficients can be transformed back to physical space using the **expand** utility described below, or further processed for other purposes.

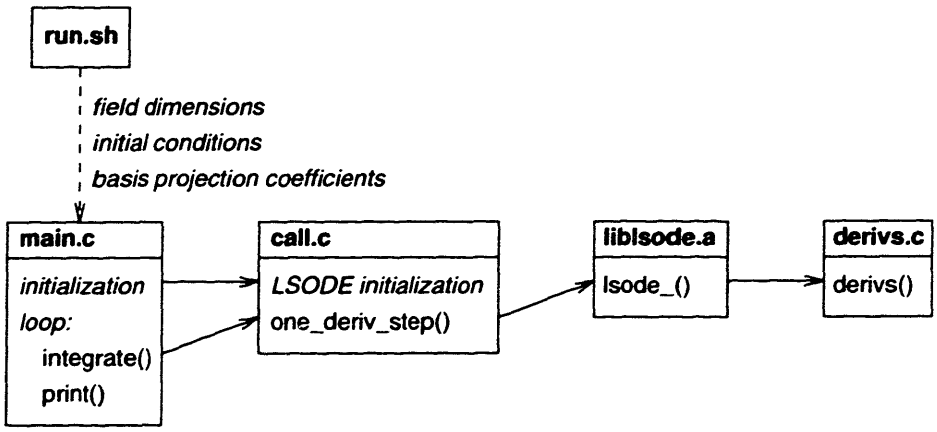


Figure 52. Example integrator pieces.

The main program is kept up to date using a `Makefile` which checks the modification times of all the involved files to see if the executable must be rebuilt. Other dependencies among the input files are catalogued in this way as well, such that changes in the fields control file or mechanism file will trigger invocation of the appropriate parser.

The remaining subsections are alphabetically ordered, and document the lower-level utilities. Most of the utilities operate on the standard basis file format, in binary, but some may still support `ASCII` mode through a user-selectable switch. The following tools are each designed to implement only a small piece of functionality, but when combined together they can provide every desired maniupulation of data files. The descriptions are all complete in the sense that every available feature is described; there should be no surprises. Also the design of each code is similar so that all warning and error messages are printed using a consistent interface, and any messages emitted are always identified with the program's name to distinguish it from the other codes which may be working together in a long pipeline.

A certain simple file format convention exists for storing basis elements in binary. The elements are in order from 1 to N in the file, each element stored with the first spatial dimension varying fastest, so that $\phi_i(x, y)$ is stored starting in the order:

$$\phi_i(0,0), \phi_i(1,0), \ldots, \phi_i(\texttt{xsize}-1,0), \phi_i(0,1), \phi_i(1,1), \ldots$$

Code which is structured in the same way can thus read a whole basis element (or list of elements) using a single **read** system call. A file which contains basis elements will have a header of four 32-bit integers which specify, in order, the number of fields in the file, the number of input example fields used to generate this basis, and two spatial dimensions: **xsize** and **ysize**. This header structure is minimal and not very portable or intuitive, but it is fast and easy to implement and if every utility which deals with bases checks the header bytes against what it expects, many bugs will be avoided.

13.1.5 appendbasis

Two or more basis files are appended using this simple program. The headers of each file are checked to verify that the spatial dimensions all agree, and the output number of fields is set to the sum of the lengths of the inputs, then the contents of the files (without their headers) are copied to the output preceded by the new header. The example,

```
unix% appendbasis a1.basis a2.basis a3.basis > all.basis
```

concatenates the three inputs together, with header checking.

13.1.6 basis_mul

The fields of two or more basis files are multiplied together under all the permutations and written to output. This is useful for verifying that calculations involving product fields are well represented by a given basis. As an example,

```
unix% basis_mul -d a.basis b.basis c.basis
```

where each of the three bases is big enough for this example, would produce on standard output in this order,

$$\mathbf{A}_0 * \mathbf{B}_0 * \mathbf{C}_0,\ \mathbf{A}_0 * \mathbf{B}_0 * \mathbf{C}_1,\ \mathbf{A}_0 * \mathbf{B}_1 * \mathbf{C}_0,\ \mathbf{A}_1 * \mathbf{B}_0 * \mathbf{C}_0,\ \mathbf{A}_0 * \mathbf{B}_1 * \mathbf{C}_1,$$
$$\mathbf{A}_1 * \mathbf{B}_0 * \mathbf{C}_1,\ \mathbf{A}_1 * \mathbf{B}_1 * \mathbf{C}_0,\ \mathbf{A}_1 * \mathbf{B}_1 * \mathbf{C}_1,\ \mathbf{A}_0 * \mathbf{B}_0 * \mathbf{C}_2,\ \mathbf{A}_0 * \mathbf{B}_2 * \mathbf{C}_0,\ \textit{etc.}$$

The option **-d** indicates "diagonal" order as illustrated above which causes it to produce the fields with the lowest total field number first, which is usually what is desired when lower numbered fields correspond to larger eigenvalues in an expansion. Non-diagonal order is the sequence

$$\mathbf{A}_0 * \mathbf{B}_0 * \mathbf{C}_0,\ \mathbf{A}_0 * \mathbf{B}_0 * \mathbf{C}_1,\ \ldots,\ \mathbf{A}_0 * \mathbf{B}_0 * \mathbf{C}_{N_C},\ \mathbf{A}_0 * \mathbf{B}_1 * \mathbf{C}_0,\ \ldots,\ \mathbf{A}_0 * \mathbf{B}_1 * \mathbf{C}_{N_C},\ \ldots.$$

The option **-v** makes the program more talkative which is reassuring when working on large domains. The final option **-m** instructs the code to cache the fields in memory instead of seeking them off the disk every time, allowing the code to run much quicker when there is enough memory to hold all the fields at once. The algorithm without **-m** is clever enough not to reload fields it is currently using, as it is based on a recursion over total field number.

13.1.7 basis_proj

To generate the projection fields specified by the mechanism parser, the constructed script **proj.sh** uses this program. It is similar to **basis_mul** in that all permutations are computed, although only a scalar projection coefficient is output for each combination. The ordering is non-diagonal. The command

```
unix% basis_proj a.basis b.basis c.basis
```

produces

$$\langle \mathbf{A}_0 * \mathbf{B}_0, \mathbf{C}_0 \rangle,\ \langle \mathbf{A}_0 * \mathbf{B}_0, \mathbf{C}_1 \rangle,\ \ldots,\ \langle \mathbf{A}_0 * \mathbf{B}_0, \mathbf{C}_{N_C} \rangle,\ \langle \mathbf{A}_0 * \mathbf{B}_1, \mathbf{C}_0 \rangle,\ \ldots,\ \langle \mathbf{A}_0 * \mathbf{B}_1, \mathbf{C}_{N_C} \rangle,\ \ldots$$

in binary to standard output. There are no other options, and projections of any length can be generated as in **basis_mul**.

13.1.8 compare

Two blocks of data are compared by calculating the L^2 norm and maximum deviation, both raw and weighted by the average value. No field structure is assumed so the grid dimensions must be specified on the command line,

```
unix% compare -x 30 -y 18 calc.field obs.field
```

with the two specified files each having $30 \cdot 18 = 540$ binary floating point values. The option `-a` causes the code to print the location of the maximum deviation as well as its value.

13.1.9 constant

This utility simply generates the field of specified size with the given value at each point, and is useful for generating extra basis elements.

```
unix% constant 30 18 1.3
```

generates a 30×18 grid with 1.3 at each point. The second dimension defaults to 1 if omitted.

13.1.10 deriv

Spatial derivative fields of a basis file are generated using this utility. One of three supported schemes: explicit (the default), implicit, or Crank, can be selected using the switches `-e`, `-i`, or `-c`, respectively. The derivative term to generate is specified using `-x` and `-y` so that

```
unix% deriv -x 1 -y 2 -i a.basis
```

generates the implicit discrete fields $\left\{ \frac{\partial^3 A}{\partial x \partial y^2} \right\}$, for example.

13.1.11 discrete

This code produces all the Kronecker delta fields of specified size, and is also useful for enlarging a basis, as the entire output from **discrete** could be taken as a basis which spans the space completely. It requires one or two arguments specifying the grid size, but no value to put in the fields as this is fixed at 1.0.

```
unix% discrete 30 18
```

generates 540 fields of dimension 30×18 in order $\delta(0,0), \delta(1,0), \ldots \delta(29,0), \delta(0,1), \ldots$ where $\delta(x,y)$ is the field consisting of all zeros except for the point (x,y) which has value 1.0.

13.1.12 dropbytes

The input is copied to the output, except for the first $\langle num \rangle$ bytes. The value of $\langle num \rangle$ is specified through the `-n` option, and defaults to zero in which case this is exactly UNIX `cat`. Alternatively, the `-d` option may be used instead to specify a number of binary floating point values (doubles) to drop. The example

```
unix% dropbytes -n 16 a.basis | dropbytes -d 540
```

strips the header off the basis for **A** and removes the first field (if the grid size is 30×18). An input file may be specified on the command line as above, otherwise standard input is used.

13.1.13 expand

Sets of coefficients are read from standard input and used to generate fields using the specified basis. one field for each set of coefficients; this is the primary implementation of superposition. The line

```
unix% expand -n 5 a.basis < integration_results > full_fields
```

where `integration_results` is a binary file of $5 \cdot n$ floating point values, produces $5 \cdot n$ output fields, each of the size specified by the header on the basis file. If the `-n` option is omitted, **expand** uses all the elements available in the basis and expects the appropriate size of coefficients sets. Since the number of sets is unspecified, the only way **expand** can detect an error is if the size of the input is not a multiple of the number of fields it has been told to use, 5 here. Erroneous results will be generated in the unlucky case of coefficient set size mismatch by a multiple of 5 (in this case).

13.1.14 fcat

This implements **cat** with many other useful options. Without any options specified, **fcat** copies standard input (or the specified file) to standard output. As usual, **-x** and **-y** can be used to specify the dimensions of the set of input fields. The option **-n** instructs it to output only the first ⟨*num*⟩ fields, while **-s** can be used to skip over the first ⟨*skip*⟩ fields. Together these options can be used to specify a range. The options **-m** and **-d** can be used to restrict output to those fields whose sequence numbers when divided by ⟨*divisor*⟩ have the specified remainder (⟨*modulo*⟩). The most useful, perhaps, option is type conversion: **-t double** is the default and specifies that the input consists of binary double floating point values, **-t float** indicates half-size floating point values, and **-t ascii** denotes values in **ASCII** format. Output always consists of binary doubles. The words given to the **-t** option can be abbreviated to a single character to reduce readability but speed typing. The complex command

```
unix% fcat -x 30 -y 18 -s 2 -n 5 -m 2 -d 3 -t ascii
```

copies the five fields numbered 8, 11, 14, 17, and 20 (starting from zero) to standard output, converting them from **ASCII** to binary in the process. The fields have dimension 30×18. The "skip" and "num" options are always applied after the modulo filtering.

