

The Computable Differential Equation

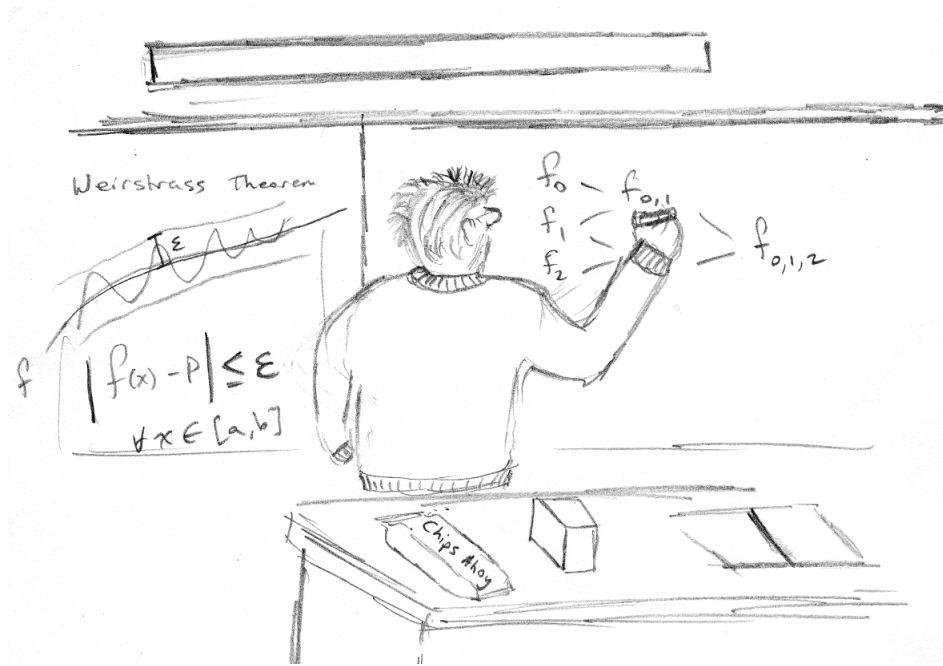
Lecture Notes for Math 582B “Topics in Numerical Analysis”

California State University Northridge
Spring 2007
Class 17770

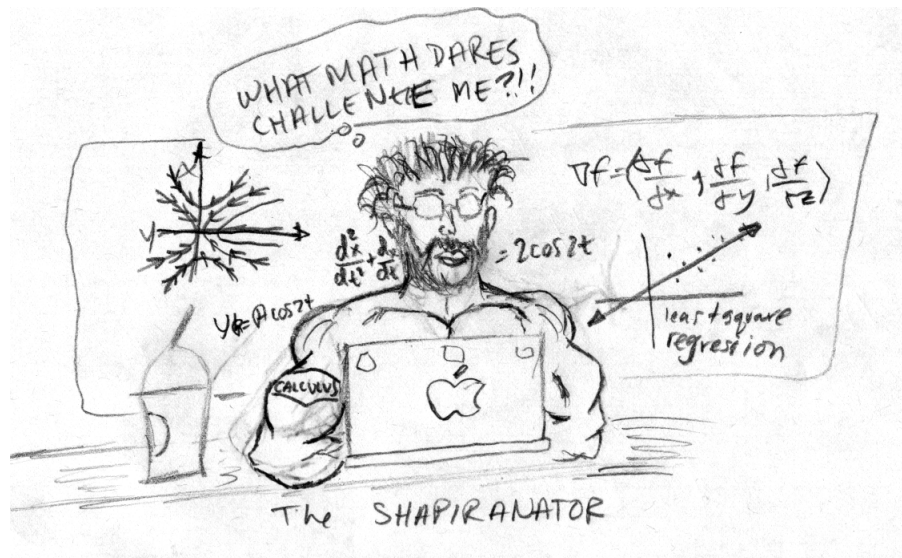
This is a DRAFT document and may contain errors.
Version: May 23, 2007. Changes are indicated by arrows in the margin

Bruce E. Shapiro, Ph.D.
©2007¹

¹The material in these notes may not be duplicated or distributed in any format without permission. These notes are intended for use by students in Math 582B at CSUN during the Spring Semester, 2007. Students may print and copy notes as required for their own use but should not distribute them to others without permission. This documentation is provided in the hope that it will be useful but without any warranty, without even the implied warranty of merchantability or fitness for a particular purpose. The document is provided on an “as is” basis and the author has no obligations to provide corrections or modifications. The author makes no claims as to the accuracy of this document. In no event shall the author be liable to any party for direct, indirect, special, incidental, or consequential damages, including lost profits, or unsatisfactory class performance or grade reports, arising out of the use of this document or any software described herein, even if the author has been



Your fearless leader. Above: Typical view during a class lecture. Below: Typical view during an exam. The pictures were drawn by former students. The consumption of cookies and caffeinated beverages during class is optional but is strongly encouraged.



advised of the possibility of such damage.

Contents

1	Classifying The Problem	1
1.1	Defining ODEs	1
1.2	Initial Value Problems	4
1.3	The Fundamental Theorem	6
1.4	Boundary Value Problems	10
1.5	Differential-Algebraic Equations	12
1.6	Delay Differential Equations	13
1.7	Numerical Solutions of Differential Equations	14
1.8	Computer Assisted Analysis	15
2	Successive Approximations	25
2.1	Picard Iteration	25
2.2	Fixed Point Theory	29
2.3	Fixed Point Iteration in a Normed Vector Space	34
2.4	Convergence of Successive Approximations	38
3	Approximate Solutions	43
3.1	The Forward Euler Method	43
3.2	Epsilon-Approximate Solutions	48
3.3	The Fundamental Inequality	51
3.4	Cauchy-Euler Existence Theory	53
3.5	Euler's Method for Systems	56
3.6	Polynomial Approximation	60
3.7	Peano Existence Theorem	64
3.8	Dependence Upon a Parameter	66
4	Improving on Euler's Method	69
4.1	The Test Equation and Problem Stability	69
4.2	Convergence, Consistency and Stability	73
4.3	Stiffness and the Backward Euler Method	79
4.4	Other Euler Methods	86
5	Runge-Kutta Methods	89
5.1	Taylor Series Methods	89
5.2	Numerical Quadrature	92

5.3	Traditional Runge-Kutta Methods	95
5.4	General Form of Runge-Kutta Methods	99
5.5	Order Conditions	102
5.6	Step Size Control for ERK Methods	104
5.7	Implicit Runge-Kutta Methods	106
5.8	IRK Methods based on Collocation	109
5.9	Páde Approximants and A-Stability	112
6	Linear Multistep Methods	117
6.1	Multistep Methods	117
6.2	The Root Condition for Linear Multistep Methods	119
6.3	Backward Difference Formula	121
6.4	Adams Methods	124
6.5	BDF Methods	126
6.6	Nyström and Milne Methods	128
6.7	Other Types of Multistep Methods	130
6.8	Prediction-Correction (P(EC) ⁿ E) Techniques	133
7	Delay Differential Equations	135
7.1	Basic Theory	135
7.2	Method Of Steps	138
7.3	Numerical Implementation of Method of Steps	143
7.4	Runge-Kutta Methods for Delay Differential Equations	147
7.5	Continuous Methods	149
8	Boundary Value Problems	153
8.1	Shooting	153
8.2	Basic Theory of Boundary Value Problems	156
8.3	Shooting Revisited	159
8.4	One-step Methods	162
9	Differential Algebraic Equations	167
9.1	Concepts	167
9.2	Linear Differential Algebraic Equations with Constant Coefficients	170
9.3	General Linear Differential Algebraic Equations	173
9.4	Hessenberg Forms for Linear and Nonlinear DAEs	174
9.5	Implementation: A Detailed Example	177
9.6	BDF Methods for DAEs	184
9.7	Runge-Kutta Methods for DAEs	187
10	Appendix on Analytic Methods (Math 280 in a Nutshell)	193
	Bibliography	199

Timeline on Computable Differential Equations

1660s	Newton circulates notes on limit theory of Calculus
1674	Leibnitz, infinitesimal theory of Calculus
1691	Raphson, root finding algorithm
17??	Newton invents Simpson's Method
1739	Newton Publishes
1740	Simpson invents Newton's Method
17??	Euler's Method
1755,1768	Euler publishes Calculus texts
1883	Bashforth, Adams
1885	Schwartz, Method of Successive Iterations
1887	Peano's theory
1890	Picard; Lindeloff
1895	Runge
1901	Kutta; Heun
1926	Moulton; Milne
1951	UNIVAC - first commercial computer
1957	First FORTRAN compiler
1963++	Butcher, Theory of RK
1963++	Driver, Theory of Delay Equations
1968-1977	Development of Macsyma
1969	APRAnet goes online
1971	Gear, BDF Methods for ODE's and DAE's
1974	GEAR (Hindmarsh)
1980	LSODE/LSODI (Hindmarsh, Petzold)
1982 - Present	DASSL Family
1983	ODEPACK
1985	Netlib goes online
1986	Petzold, RK Methods for DAE's
1986	GMRES
1988	Numerical Recipes Published
1988	Mathematica 1.0
1989	VODE
1994-1996	CVODE/PVODE family
1999	IDA (parallel version of CVODE/PVODE)
2005	Sundials

Chapter 1

Classifying The Problem

1.1 Defining ODEs

Definition 1.1 (Ordinary Differential Equation, ODE). *Let $y \in \mathbb{R}^n$ be a variable¹ that depends on t . Then we define a **differential equation** as any equation of the form*

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

where F is any function of the $2n + 1$ variables $t, y_1, y_2, \dots, y_n, y'_1, y'_2, \dots, y'_{n+1}$.

Definition 1.2. *We will call any function $\phi(t)$ that satisfies*

$$F(t, \phi(t), \phi'(t)) = 0 \tag{1.2}$$

a solution of the differential equation.

We will use the terms “Ordinary Differential Equation” and “Differential Equation”, as well as the abbreviations ODE and DE, interchangeably. More generally, on can include partial derivatives in the definition, in which case one must distinguish between “Partial” DEs (PDEs) and “Ordinary” DEs (ODEs). We will leave the study of PDEs to another class.

Equation 1.1 is, in general, very difficult, and often, impossible, to solve, either analytically (e.g., by finding a formula that describes y), or numerically (e.g., for example, by using a computer to draw a picture of the graph of the solution). Often it is possible to solve 1.1 explicitly for the derivatives:

$$y' = f(t, y) \tag{1.3}$$

Many important problems can be put into this form, and solutions are known to exist for a wide class of functions, particular as a result of theorem 1.3. The class of problems in which equation 1.1 can be converted to the form 1.3, at least locally, is not seriously restrictive from a practical point of view. The only requirements are

¹It may be either a scalar variable, in which case $n = 1$, or a vector variables, in which case n is the number of components of the vector.

that F be sufficiently smooth² and that the matrix of partial derivatives $\partial\mathbf{F}/\partial(\mathbf{y}')$ (Jacobian matrix) be nonsingular³ ($\partial F/\partial(y')$ for a scalar equation). Then by the implicit function theorem we can solve for y' locally. An equation of the form 1.1 for which the Jacobian is nonsingular is thus called an **ordinary differential equation**, and we will focus on equations of this form in the first several chapters of these notes. It turns out that an equation for which the Jacobian is singular actually has hidden constraints: it is really a combination of differential equations and algebraic constraints, and is called a **differential algebraic equation**.

Theorem 1.1. Implicit Function Theorem on \mathbb{R} .⁴ Let $F(t, y)$ have continuous derivatives $\partial F/\partial t$ and $\partial F/\partial y$ in the neighborhood of a point (x_0, y_0) , where

$$F(t_0, y_0) = 0, \quad \frac{\partial F(t_0, y_0)}{\partial y} \neq 0 \quad (1.4)$$

Then there are intervals I and J where

$$I = [t_0 - a, t_0 + a] \quad (1.5)$$

$$J = [y_0 - b, y_0 + b] \quad (1.6)$$

and a R rectangle $R = I \times J$, such that the equation $F(t, y)$ has precise one solution $y = f(t)$ lying in the rectangle R , such that

$$F(t, y(t)) = 0 \quad (1.7)$$

$$y(t) \in J \quad (1.8)$$

$$F_y(t, f(t)) \neq 0 \quad (1.9)$$

for all $t \in I$.

Example 1.1. Solve $y' = y$.

Solution. Writing $y' = dy/dt$ and integrating we find

$$\int \frac{1}{y} dy = \int dt \quad (1.10)$$

$$\ln|y| = t + C \quad (1.11)$$

$$y = Ke^t \quad (1.12)$$

where $K = \pm e^C$. There is no restriction on the values of either C or K . \square

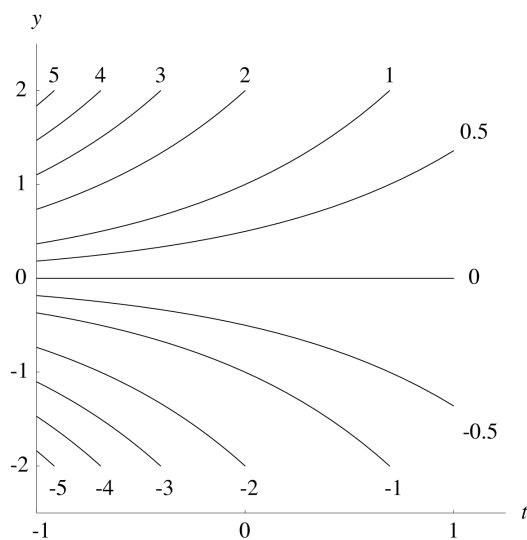
²Throughout these notes we will assume that F is sufficiently smooth without explicitly stating so. By “sufficiently” smooth we mean that F is continuously differentiable enough times to give us the results we want.

³Strictly speaking, nonsingularity is not really a requirement. Nonsingularity is sufficient to ensure that a solution exists, but is not required. There are examples of functions with singularities at points but for which solutions may exist.

⁴For a proof, see Richard Courant and Fritz John, *Introduction to Calculus and Analysis, Volume II/1*, Springer Classics in Mathematics, 1998, page 225

Thus we see that equation 1.1 (or 1.3) will often admit to an infinite number of solutions owing to arbitrary constants of integration that arise during its solution. For example 1.1 this is illustrated in figure 1.1, which shows the *one parameter family of solutions* to the example. A particular physical problem may only correspond to one member of this family. To fix down this constant, the problem must be further constrained. Such a constraint can take various forms. The nature of the constraint can have an enormous impact on our ability to solve the equation.

Figure 1.1: One parameter family of solutions to $y' = y$, showing the solutions for various values of the constant of integration.



The Implicit Function Theorem

Definition 1.3. Jacobian Matrix. Let $\mathbf{F}(y_1, y_2, \dots, z_1, z_2, \dots)$ be a vector function such that

$$\mathbf{F} = \begin{pmatrix} F_1(y_1, y_2, \dots) \\ F_2(y_1, y_2, \dots) \\ F_3(y_1, y_2, \dots) \\ \vdots \end{pmatrix} \quad (1.13)$$

The the **Jacobian Matrix** of the function F with respect to the variables \mathbf{y} is

$$\frac{\partial \mathbf{F}}{\partial \mathbf{y}} = \frac{\partial \mathbf{F}}{\partial (y_1, y_2, \dots)} = \begin{pmatrix} \partial F_1 / \partial y_1 & \partial F_1 / \partial y_2 & \partial F_1 / \partial y_3 & \dots \\ \partial F_2 / \partial y_1 & \partial F_2 / \partial y_2 & \partial F_2 / \partial y_3 & \dots \\ \partial F_3 / \partial y_1 & \partial F_3 / \partial y_2 & \partial F_3 / \partial y_3 & \dots \\ \vdots & & & \end{pmatrix} \quad (1.14)$$

Theorem 1.2. Implicit Function Theorem Let $A \subset \mathbb{R}^n \times \mathbb{R}^m$ be an open set and let $F : A \mapsto \mathbb{R}^m$ be p -times continuously differentiable. Let $(x_0, y_0) \in A$ such that $F(x_0, y_0) = 0$, and suppose that the Jacobian at (x_0, y_0) is non-singular,

$$\left| \frac{\partial F}{\partial (y_1, y_2, \dots, y_m)} \right|_{(x_0, y_0)} \neq 0 \quad (1.15)$$

where $x_0 \in \mathbb{R}^n$, $y_0, (y_1, y_2, \dots, y_m) \in \mathbb{R}^m$. Then there exists an open neighborhood $U \subset \mathbb{R}^n$ of x_0 and an open neighborhood $V \subset \mathbb{R}^m$ of y_0 and a unique function $f : U \mapsto V$, that is p -times continuously differentiable, such that

$$F(x, f(x)) = 0 \quad (1.16)$$

for all $x \in U$

1.2 Initial Value Problems

Equation 1.3 has an intuitive feeling as the description of a dynamical system: given any starting point y_0 , then the subsequent “motion” or “time-evolution” of y is given by equation 1.3. By adding an initial condition, that is, by specifying a point that the solution passes through, we obtain an **initial value problem** (IVP).

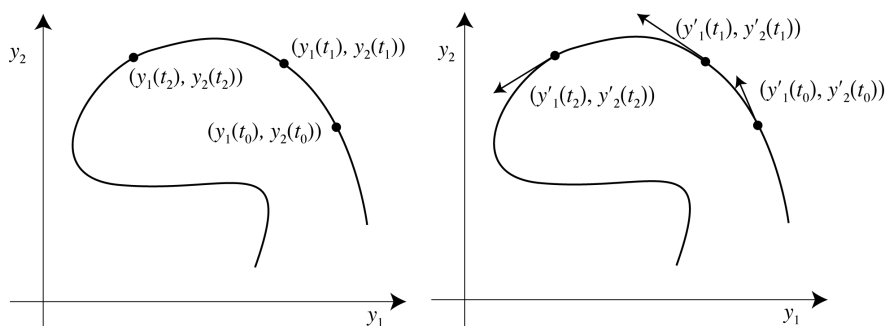
Definition 1.4. (Initial Value Problem) Let $D \in \mathbb{R}^{n+1}$ be a set. Let $(t_0, y_0) \in D$ and suppose that $f(t, y) : D \mapsto \mathbb{R}^n$. Then

$$y' = f(t, y) \quad (1.17)$$

$$y(t_0) = y_0 \quad (1.18)$$

is called an **initial value problem**. The constraint 1.18 is called an **initial condition**.

Figure 1.2: Illustration of a differential equation as a dynamical system. Given any starting point an object moves as described by the differential equation. The curve on the left shows the coordinates of an object at several time points. On the right, the points are annotated with the direction of motion, an arrow whose direction is specified by the components of the differential equation.



Example 1.2. Solve the initial value problem $y' = (3 - y)/2, y(2\pi) = 4$.

Solution. We can rearrange variables as

$$\frac{2dy}{3 - y} = dt \quad (1.19)$$

and integrate to obtain

$$-t = 2 + \ln |y - 3| + C \quad (1.20)$$

Substituting the initial condition gives

$$C = -2\pi - \ln |4 - 3| = -2\pi \quad (1.21)$$

which gives

$$|y - 3| = e^{\pi - t/2} \quad (1.22)$$

Thus either $y = 3 + e^{\pi - t/2}$ or $y = 3 - e^{\pi - t/2}$. At the initial condition, however, $y(2\pi) = 4$, which is only obtained with the plus sign in the solution. Hence

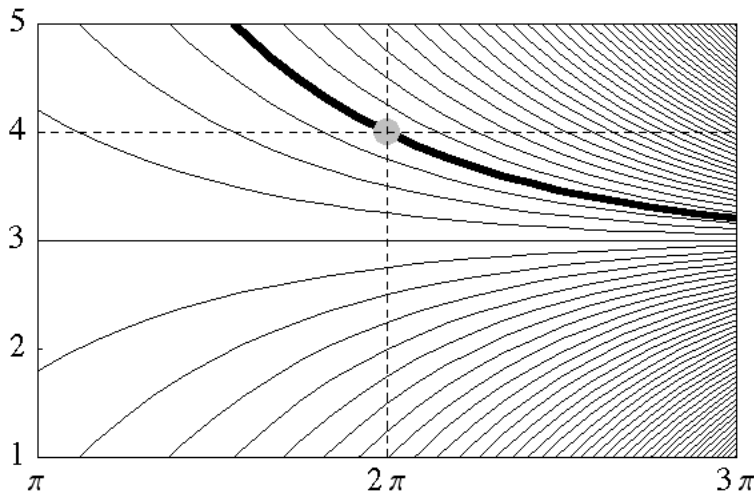
$$y = 3 + e^{\pi - t/2} \quad (1.23)$$

is the unique solution. The solution of the initial value problem is plotted in figure 1.3 in comparison with the one-parameter family. \square

We will say that an initial value problem is **well posed** if it meets the following criteria:

- A solution exists;

Figure 1.3: The one-parameter family of solutions for $y' = (3 - y)/2$ for different values of the constant of integration, and the solution to the initial value (heavy line) problem through $(t_0, y_0) = (2\pi, 4)$. The initial condition is indicated by the large gray dot.



- The solution is unique;
- The solution depends continuously on the data.

If a problem is not well posed then there is no point in trying to solve it numerically, so we begin our study of initial value problems by looking at what it takes to make a problem well posed. We will find that a **Lipshitz Condition**, defined below in definition 1.5 is sufficient to ensure that the problem is well posed.

1.3 The Fundamental Theorem

The importance (and usefulness) of initial value problems is enhanced by a general existence theorem and the fact that under appropriate conditions (namely, a *Lipshitz Condition*) the solution is unique. While we will defer the proof of this statement until later, we will present one of many different versions of the fundamental existence theorem.

Definition 1.5. [*Lipshitz Condition*]. A function $f(t, y)$ on D is said to be **Lipshitz** (or **Lipshitz continuous**, or **satisfy a Lipshitz condition**) on y if there exists some constant $K > 0$ if for all $(x, y_1), (x, y_2) \in D$ then

$$|f(x, y_1) - f(x, y_2)| \leq K|y_1 - y_2| \quad (1.24)$$

The constant K is called a **Lipshitz Constant** for f . We will sometimes denote this as $f \in \mathcal{L}(y; K)(D)$.

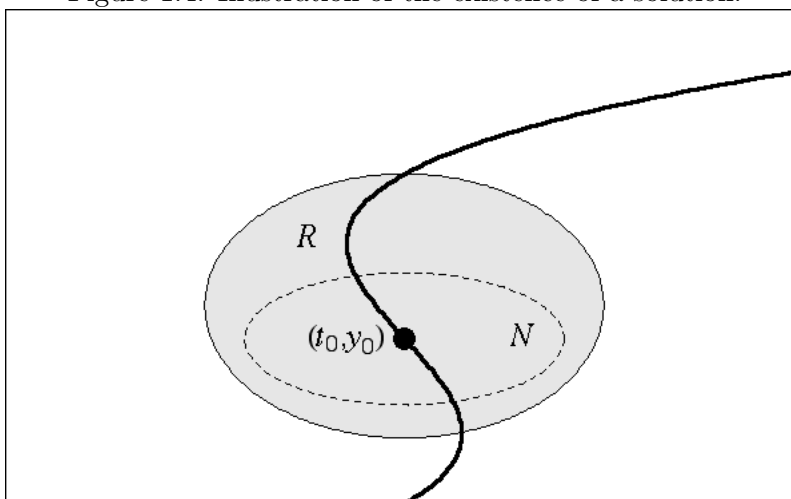
Theorem 1.3. Fundamental Existence and Uniqueness Theorem *Suppose that $f(t, y) \in L(y; K)(R)$ for some convex domain R . Then for any point $(t_0, y_0) \in R$ there exists a neighborhood N of (t_0, y_0) and a unique differentiable function $\phi(t)$ on N satisfying*

$$y' = f(t, \phi(t)) \quad (1.25)$$

such that $y'(t_0) = y_0$.

The existence theorem is illustrated in figure 1.4. Given any initial value, there is some solution that passes through the point. Observe that the existence of the solution is not guaranteed globally, only within some open neighborhood of the initial condition.

Figure 1.4: Illustration of the existence of a solution.



Theorem 1.4 (Continuous dependence on IC). *Under the same conditions, the solution depends continuously on the initial data, i.e., if \tilde{y} is a solution satisfying the same ODE with $\tilde{y}(t_0) = \tilde{y}_0$, then*

$$|y(t) - \tilde{y}(t)| \leq e^{Kt} |y_0 - \tilde{y}_0| \quad (1.26)$$

Theorem 1.5 (Perturbed Equation). *Under the same conditions, suppose that \tilde{y} is a solution of the perturbed ODE,*

$$\tilde{y}' = f(t, \tilde{y}) + r(t, \tilde{y}) \quad (1.27)$$

where r is bounded on D , i.e., there exists some $M > 0$ such that $|r(t)| \leq M$ on D . Then

$$|y(t) - \tilde{y}(t)| \leq e^{Kt} |y_0 - \tilde{y}_0| + \frac{M}{K} (e^{Kt} - 1) \quad (1.28)$$

Proving that a function is Lipschitz is considerably eased by the following theorem.

Theorem 1.6. *Suppose that $|\partial f/\partial y|$ is bounded by K on a set D . Then $f(t, y) \in \mathcal{L}(y, K)(D)$.*

Proof. The result follows immediately from the mean value theorem. Let $(t, y_1), (t, y_2) \in D$. Then there is some number c between y_1 and y_2 such that

$$|f(t, y_1) - f(t, y_2)| = |f_y(c)||y_1 - y_2| < K|y_1 - y_2| \quad (1.29)$$

Hence f is Lipschitz in y on D . \square

Example 1.3. *Show that a unique solution exists to the initial value problem*

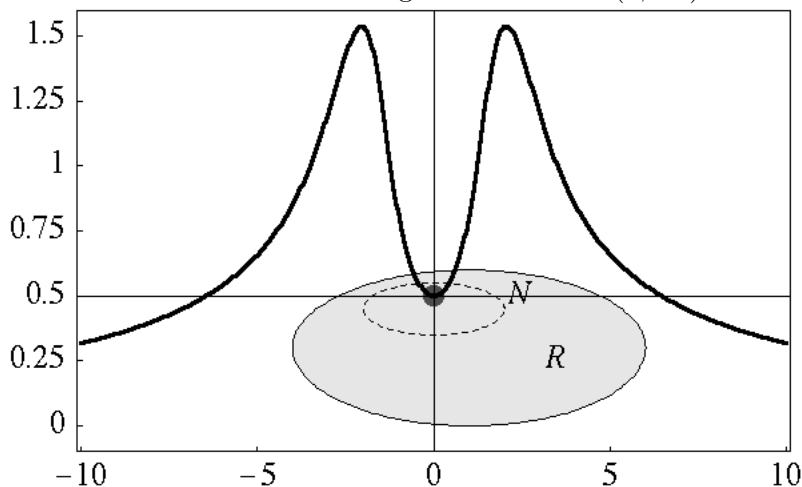
$$y' = \sin(ty), \quad y(0) = 0.5 \quad (1.30)$$

Solution. We have $f(t, y) = \sin(ty)$, hence $f_y = t \cos(ty)$. Thus $|f_y| \leq |t|$ which is bounded for any finite range of t . Let R be a bounded, convex set enclosing $(0, 0.5)$, and let

$$K = 1 + \sup_{t \in R} |t| \quad (1.31)$$

Since R is bounded we know that the supremum exists. By adding 1 we ensure that we have a number that is strictly larger than the maximum value of $|t|$. Then K is a Lipschitz constant for f and hence a unique solution exists in some neighborhood N of $(0, 0.5)$. See figure 1.5.

Figure 1.5: A solution exists in some neighborhood N of $(0, 0.5)$. See Example 1.3



Example 1.4. *Analyze the uniqueness of solutions to*

$$y' = \sqrt{4 - y^2} \quad (1.32)$$

$$y(0) = 2 \quad (1.33)$$

Solution. Finding a “solution” is easy enough. We can separate the variables and integrate. It is easily verified (by direct substitution) that

$$y = 2 \sin\left(t + \frac{\pi}{2}\right) \quad (1.34)$$

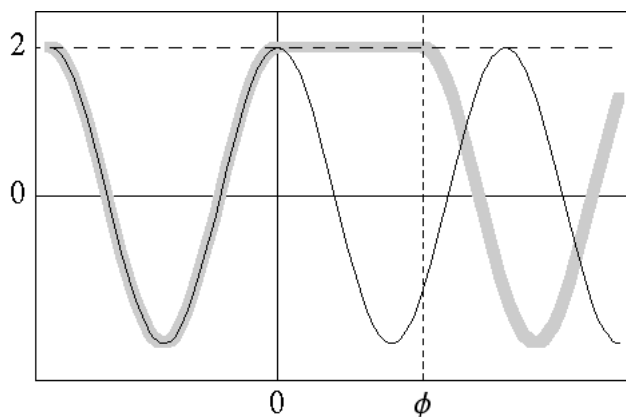
satisfies both the differential equation and the initial condition, hence it is a solution.

It is also easily verified that $y = 2$ is a solution, as are functions of the form

$$y = \begin{cases} 2 \sin\left(t + \frac{\pi}{2}\right) & t < 0 \\ 2 & 0 \leq t \leq \phi \\ 2 \sin\left(t + \frac{\pi}{2} - \phi\right) & t > \phi \end{cases} \quad (1.35)$$

for any positive real number ϕ . See Figure 1.6.

Figure 1.6: There are several solutions to $y' = \sqrt{4 - y^2}$ that pass through the point $(0, 2)$. See Example 1.4



Since the solution is not unique, any condition that guarantees existence must be violated. We have two such conditions: the boundedness of the partial derivative, and the Lipschitz condition. The first implies the second, and the second implies uniqueness. By

$$\frac{\partial f}{\partial y} = \frac{-y}{\sqrt{4 - y^2}} \quad (1.36)$$

which is unbounded at $y = 2$. So the first condition is violated. Of course, a violation of the condition does not ensure non-uniqueness, all it tells us is that uniqueness is not ensured.

