THE PREDICTED SEQUENTIAL REGULARIZATION METHOD FOR DIFFERENTIAL-ALGEBRAIC EQUATIONS

by

Colin Barr Macdonald

Thesis

submitted in partial fulfillment of

the requirements of the Degree of

Bachelor of Science with Honours in

Mathematics with Computer Science.

Acadia University

May 2001

This thesis by Colin B. Macdonald

is accepted in its present form by the

Department of Mathematics and Statistics

as satisfying the thesis requirements for the Degree of

Bachelor of Science with Honours.

Approved by the Thesis Supervisor

Dr. R. Spiteri Date

Approved by the Head of the Department

Dr. T. Archibald Date

Approved by the Honours Committee

Date

I, Colin B. Macdonald, hereby grant permission to the University Librarian at Acadia University to provide copies of the thesis, on request, on a non-profit basis.

Signature of Author

Signature of Supervisor

Date

Acknowledgements

The research for this thesis was supported financially by NSERC. I would like to thank all those people who assisted or encouraged me throughout this endeavor, particularly Dr. F. Chipman for providing access to computer hardware. I especially thank my supervisor, Dr. R. Spiteri, for his continuing assistance and infectious enthusiasm towards applied mathematics.

Table of Contents

Abstract

Differential-algebraic equations (DAEs) can be used to describe the evolution of many interesting and important systems in a variety of disciplines. Common examples include multibody systems as an aspect of mechanical engineering, biomechanics, or robotics; Navier-Stokes equations for incompressible fluid dynamics; and Kirchoff's laws from electrical engineering applications. Most of these problems present particular analytical and numerical difficulties due to the algebraic constraints that distinguish DAEs from ordinary differential equations.

This thesis involves the implementation of two recent numerical methods for solving DAEs: the Sequential Regularization Method (SRM) and the related Predicted Sequential Regularization Method (PSRM). The resulting implementation breaks new ground by the addition of automatic step-size control code to the SRM. This is an important step in the evolution of the SRM/PSRM from experimental code to useful software.

It is shown that although all DAEs can be converted into ODEs, it is usually not desirable to do so because the numerical stability of the system is undermined in the process. The techniques of regularization and stabilization are discussed as they play an important role in the SRM/PSRM. The derivation of the SRM/PSRM and the associated algorithms are covered in some detail. The implementation is discussed including a section on how to use the software to solve additional problems.

The theoretical orders of the SRM and PSRM are tested on several problems with known solutions. Finally, comparisons are made between existing SRM codes and the new implementation as well as experimental comparisons between the SRM and PSRM.

Chapter 1

Background

1.1 Introduction

Differential-Algebraic equations (DAEs) are systems of differential equations where the unknown functions satisfy additional algebraic equations. They arise in many areas of science and engineering such as robotics (via Lagrange's equations with interdependent coordinates), biomechanics, control theory, electrical engineering (via Kirchoff's laws), and fluid dynamics (via the Navier-Stokes equations for incompressible flows). DAEs present both numerical and analytical difficulties as compared with systems of ordinary differential equations (ODEs). This has made them an active area of research for engineers, applied mathematicians, and numerical analysts ever since the 1960's when the advantages of working with DAEs directly rather then attempting to convert them to ODEs was first recognized (see, [6, 18] and below). In particular, since the mid 1980's there has been a flurry of research regarding the numerical solution of DAEs. The Sequential Regularization Method and the related Predicted Sequential Regularization Method are recent numerical methods designed to deal with certain classes of DAEs, specifically index-2 problems with and without singularities and index-3 problems that arise from so-called multibody systems.

In this chapter, the general definition of a system of DAEs is presented and then narrow our focus to those DAEs in *Hessenberg form* of sizes 2 and 3. The concept of *index* as a characterization of DAEs is briefly mentioned. We define the concept of *multibody systems* as they will provide "real life" examples of index-3 problems studied in later chapters. Several analytical techniques for reformulating and solving DAEs are introduced and a discussion of the problems associated with these methods gives a sense of the special difficulties presented by DAEs. In particular, regularization and the Baumgarte stabilization technique are particularly relevant to the Sequential Regularization Method.

1.2 Differential-Algebraic Equations

This section presents definitions of DAEs in various forms and introduces the notion of *index*. This material of this section is covered nicely in the beginning chapters of [6] and is presented here for convenience.

Definition 1 (General DAE) *The general or* fully implicit *DAE is a vector function of the form*

$$F(x(t), x'(t), t) = 0, \tag{1.1}$$

where $t \in \mathbb{R}$, $x \in \mathbb{R}^n$, $F : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, $\frac{\partial F}{\partial x} = F_x$ *(i.e., the Jacobian of* F *with respect to* x*) is nonsingular, and* $\frac{\partial F}{\partial x'} = F_{x'}$ *is singular for all* $x(t), x'(t), t$.

It is this last property that separates DAEs from implicit ODEs; it means that there are *algebraic constraints*[1] on $x(t)$ in addition to the usual differential equations. In other words, the components of $x(t)$ are not all independent.

[1] Loosely speaking, at least one component of $x'(t)$, say $x'_j(t)$, does not appear in F in any significant way (because $F_{x'}$ is always singular) but because F_x is nonsingular, $x_j(t)$ must affect F in some way.

As is the case with both analytical and numerical solutions of ODEs, it is important to classify DAEs (and especially nonlinear DAEs) into forms with known properties and/or methods of solution. In particular, not all DAEs can be solved numerically by classical methods such as backward differentiation formulas (BDF) (see [10]) or implicit Runge-Kutta schemes (see, e.g., [13]). Thus it is desirable to have structural forms which predict when a DAE will be *solvable*[2]. One such classification that is of particular interest to us is the *Hessenberg form of size r*.

Definition 2 (Hessenberg Form) *The DAE* $F(x, x', t) = 0$ *is in* Hessenberg Form of size r *if it is written in the form*

$$
\begin{aligned}
x_1' &= F_1(x_1, x_2, \ldots, x_r, t), \\
x_2' &= F_2(x_1, x_2, \ldots, x_{r-1}, t), \\
&\ \vdots \\
x_i' &= F_i(x_{i-1}, x_i, \ldots, x_{r-1}, t), \quad i = 3, 4, \ldots, r-1, \\
&\ \vdots \\
0 &= F_r(x_{r-1}, t),
\end{aligned}
$$

and $\left(\frac{\partial F_r}{\partial x_{r-1}}\right)\left(\frac{\partial F_{r-1}}{\partial x_{r-2}}\right)\cdots\left(\frac{\partial F_2}{\partial x_1}\right)\left(\frac{\partial F_1}{\partial x_r}\right)$ *is nonsingular.*

Assuming that the F_i are sufficiently differentiable (i.e., certain rth-order partial derivatives exist), then a DAE in the Hessenberg form of size r is *solvable* and has index r (a definition of index follows in Section 1.2.1).

In this thesis, we concentrate on DAEs in the Hessenberg forms of size two and three. We define these as follows:

[2] A precise definition of *solvability* is outside the scope of this thesis; one may wish to consult [6] for more information.

Definition 3 *A DAE in the form*

$$x_1' = F_1(x_1, x_2, t), \tag{1.2a}$$

$$0 = F_2(x_1, t), \tag{1.2b}$$

with $(\frac{\partial F_2}{\partial x_1})(\frac{\partial F_1}{\partial x_2})$ *nonsingular is said to be in the* Hessenberg form of size two. *Similarly, a DAE in the form*

$$x_1' = F_1(x_1, x_2, x_3, t), \tag{1.3a}$$

$$x_2' = F_2(x_1, x_2, t), \tag{1.3b}$$

$$0 = F_3(x_2, t), \tag{1.3c}$$

with $(\frac{\partial F_3}{\partial x_2})(\frac{\partial F_2}{\partial x_1})(\frac{\partial F_1}{\partial x_3})$ *nonsingular is said to be in the* Hessenberg form of size three.

Specifically, for the methods discussed in this thesis, we consider *semi-explicit time-varying DAEs* of the form

$$x' = f(x, t) - B(x, t)y, \tag{1.4a}$$

$$0 = g(x, t), \tag{1.4b}$$

where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^{n_y}$, $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ and $g : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n_y}$ are vector functions and $B : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n \times \mathbb{R}^{n_y}$ is a matrix function. Any of these functions may be nonlinear. We note that a system in either Hessenberg form can be written in the form of (1.4); however, the converse is not true: given an arbitrary system in the form of (1.4), it is not necessarily possible to place the system in a Hessenberg form. Thus strictly speaking, *we are interested in those DAEs in the form of (1.4) that can be expressed in Hessenberg form of size two or three where the constraint Jacobian is non-singular almost everywhere* (see Section 1.4).

1.2.1 Index of a DAE

An important concept for us is that of the *index* of a DAE. It characterizes a DAE and quantifies its level of difficulty (for example, the class of index-0 DAEs is the set of all ODEs). Higher-index problems are more difficult than lower-index problems and indeed, index-3 problems and above are *ill-posed* in the sense that a small perturbation to the problem results in a large change in the solution (see also Section 1.5.2). The sensitivity of solutions with respect to such perturbations is known as the *perturbation index* (see [11]).

Definition 4 (Index of a General DAE) *The* index *of a DAE is the minimum number of times that all or part of (1.1) must be differentiated with respect to t to determine x' as a continuous function of x and t.*

The index of a DAE in the form of (1.4) is the number of times that (1.4b) must be differentiated with respect to t to obtain a differential equation for y. Note that in general this will involve substituting in (1.4a) one or more times.

1.3 Multibody Systems

Multibody systems (MBS) are collections of *rigid bodies* which are interconnected by *joints* such as hinges, ball joints, and sliders to restrict their relative motions. Examples of MBS include the simple pendulum (Figure 1.2) and the planar slider crank (Figure 1.1), as well as many other complicated systems in mechanical engineering and related fields such as biomechanics and robotics. We consider the equations for the motion of MBS in standard form; that is, those expressed by the index-3 DAE

$$p' = v, \tag{1.5a}$$

$$M(p,t)v' = f(p,v,t) - G^T(p,t)y, \tag{1.5b}$$

$$0 = g(p,t), \tag{1.5c}$$

where $p, v \in \mathbb{R}^{\frac{n}{2}}$ are respectively the positions and velocities of the bodies stacked to make up the differential variable $x = \binom{p}{v}$; $y \in \mathbb{R}^{n_y}$ is the algebraic variable of the DAE; $g(p,t) = 0$ is the constraint equation and $G(p,t) = \frac{\partial g}{\partial p}$; $M(p,t)$ is a positive definite matrix describing the generalized masses of the bodies (in fact M will often be constant). The term $G^T(p,t)y$ provides the reaction forces that "hold the system together" — that is, y acts as a vector of Lagrange multipliers which enforces the constraint equation. Finally, we note that if $x_1 = v$, $x_2 = p$, and $x_3 = y$, then (1.5) is in Hessenberg form of size three.

Many of the applications we consider are MBS as they provide real-world examples of the types of DAEs we wish to solve. However, because MBS are index-3 problems, they present additional difficulties over index-2 problems that are related to numerical stability (see Section 2.5).

For a wealth of information on the subject of MBS simulation consult von Schwerin's book, *MultiBody System SIMulation* [18].

1.4 Multibody Systems with Constraint Singularities

Suppose that for some isolated t, say t^*, $G(x,t^*)$ becomes singular. This can correspond to various phenomena which are difficult to deal with numerically such as impasse points or bifurcation points. On the other hand, such a t^* may simply be a mathematical artifact (though numerically this does not make them any easier

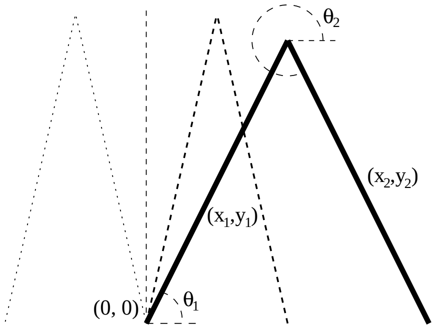


Figure 1.1: The planar slider-crank. An initial condition is shown in bold and future conditions are shown in dashed lines.

to handle!). We will motivate such problems by considering the planar slider-crank problem.

1.4.1 The Planar Slider-crank

The two-link planar slider-crank (see Figure 1.1) is a mechanical system consisting of two linked bars, each of which we take to have length 2. One of the bars is fixed at the origin such that only rotational motion is allowed. The end of the other bar is fixed in such a way as to allow translational motion along the x-axis. The planar slider-crank is similar to the mechanism used to drive steam locomotives except that in that case, the bars are of unequal length (generally leading to a more well-behaved problem if the crank arm is shorter then the slider arm).

The motion of the center of mass of the ith bar of the slider crank, x_i, y_i, and its orientation θ_i are governed by (1.5) with

$$p = \begin{pmatrix} x_1 \\ y_1 \\ \theta_1 \\ x_2 \\ y_2 \\ \theta_2 \end{pmatrix}, \quad f = \begin{pmatrix} 0 \\ -\mathfrak{g} \\ 0 \\ 0 \\ -\mathfrak{g} \\ 0 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \end{pmatrix},$$

$$g = \begin{pmatrix} x_1 - \cos\theta_1 \\ y_1 - \sin\theta_1 \\ x_2 - 2x_1 - \cos\theta_2 \\ y_2 - 2y_1 - \sin\theta_2 \\ 2\sin\theta_1 + 2\sin\theta_2 \end{pmatrix}, \quad G = \begin{pmatrix} 1 & 0 & \sin\theta_1 & 0 & 0 & 0 \\ 0 & 1 & -\cos\theta_1 & 0 & 0 & 0 \\ -2 & 0 & 0 & 1 & 0 & \sin\theta_2 \\ 0 & -2 & 0 & 0 & 1 & -\cos\theta_2 \\ 0 & 0 & 2\cos\theta_1 & 0 & 0 & 2\cos\theta_2 \end{pmatrix},$$

where $\mathfrak{g}$ is the gravitational constant.

When the bars are in a vertical configuration (i.e., when $\theta_1 = \frac{\pi}{2}, \theta_2 = \frac{3\pi}{2}$ or when $\theta_1 = \frac{3\pi}{2}, \theta_2 = \frac{\pi}{2}$), the matrix G becomes singular (note the last row vanishes). If the system reaches the upright vertical configuration with no momentum, then a bifurcation occurs: the bars may rotate as one about the origin, or the end of the second bar may slide left or right along the x-axis. We will not be concerned with such configurations but rather the case where momentum carries the sliding bar through the singularity in a smooth fashion. The solutions to such situations may be obvious in practice compared to the zero momentum case, but mathematically they present similar challenges.

Methods for the numerical solution of DAEs which ignore the singularity, such as *regularization* as discussed in Section 1.6, produce unpredictable results for problems with constraint singularities. Many existing methods will be unable to penetrate the singularity at all; they may output a solution which approaches the singularity and then reverses direction back towards the initial position!

We will examine problems where the solution is smooth through isolated singularities as they relate to the SRM and PSRM again in Section 2.4.

1.5 Index-Reduction Methods

A naïve method for solving DAEs can be constructed based on reducing the index of the problem through repeated differentiation of the constraint equation. Once an

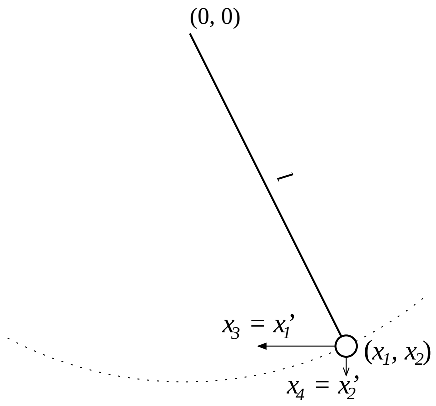


Figure 1.2: A simple pendulum.

index-0 DAE is obtained[3] then the problem has been converted from a DAE to a system of ODEs in both x and y, and can be solved as such. That is, it can be solved numerically with an established ODE solver such as MATLAB's `ode45`. As the following example will illustrate, solution methods based on index reductions introduce mild numerical instabilities to the problem.