13.1.15 fieldascii

This handy little script takes as argument a basis file name,

```
unix% fieldascii a.basis
```

and prints the field header to standard error, while converting the contents to **ASCII** on standard output. Except for the one informational line, it is completely equivalent to a two-step operation using **dropbytes** and **toascii**.

13.1.16 headbasis

To extract some number of basis elements from a basis file, this code takes the name of the file and the requested number,

```
unix% headbasis a.basis 3
```

and calculates the appropriate number of bytes to output, creates a new header, and calls on **dropbytes** and **headbytes** to do the work.

13.1.17 headbytes

The standard utility **head** works on text files only. This code does the same task for arbitrary files and is the companion to **dropbytes**. One common use is

```
unix% headbytes -n 16 a.basis | od -I
```

which shows the contents of the header of a basis file, while

```
unix% dropbytes -n 16 a.basis | headbytes -d 540 | toascii
```

converts to **ASCII** the first basis element in the file.

13.1.18 log10

This trivial utility calculates the base ten logarithms of a binary stream of double floating point values, printing the output in **ASCII**. It is useful for plotting error values. The only option **-c** causes replacement of the result of logarithms of non-positive numbers with -99.

13.1.19 makebasis

This shell script writes four binary integers (using **writeints**), then copies its input to the output, creating a basis file from a set of fields. The actual implementation of this code, however, requires

121 lines to perform robust error checking of the input argument flags since a general invocation may be rather complex:

```
makebasis -xsize 30 -ysize 18 -tsize 10 a.fields > a.basis
```

Perhaps the code is most useful as an example of robust shell programming.

13.1.20 makeorthtog

When augmenting a basis with other fields, it is essential to ensure that the new basis still satisfies the orthonormality conditions. The command

```
unix% makeorthog a.basis < a.extra > a.orthog
```

subtracts from the input file all components which fall in the span of the named basis. The output is not normalized or converted into a basis. One option `-n` ⟨*num*⟩, specifies that a smaller number of basis elements should be considered, instead of all those in the file as is the default.

13.1.21 normalize

Output from `makeorthog`, for example, may need to be normalized before appending it to another basis. The command

```
unix% normalize -x 30 -y 18 a.orthog > a.orthog.normal
```

scales the input field(s) by the appropriate amount such that they are each of unit norm.

13.1.22 orthbasis

In attempting to design perfectly complete basis sets, codes which implement Karhunen-Loève expansions through singular value decomposition are pushed to their limits, namely those of numerical precision. Output from such dangerous attempts may not be a truly orthogonal set of basis elements. The command

```
unix% orthbasis -s 3 a.basis > a_orth.basis
```

orthogonalizes the input basis using one of three different methods as specified by the "style" option. This utility is not expected to see much use, hence the styles are only numbered, and explained through comments in the source code. All three methods are inherently iterative, adjusting the basis element in question such that it is orthogonal to the previously considered ones, with various normalization steps as well. The three different options are most easily explained by the source code itself.

Style one:

```
normalize(0);
for (i=1; i<numfields; i++) {
    for (j=0; j<i; j++) {
        orthogonalize(i, j);
        normalize(i);
    }
}
```

Style two:

```
for (i=0; i<numfields; i++) {
    for (j=0; j<i; j++)
        orthogonalize(i, j);
    normalize(i);
}
```

Style three:

```
for (i=1; i<numfields; i++)
    for (j=0; j<i; j++)
        orthogonalize(i, j);
for (i=0; i<numfields; i++)
    normalize(i);
```

Decreasing amounts of normalization are performed by the three styles, attempting to reduce error caused by non-$O(1)$ calculations.

13.1.23 power

This code is a rational implementation of a basis generator. Given a set of input fields, the largest eigenvectors are extracted one at a time, between which the non-orthogonal components of the input are subtracted off, then the input is renormalized. The options are full words instead of single characters, but only the minimal disambiguous string need be supplied, and are described in Table 23. The code is a robust and clean implementation, based on the code **dsyevx**, part of the LAPACK group of **FORTRAN** numerical subroutines, which does the single eigenvalue extraction. Input and output is all in binary, but the conversion codes **fcat** and **toascii** may be used in pipelines to support the use of ASCII.

xsize ⟨*num*⟩	x-dimension extent
ysize ⟨*num*⟩	y-dimension extent
tsize ⟨*num*⟩	number of input fields (default, read until EOF)
numfields ⟨*num*⟩	number of output basis elements (default **tsize**)
mean	normalize input to zero time-average, and output that average
normalize	scale each input field to one
dumpcov	output covariance (or correlation) matrix
full	output eigenvalues and time eigenfunctions

Table 23. Option keywords available in the **power** utility.

13.1.24 proj_goodness

Interactions between the bases of multiple species are quantified by this shell script, which is to be invoked with no arguments. It reads the file **proj.sh**, which was constructed using the mechanism parser, and for each line which generates a set of basis projections, such as

```
basis_proj basis/NO.field basis/NO.field basis/O3.field >> basis/ozone.proj
```

all but the first of the bases given on the line are iteratively pointwise multiplied using **basis_mul** with the result projected onto the first basis. The logarithms (base ten) of these projections are reported to intuitively named files, such as **good.NO_O3_on_NO**. A plot of the values in these files may show peaks indicating poor projections at that point, indicating which bases should be augmented, and with which projections.

13.1.25 project

This implements the mathematically inverse function of **expand**, reducing a set of fields into a set of coefficients for the given basis. The option **-n** restricts the projection to the first ⟨*num*⟩ fields in the basis. Input field dimension is specified by the header on the basis file, and the code checks to make sure the size of the input is a multiple of the field size. Option **-s** keeps it from printing the amount of information not captured in the projection, while **-g** (for "goodness") selects only this information, and no coefficients are output. The projection method is by default Cartesian inner product, but a radial weighting suitable for cylindrical coordinates with a radial domain of $[0, 1]$ can be selected using the **-r** flag. Again, input fields and output coefficients are binary. The series

```
unix% project -n 5 a.basis < field | expand -n 5 a.basis
```

can be used to restrict a field to the space spanned by the first five (for example) elements of a basis. One variation on the standard algorithm is given through the `-a` (for "alternate") flag. In that case the projection coefficients are generated without intermediate subtraction of the projected component from the input fields. Without this flag, the algorithm reduces the size of the input fields as it generates each coefficient. These two methods produce different results only due to numerical round-off tolerances, or if the basis is not strictly orthogonal.

13.1.26 `select`

This utility picks out ranges of grid cells from from a file of fields. The grid size must be specified using the `-x` and `-y` options, although ⟨*ysize*⟩ defaults to the same values as ⟨*xsize*⟩ if omitted. Range specification is extremely versatile. The basic form is ⟨*xspec*⟩`x`⟨*yspec*⟩, range specifications for each of the two dimensions separated by `x` where each specification is a list of subranges, optionally separated by commas. A subrange can be a single number to extract just that row or column, or a range ⟨*lo*⟩`-`⟨*hi*⟩ to extract a set of rows or columns. Leaving either ⟨*lo*⟩ or ⟨*hi*⟩ off a range specification anchors the range to the low or high boundary, respectively. A single `-` specifies the entire set. The example

```
unix% select -x 30 -y 40 3, 5-12 x 17-34,37 38 a.field
```

creates a field of size 9×20 consisting of the specified ranges from the (optionally provided) input file, and illustrates the freedom given in the ranges: no quotes necessary, optional commas and spaces.

13.1.27 `snap`

This inappropriately-named code generates an orthonormal basis from a set of input fields using singular value decomposition. It is larger and more complicated than most of the other utilities, and has many options. The standard mode of operation reads the example fields from standard input and writes the basis, with its 4-integer header, on standard output, all in binary. Field dimensions must be specified using `-x` and `-y`, but the extent of the input will be determined automatically if not fixed using `-t`. The option `-n` limits the output to a smaller number of basis fields from its default, the entire set. There are further options for debugging which will not be discussed here. The implementation of SVD follows the prescription in Golub [55], modified by Ronen Barzel to eliminate that book's error, then further cleaned. A possible invocation is

```
unix% snap -bv -x 30 -y 18 -n 10 a.input > a.basis
```

which specifies binary mode (not necessary), and verbose printing of statistics related to the conversion. An alternate version of this code, **snaprad**, uses the same algorithms but adds a radial weighting to the inner product.

13.1.28 `stretch`

To increase the size of fields without adding any more information, **stretch** scales either axis by an integral factor using linear interpolation. The line

```
unix% stretch -x 30 -y 18 y 3 < a_small.field > a_big.field
```

generates the same number of fields as given in the input, but scaled to have the dimensions 30×54. Input dimensions are given as usual, with a character `x` or `y` specifying on which axis to operate, and the scale factor, and all transactions are done in binary. Used along with **select**, features in example fields may be picked out and magnified to cover a larger number of grid cells.

13.1.29 `subtract`

The second named input file is subtracted from the first and written to the output, all in binary unless the `-a` flag is used to specify `ASCII` mode. This utility is rarely useful, as places where it would be, such as in projection, already do the subtraction themselves.

13.1.30 testbasis

It is nice to ensure the property of orthonormality. The check

```
unix% testbasis a.basis
```

prints out an ASCII matrix indicating the extent to which this assumption fails. Entry (i, j) in the matrix is the absolute value of $\langle \mathbf{A}_i, \mathbf{A}_j \rangle$, but the diagonal is just zeros. A plot of this matrix should be completely flat, but regions of non-zero values indicate parts of the basis which should be modified. To test only the diagonal values, checking for normality, the option -d creates a list of the absolute values of $\langle \mathbf{A}_i, \mathbf{A}_i \rangle - 1$, which again should be all zeros. The flag -s generates only a summary listing of the locations and extents of the maximum diagonal and non-diagonal deviations from unity and zero, respectively.

13.1.31 toascii

This script implemements a one-line series of pipes to convert floating point values to ASCII using the system utility od, and puts only one output value per line. The exact command is:

```
exec od -An -v -F $1 | tr -d '\011' | sed 's/^[ ]*//; s/  / /' | tr ' ' '\012'
```

where the second space in the last sed command is actually a tab.

13.1.32 writeints

Often it is necessary to put some integers in binary form into an input stream to set up an integration. This utility scans ASCII integers from its input and writes them as binary on its output. A common use is in a script like

```
#!/bin/sh
writeints << EOF
17
17
23
EOF
cat NO.ic
⋮
```

which is run to specify the number of basis elements to use for each of three species, then the initial condition coefficient values, then projection coefficients.