What about the Lipschitz condition? Suppose that the function $f(x) = \sqrt{4 - y^2}$ is Lipschitz with Lipschitz constant $K > 0$ on some domain D . Then for any y_1, y_2 in

D ,

$$K|y_1 - y_2| \geq |f(y_1, y_2)| = \left| \sqrt{4 - y_1^2} - \sqrt{4 - y_2^2} \right| \quad (1.37)$$

Let $y_2 = 2$ and $y_1 = 2 - \epsilon$ for some small number ϵ . Then

$$K|\epsilon| \geq \left| \sqrt{4 - (2 - \epsilon)^2} \right| \quad (1.38)$$

$$\geq \left| \sqrt{4\epsilon - \epsilon^2} \right| \quad (1.39)$$

$$K^2\epsilon^2 \geq 4\epsilon - \epsilon^2 \quad (1.40)$$

$$(K^2 + 1)\epsilon^2 \geq 4\epsilon \quad (1.41)$$

$$K^2 + 1 \geq \frac{4}{\epsilon} \quad (1.42)$$

$$K^2 \geq \frac{4}{\epsilon} - 1 \quad (1.43)$$

But since we can choose ϵ to be arbitrarily small (including 0), the right hand side of the equation can be arbitrarily large. But then K is not a finite number, especially when $\epsilon = 0$. So $f(t, y)$ is not Lipschitz, either. Again, this does not guarantee non-uniqueness; it just tells us that uniqueness is not guaranteed. \square

1.4 Boundary Value Problems

Definition 1.6 (Boundary Value Problem). *Let $D \in \mathbb{R}^{n+1}$ be a set. Let $(t_0, y_0) \in D$ and suppose that $f(t, y) : D \mapsto \mathbb{R}^n$. Then*

$$y' = f(t, y) \quad (1.44)$$

$$g(y(0), y(b)) = 0 \quad (1.45)$$

for some function g on \mathbb{R}^{2n} is called a **boundary value problem (BVP)**. The constraint 1.45 is called a **boundary condition**.

In a typical boundary problem is one where the solution is constrained at both ends of the interval. A scalar first order boundary problem would thus be overdetermined; hence any BVP is by necessity a first order vector system (see box).

Example 1.5. *Solve the boundary value problem*

$$y'' + \omega^2 y = 0 \quad (1.46)$$

$$y(0) = 1 \quad (1.47)$$

$$y(2\pi) = 1 \quad (1.48)$$

Solution. We observe that any linear combination of sin and cos functions satisfies the differential equation:

$$y = a \cos t + b \sin t \quad (1.49)$$

for any values of the numbers a and b . The first boundary condition gives

$$1 = a \cos 0 + b \sin 0 = a \quad (1.50)$$

Hence

$$y = \cos t + b \sin t \quad (1.51)$$

From the second boundary condition,

$$1 = \cos 2\pi + b \sin 2\pi = 1 \quad (1.52)$$

In other words, any value of b will satisfy the boundary value problem. There are an infinite number of solutions. \square

Higher Order Equations

Any higher order equation $y^{(n)} = f(t, y, y', y'', \dots, y^{(n-1)})$ can always be reduced to a first order system by making the following change of variables:

$$\xi_1 = y \quad (1.53)$$

$$\xi_2 = y' \quad (1.54)$$

$$\vdots \quad (1.55)$$

$$\xi_n = y^{(n-1)} \quad (1.56)$$

We then have the n^{th} order system

$$\xi_1' = \xi_2 \quad (1.57)$$

$$\xi_2' = \xi_3 \quad (1.58)$$

$$\vdots \quad (1.59)$$

$$\xi_n' = f(t, \xi_1, \xi_2, \dots, \xi_{n-1}) \quad (1.60)$$

Example 1.6. Rewrite the equation $x'' + \omega^2 x = 0$ as a first order system.

Solution. We define $u = x$, $v = x'$. Then $u' = x'' = v$ and $v' = x''' = -\omega^2 x = -\omega^2 u$. We can write this as \Leftarrow

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \quad \Leftarrow \quad (1.61)$$

or $y' = Ay$, where $y = \begin{pmatrix} u \\ v \end{pmatrix}$ and $A = \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix}$. \square

Example 1.7. Repeat the previous example with boundary conditions

$$y(0) = 1 \quad (1.62)$$

$$y(2) = 1 \quad (1.63)$$

Solution. As before we obtain

$$y = \cos t + b \sin t \quad (1.64)$$

from the first boundary condition. But now the second condition gives

$$1 = \cos 2 + b \sin 2 \quad (1.65)$$

$$b = \frac{1 - \cos 2}{\sin 2} \approx 1.55741 \quad (1.66)$$

hence there is precisely one solution:

$$y = \cos t + \frac{1 - \cos 2}{\sin 2} \sin t \quad \square \quad (1.67)$$

Thus we see that the solution of a BVP is not only not necessarily unique, but there may be an infinite number of solutions. To understand why that is the case requires delving into the theory of orthogonal functions and generalized Fourier analysis, which are subjects for other classes. Boundary value problems are also closely related to partial differential equations.

1.5 Differential-Algebraic Equations

Definition 1.7 (Differential-Algebraic Equations). *Let $D \in \mathbb{R}^{2n+1}$ be a set and suppose that $F(t, y, y') : D \mapsto \mathbb{R}^n$. Then*

$$F(t, y, y') = 0 \quad (1.68)$$

where the Jacobian matrix $\partial_{y'} F$ is singular

$$\det \left| \frac{\partial(F_1, F_2, \dots)}{\partial(y'_1, y'_2, \dots)} \right| = 0 \quad (1.69)$$

is called an **explicit differential-algebraic equation (DAE)**. If there are some functions $f(t, y, z)$ and $g(t, y, z)$ and a set of additional algebraic variables $z_1(t), z_2(t), \dots$ such that equation 1.68 can be rewritten as

$$y' = f(t, y, z) \quad (1.70)$$

$$0 = g(t, y, z) \quad (1.71)$$

then the systems is called a **semi-explicit DAE**.

It is always possible to rewrite equation 1.70 as a fully explicit DAE in terms of the vector $u = \begin{pmatrix} y \\ z \end{pmatrix}$, where z are the algebraic constraints, specifically,

$$F(t, y, z) = \begin{pmatrix} f(t, y, z) - y' \\ g(t, y, z) \end{pmatrix} \quad (1.72)$$

Then we have

$$F(t, y, z) = 0 \quad (1.73)$$

but the Jacobian matrix becomes nonsingular:

$$\frac{\partial F(t, y, z)}{\partial(y, z)} = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \quad (1.74)$$

The general theory of differential algebraic equations is still being developed. A system of the form

$$Ay'(t) + By(t) = f(t) \quad (1.75)$$

where A and B are square matrices and A is singular, is called a **linear constant coefficient DAE**. If the determinant

$$\det \lambda A + B \neq 0 \quad (1.76)$$

is not identically zero (considered as a function of the variable λ), then 1.75 is solvable.

1.6 Delay Differential Equations

Definition 1.8. (*Functional differential equation*) If a differential systems can be written explicitly as a function of a function of the abscissa variable, notably

$$y' = f(t, y(t), y(u(t))) \quad (1.77)$$

then it is know as a **functional differential equation (FDE)**. If the it can be written as

$$y' = f(t, y(t), y(t - \tau_1, t - \tau_2, \dots)) \quad (1.78)$$

then it is known as a **delay differential equation (DDE)**.

Example 1.8. A typical functional differential equation is

$$y' = y(t) + y(\sin(t)) \quad (1.79)$$

A typical DDE is given by

$$y' = y(t) + y(t - 3) + 7y(t - 5) \quad (1.80)$$

DDEs arise frequently in control theory and population growth. The theory of functional differential equations is not as well advanced as other types of DEs and there are fewer numerical methods for solving them. The best understood functional equations are delay equations. Delay equations are similar to initial value problems, in that the value of the function must be known at the starting point. They are more complicated, however, in that the value must also be known over an entire interval preceding t_0 , of duration given by the greatest delay in the equation. Thus

delay equations might also be called **initial function problems** or just **initial problems**⁵.

Let

$$0 \leq \tau_1 < \tau_2 < \cdots < \tau_m \quad (1.81)$$

be a set of m positive numbers representing the delays, the largest delay being denoted by τ_m . If y appears explicitly without any delay then $\tau_1 = 0$. Then we may define the scalar **initial function problem** (IFP) then as

$$y'(t) = f(t, y(t - \tau_1, t - \tau_2, \dots, t - \tau_m)) \quad (1.82)$$

$$y(t) = g(t), \quad t_0 - \tau_m \leq t \leq t_0 \quad (1.83)$$

Then there is a general existence and uniqueness theorem, which we state here without proof.

Theorem 1.7. *Let $D \in [t_0, t_0 + \beta) \times \mathbb{R}^m$ be convex; suppose that $f : D \mapsto \mathbb{R}$ is continuous on D ; and that $g(t) : [t_0 - \tau_m] \mapsto D$. Then if $r_1 \neq 0$ the DDE (equations 1.82, 1.83) has a unique solution on $[t_0, t_0 + \alpha)$, where $0 < \alpha \leq \beta$. Furthermore, if $\alpha < \beta$ then $y(t)$ approaches the boundary of D as $t \rightarrow \alpha$. If $\tau_1 = 0$ then the DDE has at most one solution.*

1.7 Numerical Solutions of Differential Equations

Roughly speaking, a numerical solution is a set of points on the curve that defines the solution to the differential equation (we will formalize this statement more precisely in a later section). The t -values are called a **mesh** or **grid**, and the interval between successive points is called the **step size** or **mesh interval**. By connecting the dots, using some appropriate interpolation scheme, we can recover the values of the solution at any point in its domain.

We will call a numerical method **consistent** if it satisfies the differential equation and all associated constraints

We will say that a method **converges** if we can obtain any desired degree of accuracy by picking the step size sufficiently small.

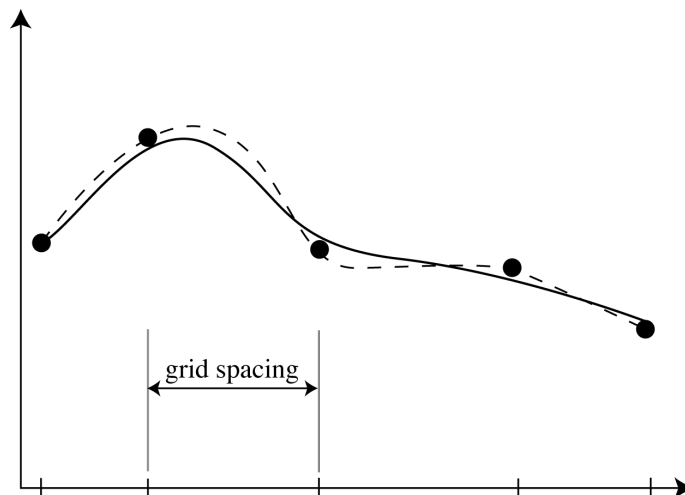
A problem will be called **well posed** if there for any small change in the problem causes a bounded change in the actual solution.

A numerical solution will be called **stable** if any small change in the problem causes a bounded change in the numerical solution.

The bulk of this class will focus on finding and analyzing various techniques for solving a differential equation numerically.

⁵this terminology follows R. D. Driver (1978) *Introduction to Ordinary Differential Equations*, Harper and Row

Figure 1.7: Illustration of a numerical solution. The actual solution is illustrated by the solid curve, the numerical by a set of points. The reconstruction of the solution at intermediate values is accomplished via an interpolation scheme (dashed curve).



1.8 Computer Assisted Analysis

The bulk of this course will focus on finding numerical solutions, and computer programs will aid us in implementing and studying these algorithms. Additionally, computers allow us to solve and study differential equations analytically, and to examine the qualitative format of the solutions. In this section we will give some examples in Mathematica.

Analytic Solution of ODEs with Mathematica

The principal function here is `DSolve`. For example, to find the one parameter family of solutions to

$$y' = 3ty \tag{1.84}$$

one would enter:

$$\text{DSolve}[y'[t]==3 t y[t], y[t], t] \tag{1.85}$$

Recall that in Mathematica, the multiplication operator `*` is optional, and if there is a space between operators, then multiplication is implied. The expressions

$$3 t y[t]$$

and

$$3*t*y[t]$$

are equivalent. The response to input 1.85 is

$$\left\{ \left\{ y[t] \rightarrow e^{\frac{3t^2}{2}} C[1] \right\} \right\} \quad (1.86)$$

This means that Mathematica has determined that there is one solution to the ODE and it has the form $y(t) = C_1 e^{3t^2/2}$, where C_1 is an arbitrary constant. Because no initial condition is specified, the system defines an arbitrary constant $C[1]$. Subsequent constants, if needed, are added as $C[i]$. For example, the second order equation

$$y'' = 3ty \quad (1.87)$$

is a form of Airy's equation (compare it with the previous example). The standard way of writing the solution to this is

$$y(t) = C_1 \text{Ai}\left(3^{1/3}t\right) + C_2 \text{Bi}\left(3^{1/3}t\right) \quad (1.88)$$

where Ai and Bi are the Airy functions of the first and second kind. To solve this in Mathematica, we enter

$$\text{DSolve}[y''[t]==3 t y[t], y[t], t] \quad (1.89)$$

and the system responds

$$\left\{ \left\{ y[t] \rightarrow \text{AiryAi}\left[3^{1/3}t\right]C[1] + \text{AiryBi}\left[3^{1/3}t\right]C[2] \right\} \right\} \quad (1.90)$$

To solve a systems of equations, such as

$$y' = 3x \quad (1.91)$$

$$x' = y \quad (1.92)$$

$$y(0) = \sqrt{3} \quad (1.93)$$

$$x(0) = 1 \quad (1.94)$$

one would type in

$$\text{DSolve}\left[\left\{y'[t] == 3 x[t], x'[t] == y[t], y[0] == \text{Sqrt}[3], x[0] == 1\right\}, \{x[t], y[t]\}, t\right]$$

and obtain the solution

$$\left\{ \left\{ y[t] \rightarrow \sqrt{3}e^{\sqrt{3}t}, x[t] \rightarrow e^{\sqrt{3}t} \right\} \right\} \quad (1.95)$$

Numerical Solutions of ODEs with Mathematica

Mathematica has a number of built in algorithms to solve differential equations numerically; the function that does this is `NDSolve`. To solve the systems of equations

$$y' = -x - .1y \quad (1.96)$$

$$x' = 3y \quad (1.97)$$

$$y(0) = \sqrt{3} \quad (1.98)$$

$$x(0) = 1 \quad (1.99)$$

for $0 \leq t \leq 100$, the command is

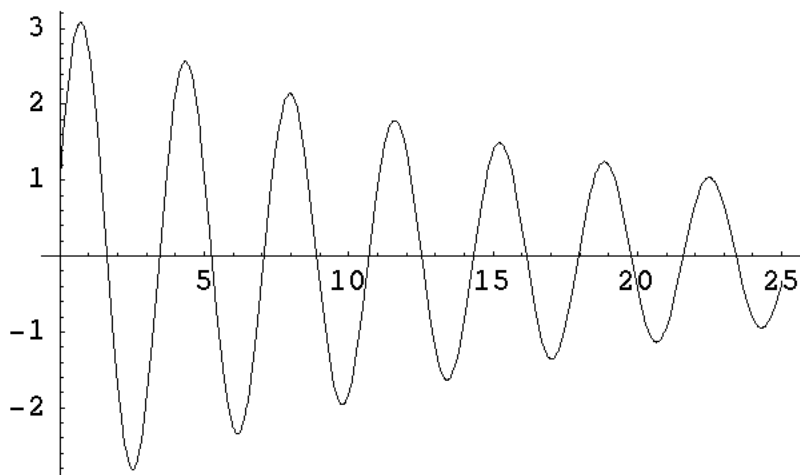
```
ns = NDSolve[{y'[t]==-x[t]-.1y[t], x'[t]==3y[t],
             y[0]==Sqrt[3], x[0]==1 },
            {x[t], y[t]}, {t, 0, 1000}]
```

The solution is returned is

```
{x[t] → InterpolatingFunction[{{0.,1000.}}, <>][t]
 y[t] → InterpolatingFunction[{{0.,1000.}}, <>][t]}
```

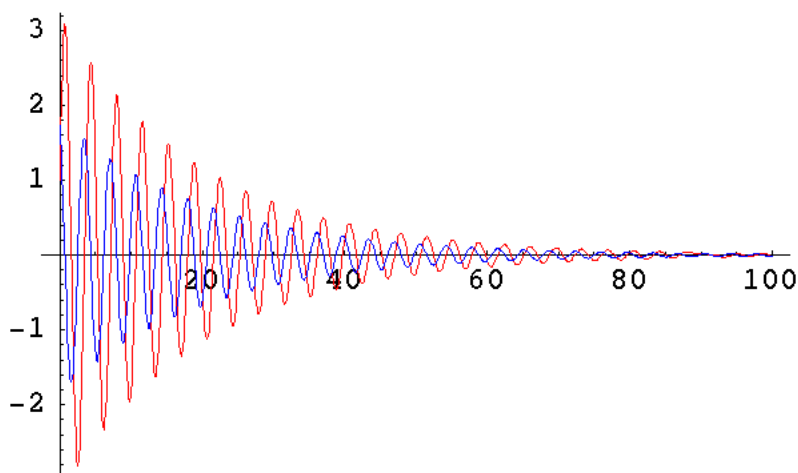
The interpolating functions can be treated like any other function. For example, to plot the solution for $x(t)$ from $t = 0$ to $t = 25$,

```
Plot[Evaluate[x[t] /. ns], t, 0, 25]
```



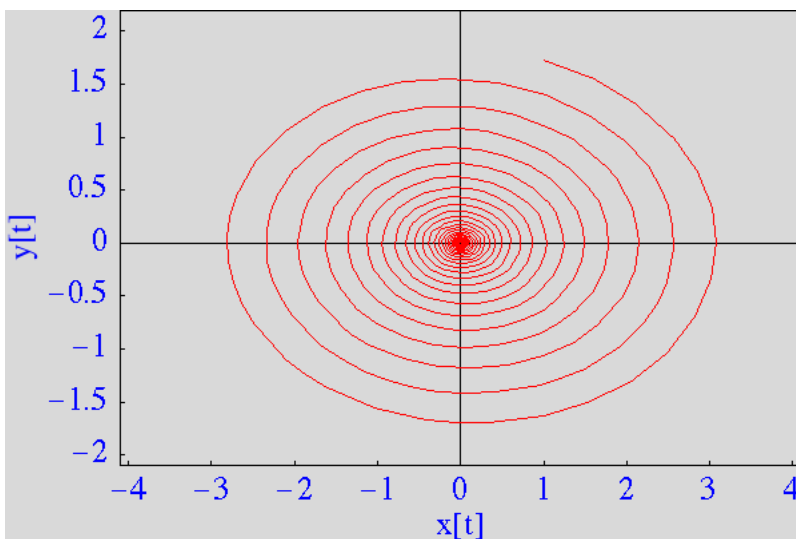
To plot both $x(t)$ and $y(t)$ from $t = 0$ to $t = 100$, with x plotted in red, and y plotted in blue, ←

```
Plot[Evaluate[{x[t], y[t]} /. ns],
      {t, 0, 100}, PlotStyle -> {Red, Blue}, PlotRange -> All]
```



To plot the set of points $(x(t), y(t))$ parametrically, one could use `ParametricPlot`. One can also set other plotting parameters, such as font sizes and styles, colors, line styles, etc. A few of these are illustrated in the following example:

```
ParametricPlot[Evaluate[x[t], y[t] /. ns], {t, 0, 100},
  PlotRange -> {{-4.1, 4.1}, {-2.1, 2.2}},
  Frame -> True, Axes -> True, AspectRatio -> 2/3,
  FrameTicks->{Range[-4, 4], Range[-2, 2, .5]},
  None, None},
  FrameLabel -> {"x[t]", "y[t]"},
  TextStyle -> {FontFamily -> Times, FontSize -> 16,
  FontColor -> Blue},
  PlotStyle -> {Red}, Background -> GrayLevel[.85]]
```



Implementing Numerical Methods with Mathematica

Finally we note that Mathematica contains a full high level programming language comparable to languages such as C and LISP. We will be using this language to implement our own versions of various algorithms as they arise.

The following piece of code gives an implementation of Euler's method using the traditional C-language like structures in Mathematica. It returns a list of number pairs $\{\{t_0, y_0\}, \{t_1, y_1\}, \dots\}$.

```
EulersMethod[f_, t0_, y0_, tmax_, h_] :=
Module[{y, t, solution},
solution = {{t0, y0}};
y = y0;
t = t0;
While[t < tmax,
  y = y + h*f[t, y];
  t = t + h;
  solution = Append[solution, {t, y}];
];
Return[solution];
]
```

←

Let us dissect this program. To begin with we have to tell Mathematica that we are defining a function:

```
EulersMethod[f_, t0_, y0_, tmax_, h_] :=
Module[{y, t, solution},
... stuff ...
]
```

This says that we are defining a new function `EulersMethod` that takes on 5 arguments (`f`, `t0`, `y0`, `tmax`, `h`) and has 3 local variables (`y`, `t`, `solution`). In the function declaration the names of the parameters always end with an underscore `_`; this tells Mathematica that these are the names of variables that are passed to the program. Within the function the underscore is not used. These arguments and the three local variables are only defined within the scope of the program, so that any other variable inside `EulersMethod` can refer to them directly, but no variable outside of the function definition knows of their existence. So if there is are global variable `f` or `y`, their values are not affected by the function definition. The next three lines initialize our local variables equal to certain combinations of argument values.

```
solution = {{t0, y0}};
y = y0;
t = t0;
```

This sets the local variable `solution` is going to be a list containing a single point (which itself is described by a list) $\{t_0, y_0\}$. The local variables `y` and `t` are then set equal to the parameter values `y0` and `t0` respectively. Next we follow with a `While` loop. The format of a `While` loop is the following:

```
While[expression,
      statement;
      statement;
      :
      statement;
]
```

When Mathematica gets to this statement, it looks at the value of *expression*. If *expression* is `True` then it executes each *statement* in succession. If *expression* is `False` the rest of the `While` loop is ignored. After executing the full sequence of statements, Mathematica returns to the top of the loop and evaluates *expression* again. The process is repeated until *expression* has a value of `False`. In our case we have the following loop:

```
While[t < tmax,
      y = y + h*f[t, y];
      t = t + h;
      solution = Append[solution, {t, y}];
];
```

Here we repeat the loop as long as the expression $t < t_{max}$ is true. Inside the loop the numbers y and t are updated according to specific formulas, and then the pair of numbers $\{t, y\}$ is added to the end of `solution`. Since the value of `t` is updated each iteration, so long as the initial value of `t` (which is given by `t0`) is smaller than `tmax`, the loop will eventually terminate. If `t0` > `tmax`, then the program will have an infinite loop. A more robust program would test for this possibility and print an error message before actually attempting to execute the infinite loop. Finally, when the loop completes we have one final statement,

```
Return[solution];
```

which tells us to return the list of values as the “value” of the function. This will be a list of numbers such as

$$\{\{t_0, y_0\}, \{t_1, y_1\}, \dots, \{t_n, y_n\}\}$$

for some integer $n = t_{max}/h$.

To invoke the function one would first define the function f . For example, to \Rightarrow solve the initial value problem

$$\begin{aligned} y' &= \cos t - 2t \sin t \\ y(0) &= 1 \end{aligned}$$

on the interval $[0, 4\pi]$ with a step size of 0.5, one would type

```
f[t_, y_] := Cos[t] - 2t Sin[t];
solution = EulersMethod[f, 0, 1, 4Pi, .5]
```

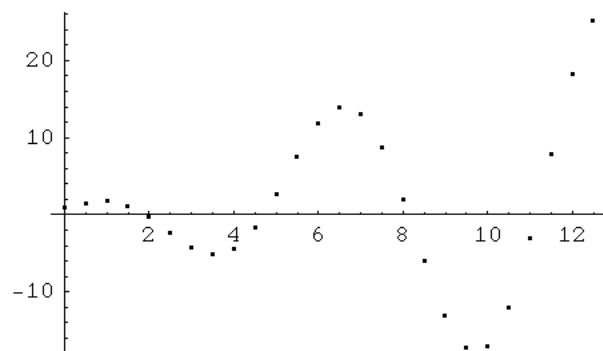
When we run the program it will return a list of values:

```
{{0, 1}, {0.5, 1.5}, {1., 1.69908}, {1.5, 1.12776}, {2., -0.333115},
{2.5, -2.35978}, {3., -4.25654}, {3.5, -5.17489}, {4., -4.41538},
{4.5, -1.71499}, {5., 2.5785}, {5.5, 7.51495}, {6., 11.7498}, {6.5, 13.9063},
{7., 12.9963}, {7.5, 8.77439}, {8., 1.91271}, {8.5, -6.07491}, {9., -13.1631},
{9.5, -17.3277}, {10., -17.1123}, {10.5, -12.0917}, {11., -3.09262},
{11.5, 7.90948}, {12., 18.2188}, {12.5, 25.0796}}
```

Since we set the value of the variable `solution` equal to the return value of `EulersMethod`, Mathematica will remember the values in this list and we can use them again later. For example, suppose we want to make a plot of the solution. We can do this with the function `ListPlot`:

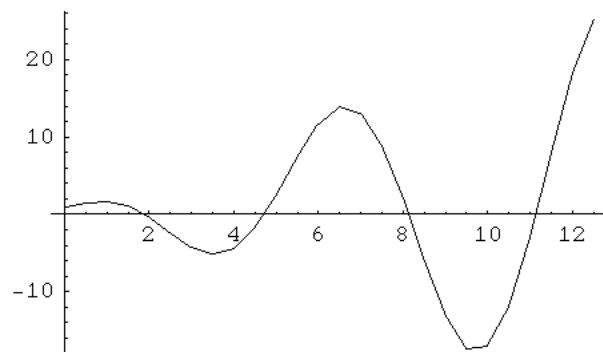
```
ListPlot[solution]
```

This plots the list of points using dots to represent each point:



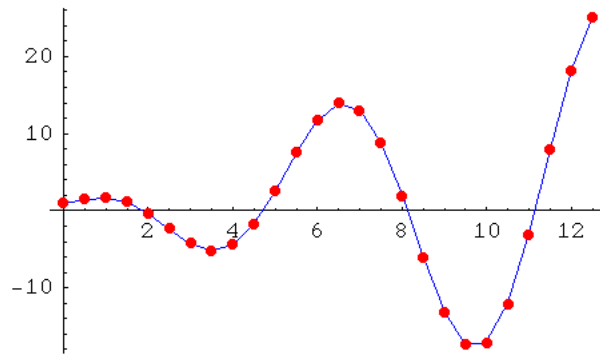
To plot the figure with the dots connected,

```
ListPlot[solution, PlotJoined->True]
```



To get a plot with a blue line and with small red dots,

```
plot1 = ListPlot[solution, PlotStyle -> Red, PointSize[.02]];
plot2 = ListPlot[solution, PlotJoined -> True, PlotStyle -> Blue];
Show[plot1, plot2];
```



Here is a more LISP-like implementation that returns the same, identical results:

```
EulersMethod2[f_, t0_, y0_, tmax_, h_] :=
  Module[{Euler},
    Euler[{x_, u_}] := {x + h, u + h * f[x, u]};
    Return[NestList[Euler, {t0, y0}, Round[tmax/h]]]
  ]
```

This makes use of the function `NestList`. `NestList` implements fixed point iteration in Mathematica. For example,

```
NestList[f, a, 3]
```

returns three fixed point iterations of the function f starting at the point a :

```
{a, f[a], f[f[a]], f[f[f[a]]]}
```

In `EulersMethod2` the fixed point iteration is being performed on the locally defined function `Euler`. `Euler` takes a pair of numbers $\{t, y\}$ and returns a new pair of numbers

$$\{t + h, y + hf(t, y)\}$$

These are same two numbers that are calculated at each step during the previous Euler's method implementation. They are then used recursively by `NestList` as the new values of the parameters x and u . The number of iterations of `NestList` is determined by rounding the ratio t_{max}/h , which might not otherwise be an integer.

Chapter 2

Successive Approximations

2.1 Picard Iteration

The Method of Successive Approximations or Picard Iteration

$$y' = f(t, y) \quad (2.1)$$

$$y(t_0) = y_0 \quad (2.2)$$

through a sequence of recursive iterations. Any function $y = \phi(t)$ that satisfies 2.1 must also solve the integral equation

$$\phi(t) = y_0 + \int_{t_0}^t f(s, \phi(s)) ds \quad (2.3)$$

The method is summarized below in Algorithm 2.1.

Algorithm 2.1. Picard Iteration *To solve the initial value problem*

$$y' = f(t, y), y(t_0) = y_0$$

for the function $y(t)$

1. input: $f(t, y), t_0, y_0, n_{max}$
2. let $\phi_0 = y_0$
3. For $i = 1, 2, \dots, n_{max}$

$$\text{let } \phi_{i+1}(t) = y_0 + \int_{t_0}^t f(s, \phi_i(s)) ds$$

4. output: $\phi_i(t)$

⇒ We will show in this chapter that when f is Lipschitz in y , algorithm 2.1 converges to the unique solution of equation 2.1. Technically speaking, however, Picard Iteration ¹ does not guarantee a solution to any specific accuracy except in the limit as $n \rightarrow \infty$. Thus it is usually quite impractical in practice. Nevertheless it has the advantage that it is easily implemented in a computer algebra system, and will sometimes yield useful results.

Example 2.1. Solve $y' = y, y(0) = 1$ using Picard Iteration.

