A Feasibility Study of the use of COLSYS in the Solution of Systems of Partial Differential Equations

by

Robert Malcolm Corless

Department of Applied Mathematics
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-82-32

September 1982

A thesis
presented to the University of Waterloo
in partial fulfillment of the
requirements for the degree of
M.Math
in
Applied Mathematics

Waterloo, Ontario, 1982

ABSTRACT

The feasibility of using an existing robust ordinary boundary value system solver, COLSYS, as the heart of a package for the solution of a class of partial differential equations is considered. Preliminary tests with Burgers' equation and a severe test with the one dimensional gas dynamics equations are reported. The main aim of this study is to see if we can achieve, with small development cost, a robust package with a sophisticated variable spatial mesh capability by co-opting the mesh selection algorithm of COLSYS. COLSYS was developed by U. Ascher, J. Christiansen, and R.D. Russell, of the University of British Columbia and Simon Fraser University.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

INTRODUCTION

The class of problem we are interested in here may be represented by the following system of partial differential equations:

$$(1.1) \qquad C(U)U_t + F(U,U_x) + (D(U)U_x)_x = G(U)$$

with initial boundary conditions on the strip $[a,b]X[0,\infty)$ given by

$$(1.2) \qquad B(\, U(a,t),\, U(b,t),\, t) = 0$$

and

$$(1.3) \qquad U(x,0) = \Phi(x)$$

where $U := U(x,t)$ is a p-dimensional vector, $\Phi(x)$ is a given function, $B(*,*,t)$ is a (generally nonlinear) vector function giving the boundary conditions, C and D are positive definite diagonal matrices, and where the matrices C and D and the vectors F and G may depend on x and t as well. This problem arises in many contexts, including heat conduction, chemical reactions, and population dynamics. For a more complete list, see [12]. The problem of original interest for the author was a large system arising out of the multi-

phase, multi-chemical-species fluid flow in a large duct. However, this thesis contains only a feasibility study of a method which could be used for this large problem, and contains no direct application to this particular problem.

In a system such as (1.1) where there may be components with widely differing time scales, or a moving subregion that is difficult to handle (such as a shock or contact discontinuity), it would seem particularly useful to have a dynamic variable spatial mesh capability, to minimize the unnecessary computational effort. In [20] it is pointed out that if a solution decays rapidly to a more slowly varying steady state solution, we would like to be able to reduce the size of the mesh, as well. With such a capability, the spatial mesh could adapt itself to concentrate on the areas of difficulty, which may move with time. Some work has been done on this problem in our context [12], but the subject of adaptive mesh selection is a young one, especially in the context of partial differential equations. On the other hand, in the context of ordinary boundary value problems, the subject is well developed [29,30], as is the related software. Thus it is natural to ask if we may use some of the ideas and techniques from ordinary boundary value problems in our context. This thesis attempts to answer this in the affirmative; by using an ordinary boundary value problem solver with adaptive mesh selection capability as the heart of a package to solve partial differential equations, we can

co-opt the algorithms used in a very simple way. Whether this procedure is effective or not is the subject of this thesis.

In order to allow the use of ordinary boundary value problem solvers in the solution of (1.1), we replace the system of partial differential equations with a sequence of systems of ordinary boundary value problems. This is done by first choosing a temporal mesh, and then replacing $U_t(x,t)$ at a fixed time level $t = t^*$ by a linear combination $L(U(x,t^*),x,t^*)$ of $U(x,t^*)$ and U at previous times. If we adopt the notation $\underline{u}(x) := U(x,t^*)$, the system (1.1) becomes, at time t^*,

$$CL(\underline{u},x,t^*) + F(\underline{u},\underline{u}_x) + (D\underline{u}_x)_x = G \tag{1.4}$$

and the boundary conditions become

$$B^*(\underline{u}(a),\underline{u}(b)) = 0 \tag{1.4a}$$

There are many existing packages for solving systems of ordinary boundary value problems [10], and each has its special points. The best choice of boundary value problem solver is likely dependent on the particular problem (1.1) one is interested in solving; certain codes will be better for some purposes than others. Our choice for the purposes of this investigation was COLSYS, a collocation code developed by Uri Ascher of the University of British Columbia and Jan Christiansen and Robert D. Russell of Simon Fraser Universi-

ty [2, 3, 4, 5, 29]. COLSYS is a robust general purpose boundary value problem solver, has the desired adaptive mesh capability, and has the added advantage to the author of familiarity.

The research done in this thesis consists of experimental computations with a computer implementation, DYNACOL, of the above process. Two major experiments are reported here; the first experiment uses Burgers' equation, while the second uses the compressible gas dynamics equations. Chapter 2 consists of a more detailed look at the algorithm we will use. Chapter 3 reports on the Burgers' equation experiments. Chapter 4 is a description of COLSYS, preparatory to the full scale experiments. Chapter 5 is the report on the gas dynamics equations, and Chapter 6 contains the conclusions of the thesis, and suggestions for further study.

The key questions that will have to be addressed here are;

1. Is the resulting algorithm for the solution of (1.1) robust? That is, will it solve difficult problems?
2. Is the method simple to implement? We do not expect that this method of solving (1.1) to be the most efficient in terms of running time; we are asking whether a robust package can be obtained quickly, for a given problem.
3. What are the economics of the method? Although we do not expect this method to be sufficiently inexpensive

to be used in ongoing production runs, we also do not expect its use to be prohibitively expensive. However, this needs to be demonstrated, as ordinary boundary value problems are not cheap to solve; the cost per step in this process is going to be high. Since we expect that the primary uses of this algorithm will be for checking custom coded production programs and for use while such programs are being developed, we must also ask if it is efficient enough for these purposes.

With all these points in mind, we will begin.

Chapter II

DESCRIPTION OF ALGORITHM

2.1 DISCRETIZATION OF PROBLEM

For reasons of simplicity, we will choose a uniform temporal mesh. This seems reasonable for an experiment of this type, but it will become apparent later that an adaptive temporal mesh would have its advantages. After the time step size is chosen, we replace the time derivative $U_t(x,t)$ with a linear combination of $U(x,t)$ at previous times. On the reasonable assumption that the problem of interest will be stiff in time, the Gear stiff equation discretizations for ordinary initial value problems [17], the backward difference formulae, seem to be good candidates for the discretization to be chosen. These, in the form we shall use, can be written as

$$\frac{dy}{dt} \cong \frac{y(t) - \sum_{i=1}^{k} a_i y(t - i*h)}{\beta_0 h} \tag{2.1}$$

where h is the step size in time.

This transforms the system of partial differential equations (1.1) plus the boundary conditions (1.2) into the following sequence of boundary value problems on the closed interval [a,b], with associated boundary conditions:

$$(2.2)\quad \frac{C\left\{ U(x,t) - \sum_{i=1}^{k} a_i U(x,t-i*h) \right\}}{\beta_0 h} - F(U,U_x) - (DU_x)_x = G$$

$$(2.3)\quad B(U(a,t),U(b,t),t) = 0$$

which, if the k functions U(x,t-i*h) are known, and if the (diagonal) matrix D is nonsingular, is an ordinary system of two-point boundary value problems for U(x,t) at a fixed time t=N*h.

So we must now solve the sequence of boundary value problems on the chosen temporal mesh, that is, the lines t=N*h, N=1,2,... This problem is naturally divided into two parts:

1. Getting an approximation to the solution on the first k time levels, i.e. U(x,h), ..., U(x,k*h) (We could use the values of U(x,0), U(x,h), ..., U(x,(k-1)h), but this turns out to be inconvenient for programming).
2. Solving the sequence of boundary value problems for U(x,N*h), for N=k+1,....

Following the traditional procedure, we shall ignore the start-up difficulties for the moment and concentrate on the solution of the boundary value problems.

2.2 SOLUTION OF THE BOUNDARY VALUE PROBLEMS

Now we have a sequence of systems of ordinary boundary value problems. There are several packages for solving such systems; for a relatively complete survey, see [13,14,34]. The package chosen for the purposes of this thesis is COLSYS, as mentioned previously. This code, in addition to being robust and of competitive efficiency, has the added advantage to the author of familiarity. It is expected that the general conclusions drawn in this thesis will be valid for the reader's favourite BVP system solver as well. To familiarize himself with COLSYS, the reader may wish to skim the chapter on COLSYS before proceeding.

Let us introduce some more compact notation:

let

$$v^{(n)} := v^{(n)}(x) = \frac{\sum_{i=1}^{k} \alpha_i U(x,(n-i)h)}{\beta_0 h}$$

$$u^{(n)} := u^{(n)}(x) = U(x,n*h).$$

In order to use COLSYS, we must rewrite the boundary value problems into a standard form, which is

$$u_{xx}(x) = P(x, u(x), u_x(x)).$$

For our problem, on rearranging (1.1) we have

$$(2.4)\quad D(u^{(n)})u_{xx}^{(n)} = F(u^{(n)},u_x^{(n)}) - D_x(u^{(n)})u_x^{(n)} + \frac{C(u^{(n)})u^{(n)}}{\beta_0 h} - C(u^{(n)})v^{(n)} - G(u^{(n)})$$

$$= H(u^{(n)})$$

or

$$(2.5)\quad u_{xx}^{(n)} = D(u^{(n)})^{-1}H(u^{(n)}) = F_1(t;x,u^{(n)},u_x^{(n)}) + D^{-1}(\frac{u^{(n)}}{h} - v^{(n)})$$

In the program DYNACOL, the problem dependent routine FUN calculates $F_1(t;\ x,\ u^{(n)},u_x^{(n)})$. For both the problems reported here, C := I, the identity, and D := ηI, where η is a constant. In general, if D (which is diagonal) is singular, it means that some of the components are determined by first order differential equations or even algebraic equations. This presents no difficulty for COLSYS, except in exceptional circumstances, but it does make for notational complexity, so this case will not be considered until necessary.

Before we actually get started on solving the boundary value problems, a little discussion of the representation of such a solution is in order. Since we want variable spatial meshes, we need to evaluate U(x,t) at any x in the interval

in question. This means some interpolation or approximation is necessary. The representation of the solution used by COLSYS is a double precision vector containing the mesh used to approximate u and the B-spline coefficients. It is natural to take full advantage of this representation and the COLSYS facility APPSLN (for APProximate SoLutioN) for evaluating the B-spline representation at any desired x-value. Thus, for our program DYNACOL the solution U(x,t) will, for a given time t, consist of a spatial mesh, a set of B-spline coefficients, and the accessory parameters (the order of the B-spline, etc).

Another consideration is that this approach to the solution of the partial differential equations requires that we be able to evaluate, at a given time t (larger than k*h), the linear combination of the solutions at previous times for any given x in the interval of interest. Therefore we will have to keep, in the routine operation of the program, the last k B-spline representations of the solution. This collection of representations is called the Past History, and is stored in the two-dimensional array PSTHIS. Now, it is clear that we need to efficiently evaluate the linear combination

$$v^{(n)} = \frac{\sum_{i=1}^{k} a_i U(x,(n-i)h)}{\beta_0 h}$$

for any x in the interval [a,b], and we need to look at this problem closely.

It is possible to simply use the B-spline representation here, by simply using APPSLN to evaluate each of $U(x,(n-i)h)$ and then calculate $v^{(n)}(x)$, but we then run into the problem of passing parameters through existing code. This is awkward, and requires extensive use of common blocks. However, this sort of problem is nearly always encountered whenever development of code uses other packages in perhaps ways they were not meant to be used. In any case, using the B-spline representation may not always be what we want to do anyway, as it will be seen to be more economical to use some other form of evaluation, such as an approximating cubic spline, calculated once only for each time step, rather than doing k B-spline evaluations on every call during a time step. We will look more closely at this in the context of the gas dynamics problem which we will investigate later. For the first problem, Burgers' equation, we have included the option of looking at a cubic spline interpolant of this linear combination, and a linear interpolant of the linear combination. It would, in the light of the experimental results, be instructive to also look at taut splines [7], or splines in tension, as well, but this is not central to the thesis.

The method chosen to evaluate $v^{(n)}$ is essentially

1. Find a good mesh for evaluating $v^{(n)}$, namely a merging of the meshes used to evaluate each of the $u^{(n-i)}$.

2. Use some reasonable method to approximate $v^{(n)}$ at a general point x, other than a mesh point. Since COLSYS exhibits superconvergence (the phenomenon of higher order accuracy at the mesh points) interpolation seems as good as any method. An ordinary cubic spline has been found to be satisfactory in many cases, although not in all.

Notice that as we iterate on n, the Past History needs to be updated, and the cubic (or linear) spline of $v^{(n)}$ needs to be recomputed. This is the function of the routine UPDATE (see the appendix on the details of the program).

We are now ready to present the outline of the algorithm for solving the sequence of boundary value problems. This is best presented in the form of a flow chart, presented in Figure 1.

ADVANCE in the flowchart is a euphemism for two routines Fastad and Cautad (for fast-advance and cautious-advance). The fast advance uses (either cubic or linear) interpolation of $v^{(n)}(x)$, and cautious advance uses the direct b-spline method of evaluating $v^{(n)}(x)$. ($v^{(n)}(x)$ is evaluated directly from the representations of the past histories.)

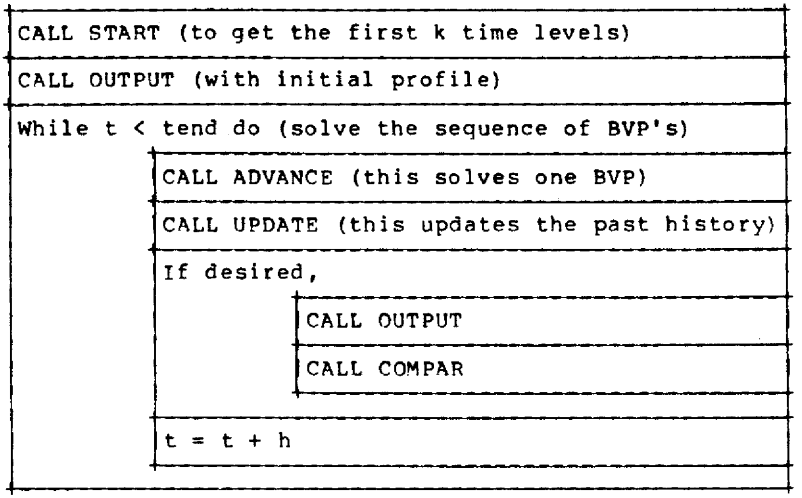


Figure 1: Flow Chart for DYNACOL

2.3 STARTING UP

There were two methods used for start-up. The first was essentially for testing purposes, and amounted to using the exact values for U(x,0), U(x,h), ..., U(x,k*h) to test the feasibility of higher values of k (the order of the time discretization). The second method of starting up is more general, though still not a "production" subroutine. What it does is start up slowly with a first order method, using the time step h/n_1 (where n_1 is some positive integer - currently n_1 is 16), until it can start using a second order method with time step h/4, and a third order method with time step h/2, etc. One point to note is that the "order"

of the time discretization is not of any great use in this analysis, so that considerations of the start-up on this basis are not relevant. Thus we can ignore the fact that this start-up procedure makes the global error of "order" h. For details of the start-up, which is a simple routine, see the appendix on documentation of DYNACOL.

2.4 STORAGE REQUIREMENTS

Since we are using interpolation, we require more space than if we were using stationary finite differences on the same size fixed mesh. Indeed, since the mesh selection strategy cannot be perfect for all problems, we must "budget" for a little wastage of space. However, for shock problems we expect that, in view of the global fineness of a fixed spatial mesh that is necessary for finite differences, the storage requirements of this method will be "asymptotically" better in some sense. For example, if the problem under consideration has a moving internal boundary layer, or shock, of width O(e), and we require N points in the layer to satisfactorily resolve it, the fixed mesh on a unit interval would have to have O(N/e) points (since each interval of width e, of which there are O(1/e), would have to have N points), while the dynamically variable mesh only needs O(N) over the entire interval. If e is small, this can prove very important. On the other hand, if the boundary layer width is not too small, the variable mesh is wasted computation, as far as storage is concerned.

Of course, one of the more desirable features that a method for solving partial differential equations could have would be a "large" time step h: in many methods there is an undesirable bound on the time step of the form $h < K H^p$, where H is the spatial mesh size. With a variable spatial mesh, it is unclear what new limitations on the time step there will be, but there is the hope that h can be larger than it could be with a fixed spatial mesh. This would hopefully mean a savings in computer time to reach the same t value. See, however, the conclusions drawn in chapter 6.

Another kind of problem where the adaptive spatial mesh would be of use was pointed out in [20], namely a problem in which the solution tended to a relatively easy to resolve steady state. This would allow mesh points to be thrown away, with a corresponding savings in time. However, the mesh selection algorithm of COLSYS does not throw away points in a given time step - the only place it allows reductions of the spatial mesh is at the beginning of a time-step, when it automatically throws away half the old mesh. This can be modified, as COLSYS allows user modification of the mesh. So, in between time steps, we could for example throw away two thirds of the mesh if the solution looked as if it were settling down. It is of course a difficult problem to automatically decide what to do here, and in fact what we wanted to examine in this thesis is whether we could gain a sophisticated automatic mesh handling facil-

ity with little development cost by using the existing capability of COLSYS.

As will be seen, the storage requirements of DYNACOL are linear in NMAX (once the problem and various auxilary parameters have been specified), where NMAX is the maximum number of subintervals desired in the solution. The exact storage requirements, and the dimensions of the (double precision) arrays are tedious to compute by hand, so a small "pre-DYNACOL" program is useful (see appendix C). The sample output below is for the gas dynamics shock tube problem, with NMAX = 100. As you can see, the program requires about 545K bytes of core, which is a relatively large amount of storage for a one-dimensional problem, but not a lot for a modern mainframe. Of course, on modern computers space is not usually a worry unless you are dealing with partial differential equations of more than one space dimension. Here the method is restricted to partial differential equations of one space dimension (although see the section on generalizations in chapter 6).

```
NCOMP =  3 KCOL = 3 NREC =  3 NMAX = 100
THE ORDERS OF THE DIFF EQS ARE
2 2 2
NSIZEF =   277 AND NSIZEI =    15
THE ARRAY SIZE OF FSPACE SHOULD BE    27787
THE ARRAY SIZE OF ISPACE SHOULD BE     1503
THE MAXIMUM SIZE OF THE MERGE MESH      300
THE MAXIMUM SIZE OF THE FUN VECTOR      900
THE SPLINE COEFFICIENT ARRAY  SIZE     2700
THE MAXIMUM ARRAY  SIZE  OF PSTHIS     1979
THE SIZE OF THE INTEG. ARRAY IPAST       10
THE TOTAL SIZE OF THE RL ARRAYS IS    67390
AND THE TOTAL STORAGE IS APPROXTLY      545 K BYTES
```

Figure 2: Sample Output from the Sizes Program

Chapter III

DESIGN OF INITIAL EXPERIMENT

3.1 DESCRIPTION OF PROBLEM

The initial experiments with DYNACOL were carried out using Burgers' equation, with simple initial conditions:

$$U_t + \left(\frac{U^2}{2}\right)_x = \nu U_{xx} \tag{3.1}$$

with initial and boundary conditions given by

$$U(-\infty,t) = 1;\ U(\infty,t) = 0;\ U(x,0) = \begin{cases} 1.0, & \text{if } x < 0.1 \\ 0.0, & \text{if } x > 0.1 \end{cases} \tag{3.2}$$

This equation has the analytical solution [1,36] given by the following formulae:

$$\text{let } A(x,t) = \exp\left(\frac{(x-t/2-1/10)}{2\nu}\right)$$

$$B(x,t) = \text{erfc}\left[\frac{(1/10-x)}{2(\nu t)^{(1/2)}}\right]$$

$$C(x,t) = \text{erfc}\left[\frac{(x-t-1/10)}{2(\nu t)^{(1/2)}}\right]$$

then the solution to the partial differential equation (3.1) is given by

$$(3.3)\quad U_e(x,t) = \frac{C(x,t)}{C(x,t) + A(x,t)B(x,t)}$$

The investigations were carried out in two parts:

1. With Pseudo-Start and discontinuous initial conditions; that is

$$(3.4)\qquad u(x,0) = U_e(x,0)$$
$$u(0,t) = U_e(0,t)$$
$$u(4,t) = U_e(4,t).$$

This was intended to test the relative effectiveness of the higher order discretizations.