13.1.33 zeros

This is quantitatively, by lines, and qualitatively, by function, the most trivial code which is still useful in designing basis space integrations. It writes the binary floating point value 0.0 to standard output some number of times (default zero) as specified using the -n option. The following two lines produce equivalent results:

```
zeros -n 10
yes 0 | fcat -n 10 -t ascii
```

13.2 Behind the scenes

This section consists of in-depth descriptions of all the computer source codes which make up the integrator package and all its support tools. Readers who are happy to use the methods without understanding the details are encouraged to skip this chapter, but hard core algorithmicians are welcome to follow along. It must be remembered in reading the following, that most of the routines fall in the category of sparse invocation, that is, they have been designed to be called only a few times during the process of designing and running an integrator. In this set, more attention has

been given to good error reporting and sometimes-redundant input validation to ensure robustness, sacrificing a few extra cycles of execution time and bytes of disk space. Sections where speed is of the essence will be duly noted at the points of their discussions. Another important design consideration was that of modularity. As seen in the discussions of the plethora of tools above, many small chunks are easier to design, debug, understand, and use, than would be one gigantic do-everything code. The sacrifices required by this design methodology are again speed and space, and heavily used sequences of these operators can be rewritten as single units.

13.2.1 Mechanism parser

The program `parse` takes a mechanism file as input and produces three output files which will be used in developing and running the integrator. The mechanism file contains up to five sections with different information about the reaction and planned integration, which were described in detail in the previous section. The design of this code is driven by the structures which hold all the information about the reaction; the forms of the structures determine the necessary manipulations to convert a mechanism into the three output files. Corresponding to the first three input file sections, *Sections*, *Constants*, and *Reactions*, there is a separate structure for each entry in such a section, and all entries are placed on the singly-linked list for that section. The first phase of the code is to parse the input file, building up the lists.

A `yacc` parser is used to do this straightforward conversion. An input file consists of a list of species, then an optional list of constants which may be either substitutes for numerical values or species whose concentrations do not vary (like oxygen in an atmospheric simulator), then a list of reactions. There may also be, at the bottom of the input, a list of partial derivatives for some species which will be converted slightly and placed in the script which generates the necessary field files. Also at the top is a section for options governing the action of the parser, for instance, whether the target code will be `C` or `FORTRAN`. Species and constant declarations are trivially handled by `yacc` support routines which add to the linked list, with a bit of consistency checking. Constants consist of a name, with an optional value assignment using an equals sign. This approach serves to decouple the mechanics of parsing from the more complicated, variable-dependent special attributions. A reaction is two lists of species joined by plus signs, separated into reactants and products by an arrow of any length (*e.g.*, `--->`), with an optional extra line specifying the complete reaction rate or just the reaction rate constant if the rate is simply the mass-action one specified by the reaction itself. A reaction is not necessarily stoichometrically balanced to allow the use of lumped reaction mechanisms, and also one side of a reaction may be empty. The form of rate expressions and constants is quite arbitrary, and corresponds to standard infix mathematical notation.

The lexical analyzer which processes tokens for the compiler grammar is divided by the use of start states into distinct sections for each of the corresponding sections in the input file, plus some globally active features to provide shell-script style comments and the arbitrary use of whitespace.

The second stage of the mechanism parser is input consistency checking. As mentioned in the introduction, pains are taken to make the code robust and debuggable since it is not part of the execution core. In this regard, there is only one routine per structure to allocate storage for a new entry on the list, and it takes care of setting all fields to good default values even though the condition of checking an uninitialized variable is never supposed to happen. One bit of consistency is to ensure that the user either provided array location numbers for all the species, or for none, and that no two species end up using the same entry in the array. Also, rate specifications commonly use the symbol '`k`' as a rate constant, but each reaction needs to have its own distinct constant. To this end, all occurrences of that symbol get promoted to the form '`k`*<number>*' where *<number>* is the same as the reaction number, its linear position in the input stream.

For each reaction which is added to the list, the reaction rate is generated assuming mass-action kinetics from the left-hand side of the reaction, if no rate is explicitly given. Then it is safe to balance the reactants and products such that each species is mentioned only on one side of the reaction. In fact, since the reactions are no longer needed in their explicit form (once the rate

has been calculated), a simple list of species and stoichiometric coefficient pairs is all that need be maintained.

The final section in the code is output generation. First the information about which species participate in each reaction is converted to state in which reactions does each species participate, essentially a transpose operation to store the information on a per-species basis instead of per-reaction. Also, the species are sorted such that the converted rate equations will be printed out in numerical order to make the output files more readable. Finally, the subroutine which is needed by the time integrator is generated, along with a header file with handy `#defines` for manipulating the coefficients. This output will compile in with the template programs directly.

13.2.2 Basis generator

The mathematical explanation of basis generation was discussed in Chapter 8. The actual implementation is straightforward given that discussion except for a few details which must be worked out. Input fields are read in, subject to dimensional constraints given on the command line, and stored in a single gigantic array, which is referenced using the macro

```
#define INPUTPOS(x,y,t) (input[(t)*xsize*ysize+(y)*xsize+(x)])
```

without bounds checking. Since the expected input sizes could be radically different for different problems, and are in general quite large, allocation of the input structure is done by chunks which double in size with each allocation. The actual sizes of the input array (in units of `xsize * ysize` doubles) are

$$10, 20, 40, 80, 160, \ldots.$$

There may be more efficient ways of doing this, but this compromise is fast and not too greedy.

After the extent of the input is known, space is allocated for the covariance matrix and its decomposed parts, as well as for some temporary fields used in the calculation. The covariance matrix calculated here should not rigorously be called that as the temporal average field is not subtracted from each input before integrating. What is actually being computed is

$$C(s,t) = \int_X u(\mathbf{x},t)u(\mathbf{x},s)\,d\mathbf{x}$$

while a true covariance would be

$$\widehat{C}(s,t) = \int_X \big(u(\mathbf{x},t) - u_0(\mathbf{x})\big)\big(u(\mathbf{x},s) - u_0(\mathbf{x})\big)\,d\mathbf{x}$$

where

$$u_0(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} u(\mathbf{x},i).$$

This difference in interpretation makes system conversion into basis space simpler, as no mean values need to be handled along with the straight summations over field number. Effectively the highest eigenvalue field in the basis becomes this mean field, and all the *real* basis elements are moved down one. Routines which implement the decomposition of this covariance matrix usually assume the input is of order one, so the maximum absolute element is determined and the entire matrix is scaled by its inverse to avoid some of the possible over- and under-flow problems.

Integration over a spatial domain presented some conceptual difficulties. Canned packages perform various discrete integrals based on different assumptions about the weighting of each point to the entire field. The tack that was taken here is to assume the values at grid points are exactly the values of the continuous field at that point. An alternative view would be to say that the grid points represent some sort of average over an area of the size of one grid cell. Given this first approach,

the assumption that values between grid points can be generated by bilinear interpolation between points gives a very trivial result:

$$\int_X f(\mathbf{x})\,d\mathbf{x} = \int_X \int_Y f(x,y)\,d\mathbf{x}\,dy \triangleq \sum_{x=1}^{N_x}\sum_{y=1}^{N_y} f(x,y)$$

(for the case of two dimensions). In cylindrical coordinates a radial weighting is imposed during the summation, which is perfect since the weighting is just r, linear in the radial coordinate. Other weighting factors may complicate this summation expression.

A few different eigenvalue solvers were considered before settling on the version in the code today. First, the *Numerical Recipes in C* [116] implementation was naively linked with the rest of the code. Although its performance usually seemed correct, it suffers from an unfortunate bug carried over from algorithm 8.3-2 in Golub [55] on which it was based. Next the EISPACK routine in **FORTRAN** was adopted, which is also based on an algorithm (in Algol) reported by Golub [54] though seems free of the particular bug encountered before. The main problem with this use was in data handling issues in linking the codes together, and its results were somewhat inaccurate, namely that the matrices U and V in $C = UWV^\top$ were not quite equal as they should be for a square C. Also cases frequently arose where the columns of U were not orthogonal. The full eigenvalue solver `Eigensystem` in *Mathematica* was used to test the SVD solvers, as it can be coerced into doing multiple word precision calculations. The final algorithm is a `C` code written by Ronen Barzel which is a translation of a reimplementation of the Golub algorithm in LISP, with the bug removed. It is fast, accurate, and hopefully, correct.

After the SVD solver has completed its work, the debugging checks on U and V described above are optionally performed. Then an output basis header is written, the eigenvalues are scaled back up, and a loop over the fields generates the spatial eigenfunctions. The columns of U are temporal eigenfunctions, which yield the spatial ones by

$$\phi_i(\mathbf{x}) = \int_T u(\mathbf{x},t)\beta_i(t)\,dt$$

where β_i is the i-th column of the matrix U. This integration is discretized using the same approach described earlier. Each $\phi(\mathbf{x})$ is further scaled such that $\langle\phi_i,\phi_j\rangle = 1$. A report on whether the condition $\langle\phi,\phi\rangle = 0$ for $i \neq j$ is satisfied may be printed, but a better check is to look at the relative magnitudes of the singular values.

In reality, however, larger orthogonal sets may be extracted using a power method, by finding exactly one eigenvalue and eigenvector pair at a time, then forcing the input data to be orthogonal to the newly extracted vector, and iterating up to the dimension of the set. This procedure requires more computation for a full decomposition, but is faster than SVD when only a few eigenvectors are needed.

13.3 Compiler technology

Many aspects of the work in this thesis have been greatly aided by the use of modern compiler-writing technology. Input files are coded in a natural, English-based language suited to the task, instead of being structured to make the machine code easier to write. What allows this easy bridging between internal code structures and human-based language is the ability to generate a parser from a specification of the language.

Being able to form a special language for each application allowed for support of the relatively nice input specification files in `parse` and `fields` described above, where the grammars were completely spelled out.

Reaction specifications—a list of species, constants, and reactions and their rates—are the fundamental starting point for utilities involved in basis space integration as well as allowing the development of two other related standalone packages.

13.3.1 xmap

The first of the two was designed during the first year of graduate study leading to this thesis as a tool to provide the ability to interactively study the properties of differential equations and iterative maps. The equations or mappings are typed directly into a text widget in a graphical display then parsed by the code and plotted. For example, typing the following lines into the text field sets up a simulation of chaotic atmospheric motion:

```
# Lorenz equations
dx/dt = pr (y - x)
dy/dt = - x z + r x - y
dz/dt = x y - b z
pr = 10;  b = 8/3;  r = 24.5  # This is the chaotic regime.
```

The input language is quite free-form: semicolons can be used to put multiple statements on a line, any valid C identifier can be used as a variable name, and text after # on a line is a comment. An example iterated map (called Henon's map) is

```
x -> 1 + y - a x^2
y -> b x
a = 1.4;  b = 0.3
```

An iterated map, such as this one, calculates the values of its variables at time $n+1$ using the values at time n, so in this case

$$x_{n+1} = 1 + y_n - ax_n^2$$
$$y_{n+1} = bx_n$$

Mappings are calculated directly, while differential equations are solved using the robust LSODE [63] solver.

The basic building block of the language in xmap is the *expression*, which resembles the mathematical concept of expression as being some string of added and subtracted terms. The following are all expressions:

```
5.743e-02
a
a + b
exp(-a) + a / b^3
-(12 + max(sin(pi*a), 0))^2
```

Standard precedence rules provide appropriate grouping of binary operators. Common functions including `exp`, `log`, `min`, and the trigonometrics are built-in. There are four types of statements understood by the parser.