Solution. Since $f(t, y) = y, t_0 = 0, y_0 = 1$, we have the following:

$$\phi_0 = 1 + \int_0^t ds = 1 + t \quad (2.4)$$

$$\phi_1 = 1 + \int_0^t (1 + s)ds = 1 + t + \frac{t^2}{2} \quad (2.5)$$

$$\phi_2 = 1 + \int_0^t \left(1 + s + \frac{s^2}{2}\right) ds \quad (2.6)$$

$$= 1 + t + \frac{t^2}{2} + \frac{t^3}{3!} \quad (2.7)$$

We begin to see a pattern that suggests to us that

$$\phi_n = \sum_{k=0}^{n+1} \frac{t^k}{k!} \quad (2.8)$$

We can check this out by induction. It certainly holds for $n = 1$. For the inductive step, assume equation eq:picard-ind and solve for ϕ_{n+1} :

$$\phi_{n+1} = 1 + \int_0^t \sum_{k=0}^{n+1} \frac{s^k}{k!} ds \quad (2.9)$$

$$= 1 + \sum_{k=0}^{n+1} \frac{t^{k+1}}{(k+1)!} \quad (2.10)$$

Making a change of index $j = k + 1$, we have

$$\phi_{n+1} = 1 + \sum_{j=1}^{n+2} \frac{t^j}{(j)!} = \sum_{j=0}^{n+2} \frac{t^j}{(j)!} \quad (2.11)$$

which is exactly what equation 2.8 gives for ϕ_{n+1} . Hence by the convergence theorem (Theorem 2.8), the corresponding infinite series converges to the actual solution of

¹The method bears the name of Charles Emile Picard (1856-1941), who popularized the technique, and published it in 1890, but gave credit to Hermann Schwartz. Guiseppe Peano in 1887, Ernst Leonard Lindeloff in 1890, and G. von Escherich in 1899 also published existence proofs based on this technique. Hartman claims that both Liouville and Cauchy were aware of this method. Schwartz, for his part, outlined the technique in a *Festschrift* honoring Karl Weierstrass' 70'th birthday in 1885.

the IVP:

$$\phi(t) = \sum_{k=0}^{\infty} \frac{t^k}{k!} = e^t \quad (2.12)$$

where the last step follows from Taylor's theorem. \square

Picard iteration is quite easy to implement in Mathematica; here is one possible implementation that will print out the first n iterations of the algorithm.

```
Picard[f_ ,t_, t0_, y0_, n_] :=
Module[{i, y=y0}
Print[Subscript["ϕ", 0], "=", y0];
For[i=0, i<n, i++,
  ynext=y0+∫t0t (f[s, y/.{t->s}]) ds;
  y=ynext;
  Print[Subscript["ϕ", i+1], "=", y];
];
Return[Expand[y]]
]
```

Function Picard has five arguments (f , t , t_0 , y_0 , n) and two local variables \Leftarrow (i , y)

```
Picard[f_ ,t_, t0_, y0_, n_] :=
Module[{i, y=y0},
...
]
```

The local variable y is initialized to the value of the parameter y_0 in the list of variable declarations. This is equivalent to initializing the value of the variable in the first line of the program. The first line of the program prints the initial iteration as $\phi_0 = \text{value of parameter } y_0$,

```
Print[Subscript["ϕ", 0], "=", y0];
```

The output will be displayed on the console in an “output cell.” The next line of the program is a For loop. A For statement takes on four arguments:

```
For[initialization,
  test,
  increment,
  statement;
  :
  statement;
]
```

The For loop takes the following actions:

1. The *initialization* statement (or sequence of statements) is executed;
2. The *test* is evaluated. If it evaluates to **False** then the rest of the **For** is ignored.
3. Each of the *statements* is evaluated in sequence.
4. The *increment* statement is evaluated.
5. Steps (2) through (4) are repeated until *test* is **False**.

In our program, we have a counter *i* that is initially set equal to zero; then the contents of the **For** are executed only so long as $i < n$; and the value of *i* is incremented by 1 on each iteration. Hence the loop will execute *n* times. Within the loop three statements are executed on each iteration:

```
For[i=0, i<n, i++,
  ynext=y0+∫t0t (f[s, y/.{t->s}]) ds;
  y=ynext;
  Print[Subscript["ϕ", i+1], "=", y];
];
```

There are two important variables used in this loop: *y* and *ynext*. At the start of each iteration, *y* refers to the value of the previous iteration ϕ_{i-1} , while at the end of each iteration (because of the statement **y=ynext**) it refers to the current iteration ϕ_i . In the first line of the iteration the next iteration after ϕ_{i-1} , namely, ϕ_i , is calculated and saved in *ynext*. The value depends on the integral

$$\int_{t_0}^t f(s, \phi_{i-1}(s)) ds$$

. But $\phi_{i-1}(s)$ is represented by the value of *y* at this point. Unfortunately, the expression for *y* depends upon *t*, and we need to integrate over *s* and not *t*. So to get the right variable in the expression for $f(s, \phi_{i-1}(s))$ we need to replace *t* everywhere by *s*. We do that with the expression

$$y/.\{t->s\}$$

which means, quite literally, take the expression for *y*, and everywhere that a *t* appears in it, replace the *t* with an *s*. To perform this substitution inside the integral only we do the following:

$$ynext=y0+\int_{t_0}^t (f[s, y/.\{t->s\}]) ds;$$

So then *ynext* (ϕ_i) is calculated and saved as *y*, and the results of the current iteration are printed on the console. The final line of the program returns the value of the final iteration in expanded form, namely, with all multiplications and factoring expanded out:

```
Return[Expand[y]]
```

To print the first 5 iterations of $y' = y \cos t, y(0) = 1$ using this function, one enters

```
g[tvariable_, yvariable_] := yvariable * Cos[tvariable];
Picard[g, t, 0, 1, 5];
```

which prints

$$\phi_0 = 1$$

$$\phi_1 = 1 + \sin[t]$$

$$\phi_2 = 1 + \sin[t] + \frac{1}{2} \sin[t]^2$$

$$\phi_3 = 1 + \sin[t] + \frac{1}{2} \sin[t]^2 + \frac{1}{6} \sin[t]^3$$

$$\phi_4 = 1 + \sin[t] + \frac{1}{2} \sin[t]^2 + \frac{1}{6} \sin[t]^3 + \frac{1}{24} \sin[t]^4$$

$$\phi_5 = 1 + \sin[t] + \frac{1}{2} \sin[t]^2 + \frac{1}{6} \sin[t]^3 + \frac{1}{24} \sin[t]^4 + \frac{1}{120} \sin[t]^5$$

and returns the value

$$1 + \sin[t] + \frac{1}{2} \sin[t]^2 + \frac{1}{6} \sin[t]^3 + \frac{1}{24} \sin[t]^4 + \frac{1}{120} \sin[t]^5$$

It appears that the sequence is converging to the series

$$\phi(t) = \sum_{k=0}^{\infty} \frac{\sin^k(t)}{k!} = e^{\sin t}$$

It is easily verified by separation of variables or direct substitution that this is, in fact, the correct solution.

2.2 Fixed Point Theory

The gist of the proof of Algorithm 2.1 is that it is a form of fixed point iteration. Anyone who has ever played with a pocket calculator by pressing the same button over and over again has learnt something about fixed point iteration. For example, suppose you have set your calculator to show your answers to 3 digits of accuracy to the right of the decimal point. Then type the number 16 and repeatedly press the $\sqrt{\quad}$ key. you will generate the following sequence (try it!):

16, 4, 2, 1.414, 1.189, 1.090, 1.044, 1.021, 1.010, 1.005, 1.002, 1.001, 1.000, 1.000, 1.000, ...

A second example starts with the number 1 and uses the \cos key. This time the sequence of numbers is:

1., 0.540, 0.857, 0.654, 0.793, 0.701, 0.763, 0.722, 0.750, 0.731, 0.744, ...

The same thing happens in both cases. Eventually the same number keeps repeating itself. Of course bad things can happen with this sort of recreation: try starting with 10 and using the `log` key

10, 1, 0, *ERROR*, *ERROR*, *ERROR*, ...

An even stranger thing seems to happen with the number 1 and the `tan` key – the answer does not seem to be going anywhere:

1, 1.557, 74.685, -0.863, -1.169, -2.359, 0.994, 1.538, 30.623, -1.0136, ...

What is happening here? In each of these cases we started with a number a and find sequences of numbers

$$a, \sqrt{a}, \sqrt{\sqrt{a}}, \sqrt{\sqrt{\sqrt{a}}}, \dots \quad (2.13)$$

$$a, \cos a, \cos \cos a, \cos \cos \cos a, \dots \quad (2.14)$$

The k^{th} iteration is the square root (cosine, log, tangent, etc) of the $k - 1^{\text{st}}$ iteration; namely,

$$a, g(a), g(g(a)), g(g(g(a))), \dots \quad (2.15)$$

or

$$a_k = g(a_{k-1}) \quad (2.16)$$

In the first two cases this sequence converged, to the roots of $a = \sqrt{a}$ ($a = 1$) and $a = \cos a$ ($a \approx 0.739$ Radians). But it did not converge to the root of $a = \log a$ or $a = \tan a$ in the third or fourth examples.

In Mathematica, this process is easily performed with the `NestList` functions. `NestList[g, t0, n]` returns the list of points

$$\{t0, g[t0], g[g[t0]], g[g[g[t0]]], g[g[g[g[t0]]]], \dots\}$$

where `n` is the desired number of iterations. For example

```
NestList[Sqrt, 65536, 10]
```

returns

$$\{65536, 256, 16, 4, 2, \sqrt{2}, 2^{1/4}, 2^{1/8}, 2^{1/16}, 2^{1/32}, 2^{1/64}\}$$

Definition 2.1. Fixed Point. A number a is called a fixed point of the function f if $f(a) = a$.

Example 2.2 (Finding a fixed point). Find the fixed points of the function $f(x) = x^4 + 2x^2 + x - 3$.

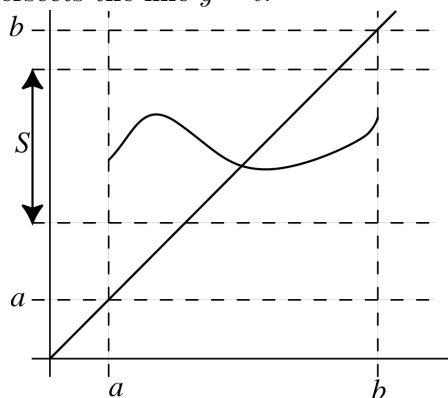
Solution.

$$\begin{aligned}x &= x^4 + 2x^2 + x - 3 \\0 &= x^4 + 2x^2 - 3 \\&= (x - 1)(x + 1)(x^2 + 3)\end{aligned}$$

Hence the real fixed points are $x = 1$ and $x = -1$. \square

A function $f : \mathbb{R} \mapsto \mathbb{R}$ has a fixed point if and only if its graph intersects with the line $y = x$. If there are multiple intersections, then there are multiple fixed points. Consequently a sufficient condition is that the range of f is contained in its domain. \Leftarrow

Figure 2.1: A sufficient condition for a bounded continuous function to have a fixed point is that the range be a subset of the domain. A fixed point occurs whenever the curve of $f(t)$ intersects the line $y = t$.



Theorem 2.1 (Sufficient condition for fixed point). *Suppose that $f(t)$ is a continuous function that maps its domain into a subset of itself, i.e.,*

$$f(t) : [a, b] \mapsto S \subset [a, b] \quad (2.17)$$

Then $f(t)$ has a fixed point in $[a, b]$.

Proof. If $f(a) = a$ or $f(b) = b$ then there is a fixed point at either a or b . So assume that both $f(a) \neq a$ and $f(b) \neq b$. By assumption, $f(t) : [a, b] \mapsto S \subset [a, b]$, so that

$$f(a) \geq a \quad \text{and} \quad f(b) \leq b \quad (2.18)$$

Since both $f(a) \neq a$ and $f(b) \neq b$, this reduces to

$$f(a) > a \quad \text{and} \quad f(b) < b \quad (2.19)$$

Let $g(t) = f(t) - t$. Then g is continuous because f is continuous. Thus \Leftarrow

$$g(a) = f(a) - a > 0 \quad (2.20)$$

$$g(b) = f(b) - b < 0 \quad (2.21)$$

Hence by the intermediate value theorem, g has a root $r \in (a, b)$, where $g(r) = 0$. \Leftarrow
Then

$$f(r) = r \quad (2.22)$$

i.e., r is a fixed point of f . \square

Review: Theorems about Functions

Theorem 2.2. Rolle's Theorem. Suppose that $f \in C[a, b]$ and $f \in \mathcal{D}(a, b)$. If $f(a) = f(b)$ then there exists some number $c \in (a, b)$ such that $f'(c) = 0$.

Theorem 2.3. Mean Value Theorem. Suppose that $f \in C[a, b]$ and $f \in \mathcal{D}(a, b)$. If $f(a) \neq f(b)$ then there exists some number $c \in (a, b)$ such that

$$f'(c) = \frac{f(b) - f(a)}{b - a} \quad (2.23)$$

Theorem 2.4. Extreme Value Theorem. If $f \in C[a, b]$ then there exist numbers $c_1, c_2 \in [a, b]$ such that $f(c_1)$ and $f(c_2)$ are the global minimum and maximum values of $f(x)$ on $[a, b]$. In other words, f has a maximum and minimum value in $[a, b]$. Furthermore, if $f \in \mathcal{D}(a, b)$ then each of c_1 and c_2 occur at either an endpoint or $[a, b]$ or where $f'(t) = 0$.

Theorem 2.5. Generalized Rolle's Theorem. Suppose that $f \in C^n[a, b]$, and that $f(t)$ has $n + 1$ distinct zeroes

$$t_0, t_1, \dots, t_n \quad (2.24)$$

in $[a, b]$. Then there is some number $c \in (a, b)$ such that $f^{(n)}(c) = 0$.

Theorem 2.6. Intermediate Value Theorem. Suppose that $f \in C[a, b]$ and that K is some number between $f(a)$ and $f(b)$. Then there exists some number $c \in [a, b]$ such that $f(c) = K$. In other words, f takes on every value between the values at the endpoints.

Theorem 2.7 (Condition for a unique fixed point). Suppose that

- (a) $f \in C[a, b]$ maps its domain into a subset of itself.
- (b) There exists some $K > 0$, $K < 1$, such that

$$|f'(t)| \leq K, \quad \forall t \in [a, b] \quad (2.25)$$

Then $f(t)$ has a unique fixed point in $[a, b]$.

Proof. By theorem 2.1 a fixed point exists. Call it p ,

$$p = f(p) \quad (2.26)$$

Suppose that a second fixed point $q \in [a, b]$, $q \neq p$ also exists, so that

$$q = f(q) \quad (2.27)$$

Hence

$$|f(p) - f(q)| = |p - q| \quad (2.28)$$

By the mean value theorem (theorem 2.3) there is some number c between p and q such that

$$f'(c) = \frac{f(p) - f(q)}{p - q} \quad (2.29)$$

Taking absolute values,

$$\left| \frac{f(p) - f(q)}{p - q} \right| = |f'(c)| \leq K < 1 \quad (2.30)$$

and thence

$$|f(p) - f(q)| < |p - q| \quad (2.31)$$

which is a contradiction. Thus our assumption that a second, different fixed point exists must be incorrect. Hence the fixed point is unique. \square

Theorem 2.8 (Convergence of fixed point iteration). *Under the same conditions as theorem 2.7 then fixed point iteration converges.*

Proof. We know from theorem 2.1 that a fixed point exists, and from theorem 2.7 that the fixed point p exists. Pick any starting point $p_0 \in [a, b]$, and generate the sequence of fixed point iterations

$$p_{i+1} = f(p_i), \quad i = 1, 2, \dots \quad (2.32)$$

We need to prove that $p_i \rightarrow p$ as $i \rightarrow \infty$.

Since f maps onto a subset of itself, every point p_i in the sequence is clearly in the interval $[a, b]$. Further, since p itself is a fixed point, \Leftarrow

$$|p_i - p| = |p_i - f(p)| = |f(p_{i-1}) - f(p)| \quad (2.33)$$

If for any value of i we have $p_i = p$ then we have reached the fixed point and the theorem is proved. So we assume that $p_i \neq p$ for all i . Then by the mean value theorem, for each value of i there exists a number c_i between p_{i-1} and p such that \Leftarrow

$$|f(p_{i-1}) - f(p)| = |f'(c_i)| |p_{i-1} - p| \leq K |p_{i-1} - p| \quad (2.34)$$

Since $f(p) = p$ and $f(p_{i-1}) = p_i$,

$$|p_i - p| \leq K |p_{i-1} - p| \quad (2.35)$$

for all i . Hence

$$|p_i - p| \leq K^2 |p_{i-2} - p| \leq K^3 |p_{i-2} - p| \leq \cdots \leq K^i |p_0 - p| \quad (2.36)$$

Since $K > 0$,

$$0 \leq \lim_{i \rightarrow \infty} |p_i - p| \leq |p_0 - p| \lim_{i \rightarrow \infty} K^i = 0 \quad (2.37)$$

Thus $p_i \rightarrow p$ as $i \rightarrow \infty$. \square

Theorem 2.9. *Under the same conditions as theorem 2.8 except that the condition of equation 2.25 is replaced with the following condition: $f(t)$ is Lipschitz with Lipschitz constant $K < 1$. Then fixed point iteration converges.*

Proof. Lipschitz gives equation 2.34. The rest of the the proof follows as before. \square

2.3 Fixed Point Iteration in a Normed Vector Space

Definition 2.2 (Vector Space). *A vector space \mathcal{V} is a set that is closed under two operations that we call addition and scalar multiplication such that the following properties hold:*

Closure *For all vectors $u, v \in \mathcal{V}$, and for all $a \in \mathbb{R}$,*

$$u + v \in \mathcal{V} \quad (2.38)$$

$$av \in \mathcal{V} \quad (2.39)$$

Commutivity of Vector Addition *For all $u, v \in \mathcal{V}$,*

$$u + v = v + u \quad (2.40)$$

Associativity of Vector Addition *For all $u, v, w \in \mathcal{V}$,*

$$u + (v + w) = (u + v) + w \quad (2.41)$$

Identity for Addition *There is some element $0 \in \mathcal{V}$ such that for all $v \in \mathcal{V}$*

$$0 + v = v + 0 = v \quad (2.42)$$

Inverse for Addition *For each $v \in \mathcal{V}$ there is a vector $-v \in \mathcal{V}$ such that*

$$v + (-v) = (-v) + v = 0 \quad (2.43)$$

Associativity of Scalar multiplication *For all $v \in \mathcal{V}$ and for all $a, b \in \mathbb{R}$,*

$$a(bv) = (ab)v \quad (2.44)$$

Distributivity For all $a, b \in \mathbb{R}$ and for all $u, v \in \mathcal{V}$,

$$(a + b)v = av + bv \quad (2.45)$$

$$a(u + v) = au + av \quad (2.46)$$

Identity for Scalar Multiplication For all vectors $v \in \mathcal{V}$,

$$1v = v \quad (2.47)$$

Example 2.3. The usual Cartesian vector space to which we are accustomed is a vector space with vectors being defined as ordered triples of coordinates $\langle x, y, z \rangle$.

Example 2.4. Show that the set $\mathcal{F}[a, b]$ of all integrable functions $f : [a, b] \mapsto \mathbb{R}$ is a vector space.

Solution. Let $f, g, h \in \mathcal{F}[a, b]$ and $c, d \in \mathbb{R}$ Then

- \mathcal{V} is closed: Let $p(t) = f(t) + g(t)$ and $q(t) = ch(t)$. Then $p, q : [a, b] \mapsto \mathbb{R}$ hence $p, q \in \mathcal{F}[a, b]$
- $f(t) + g(t) = g(t) + f(t)$ so commutivity holds.
- $(f(t) + g(t)) + h(t) = f(t) + (g(t) + h(t))$ and $c(df(t)) = (cd)f(t)$ so both associative properties hold.
- The function $f(t) = 0$ is an additive identity.
- For any function $f(t)$ the function $-f(t)$ is an additive inverse.
- $(c+d)f(t) = cf(t) + df(t)$ and $c(f(t) + g(t)) = cf(t) + cg(t)$ so both distributive properties hold.
- The number 1 acts as an identity for scalar multiplication.

Hence the set $\mathcal{F}[a, b]$ is a vector space. □

Definition 2.3 (Norm). A norm $\| \cdot \| : \mathcal{V} \mapsto \mathbb{R}$ on a vector space \mathcal{V} is a function mapping the vector space to the real numbers such that

1. For all $v \in \mathcal{V}$, $\|v\| \geq 0$.
2. $\|v\| = 0$ if and only if $v = 0$.
3. For all $v \in \mathcal{V}$ and for all $a \in \mathbb{R}$, $\|av\| = |a| \|v\|$.
4. The norm satisfies a **triangle inequality**: for all $v, w \in \mathcal{V}$,

$$\|v + w\| \leq \|v\| + \|w\| \quad (2.48)$$

Definition 2.4 (Normed Vector Space). A vector space on which a norm has been defined is a normed vector space.

Example 2.5. Let \mathcal{V} be ordinary Euclidean space and $v = \langle x, y, z \rangle$ a vector in \mathcal{V} . Then we can define many different norms on this space:

Taxicab (Manhattan, City Block) Norm The L_1 norm is: $\|v\|_1 = |x| + |y| + |z|$

Euclidean Distance Function The L_2 norm is: $\|v\|_2 = \sqrt{x^2 + y^2 + z^2}$

p-norm The L_p norm is: $\|v\|_p = (x^p + y^p + z^p)^{1/p}$ for $p \in \mathbb{Z}^+$.

sup-norm The L_∞ norm is $\|v\|_\infty = \sup(|x|, |y|, |z|)$

Example 2.6. The following norms can be defined on the vector space $\mathcal{F}[a, b]$ of integrable functions on $[a, b]$:

$$\text{L}_2\text{-norm: } \|f\|_2 = \sqrt{\int_a^b |f(x)|^2 dx}$$

$$\text{L}_p\text{-norm: } \|f\|_p = \left(\int_a^b |f(x)|^p dx \right)^{1/p}$$

$$\text{L}_\infty \text{ or sup-norm: } \|f\| = \sup_{x \in [a, b]} |f(x)|$$

Definition 2.5 (Contraction). Let \mathcal{V} be a normed vector space, $S \subset \mathcal{V}$. Then a **contraction** is any mapping $T : S \mapsto \mathcal{V}$ that satisfies

$$\|T(f) - T(g)\| \leq K \|f - g\| \quad (2.49)$$

\implies form some $K \in \mathbb{R}$, $0 < K < 1$, for all $f, g \in S$. We will call the number K the **contraction constant**.

Lemma 2.1. Let T be a contraction on a complete normed vector space \mathcal{V} with contraction constant K . Then for any $g \in \mathcal{V}$

$$\|T^n g - g\| \leq \frac{1 - K^n}{1 - K} \|Tg - g\| \quad (2.50)$$

Proof. Use induction. For $n = 1$, we have

$$\|Tg - g\| \leq \frac{1 - K}{1 - K} \|Tg - g\| \quad (2.51)$$

As our inductive hypothesis choose any $n > 1$ and suppose that equation 2.50 holds. Then by the triangle inequality

$$\|T^{n+1}g - g\| \leq \|T^{n+1}g - T^n g\| + \|T^n g - g\| \quad (2.52)$$

$$\leq \|T^{n+1}g - T^n g\| + \frac{1 - K^n}{1 - K} \|Tg - g\| \quad (2.53)$$

$$\leq K^n \|Tg - g\| + \frac{1 - K^n}{1 - K} \|Tg - g\| \quad (2.54)$$

$$= \frac{(1 - K)K^n + (1 - K^n)}{1 - K} \|Tg - g\| \quad (2.55)$$

$$= \frac{1 - K^{n+1}}{1 - K} \|Tg - g\| \quad (2.56)$$

which proves the conjecture for $n + 1$. \square

Theorem 2.10 (Contraction Mapping Theorem²). *Let T be a contraction on a normed vector space \mathcal{V} . Then T has a unique fixed point $h \in \mathcal{V}$ such that $T(h) = h$. Furthermore, any sequence of functions g_1, g_2, \dots defined by $g_k = Tg_{k-1}$ converges to the unique fixed point $Tg = g$. We denote this by $g_k \rightarrow g$.* ←

*Proof.*³ Let $\epsilon > 0$ be given. Then since $K^n/(1-K) \rightarrow 0$ as $n \rightarrow \infty$ (because T is a contraction, $K < 1$) it is possible to choose an integer N such that

$$\frac{K^n \|Tg - g\|}{1 - K} < \epsilon \quad (2.57)$$

Pick any two integers $m \geq n \geq N$, and define the sequence $g_0 = g, g_n = Tg_{n-1}$. Then

$$\|g_m - g_n\| = \|T^m g - T^n g\| \quad (2.58)$$

$$\leq K^n \|T^{m-n} g - g\| \quad (2.59)$$

$$\leq K^n \frac{1 - K^{m-n}}{1 - K} \|Tg - g\| \quad (2.60)$$

by Lemma 2.1. Hence

$$\|g_m - g_n\| \leq \frac{K^n - K^m}{1 - K} \|Tg - g\| \leq \frac{K^n}{1 - K} \|Tg - g\| < \epsilon \quad (2.61)$$

Therefore g_n is a Cauchy sequence, and every Cauchy sequence on a complete normed vector space converges. Define $f = \lim_{n \rightarrow \infty} g_n$. Then either f is a fixed point of T or it is not a fixed point of T . Suppose that it is not a fixed point of T . Then $Tf \neq f$ and hence there exists some $\delta > 0$ such that

$$\|Tf - f\| > \delta \quad (2.62)$$

On the other hand, because $g_n \rightarrow f$, there exists an integer N such that for all $n > N$,

$$\|g_n - f\| < \delta/2 \quad (2.63)$$

Hence ←

$$\|Tf - f\| \leq \|Tf - g_{n+1}\| + \|f - g_{n+1}\| \quad (2.64)$$

$$\leq K \|f - g_n\| + \|f - g_{n+1}\| \quad (2.65)$$

$$\leq \|f - g_n\| + \|f - g_{n+1}\| \quad (2.66)$$

$$= 2\|f - g_n\| \quad (2.67)$$

$$< \delta \quad (2.68)$$

This is a contradiction, and hence f must be a fixed point of T .

²The contraction mapping theorem is sometimes called the Banach Fixed Point Theorem

³The proof follows "Proof of Banach Fixed Point Theorem," *Encyclopedia of Mathematics* (Volume 2, 54A20:2034), PlanetMath.org.

To prove uniqueness, suppose that there is another fixed point $h \neq f$. Then $\|h - f\| > 0$ (otherwise they are equal). But

$$\|h - f\| = \|Th - Tf\| \quad (2.69)$$

$$\leq K\|h - f\| \quad (2.70)$$

$$< \|h - f\| \quad (2.71)$$

which is impossible and hence a contradiction. Thus f is the unique fixed point of T . \square

2.4 Convergence of Successive Approximations

We restate the fundamental existence theorem here for reference. While it is stated in terms of the scalar problem, the vector problem is not fundamentally different, and the proof is completely analogous.

Theorem 2.11 (Fundamental Existence Theorem). *Let $D \in \mathbb{R}^2$ be convex and suppose that f is continuously differentiable on D . Then the initial value problem*

$$y' = f(t, y), \quad y(t_0) = y_0 \quad (2.72)$$

has a unique solution $\phi(t)$ in the sense that $\phi'(t) = f(t, \phi(t))$, $\phi(t_0) = y_0$.

Proof. We begin by observing that ϕ is a solution of equation 2.72 if and only if it is a solution of

$$\phi(t) = y_0 + \int_{t_0}^t f(x, \phi(x)) dx \quad (2.73)$$

Our goal will be to prove 2.73.

Let S be the set of all continuous integrable functions on an interval (a, b) that contains t_0 . Corresponding to any function $\phi \in S$ we can define the mapping $T: S \mapsto S$ as

$$T[\phi] = y_0 + \int_{t_0}^t f(x, \phi(x)) dx \quad (2.74)$$

We will assume $t > t_0$. The proof for $t < t_0$ is completely analogous. Using the sup-norm on (a, b) , we calculate that for any two functions $g, h \in S$,

$$\|T[g] - T[h]\| = \left\| \int_{t_0}^t [f(x, g(x)) - f(x, h(x))] dx \right\| \quad (2.75)$$

$$\leq \sup_{a \leq t \leq b} \left| \int_{t_0}^t [f(x, g(x)) - f(x, h(x))] dx \right| \quad (2.76)$$

Since f is differentiable it is Lipschitz in its second argument, hence

$$\|T[g] - T[h]\| \leq K \sup_{a \leq t \leq b} \int_{t_0}^t |g(x) - h(x)| dx \quad (2.77)$$

$$\leq K(b - a) \sup_{a \leq t \leq b} |g(x) - h(x)| \quad (2.78)$$

$$\leq K(b - a) \|g - h\| \quad (2.79)$$

Where K is any number larger than $\sup_{(a,b)} f_y$. If we choose the endpoints a and b such that $|b - a| < 1/K$ we have $K|b - a| < 1$. Thus T is a contraction. By the contraction mapping theorem it has a fixed point; call this point ϕ . Equation 2.73 follows immediately. \square

Theorem 2.12 (Error Bounds on Picard Iterations). *Under the same conditions as before, let ϕ_n be the n^{th} Picard iterate, and let ϕ be the solution of the IVP. Then*

$$|\phi(t) - \phi_n(t)| \leq \frac{M |K(t - t_0)|^{n+1}}{K(n+1)!} e^{K|t-t_0|} \quad (2.80)$$

where $M = \sup_D |f(t, y)|$ and K is a Lipschitz constant. Furthermore, if $L = |b - a|$ then

$$\|\phi(t) - \phi_n(t)\| \leq \frac{M [KL]^{n+1} e^{KL}}{K(n+1)!} \quad (2.81)$$

where $\|\cdot\|$ denotes the sup-norm.