1.5.1 A Simple Pendulum Example

Consider a simple pendulum in Cartesian coordinates (see Figure 1.2) consisting of a ball of unit mass at position $\binom{x_1}{x_2}$ with velocity $\binom{x_3}{x_4}$ suspended from an arm of unit length. First, note that by Newton's second law, the unconstrained motion of the ball would be described by

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix}, \tag{1.6a}$$

$$\begin{pmatrix} x_3 \\ x_4 \end{pmatrix}' = \begin{pmatrix} 0 \\ -\mathfrak{g} \end{pmatrix}. \tag{1.6b}$$

[3]The conditions for this to be possible are similar to the non-singularity conditions on Hessenberg forms of size 2 and 3. Specifically, for index-2 problems, GB must be invertible.

Constrained by the arm of the pendulum, the motion of the ball would be expressed as a MBS (in the form of (1.5)) as

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix}, \tag{1.7a}$$
$$\begin{pmatrix} x_3 \\ x_4 \end{pmatrix}' = \begin{pmatrix} 0 \\ -\mathfrak{g} \end{pmatrix} - \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} y, \tag{1.7b}$$
$$0 = x_1^2 + x_2^2 - 1, \tag{1.7c}$$

or equivalently, the system could be written with p and v stacked as in

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}' = \begin{pmatrix} x_3 \\ x_4 \\ 0 \\ -\mathfrak{g} \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 2x_1 \\ 2x_2 \end{pmatrix} y, \tag{1.8a}$$
$$0 = x_1^2 + x_2^2 - 1. \tag{1.8b}$$

Repeated differentiation (1.8b) with respect to t yields

$$g'(x,t) = x_1x_3 + x_2x_4 = 0, \tag{1.9a}$$
$$g''(x,t) = x_3^2 + x_4^2 - x_2\mathfrak{g} - 2(x_1^2 + x_2^2)y = 0, \tag{1.9b}$$
$$g'''(x,t) = -4x_1x_3y - 4x_2x_4y - \tfrac{3}{2}x_4\mathfrak{g} - (x_1^2 + x_2^2)y' = 0. \tag{1.9c}$$

Rearranging $g'''(x,t) = 0$ results in an ODE for y, and thus we have the following system of ODEs in x and y:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}' = \begin{pmatrix} x_3 \\ x_4 \\ 0 \\ -\mathfrak{g} \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 2x_1 \\ 2x_2 \end{pmatrix} y, \tag{1.10a}$$
$$y' = \frac{-4x_1x_3y - 4x_2x_4y - \frac{3}{2}\mathfrak{g}x_4}{x_1^2 + x_2^2}. \tag{1.10b}$$

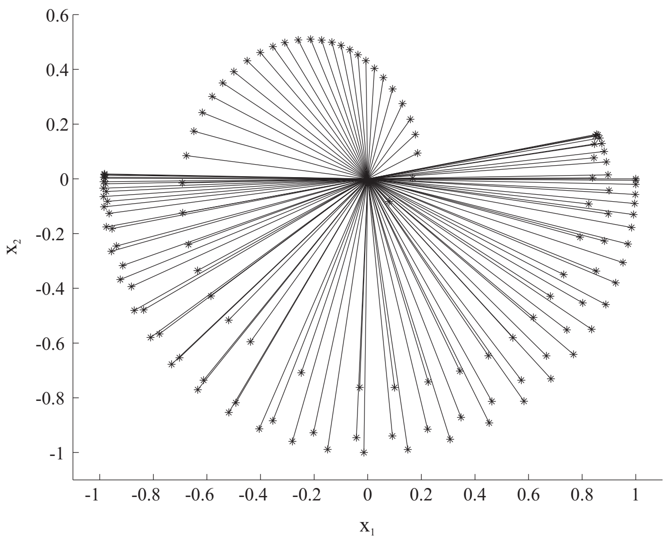


Figure 1.3: Path of a pendulum using the index reduction method from $t_0 = 0$ to $t_f = 5$.

Solving system (1.10) using MATLAB's `ode45` results in the output shown in Figure 1.3. Clearly this is an unreasonable simulation — the length of the arm of the pendulum has drastically shrunk over time! We say that the length of the pendulum has *drifted* in the course of solving the problem; that is, the value of the constraint equation is no longer zero.

Definition 5 (Drift) *The* drift *at some t and x is the value $\|g(x,t)\|$.*

Note that there is nothing mathematically incorrect about the model in either the original or index-reduced form; in fact, we expect some small amounts of drift due to numerical roundoff errors made at each time-step. However, by reducing the index, we have converted a stable problem into a numerically unstable one; that is, the roundoff errors tend to accumulate over time. Ideally, we desire a method where this does not happen.

1.5.2 Explanation for Drift Accumulation

Consider the index-reduced simple pendulum and note, that to derive $g'''(x,t) = 0$, we assumed that $g(x,t) = g'(x,t) = g''(x,t) = 0$. Assume our initial conditions for y are consistent and we can somehow solve system (1.10) in such a way as to insure that our solutions are consistent with $g'''(x,t) = 0$. Even this does not guarantee that $g(x,t) = 0$. Indeed, suppose at some point t^*, due to say, storing a real number as a floating point number, we obtain

$$\begin{aligned} g(x^*,t^*) &= \delta, \\ g'(x^*,t^*) &= \epsilon, \\ g''(x^*,t^*) &= \gamma, \\ g'''(x^*,t^*) &= 0, \end{aligned}$$

instead of $g(x^*,t^*) = 0$, $g'(x^*,t^*) = 0$, $g''(x^*,t^*) = 0$. Then for all $t \geq t^*$,

$$g(x,t) = \frac{\gamma}{2}(t-t^*)^2 + \epsilon(t-t^*) + \delta,$$

and thus the drift $\|g(x,t)\|$ will grow with time.

1.6 Regularization

Regularization seeks to convert DAEs into ODEs without using repeated differentiation of the constraint equation (1.4b) by replacing it with

$$\varepsilon T^{-1}(x,t)y' = g(x,t), \tag{1.11}$$

or equivalently

$$y' = \frac{1}{\varepsilon}T(x,t)g(x,t), \tag{1.12}$$

where ε and $T(x,t)$ are the *regularization parameter* and the *regularization matrix*, respectively. The regularization parameter ε is a (small) scalar such that $\varepsilon T^{-1}(x,t)y' \approx 0$ and thus (1.11) approximates the constraint equation. The regularization matrix stabilizes the problem and is $T(x,t) = (GB)^{-1}$. That being said, however, $(GB)^{-1}$ is computationally expensive and it is often advantageous to use $T(x,t) = (GB)^T$ or simply $T(x,t) = I$ (the latter is particularly useful for MBS where $B = G^T$). Having now regularized the problem, we now have the ODE

$$x' = f(x,t) - B(x,t)y, \tag{1.13a}$$

$$y' = \frac{1}{\varepsilon}T(x,t)g(x,t), \tag{1.13b}$$

which can then be solved using an ODE solver.

Regularization in the form shown here suffers from the fact that dividing by a small ε makes the problem *stiff*. That is, explicit methods such as `ode45` may require cripplingly small step-sizes in order to maintain numerical stability. This then generally forces one to use an implicit solver such as MATLAB's `ode15s`. This is not desirable because implicit methods are generally more computationally expensive than explicit methods for non-stiff problems.

The idea behind the Sequential Regularization Method is to solve a sequence of regularized problems at each step. As we shall see in Section 2.1, the main advantage of this over "pure" regularization is that ε does not need to be so small as to make the problem stiff. See Section 5.3 for a comparison of the SRM versus simply applying `ode45` to the regularized ODE problem.

1.7 The Baumgarte Stabilization Technique

Consider that the constraint equation of (1.4) defines a manifold on which solutions must lie. Unfortunately, written in the form of (1.4b), this constraint manifold is inherently numerically unstable. That is, if a numerical method makes a small error away from the manifold, then the error will not die out; in fact, as noted in Section 1.5, such errors will accumulate over time.

In the early 1970's, Baumgarte noted this in [5] and worked on replacing the constraint equation of an index-$(m+1)$ problem with

$$\sum_{j=0}^{m} \gamma_j \frac{\mathrm{d}^j}{\mathrm{d}t^j} g(x,t) = 0 \tag{1.14}$$

where $\gamma_m = 1$ and the roots of the polynomial $\sum_{j=0}^{m} \gamma_j x^j$ are negative. For index-2 problems, note that Baumgarte's stabilization can be written

$$\gamma_0 g(x,t) + \frac{\mathrm{d}}{\mathrm{d}t} g(x,t) = 0. \tag{1.15}$$

Baumgarte's intent was that this would make the constraint manifold attractive and thus improve its stability. Indeed, from the conditions on the γ_j, note that $\gamma_0 > 0$ and thus (1.15) is an exponentially decaying ODE so intuitively we expect that any small numerical errors should decay back to the manifold over time.

To use these ideas to solve DAEs, we must first expand the derivatives in (1.14). Consider, for simplicity, the index-2 case where (1.15) expands to

$$\gamma g + g_t + G x' = 0. \tag{1.16}$$

We can substitute (1.4a) for x'

$$\gamma g + g_t + G(f - By) = 0, \tag{1.17}$$

which reduces to an explicit formula for y

$$y = (GB)^{-1} \left[\gamma g + g_t + Gf \right]. \tag{1.18}$$

We then replace y in (1.4a) with (1.18)

$$x' = f - \gamma B(GB)^{-1} g - B(GB)^{-1}(Gf + g_t), \tag{1.19}$$

which can then be solved numerically.

Although Baumgarte stabilization is widely used by mechanical engineers (see, e.g., [2, 1]), it introduces stiffness into the system if the γ_j are not chosen correctly. Unfortunately, in practice, Baumgarte and others were unable to find a general method for determining the appropriate values for these parameters. As it turns out, they are dependent not only on the problem in question but also on the underlying step-size used by the numerical solver (see, [1]).

That being said, Baumgarte's technique plays an important part in the derivation and implementation of the Sequential Regularization Method (SRM). We discuss this relationship in more detail in the next chapter.

Chapter 2

The Sequential Regularization Method

As pointed out earlier, the basic idea behind the Sequential Regularization Method (SRM) is that that *instead of trying to solve one stiff problem, we solve a sequence of less-stiff problems at each time-step.* In this chapter, we cover the original formulation of SRM and then restrict ourselves to a particular case of the SRM that can be implemented as a purely explicit method. We describe the algorithm to implement the SRM for index-2 problems and make some comments about problems with isolated constraint singularities. The SRM for index-3 problems using a stabilized index reduction is then presented. Finally, we give an estimate of the error for the SRM and discuss how it influences our choice of the SRM parameters.

2.1 The SRM for Index-2 Problems

The original SRM was derived based on ideas from Baumgarte's work, combined with techniques from numerical methods for constrained optimization problems. It was originally formulated as follows:

$$
\begin{aligned}
\left(x^{(\sigma)}\right)' &= f(x^{(\sigma)}, t) \\
&\qquad - B(x^{(\sigma)}, t)\left(y^{(\sigma-1)} + \frac{1}{\varepsilon}T(x^{(\sigma)}, t)\left[\alpha_1 \frac{\mathrm{d}}{\mathrm{d}t} g(x^{(\sigma)}, t) + \alpha_2 g(x^{(\sigma)}, t)\right]\right), \qquad (2.1a) \\
y^{(\sigma)} &= y^{(\sigma-1)} + \frac{1}{\varepsilon}T(x^{(\sigma)}, t)\left[\alpha_1 \frac{\mathrm{d}}{\mathrm{d}t} g(x^{(\sigma)}, t) + \alpha_2 g(x^{(\sigma)}, t)\right], \qquad (2.1b)
\end{aligned}
$$

where $x^{(\sigma)}, \sigma = 1, 2, \ldots, m$ and $y^{(\sigma)}, \sigma = 0, 1, \ldots, m$ are *sequences of functions* which converge to x and y respectively. As seen in Section 1.6, the *regularization parameter* ε and *regularization matrix* T provide a regularization of the problem. As we will discuss in Section 2.6, we can use $\varepsilon = \mathcal{O}(h)$ and thus generally ε will thus not be small enough to make the problem stiff. For the regularization matrix, the eigenvalues of GBT help determine the stability of the solution manifold defined by the constraint equation. Thus the choice $T = (GB)^{-1}$ is generally the most robust (read, *safest*) because the eigenvalues of $GB(GB)^{-1}$ are all unity. However, as seen in Section 1.6, the choices $T = (GB)^T$ and $T = I$ may also be used (recall that the latter form is particular efficient for multibody systems where $B = G^T$).

The coefficients α_1, α_2 are known as *Baumgarte parameters* and are related to the γ_j discussed in Section 1.7. For DAEs without singularities, it is thought to be preferable to take $\alpha_1 \neq 0$ because no additional stiffness is introduced to the problem and also some restrictions on the choice of $y^{(0)}$ are avoided (see, [3, 4, 15]). However, even when using an explicit scheme to solve (2.1a), inverting a symmetric positive definite system will be necessary because of the $\frac{\mathrm{d}}{\mathrm{d}t}g$ term. On the other hand, the SRM with $\alpha_1 = 0$ *can* be used for problems with isolated singularities (see Section 1.4), where Baumgarte's method, and hence the SRM with $\alpha_1 \neq 0$, performs poorly. The $\alpha_1 = 0$ case also removes the requirement of a linear system inversion and thus can be implemented as a truly explicit method. The drawback is, however, that ε must be taken relatively large (as compared to the $\alpha_1 \neq 0$ case) to prevent the problem from becoming stiff.[1]

For the purposes of this thesis, we will consider only the case $\alpha_1 = 0$ with the justification that the analysis of convergence found in [3, 4, 15] holds in either case (albeit more complicated when $\alpha_1 = 0$). We thus consider the SRM method to be

$$\left(x^{(\sigma)}\right)' = f(x^{(\sigma)}, t) - B(x^{(\sigma)}, t)\left(y^{(\sigma-1)} + \frac{1}{\varepsilon}T(x^{(\sigma)}, t)g(x^{(\sigma)}, t)\right), \tag{2.2a}$$

$$y^{(\sigma)} = y^{(\sigma-1)} + \frac{1}{\varepsilon}T(x^{(\sigma)}, t)g(x^{(\sigma)}, t). \tag{2.2b}$$

Note that we can consider (2.2b) as $\varepsilon T(y^{(\sigma)} - y^{(\sigma-1)}) \approx 0 = g(x^{(\sigma)}, t)$ and in this sense, (2.2) can be related back to (1.4).

2.2 A Note on Notation

Given a time-step $[t_n, t_{n+1}]$ with step-size $h_n = t_{n+1} - t_n$, we adopt the common notation that $x_n^{(\sigma)}$ represents our numerical approximation to $x^{(\sigma)}(t_n)$ for $n \in \mathbb{N}$ and similarly extend this notation so that $x_{n+c}^{(\sigma)}$ numerically approximates $x^{(\sigma)}(t_n + ch_n)$, where $c \in [0, 1] \subset \mathbb{R}$.

2.3 The SRM Algorithm

Given an initial value $x_0^{(1)}$, a somewhat arbitrary[2] initial guess for $y^{(0)}$ (usually just $y^{(0)} \equiv 0$), and an s-stage Runge-Kutta scheme with abscissae $c_i \in [0, 1] \subset \mathbb{R}$, where $i = 1, 2, \ldots, s$, we perform the following algorithm (see also Figure 2.1):

Ⓐ Let $x_0^{(\sigma)} = x_0^{(1)}$ for $\sigma = 2, 3, \ldots, m$.

Ⓑ Use $x_0^{(\sigma)}$ to solve (2.2b) to obtain $y_0^{(\sigma)}$ for $\sigma = 1, 2, \ldots, m$.

[1]This is a drawback in the sense that more SRM iterations are required for larger values of ε.

[2]As noted above some restrictions are made upon this choice due to our choice of $\alpha_1 = 0$.