2. With the general Start, and continuous initial conditions; that is

$$(3.5)\qquad u(x,0) = U_e(x,1)$$
$$u(0,t) = U_e(0,t+1)$$
$$u(4,t) = U_e(4,t+1).$$

This was intended to test the effect of the start-up algorithm on the performance of the program.

3.2 COLSYS FORM FOR BURGERS' EQUATION

For this one-component problem, the rewriting of the partial differential equation (3.1) into COLSYS form is extremely simple:

$$(3.6)\quad u_{xx}^{(n)}(x) = \frac{u^{(n)}(x)\,u_x^{(n)}(x) + w^{(n)}(x)}{\nu}$$

where

$$w^{(n)}(x) = \frac{u^{(n)}(x) - \sum_{i=1}^{k} a_i u^{(n-i)}(x)}{\beta_0 h}$$

is the linear combination of the Past History which approximates the time derivative.

3.3 MAIN IDEAS

We felt that useful information could be obtained about DYNACOL (in a reasonably short time) by looking for the extremes of the time step h; that is, for a given set of other switches, find values of h satisfying the following:

h_{low} = a value of h for which the program produces acceptable output (i.e. the solution is reasonably accurate by maximum error and visual standards).

h_{high} = a value of h for which either of the following two conditions hold: the program produces unacceptable output for $h = h_{high}$, or fails to produce output at all (i.e. the Newton's iteration fails to converge, or COLSYS fails for some other reason).

Thus the initial experiments were designed mainly to see the relative efficiencies of the various orders of the method. By looking, in essence, for the largest value of h that would work, we are hopefully looking for the cheapest h for a given method. On this basis we compare the different orders of the method.

3.4 EXPERIMENTS WITH PSEUDO-START

In designing these experiments, selection had to be made among the many switches (that is, performance control parameters) to maximize information obtained for effort expended. A list of the relevant switches follows, with comments indicating their believed relative importance. There are more switches than are listed here; the remainder will be found in the appendix describing COLSYS.

1. Parameters to COLSYS.

   a) Spatial tolerances on the solution $u^{(n)}(x)$ and $u_x^{(n)}(x)$.

   Given a set of tolerances $\{T_i\}_{i=1}^{NTOL}$ and a set of pointers $\{J_i\}_{i=1}^{NTOL}$, where J_i is the index of the component of $z(u)$ we wish the i^{th} tolerance applied to, COLSYS attempts to satisfy the following inequalities.

(3.7) $$||z(u)_{J_i}(x) - z(v)_{J_i}(x)|| \leq T_i(1+||z(v)_{J_i}(x)||)$$

   where $z(u)_j$ and $z(v)_j$ are the j^{th} components of the exact solution vector and the computed solution vector, respectively. Thus, these parameters have a direct effect on the accuracy and the efficiency of a single step, i.e. the solution of a single BVP. In general, the smaller or tighter the error tolerance, the better the accuracy, at the price of longer CPU time.

b) The number of collocation points per subinterval k_c, and the maximum number of subintervals allowed.

Both these numbers have a bearing on the storage requirements of the program, and a less direct effect on the accuracy. The number of collocation points is also a factor in the degree of the B-spline basis for the solution of the problem. For stability reasons one would like this as low as possible, but for accuracy one prefers it higher. For our problems, we will use only the minimum value of k_c.

c) Initial mesh for the first solution $u^{(1)}(x)$.

Unless otherwise specified, we will choose this to be a uniform mesh of a few intervals. This switch can be important for hard-to-start problems, and it is easy to find examples of problems where a good initial mesh is essential to an efficient run, or indeed a sucessful run.

d) Sensitivity switch.

One sets a parameter to COLSYS, designating a problem as either "sensitive" or "regular". This makes a difference in the mode of COLSYS' Newton iterations, which can be either cautious or fast, corresponding to whether or not the Jacobian of the Newton iteration is updated often (cautious

mode), with careful control of the relaxation factor, or updated not so often, with more emphasis on efficiency than sensitivity. It is not likely that this is an important parameter.

e) Other COLSYS switches (see appendix).

2. Parameters in the calling program.

a) The time step size, h.

This is perhaps the most important parameter (together with the number of stages used, k). If h is too large, the program may fail completely, while if h is too small, we are paying too much for the solution.

b) Number of stages in the method, k.

This is the order of the approximation to the time derivative $U_t(x)$. The relation between this and the time step h is a complicated one. It would seem that the higher the order, k, the larger the time step we could take. However, this is not so, due perhaps to instabilities in the particular approximations to the time derivative that we are taking (the Gear stiff approximations, as mentioned previously). This will be discussed in a later section. So we would in fact expect an optimal value of k for a given problem, and the initial experiment seems to bear this out.

3. Problem dependent parameters.

a) Diffusion Constant in Burgers' equation.

b) Initial conditions.

There are many initial conditions that produce phenomena of interest. Initial discontinuities are perhaps the most interesting, but breaking wave phenomena [1,20] are also of interest.

Out of all these switches, we have made some definite choices for our initial experiments with the Pseudo-Start routine. Our problem is (3.6) together with initial and boundary conditions (3.4), with the diffusion constant chosen to be =0.02. The remaining switches were chosen as follows.

1. Parameters to COLSYS.

a) Spatial tolerances on the solution $u^{(n)}(x)$ and $u_x^{(n)}(x)$.

Two sets of values for the two tolerances were chosen here:

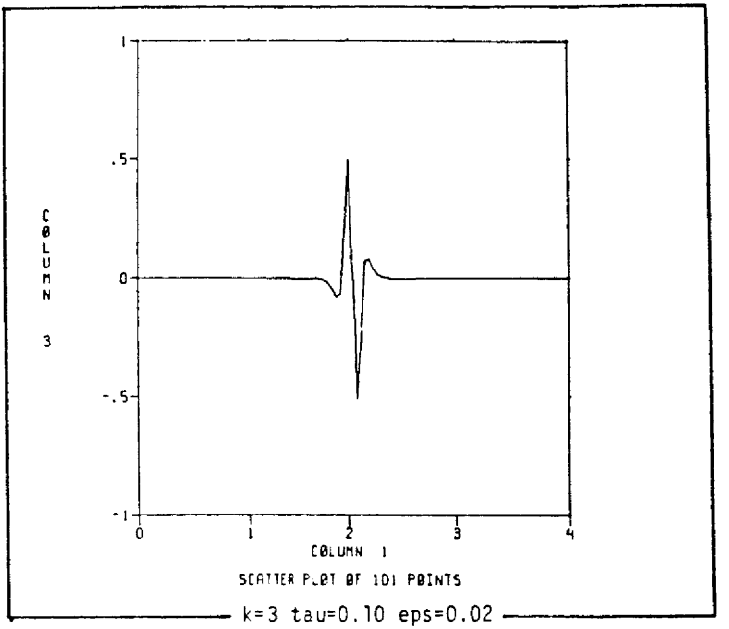
i) Loose tolerances: $T_1 = 10^{-3}$, the tolerance on u(x); $T_2=10^{-1}$, the tolerance on $u_x(x)$.

ii) Tight tolerances: $T_1 = 10^{-5}$, $T_2 = 10^{-2}$.

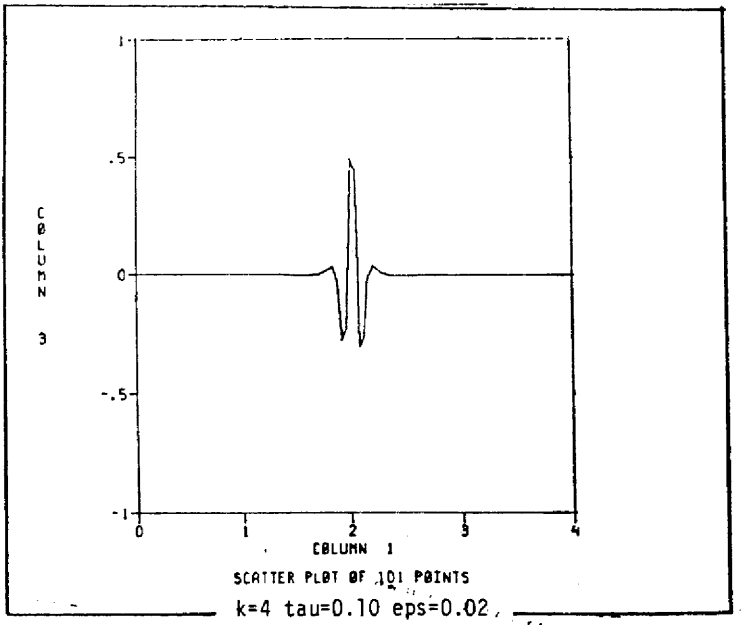
b) The number of collocation points per subinterval was chosen to be $k_c=3$.

c) Initial mesh for the first solution $u^{(1)}(x)$.

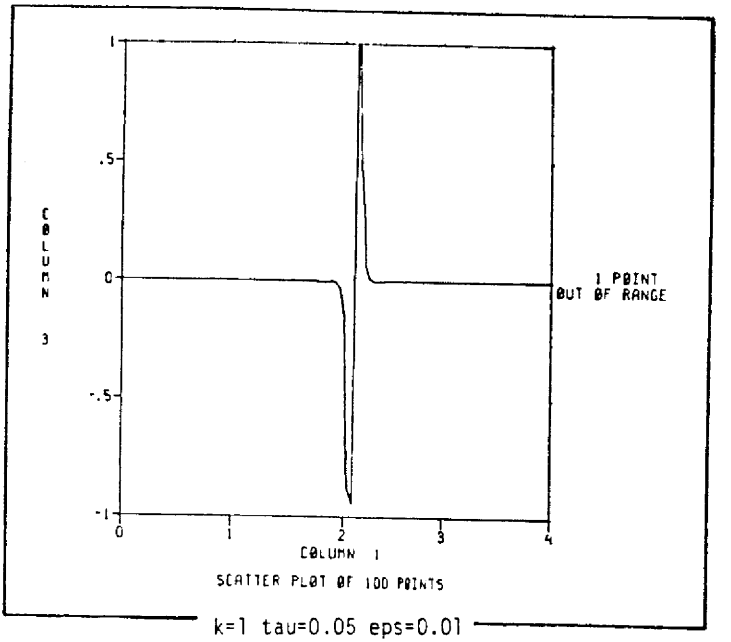
We will choose this to be a uniform mesh of 5 intervals. This is a very crude choice of an initial mesh. For this problem, however, COLSYS ov-



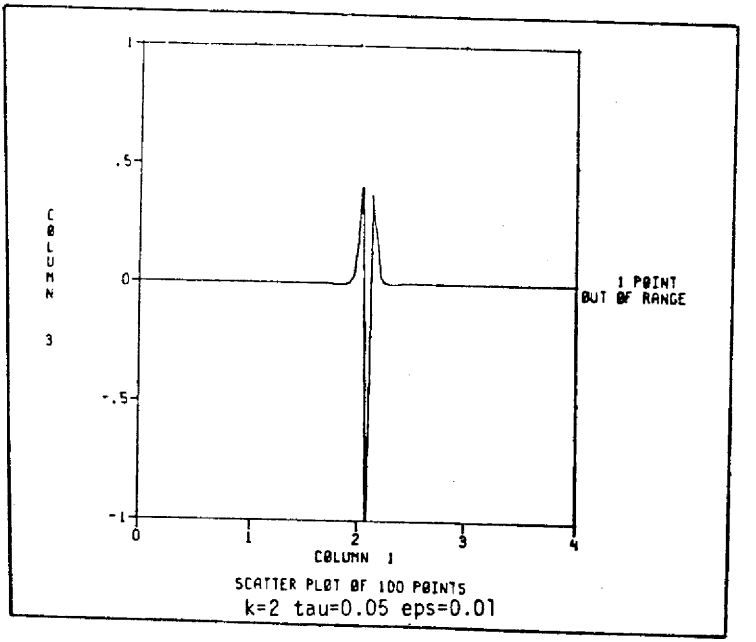
k=3 tau=0.10 eps=0.02



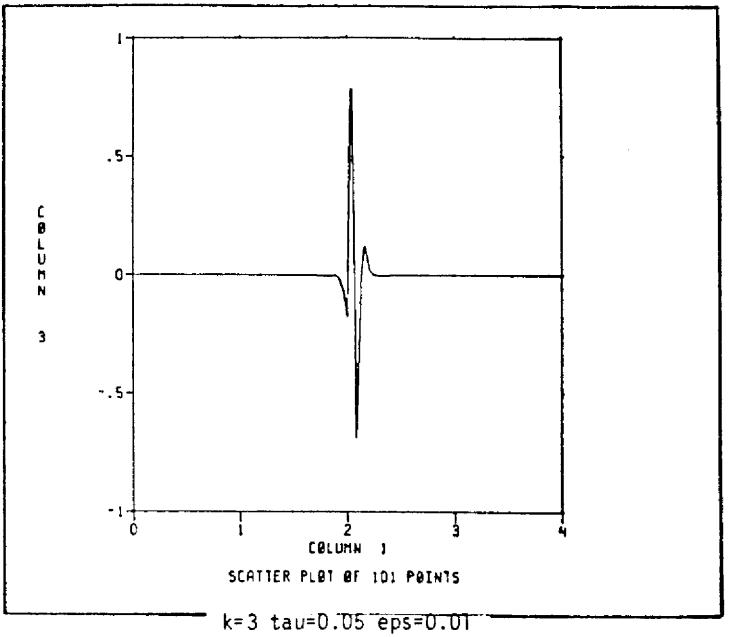
k=4 tau=0.10 eps=0.02



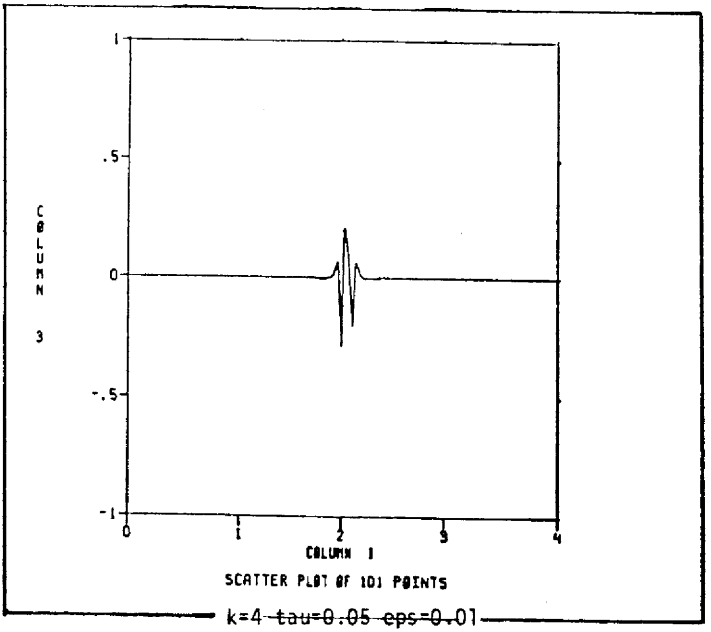
k=1 tau=0.05 eps=0.01



k=2 tau=0.05 eps=0.01



k=3 tau=0.05 eps=0.01



k=4 tau=0.05 eps=0.01

erate h), and likewise for higher orders. This is a problem, and restricts the use of higher order methods.

At this point, we see the need to consider other methods of time discretization. The lower order stiffly stable methods of Hermite type [26] are possibilities, and another is to use the box scheme, or the Crank-Nicolson method [21]. However, these are full scale projects, and we still have more information that we can glean from the Gear backward difference approach.

3.14 CONCLUSIONS DRAWN FROM INITIAL EXPERIMENTS

These, the preliminary tests of DYNACOL, were moderately successful. The expense of computing the solution to Burgers' equation was quite high, as was expected. However, the code behaved as expected, and the spatial mesh showed good movement with changing time (see graphs). This bodes quite well for the more difficult problem to be investigated later, the gas dynamics equations of a shock tube.

Chapter IV

DESCRIPTION OF COLSYS

4.1 MOTIVATION

Now that the initial experiment has been looked at, and we are able to ask more sophisticated questions about COLSYS, it is profitable to look at the details of its operation. In the last chapter, we looked mostly at the error introduced by the time discretization, tacitly assuming that the spatial error would be minimal. In order to look more closely at this assumption, we will need a working knowledge of COLSYS.

4.2 THE METHOD OF COLLOCATION.

For an elementary introduction to spline collocation, see [28], or [7]. Both of these are good introductions to the theory of spline collocation in the context of solutions of ordinary boundary value problems; while [9] contains information more pertinent to the use of spline collocation for systems of ordinary boundary value problems. Here we will give only a brief introduction, and no theory or error estimates at all.

Collocation is the process of finding an approximate solution to an operator equation by requiring that an ap-

proximate solution satisfy the operator equation at some discrete set of points, as well as satisfying the boundary conditions. We will illustrate collocation for the case of a one-component linear problem, and generalize to multi-component, nonlinear problems later. Consider the ordinary boundary value problem

$$(4.1)\quad L[u] := u^{(2)} - f_1 u^{(1)} - f_2 u = G(x) \quad 0 \leq x \leq 1$$

with the boundary conditions

$$(4.2)\quad \begin{aligned} c_1 u(0) + d_1 u^{(1)}(0) &= e_1 \\ c_2 u(1) + d_2 u^{(1)}(1) &= e_2 \end{aligned}$$

To describe the approximate solution, we introduce some subspace S of $C^{(1)}[0,1]$, with basis $\Phi_j(x)$, j=1, ... Thus we will approximate the solution to (4.1) with a function v(x), where v(x) is of the form

$$(4.3)\qquad v(x) := \sum_{j=1-p}^{N^*} a^{(j)} \Phi_j(x)$$

We will determine our approximate solution by requiring that it satisfy the differential equation exactly at some set of points (s_j), that is,

$$(4.4)\quad L[v](s_j) = G(s_j)$$

ercame this difficulty easily. It is in fact not a true test of COLSYS' mesh selection algorithm, as the exact solution is provided by Pseudo-Start for the first k values.

d) Sensitivity switch.