Proof. We begin by proving the conjecture

$$|\phi_n - \phi_{n-1}| \leq \frac{K^{n-1} M}{n!} |t - t_0|^n \quad (2.82)$$

For $n = 1$, equation 2.82 says that

$$|\phi_1 - y_0| \leq M|t - t_0| \quad (2.83)$$

which follows immediately from equation 2.73. Next, make the inductive hypothesis 2.82 and calculate

$$|\phi_{n+1} - \phi_n| = \left| \int_{t_0}^t [f(s, \phi_n(s)) - f(s, \phi_{n-1}(s))] ds \right| \quad (2.84)$$

$$\leq K \int_{t_0}^t |\phi_n(s) - \phi_{n-1}(s)| ds \quad (2.85)$$

by the definition of ϕ_n and the Lipschitz condition. Applying the inductive hypothesis and then integrating,

$$|\phi_{n+1} - \phi_n| \leq \frac{K^n M}{n!} \int_{t_0}^t |s - t_0|^n ds \quad (2.86)$$

$$\leq \frac{K^n M}{(n+1)!} |t - t_0|^{n+1} \quad (2.87)$$

which proves conjecture 2.82. Now let

$$\phi_n(t) = \phi_0(t) + \sum_{i=1}^n [\phi_i(t) - \phi_{i-1}(t)] \quad (2.88)$$

Then since the sequence of Picard iterates converges to the solution,

$$\phi(t) = \lim_{n \rightarrow \infty} \phi_n(t) = \phi_0(t) + \sum_{i=1}^{\infty} [\phi_i(t) - \phi_{i-1}(t)] \quad (2.89)$$

Hence by equation 2.82,

$$|\phi(t) - \phi_n(t)| = \left| \sum_{i=n+1}^{\infty} (\phi_i(t) - \phi_{i-1}(t)) \right| \quad (2.90)$$

$$\leq \sum_{i=n+1}^{\infty} |\phi_i(t) - \phi_{i-1}(t)| \quad (2.91)$$

$$\leq \sum_{i=n+1}^{\infty} \frac{K^{i-1}M}{i!} |t - t_0|^i \quad (2.92)$$

$$\leq \frac{M}{K} \sum_{i=n+1}^{\infty} \frac{|K(t - t_0)|^i}{i!} \quad (2.93)$$

Therefore by comparison with a Taylor series for $e^{K(b-a)}$,

$$\|\phi(t) - \phi_n(t)\| \leq \frac{M}{K} \sum_{i=n+1}^{\infty} \frac{|K(b-a)|^i}{i!} \quad (2.94)$$

$$\leq \frac{M}{K} \left(e^{K(b-a)} - \sum_{i=0}^n \frac{|K(b-a)|^i}{i!} \right) \quad (2.95)$$

$$\leq \frac{M}{K} \sup_{0 \leq t \leq KL} R_n(t) \quad (2.96)$$

where $R_n(t)$ is the Taylor Series Remainder for e^t after n terms,

$$\sup_{0 \leq t \leq KL} R_n(t) \leq \sup_{0 \leq \{c,t\} \leq KL} \frac{t^{n+1}e^c}{(n+1)!} \leq \frac{(KL)^{n+1}e^{KL}}{(n+1)!} \quad (2.97)$$

for some unknown c between a and b . Hence

$$\|\phi(t) - \phi_n(t)\| \leq \frac{M (KL)^{n+1} e^{KL}}{K (n+1)!} \quad (2.98)$$

proving the theorem. \square

The following example shows that this bounds is not very useful in practice.

Example 2.7. Estimate the number of iterations required to obtain an solution to $y' = t, y(0) = 1$ on $[0, 10]$ with a precision of no more than 10^{-7} .

Solution. Since $f(t, y) = t$ we have $f_y = 0$ and hence a Lipschitz constant is $K = 1$ (or any positive number), and we can use $M = 10$ on $[0, 10]$. The precision in the error is bounded by

$$\epsilon \leq \frac{M(KL)^{n+1}e^{KL}}{K(n+1)!} \leq \frac{10(10)^{n+1}e^{10}}{(n+1)!} \quad (2.99)$$

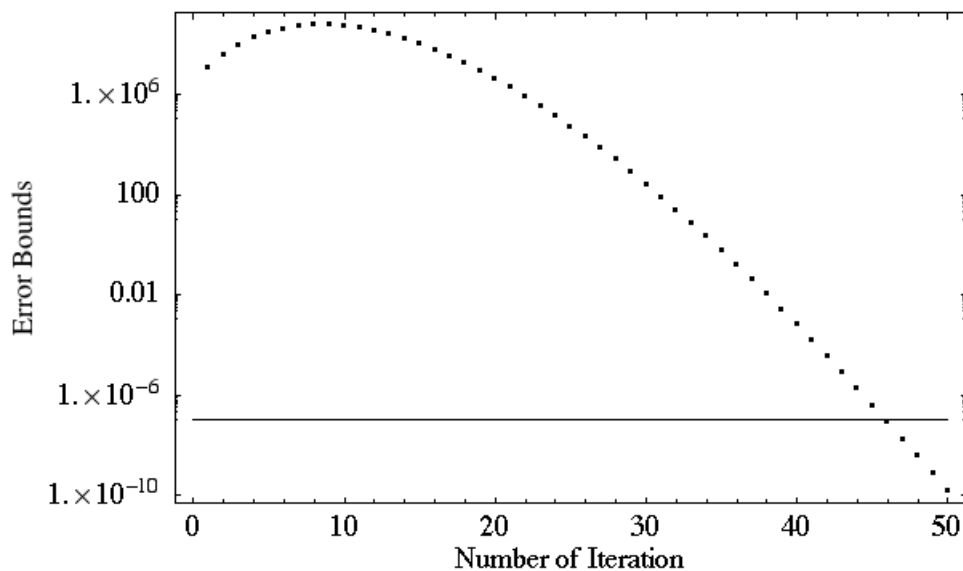
We can determine the minimum value of n by using Mathematica. The following will print a list of values of 2.99 for n ranging from 1 to 50.

```
errs = Table[{n, 10 (10) ^ (n + 1) (E ^ 10.)/(n + 1)!}, {n, 1,
50}]
```

The output is a list of number pairs, which can be plotted with `ListPlot` or

```
<<'Graphics'Graphics'
LogListPlot[errs];
```

The output of `LogListPlot` is shown below; we have annotated the plot with an additional line at the desired tolerance of 10^{-7} showing that it occurs at the 47th iteration.



This example shows that Picard iteration will produce the desired accuracy if we perform 47 iterations. For this particular problem, this suggestion is absurd, because Picard iteration converges to the exact solution after 1 iteration. The calculated solution does not change upon further calculations. Hence the method vastly over-estimates the potential error (at least for this example).

Chapter 3

Approximate Solutions

3.1 The Forward Euler Method

By a *numerical solution* of the initial value problem

$$y' = f(t, y), \quad y(t_0) = y_0 \quad (3.1)$$

we mean a sequence of values

$$y_0, y_1, y_2, \dots, y_{n-1}, y_n; \quad (3.2)$$

a corresponding *mesh* or *grid* \mathcal{M} by

$$\mathcal{M} = \{t_0 < t_1 < t_2 < \dots < t_{n-1} < t_n\}; \quad (3.3)$$

and a *grid spacing* as

$$h_j = t_{j+1} - t_j \quad (3.4)$$

Then the *numerical solution* or *numerical approximation to the solution* is the sequence of points

$$(t_0, y_0), (t_1, y_1), \dots, (t_{n-1}, y_{n-1}), (t_n, y_n) \quad (3.5)$$

In this solution the point (t_j, y_j) represents the numerical approximation to the solution point $y(t_j)$. We can imagine plotting the points (3.5) and then “connecting the dots” to represent an approximate image of the graph of $y(t), t_0 \leq t \leq t_n$. We will use the convenient notation

$$y_n \approx y(t_n) \quad (3.6)$$

which is read as “ y_n is the numerical approximation to $y(t)$ at $t = t_n$.”

Euler’s Method is constructed as follows. At grid point t_n , $y(t) \approx y_n$, and the slope of the solution is given by exactly $y' = f(t_n, y(t_n))$. If we approximate the slope by the straight line segment between the numerical solution at t_n and the numerical solution at t_{n+1} then

$$y'_n(t_n) \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n} = \frac{y_{n+1} - y_n}{h_n} \quad (3.7)$$

Since $y'(t) = f(t, y)$, we can approximate the left hand side of (3.7) by

$$y'_n(t_n) \approx f(t_n, y_n) \quad (3.8)$$

and hence

$$\boxed{y_{n+1} = y_n + h_n f(t_n, y_n)} \quad (3.9)$$

which is the iteration formula for the *Forward Euler Method*.

It is often the case that we use a fixed *step size* $h = t_{j+1} - t_j$, in which case we have

$$t_j = t_0 + jh \quad (3.10)$$

In this case the Forward Euler's method becomes

$$y_{n+1} = y_n + hf(t_n, y_n) \quad (3.11)$$

The Forward Euler's method is sometimes just called *Euler's Method*. The application of Euler's method is summarized in Algorithm 4.2.

An alternate derivation of equation (3.9) is to expand the solution $y(t)$ in a Taylor Series about the point $t = t_n$:

$$y(t_{n+1}) = y(t_n + h_n) = y(t_n) + h_n y'(t_n) + \frac{h_n^2}{2} y''(t_n) + \cdots \quad (3.12)$$

$$= y(t_n) + h_n f(t_n, y(t_n)) + \cdots \quad (3.13)$$

We then observe that since $y_n \approx y(t_n)$ and $y_{n+1} \approx y(t_{n+1})$, then (3.9) follows immediately from (3.13).

If the scalar initial value problem of equation (3.1) is replaced by a systems of equations

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad (3.14)$$

then the Forward Euler's Method has the obvious generalization

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n) \quad (3.15)$$

Algorithm 3.1. Forward Euler Method *To solve the initial value problem*

$$y' = f(t, y), y(t_0) = y_0$$

on an interval $[t_0, t_{max}]$ with a fixed step size h .

1. input: $f(t, y), t_0, y_0, h, t_{max}$
2. output: (t_0, y_0)
3. let $t = t_0, y = y_0$
4. while $t < t_{max}$
 - (a) let $y = y + hf(t, y)$
 - (b) let $t = t + h$
 - (c) let $t_n = t, y_n = y$
 - (d) output: (t_n, y_n)

Example 3.1. *Solve $y' = y, y(0) = 1$ on the interval $[0, 1]$ using $h = 0.25$.*

Solution. The exact solution is $y = e^x$. We compute the values using Euler's method. For any given time point t_k , the value y_k depends purely on the values of t_{k-1} and y_{k-1} . This is often a source of confusion for students: although the formula $y_{k+1} = y_k + hf(t_k, y_k)$ only depends on t_k and not on t_{k+1} it gives the value of y_{k+1} . We are given the following information:

$$(t_0, y_0) = (0, 1) \tag{3.16}$$

$$f(t, y) = y \tag{3.17}$$

$$h = 0.25 \tag{3.18}$$

We first compute the solution at $t = t_1$.

$$y_1 = y_0 + hf(t_0, y_0) = 1 + (0.25)(1) = 1.25 \tag{3.19}$$

$$t_1 = t_0 + h = 0 + 0.25 = 0.25 \tag{3.20}$$

$$(t_1, y_1) = (0.25, 1.25) \tag{3.21}$$

Then we compute the solutions at $t = t_1, t_2, \dots$ until $t_{k+1} = 1$.

$$y_2 = y_1 + hf(t_1, y_1) \tag{3.22}$$

$$= 1.25 + (0.25)(1.25) = 1.5625 \tag{3.23}$$

$$t_2 = t_1 + h = 0.25 + 0.25 = 0.5 \tag{3.24}$$

$$(t_2, y_2) = (0.5, 1.5625) \tag{3.25}$$

$$y_3 = y_2 + hf(t_2, y_2) \quad (3.26)$$

$$= 1.5625 + (0.25)(1.5625) = 1.953125 \quad (3.27)$$

$$t_3 = t_2 + h = 0.5 + 0.25 = 0.75 \quad (3.28)$$

$$(t_3, y_3) = (0.75, 1.953125) \quad (3.29)$$

$$y_4 = y_3 + hf(t_3, y_3) \quad (3.30)$$

$$= 1.953125 + (0.25)(1.953125) = 2.44140625 \quad (3.31)$$

$$t_4 = t_3 + 0.25 = 1.0 \quad (3.32)$$

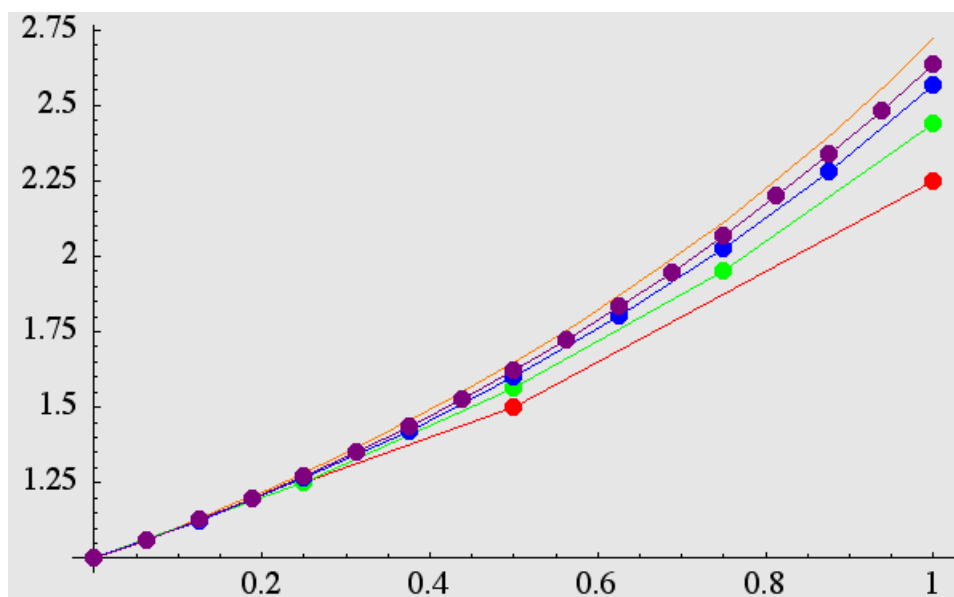
$$(t_4, y_4) = (1.0, 2.44140625) \quad (3.33)$$

Since $t_4 = 1$ we are done. The solutions are tabulated in table 3.1 for this and other step sizes. These solutions are also illustrated in figure 3.1. The green dots in the figure correspond to the numbers calculated here; the other colors correspond to other step sizes. The points are joined by straight lines in the figure, but the straight lines are not part of the solution, only the dots. The figure also shows the exact solution. \square

Table 3.1: Approximate values for different step sizes. See example 3.1.

t	$h = 1/2$	$h = 1/4$	$h = 1/8$	$h = 1/16$	exact solution
0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.0625				1.0625	1.0645
0.1250			1.1250	1.1289	1.1331
0.1875				1.1995	1.2062
0.2500		1.2500	1.2656	1.2744	1.2840
0.3125				1.3541	1.3668
0.3750			1.4238	1.4387	1.4550
0.4375				1.5286	1.5488
0.5000	1.5000	1.5625	1.6018	1.6242	1.6487
0.5625				1.7257	1.7551
0.6250			1.8020	1.8335	1.8682
0.6875				1.9481	1.9887
0.7500		1.9531	2.0273	2.0699	2.1170
0.8125				2.1993	2.2535
0.8750			2.2807	2.3367	2.3989
0.9375				2.4828	2.5536
1.0000	2.2500	2.4414	2.5658	2.6379	2.7183

Figure 3.1: Solution curves for example 3.1. The curves correspond to $h = 1$ (red); $h = 1/2$ (green); $h = 1/4$ (blue); $h = 1/8$ (purple); and the exact solution (orange).



Taylor Series

Theorem 3.1. Taylor's Theorem with Remainder. Let $f : J \mapsto \mathbb{R}$ be C^{n+1} on $J \subset \mathbb{R}$, and suppose that $a \in \text{interior}(J)$. Then for any $x \in J$, there exists some number $c \in J$, where c is between a and x , inclusive, such that

$$f(x) = P_n(x) + R_n(x) \quad (3.34)$$

where the **The Taylor Polynomial of order n for f about $x=a$** is

$$P_n(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2 f''(a)}{2!} + \dots + \frac{(x-a)^n f^{(n)}(a)}{n!} \quad (3.35)$$

and

$$R_n(x) = \frac{(x-a)^{n+1} f^{(n+1)}(c)}{(n+1)!} \quad (3.36)$$

Theorem 3.2. If, in addition, $f \in C^\infty$ then

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)(x-a)^k}{k!} \quad (3.37)$$

which is called the **Taylor Series of f about $x=a$.**

3.2 Epsilon-Approximate Solutions

To prove that a Euler's method solution can actually be constructed we need to formalize the definition of an approximate solution a little.¹

Definition 3.1. Let $f(t, y) \in C^0$ on some domain $D \subset \mathbb{R}^2$ that contains the rectangle

$$R = [t_0 - a, t_0 + a] \times [y_0 - b, y_0 + b] \quad (3.38)$$

Then $y(t)$ is an ϵ -**approximate solution of the initial value problem 3.1** if all of the following conditions are true:

1. $y(t)$ is admissible on D , i.e., $f(t, y(t)) \in D$ for all $t \in [t_0 - a, t_0 + a]$;
2. $y(t)$ is continuous on $[t_0 - a, t_0 + a]$;
3. $y'(t)$ exists except at finitely many points t_1, t_2, \dots, t_n and is continuous on $t_i < t < t_{i+1}$ for all i ;
4. $|y'(t) - f(t, y(t))| \leq \epsilon$ for all $t \in [t_0 - a, t_0 + a]$ except possibly at t_1, t_2, \dots

Theorem 3.3. Let (t_0, y_0) , R , D and f be defined as in definition (3.1); furthermore, suppose that f is bounded on R , namely that there exists some $M > 0$ such that

$$|f(t, y)| \leq M \quad (3.39)$$

for all $(t, y) \in R$; and define

$$h = \min(a, b/M) \quad (3.40)$$

Then it is possible to construct an ϵ -approximate solution to (3.1).

Proof. We will construct the solution going to the “right”, namely, for $t > t_0$. The proof for the left-side solution is analogous.

Define the rectangle $S \subset R$ by

$$S = [t_0 - h, t_0 + h] \times [y_0 - Mh, y_0 + Mh] \quad (3.41)$$

Since

$$h = \min(a, b/M) \leq a \quad (3.42)$$

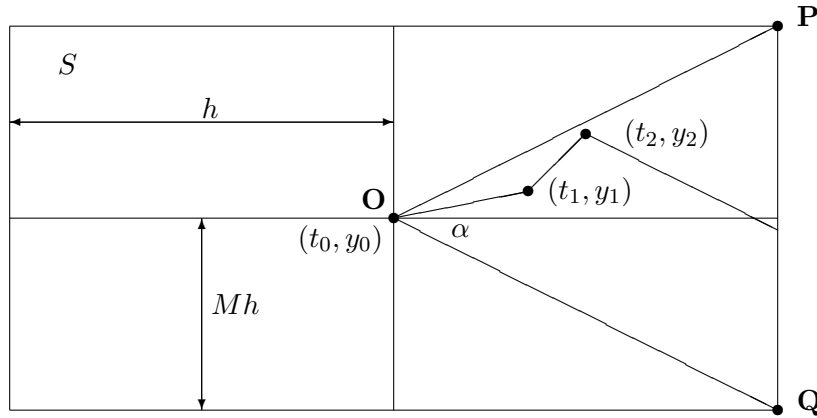
$$Mh = \min(Ma, b) \leq b \quad (3.43)$$

then $S \subset R$. Hence all of the properties of f that hold on R also hold on S .

Let $\epsilon > 0$ be given. Then since $f(t, y)$ is continuous, it is uniformly continuous \implies on S , and therefore there exists some $\delta > 0$ such that

¹This discussion parallels W. Hurewicz (1958) *Lectures on Ordinary Differential Equations*, MIT Press.

Figure 3.2: Construction of the approximate solution.



$$|\hat{t} - t|, |\hat{y} - y| < \delta \implies |f(\hat{t}, \hat{y}) - f(t, y)| \leq \epsilon \quad (3.44)$$

$$\forall(\hat{t}, \hat{y}), (t, y) \in S.$$

Pick any meshing satisfying

$$t_0 < t_1 < \dots < t_{n-1} < t_n = t_0 + h \quad (3.45)$$

$$t_i - t_{i-1} \leq \min(\delta, \delta/M), \quad i = 1, 2, \dots, n \quad (3.46)$$

To simplify the notation we define the points

$$\mathbf{O} = (t_0, y_0) \quad (3.47)$$

$$\mathbf{P} = (t_0 + h, y_0 + Mh) \quad (\text{upper right hand corner of } S) \quad (3.48)$$

$$\mathbf{Q} = (t_0 + h, y_0 - Mh) \quad (\text{lower right hand corner of } S) \quad (3.49)$$

Construct the solution by drawing a line segment from (t_0, y_0) with slope $f(t_0, y_0)$ until it intersects the line $t = t_1$. The end point then has coordinates (t_1, y_1) where

$$\frac{y_1 - y_0}{t_1 - t_0} = f(t_0, y_0) \quad (3.50)$$

$$y_1 = y_0 + (t_1 - t_0)f(t_0, y_0) \quad (3.51)$$

The reader should compare the construction of y_1 in equation(3.51) with equation(3.9); the step size h_n in (3.9) corresponds to the time-step $t_1 - t_0$ in (3.51). So we are constructing the first segment of the solution using the Forward Euler Method.

We then continue to construct additional points on the solution at subsequent steps,

e.g.,

$$y_2 = y_1 + (t_2 - t_1)f(t_1, y_1) \quad (3.52)$$

$$y_2 = y_1 + (t_2 - t_1)f(t_1, y_1) \quad (3.53)$$

$$\vdots \quad (3.54)$$

$$y_{k+1} = y_k + (t_{k+1} - t_k)f(t_k, y_k) \quad (3.55)$$

$$\vdots \quad (3.56)$$

From figure (3.2) we observe that $\tan \alpha = M$; hence, since f is bounded,

$$|f(t_1, y_1)| \leq M = \tan \alpha \quad (3.57)$$

and therefore

$$|y_1 - y_0| \leq |t_1 - t_0||f(t_0, y_0)| \leq |t_1 - t_0| \tan \alpha \quad (3.58)$$

Thus the point (t_1, y_1) is inside of the triangle **OPQ**. By a similar argument, each of $(t_2, y_2), (t_3, y_3), \dots$ are also in the triangle **OPQ**.

The curve we have just drawn is admissible, continuous, and piecewise differentiable, with derivative given by

$$y'(t) = f(t_{k-1}, y(t_{k-1})), \quad t_{k-1} < x < t_k, \quad k = 1, 2, \dots \quad (3.59)$$

Subtracting $f(t, y(t))$ from both sides of the equation and taking absolute values,

$$|y'(t) - f(t, y(t))| = |f(t_{k-1}, y(t_{k-1})) - f(t, y(t))| \quad (3.60)$$

By equations (3.46), (3.55) and the boundedness of f ,

$$|y_{k+1} - y_k| = |(t_{k+1} - t_k)f(t_k, y_k)| \quad (3.61)$$

$$\leq \min(\delta, \delta/M)M \quad (3.62)$$

$$\leq \delta \quad (3.63)$$

Hence by uniform continuity (equation (3.44)), because $|y_{k+1} - y_k| < \delta$,

$$|f(t_{k-1}, y(t_{k-1})) - f(t, y(t))| \leq \epsilon \quad (3.64)$$

From equation (3.60)

$$|y'(t) - f(t, y(t))| \leq \epsilon \quad (3.65)$$

which holds true for all $t \neq t_k, k = 1, 2, \dots, n - 1$.

Therefore the function we have constructed satisfies all of the conditions of definition 3.1, hence it is an ϵ -approximate solution. Since we have constructed an ϵ -approximate solution, we conclude that an ϵ -approximate solution exists. \square

3.3 The Fundamental Inequality

We can prove that the Epsilon-approximate solution converges to an exact solution of the differential equation, giving us a proof of the fundamental existence theorem. This proof will be presented in the following section. Here we will present a basic inequality that is often used in the theory of differential equations.

Lemma 3.1. *Suppose that $\partial f/\partial y$ is bounded for all $(t, y) \in D$. Then $f(t, y)$ is Lipschitz on D with Lipschitz Constant*

$$K = \sup \left| \frac{\partial f}{\partial y} \right| \quad (3.66)$$

Proof. By the Mean Value theorem there exists some number c between between y_1 and y_2 such that

$$f(t, y_2) - f(t, y_1) = (y_2 - y_1) \frac{\partial f(t, c)}{\partial y} \quad (3.67)$$

Since

$$\left| \frac{\partial f(t, c)}{\partial y} \right| \leq K \quad (3.68)$$

we have

$$|f(t, y_2) - f(t, y_1)| \leq K|y_2 - y_1| \quad (3.69)$$

and thus the function is Lipschitz in y . \square

Lemma 3.2. *Suppose D is not convex, but can be embedded in a convex domain D' . Let $\delta > 0$ be the shortest distance between the boundaries of D and D' . Suppose that f is continuous and bounded by M in D , and that $\partial f/\partial y$ exists and is bounded by N in D' then $f(t, y)$ is Lipschitz in y in D with Lipschitz Constant*

$$K = \max \left(N, \frac{2M}{\delta} \right) \quad (3.70)$$

Proof. Let $\mathbf{P} = (t, y_1)$, $\mathbf{Q} = (t, y_2) \in D$.

Then if $|y_1 - y_2| > \delta$,

$$\left| \frac{f(t, y_1) - f(t, y_2)}{y_1 - y_2} \right| \leq \frac{|f(t, y_1)| + |f(t, y_2)|}{\min |y_1 - y_2|} = \frac{2M}{\delta} \quad (3.71)$$

thus

$$|f(t, y_1) - f(t, y_2)| \leq \frac{2M}{\delta} |f(t, y_1) - f(t, y_2)| \quad (3.72)$$

$$\leq \max \left(N, \frac{2M}{\delta} \right) |f(t, y_1) - f(t, y_2)| \quad (3.73)$$

$$\leq K |f(t, y_1) - f(t, y_2)| \quad (3.74)$$

Alternatively, if $|y_1 - y_2| \leq \delta$, the line segment \mathbf{PQ} still lies entirely in D' . But since D' is convex, we can apply lemma 3.1 to conclude that N is a Lipschitz constant. Hence the Lemma is proved. \square

Theorem 3.4. Fundamental Inequality. *Suppose that f is continuous and Lipschitz (with constant K) on a set R that includes the point (t_0, y_0) . Let $h > 0$, and suppose that $y_{(1)}, y_{(2)}$ are admissible ϵ_1 - and ϵ_2 -approximations on $|t - t_0| < h$. Then*

$$\boxed{|y_{(2)}(t) - y_{(1)}(t)| \leq e^{K|t-t_0|} |y_{(2)}(t_0) - y_{(1)}(t_0)| + \frac{\epsilon_1 + \epsilon_2}{K} \left(e^{K|t-t_0|} - 1 \right)} \quad (3.75)$$

Proof. Since $y_{(1)}$ and $y_{(2)}$ are ϵ -approximations with ϵ_1 and ϵ_2 respectively, we have

$$\left| \frac{dy_{(1)}}{dt} - f(t, y_{(1)}) \right| \leq \epsilon_1 \quad (3.76)$$

$$\left| \frac{dy_{(2)}}{dt} - f(t, y_{(2)}) \right| \leq \epsilon_2 \quad (3.77)$$

$$(3.78)$$

Let

$$\epsilon = \epsilon_1 + \epsilon_2 \quad (3.79)$$

$$p(t) = y_{(2)}(t) - y_{(1)}(t) \quad (3.80)$$

Then

$$\left| \frac{dp}{dt} \right| \leq |f(t, y_{(1)}) - f(t, y_{(2)})| + \epsilon \quad (3.81)$$

$$\leq K |y_{(1)} - y_{(2)}| + \epsilon \quad (3.82)$$

$$\leq K |p| + \epsilon \quad (3.83)$$

except for a possible finite number of locations where dp/dt is undefined.

Suppose that $(t) \neq 0, t_0 < t \leq t_0 + h$. Then since its sign does not change, $p(t)$ is either always positive or always negative. Suppose it is positive. Then $p(t) > 0$, and

$$p'(t) \leq Kp(t) + \epsilon \quad (3.84)$$

Multiply the equation by e^{-Kt} and rearrange to give

$$e^{-Kt} [p'(t) - Kp(t)] \leq \epsilon e^{-Kt} \quad (3.85)$$

The left hand side is exactly

$$\frac{d}{dt} (p(t)e^{-Kt}) = e^{-Kt} [p'(t) - Kp(t)] \quad (3.86)$$

Hence

$$\frac{d}{dt} (p(t)e^{-Kt}) \leq \epsilon e^{-Kt} \quad (3.87)$$

Integrate both sides from t_0 to t ,

$$\int_{t_0}^t \frac{d}{dx} (p(x)e^{-Kx}) dx \leq \epsilon \int_{t_0}^t e^{-Kx} dx \quad (3.88)$$

$$p(t)e^{-Kt} - p(t_0)e^{-Kt_0} \leq -\frac{\epsilon}{K} [e^{-Kt} - e^{-Kt_0}] \quad (3.89)$$

Solving for $p(t)$ gives the fundamental inequality. The proof for $p(t) < 0$ is analogous.

Now suppose that $p(t) = 0$ identically for all t . Then the fundamental inequality is equivalent to $0 < \text{something positive}$ and follows trivially. \square

3.4 Cauchy-Euler Existence Theory

In this section the quantities $f, (t_0, y_0), D, h$ are defined as before; R is the usual $2h$ wide rectangle centered on (t_0, y_0) .