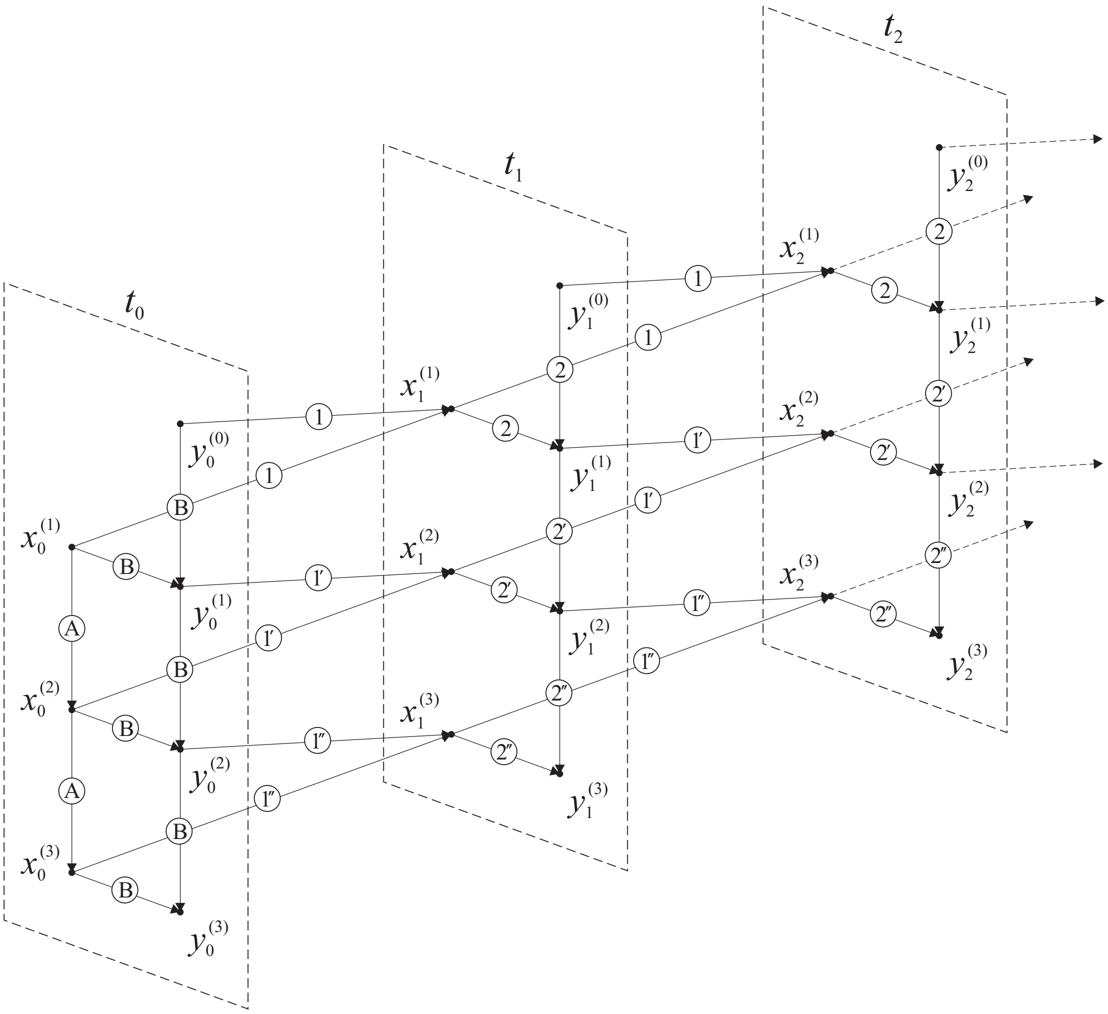


Figure 2.1: Graphical representation of the SRM algorithm.

- For each time-step $[t_n, t_{n+1}]$ with step-size $h_n = t_{n+1} - t_n$, repeat the following for $\sigma = 1, 2, \ldots, m$:

  ① Advance to $x_{n+1}^{(\sigma)}$ using the Runge-Kutta scheme on the ODE (2.2a). This will require $y_{n+c_i}^{(\sigma-1)}$.

  ② Evaluate (2.2b) using $x_{n+1}^{(\sigma)}$ and $y_{n+1}^{(\sigma-1)}$ to obtain $y_{n+1}^{(\sigma)}$.

  - If the Runge-Kutta scheme has a continuous extension mechanism[3] or some other high-order interpolant (see, e.g., [8]):
    - Then use it to obtain $x_{n+c_i}^{(\sigma)}$. Use these values along with $y_{n+c_i}^{(\sigma-1)}$ to solve (2.2b) for $y_{n+c_i}^{(\sigma)}$.
    - Else linearly interpolate between $y_n^{(\sigma)}$ and $y_{n+1}^{(\sigma)}$ to generate $y_{n+c_i}^{(\sigma)}$.

2.4 Problems with Isolated Singularities

Recall our discussion in Section 1.4 about multibody systems with isolated singularities. In general, the SRM can handle isolated singularities in either index-2 problems or the index-3 problems which we consider next in Section 2.5. However, some modifications must be made and we must be careful not to step directly onto a singular point.

Suppose GB is singular at some isolated t^*. Then because y, the algebraic variable, behaves like a Lagrange multiplier, we expect that it may be unbounded at t^*. However, the quantity By is bounded and smooth through the singularity, so update By instead of y by

$$B(x^{(\sigma)}, t)y^{(\sigma)} = B(x^{(\sigma-1)}, t)y^{(\sigma-1)} + \frac{1}{\varepsilon}B(x^{(\sigma)}, t)\left(G(x^{(\sigma)}, t)B(x^{(\sigma)}, t)\right)^{-1} g(x^{(\sigma)}, t). \tag{2.3}$$

[3]See Appendix A.4 for information on Runge-Kutta schemes with continuous extensions.

However, in general Range$B(x^{(\sigma)},t) \neq$ Range$B(x^{(\sigma-1)},t)$ and thus, as per [4], we define the *projection* $P(x) = B(GB)^{-1}G$ and modify (2.3) as follows

$$
\begin{aligned}
B(x^{(\sigma)},t)y^{(\sigma)} = P(x^{(\sigma)},t)&B(x^{(\sigma-1)},t)y^{(\sigma-1)} \\
&+ \frac{1}{\varepsilon}B(x^{(\sigma)},t)\left(G(x^{(\sigma)},t)B(x^{(\sigma)},t)\right)^{-1}g(x^{(\sigma)},t).
\end{aligned}
\tag{2.4}
$$

For the purposes of this thesis, we will not be implementing the SRM for index-2 problems with isolated constraint singularities. That being said, the ideas presented here will be useful in the next section when we consider the SRM for MBS with constraint singularities. The following practical concerns will be important for index-3 problems as well.

2.4.1 Practical Concerns

1. Our formulation still contains $(GB)^{-1}$ which was noted above to be singular at t^* and thus, although we can step near t^*, the end points of our steps must not land directly on t^* (or for that matter, on an interior stage point in the case of `erk4` or `dopr54`). If it looks like the end of the current step with size h will end up on a singular point t^* then we take a step of size $\frac{h}{2}$ instead followed by a step of size h. Note, that for `erk4` or `dopr54`, we may need to use $\frac{h}{4}$ or some other factor.

2. The ODE and DAE solvers LSODE (see, [14]) and DASSL (see, [6]), as implemented in Octave, allow the user to specify a list of times at which singular points occur and the solver will not step on those points. This saves the computational expense of checking the condition of (GB) at each step. However, *a priori* knowledge about the location of singular points is required by the user.

2.5 The SRM for Index-3 MBS

Recall from Section 1.3 that we are interested in solving particular index-3 problems known as multibody systems (MBS). However, the SRM is formulated to solve index-2 problems. One possible solution is to differentiate the constraint equation of (1.5) once to obtain an index-2 problem. However, as might be expected from our discussion on index reduction in Section 1.5, this technique introduces mild numerical instabilities into the system (see Section 5.6 for a demonstration the problems associated with using the index-2 SRM on index-reduced MBS). A better approach is to incorporate a *stabilized index reduction* (see, e.g., [2]) of the general MBS problem into the SRM.

We will assume that the problem has isolated singularities as discussed in Section 2.4 and thus our formulation will be in terms of $\hat{y} = By$ instead of y. To simplify our formulation, we define, as per [4], the *projection* $P(p^{(\sigma)}, t)$ as,

$$P(p^{(\sigma)}, t) \equiv M^{-1}G^T \left(GM^{-1}G^T\right)^{-1} G, \tag{2.5}$$

where we use $G = G(p^{(\sigma)}, t)$ and $M^{-1} = M^{-1}(p^{(\sigma)}, t)$ to simplify the notation. We note as Ascher and Lin did in [4] that, at least for the slider-crank problem, although $GM^{-1}G^T$ is singular at t^*, P and $M^{-1}G^T \left(GM^{-1}G^T\right)^{-1} g$ are smooth functions of t for the exact solution or functions p satisfying the constraints. We then replace (2.2) with the following formulation

$$\left(p^{(\sigma)}\right)' = v^{(\sigma)} - \frac{1}{\varepsilon} M^{-1}G^T \left(GM^{-1}G^T\right)^{-1} g(p^{(\sigma)}, t), \tag{2.6a}$$

$$\left(v^{(\sigma)}\right)' = M^{-1} f(p^{(\sigma)}, v^{(\sigma)}, t) - \left(P(p^{(\sigma)}, t)\hat{y}^{(\sigma-1)} + \frac{1}{\varepsilon} P(p^{(\sigma)}, t) v^{(\sigma)}\right), \tag{2.6b}$$

$$\hat{y}^{(\sigma)} = P(p^{(\sigma)}, t)\hat{y}^{(\sigma-1)} + \frac{1}{\varepsilon} P(p^{(\sigma)}, t) v^{(\sigma)}, \tag{2.6c}$$

where, again for simplicity of exposition, $G = G(p^{(\sigma)}, t)$ and $M^{-1} = M^{-1}(p^{(\sigma)}, t)$. By stacking p and v to obtain the differential variable $x = \binom{p}{v}$, the SRM algorithm for index-2 problems from above can be modified by solving (2.6a)–(2.6b) in place of (2.2a) and (2.6c) in place of (2.2b).

2.6 Error Estimates for the SRM

Ascher and Lin prove in [3, 4] that the order of the SRM is given by

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(\varepsilon^m) + \mathcal{O}(h^p), \tag{2.7}$$

where ε is the regularization parameter, h is the step-size, m is the number of SRM iterations, and p is the order of the ODE solver. Thus, we should choose $m = p$ and $\varepsilon = h$ to minimize (2.7). Using these values for the parameters, the error is thus

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(h^p). \tag{2.8}$$

We investigate these theoretical results in the Section 5.4.

2.6.1 Presence of an Initial Layer

Theorem 3.1 in [3] states that for linear DAEs in the form (1.4) with sufficiently smooth coefficients[4], the local error for the $\sigma \geq 1$ SRM iteration will satisfy

$$x^{(\sigma)}(t) - x(t) = \mathcal{O}(\varepsilon^\sigma) + \mathcal{O}\left(\varepsilon^{M+2}\rho_\sigma(\tfrac{t}{\varepsilon})e^{-\frac{t}{\varepsilon}}\right) \tag{2.9a}$$

$$By^{(\sigma)}(t) - By(t) = \mathcal{O}(\varepsilon^\sigma) + \mathcal{O}\left(\varepsilon^{M+1}\rho_\sigma(\tfrac{t}{\varepsilon})e^{-\frac{t}{\varepsilon}}\right) \tag{2.9b}$$

[4] Several other conditions must be satisfied, particularly for singular problems — consult [3] for additional information.

where $M \in \mathbb{Z}$ and $\rho_\sigma(\tau)$ is a generic polynomial with positive coefficients of degree $\sigma - M - 2$ such that

$$|\rho_\sigma(0)| = \left|(By^{(0)})^{M+1}(0) - (By)^{M+1}(0)\right|.$$

If, as in our case, we use an arbitrary $y^{(0)}$, then $M = -1$.

We note the presence of the $e^{-\frac{t}{\varepsilon}}$ in these local error bounds and thus, for some *initial layer* $0 \leq t \leq \mathcal{O}(\varepsilon)$, the local error on $x^{(\sigma)}(t)$ and $By^{(\sigma)}(t)$ may be dominated by $\rho_\sigma(\tau)$. To put it another way, for $t \geq c\varepsilon$ for some $c \in \mathbb{R}$, the term $e^{-\frac{t}{\varepsilon}}$ will be small enough that $x^{(\sigma)}(t) - x(t)$ and $By^{(\sigma)}(t) - By(t)$ will be bounded by $\mathcal{O}(\varepsilon^\sigma)$.

In the next chapter, we examine an improvement of the SRM known as the *Predicted Sequential Regularization Method (PSRM)* developed by Lin and Spiteri in [16] whereby we attempt to make a better initial guess at $y^{(0)}$ (as opposed to the SRM's arbitrary $y^{(0)}$) followed by just a single iteration of the SRM. We examine experimental results for both methods in Chapters 5.

Chapter 3

The Predicted Sequential Regularization Method

The SRM is not optimal in the sense of the number of iterations required for convergence. Recall from Section 2.6 that we generally choose m, the number of iterations, to be the order of the differential equation solver. This is in some sense inefficient because more iterations are required for higher-order methods.

In [16], Lin and Spiteri describe a *predicted sequential regularization method (PSRM)* whereby an accurate initial guess for the algebraic variable is first obtained. This allows a high-order approximation of the differential variable with only a single iteration of the SRM. Lin and Spiteri present various *prediction schemes* for calculating such a $y^{(0)}$. We will examine two of these schemes in this chapter.

3.1 PSRM with Simple Prediction

For this prediction scheme, we use the first-order extrapolation $y^{(0)}(t) = y^{(1)}(t_n)$ for $t \in [t_n, t_{n+1}]$ as our initial guess for $y^{(0)}$. We then proceed with a single step of the

SRM, which we repeat from Chapter 2 for convenience:

$$\left(x^{(\sigma)}\right)' = f(x^{(\sigma)}, t) - B(x^{(\sigma)}, t)\left(y^{(\sigma-1)} + \frac{1}{\varepsilon}T(x^{(\sigma)}, t)g(x^{(\sigma)}, t)\right), \tag{3.1a}$$

$$y^{(\sigma)} = y^{(\sigma-1)} + \frac{1}{\varepsilon}T(x^{(\sigma)}, t)g(x^{(\sigma)}, t). \tag{3.1b}$$

This method generalizes to an kth-order extrapolation based on our estimates of $y^{(1)}(t_n), y^{(1)}(t_{n-1}), \ldots, y^{(1)}(t_{n-k+1})$. However, Lin and Spiteri note that such extrapolations are generally unstable for the Baumgarte parameter $\alpha_1 = 0$.

As shall be discussed in Section 3.5, restricting ourselves to this simple first-order extrapolation predictor means that our PSRM will be limited to $\mathcal{O}(h^2)$, regardless of the Runge-Kutta method used.

3.2 PSRM Algorithm with Simple Prediction

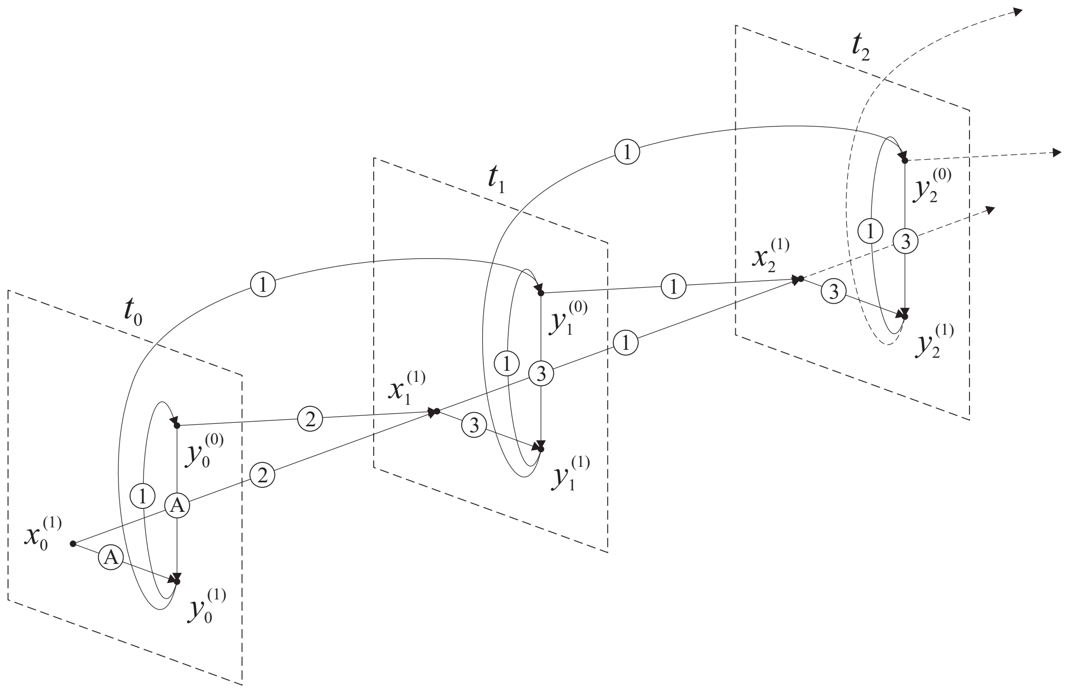


Figure 3.1: Graphical representation of the PSRM algorithm using Simple Prediction.

