PATH-FOLLOWING METHODS FOR BOUNDARY VALUE PROBLEMS AND THEIR APPLICATIONS TO COMBUSTION EQUATIONS

A thesis presented in partial fulfilment of the
requirements for the degree of
Master of Science in Mathematics
at Massey University

Miguel David Scotter

1993

Abstract

This thesis is primarily concerned with the numerical techniques involved in bifurcation analysis, in particular with the software package AUTO developed by Eusebius Doedel which performs this analysis on dynamical systems.

The techniques of AUTO are investigated and applied to a steady state heat equation. The chosen equation can be solved by analytical methods for some boundary conditions. Initially AUTO was successfully applied to such problems, which have analytical solutions confirming its reliability. The software was then used to solve dynamical system problems which do not have known analytical solutions. These problems necessitated a modification to AUTO for non-autonomous systems. The modified version of AUTO was shown to be successful in finding solutions to these problems.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Adrian Swift and Graeme Wake for their support and encouragement throughout the year. They contributed enormously to this project with many ideas and helpful suggestions.

I would also like to thank Kee Teo for abstract algebra discussions, Mark Byrne for computer discussions and Sarah for life discussions.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 BIFURCATION ANALYSIS

Bifurcation theory allows the analysis of a system (or model) with a *control parameter* which is fixed (in any single instance) but can vary depending on the system which is been modelled. For example, if the load on a vertical beam is greater than a certain amount, the beam will buckle causing a deflection (see figure 1.1). At the point of buckling, the solution structure changed. There is a solution branch where there is buckling, and a solution branch where there is none. The solution branch has *bifurcated* into two branches. The *control parameter* (λ) for this system is the load on the beam. It is fixed for any particular set of physical parameters, but if varied, the solution structure may change. This system could have many other variable parameters (for example the elasticity modulus of the beam); however if only one parameter is varied at a time then its effects may be seen without other parameter variation influencing the results. This is the technique used in *bifurcation analysis.*

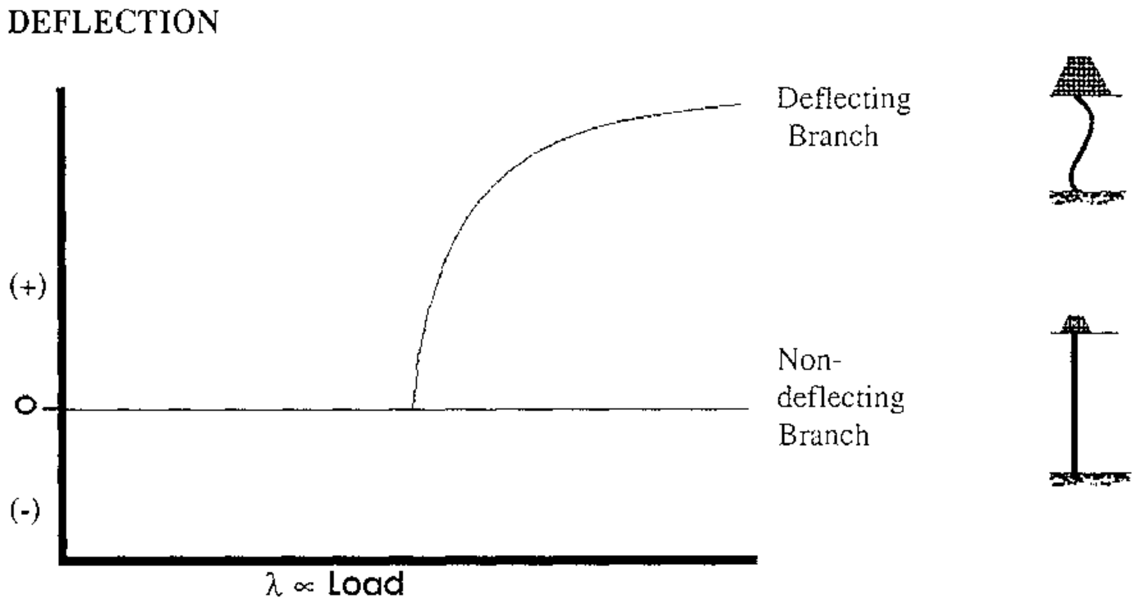


Figure 1.1: A bifurcation Graph for the load on a beam

The control parameter for a system is usually called the *distinguishing parameter*, *principle parameter* , or the *bifurcating parameter*, and will be denoted by λ. As the previous example has shown, changing λ changes the structure of the solution set.

BIFURCATION POINTS

Bifurcation analysis is the process of finding critical values of λ which change the solution structure. A *bifurcation graph* is generated to aid this task. A *bifurcation graph* plots λ against some *norm* of the solution (see figure 1.2). Solutions to the system are represented on this graph as curves or *branches*, where a *branch* is defined as a curve which can be parameterised by a single parameter. If, in a small neighbourhood of λ, there is a point λ_0 at which the number of solutions change (in every small neighbourhood), then this point is described as a *bifurcation point*. These points are important as a change in the number of solutions indicates a change in the state of the system.

The system will be defined as:

(1.1) $$F(u,\lambda)=0$$

where $u = [u_1 , u_2 , ..., u_n]'$, λ is a scalar and $F = [F_1, F_2, ... , F_n]'$ is a system of equations.

The *Jacobian* operator F_u for this system is defined as:

(1.2)
$$F_u = \begin{bmatrix} \frac{\partial F_1}{\partial u_1} & \frac{\partial F_1}{\partial u_2} & \cdots & \frac{\partial F_1}{\partial u_n} \\ \frac{\partial F_2}{\partial u_1} & \frac{\partial F_2}{\partial u_2} & \cdots & \frac{\partial F_2}{\partial u_n} \\ \cdots & \cdots & \cdots & \\ \frac{\partial F_n}{\partial u_1} & \frac{\partial F_n}{\partial u_2} & \cdots & \frac{\partial F_n}{\partial u_n} \end{bmatrix}$$

This system has a solution set:

(1.3)
$$S_\lambda = \{u \in \Re^n \mid F(u, \lambda) = 0\}$$

for a particular value of λ. A point λ_0 is a *bifurcation point* if $S_{\lambda_0} \neq \emptyset$, and there is

a $u_0 \in S_{\lambda_0}$ such that, for all sufficiently small neighbourhoods

$$U = \{ u_x \in \Re^n \mid \|u_x - u_0\| \langle \varepsilon \} \text{ and } V = \{ \lambda_x \in \Re \mid |\lambda_x - \lambda_0| \langle \delta \}:$$

there are two distinct solutions (u_1, λ_x) and $(u_2, \lambda_x) \in$ UxV (see reference [3]). That is there is more than one solution of x for a particular value of λ.

A bifurcation point on a solution branch can be sub-divided into one of two categories, a *limit point* and a *branching point* (see figure 1.2).

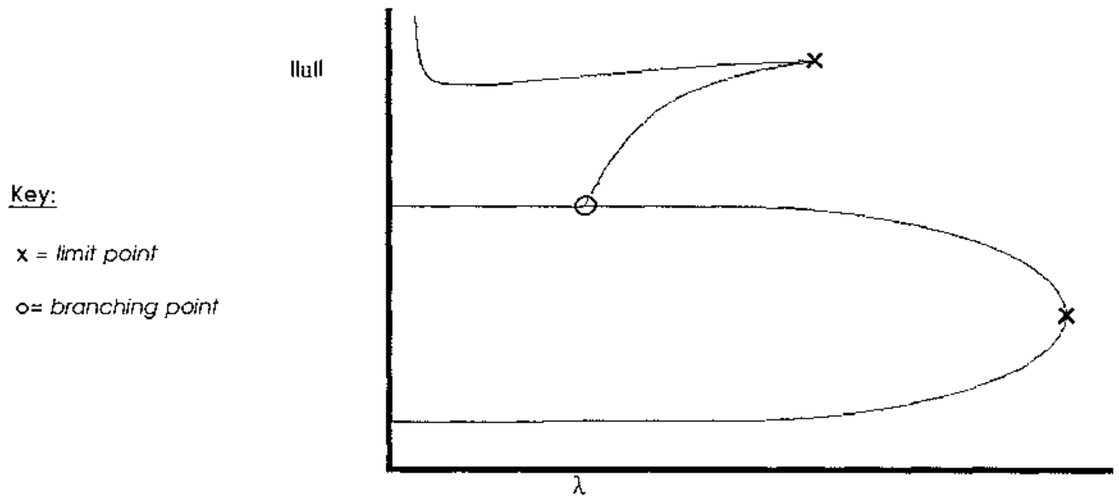


Figure 1.2: A Bifurcation Diagram

Limit point:

This is when the solution curve folds back on itself.

A *limit point* (λ_0 , u_0) is defined to be a bifurcation point which has solutions (λ_0, u_0) in the neighbourhood UxV such that :

either $\lambda_x \le \lambda_0$ for all $\lambda_x \in V$, **or** $\lambda_x \ge \lambda_0$ for all $\lambda_x \in V$.

Branching Point:

This can be defined as a bifurcation point which is **not** a *limit point*.

Note that any **single** branch can be parameterised using a single parameter t, giving a solution space $\{ (u(t), \lambda(t)) \mid a < t < b \}$. So an alternative definition of a limit point is a point $(u(t_0), \lambda(t_0))$ such that $\lambda'(t_0) = 0$.

Other conditions for bifurcation points can be found by introducing the Implicit Function Theorem:

Theorem 1.1: The Implicit Function Theorem (see [12], Page 78)

If

(i) F is continuously differentiable and

(ii) F_u has a continuous inverse at a point (u_a, λ_a) in solution space (1.3)

then for the neighbourhoods:

$U = \{ u_x \in \Re^n \mid \|u_x - u_a\| < \varepsilon \}$ and $V = \{ \lambda_x \in \Re \mid \|\lambda_x - \lambda_a\| < \delta \}$:

(a) F_u has a bounded inverse for all points $(u_x, \lambda_x) \in U \times V$.

(b) for all fixed $\lambda_x \in V$, the equation $F(u_x, \lambda_x) = 0$ has a unique solution $u_x \in U$. ■

This implies that, when F_u is non-singular, there are no bifurcation points. So a necessary (but not sufficient) condition for a bifurcation point is that F_u is singular. If Theorem 1.1 holds then a branch B of solutions can be defined:

(1.4) $$B = \{(u(\lambda), \lambda) \mid a < \lambda < b, F(u(\lambda), \lambda) = 0\}, \quad where\ a,b \in \Re$$

The solution $(0,\lambda)$ is called the *trivial branch*. This happens when $F(0,\lambda)=0$ for all values of λ (i.e $u(\lambda)=0$).

Theorem 1.2: (see [12] page 79)

If the *trivial branch* exists and $\lambda_0=0$ is an eigenvalue of the Jacobian F_u, then if λ_0 is an eigenvalue of odd multiplicity, λ_0 is a branching point from the trivial branch,

i.e. $\lambda_0 = 0$ is a root of $det(F_u(0) - \lambda_0 I) = 0$ which is of odd multiplicity (where $F_u(0)$ is the Jacobian evaluated at $u=0$). ■

Note: λ_0 and λ are different variables.

This is a **sufficient** condition for a bifurcation point. Note that if the multiplicity of the eigenvalue is not odd then it is still not known whether the point is a bifurcation point. If the trivial solution does not exist, or bifurcation points from other branches are wanted, then Theorem 1.2 cannot be applied. So this theorem only locates bifurcation points which are on the trivial branch. To find bifurcation points from other branches, the system has to be converted into this form by ***linearisation***:

> If a solution branch ($U = u(\lambda)$, λ) is **known** then equation (1.1) can be linearised such that:
>
> (1.5) $$H(h, \mu) = F(U+h, \mu) = 0$$
>
> where h is a vector $[h_1, h_2, ..h_n]'$, and $\mu = \lambda$ is the new bifurcation parameter (it is given a different symbol as λ is fixed by the solution branch U).
>
> $H(0, \mu) = 0$, for all μ, so Theorem 1.2 can be used to find bifurcation points.
>
> A bifurcation point $\mu_0=g(U)$ in system H can be related back to F by substituting μ_0 with λ and solving $\lambda=g(U)$ for λ.

Note that this requires an existing solution branch U.

Example 1.1:

Define F to be the system:

$$f_1(u_1,u_2,\lambda) = u_1^3 + u_2 - \lambda u_1 = 0$$

$$f_2(u_1,u_2,\lambda) = u_1^2 u_2 - u_1^3 - \lambda u_2 = 0$$

This system has the trivial solution branch (u', λ)= $(0,0,a)$, $a \in \Re$. But does it have any other branches? The implicit function theorem can be used to find **possible** bifurcation points along this branch.

The Jacobian F_u along the trivial branch is:

$$F_u = \begin{bmatrix} -\lambda & 1 \\ 0 & -\lambda \end{bmatrix}$$

This is singular when the determinant is zero, i.e when $\lambda=0$. So by Theorem 1.1 $\lambda=0$ is a possible bifurcation point. At that point F_u has a zero eigenvalue of multiplicity 2, which is even, so that Theorem 1.2 does not prove whether it is a bifurcation point or not.

However if some algebraic manipulation is done on the system, one finds that $\{(a, 0, \lambda)\ ,\ a > 0\}$ is **not** a solution and also $\{(0, b, \lambda),\ b > 0\}$ is not a solution so non-trivial solutions only exist if $u_1, u_2 \neq 0$. Eliminating λ gives:

$$u_2^2 = -u_1^4$$

which has only the trivial solution. So as there are no non-trivial solutions, $\lambda=0$ is not a bifurcation point. ■

Example 1.2:

Define F to be:

$$f_1(u_1,u_2,\lambda) = u_1^2 - \lambda u_1 = 0$$
$$f_2(u_1,u_2,\lambda) = u_2^2 - \lambda u_2 = 0$$

This system has the trivial solution branch. The Jacobian F_u is:

$$F_u = \begin{bmatrix} 2u_1 - \lambda & 0 \\ 0 & 2u_2 - \lambda \end{bmatrix}$$

The trivial branch $(0,0,a)$ has a possible bifurcation point when $\lambda=0$.

$F_u(0)$ has an eigenvalue of multiplicity 2 at that point, which is even, so Theorem 1.2 does not prove whether it is a bifurcation point or not. Clearly $\lambda=u_1$ and $\lambda=u_2$ are non-trivial solutions. So solution branches $(a,0,a)$ $(0,a,a)$ and (a,a,a) exist for $a \in \Re$, and $(0,0,0)$ is a bifurcation point. F_u is not singular along any of the non-trivial branches, so by Theorem 1.1, there are no **new** bifurcation points coming off these branches. The bifurcation graph for this system is shown in figure 1.3.

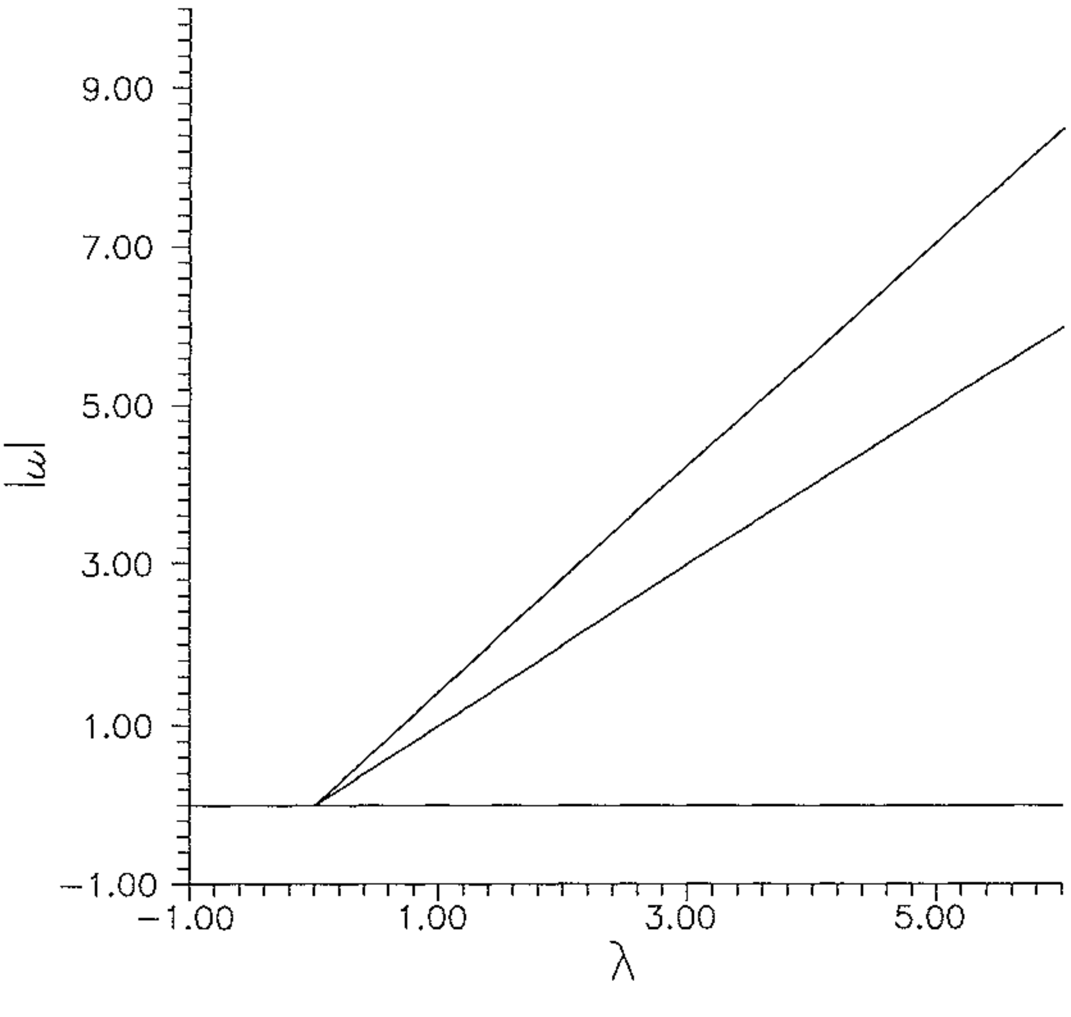


Figure 1.3: Bifurcation diagram for Example 1.2

■

Example 1.3:

Define F to be:

$$f_1(u_1,u_2,\lambda) = 16u_1 + 12u_1^3 + 24u_1u_2^2 - \lambda u_1 = 0$$

$$f_2(u_1,u_2,\lambda) = 12u_2 + 9u_2^3 + 18u_2u_1^2 - \lambda u_2 = 0$$

This system has the *trivial branch* as a solution.

For points along this branch, the Jacobian F_u becomes:

$$\begin{bmatrix} 16-\lambda & 0 \\ 0 & 12-\lambda \end{bmatrix}$$

This is singular when λ=16 and λ=12 where the nullspace is one-dimensional. So by Theorem 1.2 these points are bifurcation points. As λ has values both before and after these points (in their respective neighbourhoods) and also by Theorem 1.2, they can be sub-categorised as branching points (using the definition of a branching point defined earlier).

The branches can be found analytically and are:

$$u_a = \left[0,\ \pm\sqrt{\frac{1}{9}(\lambda-12)}\right]^t \textit{ and } u_b = \left[\pm\sqrt{\frac{1}{12}(\lambda-16)},\ 0\right]^t$$

To find bifurcation points along non-trivial branches, the system needs to be linearised about each solution branch by defining the system: $H(h,\mu) = F(U+h,\ \mu)$, where U is a non-trivial solution branch $u(\lambda)$. The Jacobian H_h along the solution branch (h=0) is:

(1.6)
$$H_h(0) = \begin{bmatrix} 16 + 36U_1^2 + 24U_2^2 - \mu & 48U_1U_2 \\ 36U_2U_1 & 12 + 27U_2^2 + 18U_1^2 - \mu \end{bmatrix}$$

For the first branch u_a (inserting it in the system H and making $\mu=\lambda$) the determinant is zero when:

$16 + 24U_2(\lambda) = \lambda$, and $12 + 27U_2(\lambda) = \lambda$ (as $U_1(\lambda)=0$)

This corresponds to $\lambda=12$ and $\lambda=9.6$. The first point is the existing bifurcation point from the trivial solution. The second point is not in the range of the solution space (of $\lambda>12$). So by theorem 1.1, there are no more bifurcation points along this branch.

The second branch u_b makes the determinant zero when $\lambda=16$ and when $\lambda=24$. The first eigenvalue is the point where this branch connects to the trivial solution. The second point is a possible bifurcation point. By theorem 1.2, $\lambda=24$ is a branching point from the branch u_b.

This new branch is:

$$\left[\pm\sqrt{\frac{(\lambda-9.6)}{21.6}},\quad \pm\sqrt{\frac{(\lambda-24)}{54}}\right]^t$$

It is more difficult to find branching points from this branch as U_1 and U_2 are both not zero.

So the analysis stops here, and a bifurcation graph of the results can be seen in figure 1.4.