The problem is designated "regular".

e) Other COLSYS switches are taken to be their default values.

2. Parameters in the calling program.

a) The time step size, h.

This is what the experiment is designed to find. We will be looking for h_{low} and h_{high} as mentioned previously.

b) Number of stages in the method, k.

These were chosen to be k=2, k=3, and k=4 for comparison purposes.

3. Problem dependent parameters.

a) Diffusion Constant in Burgers' equation.

ν=0.02, as mentioned previously.

b) Initial conditions.

The initial conditions are (3.4), the discontinuous initial conditions.

3.5 RESULTS OF EXPERIMENT WITH PSEUDO-START

For this run, ν=0.02, T_1=10^{-3}, T_2=10^{-1}, tend=5.0

TABLE 1

Loose Tolerance Runs

	k=2	k=3	k=4
h_{low}	0.10	0.125	0.10
max. error	0.28	0.28	0.46
total cpu (s)	33.9	33.4	47.1
time/step (s)	0.66	0.81	0.92
h_{high}	0.15	0.15	0.15
max. error	0.40	0.44	0.70
total cpu (s)	11.6	29.8	44.0
time/step (s)	0.34	0.88	1.29

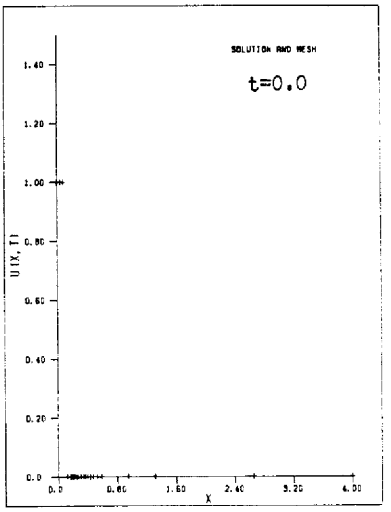
For these runs, ν=0.02, $T_1=10^{-5}$, $T_2=10^{-2}$, tend=5.0.

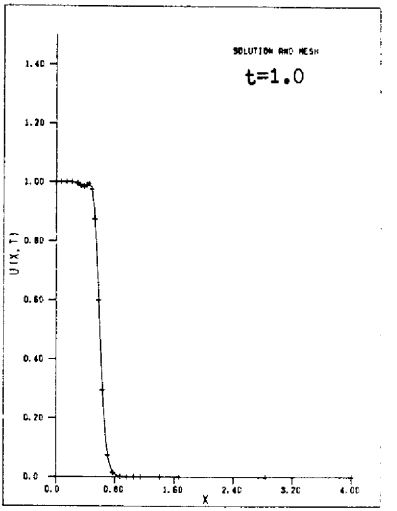
TABLE 2

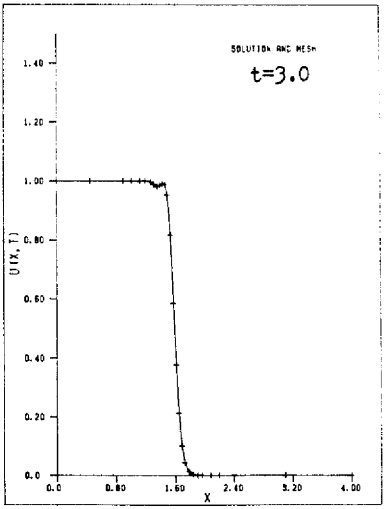
Tight Tolerance Runs

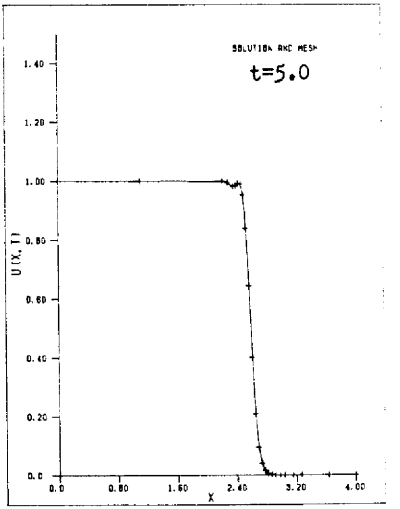
	k=2	k=3	k=4
h_{low}	0.05	0.125	0.10
max. error	1.7(-2)	3.6(-2)	2.2(-2)
total cpu (s)	151.6	78.9	127.8
time/step (s)	1.50	1.92	2.51
h_{high}	0.10	0.15	0.15
max. error	-	-	-
total cpu (s)	8.30	2.55	17.0
time/step (s)	-	-	-

3.6 SOLUTION PROFILES WITH PSEUDO-START

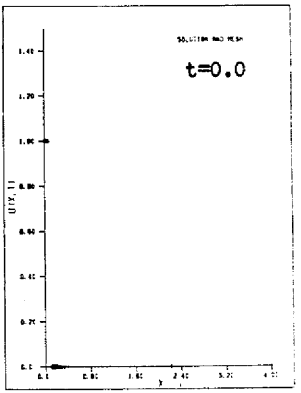








$k = 3$, $h = 0.10$, $T_1 = 0.0001$ $T_2 = 0.01$

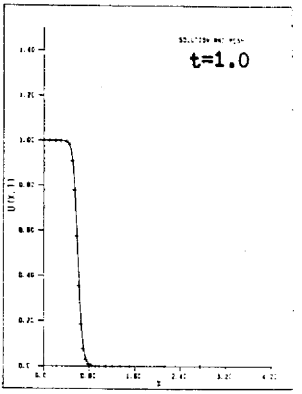


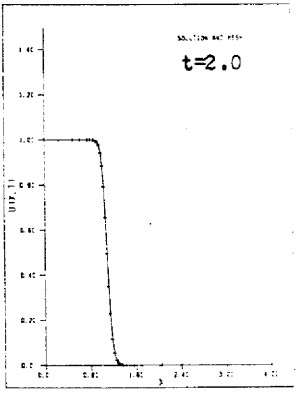
$k = 3$

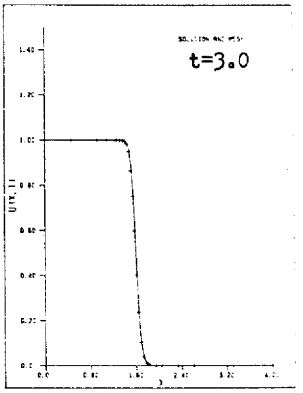
$h = 0.10$

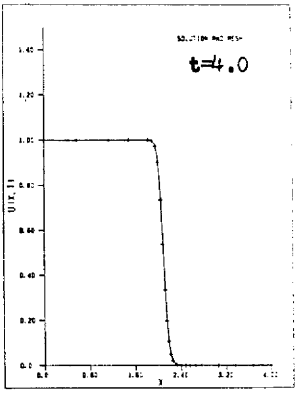
$T_1 = 10^{-5}$

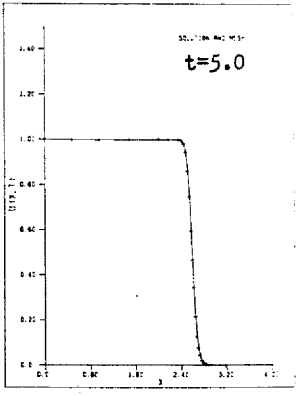
$T_2 = 10^{-3}$











3.7 DISCUSSION

There are many things to consider in the evaluation of the performance of the program. As these are initial experiments, we will discuss only two of the most important criteria here: accuracy, in a sense to be defined below, and efficiency, again in a specific sense defined below.

Out of the many possible criteria for accuracy, we have decided to use two: the maximum error, and a visual, qualitative assessment of the results. The maximum error, if small, will guarantee a good visual result, but sometimes even a reasonably small maximum error did not reflect the presence of oscillations, which made the solution unacceptable. Other criteria were measured by the program, including reporting on the shock centre position, the shock speed, and shock width. However, DYNACOL propagated the wave at the correct speed, so these reportings did not distinguish between the runs of DYNACOL at different orders, which was the purpose of this experiment. Even the maximum error was not as good a criterion for this purpose as was the visual inspection of the solution profiles. In a few cases, runs with comparable maximum errors would have quite different profiles. Oscillations were definitely a problem here, although with the averaging techniques now extant, [19], this may not be as serious as previously thought.

Due to the complicated relationship of the time step size h and the order k to the overall cost of a run, only

the simplest efficiency criteria are of value here. We will compare runs on the basis of overall cost in cpu seconds, and the average time per step taken by COLSYS (although the behaviour of COLSYS as the problem time increases is of interest). Using these criteria, it is evident that the higher order methods are of some value, as is shown in the following table.

	k=2	k=3	k=4
total solution time	151.6	78.9	127.8

Thus we can see that the method is more efficient for k=3 than for either k=2 or k=4. This is expected, and will be pursued in the case of general start. On the IBM 4341, 30 seconds of cpu time is about $1.00, on the slow batch machine.

During the course of these experiments, we noted that the form of failure or unacceptability for the runs with a higher spatial tolerance was a hard failure; that is, COLSYS would fail to find a solution, usually on the first step. This failure was due usually to a failure of Newton's method to converge on the mesh provided by the COLSYS mesh selection algorithm. Whether this represents a failure of Newton's method or a failure of the mesh-selection algorithm is irrelevant, because both need a sufficiently good initial guess to find a solution and mesh for the current time step. As DYNACOL uses the solution and mesh at the previous time

as the initial guess for the current solution and mesh, this means that the time step h must be sufficiently small. However, exactly what constitutes sufficiently small is a very difficult question to answer in this context.

What is especially not clear about this limitation is the effect on the efficiency, the time per step, near the failure point, the largest value of h. Since Newton's method converges quickly when it converges at all, one would expect only a slight increase in cost before failure. However, if the principal source of failure is the mesh selection algorithm, there is likely a considerably greater variability in the time step. The experimental evidence gathered by the author, in the course of these experiments, suggests that the first possibility is in fact the case; there is only a slight increase in cost. However, the evidence is only suggestive and by no means conclusive.

Another interesting question about efficiency is raised by the notion of appropriate spatial tolerances. By loosening the spatial tolerances, we may make large gains in total solution time (compare the time per step in Table 1 versus the time per step in Table 2). However, we obviously pay a price in accuracy. Choosing an optimal value for a spatial tolerance is not easy. Graph (4) represents an intermediate set of spatial tolerances, $T_1=10^{-4}$, $T_2=10^{-2}$, and we see that the solution is almost satisfactory, except for the stable, nonconservative blip just behind the shock. If we

tighten the tolerances to 10^{-5} and 10^{-3}, the phenomenon disappears. Thus these tolerances are quite important, and unfortunately we have only a "rule of thumb" for dealing with them: set them small (tight) enough so that the expected error behaviour is dominated by the error made in the time discretization. This rule is not very helpful, but it does point out the general behaviour of the error tolerances "in the limit".

3.8 REMARKS ON THE MESH SELECTION

The first experiments with DYNACOL are very encouraging, as far as mesh movement is concerned. It is quite obvious from the graphs that the meshpoints are concentrated in the region of the shock, and move with it as time increases. The areas of high density of mesh points are in fact the areas that are changing rapidly or whose high order derivatives are changing rapidly, as expected. Further, mesh points are removed, as the shock passes by.

The effect of the spatial tolerances on the mesh selection is quite evident. The graphs with tighter tolerances show a higher mesh density, and flatter profiles. The graphs with the loose tolerances, on the other hand, show surprisingly few mesh points, and large-period oscillations. In retrospect, this is quite reasonable.

3.9 EXPERIMENTS WITH GENERAL START

The experiments with the general Start routine are very similar to those with the Pseudo-Start routine, with a few important differences. The general Start routine does not use any exact solution profile to get the first k solutions; rather it uses a lower order method to start off slowly, and gradually works up to the higher order method required. This puts some limitations on the initial conditions that we can handle, and so we must, in order to avoid prohibitively small time steps h, use the continuous initial conditions (3.5) for our problem. Note, however, that while the initial profile is smooth, it is still quite steep.

Another difference is that the tight spatial tolerances used with Pseudo-Start were unworkable with the general Start. This is perhaps because of the relatively large time step h, or perhaps contamination by lower-order errors introduced by the general Start. So the tight tolerances for these experiments are $T_1=10^{-4}$, and $T_2=10^{-2}$.

The last important difference is that the k=4 case was abandoned, as the experiments with Pseudo-Start had shown k=3 to be more efficient. Otherwise, all parameters were the same.

3.10 RESULTS OF EXPERIMENT WITH GENERAL START

For these runs, ν=0.02, $T_1=10^{-3}$, $T_2=10^{-1}$, tstart=1.0, tend=5.0

TABLE 3

Loose Tolerance Runs

	k=2	k=3
h_{low}	0.10	0.125
max. error	0.65	0.45
total cpu (s)	23.3	19.4
time/step (s)	0.39	0.37
h_{high}	0.15	0.15
max. error	0.81	0.30
total cpu (s)	22.6	15.8
time/step (s)	0.50	0.34

For these runs, ν=0.02, $T_1=10^{-4}$, $T_2=10^{-2}$, tstart=1.0, tend=5.0

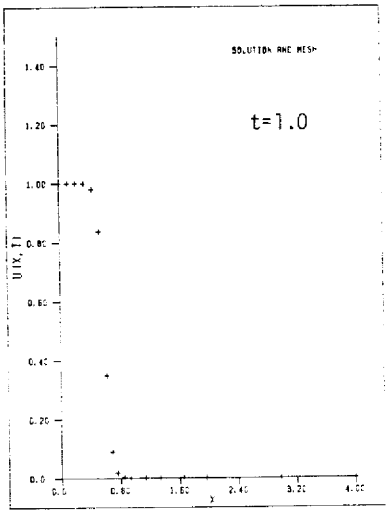
TABLE 4

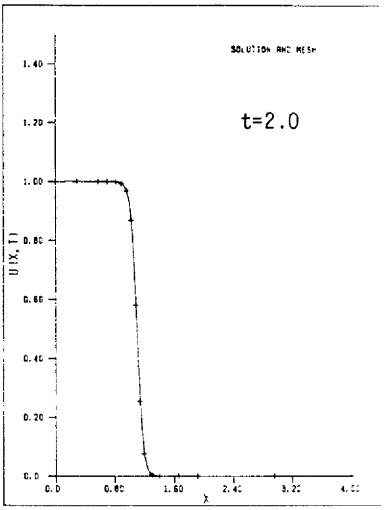
Tight Tolerance Runs

	k=2	k=3
h_{low}	0.10	0.10
max. error	3.6(-2)	3.0(-2)
total cpu (s)	36.5	32.3
time/step (s)	0.62	0.54
h_{high}	0.15	0.15
max. error	3.0(-2)	6.0(-2)
total cpu (s)	26.6	24.2
time/step (s)	0.59	0.53

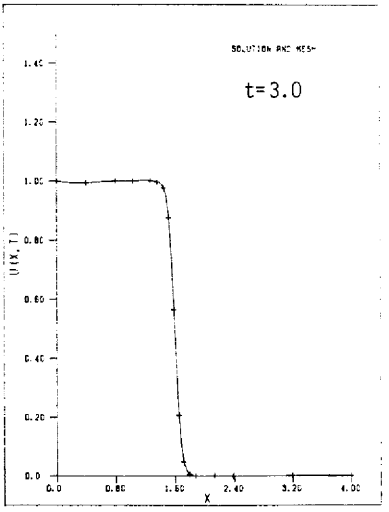
3.11 SOLUTION PROFILES FOR GENERAL START

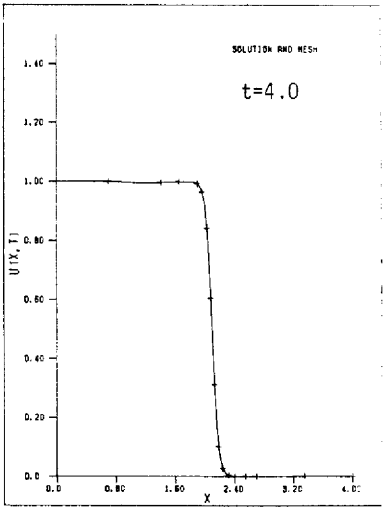
(3)

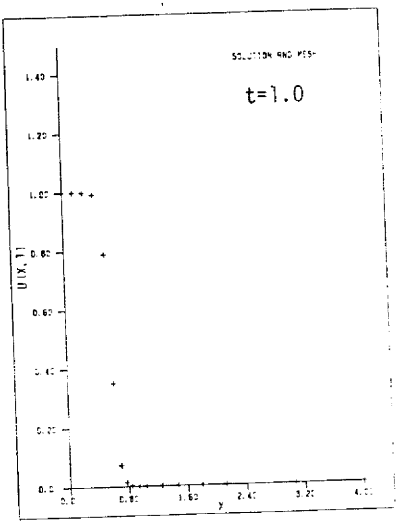


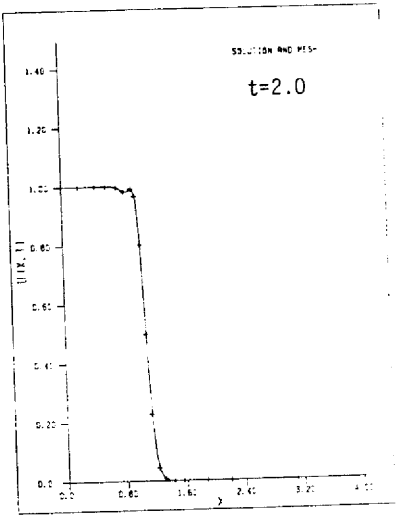


$k=2 \quad h=0.05 \quad T_1=10^{-4} \quad T_2=10^{-2}$

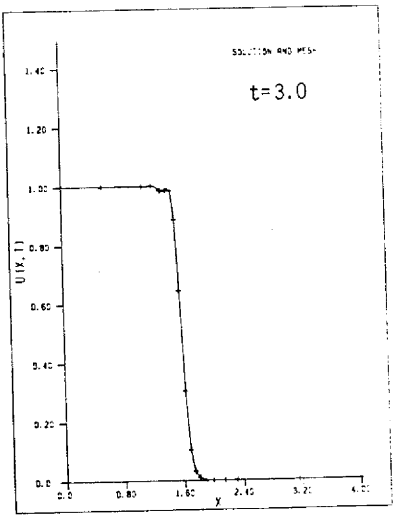


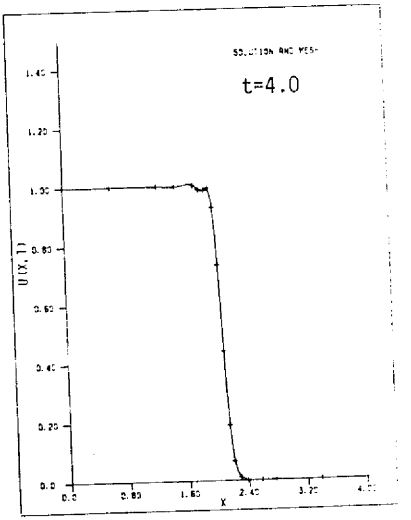




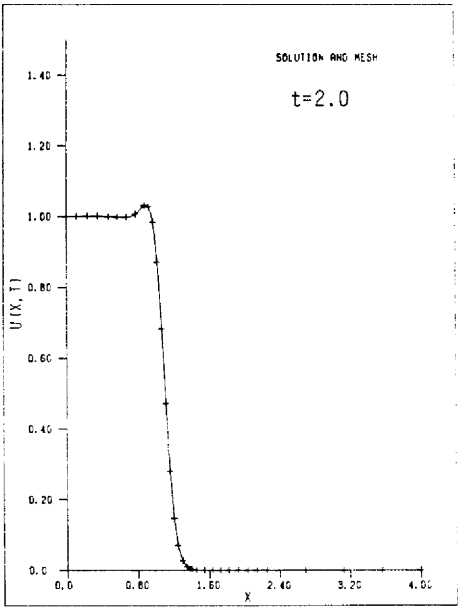


$k=2 \quad h=0.1 \quad T_1=10^{-4} \quad T_2=10^{-2}$





(5)