1. Simple assignment: ⟨*var*⟩ = ⟨*expr*⟩
   The value of the variable on the left hand side is calculated using the expression on the right, at each time step.
2. Derivative assignment: d⟨*var*⟩/dt = ⟨*expr*⟩
   The time derivative of the variable is given by the expression. Note that there is only one independent variable, called "time" and given in the variable `time` (shortened to the `d/dt` notation for derivatives); assignments to `time` are overridden.
3. Mapping assignment: ⟨*var*⟩ -> ⟨*expr*⟩
   The discrete time iteration for the variable is given by the expression. The use of both discrete and continuous time operators is forbidden, as it makes no sense.

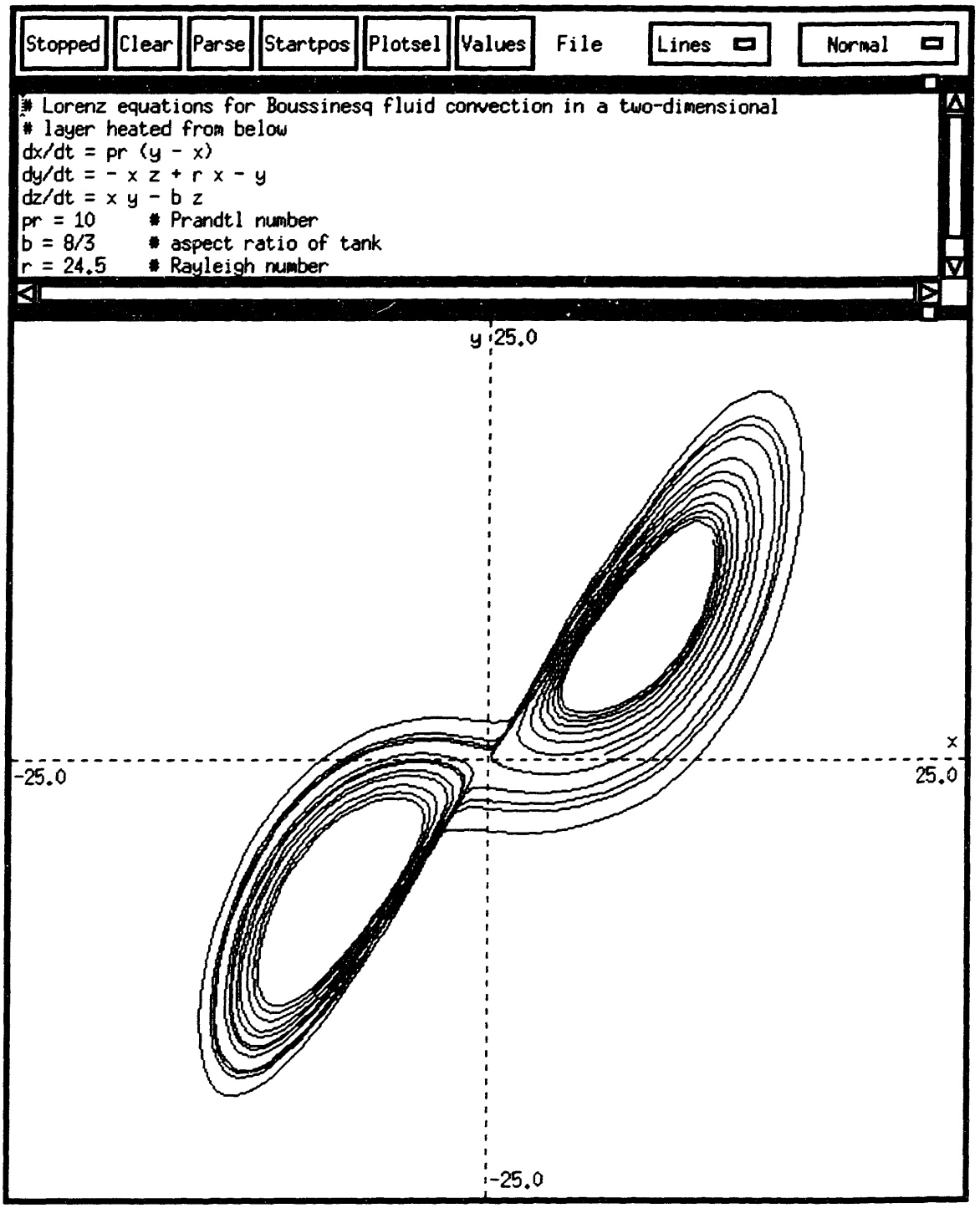


Figure 53. The xmap window. Image generated by a simple screen dump.

4. Initial condition assignment: IC ⟨*var*⟩ = ⟨*expr*⟩
   If not specified otherwise by the user, the initial condition for the variable is set using the given expression, evaluated at the initial time. The variable must be a variable for which a time derivative has been given.

There are two ways to start a simulation. The first is to click with the mouse in the plot window, where the position of the click gives the starting time and the initial condition of one distinguished variable, chosen through another option window. More complete control is given by a button in the header which prompts for values of all the derivative variables. Other functionality to control plotting parameters and the integrator are included, as well as some obscure functions. A sample snapshot of the entire window is given in Figure 53.

13.3.2 chemap

An extension to the code described above, **chemap** combines the interactive display properties of **xmap** along with the reaction mechanism parser used by **parse**. This allows the system of ordinary differential equations being simulated by **chemap** to be specified as a list of chemical reactions instead of having to generate the equations by hand.

Of the four basic expressions understood by **xmap**, only simple assignment and initial condition assignment are retained. These are augmented by reaction specifications of the form

⟨reactant⟩ + ⟨reactant⟩ + ··· ---> ⟨product⟩ + ⟨product⟩ + ···

with an optional rate specification immediately following the reaction (though possibly on a separate line), and an optional reaction rate constant specification. Reactions do not necessarily have any reactants, and some may have no products, but the reaction "--->" is unacceptable. Reaction rates are calculated using standard mass-action kinetics unless a rate is explicitly given, such as

```
rate = 1.5e-16 * NO2 * O3
```

or using the keyword **k** instead of **rate** and just specifying a term to be used as the constant in the mass-action specification. The concept of a variable is mostly replaced by the concept of the concentration of a chemical species, but variables are still used in calculating other constants or rate expressions.

Computationally it was found that very large mechanisms would take an unbearably long time to compute, thus an option was added to write the mechanism in the form of a **FORTRAN** subroutine, then fork a subprocess to make an integrator core executable which would communicate with the main program through shared memory using semaphores to interlock. The execution speed is greatly increased in this external mode, as the default internal calculation method computed the time derivatives by recursively descending a binary tree representing the differential equations.

13.4 Documentation

This section describes work done to improve the state of affairs of documentation, especially in environmental models. When environmental models are used for regulatory applications it is vital that the underlying computer codes are well documented and readily accessible to the widest possible audience. This section presents a series of tools, based on standard Unix utilities like **lex**, **awk**, and **grep**, and the word processing system TEX, which facilitate inclusion of the documentation directly into the computer code.

13.4.1 Introduction

One of the important trends in the debate about environmental issues is the use of models as an integrating framework to establish a concrete relationship between sources of pollution, like electric power plants and automobiles, and their receptors, especially human beings. For example, in the 1990 amendments to the Clean Air Act there is a specific requirement for the use of air quality modeling to develop the emissions control strategies needed to meet ambient air quality standards. This trend is also apparent in many other critically important areas including groundwater modeling, health risk evaluation, compliance assessment, and indoor air quality studies. In addition to the trend

toward the use of more models there is also a growth in the level of sophistication of the underlying mathematical formulations. One of the consequences of adding more physical and chemical detail into the models is that the underlying computer codes, even with the use of structured design techniques, become more complex. One of the most pressing problems that must be tackled by the environmental modeling community is how to document the underlying formulation and the numerical techniques embedded in their models. Quite apart from the issue of scientific validity of the codes themselves, it is also important to recognize that the models are used to develop regulatory processes that may involve extremely costly control measures. There is a clear need to pass from the "black box" approach to one that might be characterized as a "glass box" where all interested parties can see the design basis, any simplifying assumptions, and the numerical algorithms used in the code.

In most models two different types of documentation are usually employed. One is a conventional manual format that describes the purpose, algorithms, structure, and other aspsects of the code. The primary function of the documentation manual is to enable independent verification, operation, and maintenance of the software. Before the writing of the manuals, hopefully, another technique has been used to make the code itself a useful piece of documentation: comments, as provided for by structures of the language. In this paper we present examples of protocols that are being used for the latter purpose, and explore different formats for presenting code fragments with an eye toward improving the readability and simplifying the task of documenting complex computer codes.

Another attempt to producing self-documenting code is to redesign the language itself, as was done in Knuth's implementation of **WEB** [77, 75]. The main problem with this approach is that it requires programmers to learn a new language. Even if the variations are not too significant, the code is no longer portable to machines which do not speak this new language. **WEB** comes with a pair of translators, to **C** and **FORTRAN**, which yield compilable results (in an extremely unreadable format), and of course **WEB** text can be sent through TEX to create well-formatted code. The aim of this paper is not to convince programmers to switch to another language which lends itself well to self-contained documentation, but instead to augment proven languages to improve their documentability. This way there is no impairment to any optimized statements one wishes to write in **FORTRAN**, for example, and no incompatibility with sites which do not choose to employ our system. The term "automatic" is used to refer to the ease with which a programmer can add meaningful comments, making the dream of self-documentation a reality.

13.4.2 Uncommented code

As an illustration of the problem consider Figure 54, a simple function whose purpose is to compute the convective velocity scale w_*. This parameter is used as a part of the much larger calculation of vertical diffusivity in the atmosphere under unstable conditions. For the purposes of illustration it is not necessary to understand the details of atmospheric transport. The calculation is a typical representative of routines that might be found in any environmental model.

Several points are worth noting about the code. One good feature is the use of the **FORTRAN** if-then-else construct for logical branching. Apart from that attempt at structured coding in such an uncompromising language there is little else to recommend the code as a positive example. The most glaring omission is the lack of any descriptive comments whatsoever to explain what the statements are doing.

13.4.3 Code documentation using internal comments

A slightly better version of the code can be constructed if we make use of the **FORTRAN** comment statement. In addition to detailed header descriptions of each module, a major effort has been made to ensure that the software itself is self-documenting. Following the guidelines of Kernigan and Plauger [72], coding comments are structured identically for each section, as follows:

> **Description:** At the beginning of the program there is a description of its purpose and relationship to the rest of the model code.

```
      FUNCTION WS(ZR, Z0, UBAR, RMOLEN, ZI)
      DATA VK/ 0.40 /
      PZR = (1.0-15.0*ZR*RMOLEN)**(0.25)
      PZ0 = (1.0-15.0*Z0*RMOLEN)**(0.25)
      PHI = ALOG(ZR/Z0) + ALOG( (PZ0**2+1.0)*(PZ0+1.0)**2/
     &  ((PZR**2+1.0)*(PZR+1.0)**2) ) + 2.0*(ATAN(PZR) - ATAN(PZ0))
      IF (UBAR .GT. 0.4) THEN
          US = VK*UBAR/PHI
      ELSE
          US = VK*0.4/PHI
      ENDIF
      WS = ((-ZI*RMOLEN/VK)**0.333333)*US
      RETURN
      END
```

Figure 54. Example of undocumented code.