Given initial values for $x_0^{(1)}$ and $y_0^{(0)}$ along with an s-stage Runge-Kutta scheme with abscissae $c_i \in [0,1] \subset \mathbb{R}$, where $i = 1, 2, \ldots, s$, we perform the following algorithm (see also Figure 3.1):

- Ⓐ Use (3.1b) to obtain $y_0^{(1)}$ from $y_0^{(0)}$ and $x_0^{(1)}$.
- ▫ For each time-step $[t_n, t_{n+1}]$ with step-size $h_n = t_{n+1} - t_n$, perform the following:
  - ① Let $y_{n+c_i}^{(0)} = y_n^{(1)}$. Note that this means $y_n^{(0)} = y_{n+1}^{(0)} = y_n^{(1)}$.
  - ② Advance to $x_{n+1}^{(1)}$ using the Runge-Kutta scheme on the ODE (3.1a). This will in general require $y_{n+c_i}^{(0)}$
  - ③ Evaluate (3.1b) using $x_{n+1}^{(1)}$ and $y_{n+1}^{(0)}$ to obtain $y_{n+1}^{(0)}$.

3.3 PSRM with Exact y Predictor

We seek a predictor for $y^{(0)}$ which produces a sufficiently accurate guess that we may use higher-order Runge-Kutta schemes such as Classic Fourth-Order Runge-Kutta (`erk4`) or Dormand-Prince 5(4) (`dopr54`) to their full potential. We begin by recalling that the constraint equation states that

$$0 = g(x, t).$$

This can be differentiated to obtain

$$0 = G(x, t)x' + g_t(x, t).$$

Replacing x' from the differential equation results in

$$0 = G(x, t)\left(f(x, t) - B(x, t)y\right) + g_t(x, t),$$

or

$$y = (G(x,t)B(x,t))^{-1}\left(g_t(x,t) + G(x,t)f(x,t)\right). \tag{3.2}$$

Armed with (3.2), we choose a *predictor* iteration as

$$\left(x^{(0)}\right)' = f - B\left[(GB)^{-1}(Gf + g_t)\right] \tag{3.3a}$$

$$y^{(0)} = (GB)^{-1}(Gf + g_t) \tag{3.3b}$$

where f, B, g, g_t, and G are functions of $(x^{(0)}, t)$. Note that the somewhat arbitrary initial guess of the SRM is no longer present. However, as might be expected, we have decreased the numerical stability of our solution by differentiating the constraint equation. To get around this, we use a single SRM iteration as the *corrector* iteration of

$$\left(x^{(1)}\right)' = f - B\left[y^{(0)} - \frac{1}{\varepsilon}Tg\right] \tag{3.3c}$$

$$y^{(1)} = y^{(0)} - \frac{1}{\varepsilon}Tg \tag{3.3d}$$

where f, B, g, g_t, and G are now functions of $(x^{(1)}, t)$.

Note that computationally speaking, this PSRM is approximately equivalent to two iterations ($m = 2$) of the SRM. This PSRM thus does not present an advantage over SRM with a lower-order Runge-Kutta method such as Trapezoidal Rule or Heun's Method. However, with a fifth-order method such as Dormand-Prince 5(4), it presents a considerable advantage from the point of view of computational cost.

3.4 PSRM Algorithm with Exact y Prediction

Given an initial value $x_0^{(0)}$ and an s-stage Runge-Kutta scheme having abscissae $c_i \in [0,1] \subset \mathbb{R}$, where $i = 1, 2, \ldots, s$, we perform the following algorithm (see also Figure 3.2):

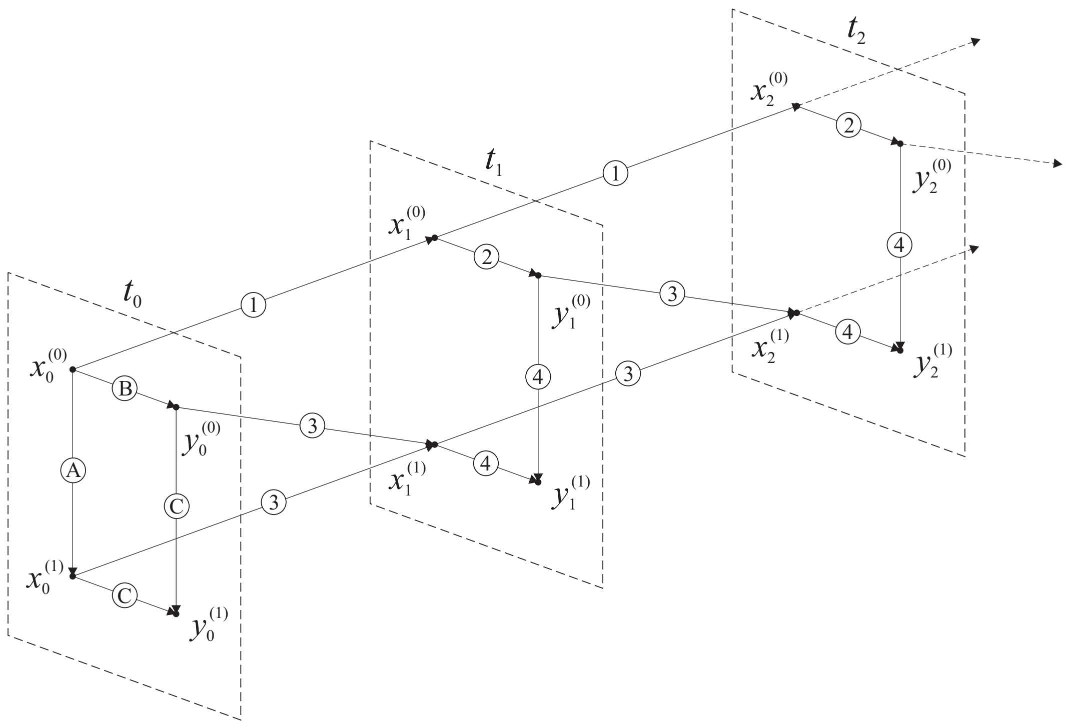


Figure 3.2: Graphical representation of the exact y PSRM algorithm.

Ⓐ Set $x_0^{(1)} = x_0^{(0)}$.

Ⓑ Solve (3.3b) to obtain $y_0^{(0)}$.

Ⓒ Using $x_0^{(1)}$ and $y_0^{(0)}$, solve (3.3d) to obtain $y_0^{(1)}$.

- For each time-step $[t_n, t_{n+1}]$ with step-size $h_n = t_{n+1} - t_n$, execute the following:
  - ① Advance to $x_{n+1}^{(0)}$ using the Runge-Kutta scheme on the ODE (3.3a).
  - ② Evaluate (3.3b) using $x_{n+1}^{(0)}$ to obtain $y_{n+1}^{(0)}$.
  - If the Runge-Kutta scheme has a continuous extension mechanism:
    - Then use it to obtain $x_{n+c_i}^{(0)}$. Use these values to solve (3.3b) for $y_{n+c_i}^{(0)}$.
    - Else linearly interpolate between $y_n^{(0)}$ and $y_{n+1}^{(0)}$ to generate $y_{n+c_i}^{(0)}$.
  - ③ Advance to $x_{n+1}^{(1)}$ using the Runge-Kutta scheme on the ODE (3.3c). This will require $y_{n+c_i}^{(0)}$.
  - ④ Evaluate (3.3d) using $x_{n+1}^{(1)}$ and $y_{n+1}^{(0)}$ to obtain $y_{n+1}^{(1)}$.

3.5 Error Estimates for the PSRM

Global error estimations for the PSRM are considered by Lin and Spiteri in [16]. The results are more complicated than in the case of the SRM because an error term due to the prediction scheme of $y^{(0)}$ must be accounted for. Using the simple prediction scheme described above, the global error due to predicting $y^{(0)}$ is $\mathcal{O}(h^2)$ and hence we do not expect global errors any better than $\mathcal{O}(h^2)$ from any of our methods. We will investigate this idea experimentally in Chapter 5.4. As per [16], we consider the global error estimate for trapezoidal rule (trap), classic fourth-order Runge-Kutta (erk4), and Dormand-Prince 5(4).

Error Estimate for Trapezoidal Rule

Trapezoidal rule is a second-order explicit Runge-Kutta scheme so

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(\varepsilon h^2) + \mathcal{O}(h^2), \tag{3.4}$$

and thus we can choose $\varepsilon = \mathcal{O}(h)$ and we expect $\mathcal{O}(h^2)$ error. That is

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(h^2), \tag{3.5}$$

Error Estimate for ERK4

Theorem 3.1 in [16] tells us that

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(\varepsilon h^2) + \mathcal{O}\left(\frac{h^4}{\varepsilon^2}\right), \tag{3.6}$$

and thus we let $\varepsilon = h^q$ and choose q such that

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(h^{q+2}) + \mathcal{O}(h^{4-2q}), \tag{3.7}$$

is as small as possible. That is we choose q to maximize $\min(q+2, 4-2q)$. Indeed, $q = \frac{2}{3}$ bounds the global error as

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}\left(h^{\frac{8}{3}}\right). \tag{3.8}$$

Error Estimate for Dormand-Prince 5(4)

Again Theorem 3.1 in [16] tells us that

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(\varepsilon h^4) + \mathcal{O}\left(\frac{h^5}{\varepsilon^2}\right). \tag{3.9}$$

As above, we let $\varepsilon = h^q$ and choose q to maximize $\min(4+q, 5-2q)$. This works out to be $q = \frac{1}{3}$ and thus the expected order of convergence is

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}\left(h^{\frac{13}{3}}\right). \tag{3.10}$$

Chapter 4

Implementation

The SRM and PSRM are implemented in the `MATLAB` language for execution with *The Mathworks* proprietary scientific computing package, MATLAB. The code should be executable with minimal modifications using OCTAVE[1] which is Free Software available under the GNU Public License.

4.1 Files

The SRM/PSRM project consists of the following files:

`srm.m` is the core SRM file. The client calls this code directly to solve a DAE problem using the SRM.

`psrm.m` is the core PSRM file. The client calls this code directly to solve a DAE problem using the PSRM.

`srmiter_index2_x.m` is the differential equation for $x^{(\sigma)}$ for index-2 problems without singularities.

[1]Modifications to the code to allow execution on this platform are in progress.

`srmiter_index2_y.m` is the updater function for $y^{(\sigma)}$ for index-2 problems without singularities.

`srmiter_index3sir_x.m` is the differential equation for $x^{(\sigma)} = \binom{p^{(\sigma)}}{v^{(\sigma)}}$ for index-3 problems with and without isolated constraint singularities using stabilized index-reduction.

`srmiter_index3sir_y.m` is the updater function for $\hat{y}^{(\sigma)} = By^{(\sigma)}$ for index-3 problems with and without isolated constraint singularities using stabilized index-reduction.

`whichepsilon.m` is called with parameters `whicheps` and the current step-size h. It calculates the value of ε and returns it. Currently, strictly positive values of `whicheps` are treated as the desired value for ε and zero or negative values are "flags" specifying functions of h such as $\sqrt{h}$ or $h^{\frac{2}{3}}$.

`srm_config.m` is called with a string `method` which is the name of an ODE solver. It answers information about that solver such as the name of an implementing function, the number of stages, and whether it has a continuous extension.

`odesolv_XXX.m` are functions that implement the various Runge-Kutta methods to solve ODEs. Each one takes a single step by calling `srmiter_index2_x.m` or `srmiter_index3sir_x.m` as required.

`odesolv_XXX_cext.m` returns the value of x at any interior point for those methods with continuous extensions.

`template/` is a directory containing a template for setting up a DAE problem to be solved with the SRM/PSRM.

- `f.m` corresponds to $f(x,t)$ in (1.4). It should answer a vertical vector of length n.
- `B.m` corresponds to $B(x,t)$ in (1.4). It should answer a matrix of size $n \times n_y$.

`fe`	Forward Euler
`trap`	Trapezoidal rule
`heun`	Heun's Method
`tvd32`	Total-Variation-Diminishing 3(2)
`erk4`	Classic fourth-order Runge-Kutta
`dopr54`	Dormand-Prince 5(4)

Table 4.1: ODE solvers provided by the SRM implementation.

`g.m` corresponds to $g(x,t)$ in (1.4). It should answer the value of the constraint equation: a vertical vector of length n_y.

`g_t.m` should implement $g_t(x,t) = \frac{\partial g(x,t)}{\partial t}$. It should answer a vertical vector of length n_y.

`Gjac.m` should implement the Jacobian of $g(x,t)$, $G(x,t) = g_x(x,t)$. It should answer a matrix of size $n_y \times n$.

4.2 ODE Solvers

The code allows the user to choose between the explicit Runge-Kutta schemes shown in Table 4.2. A generic mechanism is also in place for methods with continuous extensions such as Dormand-Prince 5(4) (see Appendix A for more information on Dormand-Prince 5(4)).

Upon execution of the `srm` function, the `srm_config` function is called with the name of the ODE solver from Table 4.2 as a argument. This function then returns information such as the number of stages of the method, the name of a function which implements it, etc. Provided that they fit the existing framework (i.e., adhere to a certain interface), new ODE solvers can be added by simply editing this `srm_config` function; i.e., without requiring any modifications to `srm.m`.

4.3 Additional Information

1. Typically `srm.m`, `srm_config.m`, and the other files that form the core SRM code will not be in the current directory with `f.m`, `G.m`, etc. Thus it is necessary to add the path to the core SRM code to MATLAB's search path, for example by the following `MATLAB` command:

   ```
   >> addpath /export/home/user/matlab/srm
   ```

2. MATLAB does not require that one explicitly define whether a vector is horizontal or vertical. However, when writing `f.m`, `B.m`, `g.m`, `g_t.m`, and `Gjac.m` it is important to make sure that the orientation of the vectors is as described above.

Chapter 5

Results

After introducing some additional problems, we begin this section with a comparison of the new `MATLAB` SRM implementation to Ascher and Lin's existing `FORTRAN` code. This is followed by a demonstration of the benefits of the SRM over regularization using an explicit solver. Using a simple test problem, several sections are then dedicated to the verification of the theoretical orders of convergence for both the SRM and the PSRM with Simple Prediction. This is proceeded by another demonstration; this time of the stable versus unstable index reduction on the index-3 slider-crank problem. Finally, the results of an implementation of automatic step-size control (as discussed in Appendix B) for the SRM are presented.

5.1 Test Problems

We test the SRM and PSRM on the following problems with exact solutions:

Problem 1 (Trivial problem)

$$x' = -y, \tag{5.1}$$

$$0 = g(t, x) = x - t^p, \tag{5.2}$$

where p is the order of the ODE solver used (e.g., $p = 2$ for ***trap****). Integrate from $t = 0$ to $t = 1$. Exact solution: $x = t^p$, $y = -pt^{p-1}$.*

Problem 2

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' = \begin{pmatrix} 1 - e^{-t} \\ \cos t + e^t \sin t \end{pmatrix} - \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} y, \tag{5.3a}$$

$$0 = \frac{1}{2}\left(x_1^2 + x_2^2 - e^{-2t} - \sin^2 t\right). \tag{5.3b}$$

Integrate from $t = 0$ to $t = 10$. Exact solution: $x_1 = e^{-t}$, $x_2 = \sin t$, $y = e^t$.

5.2 Comparison to Ascher and Lin's Code

The Sequential Regularization Method was originally coded in `FORTRAN` by Ascher and Lin and the results presented in [3] and [4]. We compare the solutions of the `FORTRAN` and `MATLAB` implementations of the SRM by considering Problem 2 (which appeared in [4] as Example 6.1). Table 5.1 confirms that both codes produce equally accurate output.

5.3 Regularization versus the SRM

As discussed in Section 1.6, we can solve a DAE such as the simple pendulum problem by regularization. Figure 5.1 shows the length of the pendulum for various values of the regularization parameter ε and Figure 5.2 shows the corresponding paths taken by the pendulum. For very small ε (10^{-6} and 10^{-8}) we observe typical behavior of a stiff equation: small step-sizes and a lack of numerical stability. We may be tempted

Table 5.1: Solution at $t_f = 1$ of Problem 2 using Ascher and Lin's SRM `FORTRAN` code and the new SRM `MATLAB` code for various Runge-Kutta schemes. These results were generated with $T = I$, $\varepsilon = 0.005$, $h = 0.001$, and m as shown in the table.