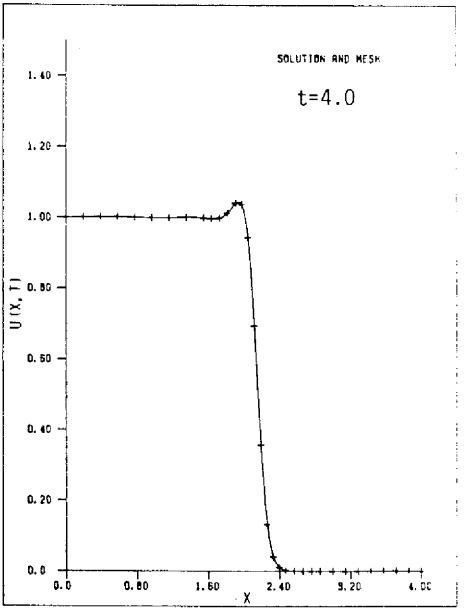
"Unacceptable"

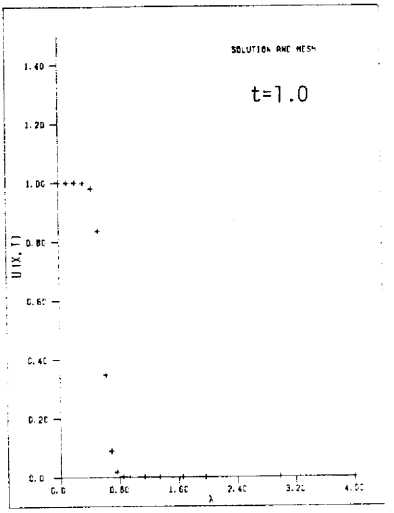
k=2

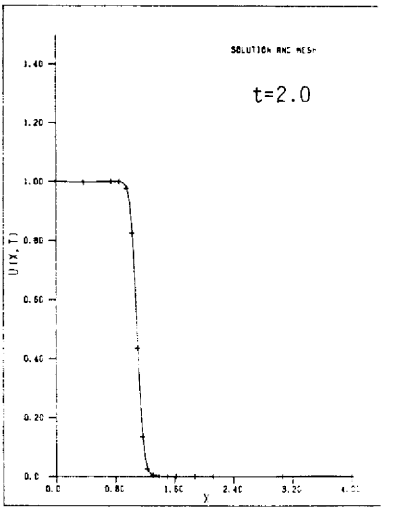
h=0.15

$T_1 = 10^{-4}$

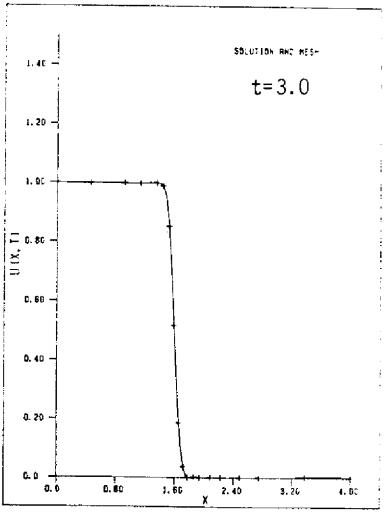
$T_2 = 10^{-2}$

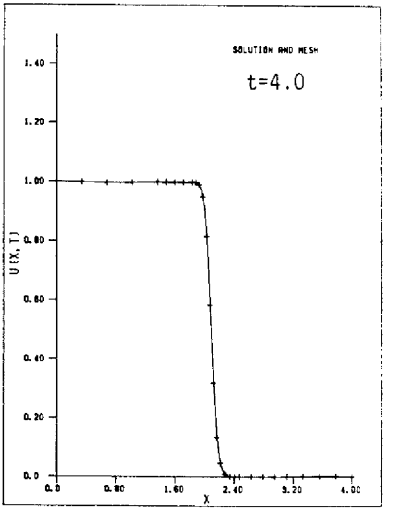


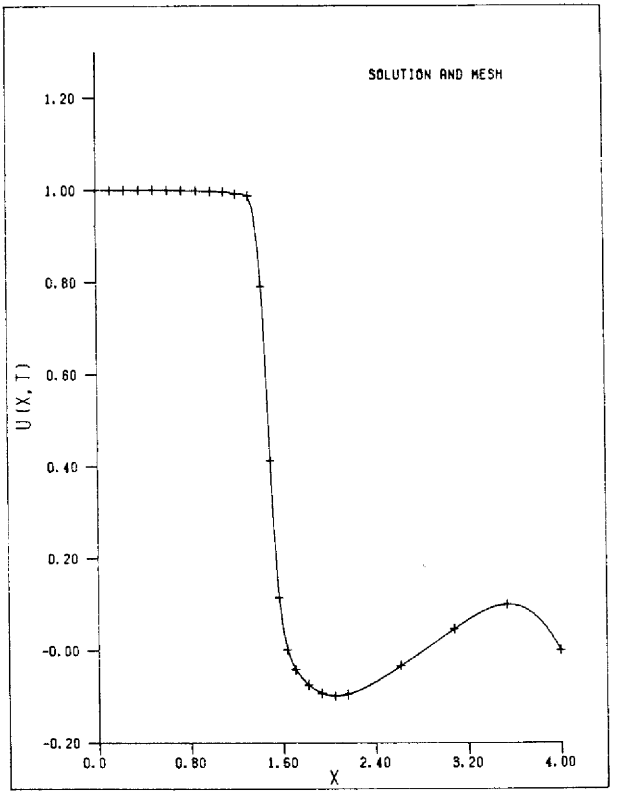




k=3 h=0.05 $T_1=10^{-4}$ $T_2=10^{-2}$







k=2 h=0.1 $T_1=10^{-3}$ $T_2=10^{-1}$

"Unacceptable"

3.12 DISCUSSION

It is evident from the times taken that k=3 no longer has the large advantage over k=2 that was previously noted. It retains a slight advantage, and so we will continue to investigate the case k=3, but perhaps due to the lower order errors introduced in the Start, it seems to be nearly as efficient to use k=2.

However, what is of more interest here is the fact that we still have oscillations, even with smooth initial conditions. After the Pseudo-Start, it was assumed that these oscillations were a natural consequence of the discontinuous initial conditions. Their presence here was surprising, and indicated that they had some other source.

3.13 FURTHER MINI-EXPERIMENTS

On the hypothesis that the oscillations were caused by the cubic spline interpolation of the Past History, a few runs were made with the cubic spline replaced by a spline in infinite tension, that is to say a broken line interpolant. This had the rather surprising effect of eliminating oscillations, at the price of increased cost and the introduction of a large amount of numerical diffusion. It is well known that introducing artificial diffusion into shock problems will reduce oscillations; I find it very interesting that this particular attempt at reducing oscillations produced the diffusion. As mentioned, however, the cost per step

rose dramatically, presumably due to the roughness of the piecewise linear interpolation. This would cause the program to see larger (high-order) derivatives than are actually present, which would force the program to use a finer mesh than necessary. Thus one would expect that a spline in tension would be better, but this needs to be investigated carefully.

There was another interesting error, which showed up in the Pseudo-Start solution profiles with the parameters k=3, h=0.1, $T_1 = 10^{-4}$, and that was a nonconservative error in the graph just behind the crest of the shock. This probably had as its source the error introduced to the problem by the use of collocation to approximate the solution to the boundary value problems. It is possible, as noted before, to eliminate this error by tightening the spatial tolerances. In order to investigate this, some rather complicated error analysis is necessary, but as this is not central to the thesis, it will be omitted.

A more appropriate analysis would be to try to investigate the error introduced by the discretization of the time derivative, and how it depends on k, h, and the problem parameter ν. This would give some idea of the relationship between k and the allowable time step, which is the parameter of major interest. Since we know the exact solution of Burgers' equation for the initial conditions we are using, this analysis is possible.

The error introduced by the approximation of the time derivative is

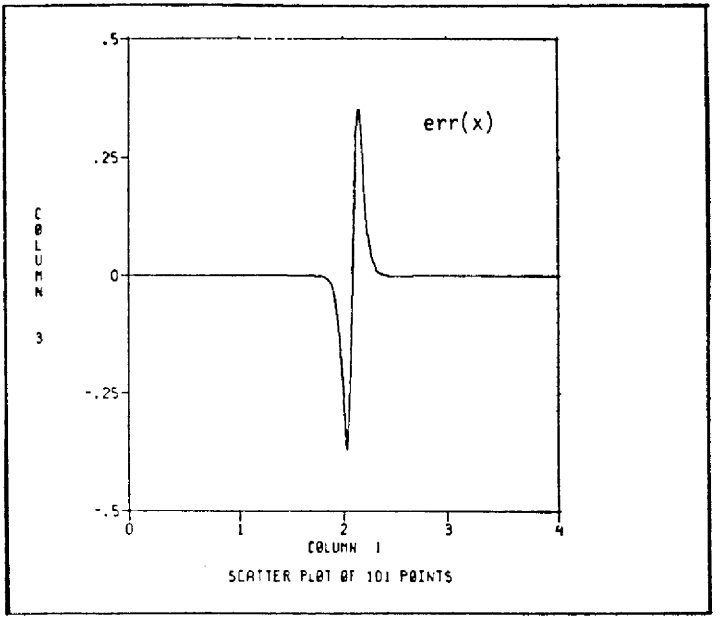
$$(3.8)\qquad \frac{d(U_e(x,t))}{dt} - \frac{U_e(x,t) - \sum_{i=1}^{k} a_i U_e(x,t-i*h)}{\beta_0 h}$$

and for various values of h, k, and , we can graph this for a fixed value of t to get an idea of the error made by this approximation. Some graphs of this function follow, and it is seen that (as expected) the error made is large only in the interval containing the shock, and again as expected looks like a constant times the $(k+1)^{st}$ time derivative of $U_e(x,t)$ times h^k. However, all we can say analytically is that

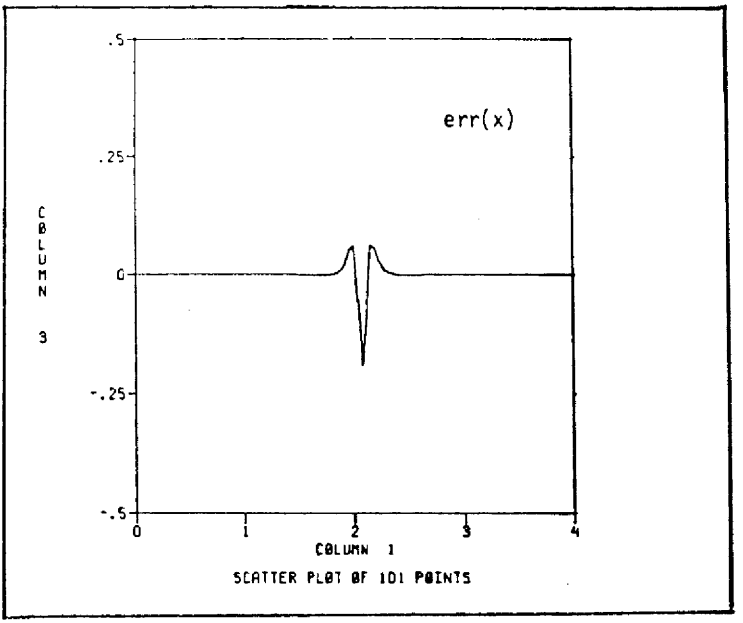
$$(3.9)\qquad err(x) = C_{k+1} \left. \frac{d^{k+1}(U_e(x,t))}{dt^{k+1}} \right|_{t=\theta(x)} h^k$$

where $t-h < \theta(x) < t$ for each x, so the resemblance to the time derivative is to be expected, but is only an approximation. The values of the constants C_{k+1} are tabulated in Appendix D.

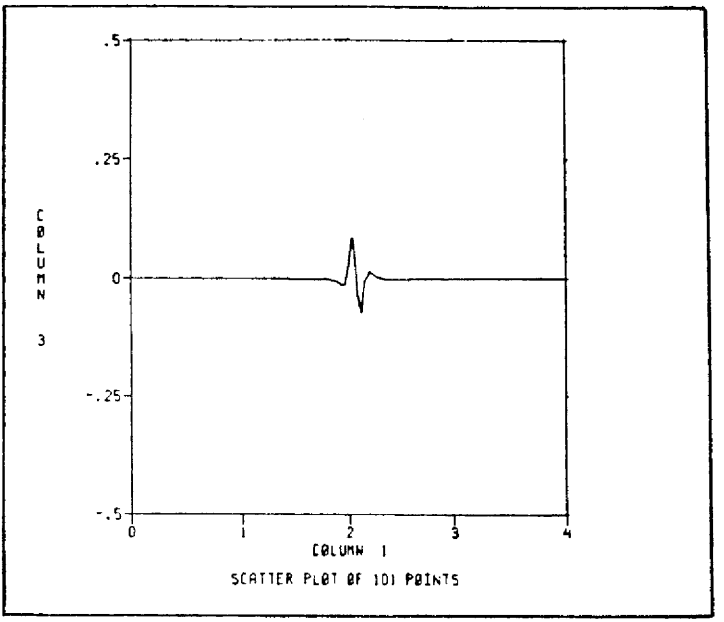
The derivatives appearing in the formula above are for $k > 3$ quite large, and proportional to ν^{-k}. Thus we expect, and we see in the graphs, the fact that k=4 is no better approximation to the time derivative than k=3 (for mod-



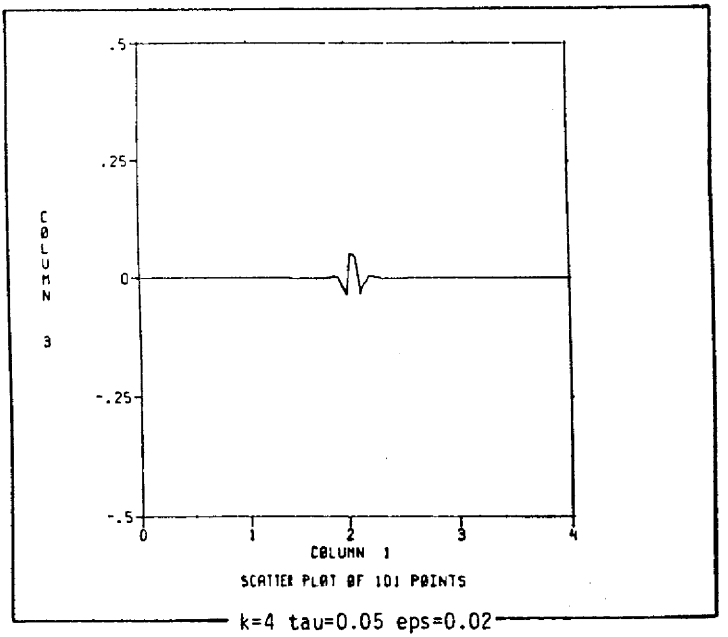
k=1 tau=0.05 eps=0.02



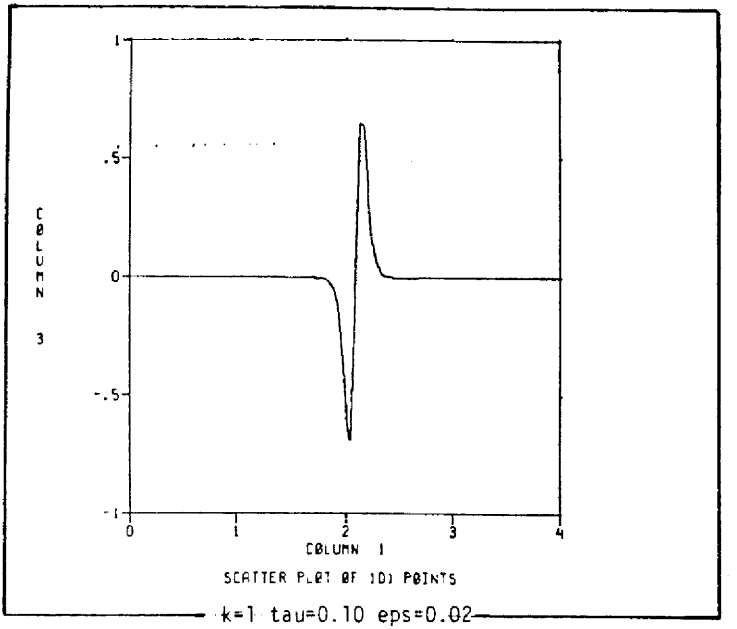
k=2 tau=0.05 eps=0.02



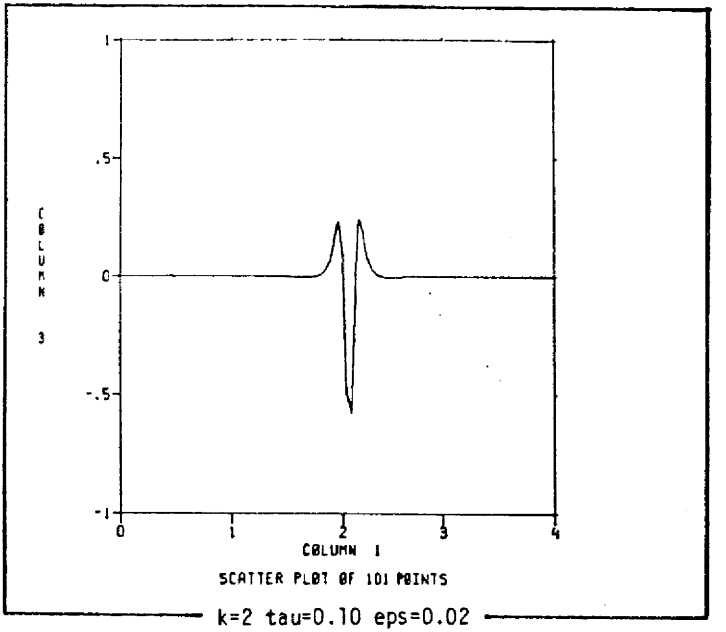
k=3 tau=0.05 eps=0.02



k=4 tau=0.05 eps=0.02



k=1 tau=0.10 eps=0.02



k=2 tau=0.10 eps=0.02

These equations are, since (4.1) is linear, linear in the $a^{(i)}$. We solve the system (4.4) for the $a^{(i)}$, and we then have an approximate solution to (4.1).