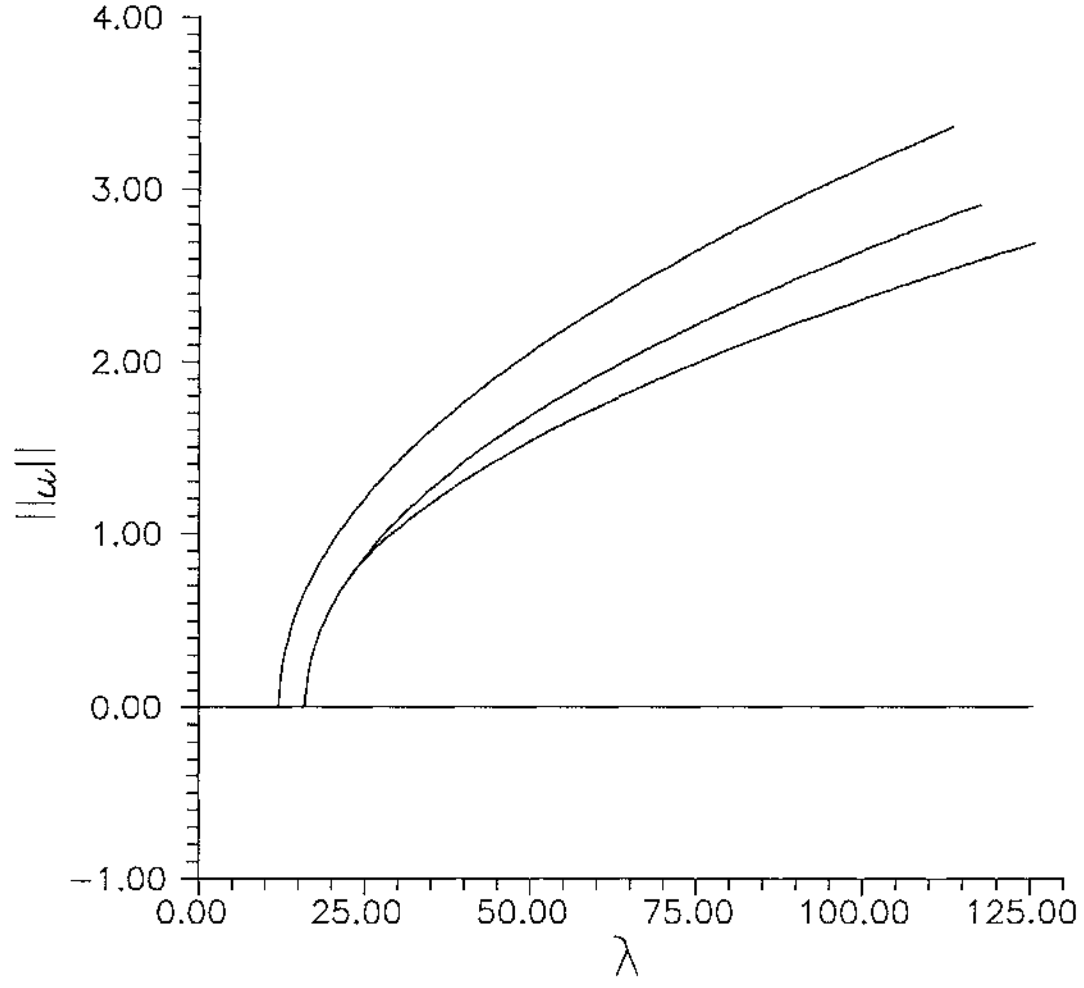


Figure 1.4: Bifurcation diagram for example 1.3

■

NORMS

Bifurcation graphs require a single value from a vector of many solution values to plot against the distinguishing parameter λ. The most commonly used value is the *Euclidean norm*:

$$\|u\|_2 = \sqrt{u_1^2 + u_2^2 + \dots + u_n^2}, \quad \textit{where } u = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix}^t$$

If a particular variable u_a is of interest, then $\|u\| = |u_a|$ can be used. If a bound is wanted on the variables the ∞-*norm* ($\|u\|_\infty$) can be used where:

$$\|u\|_\infty = \max\{|u_i| \ : i = 1, \dots, n\}.$$

Each definition will give a different bifurcation diagram. Some may not show bifurcation points, or may show branching points which do not exist.

For example, the point u=(1,2) has the same *Euclidean norm* and ∞-*norm* as the point u=(2,1) , so the two solutions from different branches will ***appear*** to be intersecting on a bifurcation graph. However the norm $\|u\| = |u_1|$ will graph the two points differently. Another example considers the curves $u_1 = (\lambda, \lambda+1)$ and $u_2 = (1-\lambda, \lambda+1)$ which intersect when $\lambda=½$. $\|u\|_\infty = \lambda+1$ for all positive λ on **both** solution curves. A bifurcation graph of λ versus $\|u\|_\infty$ will show an intersection of the two curves at all values of λ greater than zero. This problem is due to the difference between the dimension of the bifurcation graph, which is 2-dimensional, and the dimension of the system which it graphs. These two examples show that care must be taken when choosing a norm to ensure the solution curve is represented clearly on its bifurcation graph.

SYSTEMS OF DIFFERENTIAL EQUATIONS

The previous analysis is for algebraic systems of the form $F(u,\lambda)=0$. Now consider the situation when the model is a system of differential equations. i.e. $F(D(u),\lambda)=0$ where $D(u)$ is a nxn matrix of all the derivatives of u with respect to t, where u is a vector of independent variables, and t is a vector of dependent variables.

The solution space is:

$$\{\ (u(t),\lambda)\ |\ u\in\Re^n,\ t\in\Re^{Tn},\ \lambda\in\Re\},$$

For each value of λ, there exists one or more vector fields $u(t)$ of solutions.

The definitions for bifurcation points, limit points and branching points can still be applied to this system by defining a fixed point $u=u(t)$ which must satisfy the definitions for all values of $t\in\Re^{Tn}$.

Theorems 1.1 and 1.2 are not easily applied to F as the Jacobian F_u is undefined.

There is an analytical method for transforming a D.E. system into an algebraic system using an appropriate Green's integral, in which case the theorems can be used. This method will not be covered in any detail other than to say that it is very difficult to do for all systems except very simple ones. (see Gomez' thesis [5] and [9]), and then one still needs to find the function H from (1.5).

The existing norms can be applied at any fixed point u (usually the maximum). If there is only one dependent variable t defined over a fixed range $[0,T]$, then there is another norm which can be applied:

$$\|u\| = \sqrt{\frac{1}{T}}\sqrt{\int_0^T \left[u_1(t)^2 + u_2(t)^2 + \ldots + u_n(t)^2\right]dt}$$

This gives a *Euclidean*-type norm which encompasses the value for u over the whole interval, rather than just a single value, so all points have an 'effect' on the norm.

To make use of these theorems, one needs analytical solutions, which are only found in a very small subset of non-linear systems. Solution branches can be determined numerically, and Chapter 2 presents methods for doing this.

1.2 COMBUSTION MODELS

Cellulosic materials such as wood chips (or shavings or sawdust), hay or bagasse are stored in large piles under conditions which may vary in temperature and humidity. Under certain conditions, the material spontaneously combusts, possibly causing substantial damage.

For inert isotropic bodies obeying the Arrhenius Law the Fourier heat balance equation becomes (see [7] and [11]):

$$k\nabla^2 T + q\sigma A.e^{\left(\frac{-E}{RT}\right)} = C\frac{\partial T}{\partial t} \quad \textit{in the region } \hat{r}\in\hat{\Omega}\subseteq\Re^3,\ t>0$$

with boundary conditions:

$$k\frac{\partial T}{\partial n} + h(T - T_a) = 0 \ \textit{ on } \partial\hat{\Omega} \quad \textit{(assuming Newtonian Cooling)}$$

$$\frac{\partial T}{\partial x_i} = 0,\ \textit{when } x_i = 0\ \forall i$$

where $T(x,t)$ is the absolute temperature of a body at position vector x and time t, k is the thermal conductivity, and q, σ, and A are exothermicity/unit mass, density and frequency factors respectively. E is the activation energy of the oxidation reaction, C is the specific heat capacity, and R is the gas constant. $\nabla^2 T$ is the Laplacian operator. In the boundary conditions, h is the heat transfer coefficient, and T_a is the absolute ambient temperature.

The first boundary condition ensures that the temperature at the surface of the reactant is equal to the ambient temperature minus the effects of Newtonian cooling at the surface. In a symmetrically heated system, it is assumed that the temperature gradient at the centre is zero, resulting in the second boundary condition.

The steady state model for this system is:

$$k\nabla^2 T + q\sigma A . e^{\left(-\frac{E}{RT}\right)} = 0$$
$$k\frac{\partial T}{\partial n} + h(T - T_a) = 0 \quad \textit{on boundary } \partial\hat{\Omega}$$
$$\frac{dT}{dx_i} = 0, \textit{ when } x_i = 0 \quad \forall i$$

This system in symmetrical shapes can be converted to the dimensionless form (see [7] and [11]):

(1.7)
$$\frac{d^2u}{dr^2} + \frac{j}{r}\frac{du}{dr} + \lambda(a_0)e^{u} = 0$$
$$\frac{du}{dr}(0) = 0, \quad \frac{du}{dr}(1) + \mathrm{Bi}\,u(1) = 0$$

where $u = \dfrac{E(T - T_a)}{RT_a^2}$ is the dimensionless temperature excess, $r = \dfrac{\hat{r}}{a_0}$ is a dimensionless length-scale (same for objects of any size) in the interval [0,1], with a_0 representing an appropriate characteristic length such as the half-width of the body, Bi = ha_0/k (> 0) is the surface heat transfer coefficient (denoted as the Biot number), and λ is the Frank-Kamenetskii parameter [1] given by:

$$\lambda(a_0) = \frac{a_0^2 q\sigma A . e^{\left(-\frac{E}{RT_a}\right)}}{k\left(\dfrac{RT_a^2}{E}\right)}$$

The value for λ is crucial. Solutions only exist for certain values of λ. The maximum value of λ occurs at the point where the substance will combust spontaneously.

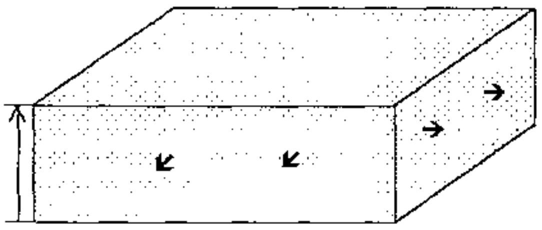
In physical terms if, for a given ambient temperature the substance radius is larger than a critical size, then there are no steady states for temperature in the substance, and it will combust spontaneously over time. This critical value for λ is denoted by λ_{crit}.

$\frac{d^2u}{dr^2}+\frac{j}{r}\frac{du}{dr}$ is the Laplacian Δu for *Class A* geometries in dimension j+1. i.e. geometries which have a single unit of measurement.

EXAMPLES

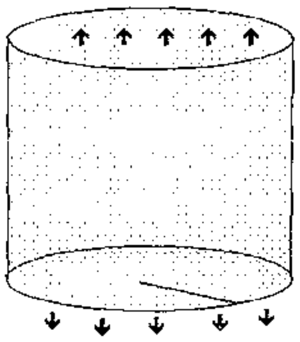
An infinite Slab (j=0):

$$\frac{d^2u}{dr^2}+\lambda e^{u}=0$$



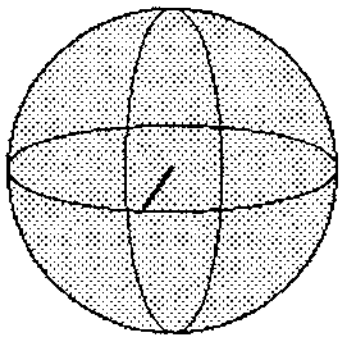
An infinite Cylinder (j=1)

$$\frac{d^2u}{dr^2}+\frac{1}{r}\frac{du}{dr}+\lambda e^{u}=0$$



A Sphere (j=2):

$$\frac{\partial^2u}{\partial r^2}+\frac{2}{r}\frac{\partial u}{\partial r}+\lambda e^{u}=0$$



There exists a well known analytical general solution to equation (1.7) when j=0 (see [7]):

(1.8)
$$u(r) = \ln(A) - 2\ln\left[\cosh\left(\sqrt{\frac{\lambda A}{2}}r + C\right)\right], \quad A>0$$

where A and C are constants. C must be zero to satisfy the first boundary condition. The norm, which will be plotted in the bifurcation diagram against λ, is the maximum value of u over $0 \leq r \leq 1$ denoted by u_{max}. Over this range the maximum value of u occurs when $r = 0$, so the norm u_{max} is:

$$u_{max} = u_0 = u\ (0) = \ln\ (A)$$

An implicit solution relating u_0 to λ, for $\lambda > 0$ is found by replacing $\ln(A)$ by the norm u_0, and using the other boundary condition:

(1.9)
$$\ln(\lambda) = \ln(2\alpha^2) - 2\ln\cosh(\alpha) - \frac{2\alpha\tanh(\alpha)}{\mathrm{Bi}}, \quad where\ \alpha = \sqrt{\lambda\frac{e^{u_0}}{2}}\ and\ \lambda>0.$$

When λ=0, u is the trivial solution $u(r)$=0 for all r. As u_0 and λ are unique for each value of α, the solution curve can be parameterised by α, giving a branch $(u_0(\alpha),\lambda(\alpha))$ of solutions.

The solution when Bi = 1 is plotted in figure 1.5.

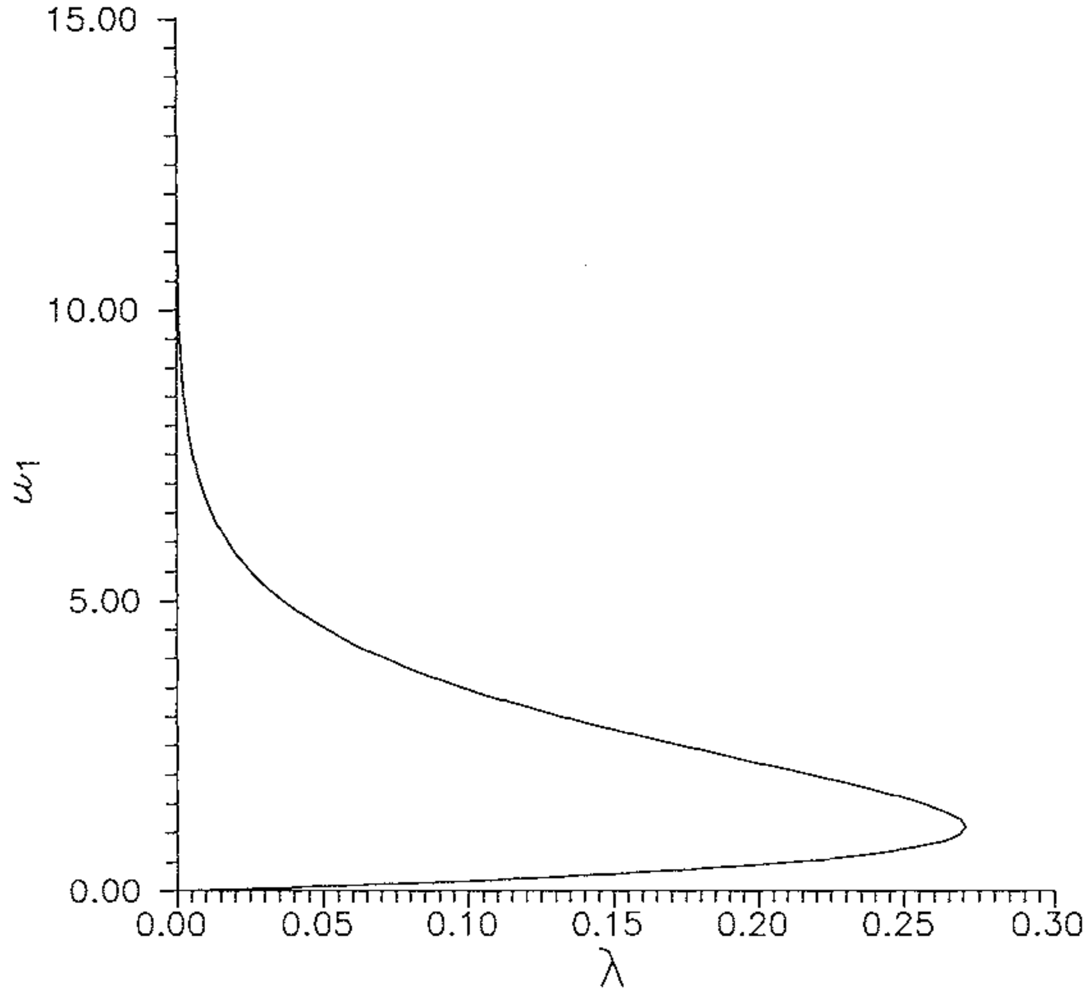


Figure 1.5: Bifurcation graph of system (1.7) with j=0 and Bi=1.

The maximum value of λ is at a limit point, λ_{crit}= 0.270671.

By the definition of a limit point (see section 1.1), a limit point is a point when $\lambda'(\alpha) = 0$, or equivalently $\frac{d(\ln\lambda)}{d\alpha} = 0$. Also as the entire solution space is parameterised by one parameter, there are no other branches, and therefore no branching points.

This gives an implicit equation for the co-ordinates of the limit point, for any given Biot number Bi:

(1.10)
$$\mathrm{Bi} = \frac{\alpha_{crit}\sinh(\alpha_{crit})\cosh(\alpha_{crit}) + \alpha_{crit}^2}{[1-\alpha_{crit}\tanh(\alpha_{crit})]\cosh^2(\alpha_{crit})}$$

where α_{crit} is the value of α when it is a limit point, from which the corresponding values $(u_0)_{crit}$ and λ_{crit} can be attained by using (1.9).

A singular solution of (1.10) occurs when the denominator is zero and Bi = ∞. This is called the Frank-Kamenetskii boundary condition, and corresponds to perfect heat transfer at the surface of the object. The last term of (1.9) becomes zero when Bi = ∞, and λ can be defined **explicitly** in terms of u_0:

$$\lambda = \frac{2}{e^{u_0}}[\cosh^{-1}\left(e^{\frac{u_0}{2}}\right)]^2$$

The limit point (when $\frac{d\lambda}{du_0}=0$) is 0.87846.

A general solution also exists for equation (1.7) when j=1 (the infinite cylinder):

(1.11)
$$u(r) = \ln\left[\frac{8A}{\lambda}(1+Ar^2)^2\right]$$

where A is a constant of integration. Similar analysis may be used on this to give an implicit equation for λ and u_0. From (1.11), $u_0 = u_{max} = \ln(8A/\lambda)$ and introducing $G = A$ results in:

$$\ln(\lambda) = \ln\left[\frac{8G}{(G+1)^2}\right] - \left[\frac{4G}{\mathrm{Bi}(G+1)}\right], \quad where\ G = \frac{\lambda}{8}e^{u_0}$$

The limit point ($(u_0)_{\text{crit}}$, λ_{crit}) occurs when $\frac{d(\ln\lambda)}{dG}=0$, and results in the implicit equation:

$$\text{Bi} = \frac{4G_{crit}}{(1-G_{crit}^2)}$$

When Bi=∞, an explicit solution for λ in terms u_0 exists:

$$\lambda = 8\left(e^{-\frac{u_0}{2}} - e^{-u_0}\right)$$

and λ_{crit} is 2.

A general solution for j=2 (or more generally for $j \neq \{0 \text{ or } 1\}$) has not been found. Chapter 3 discusses a numerical approach for solving (1.7) for any j and Bi.

THE SHAPE FACTOR

The examples so far are *Class A* shapes which have Laplacians which are of the form of equation (1.7). The Laplacian $\nabla^2 u = \frac{d^2u}{dr^2} + \frac{j}{r}\frac{du}{dr}$ can also be used to approximate non-*Class A* geometries. Boddington, Gray and Harvey [1] developed a technique for using (1.7) to model **any** shape possessing a point of symmetry by defining j as a *shape factor*:

(1.12)
$$j = 3\frac{R_0^2}{R_S^2} - 1$$

where R_S is the Seminov radius [1], and R_0 is the harmonic root-mean-square radius of the body:

$R_s = 3\dfrac{volume}{surface\ area}$, $\dfrac{1}{R_0^2} = \dfrac{1}{4\pi}\displaystyle\iint \dfrac{d\omega}{a^2}$ where $d\omega$ is the solid angle subtended at the centre O and a is the radius from O to the edge for given angle co-ordinates.

For example, a cube of volume $(2a)^3$ has $R_S = \dfrac{3(2a)^3}{6(2a)^2} = a$, and

$R_0 = a\sqrt{\dfrac{3}{1+\dfrac{2\sqrt{3}}{\pi}}} = 1.194a$ so the shape factor j for a cube of any size (as the a's cancel)

is 3.280 (using (1.12)).

The harmonic root-mean-square radius RA_0 for any **unit** *Class A* shape is defined (see [1]) as:

$$RA_0^{-2} = \frac{1}{3}(j+1)$$

A shape X is modelled by a *Class A* shape of radius ⅓$(j+1)$, where j is defined using R_s and R_0 from the original shape X. This results in solving a modified version of (1.7) which has a new parameter $\lambda(R_0)$ (see [1]):

$$\frac{d^2u}{dr^2} + \frac{j}{r}\frac{du}{dr} + \frac{1}{3}(j+1)\lambda(R_0)e^{u} = 0$$
$$\frac{du}{dr}(0) = 0, \quad \frac{du}{dr}(1) + Bi\,u(1) = 0$$

This system gives an approximate value of λ for shape X.

λ can be found for the **unit** shape X by scaling $\lambda(R_0)$ by R_0^{-2}.

So (1.7) can model a shape of unit size by:

1 -> Calculating the shape factor j from (1.12)

2 -> Solving (1.7) to get solution points (u_i , λ_i)

3 -> Scaling λ_i:

$$\lambda_i^* = \frac{3\lambda_i}{(j+1)RA_0^2}$$

Table 1.1 has some more examples of shape factors. RA_S is easy to find, but RA_0 is not. (There is list of formulae of R_0 for simple geometries in [1]).

Table 1.1: Example Shape factors

Geometry	RA_0	RA_s	j
Infinite Slab	3	3	0
Rectangular Parallelepiped (ratio 1:10:10)	1.731	5/2	0.438
Infinite Cylinder	1.225	3/2	1
Infinite Square Rod	1.354	3/2	1.444
Rectangular Parallelepiped (ratio 1:1:10)	1.354	10/7	1.694
Sphere	1	1	2
Equicylinder	1.115	1	2.729
Cube	1.194	1	3.280
Regular Tetrahedron	0.537	0.408	4.178

Boddington, Gray and Harvey [1] showed that, for convex bodies, the shape parameter has values lying between 0 (the infinite slab) and 4.178 (the regular tetrahedron).

If $N = j + 1$ is defined as the dimension of a sphere, then $\frac{d^2\theta}{dr^2} + \frac{N-1}{r}\frac{d\theta}{dr}$ can be thought of as the Laplacian of the N-dimensional sphere in spherical co-ordinates. So, using the shape factor j, an object can be modelled as a j+1 dimensional sphere, where j is a positive real number.

RESULTS FOR ARBITRARY j

A phase plane analysis of (1.7) by Wake [14], has shown that for $1<j<9$, as u increases, λ converges to λ_∞ where :

(1.14)
$$\lambda_\infty = 2(j-1)e^{-\frac{2}{\mathrm{Bi}}}$$

He also showed that there are an infinite number of solutions to (1.7) when $\lambda=\lambda_\infty$. This implies that the bifurcation curve has a damped oscillation about the line $\lambda=\lambda_\infty$, with amplitude decreasing as u increases. As there are infinitely many oscillations about a vertical line, there are also infinitely many limit points.