	`FORTRAN` code		
RK scheme	x_1	x_2	y
`fe` ($m = 1$)	0.3778594135314	0.8528025928693	2.664129728879
`trap` ($m = 2$)	0.3680954351605	0.8416941814079	2.717145854492
`erk4` ($m = 4$)	0.3678795515753	0.8414710846979	2.718280885578
`dopr54` ($m = 5$)	0.3678794427691	0.8414709879687	2.718281809182
	`MATLAB` code		
RK scheme	x_1	x_2	y
`fe` ($m = 1$)	0.37785941353142	0.85280259286934	2.66412972887902
`trap` ($m = 2$)	0.36809543516049	0.84169418140791	2.71714585449176
`erk4` ($m = 4$)	0.36787955157531	0.84147108469791	2.71828088557805
`dopr54` ($m = 5$)	0.36787944276909	0.84147098796866	2.71828180918235

to claim that the drift for larger ε (10^{-4} and 10^{-2}) looks better. However, recall if ε is not small, we are solving a different problem (thus Figure 5.2a and Figure 5.2b are in some sense not solutions of the pendulum problem).

Regularization, as used in the SRM, does not require that ε be small (recall from Section 2.6, that $\varepsilon = \mathcal{O}(h)$ is optimal in the sense of order). Figure 5.3 shows that the SRM using `trap` obtains a much more stable solution than pure regularization.

5.4 SRM Convergence Tests on Problem 1

In this section, we consider Problem 1, which is set up so that the Runge-Kutta should be able to obtain an exact solution to the ODE component of the DAE. Recall from Section 2.6, that we expect the SRM to convergence such that

$$\|x^{(m)}(t) - x(t)\| = \mathcal{O}(\varepsilon^m) + \mathcal{O}(h^p). \tag{5.4}$$

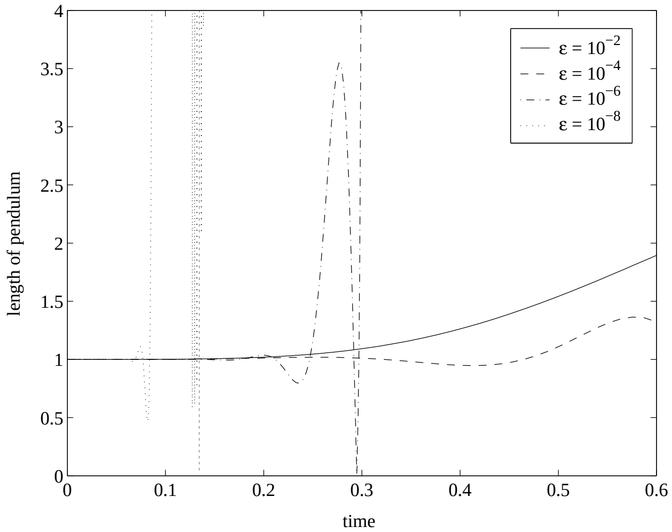
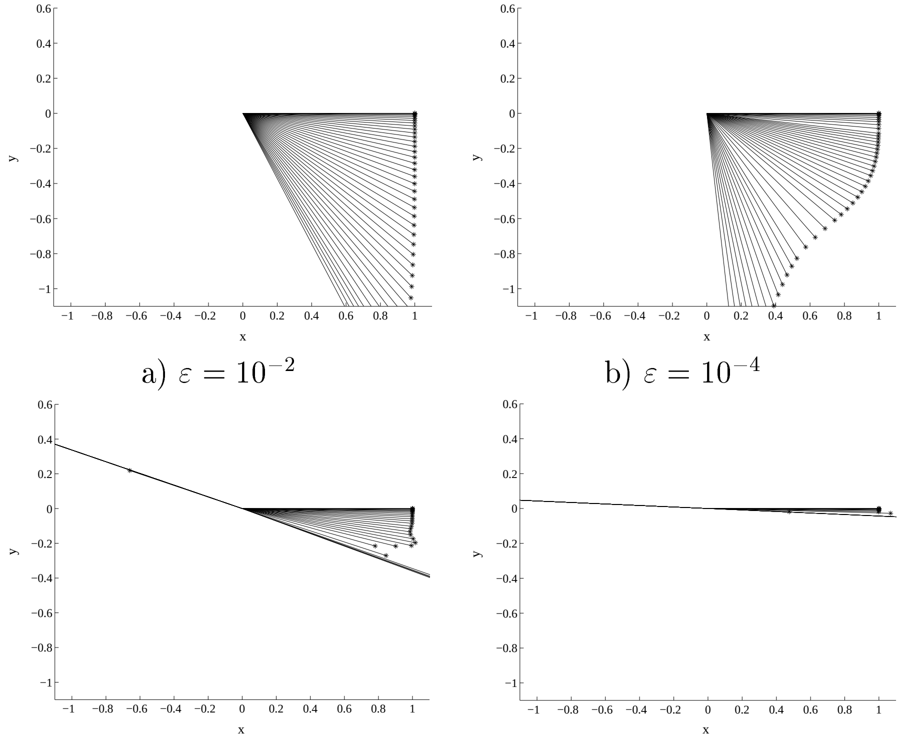


Figure 5.1: Drift while solving the regularized pendulum problem with `ode45` for various values of the regularization parameter ε.



a) $\varepsilon = 10^{-2}$ b) $\varepsilon = 10^{-4}$

c) $\varepsilon = 10^{-6}$ d) $\varepsilon = 10^{-8}$

Figure 5.2: The path of the regularized pendulum problem solved using `ode45` for various values of ε.

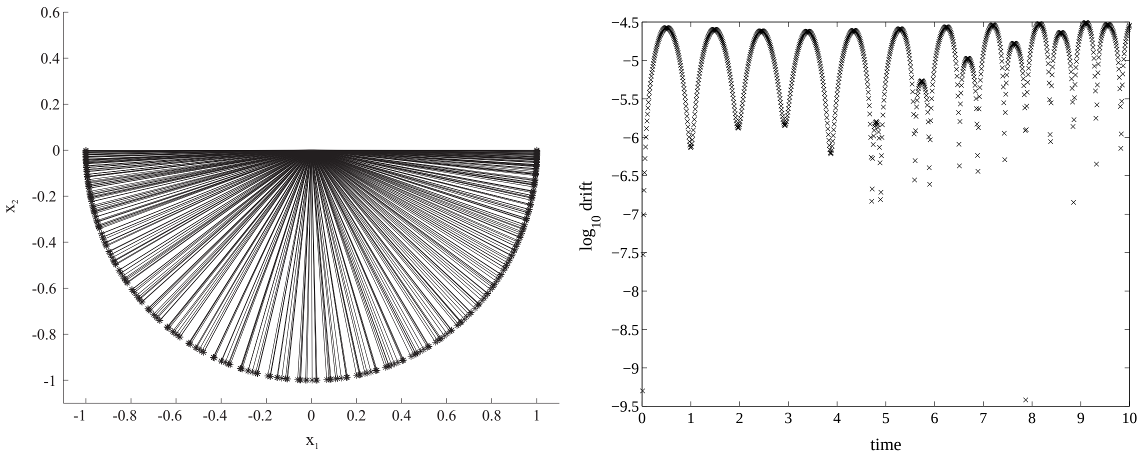


Figure 5.3: Path (left) and drift (right) of the SRM solution to the index-3 pendulum problem for $t \in [0, 20]$ using `trap` with $h = 0.001$, $\varepsilon = h$, $T = (GB)^{-1}$, and $m = 2$.

We will attempt to verify this experimentally on Problem 1, which is trivial in the sense that if we choose p to be the order of a particular Runge-Kutta method, then that Runge-Kutta method can integrate the ODE for $x^{(\sigma)}(t)$ exactly to find the exact value for $x^{(\sigma)}(t_f)$. Note however, that this does not imply that it can integrate to find $x(t_f)$ exactly. The convergence of $x^{(\sigma)}$ to x is also dependent on the value of ε. We are interested in confirming that this dependence to ε is order-m as predicted by (5.4).

For a given m and h, we calculating the error made in approximating $x(t_f)$

$$\|x_N^{(m)} - x(t_f)\|, \tag{5.5}$$

where $N = \frac{t_f}{h}$. We repeat this for various values of ε and plot the results on a plot of $\log_{10}(\text{error})$ versus $\log_{10}\varepsilon$. We expect a linear dependence with a slope of m. We perform the same experiment with $\varepsilon = h$ with the anticipation of similar results. Also, we try fixing ε and varying h and thus plot $\log_{10}(\text{error})$ versus $\log_{10} h$. For any particular ε, we expect the Runge-Kutta method to be able to solve for $x^{(\sigma)}(t_f)$ exactly as stated above, and thus we do not expect convergence in this case.

Because we anticapate that we may encounter the exact answer, we define $\log_{10}(0) := -17$. On the particular machine that ran these tests (a Sun Ultra 10 workstation), $\epsilon_{\text{MACH}} \approx 2 \times 10^{-16}$ and thus 10^{-17} is below the level of round-off errors.

5.4.1 Trapezoidal Rule

Trapezoidal rule is an $\mathcal{O}(h^2)$ Runge-Kutta method (see Appendix C for the Butcher Tableau). Table 5.2 and Figure 5.4 show the results of setting $\varepsilon = h$ and fixing h respectively. These results would seem to confirm our expected results. Figure 5.5 shows the anticipated lack of convergence when fixing ε.

Table 5.2: Convergence of the SRM using `trap` on Problem 1 with both $\varepsilon = h$ and fixed $h = 0.005$.

m	order of convergence	
	$\varepsilon = h$	fixed h
1	0.9722	0.9822
2	1.9995	2.0000
3	exact	exact

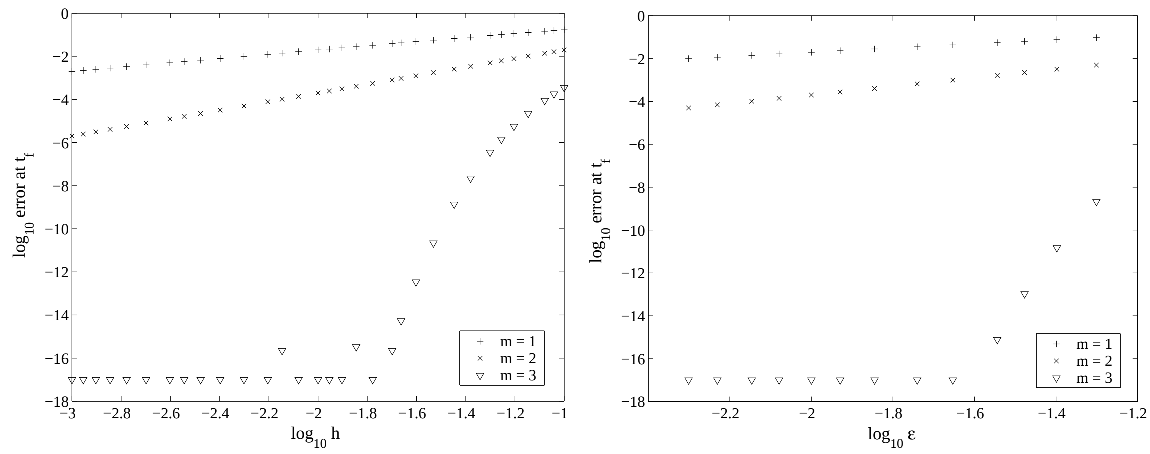


Figure 5.4: Convergence of the SRM using `trap` on Problem 1 with $\varepsilon = h$ (left) and fixed $h = 0.005$ (right).

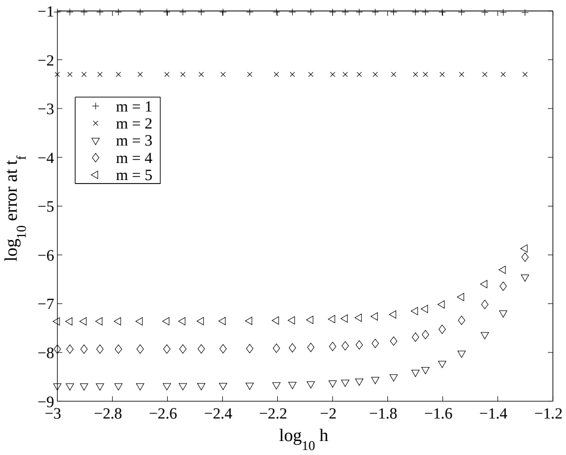


Figure 5.5: Convergence of `trap` on Problem 1 with fixed $\varepsilon = 0.05$.

5.4.2 Total Variation Diminishing 3(2)

Total Variation Diminishing 3(2) (`tvd32`) is a third-order Runge-Kutta method with Trapezoidal Rule embedded. Table 5.3 and Figures 5.6, 5.7 show the convergence of this method.

For purposes of step-size control, we iterate the embedded method $m - 1$ times instead of m. Thus for the lower-order embedding method, we expect $\mathcal{O}(\varepsilon^{m-1}) + \mathcal{O}(h^p)$ convergence.

The plots confirm these results. Note that we never attain the exact answer using the embedded Trapezoidal Rule but recall that we are solving the problem with $p = 3$ and thus the convergence of the SRM is restricted by the $\mathcal{O}(h^2)$ convergence of Trapezoidal Rule regardless of m (as seen in Figure 5.6b).

We note an inexplicable upward turn in Figure 5.7b where the error appears to increase with decreasing ε (for $m \geq 4$). Similar observations are noted in Figure 5.10, and again no explanation has been obtained within the current framework of the theory.

Table 5.3: Convergence of the SRM using `tvd32` on Problem 1 with both $\varepsilon = h$ and fixed $h = 0.005$.

higher-order Method			lower-order Method		
m	order of convergence		$m-1$	order of convergence	
	$\varepsilon = h$	fixed h		$\varepsilon = h$	fixed h
2	1.9646	1.9412	1	0.9579	0.9472
3	2.9999	2.9996	2	1.9505	2.0216
4	exact	exact	3	1.9264	1.8422
5	exact	exact	4	1.9928	-0.6472

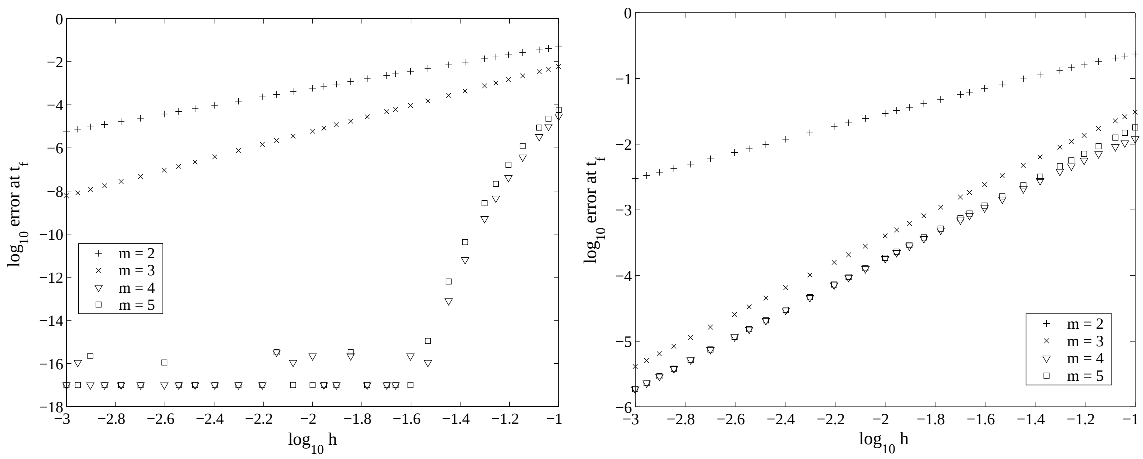


Figure 5.6: Convergence of the SRM using `tvd32` on Problem 1 using $\varepsilon = h$. The plot on the left is of the higher-order method and the right is of the lower-order (embedded) method.