Obviously, there are many things to consider in such a method. What form is the approximating function v(x)? How do we choose the points (s_j)? For reasons of practicality, we choose the form of v(x) to be a piecewise polynomial on some mesh (x_j), and for theoretical reasons (to minimize the error $||u(x)-v(x)||_2$, where $||*||_2$ is the L_2 norm) we choose the collocation points (s_j) as the Gauss-Legendre points on [-1,1] scaled to fit each subinterval of our mesh $[x_i,x_{i+1}]$ [8]. Thus the equation (4.1) becomes a linear system of equations for the spline coefficients of the function v(x), as follows.

The number of equations is the number of collocation points plus the number of boundary condition equations, and we clearly must have this equal to the dimension of the approximating space from which we are taking v(x), in order to solve for the $a^{(i)}$. The subject of an upcoming section is the choice of approximating space, and a choice of basis for this space, both of which affect the structure of the matrix of the linear system (4.4). For details on the choice of the collocation points, see [8]. For details of the error estimates for our problem here, see [5]. Now that we have an elementary idea of what collocation is, we will proceed with the description of COLSYS, expanding and generalizing the ideas introduced here as we go along.

4.3 PROBLEM DEFINITION FOR COLSYS

Consider the mixed order system of d (possibly nonlinear) ODEs given by

$$(4.5)\quad u_i^{(m_i)} = F_i(x,z(u)) \qquad i=1,\ldots,d$$

where m_i is the order of the i^{th} differential equation, satisfying $1 \le m_1 \le m_2 \le \ldots m_d \le 4$, and where $z(u) = (u_1, u_1^{(1)}, \ldots, u_1^{(m_1-1)}, u_2, \ldots, u_d^{(m_d-1)})$, the vector of derivatives of u(x), of length m*, where $m^* = m_1 + m_2 + \ldots + m_d$. The sought solution vector is $u(x) = (u_1, u_2, \ldots, u_d)(x)$, and satisfies the following (possibly nonlinear) boundary or side conditions:

$$(4.6)\quad G_j(p_j,z(u)) = 0 \ ,$$

for $j = 1,\ldots m^*$, where m* is as above.

p_j is the location of the j^{th} boundary or side condition,

$a \le p_1 \le p_2 \le \ldots p_{m^*} \le b$.

Notice that COLSYS does not explicitly convert the problem into a first order system.

4.4 SPACE OF APPROXIMANTS

Given a spatial mesh $M := (a=x_1 < x_2 < \ldots < x_{N+1}=b)$ define $h_i = x_{i+1} - x_i$, $h = \max h_i$, and $\underline{h} = \min h_i$, and define

$$P_{(n,M)} := \{v(x) \mid v(x) \text{ restricted to } x_i < x < x_{i+1} \text{ is a polynomial of degree} < n\}$$

then we will look for an approximation to the solution of the boundary value problem in the linear spaces

$P_{(k_c+m_i,M)} \cap C^{(m_i-1)}[a,b]$, where

m_i := order of the i^{th} differential equation

and k_c := number of collocation points per subinterval (typically $k_c=3$).

Notice that this notation differs from the previous and following chapters, in that h is no longer a time step but a spatial mesh size.

4.5 CHOICE OF BASIS

For reasons of stability, flexibility, and efficiency, B-spline bases were chosen for the above approximation space. That is, $\Phi_j(x) = N_{j,k_c+m_i}(x)$, where the $N_{j,k}(x)$ are the B-splines of order k on the given mesh M. Current research [6] indicates that monomial bases are perhaps better, even for our purposes, but this is not central to the thesis. The only implication for this thesis is that with monomial bases, there is the possibility that the boundary value problems may be solved using single precision (currently the condition number of the collocation matrices arising from the use of the B-spline bases is such that double precision is required to minimize the effect of roundoff). Again, however, this is not important for our purposes.

4.6 LINEAR EQUATION SOLVER

The generalization of (4.4) for the case of a multi-component linear system with linear boundary conditions (a special case of (4.5)) is as follows. Write the problem as

$$(4.7)\quad L_i[z(u)](x) := u_i^{(m_i)} - F_{i1}(x)u_i^{(m_i-1)} - \ldots - F_{i,m_i}(x)u_i(x) = G_i(x)$$

and if

$$(4.8)\qquad v_i(x) := \sum_{j=1-k_c-m_i}^{N^*} a_{j,i}N_{j,k_c+m_i}(x)$$

where $N_{j,k}(x)$ is the j^{th} B-spline of order k on the mesh $\{x_i\}_{i=1}^{N+1}$, k_c is the number of collocation points per subinterval, and $N^* = Nk_c$. If we define $x_{i,j} := x_i + h_i c_j/2$, where the c_j are the Gauss-Legendre points on [-1,1], then our linear system can be written

$$(4.9)\quad L_i[z(v)](x_{n,j}) = G_i(x_{n,j})$$

where i=1,...d, j=1,...,k_c, and n=1,...,N.

$$(4.10)\quad B_j z(v)(p_j) = b_j$$

for j=1,...,m*.

This gives a linear system for the $(Nk_cd + m^*)$ unknowns $(a_{i,j})$. Owing to the local support for the $N_{j,k}(x)$, the system (4.9) can be arranged into an almost block diagonal

structure. This can be solved efficiently using the deBoor and Weiss package, SOLVEBLOK, modified to suit the context [2,3,5].

4.7 SOLUTION OF NONLINEAR EQUATIONS

This section, along with the mesh selection algorithm, is of major interest for this thesis. The failure of the Newton's iteration to find a solution is one of the most important forms of failure of the program. It indicates that the time step is too large, and the old solution is no longer a good enough initial guess for the current solution. It is of interest to see how efficient and robust this section of COLSYS is.

Now consider the fully nonlinear problem (4.5). Define, for suitable vectors $v := (v_1, \ldots v_d)$ and $w := (w_1, \ldots, w_d)$ the family of operators

$$(4.11) \quad L_i[w] := L_i(v)[w] = w_i^{(m_i)} - \sum_{n=1}^{m^*} \frac{\partial F_i(*; z(v))}{\partial z_n} z_n(w)$$

and the boundary condition operators

$$(4.12) \quad B_j w := B_j(v) w := \sum_{n=1}^{m^*} \frac{\partial G_i(p_j; z(v))}{\partial z_n} z_n(w)$$

which are linear in the components $z(w) := (w_1, \ldots, w_1^{(m_1-1)}, w_2, \ldots, w_d^{(m_d-1)})$.

We now attempt, using (4.11) and (4.12), to solve (4.5) by the Newton process. First, choose an initial approximation

$\underline{v}^0 \in P_{(k_c+\underline{m})} \cap C^{(\underline{m}-1)}[a,b]$

(for our purposes, this approximation is neatly at hand - the solution to the problem at the previous time step.)

Given an approximate solution $\underline{v}^s$, a corrected approximation $\underline{v}^{s+1}$ is obtained by setting

(4.13) $\quad \underline{v}^{s+1} := \underline{v}^s + r_s \underline{w}^s$

where $\underline{w}^s$, the Newton correction, is the collocation solution of the linear problem

(4.14) $\quad L_i(\underline{v}^s)\underline{w} = f_i \qquad i=1, \ldots, d.$

(4.15) $\quad B_j(\underline{v}^s)\underline{w} = b_j \qquad j=1, \ldots, m^*.$

where

(4.16) $\quad f_i := f_i(*;\underline{v}^s) := -[v_i^{(m_i)} - F_i(*;z(v))]$

(4.17) $\quad b_j := b_j(\underline{v}^s) := -G_j(p_j;z(v))$

The factor r_j in (4.13) is a relaxation factor, $0 < r_j \leq 1$. When $r_j = 1$, the iteration for $\underline{v}^s$ becomes a pure Newton's iteration. The theory and practice behind the calculation of the relaxation factor r_s is given in [15]. The important thing for us to note is that this improves the robustness of the method of solution of the nonlinear equations, especially in the case of sensitive boundary value

problems. The user of COLSYS sets a flag, telling COLSYS whether the problem is regular or sensitive. If the problem is regular, the Newton's iteration will take $r_0 = 1$, and update the Jacobian only sparingly. If the code runs into problems with the convergence, that is to say if the residuals are not decreasing fast enough, the code switches to cautious mode and uses the procedure outlined above, with careful prediction and correction of the relaxation factor. If the convergence then obtained is going smoothly enough, the program will switch back to fast mode, and so on, hopefully to a good conclusion.

If the problem is designated sensitive, COLSYS never uses fast mode, and starts the iteration with $r_0 = r_{min}$, taken by COLSYS to be 0.01. This value was chosen for empirical reasons, I believe. However, we will see that in nearly all cases for our program, the Newton's iteration converges after only one iteration. This suggests that the designation of a problem as sensitive would be quite inefficient.

The convergence criterion used by COLSYS for the nonlinear iterations is related to the convergence criterion for the mesh selection process. The code COLSYS is attempting to satisfy the following tolerances

$$(4.18) \qquad ||z_i(u) - z_i(v)|| \leq T_j(1 + ||z_i(v)||)$$

where $z_i(u)$ is the exact solution vector, $z_i(v)$ is the computed solution vector, T_j is a given tolerance, and $i=J_j$ is a given pointer to the component of z requiring tolerances. In view of the superlinear convergence of the Newton process we need only take the following simple convergence criterion to match (4.18).

$$(4.19) \quad ||z_n(\underline{v}^{s+1}) - z_n(\underline{v}^{s})|| \leq T_j(1 + ||z_n(\underline{v}^{s+1})||)$$

where $||*||$ is the infinity norm on [a,b].

It is interesting to note that if we had a better initial guess than the solution at the previous time step to the current boundary value problem, say an extrapolation of some sort, we could take larger time steps. However, one would expect such a process to be unstable, and self-defeating.

4.8 ERROR ESTIMATION

The basis for the error estimation used by COLSYS is the following formula, which holds locally when k_c, the number of collocation points per subinterval, is made large enough so that $k_c > m_d$, the largest order of the ordinary boundary value problem system [2,3,5,9].

If $p = k_c + m_j$ and $q = k_c - m_j$, then at $x \in [x_i, x_{i+1}]$,

$$(4.20) \qquad e_j(x) := u_j(x) - v_j(x)$$

$$= \frac{u_j^{(p)}(x_i)}{2^p} P_j(y) h_i^p$$

$$+ O(h^{p+1})$$

where $y = \dfrac{2(x - (x_i + x_{i+1})/2)}{h_i}$

and

$$(4.21) \qquad P_j(z) := \frac{d^q}{dz^q} \left\{ \frac{(z^2-1)^{k_c}}{(2k_c)!} \right\}$$

and further, the derivatives of the error can be approximated by the derivatives of (4.20) to $O(h^{p+1-L})$ where L is the order of the desired derivative.

The neglected global error term is of higher order than the local error term retained, so we are tempted to use equation (4.20) directly for an a posteriori error estimate, using finite differences to calculate the high order derivative at the mesh points appearing in the formula (4.20). However this is unreliable in practice for at least three reasons [2].

i) The estimate depends explicitly on the high convergence rate of the local error term, which is often not achieved in practice.

ii) The computed approximation to the high order derivatives may be inaccurate.

iii) The neglected global error term is not always of negligible magnitude, especially if the mesh is inappropriate.

In practice, the following procedure [2] is more accurate and more stable, and this is the method used by COLSYS. The error estimate is obtained by halving the mesh, computing another solution on the refined mesh, and comparing the two approximations using (4.20). Let the meshes be $\{x_i\}_{i=1}^{N+1}$ and $\{y_i\}_{i=1}^{2N+1}$ where $y_{2i-1} = x_i$, and $y_{2i} = (x_i+x_{i+1})/2$. Let the corresponding solutions on the meshes $\{x_i\}$ and $\{y_i\}$ be $\underline{v}$ and $\underline{v}'$ respectively. For each n, j, and i, $1 \leq n \leq d$, $0 \leq j \leq m_n$, and $1 \leq i \leq N$, compute the values

$$\triangle_1 := |v_n^{(j)}(x_{i+1/6}) - v'^{(j)}_n(x_{i+1/6})| \tag{4.22}$$

$$\triangle_2 := |v_n^{(j)}(x_{i+1/3}) - v'^{(j)}_n(x_{i+1/3})|$$

then using (4.20) gives

$$\max_{x \in I} |u_n^{(j)}(x) - v'^{(j)}_n(x)| \tag{4.23}$$

$$\doteq w_{k_c,k_c-m_n+j}(\triangle_1 + \triangle_2)$$

where I is the interval $[y_{2i-1}, y_{2i}]$, and where $v_n^{(j)}(x)$ is the j^{th} derivative of $v_n(x)$, the n^{th} component of $\underline{v}$, and where the $w_{k_c,q}$ are precomputed weights, k_c=1, ... 5, and q=0, ... k_c-1. For the details of the derivation and use of (4.23) see [5].

This method of estimating the error has proved to be quite accurate, and is much less susceptible to the problems associated with the direct use of (4.23). However, if the mesh is inappropriate, even (4.23) may fail to hold. If, on the other hand, the mesh is even reasonable (with quite a broad interpretation of reasonable) the estimate (4.23) becomes quite accurate, even if the mesh is highly nonuniform. This brings us naturally to the next section.

4.9 MESH SELECTION ALGORITHM

Basically, the mesh selection algorithm of COLSYS is an iterative scheme to improve a given initial mesh until it allows the program to calculate a solution to the problem which satisfies the user-given error tolerances. In our context, we are taking the mesh used at the previous time step as the initial mesh for the current time step. The rationale behind the mesh selection procedure is to select a mesh which approximately equidistributes the local error, estimated by (4.20), and thus hopefully equidistributes the global error as well. Equidistribution refers to the process of making the estimated error on each subinterval satisfy the following [29,30,12]

$$(4.24) \qquad h_i \| T_i(x) \|_p^p \cong \text{constant for all } i.$$

where $T_i(x)$ is the error estimate (4.20), restricted to $[x_i, x_{i+1}]$, and $\|*\|_p$ is some suitable norm, in our case the

L_2 norm on the subinterval $[x_i, x_{i+1}]$. This essentially says that if the high-order derivative is large in an area, one should make the subintervals smaller there, because the local error (and hopefully the global error) depends on these two quantities. This is done by using the solution $v(x)$ computed on a current mesh $\{x_i\}$ to calculate a new mesh $\{x'_i\}$, consisting of a possibly different number of points at different places, which satisfy the following criteria.

1. The solution on the mesh $\{y'_i\}$, the doubled refinement of $\{x'_i\}$, approximately satisfies

   $$(4.25) \qquad ||z_i(u) - z_i(v)|| \leq T_j(1 + ||z_i(v)||)$$

   where $z(u) = (u_1, u_1^{(1)}, \ldots, u_1^{(m_1-1)}, u_2, \ldots, u_d^{(m_d-1)})$ is the vector that would result on converting (4.5) to a first order system, and $i = J_j$ is the pointer to the component of z requiring tolerances, and the T_j are the given tolerances.
2. The number of meshpoints N' is approximately the minimum N for which the solution on the halved mesh satisfies (4.25).

The details of the method used by COLSYS to calculate the mesh can be found in [2,3,5,29], so only a brief outline of the algorithm will be given here.

COLSYS Algorithm for Mesh Selection [2].

1. Given the current mesh $\{x_i\}$, compute a collocation solution to the problem (4.5) $\underline{v}$.
2. If the nonlinear iteration for $\underline{v}$ did not converge, halve the current mesh. If the new mesh is too large, (larger than NMAX, the largest that our storage will permit), then exit with failure; otherwise let the refined mesh become the current one, and go to step 1.
3. If the current mesh has been obtained by halving a former one, and convergence occurred on both, then compute the error estimate using (4.23), and check if (4.25) are satisfied. If so, then exit successfully; otherwise proceed.
4. Compute the selection parameters r_1, r_2, and r_3.[1] (For details see [2]).
5. If $r_1 < 2r_3$ then halve the current mesh, let the refined mesh become the current one, and go to step 1. This step is COLSYS' check on suspicious mesh changes, and reflects a certain amount of pessimism in the error estimation and equidistribution calculations (the r_i's).
6. Otherwise, the predicted new mesh is acceptable. Define

---

[1] See the remarks on the meaning of these parameters on the next page.

$N' := \min (NMAX/2, N, \max(N, r_2)/2)$

and compute the new mesh of N' subintervals by requiring that the error be equidistributed, let the new mesh become the current one, and go to step 1.

There are several remarks made about this algorithm in [2], here reproduced.

1. The ratio r_1/r_3 represents the amount by which the mesh fails to be equidistributed.
2. The choice of N' is made so that $N/2 \leq N' \leq N$ and so that a subsequent halving of the mesh is possible. Also, r_2 is a prediction by the code for the optimal N'.
3. As an added measure of caution, the mesh is automatically halved if
   i) the size of the new mesh is smaller than that of the former mesh, or
   ii) There have been 3 consecutive mesh selections resulting in the same mesh size N, or
   iii) there have been 3 consecutive pairs of mesh selections followed by mesh halving, resulting in the same mesh size, N.

The safeguards detailed above represent an attempt by COLSYS to guard against insufficiently good initial and intermediate meshes. It is not clear that all of this is required in our context, and indeed the mesh selection algorithm may be doing too much work from DYNACOL's point of view. This will be looked at in more detail in the next chapter.

Whether or not COLSYS performs too much work to get the meshes for our problems, or whether the pessimism in the algorithm is in fact required, it is clear that the meshes produced by the program are very good, and very appropriate. Note that the selection algorithm, though pessimistic, may require a sufficiently good initial guess as to the initial mesh. In our context, where we use the mesh from the previous time step (or a coarser version of it) as our initial mesh, this represents another limitation on the maximum allowable time step $\tilde{h}$. However, it is not clear whether the mesh selection strategy or the Newton's method provide the most stringent limitation on the time step size, as it is difficult to separate the effects of these two segments of COLSYS.

4.10 PROGRAMMING CONSIDERATIONS

In adapting an existing code (some would say perverting) to do a task it was not designed for, there are inevitably problems. Some of the more important ones encountered in the adaption of COLSYS are discussed here. Some problems specific to adapting COLSYS are mentioned, as well as some which apply to any adaption.

The first difficulty encountered was the problem of passing large arrays through COLSYS to the problem dependent routines called by COLSYS. This has the easy but awkward solution of having these large arrays in common block, where

they have to be changed every time the problem size is changed. A more serious problem along these lines was the design of the Past History array to allow direct use with the COLSYS routine APPSLN, which evaluates the B-splines at any given $x \in [a,b]$. This again required either large arrays in common block, or with their row dimensions having to be changed in each routine that needed the Past History arrays. In a more production oriented implementation, this could be improved by the use of random access scratch files, but for our purposes, the simple approach was adequate.

The next problem to come up was one of efficiency. COLSYS was designed to be called a single time, or at most a few times for continuation purposes. Thus every time it is called, it re-initializes a large number of parameters. This is inefficient, but we did not want to do any modifications whatsoever to COLSYS, as this study is concerned with simple adaption only.