Theorem 3.5. Uniqueness. *Suppose that $f \in \mathcal{C}(D), f \in \mathcal{L}(y; K)(D)$, and let $(t_0, y_0) \in D$. Then there is at most one solution to 3.1.*

Proof. Suppose that $y(t)$ and $\tilde{y}(t)$ are exact solutions of 3.1. Then

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0 \quad (3.90)$$

$$\tilde{y}'(t) = f(t, \tilde{y}(t)), \quad \tilde{y}(t_0) = y_0 \quad (3.91)$$

From the fundamental inequality, equation (3.75),

$$|y(t) - \tilde{y}(t)| \leq e^{K|t-t_0|} |y(t_0) - \tilde{y}(t_0)| + \frac{\epsilon_1 + \epsilon_2}{K} (e^{K|t-t_0|} - 1) = 0 \quad (3.92)$$

Hence the two functions are identical to one another for all t . \square

Theorem 3.6. Fundamental Existence Theorem. *Let $f(t, y) \in \mathcal{L}(y, K)(D)$. Then an exact solution of the initial value problem 3.1 exists on some interval $|t - t_0| < h$.*

Proof. Let $\epsilon_n \rightarrow 0$ monotonically. Then by theorem 3.3 a sequence of ϵ_n -approximate solutions $y_n(t)$ exist, such that \Leftarrow

$$|y_n'(t) - f(t, y_n(t))| \leq \epsilon_n, |t - t_0| \leq h \quad (3.93)$$

except possibly at the finite set of points $t_i^{(n)}, i = 1, \dots, m_n$

By the fundamental inequality, for any integers n and p ,

$$|y_n(t) - y_{n+p}(t)| \leq e^{K|t-t_0|} |y_n(t_0) - y_{n+p}(t_0)| + \frac{\epsilon_n + \epsilon_{n+p}}{K} (e^{K|t-t_0|} - 1) \quad (3.94)$$

$$\leq e^{K|t-t_0|} |y_0 - y_0| + \frac{2\epsilon_n}{K} (e^{Kh} - 1) \rightarrow 0 \quad (3.95)$$

as $n \rightarrow \infty$. Hence the sequence $y_n(t)$ converges uniformly to a continuous function $y(t)$.

Consequently, $y_n(t) \rightarrow y(t)$ uniformly, and since $f(t, y)$ is continuous on a rectangle R , the sequence $f(t, y_n(t)) \rightarrow f(t, y(t))$ uniformly on R . Hence the sequence of integrals

$$\int_{t_0}^t f(x, y_n(x)) dx \rightarrow \int_{t_0}^t f(x, y(x)) dx \quad (3.96)$$

uniformly on R .

We now integrate both sides of inequality 3.93 from t_0 to t .

$$\left| \int_{t_0}^t \left[\frac{dy_n(x)}{dx} - f(x, y_n(x)) \right] dx \right| \leq \int_{t_0}^t \left| \left[\frac{dy_n(x)}{dx} - f(x, y_n(x)) \right] \right| dx \quad (3.97)$$

$$\leq \int_{t_0}^t \epsilon_n dx \quad (3.98)$$

$$\leq \epsilon_n |t_0 - t| \quad (3.99)$$

$$\leq \epsilon_n h \quad (3.100)$$

By the fundamental theorem of calculus, since the y_n are continuous, we can also integrate the first term on the left side of the inequality,

$$\left| y_n(t) - y_n(t_0) - \int_{t_0}^t f(x, y_n(x)) dx \right| \leq \epsilon_n h \quad (3.101)$$

Taking the limit of both sides of the equation as $n \rightarrow \infty$,

$$y(t) = y(t_0) + \int_{t_0}^t f(x, y(x)) dx \quad (3.102)$$

We observe that this function satisfies the initial value problem. Hence a solution exists. \square

When we defined the ϵ -approximate solution we only required that it satisfy the initial value problem, namely, the differential equation and the initial condition. We did not set any restrictions on whether or not the "approximation" matched the real solution in any way whatsoever. The following theorem tells us that we can construct an ϵ -approximation that is arbitrarily close to the actual solution. In other words, it is possible to use this technique to find a solution to the IVP that matches the true solution with any arbitrary accuracy.

Theorem 3.7. Cauchy-Euler Approximation Theorem. *Suppose that $y(t)$ is an exact solution of the initial value problem*

$$y' = f(t, y) \quad (3.103)$$

$$y'(t_0) = y_0 \quad (3.104)$$

where $f(t, y) \in \mathcal{L}(y; K)(D)$. Let $\tilde{y}(t)$ be an ϵ -approximate solution to $y(t)$ satisfying $\tilde{y}'(t_0) = y_0$ for $|t - t_0| \leq h$. Then there exists a constant $N > 0$, independent of ϵ , such that

$$|\tilde{y}(t) - y(t)| \leq \epsilon N \quad (3.105)$$

for $|t - t_0| \leq h$.

Proof. From the fundamental inequality,

$$|\tilde{y}(t) - y(t)| \leq e^{K|t-t_0|} |\tilde{y}(t_0) - y(t_0)| + \frac{\epsilon + 0}{K} (e^{K|t-t_0|} - 1) \quad (3.106)$$

$$\leq e^{Kh} |y_0 - y_0| + \frac{\epsilon}{K} (e^{Kh} - 1) \quad (3.107)$$

$$\leq \frac{\epsilon}{K} (e^{Kh} - 1) \quad (3.108)$$

So if we let

$$N = \frac{1}{K} (e^{Kh} - 1) \quad (3.109)$$

the conclusion to the theorem follows immediately. \square

Theorem 3.8. Let D be an arbitrary bounded domain, $f \in \mathcal{L}(y; K)(D)$. Let $y(t)$ be an exact solution of the IVP

$$y' = f(t, y), y(t_0) = y_0$$

that is also defined for $t = t_1$. The value of $y(t_1)$ may be determined to any desired degree of accuracy, using Euler's method, but using a sufficiently small step size.

Proof. Let d be the minimum distance between the curve $y(t)$ and the boundary of D . If η is any number $0 < \eta < d$ then the strip

$$S = \{(t, y) | t_0 \leq t \leq t_1, |y - y(t)| \leq \eta\} \quad (3.110)$$

lies entirely in D . Let \mathcal{E} be the desired error. Choose ϵ so that

$$\epsilon = \min \left(\mathcal{E}, \frac{K\eta}{e^{K|t_1-t_0|} - 1} \right) \quad (3.111)$$

By equation 3.108

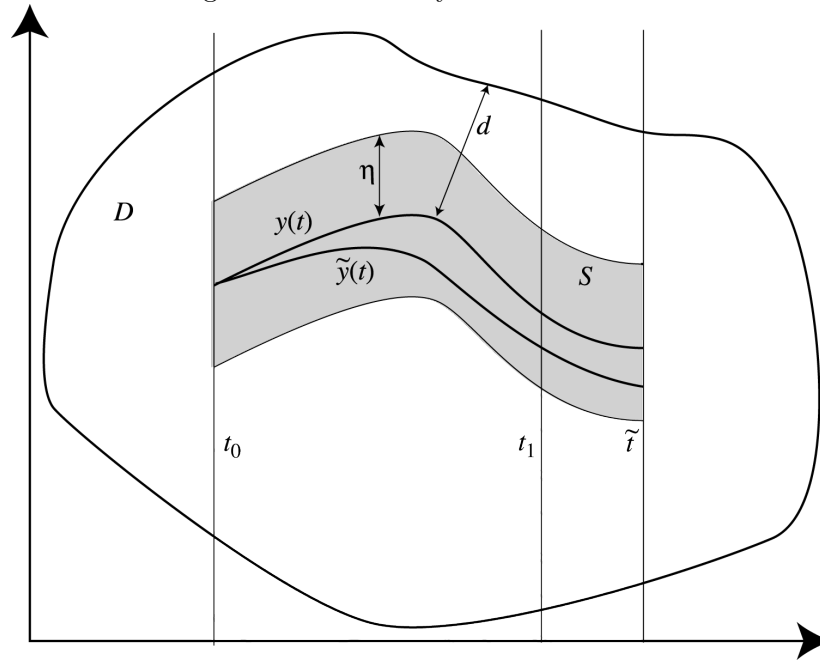
$$|\tilde{y}(\tilde{t}) - y(\tilde{t})| \leq \frac{\epsilon}{K} (e^{K|\tilde{t}-t_0|} - 1) \quad (3.112)$$

$$\leq \eta \frac{e^{K|t_1-t_0|} - 1}{e^{K|\tilde{t}-t_0|} - 1} \quad (3.113)$$

$$< \eta \quad (3.114)$$

Thus $\tilde{y}(t)$ is within the strip S . Within S , the conditions of the Fundamental Inequality are met, so we can continue to approximate y by \tilde{y} . Hence $\tilde{y}(t)$ is defined \Leftarrow up to at least $t = t_1$ with the desired error. \square

Figure 3.3: Geometry for Theorem 3.8



3.5 Euler's Method for Systems

The numerical methods we will discuss in this class can all easily be extended to systems. For a system, we have

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \quad (3.115)$$

$$\mathbf{y}(t_0) = \mathbf{y}_0 \quad (3.116)$$

To emphasize that \mathbf{y} and \mathbf{y}_0 are vectors and that \mathbf{f} is a vector function we have written them all in bold-face. Euler's method then becomes

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h_n \mathbf{f}(t_n, \mathbf{y}_n) \quad (3.117)$$

Example 3.2. Solve the initial value problem

$$y_1'(t) = y_2(t) \quad (3.118)$$

$$y_2'(t) = -2y_1(t) \quad (3.119)$$

$$y_1(0) = 1 \quad (3.120)$$

$$y_2(0) = 0 \quad (3.121)$$

on the interval $[0, \pi]$ using $h = \pi/4$.

Solution. Let $\mathbf{y} = (y_1, y_2)^T$. Then $\mathbf{y}(0) = \mathbf{y}_0 = (1, 0)^T$, and we write

$$\mathbf{f}(t, \mathbf{y}) = (y_2, -2y_1)^T \quad (3.122)$$

For the first iteration,

$$\mathbf{f}(0, \mathbf{y}(0)) = (0, -2) \quad (3.123)$$

$$\mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{f}(0, \mathbf{y}(0)) \quad (3.124)$$

$$= (1, 0)^T + \frac{\pi}{4}(0, -2)^T \approx (1, -1.5708)^T \quad (3.125)$$

$$t_1 = t_0 + h = \pi/4 \quad (3.126)$$

At the second iteration,

$$\mathbf{f}(t_1, \mathbf{y}_1) \approx \mathbf{f}(\pi/4, (1, -1.5708)^T) \quad (3.127)$$

$$\approx (-1.5708, -2)^T \quad (3.128)$$

$$\mathbf{y}_2 = \mathbf{y}_1 + h\mathbf{f}(t_1, \mathbf{y}_1) \quad (3.129)$$

$$\approx (1, -1.5708)^T + \frac{\pi}{4}(-1.5708, -2) \quad (3.130)$$

$$\approx (-.2337, -3.1416) \quad (3.131)$$

$$t_2 = t_1 + \pi/4 = \pi/2 \quad (3.132)$$

After the third iteration,

$$\mathbf{f}(t_2, \mathbf{y}_2) \approx \mathbf{f}(\pi/2, (-.2337, -3.1416)^T) \quad (3.133)$$

$$\approx (-3.1416, .4674)^T \quad (3.134)$$

$$\mathbf{y}_3 = \mathbf{y}_2 + h\mathbf{f}(t_2, \mathbf{y}_2) \quad (3.135)$$

$$\approx (-.2337, -3.1416)^T + \frac{\pi}{4}(-3.1416, .4674)^T \quad (3.136)$$

$$\approx (-2.7011, -2.7745)^T \quad (3.137)$$

$$t_3 = t_2 + \pi/4 = 3\pi/4 \quad (3.138)$$

After the fourth and final iteration,

$$\mathbf{f}(t_3, \mathbf{y}_3) \approx \mathbf{f}(3\pi/4, (-2.7011, -2.7745)^T) \quad (3.139)$$

$$\approx (-2.7745, 5.4022)^T \quad (3.140)$$

$$\mathbf{y}_4 = \mathbf{y}_3 + h\mathbf{f}(t_3, \mathbf{y}_3) \quad (3.141)$$

$$\approx (-2.7011, -2.7745)^T + \frac{\pi}{4}(-2.7745, 5.4022)^T \quad (3.142)$$

$$\approx (-4.8802, 1.4684)^T \quad (3.143)$$

$$t_4 = t_3 + \pi/4 = \pi \quad \square \quad (3.144)$$

The Mathematica implementation discussed in section 1.8 does not even need to be modified. For a scalar problem one would define a function $\mathbf{f}[\mathbf{t}, \mathbf{y}]$ that returns a scalar value, and then call `ForwardEuler[f, {t0, y0}, h, tmax]`. The return value is a list of the form

$$\{\{t_0, y_0\}, \{t_1, y_1\}, \dots\}$$

For a system, one defines a function $\mathbf{f}[t, \mathbf{y}]$ that returns a list, and then call `ForwardEuler[f, {t0, y0}, h, tmax]`, as before, where the argument \mathbf{y}_0 is also a list of the same length as \mathbf{f} and \mathbf{y} . The return value becomes instead

$$\{\{t_0, \{y_{00}, y_{01}, \dots\}\}, \{t_1, \{y_{10}, y_{11}, \dots\}\}, \dots\}$$

Algorithm 3.2. Forward Euler Method for Systems *To solve the system of initial value problems*

$$\begin{aligned} y_1' &= f_1(t, y_1, y_2, \dots), & y_1(t_0) &= y_{0,1} \\ y_2' &= f_2(t, y_1, y_2, \dots), & y_2(t_0) &= y_{0,2} \\ y_3' &= f_3(t, y_1, y_2, \dots), & y_3(t_0) &= y_{0,3} \\ & \vdots & & \end{aligned}$$

on an interval $[t_0, t_{max}]$ with a fixed step size h .

1. input: $f_1(t, y_1, \dots), f_2(t, y_1, \dots), \dots, t_0, y_{0,1}, y_{0,2}, \dots, h, t_{max}$

2. output: $(t_0, y_{10}, y_{20}, \dots)$

3. let $t = t_0, u_1 = y_{0,1}, u_2 = y_{0,2}, u_3 = y_{0,3}, \dots$

4. while $t < t_{max}$

(a) let

$$\begin{aligned} y_{n,1} &= u_1 + hf_1(t, u_1, u_2, u_3, \dots) \\ y_{n,2} &= u_2 + hf_2(t, u_1, u_2, u_3, \dots) \\ y_{n,3} &= u_3 + hf_3(t, u_1, u_2, u_3, \dots) \\ & \vdots \end{aligned}$$

(b) let $t = t + h$

(c) let $t_n = t$

(d) output: (t_n, y_n)

(e) let $u_1 = y_{n,1}, u_2 = y_{n,2}, u_3 = y_{n,3}, \dots$

Example 3.3. *Solve the initial value problem*

$$y_1'(t) = y_2(t) \tag{3.145}$$

$$y_2'(t) = -2y_1(t) \tag{3.146}$$

$$y_1(0) = 1 \tag{3.147}$$

$$y_2(0) = 0 \tag{3.148}$$

using `ForwardEuler` on $[0, 2\pi]$ using $h = 0.1\pi$, and plot the results.

Solution. First we define the function and the initial conditions, then we solve the system.

```
y0={1,0};
f[t_, y_]:= {y[[2]], -2y[[1]]};
r=ForwardEuler[f,{0, y0}, 0.1π, 2π];
```

The function `ListPlot` can be used to plot either points or a joined line, but not both on the same function call. To avoid having to repeatedly call `ListPlot`, we define the following function `JoinedListPlot` which will plot a set of points as a sequence of connected dots.

```
JoinedListPlot[data_, color_, opt___?OptionQ] := Show[
  ListPlot[data, PlotJoined -> True, PlotStyle -> color,
    DisplayFunction -> Identity],
  ListPlot[data, PlotStyle -> color, PointSize[0.02],
    DisplayFunction -> Identity],
  opt, DisplayFunction -> $DisplayFunction]
```

The two calls to `ListPlot` generate the plot with the connected lines (the first call, with `PlotJoined->True`); and then, the plot with the dots (the second call). The options `DisplayFunction->Identity` tell Mathematica to actually suppress actually showing the individual plots, and to just calculate the graphical elements that make up the plots. The call to `Show` then renders the two plots generated by `ListPlot` by using the option `DisplayFunction->$DisplayFunction`. The argument `opt` is used to pass any desired options to the `Show` function.

We actually want to generate two plots for our simulation: one for y_1 , and the other for y_2 , but we want them plotted on a single graph. Our new function `JoinedListPlot` will plot a single data set, but not two data sets. But we can repeat the same trick we used with the definition of `JoinedListPlot`: set `DisplayFunction->Identity` and then call the function twice. In outline form,

```
p1 = JoinedListPlot[FirstDataSet, Red, DisplayFunction->Identity];
p2 = JoinedListPlot[SecondDataSet, Blue, DisplayFunction->Identity];
Show[p1, p2, DisplayFunction -> $DisplayFunction]
```

The main problem now is to extract the data sets. The data is in the form

$$\{ \{ \{t_0, \{y_{0,1}, y_{0,2}, \dots\} \}, \{ \{t_1, \{y_{1,1}, y_{1,2}, \dots\} \}, \dots \}$$

and what we need for each plot is

$$\{ \{t_0, y_{0,1}\}, \{t_1, y_{1,1}\}, \{t_2, y_{2,1}\}, \dots \}$$

for the first data set, and

$$\{ \{t_0, y_{0,2}\}, \{t_1, y_{1,2}\}, \{t_2, y_{2,2}\}, \dots \}$$

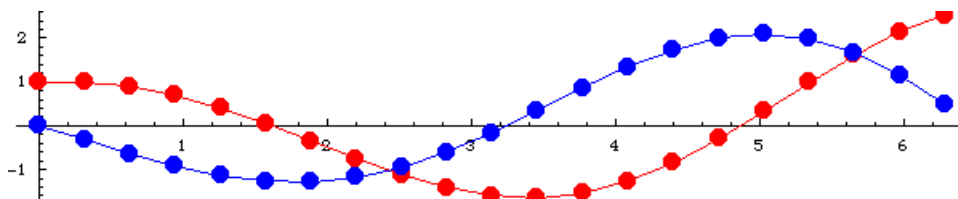
for the second data set. We get these from

```
firstDataSet = Most /@ Flatten /@ r;
secondDataSet = Drop[#, {2}] & /@ Flatten /@ r;
```

Each “@” symbol is shorthand for the Map function. Thus `Most /@Flatten /@r` is equivalent to `Map[Flatten[r]]`. The Map function applies a function to every element in a list, namely, `f/@{a, b, c}` is the same as `{f[a], f[b], f[c]}`.

Putting the whole solution together in one place, here is our program:

```
y0={1,0};
f[t_, y_]:= {y[[2]], -y[[1]]};
r=ForwardEuler[f,{0, y0}, 0.1π, 2π];
firstDataSet = Most /@ Flatten /@ r;
secondDataSet = Drop[#, {2}] & /@ Flatten /@ r;
p1 = JoinedListPlot[FirstDataSet, Red, DisplayFunction->Identity];
p2 = JoinedListPlot[SecondDataSet, Blue, DisplayFunction->Identity];
Show[p1, p2, DisplayFunction -> $DisplayFunction, AspectRatio->0.2]
```



3.6 Polynomial Approximation

Theorem 3.9 (Weierstrass Approximation Theorem). *Any continuous function can be approximated by a polynomial to any desired degree of accuracy. More specifically, suppose that $f(t) \in C(R)$. Then for any $\epsilon > 0$, there exists a polynomial $P(t)$ such that*

$$\|P - f\| < \epsilon \tag{3.149}$$

\implies *Proof.* We will prove² the theorem on $[0, 1]$; the generalization to any closed interval $[a, b]$ follows by an algebraic transformation $\hat{t} = (b - a)t + a$. We first prove the

²The proof follows the *Encyclopedia of Mathematics* object number 6145, AMS classification 41A10, online edition only at <http://planetmath.org>. The proof can be skipped without any loss of continuity.

theorem for $f(t) = 1 - \sqrt{1-t}$. Construct the sequence³ of Polynomials

$$P_0(t) = 0 \quad (3.150)$$

$$P_{n+1}(t) = \frac{1}{2}(P_n(t)^2 + t) \quad (3.151)$$

on $[0, 1]$.

We observe first that $0 \leq P(t) \leq 1$. This may be proven by induction. First, \Leftarrow
 $P_1 = t/2 \leq 1$ on $[0, 1]$. Next, assume that $P_n < 1$. Then $P_{n+1} = \frac{1}{2}(P_n^2 + t) \leq \frac{1}{2}(1^2 + 1) = 1$.

Furthermore, each P_n is monotonically increasing. Again, prove by induction. Certainly $P_1 = t/2$ is monotonically increasing with slope $1/2$. Assume $P'_n > 0$. Then $P'_{n+1} = \frac{1}{2}(2P_n P'_n + 1) > 0$ because the P_n are all positive. Hence the function is monotonically increasing.

Next we observe that the sequence of polynomials is itself increasing. To prove this, observe that

$$P_{n+2}(t) - P_{n+1}(t) = \frac{1}{2}(P_{n+1}^2(t) + t - P_n^2(t) - t) \quad (3.152)$$

$$= \frac{1}{2}(P_{n+1}^2(t) - P_n^2(t)) \quad (3.153)$$

$$= \frac{1}{2}(P_{n+1}(t) + P_n(t))(P_{n+1}(t) - P_n(t)) \quad (3.154)$$

Since $t/2 = P_1(t) \geq P_0(x) = 0$ for all $t \in [0, 1]$, by applying this recursively we see that the sequence is monotonically increasing: $P_{n+1}(t) \geq P_n(t)$ on $[0, 1]$.

Since the sequence P_0, P_1, \dots is bounded and monotonically increasing it converges to some limit $P(t)$. But

$$\lim_{n \rightarrow \infty} P_{n+1}(t) = \lim_{n \rightarrow \infty} \frac{1}{2}(P_n^2(t) + t) \quad (3.155)$$

$$P(t) = \frac{1}{2}(P^2(t) + t) \quad (3.156)$$

$$0 = P^2(t) - 2P(t) + t \quad (3.157)$$

$$P(t) = 1 \pm \sqrt{1-t} \quad (3.158)$$

We need to take the negative square root because of the restriction $P(t) \leq 1$. Hence

$$\lim_{n \rightarrow \infty} P_n(t) = 1 - \sqrt{1-t} \quad (3.159)$$

³Use of the fixed point iteration $x_0 = 0$, $x_{n+1} = \frac{1}{2}(u + x_n^2)$ to find \sqrt{z} , where $z = 1 - u$ and $x = 1 - \sqrt{z}$, was discovered in ancient Babylonia.

Finally, since the $P_n(t)$ are a monotonic sequence with $P_{n+1}(t) \geq P_n(t)$, we know that $P_m(t) \geq P_n(t)$ whenever $m > n$. Therefore

$$P_m(t) - P_m(1) \leq P_n(t) - P_n(1) \quad (3.160)$$

and hence

$$P_m(t) - P_n(t) \leq P_m(1) - P_n(1) \quad (3.161)$$

$$\lim_{m \rightarrow \infty} (P_m(t) - P_n(t)) \leq \lim_{m \rightarrow \infty} (P_m(1) - P_n(1)) \quad (3.162)$$

$$P(t) - P_n(t) \leq 1 - P_n(1) \quad (3.163)$$

$$1 - \sqrt{1-t} - P_n(t) \leq 1 - P_n(1) \quad (3.164)$$

\implies Pick any $\epsilon > 0$. Then, since $P_n(1) \rightarrow P(1)$ as $n \rightarrow \infty$, there exists some n such that $1 - P_n(1) < \epsilon$, in which case

$$1 - \sqrt{1-t} - P_n(t) < \epsilon \quad (3.165)$$

i.e.,

$$\|f(t) - P_n(t)\| < \epsilon \quad (3.166)$$

This proves the Weierstrass Approximation Theorem for the special case $f(t) = 1 - \sqrt{1-t}$.

Next, suppose that $f(t) = |t - c|$ for some number $c \in [0, 1]$. Let $u = |t - c|$ and pick any $\epsilon > 0$. Since $u, \epsilon \geq 0$, we know that

$$u^2 + \frac{\epsilon^2}{4} < u^2 + u\epsilon + \frac{\epsilon^2}{4} = \left(\frac{\epsilon}{2} + u\right)^2 \quad (3.167)$$

$$\sqrt{u^2 + \frac{\epsilon^2}{4}} < \frac{\epsilon}{2} + u \quad (3.168)$$

$$\sqrt{u^2 + \frac{\epsilon^2}{4}} - u < \frac{\epsilon}{2} \quad (3.169)$$

Now define a new variable v ,

$$v = 1 - u^2 - \epsilon^2/4 \quad (3.170)$$

By the previous case there is some polynomial $Q(v)$ that approximates $1 - \sqrt{1-v}$ arbitrarily closely:

$$|1 - \sqrt{1-v} - Q(v)| < \epsilon/2 \quad (3.171)$$

Define a new polynomial $P(v) = 1 - Q(v)$; then we have

$$|\sqrt{1-v} - P(v)| < \epsilon/2 \quad (3.172)$$

$$\left| \sqrt{u^2 + \frac{\epsilon^2}{4}} - P(v) \right| < \epsilon/2 \quad (3.173)$$

But since $P(v)$ is a polynomial in v , and v is a polynomial in u , then P is also a polynomial in u . But u is a polynomial in t , hence P is a polynomial in t , which we write as follows:

$$\left| \sqrt{u^2 + \frac{\epsilon^2}{4}} - P(t) \right| < \frac{\epsilon}{2} \quad (3.174)$$

Now since

$$u^2 + \frac{\epsilon^2}{4} < u^2 + \frac{\epsilon^2}{4} + \epsilon|u| = \left(|u| + \frac{\epsilon}{2} \right)^2 \quad (3.175)$$

$$\sqrt{u^2 + \frac{\epsilon^2}{4}} < |u| + \frac{\epsilon}{2} \quad (3.176)$$

$$\sqrt{u^2 + \frac{\epsilon^2}{4}} - |u| < \frac{\epsilon}{2} \quad (3.177)$$

Adding equations 3.174 and 3.177, we get

$$\left| \sqrt{u^2 + \frac{\epsilon^2}{4}} - P(t) \right| + \sqrt{u^2 + \frac{\epsilon^2}{4}} - |u| < \epsilon \quad (3.178)$$

Since $u^2 + \epsilon^2/4 > u^2$, we have

$$\left| \sqrt{u^2 + \frac{\epsilon^2}{4}} - P(t) \right| + \left| \sqrt{u^2 + \frac{\epsilon^2}{4}} - |u| \right| < \epsilon \quad (3.179)$$

$$\left| \sqrt{u^2 + \frac{\epsilon^2}{4}} - P(t) \right| + \left| |u| - \sqrt{u^2 + \frac{\epsilon^2}{4}} \right| < \epsilon \quad (3.180)$$

Using the triangle inequality on the last expression gives

$$|P(t) - u| < \epsilon \quad (3.181)$$

which proves the Weierstrass Approximation Theorem for $f(t) = u = |t - c|$.

Now consider any piecewise linear function

$$f(t) = b + \sum_{k=0}^N a_k |t - c_k| \quad (3.182)$$

By the previous case there exist $N + 1$ polynomials Q_0, Q_1, \dots, Q_N such that

$$|a_k |t - c_m| - Q_m(t)| < \frac{\epsilon}{N} \quad (3.183)$$

By the triangle inequality,

$$|f(t) - P(t)| < \epsilon \quad (3.184)$$

where

$$P(t) = \sum_{k=1}^N Q_k(t) \quad (3.185)$$

This proves Weierstrass' Approximation theorem for a piecewise linear function.

Since you can approximate any continuous function arbitrarily closely by a piecewise linear function this proves the theorem for any continuous function. \square

3.7 Peano Existence Theorem

Our proof of the existence of a solution to the general IVP requires a Lipschitz Condition be placed upon $f(t, y)$. It turns out that this condition is not necessary.

Equicontinuity

Definition 3.2. Let $f_k(t_1, t_2, \dots)$ be a sequence of functions. Then we say that the set of functions $\{f_k\}$ is **Uniformly Bounded** if there exists some number $M \in \mathbb{R}$ such that for all t_1, t_2, \dots, k ,

$$|f_k(t_1, t_2, \dots)| \leq M \quad (3.186)$$

In particular, the constant M is the same for every function in the set.

Definition 3.3. A set S of functions $f(t)$ is **equicontinuous** over an interval $[a, b]$ if for any $\epsilon > 0$ there exists a $\delta > 0$ such that whenever $t, \tilde{t} \in [a, b]$,

$$|t - \tilde{t}| \leq \delta \implies |f(t_1) - f(t_2)| \leq \epsilon \quad (3.187)$$

for functions $f \in S$.

Lemma 3.3. Suppose that S is an infinite set of uniformly bounded, equicontinuous functions on $[a, b]$. The S contains a sequence that converges uniformly on $[a, b]$.

Theorem 3.10 (Peano). Let $D = [t_0 - a, t_0 + a] \times [y_0 - b] \times [y_0 + b]$; Suppose that $f(t, y) \in C(D)$ be bounded by M . Then there is a solution to the initial value \implies problem

$$y' = f(t, y) \quad (3.188)$$

$$y(t_0) = y_0 \quad (3.189)$$

for $|t - t_0| < h = \min(a, b/M)$.

Proof. Let $\epsilon_n \rightarrow 0$ be a sequence of numbers, and $P_n(t, y)$ be a sequence of polynomials such that

$$|P_n(t, y) - f(t, y)| \leq \epsilon_n \quad (3.190)$$

on

$$t_0 - h \leq t \leq t_0 + h \quad (3.191)$$

$$y_0 - Mh \leq y \leq y_0 + Mh \quad (3.192)$$

where $h = \min(a, b/M)$. We know by the Weierstrass approximation theorem that such a sequence exists.