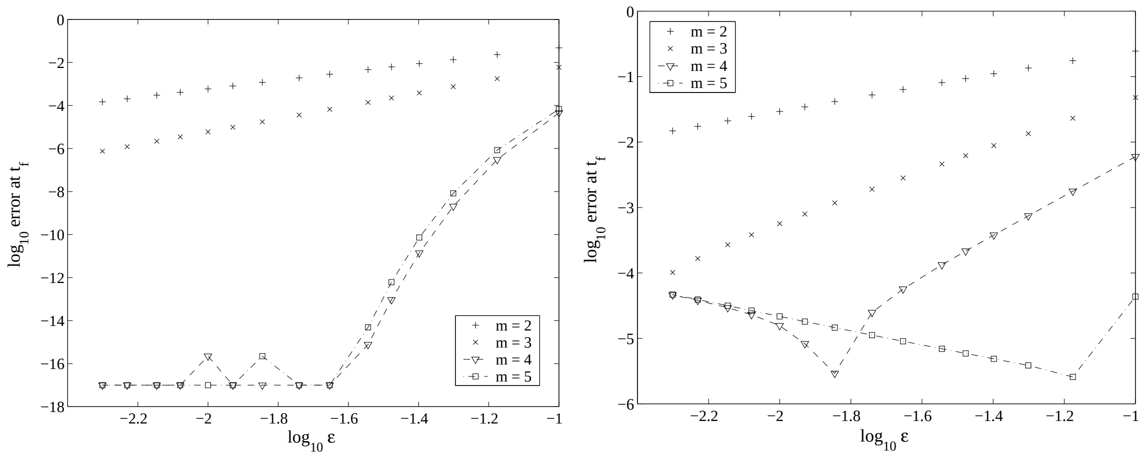


Figure 5.7: Convergence of the SRM using `tvd32` of Problem 1 with fixed $h = 0.005$. The plot on the left is of the higher-order method and the right is of the lower-order (embedded) method.

5.4.3 Classic Fourth-Order Explicit Runge-Kutta

The convergence of the SRM using Classic Fourth-Order Explicit Runge-Kutta (`erk4`) is shown in Table 5.4 and Figure 5.8. The results are generally as expected; however we note that fifth-order convergence is obtained when $m = 5$ and $\varepsilon = h$, whereas we expected the SRM to produce the exact answer in this case. Because `erk4` should have solved for $x^{(5)}(t_f)$ correctly, $x^{(5)}$ must still be converging to x and thus the convergence with respect to ε is not exactly $\mathcal{O}(\varepsilon^m)$. This might be due to a lower-order error term introduced by the linear interpolation that estimates $y^{(\sigma)}$ at the mid-points between steps (i.e., $y^{(\sigma)}_{n+\frac{1}{2}}$ in the notation introduced in Section 2.2).

5.4.4 Dormand-Prince 5(4)

The Dormand-Prince 5(4) Runge-Kutta method (`dopr54`) consists of a fifth-order method with an embedded fourth-order method for step-size control. Table 5.5 and Figures 5.9 and 5.10 show the convergence of the SRM of both methods for the $\varepsilon = h$ and fixed h cases. The convergence when ε is fixed is shown in Figure 5.11, and shows, as we anticipated, a lack of convergence.

Table 5.4: Convergence of the SRM using `erk4` on Problem 1.

m	order of convergence	
	$\varepsilon = h$	fixed h
1	0.9344	0.9235
2	1.8897	1.8895
3	2.8783	2.9078
4	3.9085	3.9985
5	4.9974	exact
6	exact	exact

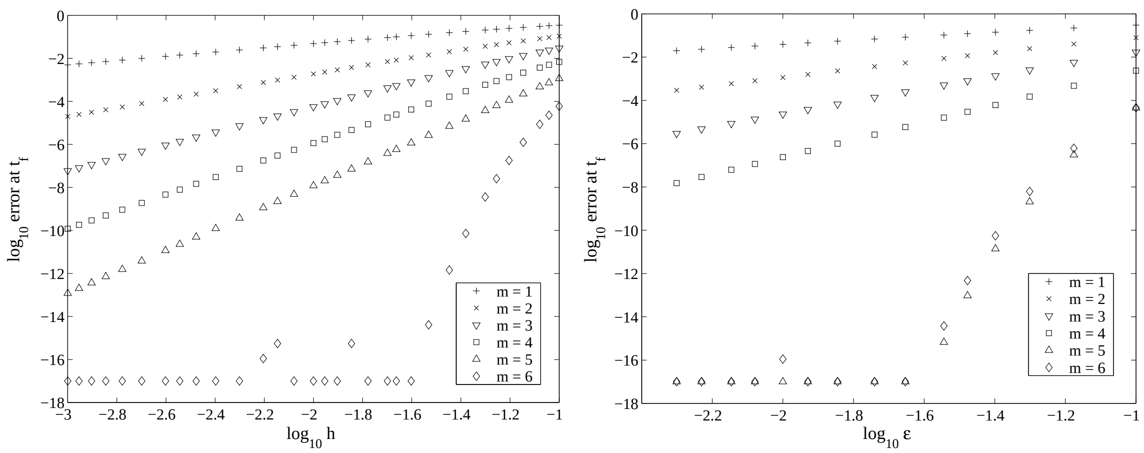


Figure 5.8: Convergence of the SRM using `erk4` on Problem 1 with $\varepsilon = h$ (left) and fixed $h = 0.005$ (right).

We do not get the expected convergence for $m \geq 3$ in Figure 5.9. However, it appears that when h and ε are greater than some value where a "kink" shows (approx. 0.025 for $m = 4$ and 0.05 for $m = 5$), then the anticipated convergence occurs. In Figure 5.10, we note that we get the correct convergence provided ε is sufficiently larger than h.

With these observations in mind, we consider Table 5.6 and Figure 5.12 where $\varepsilon = 5h$. This choice results in improved convergence! Because we are talking about *order* of convergence, the choice $\varepsilon = h$ still results in a theoretical convergence of

$$\mathcal{O}\left((5h)^m\right) + \mathcal{O}(h^p) = \mathcal{O}(h^p), \quad \text{when } m = p. \tag{5.6}$$

which is indeed confirmed by Table 5.6 and Figure 5.12.

Note that in Table 5.6 and Figure 5.12, we are getting better-than-expected convergence similar to that seen for `erk4` (see Section 5.4.3). Again, we suspect that this is due to an error term introduced by the continuous extension. In this case, the particular continuous extension is known to be non-differentiable (see, [12]), and thus we suspect that a large error term may be introduced due to discontinuities in the derivative.

Table 5.5: Convergence of the SRM using `dopr54` for Problem 1.

higher-order method			lower-order method		
m	order of convergence		$m-1$	order of convergence	
	$\varepsilon = h$	fixed h		$\varepsilon = h$	fixed h
2	1.8963	1.8418	1	0.9354	0.9019
3	2.9823	2.8414	2	1.8964	1.8416
4	2.5541	3.6501	3	2.9080	2.8299
5	1.8594	2.8742	4	2.9136	4.0920
6	1.9414	-0.5188	5	1.9723	3.3904
7	1.9561	-0.4729	6	0.9325	-0.1484

Table 5.6: Convergence of the SRM using `dopr54` for Problem 1 using $\varepsilon = 5h$.

m	higher-order	lower-order
2	1.7219	0.8371
3	2.6682	1.7219
4	3.7013	2.6682
5	4.9198	3.7010
6	4.8977	4.8706
7	4.5212	5.4115
8	4.1336	5.1793

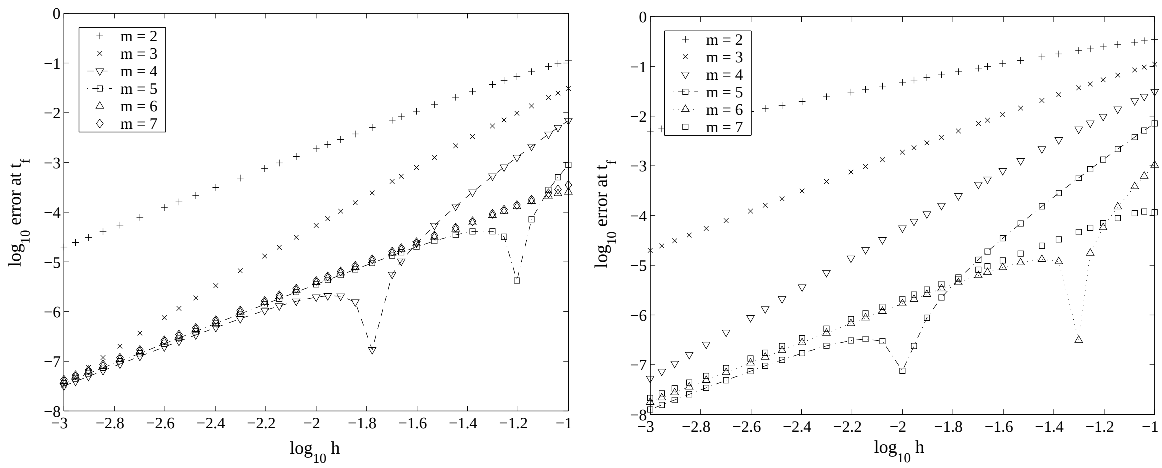


Figure 5.9: Convergence of the SRM using `dopr54` for Problem 1 with $\varepsilon = h$. The plot on the left is of the higher-order method and the right is of the lower-order (embedded) method.

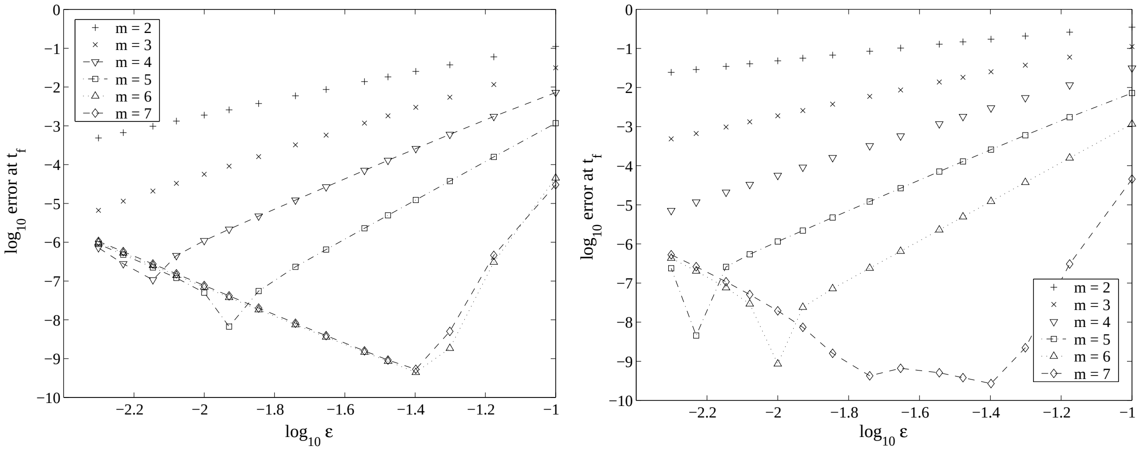


Figure 5.10: Convergence of the SRM using `dopr54` for Problem 1 with fixed $h = 0.005$. The plot on the left is of the higher-order method and the right is of the lower-order (embedded) method.

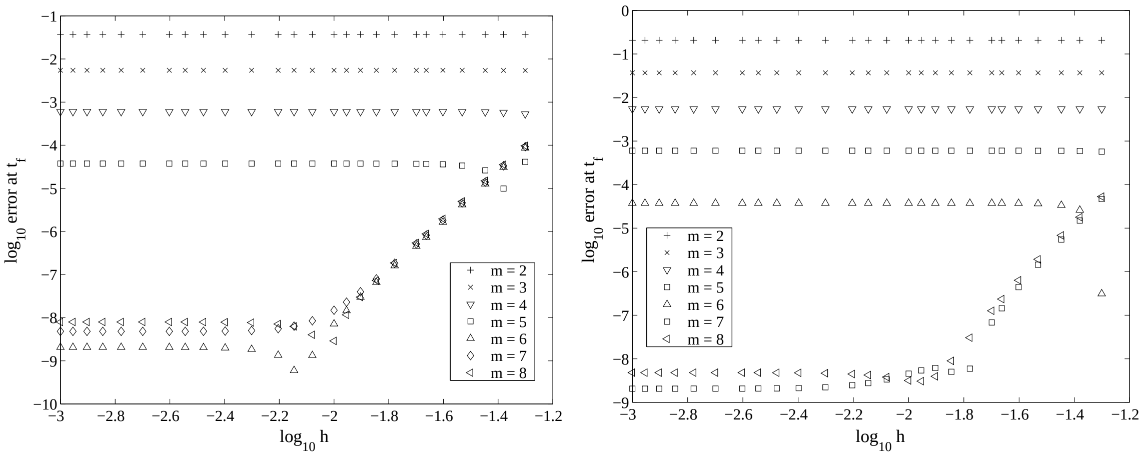


Figure 5.11: Convergence of the SRM using `dopr54` for Problem 1 with fixed $\varepsilon = 0.05$. The plot on the left is of the higher-order method and the right is of the lower-order (embedded) method.

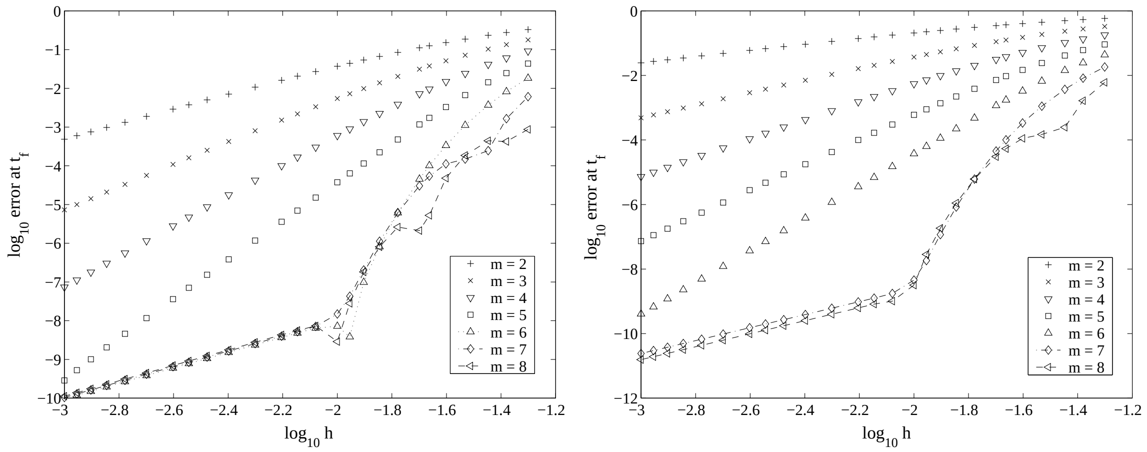


Figure 5.12: Convergence of the SRM using `dopr54` for Problem 1 with fixed $\varepsilon = 5h$. The plot on the left is of the higher-order method and the right is of the lower-order (embedded) method.

5.5 PSRM Convergence Tests on Problem 1

Recall from Section 3.5 we expect no better than $\mathcal{O}(h^2)$ from the PSRM using the simple prediction scheme. Table 5.7 confirms that this is indeed the case. Note for fixed h, we expect different results than for the SRM: consider Dormand-Prince and recall from Section 3.5 that the order is theoretically

$$\|x^{(1)}(t) - x(t)\| = \mathcal{O}(\varepsilon h^4) + \mathcal{O}(\frac{h^4}{\varepsilon^2}).$$

Then, if we fix h, we obtain

$$\begin{aligned}\|x^{(1)}(t) - x(t)\| &= \mathcal{O}(\varepsilon) + \mathcal{O}(\frac{1}{\varepsilon^2}),\\ &= \mathcal{O}(\varepsilon).\end{aligned}$$

Thus we expect only $\mathcal{O}(\varepsilon)$ convergence in the case of fixed h (as seen in Table 5.7).

Table 5.7: Convergence of the PSRM for Problem 1 for various choices of ε and h.

method	$\varepsilon = h$	fixed $h = 0.001$	$\varepsilon = 5h$	fixed $\varepsilon = 0.05$
expected →	2	1	2	1
`trap`	2.0002	0.9992	exact	1.0000
`tvd32` (higher-order)	1.9842	0.9998	1.9896	0.9907
(lower-order)	1.9786	1.0190	1.9895	0.9785
`erk4`	1.9810	0.9995	1.9764	0.9803
`dopr54` (higher-order)	1.9459	0.9988	1.9597	1.0000
(lower-order)	1.9460	0.9989	1.9597	0.9999

5.6 Stable Versus Unstable Index Reduction

Solving the index-reduced (index-2) slider-crank for $t \in [0, 20]$ using the SRM for index-2 problems results in solutions as seen in Figure 5.13. The index-2 slider-crank appears to be unstable and the SRM was unable to penetrate the singularity correctly. The SRM for index-3 problems with stabilized index-reduction performs much better: it passed smoothly through the singularity multiple times over a period of 70 seconds and still yielded an accurate solution (see Figure 5.14).