Authentication: This section of the comment header gives the author's name, program creation date and current version number. This information may alternately be inserted automatically by a source code control system like RCS.
Call sequence: Variables that must be supplied by the calling program and those that are returned are described here.
Variables: A list of the key variables used inside the module is given here. Secondary variables may be described in the body of the code itself, or ignored in the case of trivial loop control integers.
Literature references: Books and journal articles are listed to explain reasoning behind the algorithms, assumptions, and data structures used inside the module.
Warnings: Any cautions regarding the use of the code outside its original environment are given to the user here.
Test cases: When appropriate, example inputs are given along with the correct results. These test cases are particularly useful when differences in machine precision is a cause for concern.

While including the information listed above is sufficient for fully documenting the code, complex formulas are still quite difficult to interpret since the format FORTRAN uses for mathematical expressions is a very meager subset of the constructs available to humans when writing mathematics. A first step towards making the code a little easier to read is to provide a clearer separation between the comments and the FORTRAN source language itself. Consider the listing in Figure 55. Comments are set flush left in a variable width font while the FORTRAN statements themselves remain fixed.

13.4.4 Documented code using TEX

While the use of another font and removal of the C at the beginning of each line has improved the appearance of comments in the code, it has not really made the statements themselves easier to understand. Until the advent of efficient programming languages which accept natural human languages as input, we are forced to translate our thoughts into and out of FORTRAN. The main limitation of FORTRAN is that the arithmetic notation is visually linear. With the current syntax it is not possible to have sub- and super-scripts or to use common non-arabic mathematical symbols. One way around the problem is to intersperse text processing commands within the code itself. In particular by employing a formatting tool such as TEX it is a straightforward task to merge text with mathematical notation and even graphics. Moreover if the text processing commands have a C

```
      FUNCTION WS(ZR, ZO, UBAR, RMOLEN, ZI)
```

Description

This program has been designed to calculate the surface level convective velocity scale w* for use in determining mixing times. The calculations are based on the integral form of the Businger phi function for unstable conditions.

Author

Gregory J. McRae (March 3, 1993)
Massachusetts Institute of Technology
Cambridge, MA 02139
(617) 253-6564

Variables

ZR	—	Reference height for the wind measurements
ZO	—	Surface roughness
UBAR	—	Absolute value of the velocity at the reference height
WS	—	Convective velocity scale
US	—	Friction velocity
RMOLEN	—	Reciprocal of the Monin-Obukhov length
VK	—	von Karman constant
ZI	—	Mixed layer height

Notes

1. The program is restricted, by definition, to unstable conditions.
2. The units for the input data must be consistent. In particular note that ZR and ZO must be the same.

Beginning

Set the von Karman constant k

```
      DATA VK/ 0.40 /
```

Compute the phi functions for zr and z0

```
      PZR = (1.0-15.0*ZR*RMOLEN)**(0.25)
      PZO = (1.0-15.0*ZO*RMOLEN)**(0.25)
```

Calculate the friction velocity using a form that is non divergent, in a numerical sense, when 1/L=0

```
      PHI = ALOG(ZR/ZO) + ALOG( (PZO**2+1.0)*(PZO+1.0)**2/
     &  ((PZR**2+1.0)*(PZR+1.0)**2) ) + 2.0*(ATAN(PZR) - ATAN(PZO))
```

Use a minimum wind speed of 0.4 m/s to calculate u*

```
      IF (UBAR .GT. 0.4) THEN
        US = VK*UBAR/PHI
      ELSE
        US = VK*0.4/PHI
      ENDIF
```

Calculate the convective velocity scale w*

```
      WS = ((-ZI*RMOLEN/VK)**0.333333)*US
```

End of the program

```
      RETURN
      END
```

Figure 55. Same example code, this time with comments.

in the first column, they will be ignored by the compiler and the code will act exactly as before their insertion. This approach of providing auxiliary information within FORTRAN source code is often used to generate special behavior on certain architectures such as parallelization on a multiprocessor machine.

As an example of attractive comments, consider the line of code from our example that determines the convective velocity, with a few comment lines added:

```
CTEXC Calculate the convective velocity scale $w_*$
CTEXE w_* = \left(-{Z_i\over kL}\right)^{1\over3} u_*
C
      WS = ((-ZI*RMOLEN/VK)**0.333333)*US
C
```

The CTEXC directive to TEX indicates that the following characters are to be treated as text comments while CTEXE indicates a displayed equation. In accordance with FORTRAN convention and to maintain compatibility with older compilers, the text associated with the directives starts in the seventh column. Given these directives and a simple Unix shell script applied to the source file, the resulting output is:

Calculate the convective velocity scale w_*

$$w_* = \left(-\frac{Z_i}{kL}\right)^{\frac{1}{3}} u_*$$

```
WS = ((-ZI*RMOLEN/VK)**0.333333)*US
```

There are two other TEX directives in the system developed for this project. CTEXP is used to include a PostScript file from some drawing program or maybe a graph of input/output relationships or execution times. A pictorial description is sometimes invaluable in explaining the operation of a section of code and is no harder to include in the source code than are a thousand words, by this syntax:

```
CTEXP /usr/users/pw/ws/ws_descrip.i
```

CTEXF instructs the TEX processor to include the contents of some other file of commands and is exactly equivalent to TEX's "`\input`" statement except that the comment line containing CTEXF is not included in the output. We use this directive to add a standard statement about where the authors can be reached and whatever copyright information the university requires. This avoids having to change each source file every time the telephones get new numbers. The CTEXF directive is also handy for including specialized lists of fonts, or each person's personal macro package, or the school's logo.

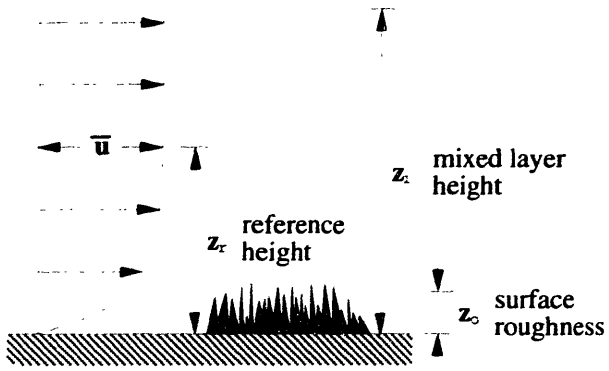
Incidentally, the text processing directives could be removed by a simple **sed** script or replaced with conventional-looking comments by stripping the TEX-specific parts. In practice there is no need, however, because all the directives and descriptive text have been treated as FORTRAN comments, but a conventional listing may be preferred by some programmers. The script **dectex** is included with the distribution to accomplish this and also as an illustration of just how useful the standard Unix utilities are for routine processing of this sort: five lines of commands to the interpreted language **sed** instead of however many pages of C or FORTRAN which would require compilation and debugging.

The result of processing our complete example function through TEX is now shown in Figure 56. It is an example that shows how the FORTRAN code can be documented in a way that is closer to the presentations likely to be encountered in conventional background manuals or research papers. The key advantage of embedding the text into the code is that the code and its documentation are always together. Also, it is much less of a hassle to modify the documentation at the same time as the code if they both reside in the same file, and future programmers are more likely to remember to do so.

```
FUNCTION WS(ZR, ZO, UBAR, RMOLEN, ZI)
```

Description

This program has been designed to calculate the surface level convective velocity scale w_* (also known as the free convection scaling velocity). The calculations are based on a similarity theory and in particular the integral form of the Businger ϕ functions for unstable meteorological conditions. For deep mixing layers, with vigorous heating at the ground w_* can be of the order of 1–2 m/s with associated mixing times, defined by $\tau = Z_i/w_*$ of O(10 minutes). The figure below illustrates the boundary layer model.



Author

Gregory J. McRae (March 3, 1993)
Department of Chemical Engineering 66-466
Massachusetts Institute of Technology
Cambridge, MA 02139
(617) 253-6564

Call variables

- `WS` — Function returns convective velocity scale (w_*)
- `ZR` — Reference height for the wind measurements (z_r)
- `ZO` — Surface roughness height (z_0)
- `UBAR` — Absolute value of the wind velocity at the reference height ($\bar{u}$)
- `RMOLEN` — Reciprocal of the Monin-Obukhov length $1/L$
- `ZI` — Mixed layer height (Z_i)

Other variables

- `US` — Friction velocity (u_*)
- `VK` — Von Karman constant (k)

Warnings

1. The program is restricted, by definition, to unstable conditions, and so the Monin-Obukhov length must be negative.
2. The units upon input must be consistent. In particular note that `ZR`, `ZO`, and `RMOLEN` must all agree.
3. The wind velocity is the magnitude of the flow field.

Test cases

z_0	z_r	z_i	$\bar{u}$	$1/L$	w_*
0.001	10.0	500.0	3.0	-1.000	1.926
0.010	10.0	500.0	5.0	-0.100	1.716
0.100	10.0	500.0	10.0	-0.010	2.140
1.000	10.0	500.0	10.0	-0.010	4.488

References

1. Businger, J. A. et al. (1971), "Flux-profile relationships in the atmospheric surface layer," *J. Atmospheric Science*, **28**, 181–189.
2. Stull, R.B., (1988), *An Introduction to Boundary Layer Meteorology*, Kluwer Academic Publishers, Dordrecht.