No analytical solution to (1.7) has been found for arbitrary j and Bi. The rest of this thesis discusses techniques for numerically solving (1.7) to generate bifurcation graphs for any j or Bi value. The analytical results of this Chapter are used to test the accuracy of the numerical results.

2. NUMERICAL TECHNIQUES

This Chapter describes the numerical techniques used by AUTO to perform bifurcation analysis on autonomous boundary value problems (BVP's). Numerical methods for bifurcation analysis of algebraic systems are presented followed by a method of discretising BVP's by *collocation.* Then it is shown how bifurcation analysis of BVP's can be performed by applying the techniques used for algebraic systems on the resulting discretised system. Finally, the numerical techniques of AUTO are extended for non-autonomous boundary value problems.

2.1 ALGEBRAIC SYSTEMS

This section looks at bifurcation analysis of algebraic systems of the form:

(2.1) $$0 = F(u,\lambda), \quad F(.,.) : \Re^{n+1} \to \Re^{n}, \ \lambda \in \Re, \ u \in \Re^{n}$$

where λ is a free parameter. If F is continuously differentiable and F_u has a continuous inverse at all solution points ($u(s)$, $\lambda(s)$) along a branch for $a \leq s \leq b$, then the Implicit Function Theorem (see chapter 1) ensures that there exists a smooth continuous branch where F_u is non-singular, and the solution has a parameterisation of the form $\{ \ (u(s), \lambda(s) \ ) \mid a \leq s \leq b \ \}$ where $\lambda(s)=s$. This is called a *simple branch.*

Then from some starting value $x_o = (u_0, \lambda_o) = (u(s_0), \lambda(s_0))$ a curve can be generated:

(2.2) $$F\left(u(s_0), \lambda(s_0)\right) = 0 \quad \Rightarrow \quad F_u \frac{du}{ds} + F_\lambda \frac{d\lambda}{ds} = 0$$

and, as $\lambda = s$; $\frac{d\lambda}{ds} = 1$, so that (2.2) becomes:

(2.3) $$F_u\left(u(\lambda_0), \lambda_0\right)\frac{du}{d\lambda} = -F_\lambda\left(u(\lambda), \lambda\right)$$

An *Euler predictor with Newton Corrector* [8] finds a predicted value from:

(2.4) $$\text{Solve} \quad F_u^0 u_\lambda^0 = -F_\lambda^0 \quad \text{to get} \quad u_\lambda^0$$

(2.5) $$\text{where} \quad F_u^0 = F_u(u_0, \lambda_0) \quad \text{and} \quad F_\lambda^0 = F_\lambda(u_0, \lambda_0)$$

and u_λ^0 is used to predict:

(2.6) $$\begin{gathered} u^0(\lambda_0 + \delta\lambda) = u(\lambda_0) + \delta\lambda u_\lambda^0 \\ \lambda^0 = \lambda_0 + \delta\lambda \end{gathered}$$

where $\delta\lambda$ is an arbitrary step size. This gives an initial value (u^0, λ^0) for the next point which can then be improved by the Newton Corrector:

(2.7) $$\text{solve} \quad F_u^\nu \delta u^\nu = -F^\nu, \quad \text{to get} \quad \delta u^\nu$$

and an improved u^ν is then given by:

(2.8) $$u^{\nu+1} = u^\nu + \delta u^\nu$$

where ν is incremented from $\nu=0$, and the sequence (2.7) and (2.8) is repeated until δu is sufficiently small.

Note that the Implicit function theorem (see Chapter 1) implies that, when the Jacobian F_u is not singular, there cannot possibly be any bifurcation points as part (b) implies that there can only be one solution for a particular value of λ. Hence the parameterisation $s=\lambda$ will work in all cases where there are **no** bifurcation points.

What if there are bifurcation points? F_u is singular at these points (by the Implicit function theorem) and when F_u is singular, (2.4) and (2.7) have no solutions.
If there is a branching point λ_{br}, then there is a possibility that the procedure could work, as there are points $\lambda < \lambda_{br}$ and $\lambda > \lambda_{br}$ which have non-singular Jacobians (by the implicit function theorem), and so can be found by this method. So, as long as the **exact** point λ_{br} is not found, the Jacobian is not singular and the procedure is likely to work. But if there is a limit point λ_l, then there is no solution for either $\lambda < \lambda_l$ or $\lambda > \lambda_l$, so the method will fail to find the points on the branch which are past the limit point.

If another parameterisation is used (instead of s=λ) which does not make F_u singular at limit points, then the Euler predictor with Newton Corrector method could still be used. It should be possible to find a single-parameter parameterisation that has a non-singular Jacobian at limit points because the curve is still unique.

If a normalisation $N(u,\lambda,s)$=0 is added to the system (2.1), a new system $P(u,\lambda,s)$ is formed, where u and λ are now dependent on the parameter s:

(2.9) $$x(s) = \begin{bmatrix} u(s) \\ \lambda(s) \end{bmatrix}, \quad P(x(s), s) = \begin{bmatrix} F(u, \lambda) \\ N(u, \lambda, s) \end{bmatrix} = 0$$

then the Jacobian becomes:

$$P_x(x(s),s) = \begin{bmatrix} F_u(u(s), \lambda(s)) & F_\lambda(u(s), \lambda(s)) \\ N_u(u(s), \lambda(s), s) & N_\lambda(u(s), \lambda(s), s) \end{bmatrix}$$

If this is non-singular then the *Euler predictor with Newton Corrector* method will work. Note that F_u is singular at bifurcation points but P_x might not be.

The following theorem about partitioned matrices from linear algebra is helpful in the context of the previous material.

__Theorem 2.1__ (see [8])

For a matrix $M=\begin{bmatrix} A & B \\ C & D \end{bmatrix}$, where A is an $n \times n$ matrix, B is a $n \times 1$, C is a $1 \times n$, and

D is a 1×1 matrix, if NS denotes nullspace, and CS denotes column space:

(1) If A is non-singular then M is non-singular iff $D - C\,A^{-1}\,B$ is non-singular

(2) If A is singular and dim NS(A) $= 1$, then M is non-singular iff dim CS(B)=1, CS(B)$\cap$CS(A)=0, dim CS(C)=1, and NS(A)$\cap$NS(C)=0

(3) If A is singular and dim NS(A) > 1 then M is singular. ■

A parameterisation which is a function of the arc-length of the curve will make it possible for P_x to be non-singular around limit points, even though F_u is singular. This is the idea behind *Pseudo-arclength parameterisation.*

__Pseudo-Arclength Parameterisation__

The arc-length $s(t)$ for the system (2.1) with solution ($u(t)$, $\lambda(t)$) is:

$$s(t) = \int_a^t \sqrt{\|\dot{u}(\tau)\|^2 + |\dot{\lambda}(\tau)|^2}\, d\tau$$

where τ is a variable of integration. This implies that $[s'(t)]^2 = \|\dot{u}\|^2 + |\dot{\lambda}|^2$.

If t=s, then the system is parameterised as the arclength such that:

$$\dot{u}\circ\dot{u}+\dot{\lambda}\circ\dot{\lambda}-1=0$$

where $\circ$ is the inner product:

$$x\circ x=\theta_1^2x_1x_1+\theta_2^2x_2x_2+\ldots+\theta_n^2x_nx_n,\ where\ x,\theta\in\Re^n$$

Note that θ acts as a scaling factor to scale each co-ordinate of the x vector.

The problem with this parameterisation is that if an iterative method is used (like the one mentioned previously) $\dot{u}$ and $\dot{\lambda}$ are not known. But they can be approximated using previous points as $\dfrac{u-u_{j-1}}{\Delta t}$ and $\dfrac{\lambda-\lambda_{j-1}}{\Delta t}$ respectively, giving:

$$\|u-u_{j-1}\|^2+\|\lambda-\lambda_{j-1}\|^2=\Delta s^2.$$

Note that the points u and λ are unknown in this equation. To make this equation linear in terms of the unknowns, Herbert Keller [8] used a different approximation to the arc-length:

(2.10)
$$\begin{aligned}u_j-u_{j-1}&=\dot{u}_{j-1}\Delta s+O(|\Delta s|^2)\\ \lambda_j-\lambda_{j-1}&=\dot{\lambda}_{j-1}\Delta s+O(|\Delta s|^2)\end{aligned}$$

so the arc-length Δs is approximated by:

(2.11)
$$N(u,\lambda,s)\equiv\theta_u^2(u-u_{j-1})\cdot\dot{u}+\theta_\lambda^2(\lambda-\lambda_{j-1})\cdot\dot{\lambda}-\Delta s=0.$$

where the direction vector $(\dot{u}, \dot{\lambda})^t$ is approximated using previous points (as in (2.10) except the direction vector is scaled):

(2.12)
$$(\dot{u},\dot{\lambda}) \approx \frac{1}{\Delta s}(u_{j-1} - u_{j-2}, \lambda_{j-1} - \lambda_{j-2})$$
$$\text{and} \quad \theta_u^2\|\dot{u}\|^2 + \theta_\lambda^2 |\dot{\lambda}|^2 = 1$$

θ_u and θ_λ are weights to scale the co-ordinates u and λ. Most of the time θ_u=θ_λ=1. Δs is the stepsize along the branch. The direction vector is normalised so the size of the change in u and λ is proportional to Δs, which is fixed, so s can be used as a step size. (2.11) has dependent variables u and λ and Δs which change at every step depending on the value of u and λ at previous steps.

Initially there are no previous points, so the first direction vector $(\dot{u}_0, \dot{\lambda}_0)^t$ is approximated as the nullspace of $[F_u \mid F_\lambda]$ which is found by solving:

(2.13)
$$[F_u \vdots F_\lambda]\begin{bmatrix}\dot{u}_0\\ \dot{\lambda}_0\end{bmatrix} = 0$$

Provided F_u is non-singular, (2.13) has a one-dimensional nullspace. A single vector is then chosen by scaling the direction vector as in (2.12). This will only work if the starting point is not a bifurcation point (i.e. F_u is non-singular).

Theorem 2.2 (see [8])

If every solution point x_j on a branch is defined as either:

(a) *a regular point* (F_u is non-singular) or

(b) *a normal limit point*

condition 1: **dim** NS(F_u)=1
condition 2: $F_\lambda \cap$ CS(F_u) = 0

Then the Euler-Newton Corrector method will numerically generate the branch.

■

The proof of Theorem 2.2 is as follows:

The Jacobian of P is:

$$P_x = \begin{bmatrix} F_u & F_\lambda \\ \theta_u^2 \dot{u}^t & \theta_\lambda^2 \dot{\lambda} \end{bmatrix}$$

At a regular point F_u is non-singular and $\dot{\lambda}_j$ is not zero (as (2.2) is satisfied and F_u is non-singular), then

$\frac{\dot{u}_j}{\dot{\lambda}_j} = -(F_{u_j})^{-1} F_{\lambda_j}$ (by using (2.2))

and part (a) of Theorem 2.1 is satisfied, so P_x is non-singular. If x_j is a *normal limit point*, then $\lambda_j = 0$ (by (2.2) and condition 2), implying $\dot{u}_j \in NS(F_{u_j})$.

Using condition 1 as well implies $\dot{u}_j \notin CS(F_{u_j}{}^t)$.

This combined with conditions 1 and 2 mean that part 2 of Theorem 2.21 is satisfied, hence P_x is non-singular. This proves that (2.9) with parameterisation N defined in (2.11) will pathfollow using the Euler predictor with Newton Corrector around regular limit points. ■

Another way of defining a regular limit point is to define it as a point where the Jacobian P_x is non-singular. But it is still possible for P_x to be singular, for example when dim NS(F_u)>1 or when $F_\lambda \in$ CS(F_u).

At a branching point, there is no **unique** direction vector $[\dot{u},\dot{\lambda}]^t$. Equation (2.2) must be satisfied so this means that the solution to (2.2) has to have at least 2 parameters (i.e. is of dimension 2):

$$[F_u \vdots F_\lambda]\begin{bmatrix}\dot{u}\\ \cdot\\ \dot{\lambda}\end{bmatrix}=0$$

This means that either NS(F_u)>1 or $F_\lambda \in$ CS(F_u). So by Theorem 2.1, the Jacobian P_x is always singular at branching points.

If this happens, then it may be possible to continue past the point by 'jumping' forward, if there is a smooth arc of solutions x(s) for $s_a<s<s_b$, of which only $P_x(s_0)$ is singular. One method is to adjust the stepsize so the corrector converges to a point which is not a bifurcation point using 2 previous points which are not bifurcation points. These points will always exist if P_x is only singular at one discrete point, in the neighbourhood of s.

So this leads to a pathfollowing algorithm based on the pseudo arc-length parameterisation and the Euler predictor with Newton corrector.

PSEUDO-ARCLENGTH PATHFOLLOWING ALGORITHM

INITIALISATION

$$\boxed{\begin{array}{c} a=0,\quad x_a=\vec{x}(s_a)=[u_a,\lambda_a]\,(\textit{not a bifurcation point}) \\ \dot{x}=NS\{F_{u_a} \vdots F_{\lambda_a}\},\ \text{such that,}\ \|\dot{x}\|=1 \end{array}}$$

$\downarrow$

If $a > 0$ then $\dot{x}$ is defined as:

$$\boxed{\begin{array}{c} (\dot{u},\dot{\lambda}) \approx \frac{1}{\Delta s}(u_a-u_{a-1},\lambda_a-\lambda_{a-1}) \\ \textit{such that}\ \theta_u^2\|\dot{u}\|+\theta_\lambda^2|\dot{\lambda}|^2=1 \end{array}}$$

$\downarrow$

PREDICTOR

$$\boxed{x_{a+1}^{(0)}=x_a+(\Delta s)\dot{x}}$$

$\downarrow$

CORRECTOR

$$\boxed{N_{a+1}^{(\nu)}=\theta_u^2(u_{a+1}^{(\nu)}-u_a)\cdot\dot{u}+\theta_\lambda^2(\lambda_{a+1}^{(\nu)}-\lambda_a)\cdot\dot{\lambda}-\Delta s}$$

$$\boxed{J_{a+1}=\begin{bmatrix} (F_u)_{a+1}{}^{(\nu)} & (F_\lambda)_{a+1}{}^{(\nu)} \\ \theta_u^2\dot{u}^t & \theta_\lambda^2\dot{\lambda} \end{bmatrix}}$$

$\downarrow$

Solve:

$$\boxed{J_{a+1}\,\boldsymbol{\delta}x=-\begin{bmatrix} F_{a+1}^{(\nu)} \\ N_{a+1}^{(\nu)} \end{bmatrix}}$$

to get δx then define:

$$x_{a+1}^{(\nu+1)}=x_{a+1}^{(\nu)}+\delta x$$

$\downarrow$

let $\nu = \nu + 1$

$\downarrow$

if $\nu > \nu_{\max}$ or $\boxed{\frac{\|\Delta x\|_\infty}{1+\|x\|_\infty}<\zeta}$ then repeat corrector

$\downarrow$

Adapt stepsize (see next page)

$\downarrow$

if $\nu<\nu_{\max}$ then let $a = a + 1$

The corrector iteration is stopped either if it has converged (the predicted relative error is less that ζ) or if it fails to converge after v_{max} iterations. Note that the relative error has a 1 in the denominator to prevent division by zero. If it fails to converge, the previous point is used again with a different (smaller) step size.

ADAPTIVE STEPSIZE

The speed of convergence of the Newton corrector iteration gives an indication about the accuracy of the initial Euler approximation, and as this approximation is proportional to the stepsize, then the convergence of the Newton corrector can be used to indicate whether to increase or decrease the step size Δs. If the Newton corrector converges in fewer than v_{min} iterations, then the stepsize is increased, and if it would require more than v_{max} iterations to converge then the stepsize is decreased. With an *adaptive stepsize*, the algorithm is less likely to fail or to be too slow, because the stepsize acts as a buffer for change in the solution structure. It also can be used to avoid 'landing' on branching points.

DETECTION OF BIFURCATION POINTS

A bifurcation point is detected by defining a function $q(x)$ (where $x = [u, \lambda]^t$), which has a zero at this point. A change of sign in $q(x)$ indicates the presence of a zero nearby. A more accurate value of the zero can be obtained using the secant method:

$$x_{v+1} = x_v - \frac{x_v - x_{v-1}}{q(x_v) - q(x_{v-1})} q(x_v)$$

where v is iterated from 1 until the process converges. $q(x)$ is defined for limit points and branching points as follows.

Limit Points

A limit point occurs when $\lambda'(s)=0$, and changes sign before and after. $\lambda'(s)$ can be determined accurately by solving the system:

$$F_u[u'(t)] + F_\lambda[\lambda'(t)] = 0$$
$$\dot{u}[u'(t)] + \dot{\lambda}[\lambda'(t)] = 1$$

The first equation comes from (2.2), and the second equation is the Pseudo-arclength parameterisation. The solution vector $[u',\lambda']^t$ is normalised so λ' is proportional to the distance away from the limit point (when $\lambda'=0$), so that:

$$q_1(x) = \frac{1}{\|[u',\lambda']^t\|}\lambda'$$

λ' is the horizontal-direction of the curve which must change sign at a limit point. So $q_1(x)$ is a suitable function for the root finder.

Branching points:

The Jacobian P_u is singular at branching points, So that its determinant is zero at a branching point:

$$q_2(x) = \det(P_x)$$

i.e. F_u has a zero eigenvalue. If this eigenvalue is of odd multiplicity, then the sign of the determinant will change at the branching point, and also by Theorem 1.2 the point is guaranteed to be a bifurcation point. If this eigenvalue is of even multiplicity, then the point is a possible bifurcation point, and the sign does not change. These points will not be detected as bifurcation points in AUTO. Note also that as P_x is not singular at limit points (by Theorem 2.2), only branching points will be detected. So $q_2(x)$ is an appropriate root finder for the detection of branching points.

CONTINUATION ON A NEW BRANCH

When one or more branching point(s) are located on a branch, the new branches can be generated, by starting at a branching point $x_0 = [u_0, \lambda_0]'$ and choosing a new direction vector.

Any direction vector must be in the null space of $[F_u^0 \mid F_\lambda^0]$ (by using equation (2.2)), using the convention: $F_u^0 = F_u$ evaluated at x_0.

If the branching point has NS(F_u)=m and $F_\lambda \in$ CS(F_u), then $\{\phi_0, \phi_1, \ldots, \phi_m\}$ can be defined such that:

$$\text{NS}(F_u^0) = a_1\phi_1 + a_2\phi_2 + \ldots + a_m\phi_m$$

and

$$F_u^0\phi_0 + F_\lambda = 0.$$

This can be substituted back into (2.2) to get:

$$F_u^0(\dot{u} + \dot{\lambda}\phi_0) = 0.$$

So $\dot{u}$ is a linear combination of the ϕ_j 's :

(2.14) $$\dot{u}_0 = \sum_{j=0}^{m} \alpha_j \phi_j, \quad where \ \alpha_0 = \dot{\lambda}_0.$$

The double derivative of F (or 'double' Jacobian) is:

(2.15) $$F_u^0\ddot{u} + F_\lambda^0\ddot{\lambda}_0 = -w, \ where \ w = \left[F_{uu}^0\dot{u}_0\dot{u}_0 + 2F_{u\lambda}^0\dot{u}_0\dot{\lambda}_0 + F_{\lambda\lambda}^0\dot{\lambda}_0\dot{\lambda}_0\right]$$

As $F_\lambda^0 \in$ CS(F_u^0), w must also be in the column space of F_u^0. So the direction vector must have the form (2.14) with w in the column space of F_u^0 (see [8]).

This provides sufficient information to compute the direction vectors, but is a very time-consuming process.

If one direction vector $\dot{x}_0$ is known, another direction vector can be approximated by x' :

(2.16)
$$F_x^0 x_0' = 0$$
$$\dot{x}_0 \cdot x_0' = 0$$

The first equation ensures that x' is in the null space of F_x^0 and the second equation makes x' orthogonal to $\dot{x}_0$. This is not a good approximation if the branches intersect at a very small angle, but if the stepsize is small enough, then the approximation works in most cases. The orthogonality condition also makes it less likely for the same branch to be traversed.

The *Pseudo-arclength Pathfollowing Algorithm* can traverse the new branch by starting at the branching point and using x' as the new starting direction.

2.2 BOUNDARY VALUE PROBLEMS

Any Autonomous Boundary Value Problem can be transformed into a system of first order differential equations. These are systems of the form:

$$\frac{du}{dt}=f(u(t),\lambda),\quad t\in[0,1],\ u(.),F(.,.):\Re^{n+1}\to\Re^{n},\ \lambda\in\Re \qquad (2.17)$$
$$\textbf{with boundary conditions } b_i(u(0),u(1),\lambda)=0,\quad i=1,2,\ldots,n_b$$

Collocation is one technique that can be used to discretise this D.E. into the form of equation (2.1).

DISCRETISATION

If we define a mesh:

$\{\ 0=t_0<\ t_1<t_2<\ldots\quad <t_N=1\ \}$ where $\Delta t_j=t_{j+1}-\ t_j$, $(0\le j\le N\text{-}1)$

then a submesh can also be defined for the interval $[t_j\ ,\ t_{j+1}\ ]$:

$$\left\{t_j,\ \ t_{j+\frac{1}{m}\Delta t_j},\ \ t_{j+\frac{2}{m}\Delta t_j},\ \ldots,\ \ t_{j+1}\right\} \qquad (2.18)$$

so there is a grid of *mN* sub-intervals over the interval [0, 1]. The points inside these sub-intervals are called *Collocation points.*

If an arbitrary function $f(t)$ is known at *mN* points on the grid, then $f(t)$ can be approximated over the interval $[t_j\ ,\ t_{j+1}\ ]$ by $\quad \mathrm{p}_j(t)=\sum_{i=0}^{m} w_{j,i}(t)f(t_{j+\frac{i}{m}})$ (cf. [2])

where the Lagrangian polynomials $w_{j,i}(t)$ are given by:

$$w_{j,i}(t)=\prod_{k=0,k\neq i}^{m}\frac{t-t_{j+\frac{k}{m}}}{t_{j+\frac{i}{m}}-t_{j+\frac{k}{m}}} \qquad (2.19)$$

As u is an n-dimensional vector, it is approximated over the interval $[t_j , t_{j+1}]$ by n collocation equations:

(2.20)
$$u(t)=\begin{bmatrix}u_1(t)\\u_2(t)\\\vdots\\u_n(t)\end{bmatrix}\approx p_j(t)=\begin{bmatrix}p_j^{(1)}(t)=\sum_{i=0}^{m}w_{j,i}(t)\,u^{(1)}_{j+\frac{i}{m}}\\p_j^{(2)}(t)=\sum_{i=0}^{m}w_{j,i}(t)\,u^{(2)}_{j+\frac{i}{m}}\\\vdots\\p_j^{(n)}(t)=\sum_{i=0}^{m}w_{j,i}(t)\,u^{(n)}_{j+\frac{i}{m}}\end{bmatrix}$$

where $p_j\,(t)$ is now a vector, and $u^{(a)}_{j+\frac{i}{m}}\approx u_a\!\left(t_{j+\frac{i}{m}}\right)$ is unknown.