The set of functions $\{P_n(t, y)\}$ is uniformly bounded. Denote by M the common bound of both f and $\{P_n(t, y)\}$. Since P_n are polynomials, they satisfy the Lipschitz condition on any bounded domain such as D , hence by the fundamental existence theorem already proven, there exists a sequence of functions $y_n(t)$, $t_0 - h \leq t \leq t_0 + h$, such that

$$y_n'(t) = P_n(t, y_n(t)) \quad (3.193)$$

$$y_n(t_0) = y_0 \quad (3.194)$$

where

$$|y_n(t) - y_n(t_0)| \leq Mh, \quad t_0 - h \leq t \leq t_0 + h \quad (3.195)$$

i.e., the set of functions $y_n(t)$ are uniformly bounded on $[t_0 - h, t_0 + h]$.

By the mean value theorem, for any t_1, t_2 , there exists some c between t_1 and t_2 such that

$$y_n(t_2) - y_n(t_1) = y_n'(c)(t_2 - t_1) \quad (3.196)$$

But by definition, the $y_n' = P_n$, hence

$$|y_n(t_2) - y_n(t_1)| = |P_n(c)||t_2 - t_1| \leq M|t_2 - t_1| \quad (3.197)$$

Let $\epsilon > 0$ and choose $\delta = \epsilon/M$. Then if $|t_2 - t_1| \leq \delta$,

$$|y_n(t_2) - y_n(t_1)| \leq \epsilon \quad (3.198)$$

Therefore $S = \{y_n\}$ is equicontinuous.

Since S is uniformly bounded and equicontinuous on $[t_0 - h, t_0 + h]$, it has a uniformly convergent sequence. Call this sequence $\phi_n(t) \rightarrow \phi$. By definition, $\phi_n \in S$ hence there exists some $P_{k(n)}$ such that $\phi_n'(t) = P_{k(n)}(t, \phi_n(t))$ and therefore

$$\phi_n(t) - y_0 = \int_{t_0}^t P_{k(n)}(s, \phi_n(s)) ds \quad (3.199)$$

$$\left| \phi_n(t) - y_0 - \int_{t_0}^t f(s, \phi_n(s)) ds \right| \leq \int_{t_0}^t |P_{k(n)}(s, \phi_n(s)) - f(s, \phi_n(s))| ds \quad (3.200)$$

$$\leq \epsilon_n h \quad (3.201)$$

Taking the limit as $n \rightarrow \infty$ gives

$$\left| \phi(t) - y_0 - \int_{t_0}^t f(s, \phi(s)) ds \right| = 0 \quad (3.202)$$

Hence ϕ is a solution of the IVP, proving existence. \square

3.8 Dependence Upon a Parameter

Theorem 3.11. (*Dependence on Initial Conditions*) Let $f(t, y) \in \mathcal{C}(R)$ where

$$R = [t_0 - a, t_0 + a] \times [\tilde{y}_0 - b, \tilde{y}_0 + b], \quad (3.203)$$

then there exists a unique solution of the IVP with perturbed IC,

$$y' = f(t, y) \quad (3.204)$$

$$y(t_0) = y_0 \quad (3.205)$$

in the region

$$|t - t_0| \leq h' \quad (3.206)$$

$$|y_0 - \tilde{y}_0| \leq b/2 \quad (3.207)$$

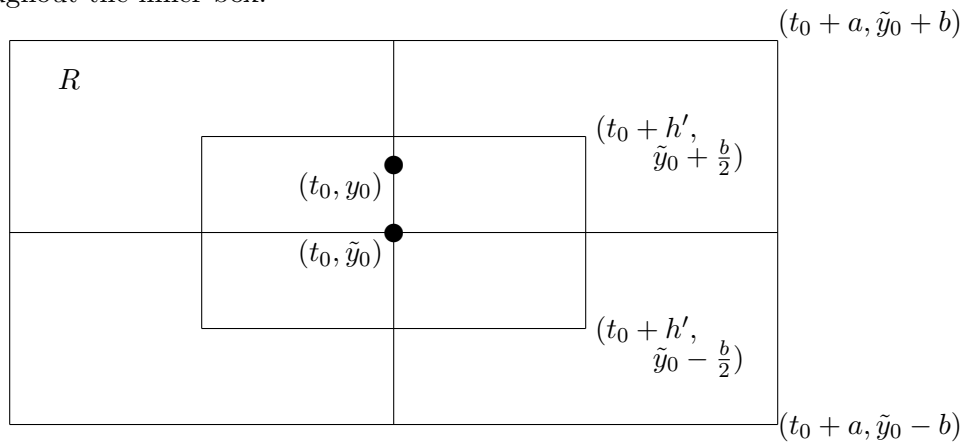
$$|y - y_0| \leq Mh' \quad (3.208)$$

where

$$h' = \min\left(a, \frac{b}{2M}\right) \quad (3.209)$$

$$|f(t, y)| \leq M \text{ in } R \quad (3.210)$$

Figure 3.4: The initial condition can move along the vertical axis and a solution exists throughout the inner box.



Uniform Continuity

A function $y = f(t) : D \mapsto R$ is **continuous** if small changes in t make only small changes in y ,

$$(\forall t_1, t_2 \in D)(\forall \epsilon > 0)(\exists \delta > 0) \ni (|t_1 - t_2| < \delta \implies |f(t_1) - f(t_2)| < \epsilon) \quad (3.211)$$

A function $y = f(t) : D \mapsto R$ is **uniformly continuous** if small changes in t make only small changes in y , and furthermore, the size of the changes in y do not depend on the size of the changes in t .

$$(\forall \epsilon > 0)(\exists \delta > 0) \ni (\forall t_1, t_2 \in D)(|t_1 - t_2| < \delta \implies |f(t_1) - f(t_2)| < \epsilon) \quad (3.212)$$

Uniform Continuity implies **continuity** but not vice-versa.

On compact (closed and bounded) domains, continuity implies uniform continuity.

Lipshitz Continuity implies uniform continuity.

The following result tells us that **solutions that start “together” stay together**.

Theorem 3.12. *Let $f(t, y) \in C(D)$, $f \in \mathcal{L}(y, K)(D)$, and let y be a solution to*

$$y' = f(t, y) \quad (3.213)$$

$$y(t_0) = y_0 \quad (3.214)$$

on a rectangle $R = [t_0 - h, t_0 + h] \times [\tilde{y}_0 - l, \tilde{y}_0 + l]$. Then $y(t, y_0)$ is continuous in both t and y_0 on R .

Proof. Suppose we have two solutions $y(t, y_{01}), y(t, y_{02})$ on R . Then by the Fundamental Inequality (equation 3.75)

$$|y(t_0, y_{01}) - y(t_0, y_{02})| \leq |y_{01} - y_{02}| e^{Kh} \quad \text{for } |t - t_0| \leq h \quad (3.215)$$

The $y(t_0, y_0)$ is continuous in y_0 *uniformly* in t . □

Theorem 3.13. *Under the same conditions as Theorem 3.12, and, in addition, if $\partial f / \partial y$ exists and is continuous in both t and y on D , then*

$$\frac{\partial y(t, y_0)}{\partial y_0}$$

exists and is continuous in both t and y .

Proof. Pick a \tilde{y}_0 , and write $\tilde{y}(t) = y(t, \tilde{y}_0)$ and $y(t) = y(t, y_0)$. Define

$$p(t, y_0) = \frac{y(t) - \tilde{y}(t)}{y_0 - \tilde{y}_0}, \quad \text{for } y_0 \neq \tilde{y}_0 \quad (3.216)$$

By the ODE ($y' = f(t, y)$) and the mean value theorem (in the second argument)

$$\frac{d}{dt} [y(t) - \tilde{y}(t)] = f(t, y(t)) - f(t, \tilde{y}(t)) \quad (3.217)$$

$$= [y(t) - \tilde{y}(t)] \frac{\partial}{\partial y} f(t, \eta(t)) \quad (3.218)$$

for some $\eta(t)$ between $y(t)$ and $\tilde{y}(t)$. Hence

$$\frac{\partial p(t, y_0)}{\partial t} = \frac{y(t) - \tilde{y}(t)}{y_0 - \tilde{y}_0} \frac{\partial}{\partial y} f(t, \eta(t)) \quad (3.219)$$

$$= p(t, y_0) \frac{\partial}{\partial y} f(t, \eta(t)) \quad (3.220)$$

where $\eta(t) \rightarrow y(t)$ as $\tilde{y}(t) \rightarrow y(t)$. Since $p \neq 0$ so long as $y_0 \neq \tilde{y}_0$, we may divide by p :

$$\frac{1}{p(t, y_0)} \frac{\partial p(t, y_0)}{\partial t} = \frac{\partial}{\partial y} f(t, \eta(t)) \quad (3.221)$$

$$\ln p(t, y_0) = \int_{t_0}^t \frac{\partial}{\partial y} f(s, \eta(s)) ds \quad (3.222)$$

The constant of integration is determined from $p(t_0, y_0) = 1$, and

$$p(t, y_0) = \exp \left\{ \int_{t_0}^t \frac{\partial}{\partial y} f(s, \eta(s)) ds \right\} \quad (3.223)$$

By the definition of p ,

$$\frac{\partial y(t, \tilde{y}_0)}{\partial y_0} = \lim_{y_0 \rightarrow \tilde{y}_0} p(t, y_0) \quad (3.224)$$

$$= \exp \left\{ \int_{t_0}^t \frac{\partial}{\partial y} f(s, \tilde{y}(s)) ds \right\} \quad (3.225)$$

which is continuous in both arguments. □

Chapter 4

Improving on Euler's Method

4.1 The Test Equation and Problem Stability

Consider the initial value problem

$$y' = f(t, y), \quad y(0) = y_0 \quad (4.1)$$

What happens if we perturb the initial condition a small amount,

$$\hat{y}' = f(t, \hat{y}), \quad \hat{y}(0) = \hat{y}_0 \quad (4.2)$$

where $\hat{y}_0 = y_0 + \delta$ for some small number δ ? How will the two solutions behave with respect to one another?

Consider, for example, the following initial value problem, which we shall refer to as the **scalar test equation**.

$$y' = \lambda y, \quad y(0) = y_0 \quad (4.3)$$

We will find it instructive to test many of our methods with the scalar test problem and will formulate much of our theory around this problem, mainly because it is easy to solve and we can describe its behavior very easily. The solution of the scalar problem is

$$y(t) = y_0 e^{\lambda t} \quad (4.4)$$

The perturbed equation has solution

$$y(t) = \hat{y}_0 e^{\lambda t} \quad (4.5)$$

and so if $\lambda \in \mathbb{R}$ the two solutions will differ by

$$\epsilon(t) = |y(t) - \hat{y}(t)| = |y_0 - \hat{y}_0| e^{\lambda t} \quad (4.6)$$

Therefore

$$\lim_{t \rightarrow \infty} \epsilon(t) = \begin{cases} 0, & \lambda < 0 \\ |y_0 - \hat{y}_0|, & \lambda = 0 \\ \infty, & \lambda > 0 \end{cases} \quad (4.7)$$

In particular, the difference is bounded is $\lambda \leq 0$, and we say that the differential equation is **stable**. If $\lambda < 0$ and the difference approaches zero, we say that the differential equation is **asymptotically stable**.

If $\lambda \in \mathbb{C}$ then the same argument holds for $Re(\lambda)$, since

$$\epsilon(t) = |y(t) - \hat{y}(t)| = |y_0 - \hat{y}_0|e^{Re(\lambda)t} \quad (4.8)$$

We can formally define stability as follows.

Definition 4.1 (Stability). *A solution $y(t)$ is **stable** if, for any $\epsilon > 0$ there is a $\delta > 0$ such that all other solutions $\hat{y}(t)$ satisfying the same IVP with*

$$|y_0 - \hat{y}_0| \leq \delta \quad (4.9)$$

also satisfies

$$|y(t) - \hat{y}(t)| \leq \epsilon \quad (4.10)$$

for all values of $t \geq 0$.

Definition 4.2 (Absolute Stability). *A solution is **absolutely stable** if it is stable and*

$$\lim_{t \rightarrow \infty} |y(t) - \hat{y}(t)| = 0 \quad (4.11)$$

Definition 4.3 (Totally Stable). *Let*

$$y' = f(t, y) + \delta(t), \quad y(t_0) = y_0 + \delta \quad (4.12)$$

*define a perturbation of the initial value problem 4.1. Let \hat{y} and y be the solutions corresponding to two different perturbations $\{\delta(t), \delta\}$ and $\{\hat{\delta}(t), \hat{\delta}\}$. Then the IVP is said to be **totally stable** if there exists a positive constant S such that for all $t \in [a, b]$, whenever*

$$|\delta(t) - \hat{\delta}(t)| \leq \epsilon \text{ and } |\delta - \hat{\delta}| \leq \epsilon \quad (4.13)$$

then

$$|y - \hat{y}| \leq S\epsilon \quad (4.14)$$

Let us examine what this means for a numerical solution. Suppose we are solving the test equation numerically in steps of size h ; at the end of each step, an additional error of δ accumulates. We can “simulate” this process by integrating the solution *exactly* on each interval, and then perturbing the next initial condition by δ as illustrated in figure 4.1. For the first interval, the calculated solution is

$$y = y_0 e^{\lambda(t-t_0)}, \quad t_0 \leq t < t_0 + h \quad (4.15)$$

At $t = t_0 + h$ the solution by δ , hence for $t_0 + h \leq t < t_0 + 2h$, we have

$$y = (y_0 e^{\lambda(t_0+h-t_0)} + \delta) e^{\lambda(t-(t_0+h))} \quad (4.16)$$

$$= y_0 e^{\lambda(t-t_0)} + \delta e^{\lambda(t-(t_0+h))} \quad (4.17)$$

At $t = t_0 + 2h$ the solution jumps again by δ , to

$$y(t_0 + 2h) = y_0 e^{2\lambda h} + \delta e^{\lambda(t_0 + 2h - (t_0 + h))} \quad (4.18)$$

$$= y_0 e^{2\lambda h} + \delta e^{\lambda h} \quad (4.19)$$

hence for $t_0 + h \leq t < t_0 + 2h$, we have

$$y(t_0 + 2h) = (y_0 e^{2\lambda h} + \delta e^{\lambda h} + \delta) e^{\lambda(t - (t_0 + 2h))} \quad (4.20)$$

$$= y_0 e^{\lambda(t - t_0)} + \delta e^{\lambda(t - (t_0 + h))} + \delta e^{\lambda(t - (t_0 + 2h))} \quad (4.21)$$

In general, the solution in the n^{th} interval will be

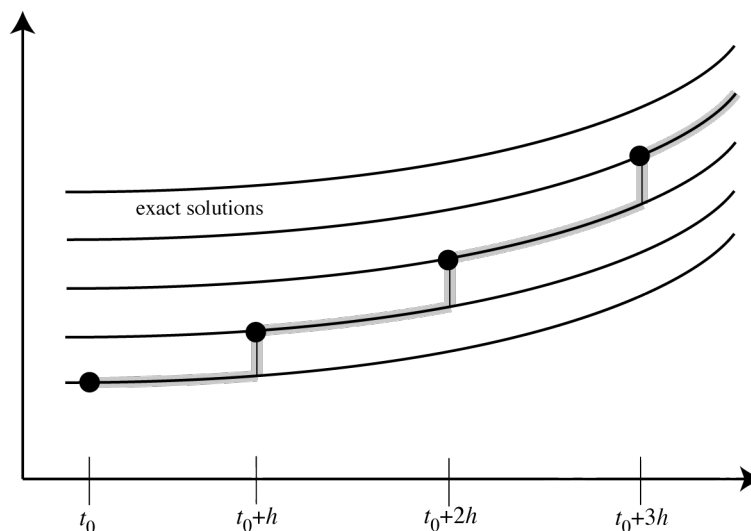
$$y = y_0 e^{\lambda(t - t_0)} + \delta e^{\lambda(t - (t_0 + h))} + \delta e^{\lambda(t - (t_0 + 2h))} + \dots + \delta e^{\lambda(t - (t_0 + n))} \quad (4.22)$$

$$= y_0 e^{\lambda(t - t_0)} + \delta e^{\lambda(t - t_0)} \sum_{k=1}^n e^{-k\lambda h} \quad (4.23)$$

Summing the geometric series,

$$y = y_0 e^{\lambda(t - t_0)} + \delta e^{\lambda(t - t_0)} \frac{1 - e^{-n\lambda h}}{e^{\lambda h} - 1} \quad (4.24)$$

Figure 4.1: Illustration of the experimental process of simulating numerical error due to numerical integration by a stepwise approach. The numerical solution is illustrated by the black dots. Exact solutions to the IVP pass through those dots. Starting at (t_0, y_0) , the exact solution is followed through $(t_0 + h, y(t_0 + h))$. Then a new initial value problem is solved, starting at $t_0 + h$, with initial value $y(t_0 + h) + \delta$. The process is then repeated. The solution that emulates the numerical solution is shaded.



The error after n steps is then

$$e_n(t) = |y(t) - \hat{y}(t)| = \delta e^{\lambda(t-t_0)} \left| \frac{1 - e^{-n\lambda h}}{e^{\lambda h} - 1} \right| \quad (4.25)$$

For $|n\lambda h| \ll 1$,

$$e_n(t) \approx \delta e^{\lambda(t-t_0)} \quad (4.26)$$

and hence for $\lambda < 0$ the error decreases asymptotically with time. For larger times, with the approximation is not valid, since $t > t_0 + nh$, the $e^{\lambda(t-t_0)}$ factor still dominates the fraction, and hence the error still tends towards zero.

The analogous **vector test equation** is written as

$$\mathbf{y}' = \mathbf{A}\mathbf{y}, \quad \mathbf{y}(0) = \mathbf{y}_0 \quad (4.27)$$

where \mathbf{A} is a square matrix of constants. The solution is given by the matrix exponential

$$\mathbf{y} = e^{\mathbf{A}t} \mathbf{y}_0 \quad (4.28)$$

where **the matrix exponential** is defined by its Taylor Series,

$$e^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{t^k \mathbf{A}^k}{k!} \quad (4.29)$$

If the matrix \mathbf{A} is diagonalizable, or equivalently, has n linearly independent eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$, with corresponding eigenvalues $\lambda_1, \dots, \lambda_n$, then

$$e^{\mathbf{A}t} = \mathbf{U}\mathbf{E}\mathbf{U}^{-1} \quad (4.30)$$

where

$$\mathbf{E} = \text{diag}(e^{\lambda_1 t}, \dots, e^{\lambda_n t}) \quad (4.31)$$

and

$$\mathbf{U} = (\mathbf{v}_1, \dots, \mathbf{v}_n) \quad (4.32)$$

Example 4.1. Solve the initial value problem

$$\mathbf{y}' = \mathbf{A}\mathbf{y}, \quad \mathbf{A} = \begin{pmatrix} 0 & 1 \\ -2 & 0 \end{pmatrix}, \quad \mathbf{y}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (4.33)$$

Solution. \mathbf{A} is diagonalizable with eigenvalues $\pm i\sqrt{2}$ and corresponding eigenvectors

$$\begin{pmatrix} \frac{i}{\sqrt{2}} \\ 1 \end{pmatrix} \text{ and } \begin{pmatrix} -\frac{i}{\sqrt{2}} \\ 1 \end{pmatrix} \quad (4.34)$$

We have then

$$\mathbf{y} = e^{\mathbf{A}t} \mathbf{y}_0 = \begin{pmatrix} -\frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} e^{i\sqrt{2}t} & 0 \\ 0 & e^{-i\sqrt{2}t} \end{pmatrix} \begin{pmatrix} \frac{i}{\sqrt{2}} & \frac{1}{2} \\ -\frac{i}{\sqrt{2}} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} \quad (4.35)$$

$$= \begin{pmatrix} \frac{e^{-i\sqrt{2}t} + e^{i\sqrt{2}t}}{2} \\ i \frac{e^{i\sqrt{2}t} - e^{-i\sqrt{2}t}}{\sqrt{2}} \end{pmatrix} \quad (4.36)$$

$$= \begin{pmatrix} \cos(\sqrt{2}t) \\ -\sqrt{2} \sin(\sqrt{2}t) \end{pmatrix} \quad \square \quad (4.37)$$

More generally, the matrix \mathbf{A} may not be diagonalizable; however, there is still a similarity transform to **Jordan Canonical Form**

$$\mathbf{J} = \mathbf{U}^{-1}\mathbf{A}\mathbf{U} \quad (4.38)$$

where \mathbf{U} is the matrix of generalized eigenvectors $\mathbf{v}_k, k = 1, \dots, n$, where

$$(\mathbf{A} - \lambda\mathbf{I})^k \mathbf{v}_k = \mathbf{0}, \quad (4.39)$$

The Jordan Canonical Form of a matrix is the block-diagonal matrix

$$\mathbf{J} = \begin{pmatrix} \mathbf{\Lambda}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{\Lambda}_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & \mathbf{\Lambda}_m \end{pmatrix} \quad (4.40)$$

where each **Jordan Block** corresponds to one eigenvalue of the original matrix

$$\mathbf{\Lambda}_i = \begin{pmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_i & 1 \\ 0 & \cdots & \cdots & 0 & \lambda_i \end{pmatrix} \quad (4.41)$$

In this case we have

$$e^{\mathbf{A}t} = \mathbf{U}e^{\mathbf{J}t}\mathbf{U}^{-1} \quad (4.42)$$

It is generally easier to calculate the solution

$$\mathbf{y} = \sum_{i=1}^m \sum_{p=1}^{k_i} \frac{t^{k_i-p}}{(k_i-p)!} \mathbf{v}_p \mathbf{i}e^{\lambda_i t} \quad (4.43)$$

where eigenvalue λ_i is repeated k_i times with generalized eigenvectors $\mathbf{v}_{p,i}, p = 1, \dots, k_i$.¹

Theorem 4.1. *The vector test equation is stable if all of the eigenvalues satisfy either $Re(\lambda) < 0$, or $Re(\lambda) = 0$ and λ is simple. The solution is asymptotically stable if all the eigenvalues satisfy $Re(\lambda) < 0$.*

4.2 Convergence, Consistency and Stability

Definition 4.4 (One Step Method). *Any method that can be written in the form*

$$y_{n+1} = y_n + h\phi(t_n, y_n, h) \quad (4.44)$$

is called a one-step method.

¹For a derivation of this result see Gray, A., Mezzino, M. & Pinsky, M.A. *Introduction to Ordinary Differential Equations with Mathematica*, Springer (1997)

Definition 4.5. [Local Truncation Error] Let $u(t_n)$ be any function defined on a mesh, and define the **difference operator** \mathcal{N} by

$$\mathcal{N}u(t_n) = \frac{u(t_n) - u(t_{n-1})}{h_n} - \phi(t_{n-1}, u(t_{n-1})) \quad (4.45)$$

The **Local Truncation Error** d_n is given by $\mathcal{N}y(t_n)$, where $y(t_n)$ is the exact solution evaluated on the mesh:

$$d_n = \mathcal{N}y(t_n) = \frac{y(t_n) - y(t_{n-1})}{h_n} - \phi(t_{n-1}, y(t_{n-1})) \quad (4.46)$$

The local truncation error gives an estimate of the error in discretizing the differential equation at y_n assuming that there are no errors at y_{n-1} - the local error in the calculation of the derivative. For **Euler's Method**,

$$y_{n+1} = y_n + h_n f(t_n, y_n) \quad (4.47)$$

we can derive the local truncation error using a Taylor Series approximation about $y(t_{n-1})$,

$$d_n = \mathcal{N}(t_n) = \frac{y(t_n) - y(t_{n-1})}{h} - f(t_{n-1}, y(t_{n-1})) \quad (4.48)$$

$$= \frac{y(t_{n-1}) + hy'(t_{n-1}) + \frac{1}{2}h^2y''(t_{n-1}) + \cdots - y(t_{n-1})}{h} - f(t_{n-1}, y(t_{n-1})) \quad (4.49)$$

Using the fact that $y'(t_{n-1}) = f(t_{n-1}, y(t_{n-1}))$,

$$d_n = \frac{h}{2}y''(t_{n-1}) + O(h^2) \quad (4.50)$$

Thus the local truncation error for Euler's method is proportional to h . We write this as $d_n = O(h)$, and say that Euler's method is a **first order method**.

Definition 4.6 (Convergence). A one step method is said to **converge** on an interval $[a, b]$ if $y_n \rightarrow y$ on $[a, b]$ as $n \rightarrow \infty$ for any IVP

$$y' = f(t, y), y(t_0) = y_0 \quad (4.51)$$

with $f(t, y)$ Lipschitz in y . A method is said to be **convergent of order p** if for some positive integer p the **global error** $e_n = |y(t_n) - y_n|$ satisfies

$$e_n = |y(t_n) - y_n| = O(h^p) \quad (4.52)$$

Definition 4.7 (Stability, Zero-Stability, 0-Stability). A one-step method is said to be **stable** if for each IVP (4.44) with f Lipschitz in y there exists $K, h_0 > 0$ such that the difference between two different mesh functions (not necessarily solutions, just functions that are defined on the mesh) y_n and \hat{y}_n satisfies

$$|y_n - \hat{y}_n| \leq K \left[|y_0 - \hat{y}_0| + \|\mathcal{N}y_n - \mathcal{N}\hat{y}_n\| \right] \quad (4.53)$$

for all $h \in [0, h_0]$, where $\|\cdot\|$ denotes the sup-norm,

$$\|u_n\| = \max_{1 \leq j \leq n} |u_j| \quad (4.54)$$

Theorem 4.2. *Euler's method is zero-stable.*

Proof.

$$\|\mathcal{N}u_n - \mathcal{N}u_n^*\| = \left\| \frac{u_n - u_n^*}{h} - \frac{u_{n-1} - u_{n-1}^*}{h} - (f(u_{n-1}) - f(u_{n-1}^*)) \right\| \quad (4.55)$$

$$\geq \left\| \frac{u_n - u_n^*}{h} \right\| - \left\| \frac{u_{n-1} - u_{n-1}^*}{h} + (f(u_{n-1}) - f(u_{n-1}^*)) \right\| \quad (4.56)$$

By the triangle inequality and the Lipschitz condition

$$\left\| \frac{u_n - u_n^*}{h} \right\| \leq \|\mathcal{N}u_n - \mathcal{N}u_n^*\| + \left\| \frac{u_{n-1} - u_{n-1}^*}{h} \right\| + K\|u_{n-1} - u_{n-1}^*\| \quad (4.57)$$

$$\|u_n - u_n^*\| \leq h\|\mathcal{N}u_n - \mathcal{N}u_n^*\| + \|u_{n-1} - u_{n-1}^*\| + hK\|u_{n-1} - u_{n-1}^*\| \quad (4.58)$$

$$= h\|\mathcal{N}u_n - \mathcal{N}u_n^*\| + (1 + hK)\|u_{n-1} - u_{n-1}^*\| \quad (4.59)$$

Apply the result recursively $n - 1$ additional times to give

$$\|u_n - u_n^*\| \leq h\|\mathcal{N}u_n - \mathcal{N}u_n^*\| + (1 + hK)\|u_{n-1} - u_{n-1}^*\| \quad (4.60)$$

$$\leq h\|\mathcal{N}u_n - \mathcal{N}u_n^*\| + \quad (4.61)$$

$$(1 + hk)[h\|\mathcal{N}u_n - \mathcal{N}u_n^*\| + (1 + hk)\|u_{n-2} - u_{n-2}^*\|] \quad (4.62)$$

$\leq \vdots$

$$\leq h\|\mathcal{N}u_n - \mathcal{N}u_n^*\| \sum_{i=1}^n (1 + hK)^{n-i} + (1 + hk)^n \|u_0 - u_0^*\| \quad (4.63)$$

$$\leq k \left\{ \|u_0 - u_0^*\| + \|\mathcal{N}u_n - \mathcal{N}u_n^*\| \right\} \quad (4.64)$$

where

$$k = \max \left\{ h \sum_{i=1}^n (1 + hK)^{n-i}, (1 + hk)^n \right\} \quad (4.65)$$

Therefore the method is 0-stable. \square

A zero-stable method depends continuously on the initial data. If a method is not zero stable that a small perturbation in the method could potentially lead to an infinite change in the results. Suppose that

$$y_n = y_n + 1 + h\phi(t_n - 1, y_{n-1}, \dots) \quad (4.66)$$

is a numerical method. Then we define a **perturbation δ of the method** as

$$\tilde{y}_n = y_n + 1 + h\phi(t_n - 1, y_{n-1}, \dots) + \delta_n \quad (4.67)$$

The following theorem is normally taken as the definition of zero-stability.

Theorem 4.3. Let δ and δ^* be two different perturbations of a method y_n and let the \tilde{y}_n and \tilde{y}_n^* be the corresponding perturbed methods. Then the method y_n is zero-stable if and only if for any $\epsilon > 0$ there exist constants S and h_0 such that for all $h \in (0, h_0]$ whenever

$$\|\delta_n - \delta_n^*\| \leq \epsilon, 0 \leq n \leq N \quad (4.68)$$

then

$$\|\tilde{y}_n - \tilde{y}_n^*\| \leq S\epsilon, 0 \leq n \leq N \quad (4.69)$$

Definition 4.8 (Consistency). A one-step method is said to be **consistent** if $\phi(t, y(t), 0) = f(t, y(t))$. A method is said to be **consistent (or accurate) to order p** or $O(h^p)$ if the leading term in the local truncation error has order p ,

$$d_n = O(h^p) \quad (4.70)$$

Theorem 4.4 (Convergence = Consistency + Stability). A one-step method converges if it is consistent of order p and 0-stable.