Beginning

Set the von Karman constant k

```
DATA VK/ 0.40 /
```

In order to evaluate the momentum scaling we need to calculate integrals of the form

$$\phi_m = \int_{z_0}^{z_r} \phi\left(\frac{z}{L}\right)\frac{dz}{z}$$

where the two "phi" functions relevant to the determination of w_* are

$$\phi(\tfrac{z_r}{L}) = \left(1.0 - 15.0\tfrac{z_r}{L}\right)^{\frac{1}{4}}, \qquad \phi(\tfrac{z_0}{L}) = \left(1.0 - 15.0\tfrac{z_0}{L}\right)^{\frac{1}{4}}$$

```
PZR = (1.0-15.0*ZR*RMOLEN)**(0.25)
PZ0 = (1.0-15.0*Z0*RMOLEN)**(0.25)
```

The resulting integral ϕ_m, in a form that is numerically non-divergent when $\frac{1}{L} \to 0$, is

$$\phi_m = \ln\left(\frac{z_r}{z_0}\right) + \ln\left[\frac{(\phi^2(\frac{z_0}{L})+1)(\phi(\frac{z_0}{L})+1)^2}{(\phi^2(\frac{z_r}{L})+1)(\phi(\frac{z_r}{L})+1)^2}\right] + 2\arctan\phi\left(\frac{z_r}{L}\right) - 2\arctan\phi\left(\frac{z_0}{L}\right)$$

```
PHI = ALOG(ZR/Z0) + ALOG( (PZ0**2+1.0)*(PZ0+1.0)**2/
&  ((PZR**2+1.0)*(PZR+1.0)**2) ) + 2.0*(ATAN(PZR) - ATAN(PZ0))
```

The friction velocity, used as a measure of the surface momentum flux, is defined by

$$u_*^2 = \frac{\tau_0}{\rho} = -\overline{u'w'}$$

where τ_0 is the shear stress per unit area of the boundary and ρ is the density of the fluid. In this module the friction velocity is based on the wind speed u_r at the reference elevation z_r. The minimum acceptable wind speed is 0.4m/s to calculate the friction velocity u_*. Similarity theory is also used in this case.

$$u_* = \begin{cases} \dfrac{k\bar{u}}{\phi(\frac{z}{L})}, & \text{if } \bar{u} > 0.4 \\ \dfrac{0.4\,k}{\phi(\frac{z}{L})}, & \text{otherwise} \end{cases}$$

```
IF (UBAR .GT. 0.4) THEN
  US = VK*UBAR/PHI
ELSE
  US = VK*0.4/PHI
ENDIF
```

Calculate the convective velocity scale w_*

$$w_* = \left(\frac{-Z_i}{kL}\right)^{\frac{1}{3}} u_*$$

```
WS = ((-ZI*RMOLEN/VK)**0.333333)*US
```

End of the program

```
RETURN
END
```

Figure 56. Completely formatted example code.

13.4.5 Simple UNIX Documentation

In developing the format for the comments, keywords have been used to separate the different sections. One of the advantages of this approach is that standard UNIX facilities such as **grep** and **awk** (see Aho [5] and Kernigan and Pike [71]) can be used to get minimal documentation of the module. By using some **awk** commands it is possible to retrieve, from the source files, all the internal documentation of the code. The particular implementation is simple and has a syntax of the form

```
% about -hdrvw <program_name>
```

where in the above example <program_name> is WS. The switches hdrvw indicate help, description, references, variable names, and warnings. For example in the current implementation of the system the command

```
% about -d WS
```

would produce

```
This program has been designed to calculate the surface level convective ve-
locity scale w* for use in determining mixing times. The calculations are based
on the integral form of the Businger phi function for unstable meteorological
conditions.
```

13.4.6 Automatic retrieval of internal documentation

One of the unfortunate facts of life is that the program manuals that describe the computer codes are often not available when software maintenance or modification must be carried out. Or worse yet, information about the code was never produced in the first place. One way to avoid this problem is to make sure that the code itself contains the needed information. This can be accomplished by including in the code some directives that identify the form of the documentation. And placing these directives right alongside program statements, it becomes a trivial matter for the programmer to add documentation. In the present system these directives are implemented by using a special code word as the only word on a CTEXC directive line.

Insertion of these key phrases enables a simple mechanism for self-documentation of each individual module. The filter can also be used to extract the functionality of the program using a variety of command-line switches. Without any flags, **about** produces a listing of the program. The available option flags correspond to the recognized code words. Together, they are:

-a	**Author**	Author(s), version information, date
-c	**Call Sequence**	Invocation statement of the subroutine
-s	**Subroutines**	Required subroutines
-d	**Description**	Purpose of the program
-v	**Variables**	Important internally used variables
-w	**Warnings**	Any limitations or warnings
-r	**References**	Literature references
-t	**Test Data**	Example inputs and outputs for verification
	Beginning	Start of the program statements
	End	End of the program statements

Note that no code words are ever required by the utilities, but it is recommended that all applicable codes by used for each routine. For example **about -r WS** would produce

```
Literature references relevant to the routine WS:
    1. Businger, J. A. et al. (1971), "Flux-profile relationships in the
    atmospheric surface layer," J. Atmospheric Science, 28, 181--189.
    2. Stull, R.B., (1988), An Introduction to Boundary Layer
    Meteorology, Kluwer Academic Publishers, Dordrecht.
```

and **about -cv WS** would return

```
Call sequence for routine WS:
     FUNCTION WS(ZR, ZO, UBAR, RMOLEN, ZI)
     ZR     - Reference height for the wind measurements
     ZO     - Surface roughness
     UBAR   - Absolute value of the velocity at the reference height
     RMOLEN - Reciprocal of the Monin-Obukhov length
     ZI     - Mixed layer height
Variable list for routine WS:
     WS     - Convective velocity scale
     US     - Friction velocity
     VK     - von Karman constant
```

and about `-w WS` would return the limitations and warning messages:

```
Limitations and warning messages for routine WS:
     1. The program is restricted, by definition, to unstable conditions.
     2. The units upon input must be consistent.  In particular note
        that ZR, ZO, and RMOLEN must all agree.
     3. The wind velocity is the magnitude of the flow field.
```

13.4.7 Source control

In a system the size of an AIRSHED model it is often a major task to keep track of software updates and modifications. Several difficulties have arisen in the past because the code has been changed by inexperienced users or by more than one user on the same working file system. In order to simplify the task of tracking down errors and making sure that all the programs are consistent, each source file is maintained by a source control system. The purpose of such a system is to maintain consistency of a set of files. To this end, only one user is allowed to lock a given file for editing. When he is finished with his changes, the file is readmitted to the system which prompts him for a description of changes. All past versions are stored in a space-efficient way by preserving only the amount of information necessary to reproduce the edit history. Each time the environmental model is recompiled, the current version numbers are included in the executable as string data and can be checked at any time using the UNIX utility `what`. Scripts which are used to run the model can be written to verify that the proper version is being used in this way. The particular source control package we prefer is RCS [76], which resides in the public domain, but similar software is available from most operating system vendors.

13.4.8 Conclusion

The system of structured comments inside source code and UNIX utilities for extraction and processing will encourage programmers to write meaningful notes about their code sections at the time when the code is being created. The advantage to having the documentation be an integral part of software development is clear. Properly documented code will ensure its reusability and continued support even after the modeling project has been deemed complete, and make it easier for researchers to share their models since all the knowledge that was involved in writing the code is explained therein.

References

The page numbers of citations are given in brackets after each reference.

[1] Abarbanel, H. D. I., Brown, R., and Kennel, M. B., "Local Lyapunov exponents computed from observed data," *Journal of Nonlinear Science*, **2**, n. 3, pp. 343–366, 1992. [41, 42]

[2] Abarbanel, H. D. I., Brown, R., and Kennel, M. B., "Lyapunov exponents in chaotic systems: their importance and their evaluation using observed data," *International Journal of Modern Physics B*, **5**, n. 9, pp. 1347–75, 1991. [41, 42]

[3] Abarbanel, H. D. I., Brown, R., Sidorowich, J. J., and Tsimring, L. S., "The analysis of observed chaotic data in physical systems," *Reviews of Modern Physics*, **65**, n. 4, pp. 1331–92, 1993. [39, 41]

[4] Abu-Mostafa, Yaser, *via* personal communication with Allen Knutson. [99]

[5] Aho, A. V., Kernigan, B. W., and Weinberger, P. J., *The AWK Programming Language*, Addison-Wesley Publishing Company, Reading, MA, 1988. [182]

[6] Ames, William F., *Numerical methods for partial differential equations*, Academic Press, New York, 1977. [140]

[7] Arvind and Iannucci, R. A., "Two fundamental issues in multiprocessing," in *Proceedings of the DFVLR—1987 conference on parallel processing in science and engineering, Bonn-Bad Godesberg, W. Germany*, Springer-Verlag LNCS 295, June 1987. [54]

[8] Aubry, N., Holmes, P., Lumley, J. L., and Stone, E., "The dynamics of coherent structures in the wall region of a turbulent boundary layer," *Journal of Fluid Mechanics*, **192**, 115–173, 1988. [83]

[9] Auerbach, D., Cvitanović, P., Eckmann, J.-P., Gunaratne, G., and Procaccia, I., "Exploring chaotic motion through periodic orbits," *Physical Review Letters*, **58**, n. 23, pp. 2387–9, 1987. [37]

[10] Bacry, E., Mallat, S., and Papanicolaou, G., "A wavelet based space-time adaptive numerical method for partial differential equations," Courant Institute preprint. [68, 76]

[11] Bajaj, A. K. and Johnson, J. M., "On the amplitude dynamics and crisis in resonant motion of stretched strings," *Philosophical Transactions of the Royal Society of London A*, **338**, n. 1649, pp. 1–41, 1992. [37, 39]

[12] Bank, R., "A multilevel iterative method for nonlinear elliptic equations," in Schultz, M., ed., *Elliptic Problem Solvers*, Academic Press, New York, 1981. p. 1. [68]

[13] Batarseh, K. I., and Stiller, A. H., "Modeling the role of bacteria in leaching of low-grade ores," *AIChE Journal*, **40**, n. 10, 1741–1768, 1994. [12]

[14] Becker, M., *The Principles and Applications of Variational Methods*, MIT Press, Cambridge, MA, 1964. [71]

[15] Beguelin, A., Dongarra, J. J., Geist, G. A., Manchek, R., and Sunderam, V. S., *A User's Guide to PVM Parallel Virtual Machine*, Technical report ORNL/TM-11826, Oak Ridge National Laboratory, 1991. [58]

[16] Berger, M. and Colella, P., "Local adaptive mesh refinement for shock hydrodynamics," *Journal of Computational Physics*, **82**, pp. 64–84, 1989. [68]

[17] Berger, M. and Oliger, J., "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of Computational Physics*, **53**, n.3, pp. 484–512, 1984. [68]

[18] Beylkin, G., Coifman, R., and Rokhlin, V., "Fast wavelet transform and numerical algorithms," *Yale University* Technical Report YALE/DCS/RR-696, August 1989. [69]

[19] Bird, R. B., Stewart, W. E., and Lightfoot, E. N., *Transport Phenomena*, John Wiley and Sons, New York, 1960. [11]

[20] Biswas, R., Devine, K. D., and Flaherty, J. E., "Parallel, adaptive finite element methods for conservation laws," to appear in *Applied Numerical Mathematics*, 1994. [68]

[21] Block, L., Guckenheimer, J., Misiurewics, M., and Young, L.-S. in *Global Theory of dynamical Systems*, Nitecki, Z. and Robinson, C., eds., Lecture Notes in Mathematics v. 819, Springer-Verlag, Berlin, 1980, p. 18. [51]

[22] Bozeman, J. and Dalton, C., *Journal of Computational Physics*, **12**, pp. 348–363, 1973. [18, 19]

[23] Brown, R., "Calculating Lyapunov exponents for short and/or noisy data sets," *Physical Review E* **47**, n. 6, pp. 3962–9, 1993. [42, 52]