To use $p_j\,(t)$ as an approximation for $u(t)$, it is required to satisfy (2.17) at discrete points $\{z_{j,i}\}$ in the interval $[t_j , t_{j+1}]$. So (2.17) becomes:

(2.21)
$$\frac{dp_j}{dt}(z_{j,i})=f(p_j(z_{j,i}),\,\lambda),\quad i=1,\dots,m,\ \ j=0,1,\dots,N-1$$

and the boundary conditions (for points $u_0^{(a)}$ and $u_N^{(a)}$ on the boundary):

(2.22)
$$b_i(u_0,u_N,\lambda)=0,\ i=1,2,\dots,n_b$$

(as $u_0 = p_0\,(0)$, and $u = p_{N-1}\,(1)$)

where

(2.23)
$$\frac{dp_j^{(a)}}{dt}(t)=\sum_{i=0}^{m}w'_{j,i}(t)\,u^{(a)}_{j+\frac{i}{m}}\quad\text{and}\quad w'_{j,i}(t)=\frac{\sum_{l=1}^{m}\left(\prod_{k=0,k\neq i,k\neq l}^{m}t-t_{j+\frac{k}{m}}\right)}{\prod_{k=0,k\neq i}^{m}t_{j+\frac{i}{m}}-t_{j+\frac{k}{m}}}$$

Note that $w'_{j,i}\,(t)$ is found by applying the product rule on (2.19).

Points $\{z_{j,i}\}$ are zeroes of the *m*th degree *Legendre* polynomial relative to that subinterval. These are the best points to choose if the function is 'polynomial' shaped.

So the D.E. is replaced by a series of (mN) algebraic systems (one for each mesh-point in the interval [0,1]) in terms of the variables $u^{(\omega)}_{j+\frac{i}{m}}$ and λ. Expanding (2.21):

$$(2.24) \qquad G_{ji} \equiv f\left(\begin{bmatrix} \sum_{k=0}^{m} w_{j,k}(z_{j,i}) u^{(1)}_{j+\frac{k}{m}} \\ \sum_{k=0}^{m} w_{j,k}(z_{j,i}) u^{(2)}_{j+\frac{k}{m}} \\ \vdots \\ \sum_{k=0}^{m} w_{j,k}(z_{j,i}) u^{(n)}_{j+\frac{k}{m}} \end{bmatrix}, \lambda\right) - \begin{bmatrix} \sum_{k=0}^{m} w'_{j,k}(z_{j,i}) u^{(1)}_{j+\frac{k}{m}} \\ \sum_{k=0}^{m} w'_{j,k}(z_{j,i}) u^{(2)}_{j+\frac{k}{m}} \\ \vdots \\ \sum_{k=0}^{m} w'_{j,k}(z_{j,i}) u^{(n)}_{j+\frac{k}{m}} \end{bmatrix} = 0, \qquad \begin{array}{l} i = 1,\dots,m \\ j = 0,\dots,N-1 \end{array}$$

and

$$(2.25) \qquad b_i(u_0, u_N, \lambda) = 0, \; i = 1,2,\dots,n_b$$

Combining (2.24) and (2.25) gives:

$$(2.26) \quad F\left(\left\{u^{(\omega)}_{j+\frac{i}{m}}\right\}_{\forall i,j,\omega}, \lambda\right) = \left[G_{01} \; G_{02} \cdots G_{0m} \; G_{11} \; G_{12} \cdots G_{1m} \cdots G_{(N-1)1} \; G_{(N-1)2} \cdots G_{(N-1)m} \; b_1 \cdots b_{n_b}\right]^t = 0$$

This is a system of the form $F(u,\lambda) = 0$. So the *pseudo-arclength pathfollowing method* (see previous section) can be used on this system. The individual linear systems G_{ji} are not independent of each other as the coupled endpoints variables u_j and u_{j+1} occur in more than one system.

The **pseudo-arclength equation** changes as there is a new variable t which is continuous over [0,1].

(2.27)
$$\theta_u^2\int_0^1 (u_{a+1}(t)-u_a(t))\cdot\dot{u}_a(t)\,dt+\theta_\lambda^2(\lambda_{a+1}-\lambda_a)-\Delta s=0$$

This integral equation can be discretised using Newton-Cotes formulae over the same interval $[t_j,\ t_{j+1}]$, with sub-intervals $[\ t_{j+i/m},\ t_{j+(i+1)/m}\ ]$. Note that the points are equally spaced.

(2.28)
$$\int_0^1 (u_{a+1}(t)-u_a(t))\cdot\dot{u}_a(t)\,dt\approx\sum_{i=0}^{n}\varpi_i\left(u_{a+1}(t_i)-u_a(t_i))\cdot\dot{u}_a(t_i)\right)$$

where ϖ_i are derived from the coefficients for 3rd 4th ,... 8th order Lagrange polynomials.

(2.29)
$$\varpi_i=\sum_a^b\left[\prod_{j=0,j\neq k}^{n}\frac{(x-x_j)}{(x_i-x_j)}\right]$$

The order (n) of the numerical integration formula is the same as the number of collocation points, as the same points are used.

So the **pseudo arc-length** equation becomes:

(2.30)
$$N=\theta_u^2\sum_{j=0}^{N-1}\sum_{i=0}^{m}\left[\varpi_{j,i}\sum_{k=0}^{n}\left\{\left(\left(u^{(k)}_{j+\frac{i}{m}}\right)_{(a+1)}-\left(u^{(k)}_{j+\frac{i}{m}}\right)_{(a)}\right)\left(\dot{u}^{(k)}_{j+\frac{i}{m}}\right)_{(a)}\right\}\right]+\theta_\lambda^2(\lambda_{a+1}-\lambda_a)\dot{\lambda}_0-\Delta s=0$$

Where $\left(\dot{u}^{(k)}_{j+\frac{i}{m}}\right)_{(a)}$ is approximated using the last 2 steps as in (2.12), and $\left(u^{(k)}_{j+\frac{i}{m}}\right)_{(a)}$ and

λ_a is known as well, leaving $\left(u^{(k)}_{j+\frac{i}{m}}\right)_{(a+1)}$ and λ_{a+1} as the unknown variables.

Solution paths of the D.E. (2.17) can now be found by using the **pseudo arc-length Pathfollowing algorithm** with the modified **pseudo arc-length** equation N (2.30) and F (2.26).

PARTIAL DERIVATIVES NEEDED FOR THE JACOBIAN

Using the chain rule on f in (2.21):

(2.31)
$$\frac{\partial f(p_j(z_{j,i}),\lambda)}{\partial u^{(a)}_{j+\frac{i}{m}}} = \frac{\partial f(p_j(z_{j,i}),\lambda)}{\partial p_j^{(a)}(z_{j,i})} \cdot \frac{\partial p_j^{(a)}(z_{j,i})}{\partial u^{(a)}_{j+\frac{i}{m}}}$$

and as $p_j^{(a)}(z_{j,i}) = u_a(z_{j,i})$:

(2.32)
$$f_{p_j^{(a)}(z_{j,i})} = f_{u_a}(p_j(z_{j,i}),\lambda)$$

so applying (2.31) and (2.32):

(2.33)
$$\frac{\partial G_{lk}}{\partial u^{(a)}_{j+\frac{i}{m}}} = \begin{cases} f_{u_a}(p_j(z_{j,i}),\ \lambda)\sum\limits_{k=0}^{m} w_{j,k}(z_{j,i}) - \sum\limits_{k=0}^{m} w'_{j,k}(z_{j,i}), & j=l \ \textit{or} \ (j=l+1 \ \textit{and} \ i=0) \\ 0, & \textit{otherwise} \end{cases}$$

and similarly:

(2.34)
$$\frac{\partial b_h}{\partial u^{(a)}_{j+\frac{i}{m}}} = \begin{cases} (b_h)_{u_a}(p_j(z_{j,i}),\lambda)\sum\limits_{k=0}^{m} w_{j,k}(z_{j,i}), & (j=0 \ \textit{and} \ i=0) \ \textit{or} \ (j=N-1 \ \textit{and} \ i=m) \\ 0, & \textit{otherwise} \end{cases}$$

(2.35)
$$\frac{\partial G_{ji}}{\partial \lambda} = f_\lambda(p_j(z_{j,i}),\lambda) \qquad \frac{\partial b_i}{\partial \lambda} = (b_i)_\lambda(u_0,u_N,\lambda)$$

f_{u_a}, f_λ, $(b_h)_u$, and $(b_h)_\lambda$ can be defined analytically (before starting) by partially differentiating the equations defined in (2.17).

If this is not possible (or is not very easy) then they can be generated numerically by differencing.

$$\frac{\partial f}{\partial u_a} = \frac{f(u + \Delta e_a, \lambda) - f(u - \Delta e_a, \lambda)}{2\Delta} \quad , \quad \textit{where} \quad \begin{array}{l} e_1 = [1,0,0,...,0]^t \\ e_2 = [0,1,0,...,0]^t \\ \quad\quad \textit{etc} \end{array}$$

where Δ is a very small number (see Appendix B for size of Δ in AUTO)

and similarly:

$$\frac{\partial f}{\partial \lambda} = \frac{f(u, \lambda + \Delta) - f(u, \lambda)}{\Delta} \qquad \frac{\partial b_h}{\partial u_N} = \frac{b_h(u_0, u_1 + \Delta e_a, \lambda) - b_h(u_0, u_1 - \Delta e_a, \lambda)}{2\Delta}$$

$$\frac{\partial b_h}{\partial \lambda} = \frac{b_h(u_0, u_N, \lambda + \Delta) - b_h(u_0, u_N, \lambda)}{\Delta} \qquad \frac{\partial b_h}{\partial u_0} = \frac{b_h(u_0 - \Delta e_a, u_1, \lambda) - b_h(u_0 + \Delta e_a, u_1, \lambda)}{2\Delta}$$

The pseudo-arclength equation N (2.30) has partial derivatives:

$$\frac{\partial N}{\partial u^{(a)}_{j+\frac{i}{m}}} = \theta_u^2 \, \omega_{j,i} \, \dot{u}^{(a)}_{j+\frac{i}{m}} \qquad \frac{\partial N}{\partial \lambda} = \theta_\lambda^2 \, \dot{\theta}_0$$

<u>**Example of a Newton Corrector equation**</u>

Consider a 2 dimensional system (n=2) with 2 boundary conditions (n_b=2) , with 3 collocation points (m = 3), and 3 mesh intervals (N=3).

The Jacobian is a $mnN + n_b + 1$ = 21X21 matrix and the Newton Corrector equation is:

(2.36)
$$\begin{bmatrix}
\frac{\partial G_{01}}{\partial u_0} & \frac{\partial G_{01}}{\partial u_{0+\frac{1}{3}}} & \frac{\partial G_{01}}{\partial u_{0+\frac{2}{3}}} & \frac{\partial G_{01}}{\partial u_1} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial G_{01}}{\partial \lambda} \\
\frac{\partial G_{02}}{\partial u_0} & \frac{\partial G_{02}}{\partial u_{0+\frac{1}{3}}} & \frac{\partial G_{02}}{\partial u_{0+\frac{2}{3}}} & \frac{\partial G_{01}}{\partial u_1} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial G_{02}}{\partial \lambda} \\
\frac{\partial G_{03}}{\partial u_0} & \frac{\partial G_{03}}{\partial u_{0+\frac{1}{3}}} & \frac{\partial G_{03}}{\partial u_{0+\frac{2}{3}}} & \frac{\partial G_{01}}{\partial u_1} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial G_{03}}{\partial \lambda} \\
0 & 0 & 0 & \frac{\partial G_{11}}{\partial u_1} & \frac{\partial G_{11}}{\partial u_{1+\frac{1}{3}}} & \frac{\partial G_{11}}{\partial u_{1+\frac{2}{3}}} & \frac{\partial G_{11}}{\partial u_2} & 0 & 0 & 0 & \frac{\partial G_{11}}{\partial \lambda} \\
0 & 0 & 0 & \frac{\partial G_{12}}{\partial u_1} & \frac{\partial G_{12}}{\partial u_{1+\frac{1}{3}}} & \frac{\partial G_{12}}{\partial u_{1+\frac{2}{3}}} & \frac{\partial G_{12}}{\partial u_2} & 0 & 0 & 0 & \frac{\partial G_{12}}{\partial \lambda} \\
0 & 0 & 0 & \frac{\partial G_{13}}{\partial u_1} & \frac{\partial G_{13}}{\partial u_{1+\frac{1}{3}}} & \frac{\partial G_{13}}{\partial u_{1+\frac{2}{3}}} & \frac{\partial G_{13}}{\partial u_2} & 0 & 0 & 0 & \frac{\partial G_{13}}{\partial \lambda} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial G_{21}}{\partial u_2} & \frac{\partial G_{21}}{\partial u_{2+\frac{1}{3}}} & \frac{\partial G_{21}}{\partial u_{2+\frac{2}{3}}} & \frac{\partial G_{21}}{\partial u_3} & \frac{\partial G_{21}}{\partial \lambda} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial G_{22}}{\partial u_2} & \frac{\partial G_{22}}{\partial u_{2+\frac{1}{3}}} & \frac{\partial G_{22}}{\partial u_{2+\frac{2}{3}}} & \frac{\partial G_{22}}{\partial u_3} & \frac{\partial G_{22}}{\partial \lambda} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial G_{23}}{\partial u_2} & \frac{\partial G_{23}}{\partial u_{2+\frac{1}{3}}} & \frac{\partial G_{23}}{\partial u_{2+\frac{2}{3}}} & \frac{\partial G_{23}}{\partial u_3} & \frac{\partial G_{23}}{\partial \lambda} \\
\frac{\partial b_1}{\partial u_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial b_1}{\partial u_4} & \frac{\partial b_1}{\partial \lambda} \\
\frac{\partial b_2}{\partial u_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial b_2}{\partial u_4} & \frac{\partial b_2}{\partial \lambda} \\
\frac{\partial N}{\partial u_0} & \frac{\partial N}{\partial u_{0+\frac{1}{3}}} & \frac{\partial N}{\partial u_{0+\frac{2}{3}}} & \frac{\partial N}{\partial u_1} & \frac{\partial N}{\partial u_{1+\frac{1}{3}}} & \frac{\partial N}{\partial u_{1+\frac{2}{3}}} & \frac{\partial N}{\partial u_3} & \frac{\partial N}{\partial u_{3+\frac{1}{3}}} & \frac{\partial N}{\partial u_{3+\frac{2}{3}}} & \frac{\partial N}{\partial u_4} & \frac{\partial N}{\partial \lambda}
\end{bmatrix} \delta x = -\begin{bmatrix} G01 \\ G02 \\ G03 \\ G11 \\ G12 \\ G13 \\ G21 \\ G22 \\ G23 \\ b1 \\ b2 \\ N \end{bmatrix}$$

where $\frac{\partial}{\partial u_{j+\frac{i}{m}}}$ represents $\begin{bmatrix} \frac{\partial}{\partial u^{(1)}_{j+\frac{i}{m}}} & \frac{\partial}{\partial u^{(2)}_{j+\frac{i}{m}}} \end{bmatrix}$ and G_{ji} has 2 equations.

This Jacobian is close to a block-diagonal matrix. When solving this system advantage can be taken of this structure by using the *method of condensation of parameters.* This method uses Gauss elimination with partial pivoting. The matrix is subdivided and sections eliminated separately. This method is illustrated very well in [4].

BIFURCATION POINT DETECTION

The discretised system can be treated as a large Algebraic system, so the zero-functions discussed in 2.1, can be used on this system to find accurate locations of *limit points* and *branching points.*

ADAPTIVE MESH

The interval $[t_j,\ t_{j+1}]$ (from 2.18) does not have to be the same size for all j. If the solution curve $x(t)$ has small curvature in an interval $[a, b]$, then the interval size can be greater than a region where the curvature is greater. This process distributes the approximation error in a more uniform manner.

Divided differences are used to get an indicator of the change in each interval $[t_j,\ t_{j+1}]$. For example if there are m=3 points in each interval, the Divided Difference Table is:

$$
\begin{array}{llll}
u_{j+0} & & & \\
 & (u_{j+1/3} - u_j)/(1/3)\Delta t_j & & \\
u_{j+1/3} & & (u_{j+2/3} - 2\, u_{j+1/3} + u_j)/(2/3^2)\Delta t_j^2 & \\
 & (u_{j+2/3} - u_{j+1/3})/(1/3)\Delta t_j & & (u_{j+1} - 3\, u_{j+2/3} + 3\, u_{j+1/3} - u_j)/(2/3^2)\Delta t_j^3 \\
u_{j+2/3} & & (u_{j+1} - 2\, u_{j+2/3} + u_{j+1/3})/(2/3^2)\Delta t_j^2 & \\
 & (u_{j+1} - u_{j+2/3})/(1/3)\Delta t_j & & \\
u_{j+1} & & &
\end{array}
$$

Figure 2.1: Divided Difference Table for m=3

The term $DD_j^{(3)} = (u_{j+1} - 3\, u_{j+2/3} + 3\, u_{j+1/3} - u_j)/(2/3^2)\Delta t_j^3$ is the 3rd divided difference. A fourth divided difference can be gained by combining the 3rd divided difference from consecutive intervals $[t_j,\ t_{j+1}]$ and $[t_{j+1},\ t_{j+2}]$:

(2.37)
$$DD_j^{(4)} \approx \frac{DD_{j+1}^{(3)} - DD_j^{(3)}}{\frac{1}{2}(\Delta t_j + \Delta t_{j+1})}$$

Over the unit interval, if $DD_j^{(4)} > 1$ then the change in u is too great; if $DD_j^{(4)} < 1$, over the unit interval then the change in u is too small. $\mathrm{DD}_i^{(4)}$ is fixed to 1 by defining a new interval $\Delta\tau$, starting at a point μ in the interval $[t_j\ ,\ t_{j+2}]$ such that:

$$DD_j^{(4)}\left(\frac{\Delta t}{\Delta\tau}\right)^4 = 1$$

So the new interval $\Delta\tau$ has size $\Delta t\left(DD_j^{(4)}\right)^{\frac{1}{4}}$

or more generally, for the interval (2.18) with u a vector:
for all u_i $(i=1..,n)$:

$[DD_j^{(m)}]_i$ can be found for $j = 0,1...,\ N-1$.
$[DD_N^{(m)}]_i = 2[DD_{N-1}^{(m)}]_i - [DD_{N-2}^{(m)}]_i$
(extrapolating from the 2 previous divided difference values)

Then the (m+1)th divided differences can be found:

$$[DD_j^{(m+1)}]_i \approx \frac{[DD_{j+1}^{(m)}]_i - [DD_j^{(m)}]_i}{\frac{1}{2}(\Delta t_j + \Delta t_{j+1})}$$

And the new interval sizes $\Delta\tau_j$ are defined:

$$\Delta\tau_j = \Delta t\left(\sum_{i=1}^{m} [DD_j^{(m+1)}]_i\right)^{\frac{1}{4}}$$

So the mesh intervals are adaptive, with their size based on the shape of the $u(t)$ at the previous point. The existing values for u are for t at the previous mesh size. Interpolation is used get $u(t_{j+i/m})$ for the new values of $t_{j+i/m}$.

NON-AUTONOMOUS SYSTEMS

The methods used by AUTO work for autonomous systems, where the dependent variable (t in (2.17)) does not appear explicitly in the differential equation. Non-autonomous Boundary Value Problems are:

(2.38) $$\frac{du}{dt} = f(u(t),\lambda,t), \quad t\in[0,1],\ u(.),F(.,.):\Re^{n+1}\rightarrow\Re^{n},\ \lambda\in\Re$$
$$\text{with boundary conditions } b_i(u(0),u(1),\lambda)=0, \quad i=1,2,\ldots,n_b$$

$u(t)$ is predicted by $p_j(t)$ for $t\in[t_j,\ t_{j+1}]$, and is evaluated at the points $\{z_{j,i}\}$in the interval. So (2.21) becomes:

(2.39) $$\frac{dp_j}{dt}(z_{j,i}) = f(p_j(z_{j,i}),\lambda,z_{j,i}), \quad i=1,\ldots,m, \quad j=0,1,\ldots,N-1$$

The Jacobian of f must also include t:

$$f_x = \frac{\partial f(x(t),\lambda,t)}{\partial x(t)}$$

And similarly for the numerical methods for generating the Jacobian.

With these modifications, the existing methods can be used for Non-autonomous boundary value problems.

3 IMPLEMENTATION and RESULTS

3.1 AUTO

The numerical techniques of Chapter 2 have been implemented by Eusebius Doedel in a computer package called AUTO. It performs bifurcation analysis on algebraic and differential systems, producing output files representing solution branches on a bifurcation diagram, and also solution curves at points on these branches (for differential systems). A plotting program called PLAUT is incorporated as well to extract the numbers from the Fortran output files and plot graphs. This eliminates the need to manipulate the numbers into the format for an appropriate graphics package.

To model a system using AUTO, a Fortran *model file* is created (see Appendix A for an example). This file defines:

a. $F(u,\lambda)$

b. the Jacobians F_u and F_λ (these may optionally be generated numerically)

c. a starting point $x_0 = (u_0^t, \lambda_0)$ for the first branch (note that u may be a function of t). This point cannot be a bifurcation point.

c. the type of system (for example if it is an algebraic system or a Boundary Value Problem)

d. the boundary conditions (if the system is a BVP)

e. the number of mesh intervals (N), and the number of collocation points (m).

f. Other AUTO parameters defining the numerical method used.

The AUTO parameters define all the constants involved in the numerical techniques of Chapter 2. For example, the stepsize and mesh can be fixed or adapted regularly (after one or more points), and the initial stepsize is defined as well as a minimum and maximum stepsize. This module is compiled and linked with the rest of the program (see Appendix A). For algebraic systems, this compiled program generates **all** the branches which come out of the initial point x_0. For boundary value problems, a single branch is generated with bifurcation points identified. Further branches can be generated by continuing from branching points of previous runs.