Proof. By zero-stability

$$|y_n - y(t_n)| \leq K \left[|y_0 - y(0)| + \|\mathcal{N}y_n - \mathcal{N}y(t_n)\| \right] \quad (4.71)$$

$$= K \|\mathcal{N}y_n - d_n\| \quad (4.72)$$

$$= K \left\| \frac{y_n - y_{n-1}}{h_n} - \phi(t_n, y_n) - d_n \right\| \quad (4.73)$$

$$= K \|d_n\| = O(h^p) \quad (4.74)$$

where the last step follows by consistency (equation 4.70). Hence the method is convergent of order p by equation 4.52. \square

Definition 4.9 (Local Error). The **local error**

$$I_n = \tilde{y}(t_n) - y_n \quad (4.75)$$

is the amount by which the numerical solution y_n differs from the exact solution of the **Local IVP**

$$\tilde{y}'(t) = f(t, \tilde{y}(t)) \quad (4.76)$$

$$\tilde{y}(t_{n-1}) = y_{n-1} \quad (4.77)$$

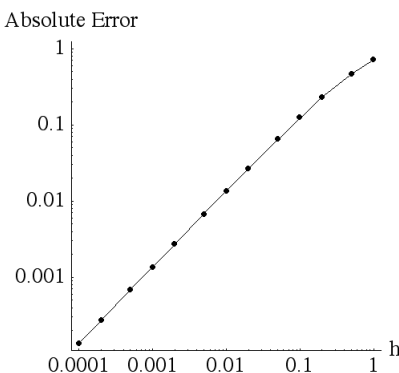
Theorem 4.5. In general for the IVP methods we will consider,

$$h|\mathcal{N}\tilde{y}(t_n)| = |I_n|(1 + O(h)) \quad (4.78)$$

$$|d_n| = |\mathcal{N}\tilde{y}(t_n)| + O(h^{p+1}) \quad (4.79)$$

Ideally we want a method to be as accurate as possible, in the sense that the absolute error is minimized. In general, reducing the step size of an $O(h^p)$ method will reduce the error by a factor of h^p . The question naturally arises as to what is a sufficiently small step size. For example, consider the test equation $y' = y, y(0) = 1$. We

Figure 4.2: Absolute error in the computed solution to $y' = y, y(0) = 1$ at $t = 1$ as a function of step size, using Euler's method.



have already seen in example 3.1 that the numerical solution smoothly approaches the exact solution on $[0,1]$ as h decreases. The exact solution is $y = e^t$; at $t = 1$, we find that $y(1) = e$. Figure 4.2 shows the absolute error $|y_n(1) - e|$ for various values of h , illustrating that in this case the error does indeed decrease linearly with h .

Now consider the IVP

$$y' = -5ty^2 + \frac{5}{t} - \frac{1}{t^2}, \quad y(1) = 1 \quad (4.80)$$

The exact solution is $y = 1/t$. The numerical solution is plotted for three different step sizes on the interval $[1, 25]$ in figure 4.3. Clearly something appears to be happening here around $h = 0.2$, but what is it? For smaller step sizes, a relatively smooth solution is obtained, and for larger values of h the solution becomes progressively more jagged.

Some insight into this question can be obtained by returning to the test equation $y' = \lambda y$. Euler's method for $f(t, y) = y$ gives

$$y_n = (1 + h\lambda)y_{n-1} \quad (4.81)$$

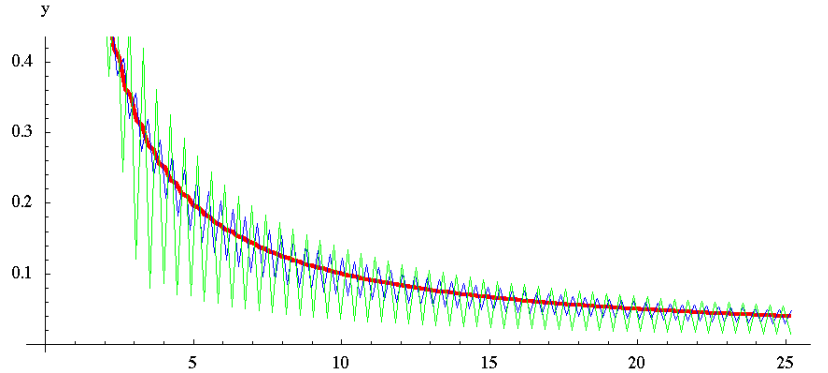
$$= (1 + h\lambda)^2 y_{n-2} \quad (4.82)$$

$$\vdots \quad (4.83)$$

$$= (1 + h\lambda)^n y_0 \quad (4.84)$$

Unless $|y_{n+1}/y_n| = |1 + h\lambda| \leq 1$ the numerical solution will eventually grow without bounds. The **region of absolute stability** is that subset of the complex plane in which this condition is satisfied (cf figure 4.4). For a system of equations, all of the eigenvalues must fall within the region of absolute stability. For a scalar equation, only the single value of λ need fall within the region. If the region of absolute stability includes the entire left hand-side of the plane (corresponding to

Figure 4.3: Numerical solution of equation 4.80 for $h = 0.190$ (red), $h = 0.205$ (blue), and $h = 0.23$ (green).



$\text{Re}(h\lambda) \leq 0$) then the method is said to be **A-Stable**. A-Stable methods are preferable for stiff problems. A stronger requirement is **Stiff Decay**: consider the generalized test equation

$$y' = \lambda(y - g(t)) \quad (4.85)$$

where $g(t)$ is any arbitrary, bounded function. Then we say that equation 4.85 has *stiff decay* if for any fixed $t_n > 0$,

$$\lim_{h\text{Re}(\lambda) \rightarrow -\infty} |y_n(t) - g(t_n)| = 0. \quad (4.86)$$

To determine the shape of this region we observe that in the complex plane we can write

$$h\lambda = x + iy \quad (4.87)$$

and therefore

$$1 \geq |1 + x + iy|^2 \quad (4.88)$$

$$= (1 + x + iy)(1 + x - iy)^2 \quad (4.89)$$

$$= (1 + x)^2 + y^2 \quad (4.90)$$

which is a disk of radius 1 centered at the point $(-1, 0)$.

If $\lambda \in \mathbb{R}$ then this condition gives

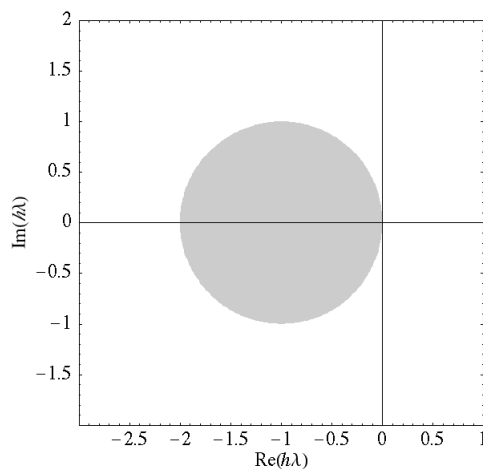
$$-1 \leq 1 + h\lambda \leq 1 \quad (4.91)$$

$$-2 \leq h\lambda \leq 0 \quad (4.92)$$

If $\lambda < 0$ then

$$h \leq -\frac{2}{\lambda} \quad (4.93)$$

Figure 4.4: The region of absolute stability for Euler's method is indicated by the gray area.



To find the eigenvalue of equation 4.80 we linearize. Since the system is scalar, the Jacobian reduces to a single partial derivative $\partial f/\partial y$. For $f(t, y) = -5ty^2 + 5/t - 1/t^2$ we have the linearized equation

$$y' \approx f_y(1, 1)y \quad (4.94)$$

$$= \left(-10(t)(y)|_{t=1, y=1} \right) y \quad (4.95)$$

$$= -10y \quad (4.96)$$

Hence $\lambda = -10$ and we need $h \leq -2/(-10) = 0.2$ to remain in the region of absolute stability. Of course, this eigenvalue will change as the solution progresses, and values of t and y change, so the step size needs to be adjusted dynamically to remain within the desired region.

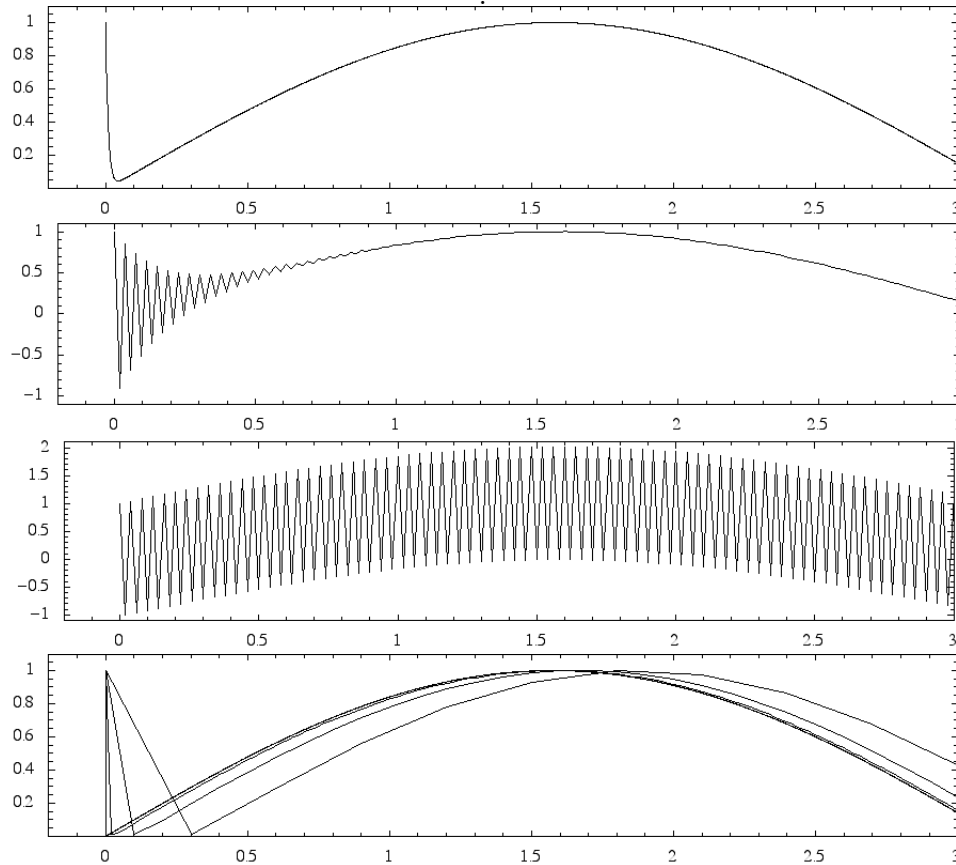
4.3 Stiffness and the Backward Euler Method

An initial value problem is said to be **stiff** if the absolute stability requirement leads to a much smaller value of h than would otherwise be needed to satisfy the accuracy requirements. Thus stiffness depends on three factors: (1) accuracy; (2) the length of the interval of integration; and (3) the region of absolute stability of the problem. An example is given in figure 4.5. From equation 4.92 this problem, which has $\lambda = -100$ requires $h < 0.02$ for small t . For the test equation the problem is **stiff on the interval** $[0, b]$ if

$$b\text{Re}(\lambda) \ll -1 \quad (4.97)$$

One way to fix Euler's method is to calculate the function f at the *forward* time point rather than the *backward* time point. This gives the **backward Euler**

Figure 4.5: Result of the forward Euler method to solve $y' = -100(y - \sin t)$, $y(0) = 1$ with $h = 0.001$ (top), $h = 0.019$ (middle), and $h = 0.02$ (third). The bottom figure shows the same equation solved with the backward Euler method for step sizes of $h = 0.001, 0.02, 0.1, 0.3$, left to right curves, respectively



method:

$$y_n = y_{n-1} + h_n f(t_n, y_n) \quad (4.98)$$

For the scalar test equation this gives

$$y_n = y_{n-1} + h\lambda y_n \quad (4.99)$$

Solving for y_n ,

$$y_n = \frac{1}{1 - h\lambda} y_{n-1} \quad (4.100)$$

The absolute stability requirement $|y_n/y_{n-1}| < 1$ gives

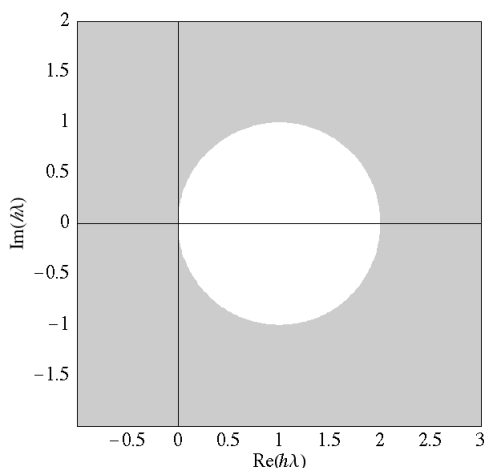
$$|1 - h\lambda| \geq 1 \quad (4.101)$$

or (substituting $z = h\lambda = x + iy$,

$$(1 - x)^2 + y^2 \geq 1 \quad (4.102)$$

Thus the region of absolute stability for the Backwards Euler Method encompasses the entire complex plane *outside* the circle of radius 1 centered at the point $(1, 0)$, as illustrated in figure 4.6.

Figure 4.6: The gray area indicates the region of absolute stability for the Backwards Euler Method.



The Backwards Euler Method is an example of an **implicit method**, because it contains y_n implicitly. In general it is not possible to solve for y_n explicitly as a function of y_{n-1} in equation 4.98, even though it is sometimes possible to do so for specific differential equations. Thus at each mesh point one needs to make some first guess to the value of y_n and then perform some additional refinement to improve the calculation of y_n before moving on to the next mesh point. A common method is to use **fixed point iteration** on the equation

$$y = k + hf(t, y) \quad (4.103)$$

where $k = y_{n-1}$. The technique is summarized here:

- Make a first guess at y_n and use that in right hand side of 4.98. A common first guess that works reasonably well is

$$y_n^{(0)} = y_{n-1} \quad (4.104)$$

- Use the better estimate of y_n produced by 4.98 and then evaluate 4.98 again to get a third guess, e.g.,

$$y_n^{(\nu+1)} = y_{n-1} + hf(t_n, y_n^{(\nu)}) \quad (4.105)$$

- Repeat the process until the difference between two successive guesses is smaller than the desired tolerance.

We could implement this in Mathematica as follows:

```

BackwardEulerFP[f_, {t0_, y0_}, h_, tmax_, tol_, imax_] :=
⇒ Module[{ time, yval, yvaln, yvalp, delta, eps, r},
r = {{t0, y0}}; yval = y0;
⇒ For[time = t0, time < tmax, time += h,
(***** Do next Backward Euler Step *****)
⇒ yvaln = yval + f[time+h, yval];
(***** Do Fixed Point Iteration on y *****)
⇒ For[i = 1, i ≤ imax, i++,
yvalp = yvaln;
⇒ yvaln = yval + f[time+h, yvalp];
delta = yvaln - yvalp;
If[Abs[delta] < tol, Break[]];
]
yval = yvaln;
AppendTo[r, {time + h, yval}]; ];
Return[r]; ]

```

We know that equation 4.103 is only guaranteed to have a fixed point, and iteration is only guaranteed to converge, if it is a contraction mapping. This requires $\|hf_y(t, y)\| < 1$. Convergence becomes rapid when $\|hf_y(t, y)\| \ll 1$; for values close to 1 the rate of convergence can be quite slow. Unfortunately, this can place a rather strong restriction on the step size, and hence other techniques such as Newton's method tend to be preferred.

Testing to see if the problem is ill-conditioned is quite easy in Mathematica if you have an analytic expression for $f(t, y)$. For example, in the code block just presented, one could add the following calculation at the top of the program:

```

J = D[f[t, y], {y, 1}];
JV = J /. y -> y0, t -> t0;
If[Abs[JV] >= 1,
Print["This problem is ill-conditioned for fixed point iteration."];
Return[$Failed];
];

```

This calculates the scalar Jacobian analytically, and places the formula in the variable `J`, and then calculates the numerical value of the Jacobian in the variable `JV`. If the magnitude of the Jacobian is larger than or equal to one, then fixed point iteration is not guaranteed to work, so the program aborts. Later in the algorithm, after the next step is calculated, one also needs to test to see if the problem has entered an ill-conditioned region. To do this one can recalculate the value of the Jacobian, using the analytic formula already determined, merely substituting the current values of y and t .

```

JV = J/.y-> ynext, t-> time
If[Abs[JV]>= 1 ...

```

Algorithm 4.1. Backward Euler Method Using Fixed Point Iteration. *To solve the initial value problem*

$$y' = f(t, y), y(t_0) = y_0$$

on an interval $[t_0, t_{max}]$ with a fixed step size h .

1. input: $f(t, y), t_0, y_0, h, t_{max}, tol$
2. output: (t_0, y_0)
3. let $t = t_0, y = y_0$
4. while $t < t_{max}$
 - (a) let $i = 0$
 - (b) let $y_{next}^{(0)} = y + hf(t + h, y) \quad \Leftarrow$
 - (c) Repeat:

$$\begin{aligned} y_{next}^{(i+1)} &= y + hf(t + h, y_{next}^{(i)}) \quad \Leftarrow \\ i &= i + 1 \end{aligned}$$

until $|y_{next}^{(i+1)} - y_{next}^{(i)}| < tol$

- (d) let $y = y_{next}^{(i+1)}$
- (e) let $t = t + h$
- (f) let $t_n = t, y_n = y$
- (g) output: $(t_0, y_0), \dots, (t_n, y_n)$

Theorem 4.6 (Newton's Method). *To find the root r of the scalar equation $g(t) = 0$, iterate on*

$$r^{(i+1)} = r^{(i)} - \frac{g(r^{(i)})}{g'(r^{(i)})} \quad (4.106)$$

To find the root \mathbf{r} of the vector equation $\mathbf{g}(\mathbf{y}) = \mathbf{0}$, iterate on

$$\mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} - \left(\frac{\partial \mathbf{g}(\mathbf{y})}{\partial \mathbf{y}} \right)^{-1} \Bigg|_{\mathbf{y}=\mathbf{r}^{(i)}} \mathbf{g}(\mathbf{r}^{(i)}) \quad \Leftarrow \quad (4.107)$$

For the scalar IVP, we can use Newton's method to find the root of

$$g(y_n) = y_n - y_{n-1} - hf(t, y_n) = 0 \quad (4.108)$$

The iteration formula is

$$y_n^{(i+1)} = y_n^{(i)} - \frac{g(y_n^{(i)})}{g'(y_n^{(i)})} \quad (4.109)$$

$$= y_n^{(i)} - \frac{y_n^{(i)} - y_{n-1} - hf(t, y_n^{(i)})}{1 - hf_y(t, y_n^{(i)})} \quad (4.110)$$

For the vector initial value problem, the corresponding iteration formula is

$$\mathbf{y}_n^{(i+1)} = \mathbf{y}_n^{(i)} - \left(I - h \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right)^{-1} \left(\mathbf{y}_n^{(i)} - \mathbf{y}_{n-1} - h\mathbf{f}(t_n, \mathbf{y}_n^{(i)}) \right) \quad (4.111)$$

Again, it is not generally efficient to calculate a matrix inverse; it is better to solve a linear system of equations. We can reformulate the last equation as

$$\left(\mathbf{y}_n^{(i+1)} - \mathbf{y}_n^{(i)} \right) \left(I - h \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right) = - \left(\mathbf{y}_n^{(i)} - \mathbf{y}_{n-1} - h\mathbf{f}(t_n, \mathbf{y}_n^{(i)}) \right) \quad (4.112)$$

We can solve the linear system for $\delta = \mathbf{y}_n^{(i+1)} - \mathbf{y}_n^{(i)}$ and then calculate $\mathbf{y}_n^{(i+1)} = \delta + \mathbf{y}_n^{(i)}$.

We then have the following Mathematica Implementation.

```

BackwardEulerNewtonsMethod[f_, {t0_, y0_}, h_, tmax_, tol_:0.003,
  nmax_:5] :=
Module[{t, y, n, time, yval, yvaln, yvalp, delta, r, J, JV,
  fv},
r = {{t0, y0}};
yval = y0;
J = D[f[t, y], {y, 1}];
For[time = t0, time < tmax, time += h,
  (***** calculate first guess at next grid point *****)
=> fv = f[time+h, yval];
=> JV = J /. {t -> time+h, y -> yval};
yvaln = (-yval + h (-fv + JV *yval))/(-1+h*JV);
  (***** iterate using Newton's method *****)
=> For[i = 1, i <= nmax, i++,
  yvalp = yvaln;
  (***** update derivative *****)
=> JV = J /. {t -> time+h, y -> yvalp};
  (***** update function value *****)
=> fv = f[time+h, yvalp];
  (***** Newton Iteration Formula *****)
yvaln = (yval + h(fv - JV* yvalp))/(1-h*JV);
  (***** Within desired tolerance? *****)

```

```

    delta = yvaln-yvalp;
    If[Abs[delta] < tol, Break[]];
  ];
  yval = yvaln;
  AppendTo[r, {time + h, yval}];
];
Return[r];
]

```

Algorithm 4.2. Backward Euler Method Using Newton's Method *To solve the initial value problem*

$$y' = f(t, y), y(t_0) = y_0$$

on an interval $[t_0, t_{max}]$ with a fixed step size h .

1. input: $f(t, y), t_0, y_0, h, t_{max}, tol$
2. output: (t_0, y_0)
3. let $t = t_0, y = y_0$
4. while $t < t_{max}$

(a) let $i = 0, y_n^{(0)} = y$

(b) Repeat:

$$y_n^{(i+1)} = y_n^{(i)} - \frac{y_n^{(i)} - y_{n-1} - hf(t, y_n^{(i)})}{1 - hf_y(t, y_n^{(i)})}$$

$$i = i + 1$$

until $|y_n^{(i+1)} - y_n^{(i)}| < tol$

(c) let $y = y_n^{(i+1)}$

(d) let $t = t + h$

(e) let $t_n = t, y_n = y$

(f) output: $(t_0, y_0), \dots, (t_n, y_n)$

4.4 Other Euler Methods

The **Trapezoidal Method** has an iteration formula

$$y_n = y_{n-1} + \frac{h_n}{2} (f(t_n, y_n) + f(t_{n-1}, y_{n-1})) \quad (4.113)$$

The local truncation error for the Trapezoidal Method is $O(h^2)$:

$$d_n = \frac{y(t_n) - y(t_{n-1})}{h} - \frac{1}{2} [f(t, y(t_n)) + f(t_{n-1}, y(t_{n-1}))] \quad (4.114)$$

Use Taylor series expansions about $t = t_n$,

$$y(t_{n-1}) = y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(t_n) - \frac{h^3}{6}y'''(t_n) + \dots \quad \Leftarrow \quad (4.115)$$

$$f(t_{n-1}, y(t_{n-1})) = y'(t_{n-1}) = y'(t_n) - hy''(t_n) + \frac{h^2}{2}y'''(t_n) \dots \quad (4.116)$$

Hence the local truncation error is

$$d_n = y'(t_n) - \frac{h}{2}y''(t_n) + \frac{h^2}{8}y'''(t_n) + \dots \quad (4.117)$$

$$\dots + \frac{1}{2} \left[y'(t_n) + y'(t_n) - hy''(t_n) + \frac{h^2}{2}y'''(t_n) + \dots \right] \quad (4.118)$$

$$= \frac{3h^2}{8}y'''(t_n) \quad (4.119)$$

$$= O(h^2) \quad (4.120)$$

For the test equation, the trapezoidal method gives

$$y_n = \frac{2 + h\lambda}{2 - h\lambda} y_{n-1} \quad (4.121)$$

Substituting $h\lambda = x + iy$ we find that the region of absolute stability is precisely the left-half plane $x \leq 0$.

The **theta method** is given by

$$y_n = y_{n-1} + h [\theta f(t_{n-1}, y_{n-1}) + (1 - \theta)f(t_n, y_n)] \quad (4.122)$$

The theta method is implicit except when $\theta = 1$, where it reduces to Euler's method. For $\theta = 1/2$ it becomes the trapezoidal method. The method is first order except when $\theta = 1/2$. The theta method can be used to eliminate error in specific terms in the Taylor expansion besides the lowest order term. For example, setting $\theta = 2/3$ gets rid of the $O(h^3)$ term in the error even though the $O(h^2)$ term remains. This could be of use in cases where the higher order term has a sufficiently high coefficient that for larger step sizes it overwhelms the lower order term. The theta-method is A-stable if and only if $0 \leq \theta \leq 1/2$.

The **midpoint method** is

$$y_n = y_{n-1} + h_n f \left(t_{n-1/2}, \frac{1}{2}[y_n + y_{n-1}] \right) \quad (4.123)$$

The midpoint method is second-order and A-stable.

The **modified Euler Method** is

$$y_n = y_{n-1} + \frac{h_n}{2} [f(t_{n-1}, y_{n-1}) + f(t_n, y_{n-1} + hf(t_{n-1}, y_{n-1}))] \quad (4.124)$$

Heun's Method is

$$y_n = y_{n-1} + \frac{h_n}{4} \left[f(t_{n-1}, y_{n-1}) + 3f \left(t_{n-1} + \frac{2}{3}h, y_{n-1} + \frac{2}{3}hf(t_{n-1}, y_{n-1}) \right) \right] \quad (4.125)$$

Both Heun's method and the modified Euler method are second order and are examples of two-step Runge-Kutta methods, which we shall discuss later. It is clearer to implement these in two "stages," eg., for the modified Euler method,

$$\tilde{y}_n = y_{n-1} + hf(t_{n-1}, y_{n-1}) \quad (4.126)$$

$$y_n = y_{n-1} + \frac{h_n}{2} [f(t_{n-1}, y_{n-1}) + f(t_n, \tilde{y}_n)] \quad (4.127)$$

while for Heun's method,

$$\tilde{y}_n = y_{n-1} + \frac{2}{3}hf(t_{n-1}, y_{n-1}) \quad (4.128)$$

$$y_n = y_{n-1} + \frac{h_n}{4} \left[f(t_{n-1}, y_{n-1}) + 3f \left(t_{n-1} + \frac{2}{3}h, \tilde{y}_n \right) \right] \quad (4.129)$$

Chapter 5

Runge-Kutta Methods

5.1 Taylor Series Methods

We begin by expanding y_n as a Taylor series about t_{n-1} ,

$$y_n = y_{n-1} + hy'_{n-1} + \frac{h^2}{2}y''_{n-1} + \cdots + \frac{h^p}{p!}y^{(p)}_{n-1} + \cdots \quad (5.1)$$

Using the chain rule for derivatives, and denoting $f(t_{n-1}, y_{n-1})$ by f , we have

$$y' = f \quad (5.2)$$

$$y'' = \frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{dy}{dt} \quad (5.3)$$

$$= f_t + f_y f = F \quad (5.4)$$

$$y''' = \frac{d}{dt}(f_t + f_y f) = \frac{\partial}{\partial t}(f_t + f_y f) + \frac{\partial}{\partial y}(f_t + f_y f) \frac{dy}{dt} \quad (5.5)$$

$$= f_{tt} + f_{yt}f + f_y f_t + (f_{yt} + f_{yy}f + f_y f_y)f \quad (5.6)$$

$$= f_{tt} + 2f_{yt}f + f_y f_t + f_{yy}f^2 + f_y^2 f \quad (5.7)$$

$$= f_y F + G \quad (5.8)$$

where

$$F = f_t + f_y f$$

and

$$G = f_{tt} + 2f_{yt}f + f_{yy}f^2$$

Hence

$$y_n = y_{n-1} + hf + \frac{h^2}{2}(f_t + f_y f) + \quad (5.9)$$

$$\frac{h^3}{6}(f_{tt} + 2f_{yt}f + f_y f_t + f_{yy}f^2 + f_y^2 f) + \cdots \quad (5.10)$$

Thus we can define a sequence of methods, depending where we truncate the series. A Taylor method that includes terms through h^p therefore has a local truncation

error of h^p . We also observe that $O(h)$ Taylor method is equivalent to Euler's method.