5.7 Automatic Step-size Control for the SRM

If we apply step-size control to the index-3 slider-crank problem for $t \in [0, 10\pi]$, then, as seen in Figure 5.15, the algorithm performs acceptably with 242 accepted steps and 15 rejected steps.[1] However, if we integrate further to, say, $t_f = 34.5$, then, as seen in Figure 5.16a, the step-size control scheme fails in the sense that it reaches a point near $t = 34.5$ after which it rejects every step (the SRM thus progresses to the next step only when forced[2]). The number of accepted steps has increased to 800

[1] Also note that initial step-size selection as discussed in Section B.2 is not yet implemented in the SRM and thus 5 of those rejections are due to a poorly chosen (hard-coded) initial step-size.

[2] For practical reasons, the SRM code will force a step if the recommended step-size from the step-size control scheme falls below a threshold value — this behavior can be observed in Figure 5.16b.

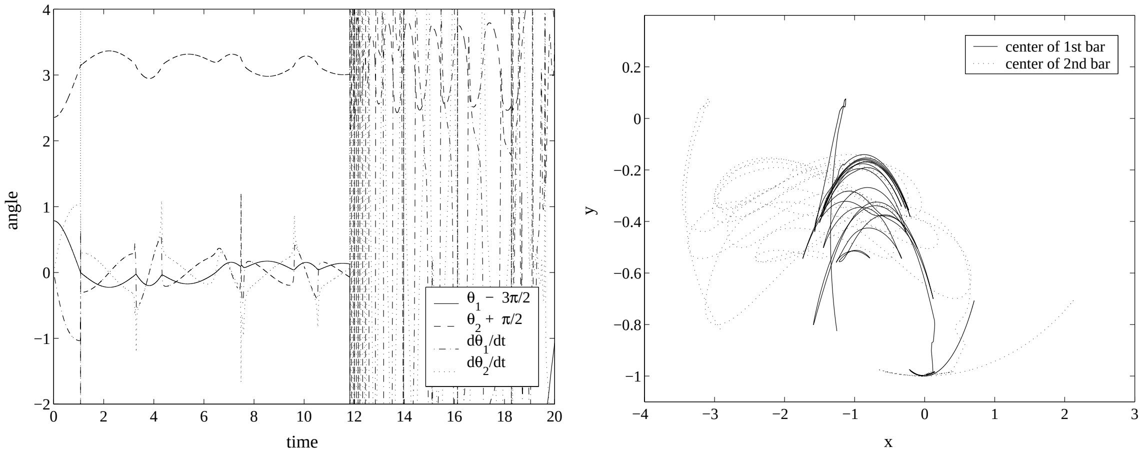


Figure 5.13: Angles and their derivatives (left) and paths of the centers of the bars (right) in the solution of the index-2 slider-crank obtained using the SRM with `tvd32` with $m = 3$, $h = 0.001$, $\varepsilon = h$, $t_f = 20$.

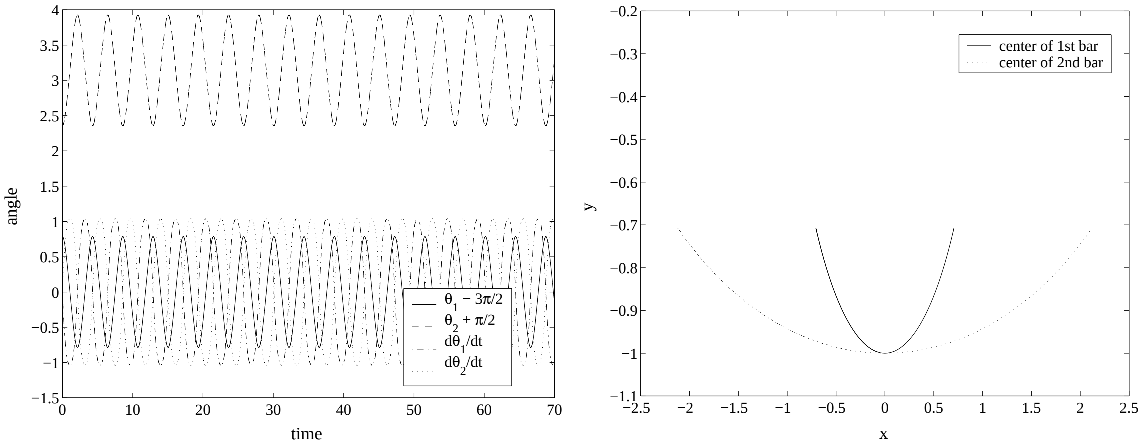


Figure 5.14: Angles and their derivatives (left) and paths of the centers of the bars (right) in the solution of the index-3 slider-crank obtained using the SRM with `tvd32` with $m = 3$, $h = 0.001$, $\varepsilon = h$, $t_f = 70$.

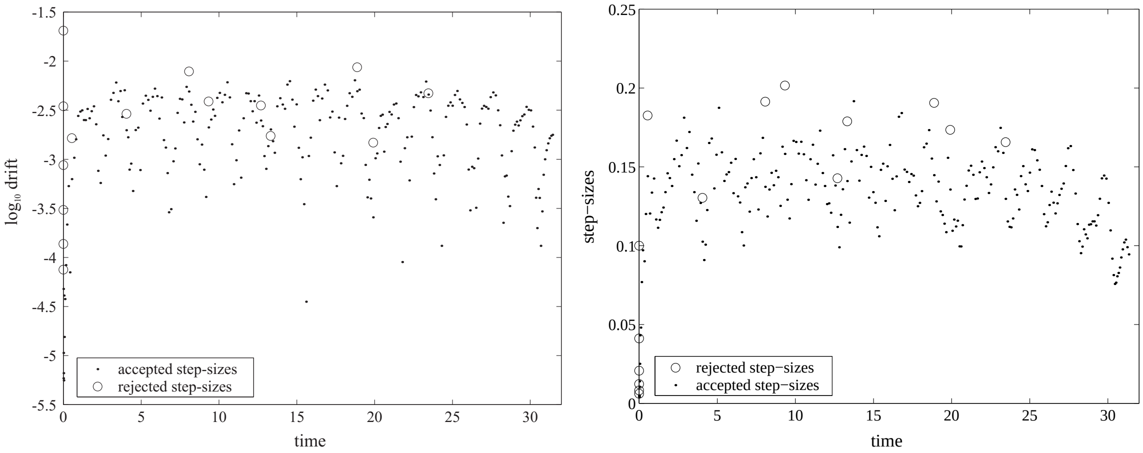


Figure 5.15: Solving the slider-crank using SRM with `tvd32` and step-size control. In this case, $t \in [0, 10\pi]$, $m = 8$, ATOL = RTOL = 0.01, and $\varepsilon = h$.

and the number of rejected steps has increased to 655; this is clearly unacceptable and also, from examining Figure 5.16c, we note these rejections appear unnecessary from the point of view of drift. That is, before encountering this *impasse point*, the drift showed no tendency to increase and indeed, the decreasing step-sizes observed in Figure 5.16a result in substantial decrease in drift (as seen on the right-most side of Figure 5.16c) and thus we expect that the computed solution is (relatively) near the solution manifold as defined by the constraint equation (see Section 1.7). Hence, the error should be relatively low and we do not expect the observed step-size rejections.

Additionally, we note that the location in time of this impasse point moves if we vary the parameters (ε, m, ATOL, RTOL, etc.) but no obvious pattern emerges. In particular, the location is independent of points of singularity (indeed, in Figure 5.16 the solution has already successfully passed through many singularities).

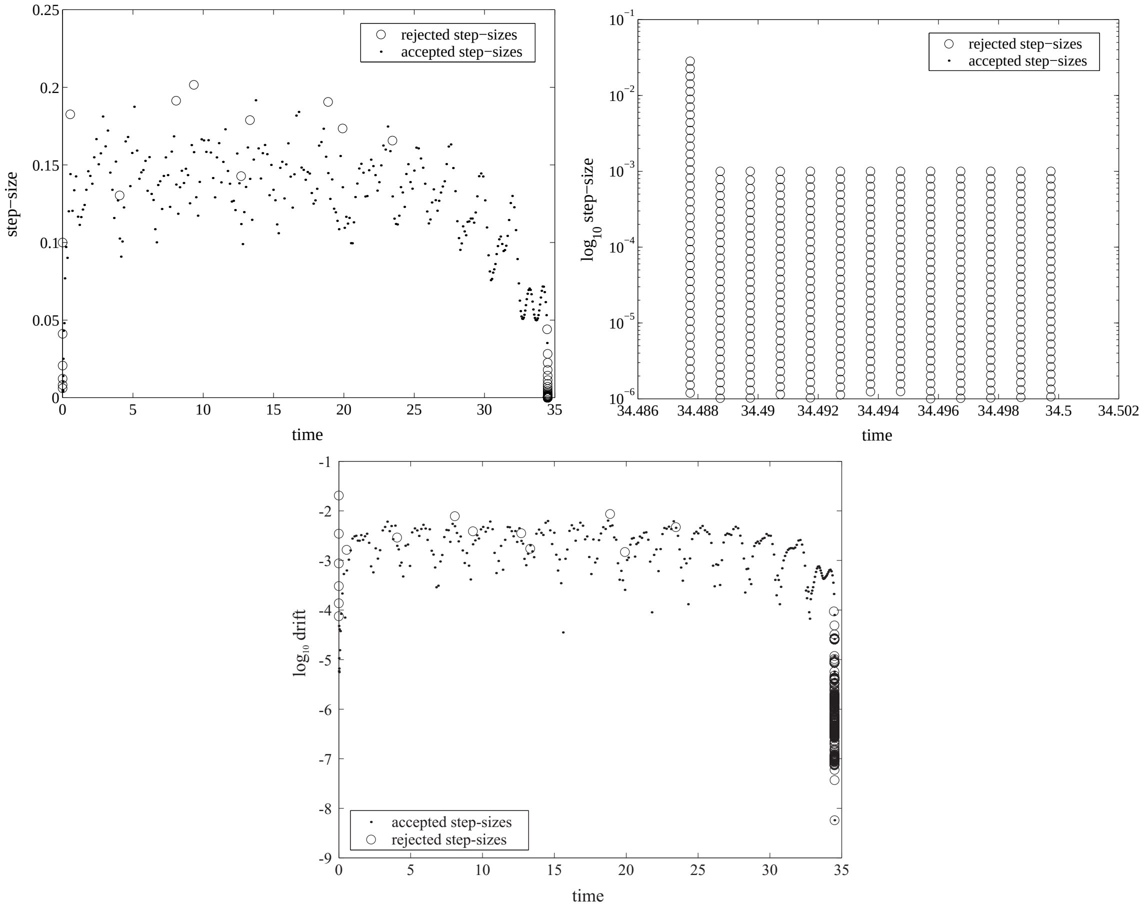


Figure 5.16: Solving the slider-crank using SRM with `tvd32` and step-size control. In this case, $t \in [0, 34.5]$, $m = 8$, ATOL = RTOL = 0.01, and $\varepsilon = h$.

Chapter 6

Conclusions

The Sequential Regularization Method (SRM) presents considerable advantages over some traditional methods for solving differential-algebraic equations. The Predicted Sequential Regularization Method (PSRM) represents a considerable saving in computation expense over the SRM, particularly when using a higher-order ODE solver.

The SRM has order $\mathcal{O}(\varepsilon^m) + \mathcal{O}(h^p)$ where ε is the *regularization parameter*, m is the number of SRM iterations, h is the step-size, and p is the order of the ODE solver. These theoretical results were supported with a simple example problem with a known solution. However, complications that are not explicitly explained by the theory arise on this simple problem and on other problems. These are particularly noticeable when using Runge-Kutta schemes with intermediate stage points (i.e., abscissae between mesh points).

The order conditions for the PSRM are more complicated and must be treated on a case-by-case basis for each ODE solver. For this thesis, we considered a simple prediction scheme that limits convergence to $\mathcal{O}(h^2)$ and thus future research endeavors will include an investigation of alternative prediction schemes. That being said, $\mathcal{O}(h^2)$ is, however, sufficient for many applications.

The most significant area of this research is in the direction of automatic step-size control; this important technique has never been attempted with either the SRM or the PSRM. As seen in Section 5.7, we have encountered partial success in this direction. The step-size control behaves locally as we expect from the theory, but it seems to encounter points that it cannot penetrate. These appear to be, at least superficially, unrelated to the geometry of the problem and it is hypothesized that they may be due to regions of local stiffness. Again, this is an area requiring further research given the crucial nature of step-size control to any code that is to be widely applied.

Appendix A

Runge-Kutta Methods

A.1 Forward Euler

Runge-Kutta methods are in some sense a generalization of Euler's classic method for solving ODEs. Euler's method, commonly referred to by numerical analysts as *Forward Euler*, is used to solve initial-value problems (IVPs) of the form

$$x' = f(x,t), \qquad x(t_0) = x_0, \tag{A.1}$$

where $x \in \mathbb{R}^n$, $t \in [t_0, t_f] \subset \mathbb{R}$. Suppose we are interested in approximating the value $x(t_f)$. We begin by discretizing the domain of t into small intervals $[t_n, t_{n+1}]$ which we refer to as *steps*. The width, $h_n = t_{n+1} - t_n$, of each step is referred to as the *(local) step-size*. We then approximate the value of x at each of $t_n, n \geq 1$ by evaluating

$$x_{n+1} = x_n + f(x_n, t_n)h_n, \tag{A.2}$$

which can be thought of as a projection along a tangent plane to the (unknown) function governing x. We then simply repeat this this process for each x_n until we obtain an approximation to $x(t_f)$.

This process is simple to understand and implement with software. Its simplicity also makes it an excellent candidate for analysis of numerical methods. However, the global error of Forward Euler is $\mathcal{O}(h)$ where $h = \max h_n$. Thus, if we are numerically solving a IVP with $t_f - t_0 = 1$ and we desire, say 6 decimal places of accuracy, then we would need $\mathcal{O}(10^6)$ steps!

A.2 Runge-Kutta Methods

Without loss of generality, let us simplify notation slightly by denoting $x_n = x_0$, $x_{n+1} = x_1$, $t_n = t_0$, and $t_{n+1} = t_1$.

Definition 6 (Explicit Runge-Kutta Method) *Let $s \in \mathbb{N}$ be the number of stages and let $a_{21}, a_{31}, a_{32}, \ldots, a_{s1}, \ldots, a_{s,s-1}, b_1, \ldots, b_s, c_2, \ldots, c_s \in \mathbb{R}$. Then the* s-stage explicit Runge-Kutta method *is given by*

$$\begin{aligned} K_1 &= f(x_0, t_0), \\ K_2 &= f(x_0 + ha_{21}K_1, t_0 + c_2 h), \\ &\vdots \\ K_s &= f(x_0 + h(a_{s1}K_1 + \cdots + a_{s,s-1}K_{s-1}), t_0 + c_s h), \qquad \text{(A.3a)} \\ x_1 &= x_0 + h(b_1 K_1 + \cdots + b_s K_s), \qquad \text{(A.3b)} \end{aligned}$$

with the condition that $c_i = \sum_{j=1}^{i-1} a_{ij}$.

The values of the constants $a_{21}, a_{31}, a_{32}, \ldots, a_{s1}, \ldots, a_{s,s-1}, b_1, \ldots, b_s, c_2, \ldots, c_s$ can be displayed (and thus a Runge-Kutta scheme can be uniquely determined) in a

Butcher tableau as follows:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_1 & \cdots & b_{s-1} & b_s
\end{array}
\tag{A.4}
$$

or more compactly as

$$
\begin{array}{c|c}
c & A \\
\hline
 & b^T
\end{array},
\tag{A.5}
$$

where $c, b \in \mathbb{R}^s$ and A is a strictly lower triangular real $s \times s$ matrix known as simply the *Butcher matrix*. The elements of the vectors b and c are respectively known as the *quadrature weights* and *abscissae* of the Runge-Kutta method.