MODIFICATION OF AUTO FOR NON-AUTONOMOUS SYSTEMS

The variables $u_{j+i/m}{}^{(a)}$ are defined in an array U(j , $in + a$), where n is the dimension of u. The value of u_a at a point $z_{j,i}$ is defined by the Lagrange polynomial $p_j(z_{j,i})^{(a)}$ generated by $\sum_{i=1}^{m} W(j,i)\, U(j,(i-1)*n+a)$, where W($j$,$i$) is predefined as $w_{j,i}(z_{j,i})$ from

(2.37). So u_a is the value of $u(t)$ at $t = z_{j,i}$.

The system f from equation (2.17) is defined in subroutine FUNC which is in the *model file*. Variables $u=[u_1 , \ldots u_n]$ and λ are passed to this subroutine, and the variable F (representing $F(u,\lambda)$) is returned along with the Jacobian derivatives F_u and F_λ (if they are not generated numerically). Along with these variables, t and Δt_0 are also passed to this subroutine, so they are accessible when evaluating F. The reason for Δt_0 being passed will be explained later in this Chapter. Appendix B shows the subroutines affected and how they have changed.

3.2 EXAMPLE PROBLEMS

These example problems have 2 purposes:

(1) To test AUTO (when analytical solutions are known)

(2) To generate bifurcation graphs

The pathfollowing algorithm of AUTO is tested by using the example problems in Section 1.1, which have known analytical solutions. Then AUTO is applied to the combustion equation (1.7). This equation is non-autonomous, so the modified version of AUTO has to be used. The effectiveness of AUTO for this system can then be tested by comparing the computational results with the analytical results from section 1.2. Finally the model is used to evaluate λ_{crit} for non-*Class A* shapes and compared with the numerical results from other models.

ALGEBRAIC SYSTEM EXAMPLES:

Example 3.1:

$$f_1(u_1,u_2,\lambda) = u_1^3 + u_2 - \lambda u_1 = 0$$
$$f_2(u_1,u_2,\lambda) = u_1^2 u_2 - u_1^3 - \lambda u_2 = 0$$

This is the same system as in Example 1.1 where there is a point (0,0,0) on the trivial branch where the Jacobian is singular but there is no branching point.

Branching points are detected by AUTO as points where there is a change of sign in the determinant of P_x :

$$P_x = \begin{bmatrix} 3u_1^2-\lambda & 1 & -u_1 \\ 2u_1u_2-3u_1^2 & u_1^2-\lambda & -u_2 \\ \dot{u}_1 & \dot{u}_2 & \dot{\lambda} \end{bmatrix}$$

(in this case, θ_u and θ_λ are both one). Along the trivial branch, the direction vector $(\dot{u}_1,\dot{u}_2,\dot{\lambda})^t$= (0,0,1) and u=(0,0) so the determinant Det(P_x) is λ^2 . This has a zero at

(u^t,λ) = (0,0,0), but the sign of Det(P_x) is positive for $\lambda< 0$ and $\lambda> 0$ (see Section 2.1). So there is no change in sign and a Branching point is not detected. The trivial branch can be generated using AUTO by starting at a non-bifurcation point for example x_0= (0,0,-1).

<u>Example 3.2</u>

$$f_1(u_1,u_2,\lambda) = u_1^2 - \lambda u_1 = 0$$
$$f_2(u_1,u_2,\lambda) = u_2^2 - \lambda u_2 = 0$$

The analysis in Chapter 1 showed that this system has 3 branches emanating from the trivial branch at point (0,0,0). When this is put into AUTO with starting point (0,0,-1), this bifurcation point is not found.

The Jacobian P_x is:

$$P_x = \begin{bmatrix} 2u_1-\lambda & 0 & -u_1 \\ 0 & 2u_2-\lambda & -u_2 \\ \dot{u}_1 & \dot{u}_2 & \dot{\lambda} \end{bmatrix}$$

Along the trivial branch the det(P_x) = λ^2 which is ≥ 0 for all λ. Because there is no change in sign, the bifurcation point was not detected by AUTO.

Also P_x has a 2-dimensional nullspace at this point. So (2.16) does not have a unique solution and branching at this point using AUTO is not possible.

Example 3.3

Consider the system in example 1.3. Starting at a regular point (0,0,-1), AUTO generates the bifurcation graph for this system without any problem. The eigenvalues at all of the branching points are all odd, which means that the determinant changes sign at these points (see Section 2.1).

The initial, minimum and maximum stepsizes were initialised in the model file as DS = 0.005, DSMIN = 0.001 and 1 respectively. Other parameters, for example the number of times the Newton Corrector iterates (see Appendix A for a list of all the parameters) are set to their default values. The compiling and running of AUTO with this system took 19 seconds of CPU time. The results of AUTO are compared with the analytical solution in figure 3.1.

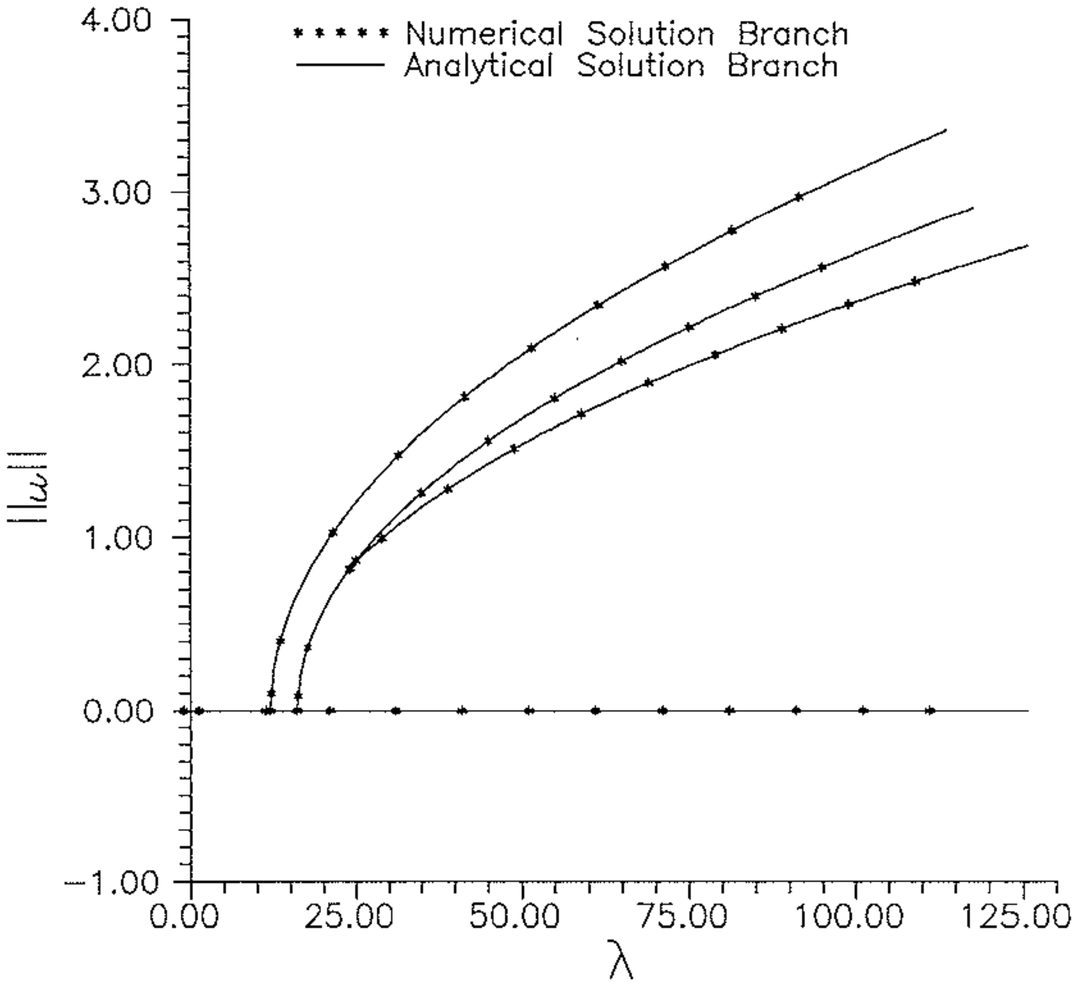


Figure 3.1: Analytical and Numerical Solutions to Example 3.3

The spacing of the numerical solution points is proportional to the stepsize. Note that the initial stepsize rapidly adapted to a much larger one (see Appendix B for the way AUTO adapts the stepsize). This implies that the initial stepsize was too small, and the same accuracy can be gained with a larger one.

THE COMBUSTION EQUATION

Bifurcation graphs for (1.7) can be generated numerically. AUTO is designed for systems written as first order D.E's. Equation (1.7) can be defined as a system of two first order D.E's:

$$\text{(3.1)} \qquad \begin{aligned} \frac{du_1}{dt} &= u_2 \\ \frac{du_2}{dt} &= -\frac{j}{t}u_2 - \lambda e^{u_1} \quad , j \neq 0 \end{aligned}$$

with boundary conditions:

$$u_2(0) = 0 \ , \quad u_2(1) + \mathrm{Bi}\, u_1(1) = 0$$

where $t(=r)$ is the dependent variable, and the Biot number (Bi) and Shape factor (j) are constants. As t appears explicitly in the equations, (3.1) is non-autonomous and the modified version of AUTO has to be used.

There is a special case for (3.1) when j=0:

$$\text{(3.2)} \qquad \begin{aligned} \frac{du_1}{dt} &= u_2 \\ \frac{du_2}{dt} &= -\lambda e^{u_1} \end{aligned}$$

with the same boundary conditions. This is autonomous, and can be analyzed by the existing version of AUTO.

The accuracy of the numerically generated bifurcation graph for different values of j will be investigated in Examples 3.5, 3.6, 3.7, and 3.8 which use a fixed value of Bi corresponding to the Frank-Kamenetskii Boundary Conditions (Bi=∞).

Later, the accuracy of AUTO for different Bi numbers will be tested in Examples 3.9 and 3.10, which have j fixed to 1. From these examples, one can get a good idea of the effectiveness of AUTO for this system. Finally the AUTO results for different non-*class A* shapes (using the shape factor technique discussed in Section 1.2) are compared with numerical results from Wake and Jackson [15] (which use the exact Laplacian for the shape, rather than an approximation).

Example 3.5: The Infinite Slab with Frank-Kamenetskii Boundary Condition

This is system (3.2) where $j = 0$ and $\mathrm{Bi} = \infty$. The boundary conditions become:

$$u_2(0) = 0 , \quad u_1(1) = 0.$$

This has an implicit analytical solution (see Section 1.2). A model file of this system was created, with 10 collocation points (NTST=10), and default values for other parameters. The model file was linked with the un-modified version of AUTO and AUTO was executed, taking 30 seconds of CPU time. The results are compared with the analytical solution in figure 3.2, where u_1=max[$u(t)$].

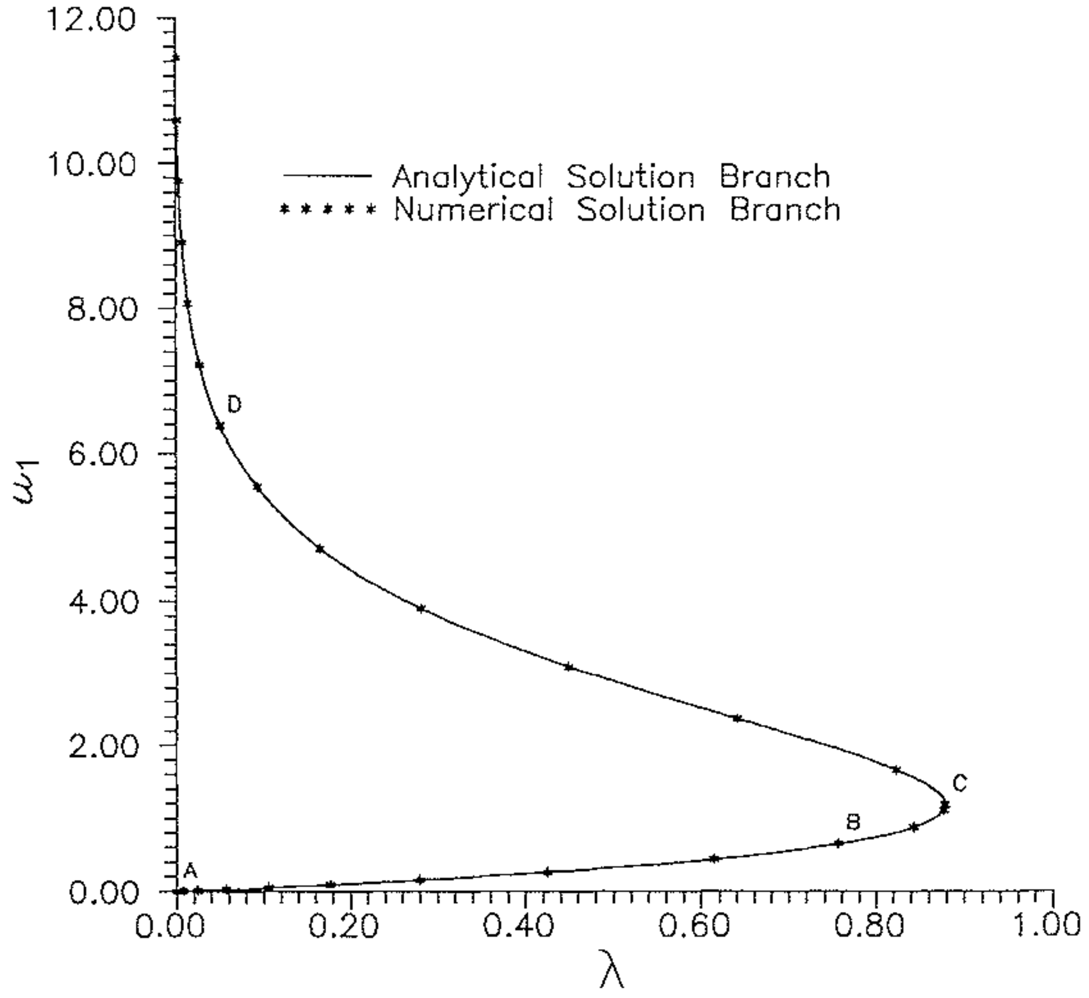


Figure 3.2: Analytical and Numerical Solutions for System (3.2), (where j=0 and Bi=∞)

As in figure 3.1, the spacing of the *'s, corresponds to the adaptive stepsize of the pathfollowing method. This shows that as would be expected, the Newton Corrector converges better when the curvature is small. AUTO also outputs the value of the solution $u_1(t)$ and $u_2(t)$ at various points on the bifurcation branch (see Appendix A for more information on output files). The curves of $u_1(t)$ at points labelled A, B, C, and D in figure 3.2, are shown in figure 3.3.

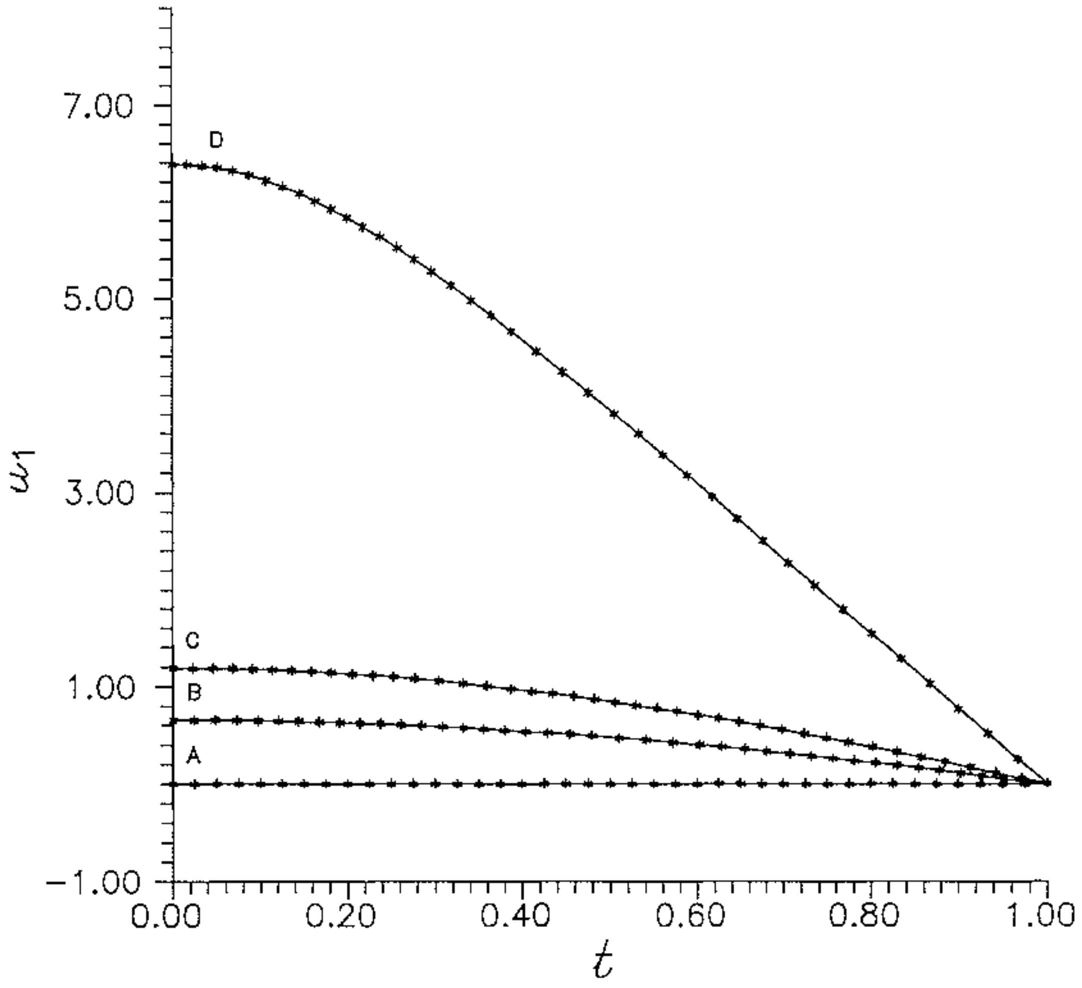


Figure 3.3: $u_1(t)$ at points A, B, C, and D in figure 3.2

The spacing of the *'s in figure 3.3 corresponds to the mesh size. At points A, B and C, the spacings do not vary greatly, but at point D (when u_1 is proportionally much larger), the spacing becomes very small when t is close to zero as the curvature changes more at these points. From (3.2), it can be seen that this is because du_2/dt increases exponentially with u_1.

When the resulting system with $j\neq 0$ is entered into AUTO, a divide-by-zero condition when t=0 means AUTO will stop at this point, giving a divide-by-zero error. These systems have a solution when t = 0, but there is an apparent singularity in the derivative of u_2 at this point.

When t is close to zero:

$$u''(0) = \lim_{t \to zero} \frac{u'(t) - u'(0)}{t}$$

and as $u'(0) = 0$,

(3.3)
$$\frac{u'(t)}{t} \approx u''(t), \quad for\ t \to 0$$

Approximation (3.3) can be used to eliminate the singularity in the derivative when t=0, by defining a piecewise system:

(3.4)
$$\frac{du_1}{dt} = u_2$$
$$\frac{du_2}{dt} = \begin{cases} -\dfrac{1}{(1+j)}\lambda e^{u_1}, & t<\xi \\ -\dfrac{j}{t}u_2 - \lambda e^{u_1}, & t\geq\xi \end{cases} \qquad \xi \neq 0$$

with boundary conditions:

$$u_2(0) = 0 \ , \quad u_2(1) + \mathbf{Bi}\, u_1(1) = 0$$

where ξ is very close to zero.

An obvious strategy for solving (3.1) is to fix ξ to a small number (say 0.01), and approximate (3.1) by (3.4). Example 3.6 shows that this strategy does not work.

<u>**Example 3.6**: System (3.4) where j=2, Bi=∞, and ξ=0.01 (fixed)</u>

Default values for AUTO parameters were used except for NTST = 20, DS = 0.001, and DSMIN = 0.00001. Figure 3.4 shows that the bifurcation diagram of the numerical solution.

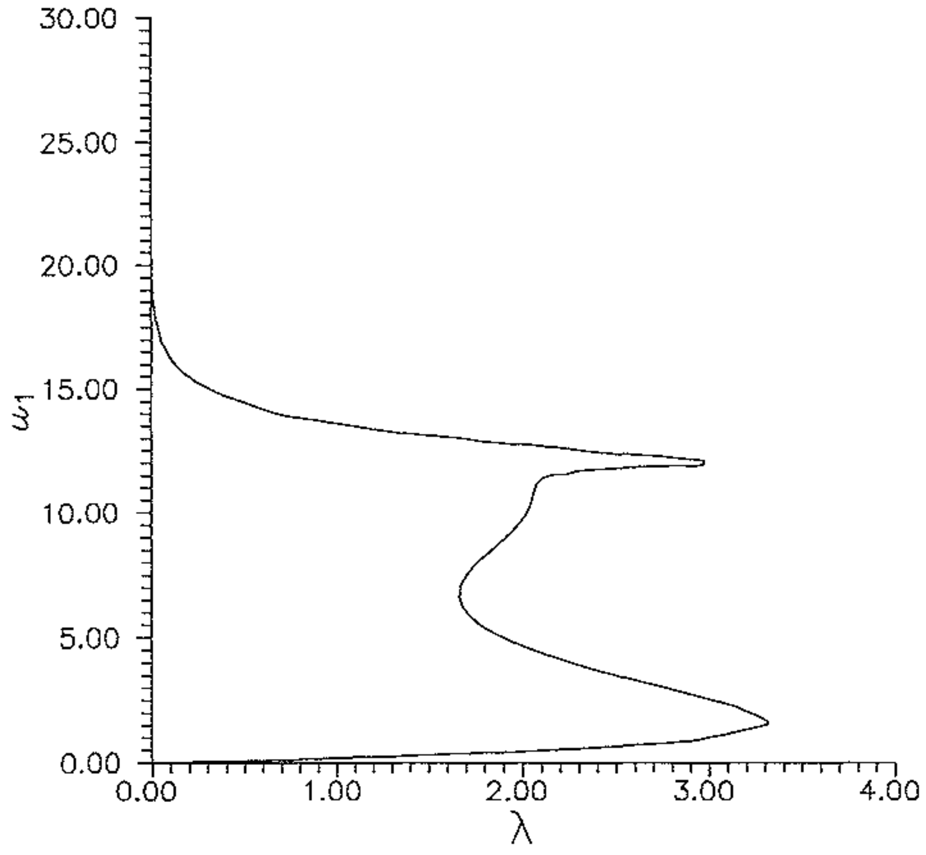


Figure 3.4: System (3.4) with Bi = ∞, j = 2, and ξ fixed to 0.01

When j = 2 and Bi = ∞, The result by Wake [14] in section 1.2, proved that λ converges to 2 as $u_1 = \max[u_1(t)]$ increases to ∞. Figure 3.4 shows that Example 3.6 converges to 0 as u_1 increases to ∞.