Some possible problems which turned out to be non-problems because of thoughtful and foresighted design on the part of the authors of COLSYS were the portability of COLSYS, which is excellent, and the continuation process mentioned above. The simple continuation process applies to problems where there is a simple natural parameter where the problem is easy to solve for one value of the parameter, and difficult for others. One solves the easy problem and uses this solution as an initial approximation for a slightly

harder problem chosen from the family, and so on iteratively until the desired parameter value is attained. If there is not a natural parameter, it is often the case that an artificial imbedding into a family of problems can be of use. The mechanisms incorporated into the package COLSYS for handling simple continuation proved to be massively useful in the adaption to partial differential equations. They allowed in a very simple way, with just the setting of a switch, the use of the previous solution as an initial guess to the solution at the current time step, and the use of the former mesh (or a coarser version of the former mesh) as an initial mesh for the solution at the current time step. This turns out to be a very robust procedure, although not the only way to proceed, as we shall see in the next chapter.

Chapter V

GAS DYNAMICS EXPERIMENTS

5.1 HYPERBOLIC FORM OF PROBLEM

The next test problem we will use is the prototype multi-component hyperbolic conservation law problem, the gas dynamics of a shock tube. The equations are, in conservation form,

$$r_t + m_x = 0 \tag{5.1a}$$

$$m_t + \left(\frac{m^2}{r} + p\right)_x = 0 \tag{5.1b}$$

$$E_t + \left(\frac{m}{r}(E + p)\right)_x = 0 \tag{5.1c}$$

or in vector notation

$$U_t + \{F(U)\}_x = 0. \tag{5.2}$$

or

$$U_t + A(U)U_x = 0. \tag{5.2b}$$

The density of the gas is r, m = rv is the momentum (v is the eulerian velocity), p is the pressure, and E is the energy per unit volume. We are assuming further that the gas is polytropic, so we have the equations of state

$$e = \frac{p}{(\gamma - 1)r} \tag{5.3}$$

and

$$p = Kr^{\gamma} \tag{5.4}$$

where $\gamma > 1$ is the ratio of specific heats, (taken here to be $\gamma = 1.4$, corresponding to a diatomic gas), and e is the internal energy per unit mass, and K is a constant that depends only on the entropy. Then

$$E = re + \frac{1}{2}\left(\frac{m}{r}\right)^2 \tag{5.5}$$

so

$$p = (\gamma - 1)\left(E - \frac{1}{2}\left(\frac{m}{r}\right)^2\right) \tag{5.6}$$

The initial conditions we will use for this set of experiments come from [33], but the problem is in fact a standard test problem, going back at least to 1954 [22]. This problem is used extensively as a test problem for schemes for solving hyperbolic (nonlinear) conservation laws. Sod [33] and later Orzag and others [19] have used this problem with the following initial conditions as a test problem for several different types of codes. This problem represents the next level of complexity after Burgers' equation.

The initial conditions correspond to the initial state of a shock tube, which is a long narrow tube partitioned into two sections by a diaphragm. The gas behind the dia-

phragm is at a higher pressure than that in front, and when the diaphragm is burst at t=0 the behaviour of the gas is modeled by the equations (5.1), neglecting such effects as heat conduction and viscosity. We also make the idealization that the effects of the bursting of the diaphragm can be ignored. The initial conditions, then, are as follows:

$$(5.7)\qquad r(x,0) = \begin{cases} 1 & -5 \le x \le 0 \\ 0.125 & 0 < x \le 5 \end{cases}$$

$$m(x,0) = 0 \qquad -5 \le x \le 5$$

$$E(x,0) = \begin{cases} 2.50 & -5 \le x \le 0 \\ 0.25 & 0 < x \le 5 \end{cases}$$

It is well known that for hyperbolic conservation laws, even smooth initial conditions can produce solutions which eventually become discontinuous. So when we speak here of a solution of (5.1), we will mean a weak solution, in the sense of Lax [22], where a weak solution of (5.1) is a vector-valued function U satisfying [22,24,36]

$$(5.8)\qquad \int_0^\infty\int_{-\infty}^{\infty} \{W_t U + W_x F(U)\}dx\,dt$$

$$= -\int_{-\infty}^{\infty} \{W(x,0)\Phi(x)\}\,dx$$

where $W(x,t)$ is any smooth test vector vanishing for $|x| + t$ large enough, and the initial conditions are represented by $U(x,0) = \Phi(x)$. This equation is obtained from (5.2) by multiplication through by W and integration by parts over any region in the upper half plane. This definition of weak solution is similar to the definition used in the construction of finite element (Galerkin) methods.

Thus any classical solution to (5.1) is also a weak solution, though if u is a weak solution, it is a classical solution only if it is smooth enough. For example, if u is a $C^{(1)}$ weak solution, it is also a classical solution.

5.2 ANALYTICAL SOLUTION OF THE HYPERBOLIC PROBLEM

The integrated form (5.8) places a restriction on the kinds of discontinuities that may occur in U. If $(x(t),t)$ is the path of a discontinuity in U and $S := 1/x'(t)$, then

$$S[U] = [F(U)] \tag{5.9}$$

where $[*]$ means the difference in * across the discontinuity. Consequently, if

U_2 = value of $U := (r, m, E)$ at x_2 "just behind" the shock, and if

U_1 = value of U at x_1, "just in front" of the shock, then

$$[U] := U_2 - U_1$$

The physically meaningful weak solutions of (5.1) will also satisfy entropy conditions [24]. If $\mu_k(u)$ is one of the three real and distinct eigenvalues of the matrix A(U) (from 5.2b)), and S (as above) is the speed of the shock separating U_2 and U_1, then the entropy conditions ensuring that U is a physically realizable weak solution are

$$\mu_k(U_2) > S > \mu_k(U_1) \tag{5.10}$$

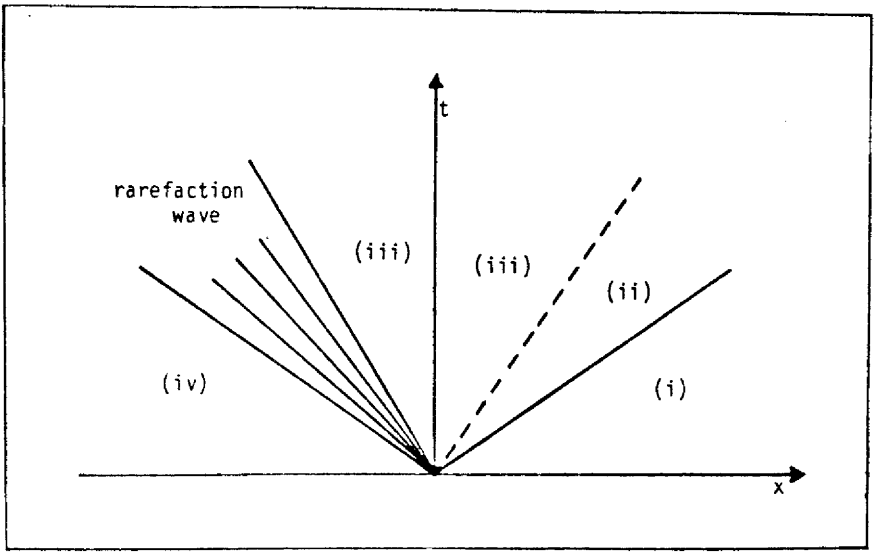


Figure 3: (x,t) diagram for shock tube

Figure 3 is the (x,t) diagram for the solution to the shock tube problem. There are 4 regions of constant states, and a centred shock wave or rarefaction wave between states

(iii) and (iv). Regions (i) and (iv) are unchanged after t=0, so they remain at the low pressure state and the high pressure state, respectively. Region (ii) is separated from region (i) by a shock wave, and the jump conditions (5.9) may be used [36] to determine the values of r, m, and E behind the shock, giving

$$r_{ii} = 0.266$$

$$m_{ii} = 0.246$$

$$E_{ii} = 0.872$$

and the further (derived) values

$$p_{ii} = 0.303$$

$$v_{ii} = 0.927$$

$$e_{ii} = 2.89$$

Region (iii) is separated from (ii) by a discontinuity called a contact discontinuity, across which the velocity and the pressure are continuous. Using the jump conditions again, we get

$$r_{iii} = 0.426$$

$$m_{iii} = 0.395$$

$$E_{iii} = 0.941$$

and (derived)

$$e_{iii} = 1.77$$

Now the only quantities remaining to be calculated are the speeds of the shock between (i) and (ii), the contact discontinuity between (ii) and (iii), and the details of the rarefaction wave. These are

S = speed of shock wave = 1.75

v_c = speed of contact discontinuity = 0.927

and if

$$a_{iv} = \gamma^{\frac{1}{2}} = \text{sound speed in region (iv)}$$

then for $-1 \leq (\frac{1}{a_{iv}})(\frac{x}{t}) \leq -1 + \frac{\gamma+1}{2a_{iv}}v_c$, the following equations hold:

$$u = \frac{2a_{iv}}{\gamma+1}(1 + (\frac{1}{a_{iv}})(\frac{x}{t}))$$

$$a = a_{iv}(1 - \frac{\gamma-1}{\gamma+1}(1 + (\frac{1}{a_{iv}})(\frac{x}{t})))$$

$$r = \left\{ \frac{a^2}{\gamma} \right\}^{(1/(\gamma-1))}$$

$$p = r^{\gamma}$$

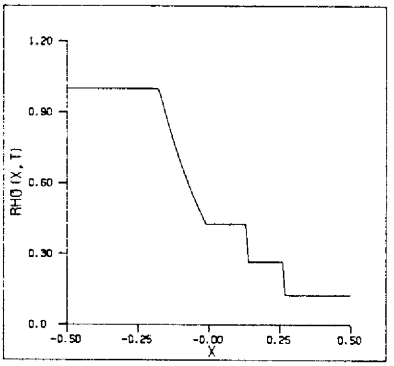
$$E = \frac{p}{\gamma-1} + \frac{1}{2}(\frac{m^2}{r})$$

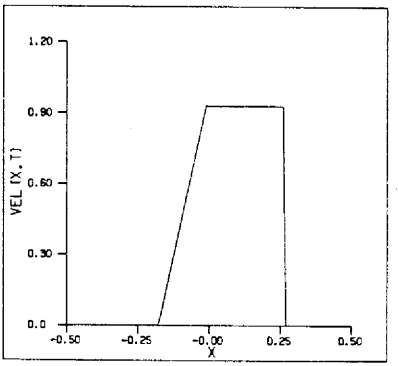
$$e = \frac{a^2}{\gamma(\gamma-1)}.$$

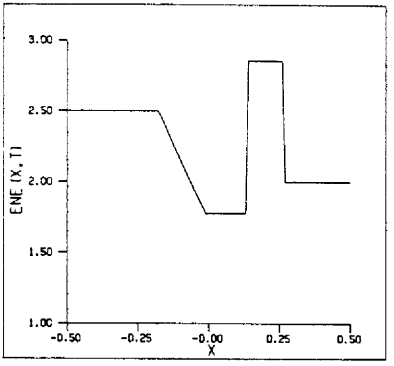
These are all continuous monotone functions of $\frac{x}{t}$.

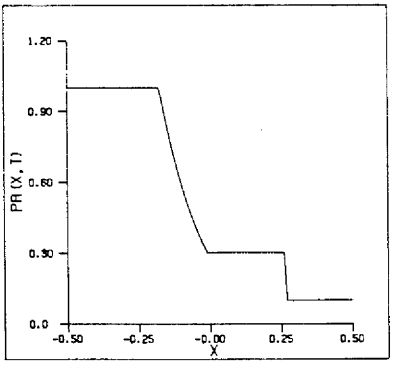
This analytical solution will appear in some of the graphs as a continuous line.

5.3 ANALYTICAL SOLUTION PROFILES









5.4 CONVERSION TO COLSYS FORM

If we attempt to convert (5.1) to COLSYS form directly, replacing U_t by the appropriate linear combination, the boundary value problems that result are systems of first order differential equations and a coupled algebraic equation for r. The appropriate boundary conditions for these systems are not clear, and it is reported [19] that the hyperbolic problem is in fact very sensitive to the boundary conditions. Also, even when the boundary conditions are applied correctly, we might be asking COLSYS to compute approximate solutions to boundary value (or initial value) problems with discontinuous solutions. This could well be too much to ask from COLSYS. Thus, to make the problem simpler, we use the classical technique of adding artificial viscosity to the hyperbolic system, by adding a term proportional to U_{xx}. Our system (5.1) then becomes

$$U_t + \{F(U)\}_x = \lambda U_{xx}. \tag{5.11}$$

In [22], Lax proves that if the solutions $U(x,t;\lambda)$ of (5.11) converge in the L_1 sense to a limit $\underline{U}(x,t)$ as λ goes to 0^+, then $\underline{U}(x,t)$ is a weak solution of (5.1). Further, in [24] he proves that $\underline{U}(x,t)$ is the correct, physically realizable weak solution of (5.1) in that it satisfies his entropy conditions (5.10). Finally, Foy [16] proves that the solutions of (5.11) do indeed converge if the original shocks are weak enough. Therefore, the addition of artifi-

cial viscosity, while simple, will not destroy the essential character of the hyperbolic equations (5.1). It is no surprise that this set of equations is easy to transform to COLSYS form, although somewhat tedious. In an effort to avoid arithmetic mistakes, the symbolic manipulation package MAPLE developed at Waterloo by Drs. K.O. Geddes and G.H. Gonnet was used to take all the necessary derivatives [18].

The other main difficulty with the hyperbolic problem is the discontinuous initial conditions. As remarked in the context of Burgers' equation, a discontinuity in the initial conditions puts too severe a limitation on the allowable time step size. We smoothed out the initial conditions, making them $C^{(2)}$ everywhere, replacing the discontinuity with a smoothed drop of width s (typically $s = 4. \times 10^{-4}$). This was done by fitting a polynomial of sixth degree to the following conditions.

$$
\begin{aligned}
(5.12)\quad & U(-s/2,\ 0) = U_{left} && U(s/2,\ 0) = U_{right} \\
& U_x(-s/2,0) = 0.0 && U_x(s/2,0) = 0.0 \\
& U_{xx}(-s/2,0) = 0.0 && U_{xx}(s/2,0) = 0.0
\end{aligned}
$$

5.5 FIRST EXPERIMENTS WITH GAS DYNAMICS PROBLEM

Several questions arose on examination of the Burgers' equation results. One unlooked-for problem was the effect of the method of approximating $v^{(n)}$, the linear combination of $U(x,t)$ at previous times (see chapter 2). We decided to do some rough preliminary experiments to help decide which method to use for the experiments with the gas dynamics problem. We wished to have only one method used throughout these experiments, as they are sufficiently complicated already.

The three methods of approximating $v^{(n)}(x)$ were

1. Direct.

   This method used the COLSYS representation of $U(x,(n-i)h)$ for $i=1, \ldots,k$ directly, by using the COLSYS routine APPSLN to evaluate $U(x,(n-i)h)$ for a given x and then taking $v^{(n)}$ as the appropriate linear combination of these values.

2. Cubic.

   This method, immediately previous to the solution of the current boundary value problem, used method 1 to calculate $v^{(n)}$ at a discrete set of points (a merging of the meshes used by COLSYS to approximate each $u^{(n-i)}$) and calculated a cubic spline interpolant for this. This was done so that $v^{(n)}(x)$ is evaluated as cheaply as possible. As function evaluations are a significant part of the cost of

boundary value problem solvers in general and COLSYS in particular, it was expected that this would prove useful.

3. Linear.

This method is similar to method 2, except that interpolation is done by a "spline in infinite tension", i.e. a piecewise linear interpolant.

Briefly, the results of these "pre-experiments" were inconclusive. The cubic method is indeed the cheapest for a given set of other parameters, but it also produces the worst graphs. Linear interpolation is next in cost, though often producing better graphs. The most expensive method, as expected, is the direct method; however, it is the most accurate, producing the flattest and smoothest profiles. The ratios of the costs varied, depending on the order of the method especially, but two typical results were as follows.

If

D = cost of direct method per time step

L = cost of linear method per time step

C = cost of cubic method per time step

then

(i) $D \cong 2L \cong 4C$

(ii) $D \cong 1.6L \cong 2C$.

The only conclusion we may draw from these rough experiments is that the question of efficiency of the methods needs further study. However, as we are more interested in the robustness of the method than we are in the cost, we will do all our experiments with the most expensive method, the direct method.

Another phenomenon that occurred in these preliminary experiments was the fact that COLSYS was often using more cpu time than seemed necessary to select a mesh, and when it had done so, the difference between the current mesh and the previous mesh was not all that great. This was not due to any fault of COLSYS; rather, DYNACOL would provide an initial mesh for each time step which was the previous mesh coarsened by a factor of two. This kept the total mesh size manageable, but it turned out not to be the most efficient strategy. As a minor modification, we changed this so that COLSYS would use the full previous mesh as an initial mesh if it was small enough (less than 40 subintervals), and otherwise use the coarsened mesh. This, surprisingly, improved the performance of DYNACOL markedly. This modified algorithm is used throughout the gas dynamics experiments.

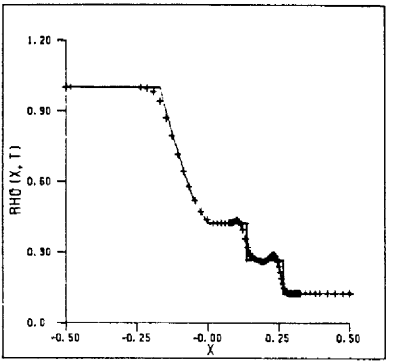
Now that the preliminaries have been dealt with, we may proceed to the gas dynamics experiments proper. We wish to find, as with Burgers' equation, the "extremes" of h: values of the time step which represent the upper limits on the time step size h. The values of the other parameters were chosen (by use of judicious trial runs) to be the following.

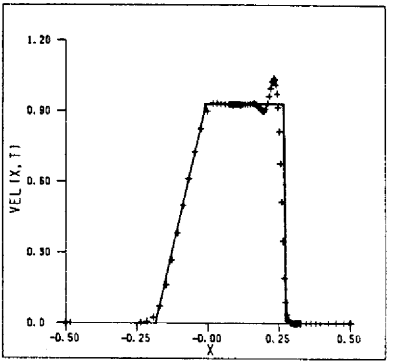
λ	= $5. \times 10^{-4}$	
T_1	= $5. \times 10^{-5}$	tolerances were placed
T_2	= $5. \times 10^{-3}$	on all six components. T_1 is the tolerance placed on the components r, m, and E; T_2 is the tolerance on the 1^{st} derivatives.
t_{end}	= 0.15	The final time for the profiles presented in the review by Sod [33].
initial mesh		uniform, of 5 subintervals.
sensitivity		regular.

The results of the experiments are reported in the following table.