A.2.1 Implicit Runge-Kutta Methods

For completeness, note that Runge-Kutta methods exist when A is not strictly lower triangular; such methods are referred to as *implicit Runge-Kutta schemes* and certain families are especially effective for stiff equations. Solving for the K_i of an implicit Runge-Kutta scheme requires that one solves a system of implicit equations and is thus a more computationally expensive process per step than explicit Runge-Kutta schemes.

A.3 Embedded Runge-Kutta Methods

One particularly convenient method for automatic step-size control (see Appendix B) requires two estimates whose orders differ by one. Ideally, we would like to obtain the second estimate with a minimal amount of additional work. In particular, we would like to avoid additional evaluations of our function for the righthand-side of the differential equation because this is typically the most expensive part of the computation. By sharing stage information, an embedded Runge-Kutta method obtains an second estimate with no additional function evaluations. Consider the Butcher tableau:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_1 & \cdots & b_{s-1} & b_s \\
 & \hat{b}_1 & \hat{b}_1 & \cdots & \hat{b}_{s-1} & \hat{b}_s
\end{array}
\tag{A.6}
$$

with

$$x_1 = x_0 + h(b_1 K_1 + \cdots + b_s K_s), \tag{A.7a}$$

$$\hat{x}_1 = x_0 + h(\hat{b}_1 K_1 + \cdots + \hat{b}_s K_s), \tag{A.7b}$$

such that x_1 is of order p and the error estimator $\hat{x}_1$ is of order q. Usually $q = p - 1$ or $q = p + 1$. Embedded Runge-Kutta methods are usually referred to by "<name> $p(q)$" for example Fehlberg 4(5) describes a method with fourth-order x_1 and fifth-order $\hat{x}_1$ to be used as an error estimator.

The early embedded Runge-Kutta schemes where similar to Fehlberg 4(5) in that the error estimator $\hat{x}_1$ was of a higher order than x_1. It seems natural that we would want to use the better (higher-order) result as our numerical answer. Indeed, Dormand and Prince have constructed schemes where the errors of the higher-order result are minimized and the lower-order result is only used for the step-size selection (see, [7]). The Butcher tableau for the fifth-order Dormand-Prince 5(4) is shown in Table C.5.

A.4 Continuous Runge-Kutta Methods

The usual Runge-Kutta schemes provide high-order approximations only at the end points of each step. However, there is often a desire for accurate approximates at points between those chosen by the automatic step-size control mechanism, for example, for the purpose of creating an accurate graphical representation of the solution. This is particularly true when solving an ODE with a higher-order Runge-Kutta scheme: often the step-size may be so large that graphics output is visibly polygonal.

One solution is of course to use a smaller step-size, but this can be very inefficient and unnecessary from an accuracy standpoint. The continuous extension to an order-p Runge-Kutta scheme provides a much more satisfactory solution by generating accurate *dense* output; specifically, order-$(p-1)$ output at arbitrary points $x^* = x_0 + \theta h$, $\theta \in (0, 1] \subset \mathbb{R}$ by means of interpolating polynomials of sufficiently high degree.

To construct these polynomials, we begin with an s-stage, order-p Runge-Kutta method and possibly add one or more stages to obtain the s^*-stage, order-p method

$$\begin{array}{c|c} c & A \\ \hline & b^T(\theta) \end{array}, \tag{A.8}$$

with

$$b_i(\theta) = \sum_{k=1}^{p-1} b_{ik}\theta^k. \tag{A.9}$$

The continuous extension polynomials can then be used to find $x(t_0 + \theta h)$ as follows:

$$x(t_0 + \theta h) \approx x_{0+\theta} = x_0 + h \sum_{i=1}^{s^*} b_i(\theta) K_i, \tag{A.10}$$

where the K_i are calculated as above in Appendix A.2. Note that continuous extensions, although continuous by definition, are generally not differentiable even when $x(t)$ happens to be.

A fourth-order continuous extension of Dormand-Prince 5(4) (obtained from [12]), which incidently does not require any additional stages (i.e., $s^* = s$), is shown in Table C.6.

Appendix B

Automatic Step-Size Control

Automatic step-size control permits the solution of problems by prescribing error tolerances in the solution rather than specifying a step-size. For practical applications, this is usually more natural; it is not usually obvious beforehand what step-size will be required to produce a prescribed accuracy in the solution.

Even under circumstances where such ease-of-use conditions are less important, step-size control still provides tangible benefits if the problem is more stiff on some regions of the domain. Using a fixed step-size implementation, we would be forced to apply the restrictive step-size required for numerical stability in the stiff regions over the entire domain. Using a step-size control scheme on such problems allows the use of small steps for the stiff regions and larger steps for the non-stiff regions. In this sense, step-size control actually stabilizes a numerical integration (see, [19]).

B.1 Step-Size Control Theory

Step-size control requires that we calculate two estimates of the solution which can be used to estimate the error. We use this estimate of the error to either *accept* or *reject* the current step-size and estimate a new step-size for future steps. One of the most efficient (in terms of computational expense) methods for obtaining the second estimate of the solution is through the use of embedded Runge-Kutta schemes as discussed in Appendix A.3.

We begin our description of step-size control by assuming that we have two solution estimates x and $\hat{x}$ of order $\mathcal{O}(p)$ and $\mathcal{O}(\hat{p})$ such that $|p-\hat{p}|=1$. Also, as in Appendix A.2, we simplify notation slightly by denoting $x_n = x_0$, $\hat{x}_n = \hat{x}_0$, $x_{n+1} = x_1$, $\hat{x}_{n+1} = \hat{x}_1$, $t_n = t_0$, and $t_{n+1} = t_1$.

Now assuming we are at t_0 with a current step-size h, we calculate x_1 and $\hat{x}_1$ from the embedded Runge-Kutta method. The error on x_1 is $x_1 - \hat{x}_1$ (recall from Appendix A.3 that $\hat{x}$ exists to control the error on x) and if we assume $x \in \mathbb{R}^n$ then for $i = 1, 2, \ldots, n$ let

$$\delta_i = \mathrm{ATOL}_i + \max(|x_{0i}|, |x_{1i}|)\mathrm{RTOL}_i. \tag{B.1}$$

where ATOL_i and RTOL_i are the user-defined absolute and relative tolerances respectively. We want the error $x_1 - \hat{x}_1$ to be such that for each i,

$$|x_{1i} - \hat{x}_{1i}| \leq \delta_i, \tag{B.2}$$

and thus

$$\frac{|x_{1i} - \hat{x}_{1i}|}{\delta_i} \leq 1. \tag{B.3}$$

Now we can measure the total error $x_1 - \hat{x}_1$ with

$$\xi = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_{1i} - \hat{x}_{1i}}{\delta_i}\right)^2}. \tag{B.4}$$

Ideally, if the error is distributed evenly among the components, then $|x_{1i} - \hat{x}_{1i}| = \delta_i$ and hence $\xi = 1$. In this sense, the optimal step-size will correspond to $\xi = 1$.

Now considering the Taylor expansions of x_1 and $\hat{x}_1$, note that $\xi = \mathcal{O}(h^{q+1})$ where $q = \min(p, \hat{p})$. Thus

$$\xi \approx ch^{q+1} \qquad \text{for some } c \in \mathbb{R}, \tag{B.5}$$

and, as noted above, the optimal step-size h_{opt} occurs when $\xi = 1$ so

$$1 \approx c\,(h_{\text{opt}})^{q+1}. \tag{B.6}$$

Putting (B.5) and (B.6) together, we get

$$h_{\text{opt}} = h\left(\frac{1}{\xi}\right)^{\frac{1}{q+1}}. \tag{B.7}$$

At this point, if $\xi \leq 1$ then x_1 is *accepted* and we use h_{opt} as the step-size for the next step. On the other hand, if $\xi > 1$ then x_1 is *rejected* and we repeat the step using a step-size of h_{opt}.

B.1.1 Practical Considerations

Note that it is computationally expensive to reject x_1 and repeat a step. In fact, we would rather take a few steps which are smaller than necessary than have to repeat a step. This leads to the following practical considerations:

1. We usually multiply (B.7) but a safety factor $\alpha < 1$ to make h_{opt} slightly smaller, thus increasing the likeliness of accepting the next step. Some common values for α are 0.8, 0.9, $(0.25)^{\frac{1}{q+1}}$, or $(0.38)^{\frac{1}{q+1}}$.

2. We do not want the step-size to change too rapidly so we set minimum and maximum factors, $\alpha_{\min}$ and $\alpha_{\max}$, by which h can increase. That is, we define h_{new} as

$$h_{\text{new}} = h\min\left(\alpha_{\max}, \max\left(\alpha_{\min}, \alpha\left(\tfrac{1}{\xi}\right)^{\frac{1}{q+1}}\right)\right), \tag{B.8}$$

where α is the safety factor from above, and typically $\alpha_{\max} \in [1, 5]$, $\alpha_{\min} = \frac{1}{\alpha_{\max}}$. Also it has been shown to be advisable to use $\alpha_{\max} = 1$ after a step has been rejected (see, [20]).

Thus for the revised test, if $\xi \leq 1$ then y_1 is accepted and h_{new} is used as the step-size for the next step. Otherwise, if $\xi > 1$, then y_1 is rejected and the step is repeated with a new step-size of h_{new}.

B.2 Initial Step-size Selection

This above method works fine for automatic step-size control however, we still need a way to choose the initial step-size. Note from above, that the local error for x_1 is $\mathcal{O}(h^{q+1})$ and hence for some $c' \in \mathbb{R}$, the local error is approximately

$$\xi = \left(c' x^{(q+1)}(t_0)\right) h^{q+1} = c' h^{q+1} x^{(q+1)}(t_0). \tag{B.9}$$

$x^{(q+1)}(t_0)$ is unknown but we can replace it with approximations of the first and second derivatives of the solution. In [9], the following algorithm based on Forward Euler is presented:

1. Set $d_0 = \|x_0\|$, $d_1 = \|f(t_0, x_0)\|$, and $\delta_i = \text{ATOL}_i + |x_{0i}|\text{RTOL}_i$. Note that d_0 and d_1 are the magnitudes of x_0 and x_0' respectively.

2. As a first guess, set
$$h_0 = 0.01 \left(\frac{d_0}{d_1}\right), \tag{B.10}$$
unless $d_0 < 10^{-5}$ or $d_1 < 10^{-5}$, in which case take $h_0 = 10^{-6}$.

3. Take a step of Forward Euler (i.e., calculate $x_1 = x_0 + h_0 f(t_0, x_0)$) and use the result to compute $f(t_0 + h_0, x_1)$.

4. Using a finite difference, compute d_2 as an estimate of the magnitude of x'',

$$d_2 = \left\| \frac{f(t_0 + h_0, x_1) - f(t_0, x_0)}{h_0} \right\|. \tag{B.11}$$

5. Set

$$h_1 = \sqrt[q+1]{\frac{0.01}{\max(d_1, d_2)}}, \tag{B.12}$$

   unless $\max(d_1, d_2) \leq 10^{-15}$, in which case set $h_1 = \max\left(10^{-6}, 10^{-3}h_0\right)$.

6. Take the initial step-size to be

$$h = \min(100h_0, h_1). \tag{B.13}$$

Following this algorithm requires only two function evaluations, both of which are required for embedded Runge-Kutta so it is a cheap way to avoid choosing a very bad initial step-size.

Appendix C

Butcher Tableaux

Table C.1: Butcher tableau for Trapezoidal Rule (`trap`).

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Table C.2: Butcher tableau for Heun's Method (`heun`).

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

Table C.3: Butcher tableau for Total-Variation-Diminishing 3(2) (`tvd32`).

$$
\begin{array}{c|ccc}
0 & & & \\
1 & 1 & & \\
\frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\
\hline
y_1 & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \\
\hline
\hat{y}_1 & \frac{1}{2} & 0 & \frac{1}{2}
\end{array}
$$

Table C.4: Butcher tableau for Classic Fourth-Order Runge-Kutta (`erk4`).

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Table C.5: Butcher tableau for Dormand-Prince 5(4) (`dopr54`).

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{35}{33} & \frac{46732}{5247} & \frac{49}{76} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
y_1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 \\
\hline
\hat{y}_1 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

Table C.6: A Continuous Extension of Dormand-Prince 5(4).

$$
\begin{aligned}
b_1(\theta) &= \theta\left(1 + \theta\left(-\tfrac{1337}{480} + \theta\left(\tfrac{1039}{360} + \theta\left(-\tfrac{1163}{1152}\right)\right)\right)\right)\\
b_2(\theta) &= 0\\
b_3(\theta) &= \tfrac{100}{3}\theta^2\left(\tfrac{1054}{9275} + \theta\left(-\tfrac{4682}{27825} + \theta\left(\tfrac{379}{5565}\right)\right)\right)\\
b_4(\theta) &= -\tfrac{5}{2}\theta^2\left(\tfrac{27}{40} + \theta\left(-\tfrac{9}{5} + \theta\left(\tfrac{83}{96}\right)\right)\right)\\
b_5(\theta) &= \tfrac{18225}{848}\theta^2\left(-\tfrac{3}{250} + \theta\left(\tfrac{22}{375} + \theta\left(-\tfrac{37}{600}\right)\right)\right)\\
b_6(\theta) &= -\tfrac{22}{7}\theta^2\left(-\tfrac{3}{10} + \theta\left(\tfrac{29}{30} + \theta\left(-\tfrac{17}{24}\right)\right)\right)\\
b_7(\theta) &= 0
\end{aligned}
$$

Bibliography

[1] U. Ascher, H. Chin, L. Petzold, and S. Reich, *Stabilization of constrained mechanical systems with DAEs and invariant manifolds*, J. Mech. Structures Mach., **23** (1995), 135–157.

[2] U. Ascher, H. Chin, and S. Reich, *Stabilization of DAEs and invariant manifolds*, Numer. Math., **67** (1994), 131–149.

[3] U. Ascher and P. Lin, *Sequential regularization methods for higher index DAEs with constraint singularities: the linear index-2 case*, SIAM J. Numer. Anal. **33** (1996), 1921–1940.

[4] U. Ascher and P. Lin, *Sequential regularization methods for nonlinear higher-index differential-algebraic equations*, SIAM J. Sci. Comput. **18** (1997), 160–181.

[5] J. Baumgarte, *Stabilization of constraints and integrals of motion in dynamical systems*, Comp. Math. Appl. Mech. Eng. **1** (1972), 1–16.

[6] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland Publishing Co., New York, 1989.

[7] J. Dormand and P. Prince, *A family of embedded Runge-Kutta formulae*, J. Comp. Appl. Math. **6** (1980), 19–26.

[8] W. Enright and P. Muir, *Superconvergent interpolants for the collocation solution of boundary value ordinary differential equations*, SIAM J. Sci. Comput. **21** (1999), 227–254.

[9] I. Gladwell, L. Shampine, and R. Brankin, *Automatic selection of the initial step size for an ODE solver* J. Comput. Appl. Math. **18** (1987), 175–192.

[10] C. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1971.

[11] E. Hairer, C. Lubich, and M. Roche, *The Numerical Solution of Differential-Algebraic System by Runge-Kutta Methods*, Springer-Verlag, Berlin, 1989.

[12] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer-Verlag, Berlin, 1987.

[13] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer-Verlag, Berlin, 1991.

[14] A. Hindmarsh, *ODEPACK, a systematized collection of ODE solvers*, Scientific Computing, (1982), 55–64.

[15] P. Lin, *A sequential regularization method for time-dependent incompressible Navier-Stokes equations*, SIAM J. Numer. Anal. **34** (1997), 1051–1071.

[16] P. Lin and R. Spiteri, *A predicted sequential regularization method for index-*2 *hessenberg DAEs*, unpublished manuscript.

[17] L. Petzold, *Differential/algebraic equations are not ODE's*, SIAM J. Sci. Stat. Comput. **3** (1982), 367–384.

[18] R. von Schwerin, *MultiBody System SIMulation: Numerical Methods, Algorithms, and Software*, *Lecture notes in computational science and engineering, 7*, Springer-Verlag, Berlin, 1999.

[19] L. Shampine and A. Witt, *Control of local error stabilizes integrations*, J. Comput. Appl. Math. **62** (1995), 333–351.

[20] L. Shampire and A. Witt, *The art of writing a Runge-Kutta code. II*, Appl. Math. Comput. **5** 93–121.