So (3.4) with ξ fixed is not a good approximation to (3.1).

If the solution u_1(t) is approximated by a power series:

$$u_1(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + ...$$

Then the solution for the original system (3.1) and the piecewise system (3.4) are (3.5) and (3.6) respectively.

(3.5) $$-\lambda e^{u_1} = (2+2j)a_2 + (6+3j)a_3 t + (12+4j)a_4 t^2 + ...((n+1)(n+2)+(n+2)j)a_{n+2} t^n$$

and

(3.6) $$-\lambda e^{u_1} = (2+2j)a_2 + (6+6j)a_3 t + (12+12j)a_4 t^2 + ... +(n+2)(n+1)(1+j)t^n$$

The value of $-\lambda e^{u_1}$ increases exponentially with u_1, and as it becomes large, the difference between the two systems increases. So to keep a constant error ξ must decrease exponentially as u_1 increases. Another problem arises when ξ is less than the first sub-interval length :

$$\xi < \frac{\Delta t_0}{m}$$

where m is the number of collocation points and Δt_0 is the first interval length, (as in (2.48)). In this case, the piecewise partition is at t = 0, because u is defined only at discrete points. This gives the same result as if ξ = 0, (because there are no non-zero t values between 0 and ξ) which contradicts the condition that ξ > 0. If ξ is defined:

(3.7) $$\xi = \Delta t_0$$

then ξ is always m points away from the zero interval.

Also as the change in curvature is proportional to derivative du_2/dt , the adaptive mesh procedure will ensure that the interval Δt_0 decreases exponentially as u_1 decreases. This means that Δt_0 needs to be passed to the subroutine FUNC in the user *model file*.

Modification to AUTO:	The modification to AUTO (referred to earlier in this Chapter) was made so that Δt_0 is indeed passed to the subroutine FUNC (see Appendices A and B for an example *model file* and implementation).

Example 3.7: System (3.4) where j=1, Bi=∞, and $\xi=\Delta t_0$

This system was chosen because it has an analytical solution, and the numerical and analytical values can be compared to test the effectiveness of (3.4) in approximating (3.1). As (3.4) is non-autonomous, and requires Δt_0, the modified version of AUTO was required to solve this system. Initial parameters were NTST=20, DS=0.001, DSMIN=0.00001, and the maximum number of points evaluated on a branch (NMX) was set to 500, with the other parameters set to their default values. Because of the shape of the solution curve (see figure 3.3), the overall change in slope increases at each point evaluated along the branch, requiring progressively smaller stepsizes. For this reason, NTST was increased to 20. This value had been worked out from previous runs. If NTST is too large, then the program takes too long to run, and if it is too small, then inaccuracies occur when evaluating u_1.

The compilation and running of AUTO took 136 seconds of CPU time. The output is compared with the analytical solution in figure 3.5.

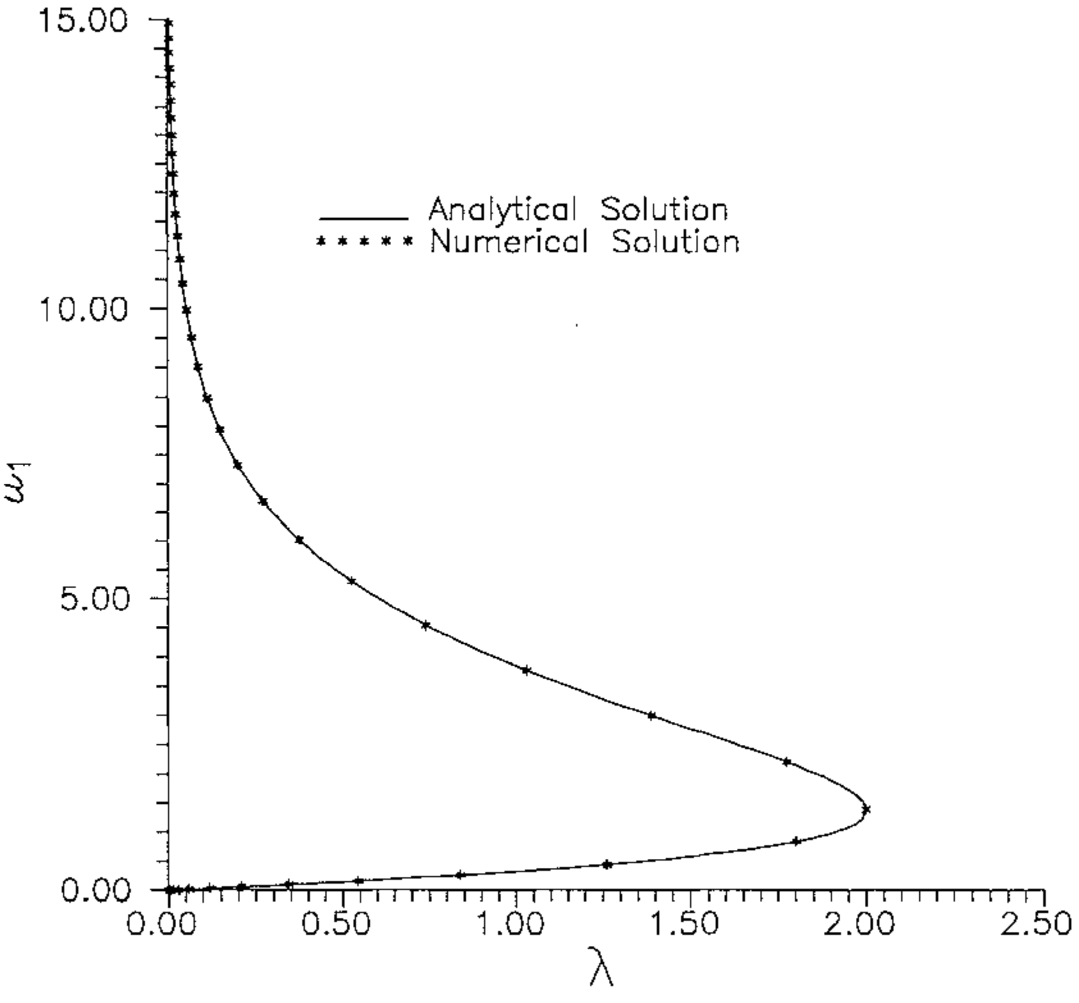


Figure 3.5: Analytical and numerical solution to Example 3.7 (j=1, Bi = ∞, ξ=Δt_0)

For other values of j there are no known analytical solutions, but λ converges to a known value as u_1 increases for $1<j<9$ (see Section 1.2). The analytical and numerical values for these numbers are compared in the following example.

Example 3.8: System (3.4) for different values j, where Bi=∞ and ξ=Δt_0

AUTO was used 10 times on (3.4) with j ranging from 0 to 10. Initial parameters were the same as Example 3.7. The entire computation took approximately 23 minutes of CPU time. Figure 3.6 shows the shape of the resulting curves.

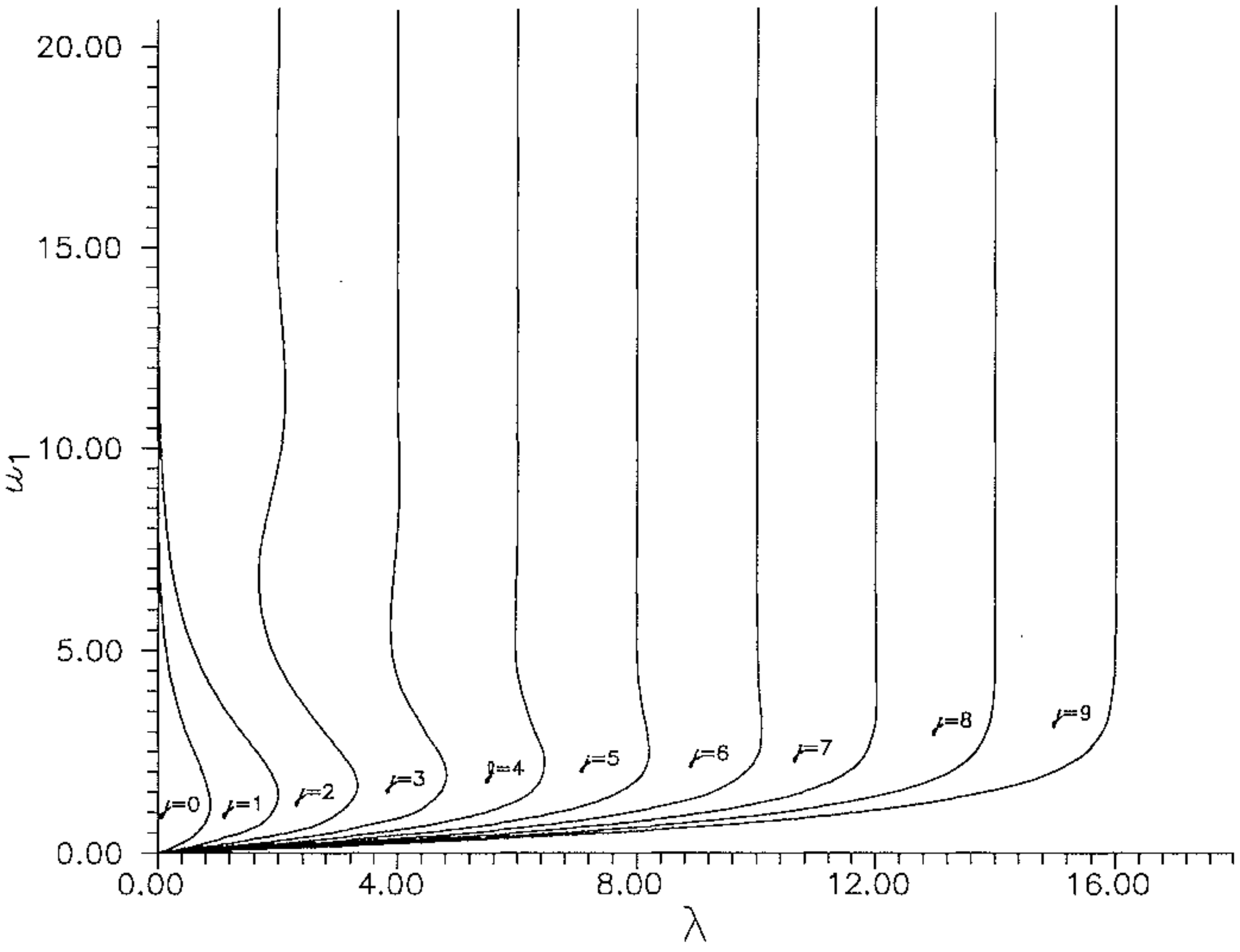


Figure 3.6: Example 3.8 for j = 0,1,2,3,4,5,6,7,8,9

For $1 < j < 9$, the curves in figure 3.6, have multiple limit points. Figure 3.7 blows up the curve $j = 8$ to show that it does fold back on itself.

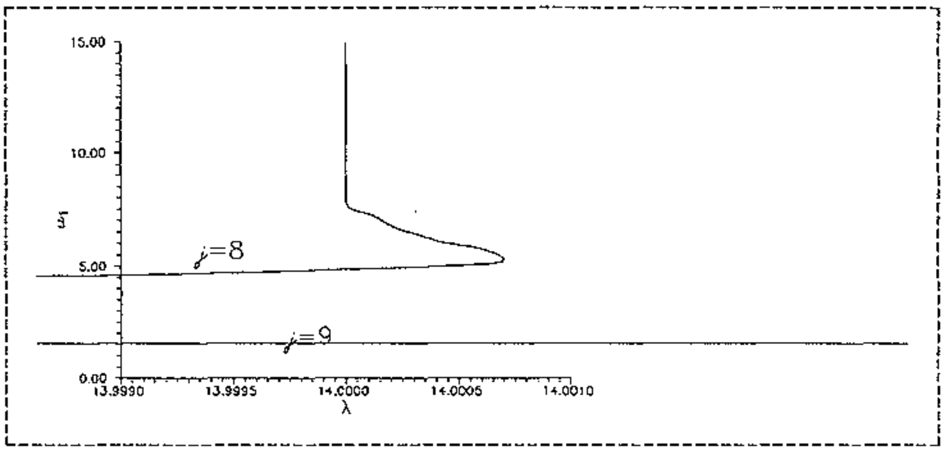


Figure 3.7: Close up of dotted area in figure 3.5

Analytically it has been proven [14] that there are an infinite number of "wiggles" for j in this range. As u_1 increases, λ converges to a value denoted by λ_∞. From Section 1.2:

$$\lambda_\infty = 2(j-1)e^{\left(-\frac{2}{\mathrm{Bi}}\right)}$$

As $\mathrm{Bi} = \infty$, this simplifies to $\lambda_\infty = 2(j-1)$. Numerically, this can be predicted as the last point which was found on the branch, i.e. the 500'th point (as NMX=500).

Table 3.1: Numerical and Analytical values for λ_∞ , (Bi = ∞)

j	Analytical value of λ_∞ 2(j-1)	Number of *limit points* found along the branch	Final value of λ and u_1 along branch (500 points) λ	u_1
1.0	Does not exist	1	0.00000009	36.7
1.5	1	3	0.9038190	21.2
2.0	2	5	2.009908	20.9
2.5	3	5	2.999384	20.9
3.0	4	6	3.999282	20.9
3.5	5	6	4.999305	20.9
4.0	6	5	5.999492	20.9
4.5	7	5	6.999835	20.9
5.0	8	6	8.000241	20.9
5.5	9	5	9.000608	20.9
6.0	10	6	10.00090	20.9
6.5	11	6	11.00107	20.9
7.0	12	5	12.00116	20.9
7.5	13	5	13.00117	20.9
8.0	14	4	14.00027	28.2
8.5	15	2	15.00105	20.9
9.0	does not exist	0	16.0097	20.9

The 500'th point was to within one decimal place of λ_∞ (after 1 or more oscillations or 2 or more limit points) , so that the branches appear to be converging to λ_∞.

From table 3.1, the number of limit points in the curve found numerically in 500 steps, diminished as j approached the end-points 1 and 9.

To see what happens at these endpoints, the number of steps (NMX) was increased to 2000, and AUTO *model files* were created and run for j=0.9, 0.95, 1, 1.05, 1.1, 8.9, 8.95, 9, 9.05, and 9.1. This process took 74 minutes of CPU time.

Table 3.2: Numerical and Analytical values for λ_∞ , (Bi = ∞)

j	Analytical value of λ_∞ { $=2(j-1)$ }	Number of *limit points* found along the branch	Final value of λ and u_1 along branch (after 2000 points)	
			λ	u_1
0.90	Does not exist	1	0.0000000	83.9
0.95	Does not exist	1	0.0000000	62.5
1.00	Does not exist	1	0.0000000	47.9
1.05	0.1	1	0.0000079	37.9
1.10	0.2	2	0.0086138	31.0
8.90	15.8	2	15.81766	26.6
8.95	15.9	0	15.91730	26.6
9.00	Does not exist	5	16.01697	26.6
9.05	Does not exist	6	16.11665	26.6
9.10	Does not exist	5	16.21633	26.6

Even after 2000 iterations when u_1 is greater than 26, the number of limit points found near the endpoints $j=1$ and $j=9$ is smaller than the number found when j is further away from the endpoints (in Table 3.1). As the maximum value of u_1 does not decrease (in fact it increases) as j approaches the endpoints, the period of oscillation must have increased.

Example 3.9: System (3.4), j=1, Bi varied and $\xi=\Delta t_0$.

Five exponentially increasing Biot numbers {0.125, 0.5, 2, 8, 32} were chosen, and entered as 5 systems (as in Example 3.7, and using the same AUTO parameters). The numerical results are compared with the analytical results (see Section 1.2) in Figure 3.8.

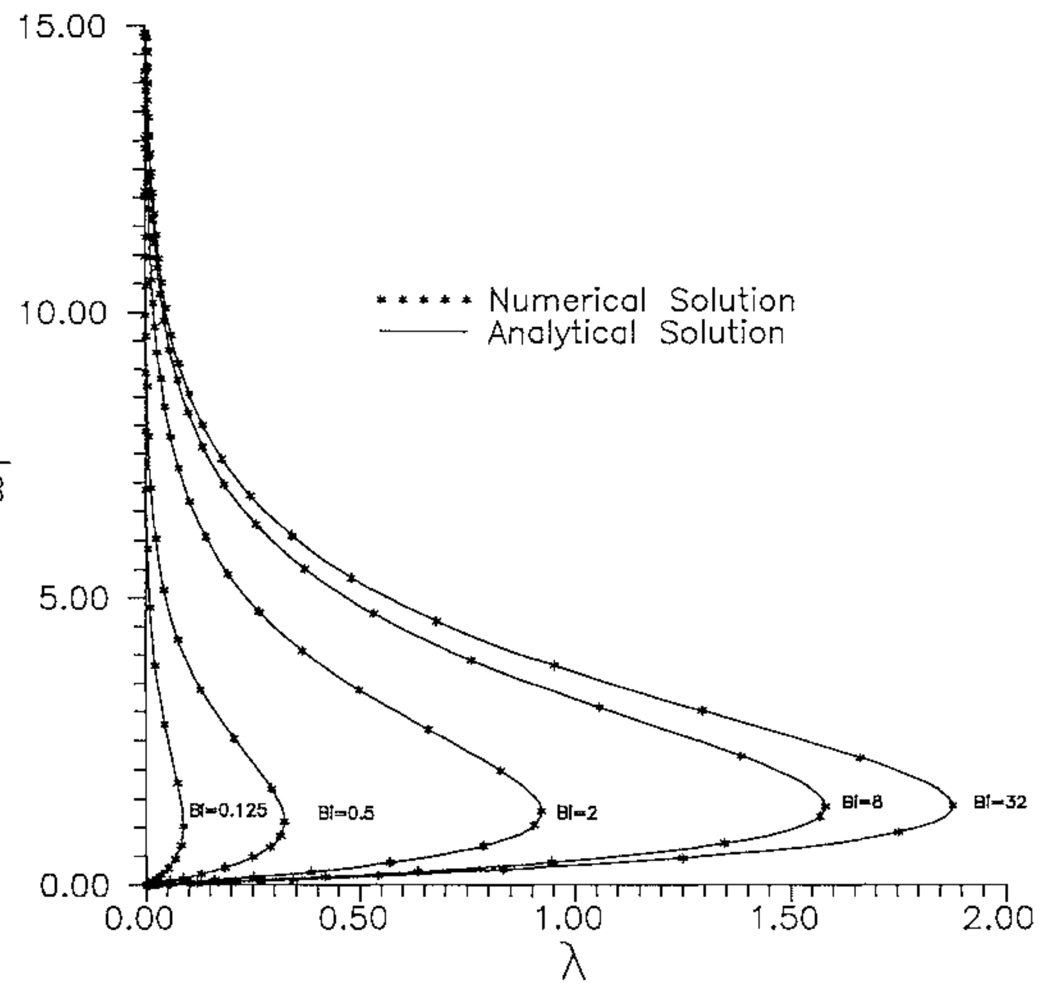


Figure 3.8: Numerical and analytical solutions for Example 3.9

Note that as Bi increases, the limit point converges to 2, which is the case when Bi=∞ (see figure 3.5).

<u>Example 3.10</u>: <u>System (3.4), j=2, Bi varied and $\xi=\Delta t_0$.</u>

This has no analytical solution, but the final point evaluated on each branch can be compared with λ_∞ as in example 3.8. The AUTO model file for this system when Bi=2, as well as the output, is in Appendix A. When j=2, λ_∞ becomes $2e^{-\frac{2}{\mathrm{Bi}}}$. This value is compared with the 500'th point in Table 3.3.

Table 3.3: Numerical and Analytical values for λ_∞ , (j = 2, and Bi is varied)

Bi	Analytical value of λ_∞ ($=2e^{-\frac{2}{\mathrm{Bi}}}$)	Number of *limit points* found along the branch	Final value of λ and u_1 along branch (after 500 points)	
			λ	u_1
0.125	0.000000225	5	0.000000227	36.8
0.5	0.036631277	5	0.03680918	24.9
2	0.73575888	5	0.7393332	21.9
8	1.557601566	5	1.565294	21.1
32	1.878826126	5	1.888150	21.0

The curve has the same number of limit points for all the Bi values used, showing that the period is oscillation is independent of Bi.

After less than 3 oscillations (or 5 limit points) the final value of λ approximates λ_∞ to 2 significant figures.

Example 3.11: System (3.4) with j representing different shapes

Section 1.2 showed how the shape factor j in System (3.4) can be used to approximate the Laplacian over any body. So the system can be used to generate solution curves for any shape. Wake and Jackson in [15] numerically calculated λ_{crit} for objects using the dimensionless heat equation:

(3.8)
$$\nabla^2 u + \lambda e^u = 0$$

using the exact Laplacian $\nabla^2 u$ for each body.

Different objects ranging from the infinite slab (j=0) to the regular tetrahedron (j=4.178) were entered as model files into AUTO using (3.4) with the same parameters as in Example 3.8. The resulting value λ_{crit} (the value of λ at the first limit point) is scaled as discussed in Section 1.2 (page 23) so it can be compared to the results of [15]:

$$\lambda^*_{crit} = \frac{3\lambda_{crit}}{(j+1)RA_0^2}$$

Table 3.4 shows the value of λ_{crit}^* and λ_{crit} from [15], for objects with j ranging from 0 to 4.178.