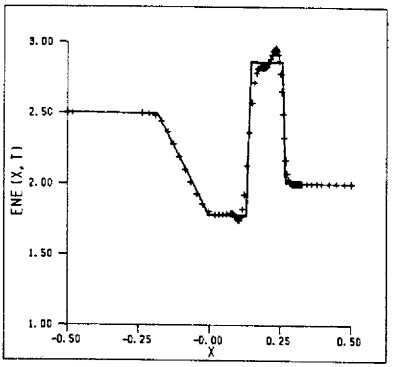
5.6 RESULTS OF GAS DYNAMICS EXPERIMENT

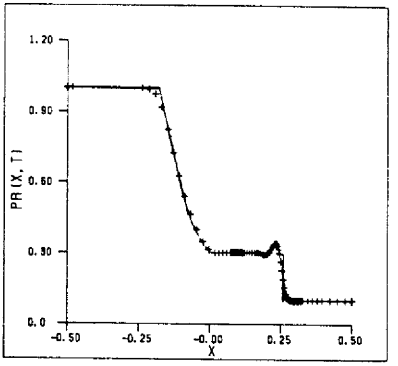
	k=2	k=3
h_{low}	0.005	0.005
start time (s)	200	235
total cpu (s)	590	>600
time/step (s)	13.0	22.0
h_{high}	0.01	0.01
start time (s)	212	244
total cpu (s)	445	550
time/step (s)	15.0	23.0

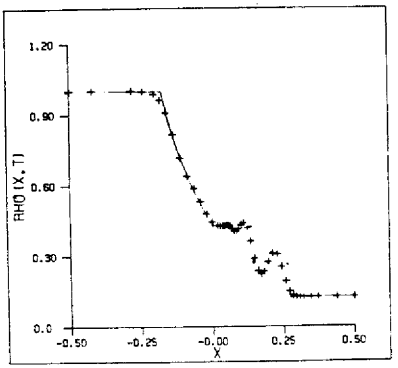


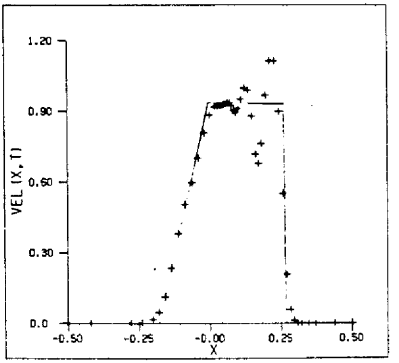


k=2 h=0.005 λ=5x10^{-4}

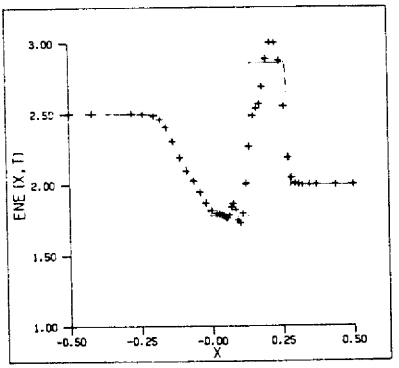


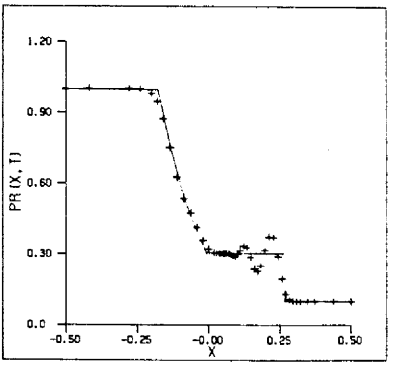




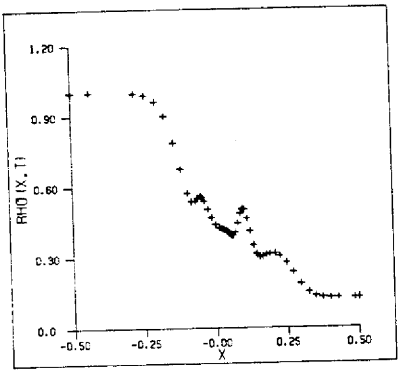


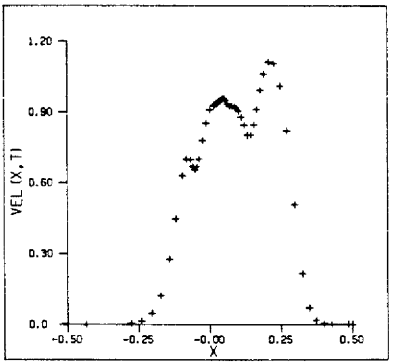
k=3 h=0.005 $\lambda=5\times10^{-4}$



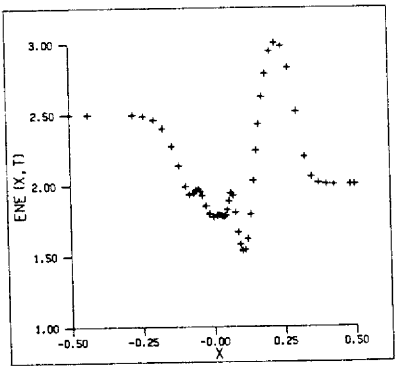


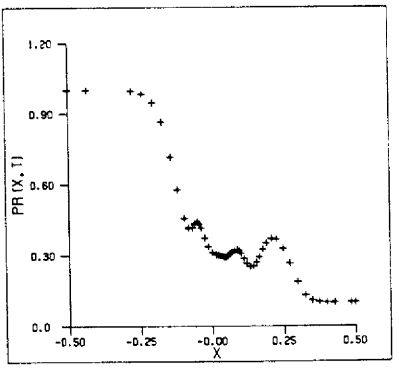
Sample Rejection Graph:





k=3 $\lambda=5\times10^{-4}$ h=0.025





5.7 DISCUSSION

The reason for rejection of the solutions for $h = h_{high}$, as is evident from the profiles shown, was the presence of unacceptable oscillations behind the shock. It is important to note, however, that these are stable oscillations, and as the number of time steps increases, these oscillations move with the shock, and leave the correct profile behind them. Note also that the profiles for k=2 are surprisingly better than the corresponding profiles for k=3. They are significantly less expensive to obtain, in contradiction to the Burgers' equation results, where k=3 was superior. This is believed principally due to the small size of the diffusion: the error made in approximating U_t by the Gear backward difference is (as a similar argument to the one used for Burgers' equation shows) $O(\{h\lambda^{-1}\}^{k+1})$; here h = 0.005, and $\lambda = 5. \times 10^{-4}$, so the error introduced (speaking very roughly, as the constants differ) for k=3 is approximately 10 times larger than the corresponding error for k=2. If, on the other hand, the ratio $h\lambda^{-1} < 1$, the higher order method could be expected to perform better. In this case, this is impractical because a radically larger artificial viscosity smears the contact discontinuity, while a smaller time step is too expensive.

Thus, the increased cost of the k=3 case has at least two contributing factors:

1. the direct method uses three evaluations per function call rather than two in the k=2 case.
2. The linear combination of the past history, which is used to approximate the time derivative, is noisier, which forces the solution to have more mesh points than are needed (remember that the mesh selection algorithm depends on the values of the high order derivatives, which are larger in an oscillatory solution than they should be). This last factor can be seen in the fact that for k=2, the average number of mesh points is roughly 35, while for k=3 the average is over 65.

The superiority of k=2 raises the question of what the results would be for k=1, the backwards Euler method. Unfortunately, k=1 with the other parameter values used here is one of the "unlucky" sets of parameters discussed later, and we were not able to compare the results directly. In the next set of experiments, however, we will see a comparison of k=1 with k=2. It is noted here that the case k=1 does indeed produce a better profile near the top of the shock, but fails in other respects, as will be seen. Heretofore k=1 was not considered, as a first order method is believed to be more expensive for a given accuracy.

It is instructive to compare these time steps and profiles with those reported by Sod. The limitation on the time step for the finite difference schemes is a stability

requirement, that $\triangle t < K\triangle x$ for some constant K. However, for $\triangle x = 0.01$, corresponding to 100 uniform intervals on [0,1], the allowable time step for most of the schemes is about 0.005. Thus our method, even though implicit, does not give a larger allowable time step, as hoped.

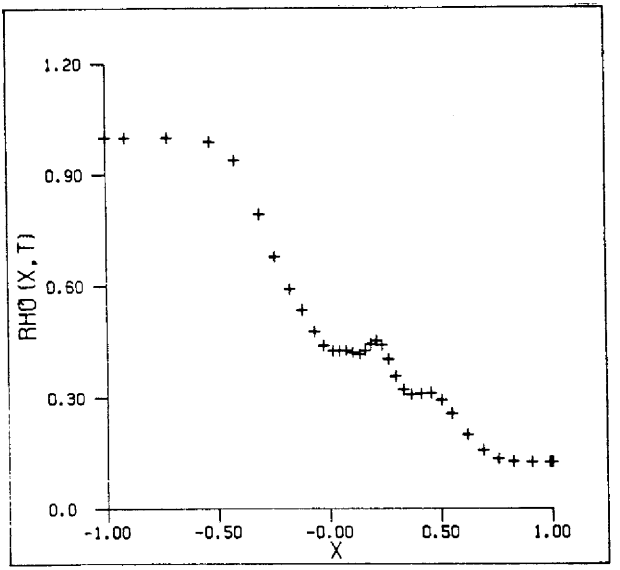


Figure 4: Profile of large time step solution

However, it is to be noted that the method will produce recognizable profiles at truly large time steps: e.g. h = 0.125 (approximately 25 times as large as is possible for the finite difference schemes on 100 points) produces the profile presented in Figure 4. This profile, while extremely inaccurate, is recognizable, thus indicating that the limitation on the time step for our method is an accuracy

limit, and not a stability requirement. For this (severe) test problem, there is no difference, but for easier problems, with milder phenomena, this could make a large difference.

There is another problem with large time steps, or even moderately small time steps. Some time steps are "unlucky", in that with the crude initial mesh, COLSYS is unable to find a first solution (i.e. a good enough first mesh). For example, h = 0.03125 produces recognizable (though bad) profiles, while h = 0.03, only slightly smaller, fails on the first step. This is a "region of accessibility" problem similar to that of Newton's method for solving nonlinear equations; we can guarantee a success only when either the initial mesh or the initial guess as to the solution are "good enough"; that is, when the time step is small enough. Of course, provision of a better initial mesh than the 5 uniform subintervals may help a great deal. In the case quoted above, however, there is a simpler solution, and that is to increase the maximum number of mesh points, NMAX. This allows COLSYS to get a sufficiently fine initial mesh to be able to continue.

Some discussion of the overshoot in the "acceptable" profile for k=2 is in order. The origin of this phenomenon is not clear. It could be a Gibbs phenomenon, resulting from the approximation of a sharp rise by a finite number of smooth functions; it could be an effect introduced by the

addition of artificial viscosity (we are guaranteed only L_1 convergence, not pointwise convergence); or it could be an effect introduced by the time discretization. The first possibility seemed the most likely; however, we expected the dynamic spatial mesh to be able to move the points close enough together to handle this, and as it is not unknown for COLSYS to have subintervals of greatly varying size next to each other, (as much as 800 times as large for adjacent intervals, and ratios as large as 10^6 for the ratio of the maximum size to minimum size [5]) it would be surprising if the mesh selection could not ensure that the corner was approximated sufficiently well. Comparison of the runs with k=1, k=2, and k=3 (in pairs) eliminate the second possibility, that the overshoot is due to the model, and suggest that the time discretization is indeed the source of this error. This will be explored in the next experiment.

In any case, though aesthetically displeasing, the overshoot is of minor consequence. It is stable, and does not contaminate the profile behind the shock as the number of time steps increases, so it is a phenomenon associated with the shock front only. The notion of modelling shocks with discontinuities is useful only so long as it is easier to do than some other method; here, it is felt, a small, stable overshoot does not detract from the accuracy of the solution. If the details of the shock are of interest, we might have to move to a more appropriate physical model, including the effects of heat conduction and physical viscosity.

5.8 REMARKS ON MESH SELECTION

The mesh selection algorithm of COLSYS has, overall, performed well. However, there are some interesting questions raised about the way it has achieved the meshes it did. There were essentially three types of observed behaviour of the mesh selection algorithm. We will give examples of each.

i) Steady.

Where the size of the mesh remained more or less steady throughout the computation, from time step to time step (within a time step there is always some movement).

ii) Alternating.

Sometimes the mesh size would be steady for a few time steps, and then suddenly get large, and then go back to about its original size. This may be a phenomenon associated with our "forced mesh density" modification of DYNACOL.

iii) Wild.

This happened only on large time steps. As the solution was not behaving well, this type of behaviour, oscillating from large size meshes to small and in general being unpredictable, may be a good indicator of an incorrect size of time step.

In this dynamic context, it is difficult to find out just what the mesh selection algorithm is doing.

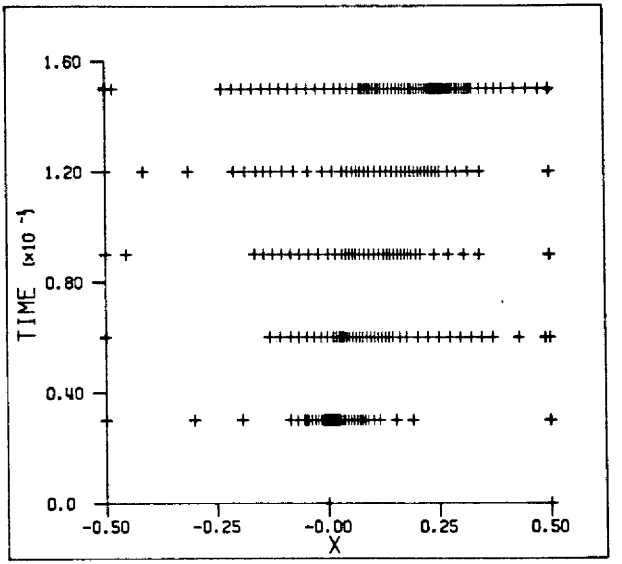


Figure 5: Mesh movement graph

Figure 5 shows, however, that the performance is quite robust, if erratic. There is no question about the appropriateness of the meshes shown here; the meshpoints are concentrated around the shock and contact discontinuity, there are a few in the rarefaction wave, and almost none outside the region of interest. (The boundaries of the interval are at -5 and 5; we are showing here only a small portion of the interval. There are usually not more than five or six meshpoints outside this portion.) However, it is possible to

conclude from Figure 5 that the mesh size is increasing with time, as it appears that the last mesh has significantly more points than the ones immediately preceding. The following table, showing the behaviour of the size of the mesh for several graphs, shows that the increase in sizes is likely a mesh doubling, which happens on occasion, but the size of the mesh can remain more or less constant, even when such a mesh doubling happens.

For the run with k=2, h=0.005, and $\lambda = 5. \times 10^{-4}$
sampling the mesh size at every fourth time step,

46 46 46 42 88 46 92

For the run with k=2, h=0.01, and $\lambda = 5. \times 10^{-4}$
sampling the mesh size at every third step

40 40 40 40 42

For the run with k=3, h=0.005, and $\lambda = 5. \times 10^{-4}$
sampling the mesh size at every step

48 48 46 48 48 50 54 100 52 52 54 58 54 54 100 60

For the run with k=3, h=0.01, and $\lambda = 5. \times 10^{-4}$
sampling the mesh size at every step

50 100 54 100 54 100 56 56 56

And for the long time run described next, with
h=0.005, $\lambda = 0.001$, and k=2, sampling at every time step

38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 40 36 42 76
40 72 50 52
52 100 100 52 52 52 54 54 50 52 54 100 52 52 52 96 96 52 52
58 54 52 100 52 52 52 52 100 50 50

Figure 6: Numbers of Mesh Points

In Figure 6 we see, in the last run, that a single run may exhibit all three types of behaviour. Just after the program starts up, for the first 16 time steps, the number of mesh points remains absolutely steady. Then, a period of turbulence, or wildness, followed by a (seemingly) stable alternating phase. It is not clear what the mesh selection algorithm is doing in this context. However, as noted before, the meshes seem to be good: it is just that it is not possible to predict in advance what the size or behaviour of the meshes will be.

5.9 LONG TIME RUN EXPERIMENT

We are now ready for the longer time interval tests. We are interested in knowing if the algorithm is capable of propagating the shocks over many time steps, or if we must take smaller time steps to achieve a satisfactory resolution. Since the code has a "restart" capability, allowing it to pick up a computation in the middle, we may start up some candidates for the long time run and compare them at intermediate times. We are also interested in the long-term behaviour of the mesh selection algorithm: does it settle down after the start to a more or less steady state? We would also like to observe the separation of the regions of concentration of the mesh points as the number of time steps increases.

As the overshoot in the profile for k=2 is due at least in part to the artificial viscosity λ, we will increase λ slightly, to produce a better profile. This also allows computation of the k=1 case, for comparison purposes. The parameters used for both runs are reproduced here.

k	= 1,2	
h	= 0.005	
λ	= 10^{-3}	
T_1	= 5. X 10^{-5}	tolerances were placed
T_2	= 5. X 10^{-3}	on all six components. T_1 is the tolerance placed on the components r, m, and E; T_2 is the tolerance on the 1^{st} derivatives.
t_{int}	= 0.15	For comparison of k=1 and 2 The run with the best profile will be continued to t_{end}
t_{end}	= 0.60	This is 120 time steps, or 90 after the intermediate t_{int}
initial mesh		uniform, of five subintervals.
sensitivity		regular.

The partial results are contained in the following table.