[24] Brown, R., "Orthonormal polynomials as prediction functions in arbitrary phase space dimensions," preprint, 1993. [37, 40, 41, 52]

[25] Brown, R., Bryant, P., and Abarbanel, H. D. I., "Computing the Lyapunov spectrum of a dynamical system from an observed time series," *Physical Review A*, **43**, n. 6, pp. 2787–806, 1991. [42]

[26] Brown, R., Rulkov, N. F., and Tracy, E. R., "Modeling and synchronizing chaotic systems from time-series data," *Physical Review E*, **49**, n. 5A, pp. 3784–3800, 1994. [37, 40, 41, 52]

[27] Brown, R., Rulkov, N., and Tufillaro, N. B., "Synchronization of chaotic systems: the effects of additive noise and drift in the dynamics and driving," *Physical Review E*, submitted June 1994). [37, 41]

[28] Burgers, J. M., *The nonlinear diffusion equation: asymptotic solutions and statistical problems*, D. Reidel Pub. Co., Boston, 1974. [140]

[29] Burggraf, O., *Journal of Fluid Mechanics*, **24**, pp. 113–151, 1966. [18]

[30] Candy, J. C., *Signal processing: The modern approach*, McGraw-Hill, New York, 1988. [52]

[31] Carter, W., "A detailed mechanism for the gas-phase atmospheric reactions of organic compounds," *Atmospheric Environment*, **24A**, n. 3, 481–518, 1990. [126]

[32] Ciric, A. R., and Gu, D., "Synthesis of nonequilibrium reactive distillation processes by MINLP optimization," *AIChE Journal*, **40**, n. 9, 1479–87, 1994. [12]

[33] Cole, Julian D., "On a quasi-linear parabolic equation occuring in aerodynamics," *Quarterly of Applied Mathematics*, **9**, pp. 225–236 (1951). [140]

[34] Collatz, L., *The Numerical Treatment of Differential Equations*, Springer-Verlag, New York, 1966. [71]

[35] Constantinou, L., and Gani, R., "New group contribution method for estimating properties of pure compounds," *AIChE Journal*, **40**, n. 10, 1697–710, 1994. [12]

[36] Cvitanović, P., Gunaratne, G. H., and Procaccia, I., "Topological and metric properties of Hénon-type strange attractors," *Physical Review A*, **38**, n. 3, pp. 1503–20, 1988. [37, 44, 47, 52]

[37] Davis, G., Mallat, S., and Avellaneda, M., "Adaptive nonlinear approximations," Courant Institute preprint, New York University. [77]

[38] Davis, G., Mallat, S., and Zhang, Z., "Adaptive time-frequency approximations with matching pursuits," Courant Institute preprint, New York University. [77]

[39] Doherty, M. F. and Perkins, J. D., "On the dynamics of distillation process. III: the topological structure of ternary residue curve maps," *Chemical Engineering Science*, **34**, n. 12, pp. 1401–14. [27]

[40] Duncan, W. J., *ARC R&M*, p. 1798, 1937. [70]

[41] Duncan, W. J., *ARC R&M*, p. 1848, 1938. [70]

[42] Eckmann, J.-P. and Ruelle, D., "Ergodic theory of chaos and strange attractors," *Reviews of Modern Physics*, **57**, n. 3, pp. 617–56, 1985. [41, 42]

[43] Engquist, B., Osher, S., and Zhong, S., "Fast wavelet based algorithms for linear evolution equations," *SIAM Journal of Scientific Computing*, **15**, n. 4, pp. 755–775, 1994. [69]

[44] Fick, Th., Dath, J.-P., Imbihl, R., and Ertl, G., "The mechanism of kinetic oscillations in the NO + CO reaction on Pt(100)," *Surface Science*, **251/252**, pp. 985–989, 1991. [28]

[45] Field, R., and Noyes, R., "Oscillations in Chemical Systems. IV. Limit cycle behavior in a model of a real chemical reaction," *Journal of Chemical Physics*, **60**, pp. 1877–1884, 1974. [28]

[46] Finlayson, B. A., *The Method of Weighted Residuals and Variational Principles*, Academic Press, New York, 1972. [72]

[47] Fletcher, C. A. J., "Burgers' equation: a model for all reasons," in Noye, J., ed. *Numerical Solutions of Partial Differential Equations*, North-Holland, New York, pp. 139–225, 1982. [140]

[48] Fletcher, C. A. J., *Computational Galerkin methods*, Springer-Verlag, New York, 1984. [70]

[49] Fraser, A. M., "Information and entropy in strange attractors," *IEEE Transactions on Information Theory*, **35**, n. 2, pp. 245–62, 1989. [39]

[50] Fujisaka, H. and Yamada, T., "Stability theory of synchronized motion in coupled-oscillator systems," *Progress of Theoretical Physics*, **69**, n. 1, pp. 32–47, 1983. [40, 52]

[51] Galërkin, B. G., *Vestnik Inzhenerov i Tekhnikov*, **19**, pp. 897–908, 1915. [70]

[52] Garfinkel, R. S., and Nemhauser, G. L., "The set-partitioning problem: set covering with equality constraints," *Operations Reseach*, **17**, n. 5, 848–856, 1969. [58]

[53] Garfinkel, Robert S., and Nemhauser, George L., *Integer Programming*, John Wiley & Sons, New York, 1972. [58]

[54] Golub, G. and Reinsch, C., *Numerische Mathematik*, **14**, 403–420, 1970. [172]

[55] Golub, Gene H., and Van Loan, Charles F., *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1983. [168, 172]

[56] Grassberger, P., Hegger, R., Kantz, H., "On noise reduction methods for chaotic data," *Chaos* **3**, n. 2, pp. 127–42, 1993. [39]

[57] Guida, V., Ocone, R., and Astarita, G., "Diffusion-convention-reaction in multicomponent mixtures," *AIChE Journal*, **40**, n. 10, 1665–68, 1994. [12]

[58] Hall, T., "The creation of horseshoes," *Nonlinearity*, **7**, n. 3, pp. 861–924, 1994. [44, 50]

[59] Hall, T., "Weak universality in two-dimensional transitions to chaos," *Physical Review Letters*, **71**, n. 1, pp. 58–61, 1993. [51]

[60] Hall, T., "Fat one-dimensional representatives of pseudo-Anosov isotopy classes with minimal periodic orbit structure," *Nonlinearity*, **7**, n. 2, pp. 367–84, 1994. [44, 50]

[61] Hamming, R. W., *Numerical Methods for Scientists and Engineers*, McGraw-Hill, New York, 1973. [75]

[62] Hansen, K. T., *Symbolic dynamics in chaotic systems*, PhD Thesis, University of Oslo, 1993. [37, 44, 47, 52]

[63] Hindmarsh, A. C., "ODEPACK, a systematized collection of ODE solvers," in Stepleman, R. S., *et al.*, eds., *Scientific Computing*, (IMACS Transactions on Scientific Computation, v. 1), North-Holland, Amsterdam, 1983, pp. 55–64. [119, 124, 162, 173]

[64] Hindmarsh, A. C., **GEARB**—Solution of ordinary differential equations having banded Jacobian, UCID-30059 rev. 2, Lawrence Livermore Laboratory, Livermore, CA, June 1977. [108]

[65] Hopf, E., "The partial differential equation $u_t + uu_x = \mu u_{xx}$," *Communications on Pure and Applied Mathematics*, **3**, n. 3, 201–230, 1950. [140]

[66] Johnson, J. M. and Bajaj, A. K., "Amplitude modulated and chaotic dynamics in resonant motion of strings," *Journal of Sound and Vibration*, **128**, n. 1, pp. 87–107, 1989. [36]

[67] Jolliffe, I. T., *Principal Component Analysis*, Springer-Verlag, New York, 1986. [82]

[68] Kantorovich, L. V. and Krylov, V. I., *Approximate Methods of Higher Analysis*, Interscience Publishers, Inc., New York, 1964. [73, 74]

[69] Kantz, H., Schreiber, T., Hoffmann, I., "Nonlinear noise reduction: A case study on experimental data," *Physical Review E*, **48**, n. 2, pp. 1529–38, 1993. [39]

[70] Karhunen, K., "Zur spektraltheorie stochastischer prozesse," *Annales Academiae Scientiarum Fennicae*, Series A, **1**, p. 34, 1946. [78]

[71] Kernigan, B. W., and Pike, R., *The UNIX Programming Environment*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984. [182]

[72] Kernigan, B. W., and Plauger, P. J., *The Elements of Programming Style*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974. [176]

[73] Kernighan, B. W., "Little languages," talk at Harvard University, November 3, 1994, 4:15 P.M. [12]

[74] Kirchhoff, G. R., *Vorlesungen über Mathematische Physik*, B. G. Teubner, Leipzig, 1883. [36]

[75] Knuth, D. E., "Literate programming," *The Computer Journal*, **27**, pp. 97–111, 1984. [176]

[76] Tichy, W. F., "RCS—a system for version control," *Software—Practice & Experience*, **15**, 7, pp. 636–654, July 1985. [183]

[77] Knuth, D. E., "The **WEB** system of structured documentation," Stanford Computer Science Report 980, Stanford, CA, September 1983. [176]

[78] Kontogeorgis, G. M., Fredenslud, A., and Tassios, D. P., "Equations of state and activity coefficient models for vapor-liquid equilibria of polymer solutions," *AIChE Journal*, **40**, n. 10, 1711–27, 1994. [12]

[79] Kosniowski, C., *A First Course in Algebraic Topology*, Cambridge University Press, Cambridge, 1980. [19]

[80] Kostelich, E. J. and Schreiber, T., "Noise reduction in chaotic time-series data: A survey of common methods," *Physical Review E*, **48**, n. 3, pp. 1752–63, 1993. [39]

[81] Lefschetz, S., *Topology*, American Mathematical Society, New York, pp. 274–278 and 374–377, 1930. [23]

[82] Loève, M., "Fonctions aléatoire de second ordre," *Comptes Rendus des Seances de l'Academie des Sciences*, **220**, 1945. [78]

[83] Lorenz, E., "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, **20**, pp. 130–141, 1963. [28]

[84] Madday, Y. and Ravel, J. C., "Adaptivite par ondelettes: conditions aux limites et dimensions superieures," *Université Pierre et Marie Curie, Laboratoire d'Analyse Numerique* Technical Report, January 1992. [69]

[85] Mallat, S., and Zhang, Z., "Matching pursuit with time-frequency dictionaries," Courant Institute technical report 619, New York University, 1992. [77]

[86] McRae, G. J., *Mathematical modeling of photochemical air pollution, §8.6, photolytic rate constants*, PhD thesis, California Institute of Technology, Pasadena, CA, 1981. [130]

[87] McRae, G. J., Consulting Report to the California Air Resources Board (CALGRID). [104, 108]

[88] Melvin P. and Tufillaro, N. B., "Templates and framed braids," *Physical Review A*, **44**, n. 6, pp. 3419–22, 1991. [44]