Table 3.4: λ_{crit} for different shapes using real and approximate Laplacians

j	**Shape**	RA_0	λ_{crit} from AUTO	λ_{crit}^*	λ_{crit} from [15]	**Error** %
0	Infinite Slab	3	0.878	0.878	0.878	0.0
0.438	Rectangular Parallelepiped (ratio 1:10:10)	1.731	1.343	0.935	0.832	11.0
1	Infinite Cylinder	1.225	2.000	2.000	2.000	0.0
1.444	Infinite Square Rod	1.354	2.564	1.717	1.692	1.4
1.694	Rectangular Parallelepiped (ratio 1:1:10)	1.354	2.898	1.760	1.640	6.8
2	Sphere	1	3.322	3.222	3.324	0.0
2.729	Equicylinder	1.115	4.395	2.844	2.774	2.5
3.280	Cube	1.194	5.260	2.586	2.448	5.3
4.178	Regular Tetrahedron	0.537	6.759	13.580	*not evaluated*	-

Equation (3.8) is the same as (3.4) for *Class A* geometries (when j is an integer). Table 3.4 shows that for non-*Class A* geometries, (3.4) approximates (3.8), with the accuracy of the approximation depending on the shape.

4 CONCLUSIONS

The numerical analysis and running of AUTO have shown the capability of AUTO for bifurcation analysis on algebraic systems as well as Autonomous Boundary Value Problems. With modifications done to AUTO, it can now also generate bifurcation diagrams for non-autonomous Boundary Value Problems.

The numerical methods of AUTO will generate branches of dynamical systems if:

1 - The branching point with a Jacobian F_u of the system F has a zero eigenvalue of odd multiplicity.

and

2 - The dimension of the nullspace NS(F_u) is greater than 1 only at discrete points.

The Frank-Kamenetskii Heat equation (4.1) fits these criteria.

$$\begin{gathered}\frac{d^2u}{dr^2}+\frac{j}{r}\frac{du}{dr}+\lambda(x)e^{u}=0\\ \frac{du}{dr}(0)=0,\quad \frac{du}{dr}(1)+Bi\,u(1)=0\end{gathered}\tag{4.1}$$

This system is solvable by the modified version of AUTO for any value of j and Bi, with the process accomplished in a matter of seconds.

The tests in Chapter 3 showed that the curves generated by AUTO compared favourably with the analytical solutions when j is 0 and 1. Another analytical result from Chapter 1 is that for $1< j < 9$, the branch converges to the line $\lambda = \lambda_\infty$. The numerically generated curves were shown to have this property as well. A similar process for five different Bi number and j=1 and 2, showed that the results were accurate irrespective of the Bi number used.

These test results imply that the modified version of AUTO gives accurate results for any value of j or Bi.

Applying the ideas about shape factor of Boddington, Gray, and Harvey [1], (4.1) was used to approximate the exact Laplacian equation:

$$\nabla^2 u + \lambda e^u = 0. \tag{4.2}$$

The value λ_{crit} (the threshold value of λ for thermal ignition) was generated by AUTO using (4.1) for different shapes. This value was compared with the result using (4.2) calculated by Wake and Jackson [15], showing that a good approximation to λ_{crit} can be gained for any shape possessing a point of symmetry.

This is a successful test-run for the effectiveness of the modified version of AUTO, and opens the way for modelling any non-autonomous Boundary Value Problem.

FUTURE RESEARCH

Bifurcation graphs are highly non-linear and require some kind of pathfollowing technique to generate them. AUTO is restricted to modelling systems of ordinary first order D.E.'s. The Boundary Value Problems investigated in this thesis are models which use a single length dimension r for volume, and a shape factor j. Other models have 2 or 3 units for length. These are partial differential equations and cannot be solved by AUTO. The pathfollowing algorithm could be applied to these problems resulting in a numerical technique for generating bifurcation graphs. This could then be incorporated into AUTO, giving a very powerful tool for pathfollowing any system of first order partial D.E.'s.

APPENDIX A: USING AUTO

This section contains information for using AUTO for algebraic systems and Boundary Value Problems.

AUTO can be used on a dynamical system by making a model file which is a FORTRAN file containing the information required by AUTO for analyzing this system. This file has seven subroutines:

SUBROUTINE FUNC:

This contains the definition of the system where U(i) is u_i, PAR(1) is λ, and F(i) is F_i. The derivatives F_u and F_λ are also entered in this subroutine in arrays DFDU and DFDP, where DFDU(i,j) = dF_i/du_j and DFDP(i,1) = $dF_i/d\lambda$, although they will not be used if the Jacobian is generated numerically by AUTO.

SUBROUTINE STPNT

This contains the solution curves $u(t)$ for a particular value λ, and so gives a starting point for the bifurcation curve.

SUBROUTINE INIT

This contains the values of the AUTO parameters. For example, JAC=0 tells AUTO that the Jacobian is to be generated numerically, and IPS=4 tells AUTO that the system is a Boundary Value Problem. If the default value for a parameter is used, it does not need to be defined. (See pp. 134-143 of [4] for a description of all the parameters and their default values).

SUBROUTINE BCND

This contains the Boundary Conditions. This subroutine will be ignored if the problem does not have boundary conditions. AUTO uses a fixed interval [0,1] for t where $u_i(0)$ = U0(i), $u_i(1)$ = U1(i). Boundary conditions are entered as FB(i) which are functions of U0 and U1 and PAR(1). The boundary conditions are therefore FB(i) =0. The derivatives of FB are also entered here, but may not be used if derivatives are generated numerically.

SUBROUTINE INCD

This contains integral conditions. Variables FI(i) are defined as functions of U and PR(1), then the integral condition is: $\int_0^1 \mathrm{FI}(i)\,dt = 0$. The derivatives of FI are entered here too.

FUNCTION USZR

One or more variables USZR are defined. USZR is a function of PAR(1), and when USZR=0, then the point is given a label number.

The compilation and running of AUTO described in the thesis were performed on a Sun Sparc Station SLC, although these could be done on any platform which has a F77 Fortran Compiler. The graphics output (from PLAUT) does require that the platform has a suitable Tektronics emulator.

3 commands are needed to run AUTO:

@auto {name} :	runs AUTO with *model file* **aut{name}.f**, deleting all previous output created from previous runs with this *model file*. Output files are p.{name}, q.{name} and d.{name}.
@autocont {name}:	runs AUTO with *model file* **aut{name}.f**, using previous output files p.{name} q.{name} and d.{name} as input files and appending the resulting output to them.
@plaut {name}:	Initiates the plotting program using output files p.{name}, and q.{name}.

@auto and **@autocont** run a sequence of system commands. In these files, the *model file* is linked to a preprocessor program which generates the *main program*. The *main program* changes depending on the type of analysis required for example, whether the model is an algebraic system or a BVP, or whether the analysis is continued from a previously calculated point. The type of analysis required is defined by the AUTO parameters in the *model file*. The *main program* sets up the required calls and necessary workspace, and runs the appropriate AUTO routines. It is compiled and linked with 3 AUTO libraries and the *model file*, and then run. The modified version of AUTO works the same as this except the commands are ***@autom*** and ***@autocontm***.

Output is in the form of 3 Fortran output files and the screen:

fort.7 (p.{name}): At the beginning of this file, all the parameters values defined from INIT in the *model file* are stored. The rest of the file contains all the points evaluated for the *bifurcation graph*. Each point (u', λ) has an identification number, a type, an L2-NORM, and the maximum value for each u_i in the interval [0,1]. The point type is one of:

EP - An end point of a branch, normal termination.

MX - End point of a branch, abnormal termination (no-convergence)

UZ- Zero of function in USZR (from SUBROUTINE USZR in *model file)*

BP- Branching point

LP- Limit point

Regular Point

When the point type is not a *Regular point*, it is assigned a label, which is also output in this file.

fort.8 (q.{name}): This file is only generated for Boundary Value problems. It has the restart information at label points for continuing bifurcation diagrams in future runs of AUTO using **autocont**. It contains the Values of $t_{j+i/m}$ and $u_{j+i/m}{}^{[a]}$, and the direction vector $[u'_{j+i/m}{}^{[a]}, \lambda]'$.

fort.9 (d.{name}):	This file has a summary of the Newton Corrector method including the number of iterations and stepsize at each point. Branching points, limit points and user defined points detection are summarised as well.
Screen:	This is the same as the contents of output fort.7 except that only label points are displayed, with the addition of columns headings.

The bifurcation graph can be generated from p.{name} by using the **@plaut** command. This program can also be used to graph the solution $u(t)$ from q.{name}. **@plaut** is a text-driven graph tool. It requires a tektronics terminal. There is available a SUN tek emulator called **tektool** which sets up a tektronics window. The command "BD0" plots the bifurcation graph from p.{name} on the screen. "2D" plots data from q.{name}. Help can be gained by typing "HELP".

<u>An example :</u> **Example 3.10 when Bi=2**

The following Fortran code is the *model file* for this system. Changes to this file for the modified version of AUTO and comments are in italicised bold.

```
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C  Example Problem AUTBi2 : A Boundary Value Problem. for modified AUTO
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
      SUBROUTINE FUNC(T,S,NDIM,U,ICP,PAR,IJAC,F,DFDU,DFDP)
C     ---------- ----
C
C !!!! T is the variable t corresponding to u(t) and S is Δt0
C
C This subroutine evaluates the right hand side of the first order
C system and the derivatives with respect to (U(1),U(2)) and PAR(1).
C (For documentation see the example problem AUTPP2).
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL (A-H,O-Z)
C
      DIMENSION U(NDIM),PAR(20),F(NDIM),DFDU(NDIM,NDIM),DFDP(NDIM,20)
      DOUBLE PRECISION A,S,T,E,B
C
       B=3.0
C
C   B is j+1 where j is the shape factor
C
       E=EXP(U(1))
C
C       IF (A.NE.S) PRINT '(F8.4)',S
       A=S
       IF (T.GT.A) THEN
         F(1)=U(2)
         F(2)=(-(B-1.0)/T)*U(2) - PAR(1)*E
C        PRINT '(F8.3,F8.3,F8.3)',T,F(2),-0.5*PAR(1)*E
        ELSE
           F(1)=U(2)
           F(2)=-(1.0/B)*PAR(1)*E
        END IF
C
C   F is the system definition
C
       IF(IJAC.EQ.0)RETURN
C
       IF (T.GT.A) THEN
         DFDU(1,1)=0.0
         DFDU(1,2)=1
         DFDU(2,1)=-PAR(1)*E
         DFDU(2,2)=-(B-1.0)/T
C
         DFDP(1,1)=0.0
         DFDP(2,1)=-E
       ELSE
         DFDU(1,1)=0.0
         DFDU(1,2)=1
         DFDU(2,1)=-(1.0/B)*PAR(1)*E
         DFDU(2,2)=0.0
C
         DFDP(1,1)=0.0
         DFDP(2,1)=-(1.0/B)*E
        END IF
C
C  DFDU and DFDP are the derivatives of F
C
      RETURN
      END
C
```

```
      SUBROUTINE STPNT(NDIM,U,PAR,T)
C     ---------- -----
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL (A-H,O-Z)
C
C This subroutine must used to generate an initial starting point
C (i.e., when not restarting from a previously computed solution).
C The solution vector U must be given as a function of the independent
C variable T (T takes on values between zero and one).
C
C       NDIM   -   Dimension of the system of differential equations.
C       U      -   Vector of dimension NDIM.
C                  Upon return, U must contain a solution of the
C                  differential equation evaluated at 'time' T.
C       PAR    -   Array of parameters in the differential equations.
C                  These may be initialized here, or else in INIT.
C                  (STPNT is called after INIT).
C       T      -   Contains a value of the independent variable in \0,1\
C                  where the solution is to be evaluated.
C
C
      DIMENSION U(NDIM),PAR(20)
C
C (In this problem the starting solution is actually independent of t.)
C
       U(1)=0.0
       U(2)=0.0
       PAR(1)=0.0
C
C  This is the starting point on the branch
C
      RETURN
      END
C
      SUBROUTINE INIT
C     ---------- ----
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL (A-H,O-Z)
C
      COMMON /BLBCN/ NDIM,IPS,IRS,ILP,ICP(20),PAR(20)
      COMMON /BLCDE/ NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
      COMMON /BLDLS/ DS,DSMIN,DSMAX,IADS
      COMMON /BLLIM/ NMX,NUZR,RL0,RL1,A0,A1
      COMMON /BLMAX/ NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
C
C In this subroutine the user should set those constants that require
C values that differ from the default values assigned in DFINIT.
C (See the main documentation for the default assignments).
C
C
       NDIM=2
       IPS=4
       IRS=0
       ILP=1
       ICP(1)=1
       NTST=20
       NBC=2
       NINT=0
       DS=0.001
       DSMIN=0.00001
       DSMAX=1
       NPR=500
       JAC=1
       NMX=500
       NUZR=0
       RL0=0.0
       RL1=2000.0
       A0=0.0
       A1=2000.0
C
      RETURN
      END
C
```

```
      SUBROUTINE BCND(NDIM,PAR,ICP,NBC,U0,U1,FB,IJAC,DBC)
C     ---------- ----
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL (A-H,O-Z)
C
C This subroutine defines the boundary conditions.
C
C Supplied variables :
C
C      NDIM   :   Dimension of the first order autonomous system.
C      PAR    :   Vector of problem parameters.
C      ICP    :   Vector of indices of the free parameters.
C      NBC    :   Number of boundary conditions.
C      U0     :   Value of the vector function U at t=0.
C      U1     :   Value of the vector function U at t=1.
C      IJAC   :   IJAC=0    :   Derivatives need not be returned.
C                 IJAC=1    :   Derivatives must be returned also.
C
C Variables to be returned upon completion of call :
C
C      FB     :   The vector function defining the boundary conditions:
C
C                           FB ( U0 , U1 , PAR) = 0.
C
C      DBC    :   The Jacobian of the boundary conditions :
C
C                 DBC(i,j) : The derivative of the i'th boundary
C                  condition with respect to the j'th component of U0.
C
C                 DBC(i,NDIM+j) :  The derivative of the i'th boundary
C                 condition with respect to the j'th component of U1.
C
C                 DBC(i,2*NDIM+j) :  The derivative of the i'th boundary
C                 condition with respect to PAR(j) (For free parameters)
C
      DIMENSION PAR(20),ICP(20),U0(NDIM),U1(NDIM),FB(NBC),DBC(NBC,20)
C
C Define the two boundary conditions FB(1) and FB(2):
C
C
        B=2
C
C  B is the Bi number
C
       FB(1)=U0(2)
       FB(2)=U1(2)+B*U1(1)
C
C
      IF(IJAC.EQ.0)RETURN
C
C Set up the derivatives of the boundary conditions :
C
C With respect to U0    (U0 = U at 'time' T=0).
C
       DBC(1,1)=0.0
       DBC(1,2)=1.0
       DBC(2,1)=0.0
       DBC(2,2)=0.0
C
C With respect to U1    (U1 = U at 'time' T=1).
C
       DBC(1,3)=0.0
       DBC(1,4)=0.0
       DBC(2,3)=B
       DBC(2,4)=1.0
C
C With respect to the free parameter (Here PAR(1) ).
C
       DBC(1,5)=0.0
       DBC(2,5)=0.0
C
C     RETURN
      END
C
      SUBROUTINE ICND(NDIM,PAR,ICP,NINT,U,UOLD,UDOT,UPOLD,FI,IJAC,DINT)
C     ---------- ----
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL (A-H,O-Z)
C
C (This problem has no integral constraints.)
      RETURN
      END
```

```
C
        FUNCTION USZR(I,NUZR,PAR)
C       -------- ----
C
       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL (A-H,O-Z)
C
       DIMENSION PAR(20)
C
C This subroutine makes it possible to generate plotting and restart
C data at user-selected values of the free parameter(s).
C Plotting and restart data will be written in unit 8 at zeroes of
C functions defined below.
C
C
       USZR=0
C
C  not used in this example
C
        RETURN
C
       END
```

To run AUTO with this *model file* the following command was entered:

@auto Bi2

The screen then shows the results from compiling and running AUTO:

```
autBi2.f:
        func:
        stpnt:
        init:
        bcnd:
        icnd:
        uszr:
        fopt:
ld: Undefined symbol
    _MAIN_
autBi2.o: _bcnd_: multiply defined
autBi2.o: _fopt_: multiply defined
autBi2.o: _func_: multiply defined
autBi2.o: _icnd_: multiply defined
autBi2.o: _init_: multiply defined
autBi2.o: _uszr_: multiply defined
autBi2.o: _stpnt_: multiply defined
main.f:
 MAIN auto:

  BR    PT TY LAB     PAR(1)        L2-NORM       MAX U(1)       MAX U(2)
   1     1 EP   1  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
   1    15 LP   2  1.480028E+00  1.437947E+00  1.407336E+00  0.000000E+00
   1    23 LP   3  5.799899E-01  7.245770E+00  6.673988E+00  0.000000E+00
   1    48 LP   4  7.888911E-01  2.260699E+01  1.127426E+01  0.000000E+00
   1   126 LP   5  7.201040E-01  7.275749E+01  1.606952E+01  0.000000E+00
   1   383 LP   6  7.405557E-01  2.391925E+02  2.080520E+01  0.000000E+00
   1   500 EP   7  7.393332E-01  3.151358E+02  2.190991E+01  0.000000E+00
```

Note that there are 4 *limit points* labelled 2 to 6 and 2 end points labelled 1 and 7.

The file p.Bi2 Contains all the points found by the pathfollowing algorithm which form a branch (added comments are in italicised bold) :

```
{initial values for the starting point}
   0  0.0000E+00  2.0000E+03  0.0000E+00  2.0000E+03
   0 PAR(.): 0.0000E+00          0.0000E+00          0.0000E+00          0.0000E+00
   0         0.0000E+00          0.0000E+00          0.0000E+00          0.0000E+00
   0         0.0000E+00          0.0000E+00          0.0000E+00          0.0000E+00
   0         0.0000E+00          0.0000E+00          0.0000E+00          0.0000E+00
   0         0.0000E+00          0.0000E+00          0.0000E+00          0.0000E+00
{values for user defined constants}
   0       EPSU= 1.0000E-04       EPSS= 1.0000E-04   EPSL(1)= 1.0000E-04   EPSL(2)=
1.0000E-04
   0         DS= 1.0000E-03      DSMIN= 1.0000E-05      DSMAX= 1.0000E+00
   0     THETAU= 1.0000E+00 THETAL(1)= 1.0000E+00 THETAL(2)= 0.0000E+00
   0 NDIM=    2   IPS=    4   IRS=    0   ILP=    1
   0 NTST=   20  NCOL=    4   IAD=    3   ISP=    1   ISW=    1  IPLT=    0
   0  NBC=    2  NINT=    0   NMX=  500   NPR=  500  MXBF=    5   IID=    2
   0 ITMX=    8  ITNW=    5  NWTN=    3   JAC=    1  NUZR=    0
   0  ICP( . )=    1
   0
(point data for bifurcation graph)
   0   PT  TY LAB    PAR(1)         L2-NORM        MAX U(1)       MAX U(2)
   1    1   9   1  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
   1    2   0   0  9.462964E-04  3.233006E-04  3.155093E-04  0.000000E+00
   1    3   0   0  2.838744E-03  9.703262E-04  9.469428E-04  0.000000E+00
   1    4   0   0  6.623059E-03  2.266078E-03  2.211473E-03  0.000000E+00
   1    5   0   0  1.418935E-02  4.864411E-03  4.747214E-03  0.000000E+00
   1    6   0   0  2.931246E-02  1.008863E-02  9.845641E-03  0.000000E+00
   1    7   0   0  5.951991E-02  2.064912E-02  2.015211E-02  0.000000E+00
   1    8   0   0  1.047299E-01  3.677822E-02  3.589386E-02  0.000000E+00
   1    9   0   0  1.723010E-01  6.165143E-02  6.017126E-02  0.000000E+00
   1   10   0   0  2.730470E-01  1.006039E-01  9.819444E-02  0.000000E+00
   1   11   0   0  4.225442E-01  1.631678E-01  1.592757E-01  0.000000E+00
   1   12   0   0  6.420272E-01  2.681558E-01  2.618045E-01  0.000000E+00
   1   13   0   0  9.545536E-01  4.594868E-01  4.487539E-01  0.000000E+00
   1   14   0   0  1.341724E+00  8.741804E-01  8.544567E-01  0.000000E+00
   1   15   5   2  1.480028E+00  1.437947E+00  1.407336E+00  0.000000E+00
   1   16   0   0  1.289008E+00  2.430124E+00  2.384808E+00  0.000000E+00
   1   17   0   0  1.011955E+00  3.378580E+00  3.323067E+00  0.000000E+00
   1   18   0   0  8.021947E-01  4.288009E+00  4.216039E+00  0.000000E+00
   1   19   0   0  6.777413E-01  5.120949E+00  5.004512E+00  0.000000E+00
   1   20   0   0  6.150687E-01  5.861983E+00  5.654113E+00  0.000000E+00
   .
   ..
   ...
   1  499   0   0  7.393484E-01  3.144875E+02  2.190164E+01  0.000000E+00
   1  500   9   7  7.393332E-01  3.151358E+02  2.190991E+01  0.000000E+00
```

This has **all** the point data. Note that the type is a number and not a Character code. type 5 points are limit points, type 9 end points, and 0 are regular points.