5.10 RESULTS OF COMPARISON EXPERIMENT

	k=1	k=2
t_{int}	0.15	0.15
start time (s)	130	181
total cpu (s)	328	593
time/step (s)	6.5	15.0

and the results were, for k=2 on the longer run to t_{end}

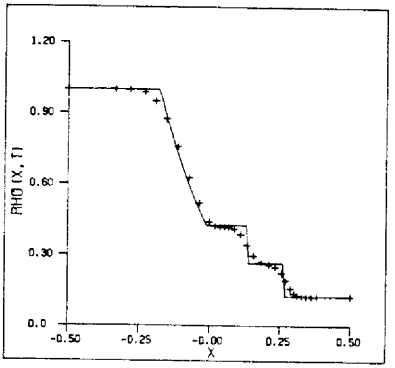
time taken in start = 181 cpu seconds

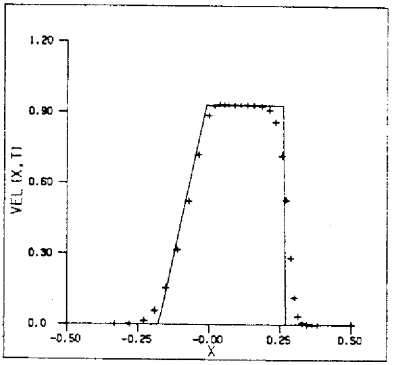
total time taken = 1955 cpu seconds

time per step = 16.3 cpu seconds

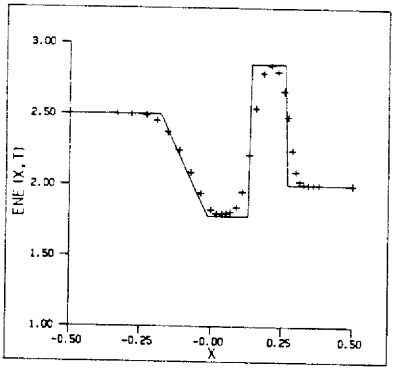
avg no. mesh points = 60

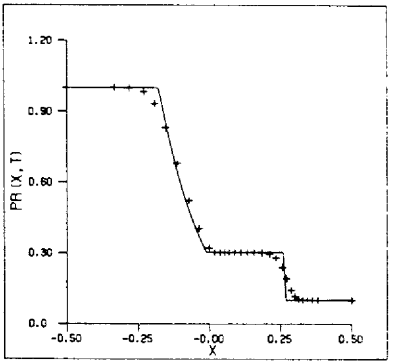
5.11 LONG TIME RUN SOLUTION PROFILES

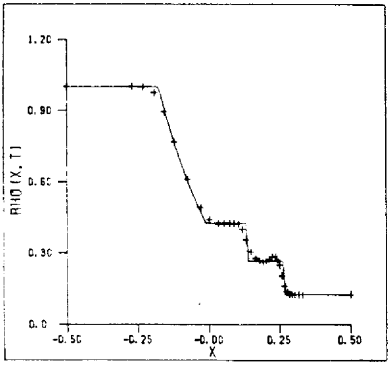


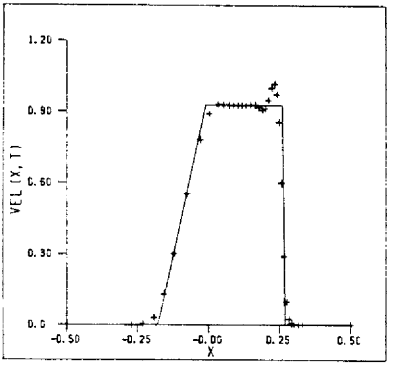


k=1 h=0.005 $\lambda=10^{-3}$ $t_{int}=0.15$

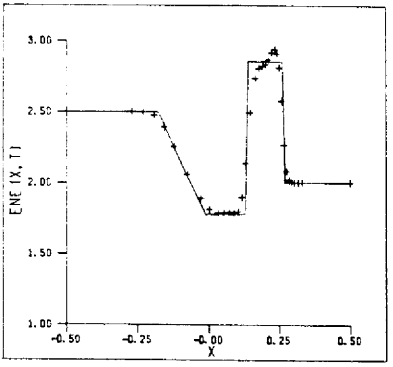


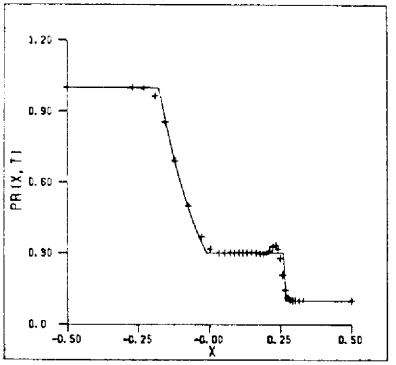


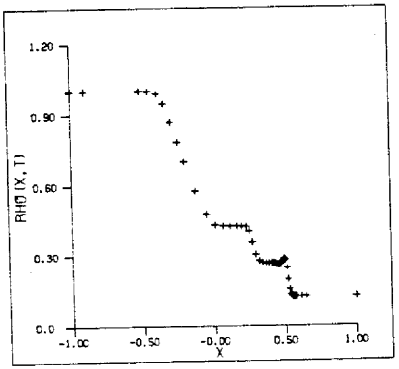


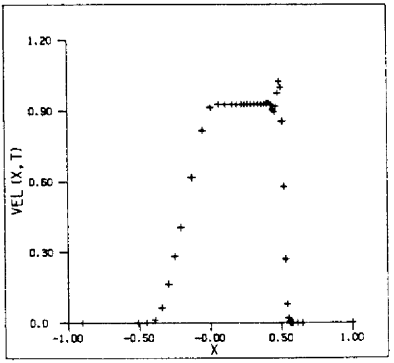


k=2 h=0.005 $\lambda=10^{-3}$ $t_{int}=0.15$

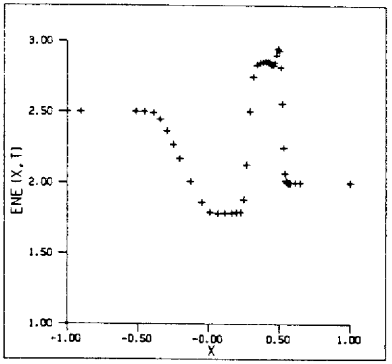


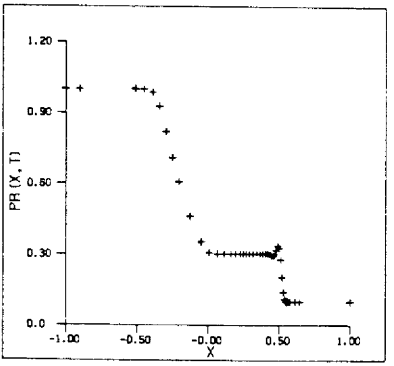


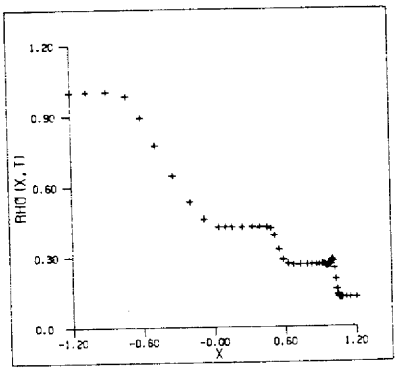


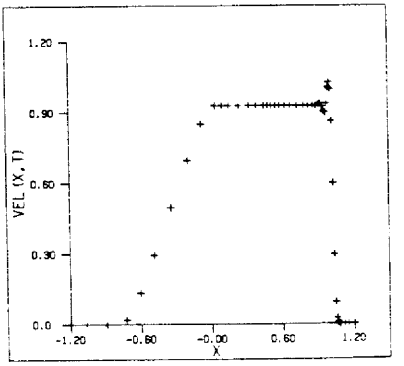


k=2 h=0.005 $\lambda = 10^{-3}$ t_{int}=0.30

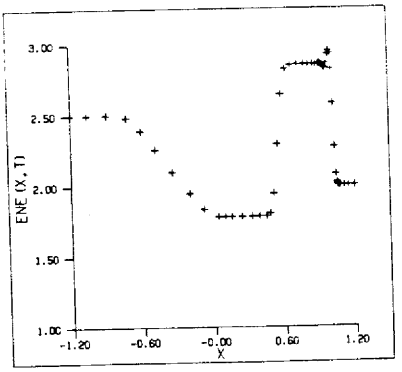


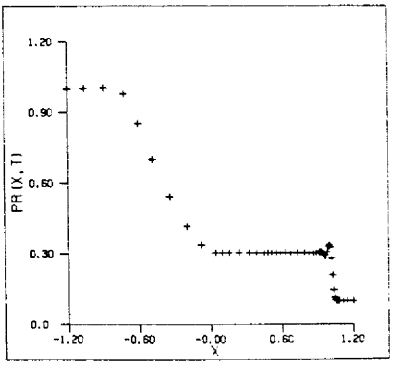






k=2 h=0.005 $\lambda = 10^{-3}$ $t_{end} = 0.60$





5.12 DISCUSSION

Note that in the runs comparing k=1 to k=2, we find that k=1 does indeed resolve the shock without the overshoot present in the k=2 case. However, the profiles are also noticeably diffused, and the details of the rarefaction wave are not as good as with k=2. This is a standard problem encountered with methods designed to solve problems such as our test problem. Those methods which resolve shocks well will not be as good on the smoother parts of the profile, and those methods which are good for smooth problems do not usually handle shocks well. The case k=2 here seems to be an adequate compromise; we have resolved the shock in a stable manner, while keeping the smoother parts of the profile accurate. The higher order method k=3 does not do so well with the shock, while not doing visibly better on the smooth parts. Thus the order k=2 was chosen for the long time run. The behaviour over a large number of time steps was of interest, and it is evident from the profiles exhibited that the method, with the parameters chosen, performs well. The shock remains steady, and moves at the correct speed. The rarefaction wave and the flat parts of the profile are quite accurate, and the mesh movement is clearly visible in the graphs presented.

Chapter VI

CONCLUSIONS

The longer interval experiment with the gas dynamics equations provides some answers to our most important questions. We see that this method can indeed solve hard problems, giving results that are quite comparable to even the best custom coded finite difference schemes. The random choice method, Glimm's method, and Godunov's method appear to be superior, in that they manage to resolve the shocks very sharply (in the case of Glimm's method, over one grid point) [33, 11]. However, as is pointed out in [11], Glimm's method will take on only the exact values, and is at its best, for the shock tube problem, and does not perform so well on others. For our method, this test problem represents an extreme test of the code's ability to resolve boundary layers.

However, the boundary value problem method here is not as simple as was hoped. There are many parameters to choose from in the operation of this method, which have been shown to be quite problem dependent. This gives flexibility, but also complexity. There may be a way to make "optimal" choices from many of these options, but this needs further study. We have here exhibited a problem (Burgers' equation with ν=0.02) where the higher order method with k=3 proved

to be superior, and a problem (the gas dynamics of a shock tube with $\lambda=10^{-3}$) where the method with k=2 proved superior. This result, together with the error analysis made in Chapter 3, suggest that the parameters need to be chosen in a problem dependent way.

In any case, with not too much development time, we were able to produce a code which solved a rather difficult test problem; it is to be expected, now that some of the groundwork has been done, that persons with a system of differential equations of the type considered here would be able to quickly develop a code along these lines and expect it to perform well.

The other major question was that of cost. Here we are hampered by the lack of cost estimates for other methods, so we cannot give any relative estimates of the cost. This is a serious lack, and further study on this question would be useful. We can, however, report the costs encountered, and give a qualitative judgement on the usefulness of the method in that context.

As noted in the last chapter, the long run on the gas dynamics problem took approximately 2000 cpu seconds, or approximately 33 cpu minutes. This is expensive, there is no doubt of it. The cost per step for k=2 was approximately 16 cpu seconds per step (or about 50 cents on the Waterloo IBM 4341). There are many suggestions for improving the efficiency of this approach. One is a variable time step, which

would allow the program to take precisely as large steps as was consistent with a given level of accuracy; another is to streamline COLSYS, and yet another is to (possibly) use an interpolation method for the calculation of $v^{(n)}(x)$. Notice that the time per step of the gas dynamics equation runs was significantly larger than the cost per step of the Burgers' equation; this is in part due to the increased number of components, but the method is sensitive to the amount of diffusion in the problem as well.

However, all things considered, it seems that this method has at least limited usefulness, in spite of the cost. As it is quick (to program) and robust, it may be used sparingly as a check on other, custom coded programs, or for preliminary or limited investigations. The generality of the method is of great use here; it would be very easy to construct a general purpose package for the solution of partial differential equations by this method. Indeed, DYNACOL is very nearly that now.

6.1 SUGGESTIONS FOR FURTHER STUDY

This study has raised many questions. I shall list a representative sample, though not all.

i) Is there a "best" way of approximating $U_t(x,t)$ for the purposes of this method? There are many possible choices here, none of which have been investigated. Some suggestions are the stiffly stable

Hermite methods, or an adaption of the Crank-Nicolson method, as mentioned previously [26, 21].

ii) In conjunction with the method, is a uniform time step necessary? It would seem simple to design an adaptive temporal mesh algorithm along the lines of those for initial value problems for this method. In the case of transient phenomena, this would provide large gains in efficiency.

iii) Is there an optimal method of calculating $v^{(n)}(x)$ (if such a thing exists in the chosen method)? Is the direct method really the best, here? Could we improve the cubic method by using splines in tension or taut splines [7]?

iv) Can we streamline COLSYS (or the other boundary value problem solvers) so that the mesh selection, solution of nonlinear equations, etc., is more appropriate to the partial differential equation context?

v) Can we find an optimal (or simply better) way of using the existing mesh selection algorithm? Perhaps some form of damped extrapolation would work well in this context, giving a sort of prediction-correction algortithm for the calculation of the meshes.

vi) What class of problems is this algorithm suited for? There are a few immediate generalizations to

the method as outlined. The restriction to second order problems was totally arbitrary, and in fact DYNACOL as it is written will solve problems of the form

$$C\,U_t = F(t, x, U, \frac{\partial U}{\partial x}, \frac{\partial^2 U}{\partial x^2}, \ldots, \frac{\partial^n U}{\partial x^n})$$

if we use a nonlinear solver to isolate $\frac{\partial^n U}{\partial x^n}$. This class of equations includes such equations of interest as the Koortewreg De-Vries equation, and many others. The generalization to more than one space dimension is difficult, and it is not likely that this approach will succeed. It is conceivable that this approach, using an elliptic partial differential equation solver rather than an ordinary boundary value problem solver, could generalize to more than one space dimension, but I have no idea how practical it would be.

Each of the above questions woud require a great deal of effort and time to study them.

6.2 SUMMARY

The method was shown to be robust, by exhibiting the solution to a severe test problem, the gas dynamics of a shock tube. The method is not as simple as was hoped, as it requires selection among many options, to which the behaviour of the method is quite sensitive, depending on the problem,

and the smoothness of the solution. The lower order methods, k=1 and k=2, were shown to perform better on the gas dynamics problem than k=3, while the reverse was true for Burgers' equation. The economics of the method are suggested to be at least of limited feasibility, depending on the application. Further study would be helpful, and suggestions for the kinds of things to look for were made.

BIBLIOGRAPHY

1. Alp, Sule; "Hopscotch Schemes for the solution of Partial Differential Equations". M.Phil Thesis. The University of Waterloo, 1981.

2. Ascher, U; Christiansen, J; Russell, R.D.; "COLSYS--A Collocation Code for Boundary Value Problems". Paper 12 in [10].

3. Ascher, U; Christiansen, J; Russell, R.D.; "Collocation Software for Boundary Value ODEs." ACM Transactions on Mathematical Software, vol. 7, no. 2, June 1981, pp 209-222.

4. Ascher, U; Christiansen, J; Russell, R.D.; "ALGORITHM 569 - COLSYS: Collocation Software for Boundary Value ODEs [D2]". ACM Transactions on Mathematical Software, vol. 7, no. 2, June 1981, pp 223-229.

5. Ascher, U; Christiansen, J; Russell, R.D.; "A collocation solver for mixed order systems of boundary value problems." Technical Report 77-13, Computer Science Department, University of British Columbia, 1977.

6. Ascher, U; Pruess, S; Russell, R.D.; "On Spline Basis Selection For Solving Differential Equations". Technical Report 81-4, Computer Science Department, University of British Columbia, 1981.

7. de Boor, Carl; A Practical Guide to Splines. Springer-Verlag, Applied Mathematical Sciences series vol 27, 1978.

8. de Boor, Carl; Swartz, Blair; "Collocation at Gaussian Points". SIAM Journal on Numerical Analysis, vol. 10, no. 4, September 1973, pp 582-606.

9. Cerutti, J.; "Collocation for systems of ordinary differential equations". Computer Science Technical Report 230, University of Wisconsin-Madison, 1974.

10. Childs, B. et al (Eds); Codes for Boundary Value Problems. Lecture Notes in Computer Science 76, Springer-Verlag, N.Y. 1979.

11. Colella, Phillip; "Glimm's Method for Gas Dynamics", SIAM Jounal of Scientific and Statistical Computing, vol. 3, no. 1, March 1982, pp 76-110.

12. Davis, Stephen F.; Flaherty, Joseph E.; "An Adaptive Finite Element Method for Initial-Boundary Value Problems for Partial Differential Equations". SIAM Jounal of Scientific and Statistical Computing, vol. 3, no. 1, March 1982, pp 6-27.

13. Daniel, James W.; "A Road Map of Methods for Approximating Solutions of Two-Point Boundary Value Problems". paper 1 in [10].

14. Deuflhard, Peter; "Nonlinear Equation Solvers in Boundary Value Problem Codes". paper 3 in [10].

15. Deuflhard, P.; "A relaxation strategy for the modified Newton method". in Conference Proceedings on Optimization and Optimal Control; Bulirsch, Oettli, and Stoer (eds.) Lecture Notes 477. 1975.

16. Foy, Linus Richard; "Steady state solutions of hyperbolic systems of conservation laws with viscosity terms". Communications on Pure and Applied Mathematics, vol. 17, 1964, pp 177-188.

17. Gear, C. William; Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall Series in Automatic Computation, N.J. 1972.

18. Geddes, K.O.; Gonnet, G.H.; Char, B.W.; "MAPLE Users Manual", preprint edition. Department of Computer Science, University of Waterloo. (May 1982).

19. Gottlieb, D.; Lustman, L.; Orzag, S.; "Spectral calculations of one-dimensional inviscid compressible flows". SIAM Jounal of Scientific and Statistical Computing, vol. 2, no. 3, September 1981, pp 296-310.

20. Graney, L.; Richardson, A.A.; "The numerical solution of non-linear partial differential equations by the method of lines". Journal of Computational and Applied Mathematics, vol. 7, no. 4., 1981, pp 229-236.

21. Keller, H.B. Personal communication.

22. Lax, P. D.; "Weak Solutions of nonlinear hyperbolic equations and their numerical computation". Communications on Pure and Applied Mathematics, vol. 7, 1954, pp 159-193.

23. Lax, P. D.; "Hyperbolic Systems of Conservation Laws II". Communications on Pure and Applied Mathematics, vol. 10, 1957, pp 537-566.

24. Lax, P.D; Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves. SIAM Regional Conference Series, Arrowsmith 1973.

25. Madsen, N.K., and Sincovec, R.F.; "PDECOL: General collocation software for partial differential equations". ACM Transactions on Mathematical Software, vol. 5, book 3, September 1979, pp 326-351.

26. Norsett, Syvert P.; "One-Step Methods of Hermite Type for Numerical Integration of Stiff Systems". BIT vol. 14, 1974. pp 63-77.

27. Pereyra, Victor; "PASVA3: An Adaptive Finite Difference Fortran Program for First Order Nonlinear, Ordinary Boundary Problems". paper 4 in [10].

28. Prenter, P.M.; Splines and Variational Methods. Wiley Interscience Series in Pure and Applied Mathematics, N.Y. 1975.

29. Russell, R.D.; Christiansen, J.; "Adaptive mesh selection strategies for solving boundary value problems". SIAM Journal on Numerical Analysis, vol 7, 1978, pp 59-80.

30. Russell, R.D.; "Mesh Selection Methods". paper 17 in [10].

31. Russell, R.D.; "Collocation for Systems of boundary value problems". Numerische Mathematik, vol. 23, 1974, pp 119-133.

32. Russell, R.D; Personal communication.

33. Sod, Gary A.; "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws". Journal of Computational Physics, vol. 27, 1978, pp 1-31.

34. Watts, H.A.; "Initial Value Integrators in BVP Codes". paper 2 in [10].

35. White, Andy; "Mesh Selection for Boundary Value Codes - A Workshop Summary". paper 22 in [10].

36. Whitham, G.B.; Linear and Nonlinear Waves. Wiley-Interscience series in Pure and Applied Mathematics, N.Y. 1976. N.Y. 1976.