[89] Mielke, A., Holmes, P., and O'Reilly, O., "Cascades of homoclinic orbits to, and chaos near, a Hamiltonian saddle-center," *Journal of Dynamics and Differential Equations*, **4**, n. 1, pp. 95–126, 1992. [37]

[90] Miles, J., "Resonant, nonplanar motion of a stretched string," *Journal of the Acoustical Society of America*, **75**, n. 5, pp. 1505–10, 1984. [36, 37]

[91] Milnor, J. W., *Topology from the Differentiable Viewpoint*, University Press of Virginia, Charlottesville, 1965. [23]

[92] Milnor, J. and Thurston, W. in *Dynamical Systems*, Alexander, J. C., ed., Lecture Notes in Mathematics v. 1342, Springer-Verlag, Berlin, 1988, p. 465. [47]

[93] Mindlin, G. B. and Gilmore, R., "Topological analysis and synthesis of chaotic time series," *Physica D*, **58**, n. 3, pp. 229–42, 1992. [44]

[94] Mindlin, G. B., Hou, X.-J., Solari, H. G., "Classification of strange attractors by integers," *Physical Review Letters*, **64**, n. 20, pp. 2350–3, 1990. [44]

[95] Mindlin, G. B., López-Ruiz, R., Solari, H. G., and Gilmore, R., "Horseshoe implications," *Physical Review E*, **48**, n. 6, pp. 4297–4304, 1993. [51]

[96] Mindlin, G. B., Solari, H. G., Natiello, M. A., Gilmore, R., and Hou, X.-J., "Topological analysis of chaotic time series data from the Belousov-Zhabotinskii reaction," *Journal of Nonlinear Science*, **1**, n. 2, pp. 147–74, 1991. [44]

[97] Molteno, T., *Chaos and crisis in strings*, PhD Thesis, Otago University, 1994. [37]

[98] Molteno, T. C. A., "Fast $O(N)$ box-counting algorithm for estimating dimensions," *Physical Review E*, **48**, n. 5, pp. 3263–6, 1993. [39]

[99] Molteno, T. C. A. and Tufillaro, N. B., "Torus doubling and chaotic string vibrations: experimental results," *Journal of Sound and Vibration*, **137**, n. 2, pp. 327–30, 1990. [37, 38, 39]

[100] Mullish, H., *Structured COBOL*, Harper & Row, New York, 1983. [54]

[101] Narasimha, R., "Non-linear vibration of an elastic string," *Journal of Sound and Vibration*, **8**, n. 1, pp. 134–146, 1968. [36]

[102] Nemhauser, George L. and Wolsey, Laurence A., *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988. [58]

[103] Neuman, C. P., "Recent development in discrete weighted residual methods," in Oden, J. T., ed., *Computational Methods in Nonlinear Mechanics*, The Texas Institute for Computational Mechanics, Austin, pp. 361–70, 1974. [73]

[104] O'Neill, B., *Elementary Differential Geometry*, Academic Press, New York, pp. 377–380, 1966. [21]

[105] O'Reilly, O., "Global bifurcations in the forced vibration of a damped string," *International Journal of Non-Linear Mechanics*, **28**, n. 3, pp. 337-51, 1993. [37]

[106] O'Reilly, O., *The chaotic vibration of a string*, PhD Thesis, Cornell University, 1990. [37]

[107] O'Reilly, O. and Holmes, P. J., "Non-linear, non-planar and non-periodic vibrations of a string," *Journal of Sound and Vibration*, **153**, n. 3, pp. 413–35, 1992. [37, 38]

[108] Palis, Jr., J. and de Melo, Welington, *Geometric Theory of Dynamical Systems*, Springer-Verlag, New York, 1982. [17]

[109] Pan, F. and Acrivos, A., *Journal of Fluid Mechanics*, **28**, n. 4, pp. 643–655, 1967. [18, 19, 26]

[110] Papoff, F., Fioretti, F., Arimondo, E., Mindlin, G. B., Solari, H., and R. Gilmore, "Structure of chaos in the laser with saturable absorber," *Physical Review Letters*, **68**, n. 8, pp. 1128–31, 1992. [44]

[111] Parker, T. S. and Chua, L. O., *Practical Numerical Algorithms for Chaotic Systems*, Springer-Verlag, New York, 1989. [41]

[112] Pecora, L. M. and Carroll, T. L., "Driving systems with chaotic signals," *Physical Review A*, **44**, n. 4, pp. 2374–83, 1991. [40, 52]

[113] Pecora, L. M. and Carroll, T. L., "Synchronization in chaotic systems," *Physical Review Letters*, **64**, n. 8, pp. 821–4, 1990. [40, 52]

[114] Politi, A. in *From Statistical Physics to Statistical Inference and Back*, Grassberger, P. and Nadal, J.-P., eds., Kluwer Academic, Boston, 1994. [37]

[115] Preisendorfer, R. W., *Principal Component Analysis in Meteorology and Oceanography*, Elsevier, 1988. [83]

[116] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., *Numerical Recipes in C*, Cambridge University Press, New York, 1988. [172]

[117] Reed, M. and Simon, B., *Functional Analysis*, Academic Press, Inc., Boston, 1980. [77]

[118] Rissanen, J. in *From Statistical Physics to Statistical Inference and Back*, Grassberger, P. and Nadal, J.-P., eds., Kluwer Academic, Boston, 1994. [37, 41]

[119] Roy, B., "An algorithm for a general constrained set covering problem," in Read, Ronald C., ed., *Graph Theory and Computing*, Academic Press, New York, 1972, pp. 267–283. [58]

[120] Schetz, J. A., "On the aproximate solution of viscous-flow problems," *Journal of Applied Mechanics*, **30E**, n. 2, 263–8, 1963. [72]

[121] Schmidt, S., and Ortoleva, P., "Electrical field effects on propagating BZ waves: predictions of an Oregonator and new pulse supporting models," *Journal of Chemical Physics*, **74**, pp. 4488–4500, 1981. [28]

[122] Schreiber, T., "Extremely simple nonlinear noise-reduction method," *Physical Review E*, **47**, n. 4, pp. 2401–4, 1993. [39]

[123] Schreiber, T., "Determination of the noise level of chaotic time series," *Physical Review E*, **48**, n. 1, pp. 13–6, 1993. [39]

[124] Seinfeld, John H., *Atmospheric chemistry and physics of air pollution*, John Wiley & Sons, Inc., New York, 1986. [123]

[125] Selten, F. M., "Toward an optimal description of atmospheric flow," *Journal of the Atmospheric Sciences*, **50**, 861–877 (1993). [83]

[126] Sirovich, L., and Everson, R., "Management and analysis of large scientific datasets," *The International Journal of Supercomputer Applications*, **6**, n. 1, 50–68, 1992. [83]

[127] Stoker, J. J., *Nonlinear Vibrations in Mechanical and Electrical Systems*, Interscience Publishers, New York, pp. 36–48, 1950. [23]

[128] Strikwerda, John C., *Finite Difference Schemes and Partial Differential Equations*, Wadsworth & Brooks/Cole Advanced Books and Software, Pacific Grove, CA, 1989. [75, 102, 103, 105, 142]

[129] Temperton, C., "On the FACR(l) algorithm for the discrete Poisson equation," Journal of Computational Physics, **34**, n. 3, pp. 314–329, 1980. [149]

[130] Tufillaro, N. B., "Nonlinear and chaotic string vibrations," *American Journal of Physics*, **57**, n. 5, pp. 408–14, 1989. [37]

[131] Tufillaro, N. B., *Chaotic themes from strings*, PhD Thesis, Bryn Mawr College, 1990. [37]

[132] Tufillaro, N. B. in *From Statistical Physics to Statistical Inference and Back*, Grassberger, P. and Nadal, J.-P., eds., Kluwer Academic, Boston, 1994. [37]

[133] Tufillaro, N. B., "Braid analysis of a bouncing ball," *Physical Review E*, **50**, n. 6, pp. 4509–22, 1994. [47, 50]

[134] Tufillaro, N. B., "Braid analysis of (low-dimensional) chaos," preprint, 1993. [37, 44]

[135] Tufillaro, N. B., Wyckoff, P., Brown, R., Schreiber, T., and Molteno, T., "Topological time series analysis of a string experiment and its synchronized model," *Physical Review E*, **51**, n. 1, pp. 164–174, 1995. [36]

[136] Tufillaro, N., Abbott, T., and Reilly, J., *An Experimental Approach to Nonlinear Dynamics and Chaos*, Addison-Wesley, Redwood City, CA, 1992. [37, 44]

[137] Tyson, J., *The Belousov-Zhabotinskii Reaction*, Springer-Verlag, New York, 1976. [28]

[138] Varga, R. S., *Matrix Iterative Analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1962. [149]

[139] Venimadhavan, G., Buzad, G., and Malone, M. F., "Effects of kinetics on residue curve maps for reactive distillation," *AIChE Journal*, **40**, n. 11, 1814–24, 1994. [12]

[140] Weiss. R. and Florsheim. B., *Physics of Fluids*, **8**. pp. 1631–1635. 1965. [18]

[141] Wheat. S. R.. Devine. K. D., and Maccabe. A. B.. "Experience with automatic. dynamic load balancing and adaptive finite element computation." in *Proceedings of the 27th Hawaii International Conference on System Sciences*. 1994. [68]

[142] Wheat. S. R.. Devine, K. D.. and Maccabe, A., B., "Experience with automatic. dynamic load balancing and adaptive finite element computation." Sandia National Laboratories Technical Report SAND93-2172A. in *Proceedings of the 27th Hawaii International Conference on System Sciences*. January 1994. [61]

[143] Wiggins. S.. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Springer-Verlag. New York. 1990. [30]

[144] Xu. X.. and Antal Jr.. M. J.. "Kinetics and mechanism of isobutene formation from T-butanol in hot liquid water." *AIChE Journal*. **40**. n. 9. 1524–34. 1994. [12]

[145] Zeng. X.. Eykholt. R., and Pielke. R. A., "Estimating the Lyapunov-exponent spectrum from short time series of low precision." *Physical Review Letters* **66**. n. 25. pp. 3229–32. 1991. [42]

[146] Zhabotinsky. A.. Zaikin. A.. Korzukhin. M.. and Kreitser. G.. "Mathematical model of autovibrational chemical reaction (oxidation of bromomalonic acid with bromate. catalyzed with cerium ions)." *Kinetics and Catalysis*. **12**. pp. 584–590. 1971. [28]

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ COPIES: (Archives) Aero Dewey Eng Hum

Lindgren Music Rotch (Science)

TITLE VARIES: ▶☐ _____

NAME VARIES: ▶☑ Scott

IMPRINT: (COPYRIGHT) _____

▶ COLLATION: 189 p

▶ ADD DEGREE: _____ ▶ DEPT.: _____

SUPERVISORS: _____

NOTES:

cat'r: date:

page: 544

▶ DEPT: Chem Eng

▶ YEAR: 1995 ▶ DEGREE: Ph.D.

▶ NAME: WYCKOFF, Peter S.