Each labelled point in p.Bi2 has an entry in q.Bi2 containing all the information needed to continue from those point in future runs:

```
{initial parameters for continuation}
    1    1    9    1    1    1   81    3  166   20    4  1
{values for t and u at discrete point t in [0,1] }
       0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
       1.2500000000E-02  0.0000000000E+00  0.0000000000E+00
       2.5000000000E-02  0.0000000000E+00  0.0000000000E+00
       3.7500000000E-02  0.0000000000E+00  0.0000000000E+00
       5.0000000000E-02  0.0000000000E+00  0.0000000000E+00
       6.2500000000E-02  0.0000000000E+00  0.0000000000E+00
       7.5000000000E-02  0.0000000000E+00  0.0000000000E+00
       8.7500000000E-02  0.0000000000E+00  0.0000000000E+00
       1.0000000000E-01  0.0000000000E+00  0.0000000000E+00
       1.1250000000E-01  0.0000000000E+00  0.0000000000E+00
       .
       ..
       ...
       8.7500000000E-01  0.0000000000E+00  0.0000000000E+00
       8.8750000000E-01  0.0000000000E+00  0.0000000000E+00
       9.0000000000E-01  0.0000000000E+00  0.0000000000E+00
       9.1250000000E-01  0.0000000000E+00  0.0000000000E+00
       9.2500000000E-01  0.0000000000E+00  0.0000000000E+00
       9.3750000000E-01  0.0000000000E+00  0.0000000000E+00
       9.5000000000E-01  0.0000000000E+00  0.0000000000E+00
       9.6250000000E-01  0.0000000000E+00  0.0000000000E+00
       9.7500000000E-01  0.0000000000E+00  0.0000000000E+00
       9.8750000000E-01  0.0000000000E+00  0.0000000000E+00
       1.0000000000E+00  0.0000000000E+00  0.0000000000E+00
{value of λ' for direction vector x'}
       9.4632044681E-01
{value of u' for direction vector x'}
       3.1544014894E-01  0.0000000000E+00
       3.1541550518E-01 -3.9430018617E-03
       3.1534157389E-01 -7.8860037235E-03
       3.1521835508E-01 -1.1829005585E-02
       3.1504584875E-01 -1.5772007447E-02
       .
       ..
       ...
       1.8768688862E-01 -2.8389613404E-01
       1.8411354318E-01 -2.8783913591E-01
       1.8049091022E-01 -2.9178213777E-01
       1.7681898974E-01 -2.9572513963E-01
       1.7309778173E-01 -2.9966814149E-01
       1.6932728620E-01 -3.0361114335E-01
       1.6550750315E-01 -3.0755414521E-01
       1.6163843257E-01 -3.1149714708E-01
       1.5772007447E-01 -3.1544014894E-01
```

... followed by the next point...

d.BI2 contains information from the Newton Corrector method at each point on the branch:

```
NUMBER OF ITERATIONS =  1   NEXT STEPSIZE =     0.100E+01
BRANCH  1 N=  14 IT= 0 A= 7.44985501E-01  PAR=  1.42082915E+00
BRANCH  1 N=  14 IT= 1 A= 8.75296843E-01  PAR=  1.34103914E+00
BRANCH  1 N=  14 IT= 2 A= 8.74180420E-01  PAR=  1.34172434E+00
BRANCH  1 N=  14 IT= 3 A= 8.74180374E-01  PAR=  1.34172437E+00
LIMIT POINT FUNCTION =   0.493E+00
NUMBER OF ITERATIONS =  3   NEXT STEPSIZE =     0.547E+00
```

This example point (number 14 on Branch 1) has 3 Newton iterations with λ converging to 1.32, and the next stepsize is 0.547 (based on the number of Newton iterations).

The limit point Function changes sign when there is a limit point. For example:

```
BRANCH  1 N=  14 IT= 0 A= 7.44985501E-01  PAR=  1.42082915E+00
BRANCH  1 N=  14 IT= 1 A= 8.75296843E-01  PAR=  1.34103914E+00
BRANCH  1 N=  14 IT= 2 A= 8.74180420E-01  PAR=  1.34172434E+00
BRANCH  1 N=  14 IT= 3 A= 8.74180374E-01  PAR=  1.34172437E+00
LIMIT POINT FUNCTION =   0.493E+00
NUMBER OF ITERATIONS =  3   NEXT STEPSIZE =     0.547E+00

BRANCH  1 N=  15 IT= 0 A= 1.27386060E+00  PAR=  1.71467895E+00
BRANCH  1 N=  15 IT= 1 A= 1.50947747E+00  PAR=  1.46203950E+00
BRANCH  1 N=  15 IT= 2 A= 1.49366043E+00  PAR=  1.47896813E+00
BRANCH  1 N=  15 IT= 3 A= 1.49357184E+00  PAR=  1.47906290E+00
LIMIT POINT FUNCTION =  -0.339E-01
* DETECTION OF SINGULAR POINT : ITERATION   0 STEPSIZE = -0.352E-01

BRANCH  1 N=  15 IT= 0 A= 1.45923601E+00  PAR=  1.47145998E+00
BRANCH  1 N=  15 IT= 1 A= 1.45748977E+00  PAR=  1.47990434E+00
BRANCH  1 N=  15 IT= 2 A= 1.45748950E+00  PAR=  1.47990631E+00
LIMIT POINT FUNCTION =  -0.123E-01
* DETECTION OF SINGULAR POINT : ITERATION   1 STEPSIZE = -0.200E-01

BRANCH  1 N=  15 IT= 0 A= 1.43764409E+00  PAR=  1.48037031E+00
BRANCH  1 N=  15 IT= 1 A= 1.43762453E+00  PAR=  1.48002772E+00
BRANCH  1 N=  15 IT= 2 A= 1.43762453E+00  PAR=  1.48002769E+00
LIMIT POINT FUNCTION =   0.203E-03
* DETECTION OF SINGULAR POINT : ITERATION   2 STEPSIZE =  0.324E-03

BRANCH  1 N=  15 IT= 0 A= 1.43794692E+00  PAR=  1.48002572E+00
BRANCH  1 N=  15 IT= 1 A= 1.43794700E+00  PAR=  1.48002773E+00
LIMIT POINT FUNCTION =  -0.168E-05
* DETECTION OF SINGULAR POINT : ITERATION   3 STEPSIZE = -0.266E-05
NUMBER OF ITERATIONS =  1   NEXT STEPSIZE =     0.100E+01
```

The limit point function changed sign and the Newton Corrector method was used to converge on the place where it changed sign.

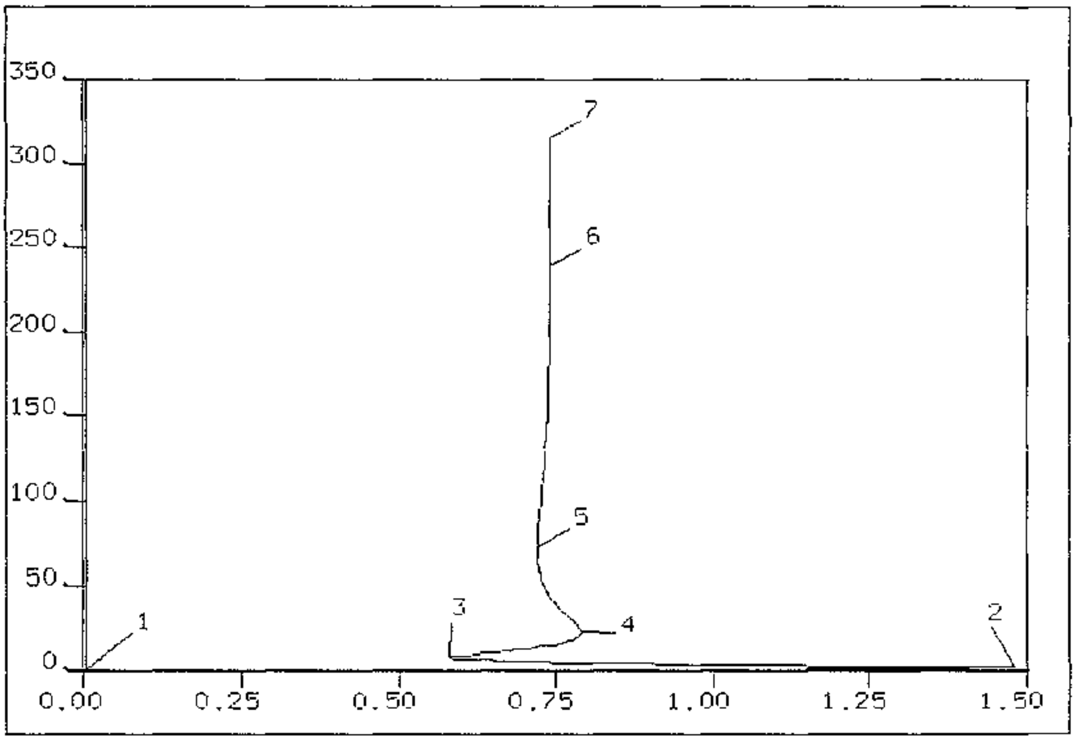
These output files can be used to generate a bifurcation graph using the **plaut** program. First a tektronics window is set up by typing **tektool**. The plaut program is initiated by the command:

@plaut Bi2

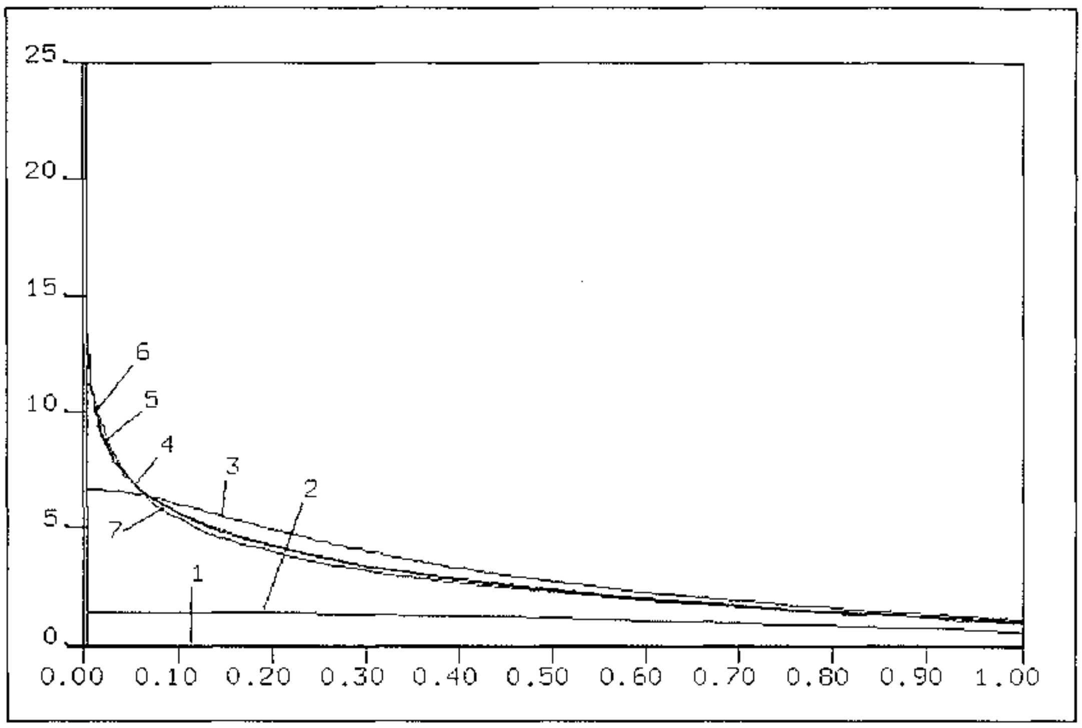
The screen then shows:

```
ENTER <HELP> IN CASE OF DIFFICULTY
ENTER COMMAND
```

typing BD0 will result in a plot of a bifurcation diagram for the system:



Typing 2D gives the curve $u(t)$ at the label points 1, 2, 3, 4, 5, 6, 7 and 8:



APPENDIX B: THE STRUCTURE OF AUTO

INTERNAL SYSTEM CONSTANTS

The smallest and largest acceptable real numbers for the machine are defined as:

RSMALL=1.0D -30 (1×10^{-30} in double precision)

RLARGE=1.0D 30 (1×10^{30} in double precision)

These numbers are used to stop underflow and overflow errors. For example, in Gauss elimination, the pivot element must be larger than RSMALL, otherwise a divide-by-zero error will result.

The machine has a 14 decimal digit mantissa. HMACH=1.0D-7 is the approximate half exponent machine accuracy. This is used as Δ in (2.46) for numerically generating the Jacobian, HMACH can be multiplied by itself 4 times, before the number goes out of range. This ensures that as long as the function *f* is within the range of the machine, then the resulting number from (2.46) is also within the accuracy of the machine.

Every IADS steps along the branch (where IADS is a user defined constant), the stepsize is adapted. If the number of Newton iterations to find a point is NIT, the existing stepsize is RDS and ITNW is the maximum number of Newton iterations (ν_{max} in Section 2.1) then:

- If NIT<= 1 Then RDS=2(RDS)
- If NIT=2 Then RDS=3/2 (RDS)
- If NIT>2 and NIT< ITNW/2 Then RDS=1.1(RDS)
- If NIT >=ITNW Then RDS=0.5(RDS)

These settings are fixed by AUTO and cannot be changed in the model file.

FLOW CHART

A Flow Chart is included for AUTO when the *main program* is set up for a BVP (IPS=4) with 2 boundary conditions (NBC=2), no integral conditions (NINT) and with no previous runs used (IRS=0) This is the type of analysis done for all of BVP's in section 3.2.

The program is represented as a text-based flow-chart with the convention:

EXAMPLE - Name of Routine

EXAMPLE - A Subroutine call

EXAMPLE - A Subroutine call which contains other subroutine calls

Example or EXAMPLE - Logic statements like if, then and loops

Example - A Comment

MAIN PROGRAM
CNSTNT:
    Define problem independent constants e.g. maximum and minimum computable numbers
DEFINIT:
    Sets default values of the constants. (These values are on page 143 of the manual.)
INIT:
    Define constants from the users *model file*
AUTOBV:
    Pathfollow for Boundary Value problems

AUTOBV
WSBV:
    Assign workspace.i.e declare variables
CNRLBV:
    Compute solution branches

```
CNRLBV
ntot=0
RSPTBV:
    Gets initial value for $u(t)$ and $\lambda$, defined in STPNT in the model file and put them
    into discrete variables UPS(i,j), and RL(1) respectively.
    get $t$ values for these points
    ADAPT:          {Adapt existing Mesh to this initial solution curve.}
STDRBV: {get starting direction vector}
        SETUBV:
            Generate Jacobian Matrix $P_x$ from UPS values with $N_x$=0 (i.e. using a 0
            direction vector) to get $\{F_u \mid F_\lambda\}$
            Set right hand side of Matrix equation to 0
        BRBD:
            Find null vector of $\{F_u \mid F_\lambda\}$ to get an initial direction vector (UDOTPS,
RLDOTPS)
        SCALEB:
            Scale this direction vector to unit length.
STHD:
    Write the values of the AUTO user defined parameters in fort.7.
EXTRBV:     {Extrapolate to get approximation to next point}

LOOP UNTIL {the point evaluated is out of range} OR {the number of points is out of
range}
    IF {adaptive mesh chosen (IAD=1)} AND {it has been IAD iterations since it has
    been adapted} THEN
        ADAPT:      {Adapt existing mesh}
    IF {adaptive stepsize chosen (IADS=1)} AND {it has been IADS iterations since it has
    been adapted} THEN
        ADPTDS:
            Adapt existing stepsize
    CONTBV: {Use Euler Predictor as initial approximation for next point}
        Use 2 previous points UPOLDS, RLOLD and UPS,RL to approximate the next
        direction vector using the stepsize DDS.
        EXTRBV:     {Extrapolate to get approximation to next point}
    SOLVEBV:
        Perform the Newton Corrector using initial approximation
    IF {Checking for limit points ILP>0} THEN
        LCSPBV(FNLPBV)
    IF {Checking for branching points ISP>0} THEN
        LCSPBV(FNBPBV)

    IF {Checking for user defined points NUZR>0} THEN
        LCSPBV(FNUZBV)
    STPLBV
        Store plotting data to fort.7 and fort.8
END OF LOOP
```

ADAPT

```
    NEWMESH:
        Use last solution points $u(t)$ to get new mesh intervals (As in Chapter 2)
    INTERP:
        Replace last solution curve points by new point by interpolating to get u
        values for the new mesh interval's
```

```
EXTRBV:
    Define UOLDPS,RLOLD as UPS and RL respectively.
    get next point UPS,RL from previous point UOLDPS, RLOLD and direction vector
    UDOTPS,RLDOT

SOLVEBV
    LABEL 1
    LOOP for NIT=1 to ITNW
        Define IFST=0
        IF {Number of iterations is less than user variable NWTS} THEN
            Define IFTS=1
        IF IFTS =1 THEN
            SETUBV:
                Generate Jacobian $P_x$ and Generate Right Hand side $(F,N)^t$
        ELSE
            SETRBV:
                Only Generate the Right Hand side (using Jacobian at the previous
                point)

        BRBD:
            Solve the Matrix equation $P_x=(F,N)^t$ to get solution RHSD
        Use RHSD as a Newton increment to get a better approximation for UPS,RL
        Check whether user-supplied tolerances for the new solution have been met
        and QUIT if they have.

    The Maximum number of iterations has been reached. so Reduce stepsize (if
    adaptive)
    IF minimum stepsize not reached THEN
        GOTO label 1
    ELSE
        Display error message - NO convergence.


SETUBV
    Generate U values at z values using weights W(i,j)
    FUNI:
        Call user subroutine FUNC in the Model file and get F(U,RL) and derivatives
        DFDU and DFDP
    Use DFDU and DFDP for Jacobian $P_x$
    Use F and the Pseudo-arclength equation for N for the Right hand side
```

MODIFICATIONS TO AUTO

The co-ordinates i and j of the u value $u_{j+i/m}$ are passed to the Subroutine FUNI, as well as the current mesh points TM ($t_{j+i/m}$) and interval sizes DTM (Δt_j). Another parameter k indicates whether the mesh points are at regular intervals or at z values.

The following Fortran Code shows the modifications to the subroutine FUNI:

```
FUNI(K,I,J,NDIM,U,UOLD,ICP,PAR,IJAC,F,DFDU,DFDP,TM,DTM)
C
C !!!!!!This has been modified, K says whether it is z-values or not
C       Parameters I and J have been added
C       J is the MESH step number and I is the Collocation point number
C      TM(J)+(I-1)/NCOL*DTM(J) is the T value which corresponds to be value of T
C       also as arrays TM and DTM are also needed they are passed as well.
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CSGLE IMPLICIT REAL            (A-H,O-Z)
C
C Interface subroutine to user supplied FUNC.
C
      COMMON /BLICN/ NDM,NDMP1,NROW,NCLM,NRC,NCC,NPAR,NFPAR,NBC0,NINTC
      COMMON /BLMAX/ NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
      COMMON /BLRCN/ HALF,ZERO,ONE,TWO,HMACH,RSMALL,RLARGE
      COMMON /BLDIF/ U1ZZ(50),U2ZZ(50),F1ZZ(50),F2ZZ(50)
      COMMON /BLCDE/ NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
C !!!!!! Note that NTST is needed
C
      COMMON /BLWTS/ W(8,7),WP(8,7),WH(8),WI(8),ZM(7)
C  !!!!!! new common variables added to get z values defined initially
C          for weights
C
      DIMENSION U(NDIM),UOLD(NDIM),ICP(20),PAR(20),F(NDIM)
      DIMENSION DFDU(NDIM,NDIM),DFDP(NDIM,20),TM(NTSTP1),DTM(NTSTP1)

      DOUBLE PRECISION A1,A,B,C
      INTEGER I,J,K,L
C
C Generate the function.
       L=I
       A1=(I-1)
       A=A1/NCOL
      IF (L.EQ.ZERO) THEN
           B=TM(J)
        ELSE
           IF (K.EQ.ZERO) THEN
              B=TM(J) + A*DTM(J)
           ELSE
               B=TM(J)+DTM(J)*ZM(L)
           ENDIF
        ENDIF
       C=DTM(1)
C
      CALL FUNC(B,C,NDIM,U,ICP,PAR,IJAC,F,DFDU,DFDP)
C !!!!!! B is the Value of T, and C is the first mesh interval length
C
.
..
...
```

APPENDIX C: AUTO AND ORDINARY DIFFERENTIAL EQUATIONS

The pathfollowing algorithm can be applied to systems of Ordinary Differential Equations:

$$\frac{dx}{dt} = F(x,\lambda)$$

The steady states of these systems, correspond to Solving $F(x,\lambda) = 0$. The stability of a solution point corresponds to the sign of the eigenvalues of F_u. If the eigenvalues are all negative , then the branch is stable, otherwise, the branch is stable. If the eigenvalues are complex with the real part zero, then the point is a Hopf point. Hopf points can be detected in this way.

Periodic solutions from Hopf points, can be gained by forcing the period to be 1 adding boundary condition u(0) = u(1) , and an integral condition which minimises the phase change.

The manual [4] has more information on how it is implemented, and how to use AUTO to get steady states and periodic solutions of O.D.E.'s

REFERENCES

[1] Boddington T., Gray P., Harvey D.I.,
Thermal theory of spontaneous ignition: Criticality in bodies of arbitrary shape.
Philos. Trans. R. Soc., A270 (1971) , 467-506

[2] Burden R. L., Faires J. D.,
Numerical Analysis
Third Edition, Prindle, Webber & Schmidt, Boston, 1985

[3] Chow S., Hale J. K.,
Methods of bifurcation theory.
Springer-Verlag New York Heidelberg Berlin, 1982

[4] Doedel E.,
AUTO: Software for continuation and bifurcation problems in ordinary differential equations.
Latex Document, AUTO software, Cornell University (I.P. 128.84.237.12), 1986

[5] Gomez, A.J.,
Friction and Localised heat initiation of ignition: the asymmetrical slab and cylindrical annulus.
Masters Thesis, Victoria University, 1986

[6] Gray B. F., Griffiths J. F., Hasko S. M.,
Spontaneous ignition hazards in stockpiles of cellulosic materials: criteria for safe storage.
J. Chem.Tech. Biotechnol. 34A (1984), 453-463

[7] Gray P., Lee P. R.,
Thermal Explosion Theory.
in: Oxidation and Combustion Reviews, Vol 2, Elsevier Publishing Company, C.F.H. Tipper, ed., Amsterdam / London / New York, 1967

[8] Keller H.B,
Numerical solution of bifurcation and nonlinear eigenvalue problems.
in: Applications of bifurcation theory, P. H. Rabinowitz, ed., Academic press, 1977, 359-384

[9] Keller J. B., Antman S.,
Bifurcation theory and nonlinear eigenvalue problems.
W. A. Benjamin, INC. New York 10016, 1969

[10] Sisson R. A., Swift A., Wake G.C.,
Spontaneous ignition of materials on hot surfaces.
Math Engng. Ind. 2 (1990), 287-301

[11] Sisson R. A.,
On mathematical modelling of the self-heating of cellulosic materials.
PhD Thesis , Massey University, 1991

[12] Temme N.M.,
Nonlinear analysis (Vol 1).
Mathematisch Centrum, 1976

[13] Temme N.M.,
Nonlinear analysis (Vol 2).
Mathematisch Centrum, 1976

[14] Wake G. C., Hood M. J.,
Multiplicity of solutions of a quasilinear elliptical equation in spherical domains
forthcoming book titled: Similarity, symmetry and other methods for nonlinear boundary problems, Academic Press, ≥ 1992.

[15] Wake G. C., Jackson F. H..,
The heat balance in spontaneous ignition
New Zealand Journal of Science, 1976, Vol 19, 1976, 23-27