The second order Taylor method has

$$y_n = y_{n-1} + hf + \frac{h^2}{2}(f_t + f_y f) \quad (5.11)$$

For the test equation $y' = \lambda y$ we have $f = \lambda y$, $f_t = 0$, $f_y = \lambda$, and hence

$$y_n = y_{n-1} + h\lambda y_n + \frac{h^2}{2}\lambda^2 y_n \quad (5.12)$$

$$= \left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right) y_{n-1} \quad (5.13)$$

Absolute stability occurs when

$$\left|\frac{y_n}{y_{n-1}}\right| = \left|1 + h\lambda + \frac{(h\lambda)^2}{2}\right| \leq 1 \quad (5.14)$$

To find the shape of this region, set $z = h\lambda = re^{i\theta}$, and square both sides of equation 5.14 using the identity $|z|^2 = zz^*$ to get

$$\left(1 + re^{i\theta} + \frac{r^2 e^{2i\theta}}{2}\right) \left(1 + re^{-i\theta} + \frac{r^2 e^{-2i\theta}}{2}\right) \leq 1 \quad (5.15)$$

$$1 + re^{-i\theta} + \frac{r^2 e^{-2i\theta}}{2} + re^{i\theta} + r^2 + r^3 e^{-i\theta} \frac{r^2 e^{2i\theta}}{2} + r^3 e^{i\theta} + \frac{r^4}{4} \leq 1 \quad (5.16)$$

$$2r \left(\frac{e^{-i\theta} + e^{i\theta}}{2}\right) + r^2 \frac{e^{-2i\theta} + e^{2i\theta}}{2} + r^3 \frac{e^{-i\theta} + e^{i\theta}}{2} + r^2 + \frac{r^4}{4} \leq 0 \quad (5.17)$$

$$2r \cos \theta + r^2 \cos 2\theta + r^3 \cos \theta + r^2 + \frac{r^4}{4} \leq 0 \quad (5.18)$$

To plot this equation pick some small interval $\delta\theta$ and do the following:

- Set $\theta = 0$
- Repeat
 1. Set $\theta = \theta + \delta\theta$
 2. Solve 5.18 for r , accepting only real solutions
 3. Plot the point (r, θ)

until $\theta = 2\pi$

In Mathematica we can use `Solve` to find the solution of 5.18. For example, to find $r(\pi)$,

```
f[r_, theta_] := 2r * Cos[theta] + r^2 * Cos[2*theta] + r^3 Cos[theta]
+ 0.25* r^4 + r^2;
Solve[f[r, Pi]==0]
```

which returns a list of rules for r ,

```
{r->0.},{r->1.-1.73205i}, {r->1.+1.73205i}, {r->2.}}
```

To convert this to a list of numbers, replace the `Solve` with

```
r/.Solve[f[r,Pi]==0]
```

This returns

```
{0.,1.-1.73205i, 1.+1.73205i, 2.}
```

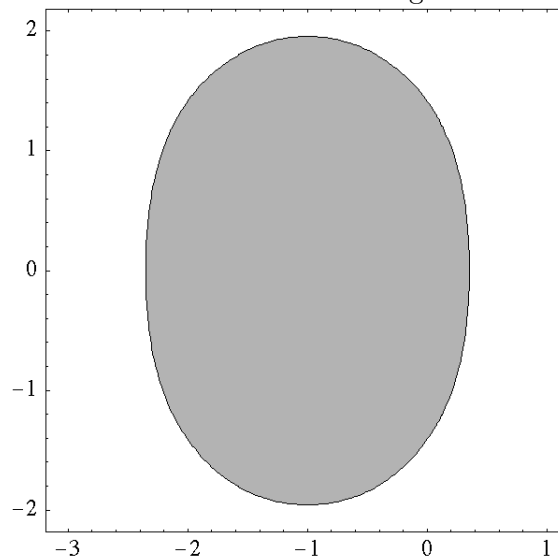
To get only the real roots,

```
Select[r/.Solve[f[r,Pi]==0], And[Im[#] == 0, Re[#] >= 0]&]
```

which now returns the list

```
{0., 2.}
```

Figure 5.1: Region of absolute stability for second-order Taylor series methods. Numerical solutions with $h\lambda$ that fall inside this region are absolutely stable



Putting the whole plotting algorithm together, the following will plot 100 points on the solution.

```
points = {};
Cartesian[{r_, theta_}] := {r*Cos[theta], r*Sin[theta]};
For[theta = 0, theta <= 2Pi, theta += Pi/100,
point = Select[
  r /. Solve[f[r, theta] == 0, r],
  And[Im[#]==0, Re[#] >= 0]&];
```

```

point = Cartesian[{#, theta}] & /@ point;
points = Join[points, point];
];
ListPlot[points];

```

An easier way to do this in Mathematica is to use `ContourPlot`. The following code block will also automatically fill in the space that corresponds to values of z that are less than 1 with a 70% gray shading.

```

ContourPlot[g[x, y], {x, -3.1, 1.1}, {y, -2.1, 2.1},
  PlotPoints -> 100,
  Contours -> {1}, (* only draw one contour line, at z = 1 *)
  PlotRange -> {0, 1}, (* only plot the contours in the range
    0 ≤ z ≤ 1 *)
  ColorFunction -> ( GrayLevel[.3# + .7] &)
]

```

5.2 Numerical Quadrature

Since the initial value problem

$$y' = f(t, y), \quad y_{t_0} = y_0 \quad (5.19)$$

is equivalent to the integral equation

$$y = y_0 + \int_{t_0}^t f(s, y(s)) ds \quad (5.20)$$

we can formulate a numerical method by substituting any desired numerical approximation of the integral and integrating to get an iteration formula. For example, if we approximate

$$f(s, y(s)) \approx f(t_0, y(t_0)) \quad (5.21)$$

we obtain Euler's Method:

$$y \approx y_0 + (t - t_0)f(t_0, y_0) \quad (5.22)$$

(Replace y with y_n and y_0 with y_{n-1} to get the iteration formula). On the other hand, if we use $f(s, y(s)) \approx f(t, y(t))$ we get the backward's Euler method,

$$y(t) \approx y_0 + f(t, y(t)) \quad (5.23)$$

If we approximate the integral by its midpoint

$$f(t, s) \approx f\left(\frac{t-t_0}{2}, f\left(\frac{t-t_0}{2}, y\left(\frac{t-t_0}{2}\right)\right)\right) \quad (5.24)$$

we obtain

$$y(t) \approx y_0 + (t - t_0)f\left(\frac{t-t_0}{2}, f\left(\frac{t-t_0}{2}, y\left(\frac{t-t_0}{2}\right)\right)\right) \quad \Leftarrow \quad (5.25)$$

which gives the iteration formula

$$y_n = y_{n-1} + hf(t_{n-1/2}, f(t_{n-1/2}, y_{n-1/2})) \quad (5.26)$$

We can estimate $y_{n+1/2}$ using the forward Euler method:

$$y_{n-1/2} = y_{n-1} + \frac{h}{2}f(t_{n-1}, y_{n-1}) \quad \Leftarrow \quad (5.27)$$

Equations 5.26 and 5.27 define the explicit **Midpoint Method**.

We obtain the **implicit Trapezoidal Rule** by approximating

$$\int_{t_0}^t f(s, y(s))ds \approx \frac{t-t_0}{2}(f(t_0, y(t_0)) + f(t, y)) \quad (5.28)$$

The corresponding iteration formula is then

$$y_n = y_{n-1} + \frac{h}{2}(f(t_{n-1}, y_{n-1}) + f(t_n, y_n)) \quad (5.29)$$

If we also approximate $f(t_n, y_n)$ with Euler's method, we get a two-stage **explicit Trapezoidal Rule**

$$\hat{y}_n = y_{n-1} + hf(t_{n-1}, y_{n-1}) \quad (5.30)$$

$$y_n = y_{n-1} + \frac{h}{2}(f(t_{n-1}, y_{n-1}) + f(t_n, \hat{y}_n)) \quad (5.31)$$

In general one can fit a polynomial through n points $(t_i, y_i), i = 1, 2, \dots, n$ using the **Lagrange interpolation formula**,

$$P(t) = \sum_{i=0}^n \prod_{j=0, j \neq i}^n \frac{t-t_j}{t_i-t_j} y_i \quad (5.32)$$

Using these Lagrange polynomials one obtains a general quadrature formula, known as the **open Newton-Cotes formula**, which is given by

$$\int_a^b f(t)dt = \sum_{i=0}^n na_i f(t_i) \quad (5.33)$$

where

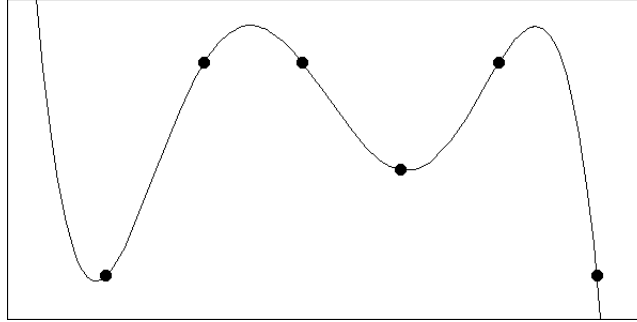
$$a_i = \int_{t_0}^{t_n} \prod_{j=0, j \neq i}^n \frac{t-t_j}{t_i-t_j} dt \quad (5.34)$$

Gaussian Quadrature seeks to find the best numerical estimate of the integral by optimizing the choice of mesh points (t_1, t_2, \dots, t_n) at which the integrand is evaluated. The best estimate is

$$\int_a^b f(t)dt = \sum_{i=1}^n w_i(t) f(t_i) \quad (5.35)$$

where $w_i(t)$ is a **weight function**, an orthogonal polynomial on the interval $[a, b]$. The **Legendre orthogonal polynomials** $p_0(t), p_1(t), \dots$, for example, have the following properties:

Figure 5.2: Polynomial interpolation through a set of points using the Lagrange method.



- Polynomial $p_i(t)$ is a polynomial of degree i ;
- $\langle p_i(t), p_j(t) \rangle = \delta_{ij}$ where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise, and

$$\langle f, g \rangle = \int_{-1}^1 f(t)g(t)dt \quad (5.36)$$

- $p_0(t) = 1, p_1(t) = t$, and

$$(i + 1)P_{i+1}(t) = (2i + 1)tP_i(t) - iP_{i-1}(t) \quad (5.37)$$

- If $q(t)$ is any polynomial of degree less than $2n$ and t_1, t_2, \dots are the roots of the n^{th} Legendre Orthogonal Polynomial $p_n(t)$ then

$$\int_{-1}^1 q(t)dt = \sum_{i=1}^n w_i q(t_i) \quad (5.38)$$

where

$$w_i = \int_{-1}^1 P(t)dt \quad (5.39)$$

where $P(t)$ is the n^{th} Lagrange Interpolating Polynomial (equation 5.32).

The problem can be translated from any interval $[a, b]$ (which corresponds, say, to $[t_0, t]$), to the interval $[-1, 1]$ by the transformation

$$\int_a^b f(t)dt = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) dt \quad (5.40)$$

Gaussian quadrature has the highest precision possible for polynomial quadrature, and it forms the basis of Implicit Runge Kutta methods.¹

¹For more details and proofs of these observations see [6], section 4.7 and [13], section 3.1.

5.3 Traditional Runge-Kutta Methods

Simpson's quadrature rule gives

$$\int_{t_0}^t f(s, y(s)) ds \approx \frac{t-t_0}{6} \left[f(t_0, y_0) + 4f\left(\frac{t+t_0}{2}, y\left(\frac{t+t_0}{2}\right)\right) + f(t, y(t)) \right] \quad (5.41)$$

and therefore

$$y_n = y_{n-1} + \frac{h}{6} [f(t_{n-1}, y_{n-1}) + 4f(t_{n-1/2}, y(t_{n-1/2})) + f(t_n, y_n)] \quad (5.42)$$

We can derive a three-stage explicit method by setting

$$k_1 = f(t_{n-1}, y_{n-1}) \quad (5.43)$$

$$k_2 = y_{n-1} + \frac{h}{2} f(t_{n-1/2}, k_1) \quad (5.44)$$

$$k_3 = y_{n-1} + h f(t_n, k_2) \quad (5.45)$$

$$y_n = y_{n-1} + \frac{h}{6} (k_1 + 4k_2 + k_3) \quad (5.46)$$

A better approximation performs a second round of function evaluations:

$$y_n = y_{n-1} + \frac{h}{6} (f(t_{n-1}, k_1) + 4f(t_{n-1/2}, k_2) + f(t_n, k_3)) \quad \Leftarrow \quad (5.47)$$

We obtain the **traditional explicit 4-stage Runge-Kutta Method** by splitting up the term in the center:

$$y_n = y_{n-1} + \frac{h}{6} [f(t_{n-1}, y_{n-1}) + 2f(t_{n-1/2}, y(t_{n-1/2})) + \quad (5.48)$$

$$2f(t_{n-1/2}, y(t_{n-1/2})) + f(t_n, y_n)] \quad (5.49)$$

The iteration formulas are

$$k_1 = y_{n-1} \quad (5.50)$$

$$k_2 = y_{n-1} + \frac{h}{2} f(t_{n-1}, k_1) \quad (5.51)$$

$$k_3 = y_{n-1} + \frac{h}{2} f(t_{n-1/2}, k_2) \quad (5.52)$$

$$k_4 = y_{n-1} + h f(t_{n-1/2}, k_3) \quad (5.53)$$

$$y_n = y_{n-1} + \frac{h}{6} (f(t_{n-1}, k_1) + 2f(t_{n-1/2}, k_2) + 2f(t_{n-1/2}, k_3) + f(t_n, k_4)) \quad (5.54)$$

For the test equation,

$$k_1 = y_{n-1} \quad (5.55)$$

$$k_2 = y_{n-1} + \frac{h}{2}(\lambda k_1) \quad (5.56)$$

$$= y_{n-1} + \frac{h\lambda}{2}y_{n-1} \quad (5.57)$$

$$= \left(1 + \frac{h\lambda}{2}\right)y_{n-1} \quad (5.58)$$

$$k_3 = y_{n-1} + \frac{h}{2}(\lambda k_2) \quad (5.59)$$

$$= y_{n-1} + \frac{h\lambda}{2}\left(1 + \frac{h\lambda}{2}\right)y_{n-1} \quad (5.60)$$

$$= \left(1 + \frac{h\lambda}{2} + \frac{h^2\lambda^2}{4}\right)y_{n-1} \quad (5.61)$$

$$k_4 = y_{n-1} + h(\lambda k_3) \quad (5.62)$$

$$= y_{n-1} + h\lambda\left(1 + \frac{h\lambda}{2} + \frac{h^2\lambda^2}{4}\right)y_{n-1} \quad (5.63)$$

$$= \left(1 + h\lambda + \frac{h^2\lambda^2}{2} + \frac{h^3\lambda^3}{4}\right)y_{n-1} \quad (5.64)$$

Hence

$$y_n = y_{n-1} + \frac{h}{6}(\lambda k_1 + 2\lambda k_2 + 2\lambda k_3 + \lambda k_4) \quad (5.65)$$

$$= \left[1 + \frac{h\lambda}{6}\left(1 + 2\left(1 + \frac{h\lambda}{2}\right) + 2\left(1 + \frac{h\lambda}{2} + \frac{h^2\lambda^2}{4}\right)\right.\right. \quad (5.66)$$

$$\left.\left. + \left(1 + h\lambda + \frac{h^2\lambda^2}{2} + \frac{h^3\lambda^3}{4}\right)\right)\right]y_{n-1} \quad (5.67)$$

$$= \left[1 + h\lambda + \frac{h^2\lambda^2}{2} + \frac{h^3\lambda^3}{6} + \frac{h^4\lambda^4}{24}\right]y_{n-1} \quad (5.68)$$

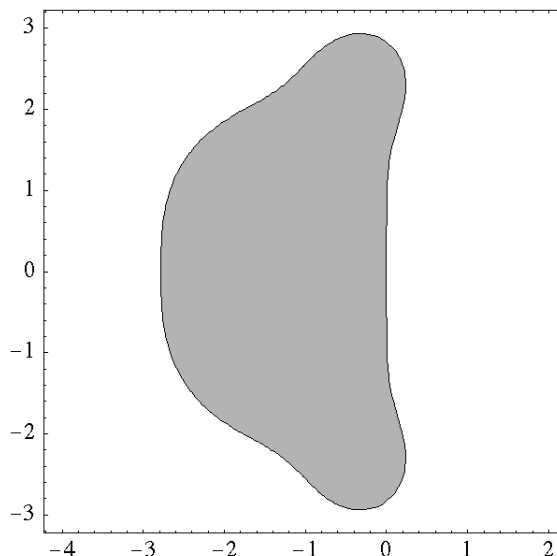
In other words the RK-4 method reproduces a 4-term Taylor expansion for the test equation. Absolute stability requires

$$1 \geq \left|1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}\right|^2 \quad (5.69)$$

$$= \left[1 + re^{i\theta} + \frac{re^{2i\theta}}{2} + \frac{re^{3i\theta}}{6} + \frac{re^{4i\theta}}{24}\right] \times \quad (5.70)$$

$$\left[1 + re^{-i\theta} + \frac{re^{-2i\theta}}{2} + \frac{re^{-3i\theta}}{6} + \frac{re^{-4i\theta}}{24}\right] \quad (5.71)$$

Figure 5.3: Region of absolute stability for the “classical” 4-stage Runge-Kutta method.



Using the trigonometric identity $2 \cos \theta = e^{i\theta} + e^{-i\theta}$,

$$1 \geq \frac{2}{576} (24r^4 \cos 4\theta + (24)(4 + r^2) \cos 3\theta + \quad (5.72)$$

$$12(24 + 8r^2 + r^4) \cos 2\theta + \quad (5.73)$$

$$4(144 + 72r^2 + 12r^4 + r^6) \cos \theta + \quad (5.74)$$

$$(576 + 576r^2 + 144r^4 + 16r^6 + r^8)) \quad (5.75)$$

The region of absolute stability is illustrated in figure 5.3.

Figure 5.4 compares the absolute error $(y_n(1) - e)$ at $t = 1$ using the 4-stage Runge-Kutta and Euler’s method. Clearly the RK4 method, which is fourth order, has much lower error and the error improves much faster with small step size than does Euler.

Example 5.1. Compute the solution to the test equation $y' = y, y(0) = 1$ on $[0, 1]$ using the 4-stage Runge Kutta method with $h = 1/2$.

Solution. Since we start at $t = 0$ and need to compute through $t = 1$ we have to compute two iterations of RK. For the first iteration,

$$k_1 = y_0 = 1 \quad (5.76)$$

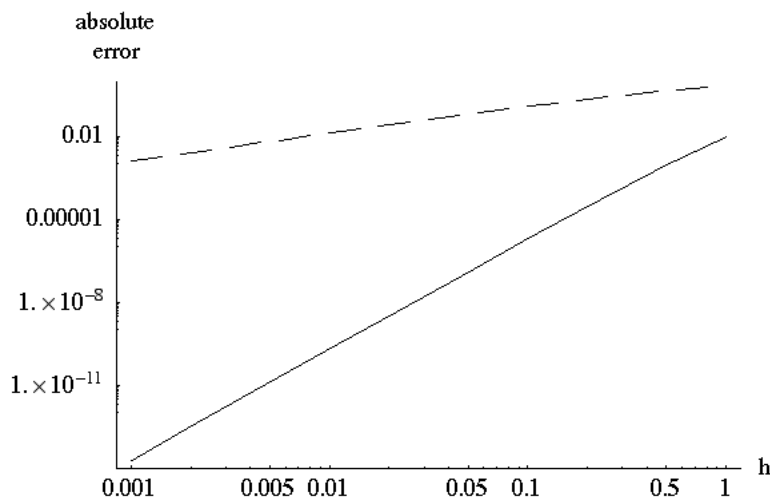
$$k_2 = y_0 + \frac{h}{2} f(t_0, k_1) \quad (5.77)$$

$$= 1 + (0.25)(1) = 1.25 \quad (5.78)$$

$$k_3 = y_0 + \frac{h}{2} f(t_{1/2}, k_2) \quad (5.79)$$

$$= 1 + (0.25)(1.25) = 1.3125 \quad (5.80)$$

Figure 5.4: Comparison of absolute error at $t = 1$ as a function of step size for the test problem. Dashed Line: Euler's Method; Solid Line: Runge-Kutta 4-stage method



$$k_4 = y_0 + hf(t_{1/2}, k_3) \quad (5.81)$$

$$= 1 + (0.5)(1.3125) = 1.65625 \quad (5.82)$$

$$y_1 = y_0 + \frac{h}{6}(f(t_0, k_1) + 2f(t_{1/2}, k_2) + 2f(t_{1/2}, k_3) + f(t_1, k_4)) \quad (5.83)$$

$$= y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (5.84)$$

$$= 1 + \frac{.5}{6}1 + 2(1.25) + 2(1.3125) + 1.65625 = 1.64844 \quad (5.85)$$

Thus the numerical approximation to $y(0.5)$ is $y_1 \approx 1.64844$. For the second step,

$$k_1 = y_1 = 1.64844 \quad (5.86)$$

$$k_2 = y_1 + \frac{h}{2}f(t_1, k_1) \quad (5.87)$$

$$= 1.64844 + (0.25)(1.64844) = 2.06055 \quad (5.88)$$

$$k_3 = y_1 + \frac{h}{2}f(t_{1.5}, k_2) \quad (5.89)$$

$$= 1.64844 + (0.25)(2.06055) = 2.16358 \quad (5.90)$$

$$k_4 = y_1 + hf(t_{1.5}, k_3) \quad (5.91)$$

$$= 1.64844 + (0.5)(2.16358) = 2.73023 \quad (5.92)$$

$$y_2 = y_1 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (5.93)$$

$$= 1.64844 + \frac{.5}{6}1.64844 + 2(2.06055) + 2(2.16358) + 2.73023 = 2.71735 \quad (5.94)$$

This gives us a numerical approximation of $y(1) \approx 2.71735$, and error of approximately 0.034% (the exact value is $e \approx 2.71828$). By comparison, a forward Euler computation with the same step size will yield a numerical result of 2.25, an error approximately 17%. \square

Since it is an explicit method, the Runge-Kutta 4-stage method is very easy to implement in a computer, even though calculations are very tedious to do by hand. Here is a Mathematica implementation.

```

RK4[f_, {t0_, y0_}, h_, tmax_] := Module[
  {k1, k2, k3, k4, t, yval, r},
  r = {{t0, y0}};
  yval = y0;
  For[t = t0, t < tmax, t += h,
    k1 = yval;
    k2 = yval + (h/2) f[t, k1];
    k3 = yval + (h/2) f[t + h/2, k2];
    k4 = yval + h f[t + h/2, k3];
    yval = yval + (h/6) *(f[t, k1] + 2f[t + h/2, k2] + 2 f[t
      + h/2, k3] + f[t + h, k4]);
    AppendTo[r, {t + h, yval}];
  ];
  Return[r];
]

```

5.4 General Form of Runge-Kutta Methods

The general form of the **s-stage Runge-Kutta Method** is

$$y_n = y_{n-1} + h \sum_{i=1}^s b_i K_i \quad (5.95)$$

where

$$K_i = f \left(t_{n-1} + c_i h, y_{n-1} + h \sum_{j=1}^s a_{ij} K_j \right) \quad (5.96)$$

and

$$c_i = \sum_{j=1}^s a_{ij} \quad (5.97)$$

The coefficients a_{ij}, b_j are chosen by comparing the method with a Taylor series expansion and choosing values that cancel specific terms in the error. There are many

different Runge-Kutta methods; individual ones are usually described by presenting their **Butcher Array**,

$$\begin{array}{c|cccc}
 c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
 c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
 \vdots & \vdots & & & \vdots \\
 c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
 \hline
 & b_1 & b_2 & \cdots & b_s
 \end{array} \tag{5.98}$$

The c_i are the row-sums of the matrix. For explicit methods, the Butcher array is strictly lower triangular. Thus it is common to omit the upper-diagonal terms when writing the Butcher array.

An equivalent formulation to equations 5.95 and 5.96 is given by

$$Y_i = y_{n-1} + h \sum_{j=1}^s a_{ij} f(t_{n-1} + c_j h, Y_j) \tag{5.99}$$

$$y_n = y_{n-1} + h \sum_{i=1}^s b_i f(t_{n-1} + c_i h, Y_i) \tag{5.100}$$

Example 5.2. Show that Euler's method is specified by the following Butcher array:

$$\begin{array}{c|c}
 0 & 0 \\
 \hline
 & 1
 \end{array} \tag{5.101}$$

Solution. We have $s = 1$, $a_{11} = 0$, $c_1 = 0$, and $b_1 = 0$. Hence

$$K_1 = f(t_{n-1} + c_1 h, y_{n-1} + h a_{11} K_1) \tag{5.102}$$

$$= f(t_{n-1}, y_{n-1}) \tag{5.103}$$

$$y_n = y_{n-1} + h b_1 K_1 \tag{5.104}$$

$$= y_{n-1} + h f(t_{n-1}, y_{n-1}) \tag{5.105}$$

which reproduces Euler's method. \square

\implies The most general two-stage explicit method takes the form

$$\begin{array}{c|cc}
 c_1 & 0 & \\
 c_2 & a_{21} & 0 \\
 \hline
 & b_1 & b_2
 \end{array} \tag{5.106}$$

Since the c_i are row sums this simplifies to

$$\begin{array}{c|cc}
 0 & 0 & \\
 a & a & 0 \\
 \hline
 & b_1 & b_2
 \end{array} \tag{5.107}$$

and hence the most general form of the iteration formula is

$$K_1 = f(t_{n-1} + c_1 h, y_{n-1} + ha_{11}K_1 + ha_{12}K_2) \quad (5.108)$$

$$= f(t_{n-1}, y_{n-1}) = f \quad (5.109)$$

$$K_2 = f(t_{n-1} + c_2 h, y_{n-1} + ha_{21}K_1 + ha_{22}K_2) \quad (5.110)$$

$$= f(t_{n-1} + ah, y_{n-1} + haK_1) \quad \Leftarrow \quad (5.111)$$

$$= f(t_{n-1} + ah, y_{n-1} + haf) \quad \Leftarrow \quad (5.112)$$

$$y_n = y_{n-1} + hb_1K_1 + hb_2K_2 \quad (5.113)$$

$$= y_{n-1} + hb_1f + hb_2f(t_{n-1} + ah, y_{n-1} + haf) \quad (5.114)$$

where we have use the shorthand notation $f = f(t_{n-1}, y_{n-1})$. To determine the coefficients to minimize error, we expand the last term in a Taylor series about (t_{n-1}, y_{n-1})

$$f(t_{n-1} + ah, y_{n-1} + haf) = f + ahf_t + haf f_y + \quad (5.115)$$

$$\frac{a^2 h^2}{2} (f_{tt} + 2f f_{ty} + f^2 f_{yy} + f_y(f_t + f f_y)) + O(h^3) \quad (5.116)$$

To simplify the notation we let (after Lambert) $F = f_t + f f_y$ and $G = f_{tt} + 2f f_{ty} + f^2 f_{yy}$

$$y_n = y_{n-1} + hb_1f + hb_2 \left(f + ahf_t + haf f_y + \frac{a^2 h^2}{2} (G + f_y F) \right) \quad (5.117)$$

$$= y_{n-1} + h(b_1 + b_2)f + h^2 b_2 a F + O(h^3) \quad (5.118)$$

Hence we need

$$b_1 + b_2 = 1 \quad (5.119)$$

$$b_2 a = \frac{1}{2} \quad (5.120)$$

to match a Taylor expansion. This gives us a family of two stage explicit methods that are second order:

$$\begin{array}{c|cc} 0 & 0 & \\ a & a & 0 \\ \hline & 1 - \frac{1}{2a} & \frac{1}{2a} \end{array} \quad (5.121)$$

When $a = 1$, this gives the trapezoidal method, and when $a = 1/2$, it gives the explicit midpoint method.

Two popular explicit three-stage third-order methods include **Heun's third-order method**

$$\begin{array}{c|ccc} 0 & 0 & & \\ 1/3 & 1/3 & 0 & \\ 2/3 & 0 & 2/3 & 0 \\ \hline & 1/4 & 0 & 3/4 \end{array} \quad (5.122)$$

and **Kutta's third-order formula**

$$\begin{array}{c|ccc}
 0 & 0 & & \\
 1/2 & 1/2 & 0 & \\
 1 & -1 & 2 & 0 \\
 \hline
 & 1/6 & 2/3 & 1/6
 \end{array} \tag{5.123}$$

The **classical Runge-Kutta method** is the four-stage fourth-order method we have already met:

$$\begin{array}{c|cccc}
 0 & 0 & & & \\
 1/2 & 1/2 & 0 & & \\
 1/2 & 0 & 1/2 & 0 & \\
 1 & 0 & 0 & 1 & 0 \\
 \hline
 & 1/6 & 1/3 & 1/3 & 1/6
 \end{array} \tag{5.124}$$

The region of absolute stability for all ERK methods will always be bounded, and hence none of them can be A-stable. This is because the test equation will always produce

$$y_n = P(z)y_{n-1} \tag{5.125}$$

where $P(z)$ is a polynomial. There will always be sufficiently large values of z so that $|P(z)| > 1$. Since none of these methods are A-stable, they are generally not good for stiff problems, as we have already seen for the Forward Euler Method (which is a one-stage IRK). Thus one commonly uses implicit RK methods (IRK) instead.

5.5 Order Conditions

Explicit Runge-Kutta (ERK) methods have an order that is at most equal to the number of stages; we have seen two, three, and four stage methods that have orders equal to the number of stages. However, for $n > 4$, it is not possible to find an ERK with an order that is equal to the number of stages. The best one can do is the following.

Number of Stages	1	2	3	4-5	6	7-8	9-10	11	12	13-17
Best Possible Order	1	2	3	4	5	6	7	8	9	10

Unfortunately the proof of this statement is tedious - there is an explosion in the number of terms that must be compared as the Taylor order is increased. Interested students are referred to [7] for details.

To determine the error in carrying out a single step of a Runge-Kutta method, one must compare the successive terms in a Taylor series expansion of the exact and computed solutions. This calculation is quite tedious: "The reader is not asked to take a deep breath, take five sheets of reversed computer paper, remember the basic rules of differential calculus, and begin the following computations..." (see [10, page 144]). The result is a set of **order conditions** that are *necessary* if a method is to have a given order; however, these conditions are not, in general, sufficient.

The complete derivation of the order conditions makes use of graph-theoretic methods that are beyond the scope of this presentation (see [7]). Instead, we quote some of the simpler results. For example, a method must satisfy the following if it is of order p :

$$\mathbf{b}^T C^{k-1} \mathbf{1} = \frac{1}{k}, k = 1, 2, \dots, p \quad (5.126)$$

$$\mathbf{b}^T A^{k-1} \mathbf{1} = \frac{1}{k!}, k = 1, 2, \dots, p \quad (5.127)$$

where $C = \text{diagonal}(c_1, \dots, c_s)$, $\mathbf{b}^T = (b_1, \dots, b_s)$, A is the matrix of coefficients a_{ij} , and s is the number of stages. Since $\sum_k a_{jk} = c_j$ we have the additional condition $A\mathbf{1} = \mathbf{c}$, where $\mathbf{c} = (c_1, \dots, c_s)^T$. Expanding to individual components we have the following order conditions. For order 1,

$$\mathbf{b}^T \mathbf{1} = \sum_i b_i = 1 \quad (5.128)$$

For order 2, in addition to the order 1 conditions, we must have

$$\mathbf{b}^T \mathbf{c} = \sum_i b_i c_i = \frac{1}{2} \quad (5.129)$$

For order 3 we must also have the following conditions:

$$\mathbf{b}^T C \mathbf{c} = \sum_i b_i c_i^2 = \frac{1}{3} \quad (5.130)$$

$$\mathbf{b}^T A \mathbf{c} = \sum_{i,j} b_i a_{ij} c_j = \frac{1}{6} \quad (5.131)$$

The upper limit of these sums in 5.128 through 5.131 is the number of stages, so there are different conditions depending on the number of stages in the methods. The formulas have been automated in the Mathematica package `NumericalMath`Butcher`` as the function `RungeKuttaOrderConditions`. To access this function enter

```
<<NumericalMath`Butcher`
```

Then the expression

```
RungeKuttaOrderConditions[4,s]
```

will give a list of formulas in summation form that an s -stage method must satisfy to be fourth order, while

```
RungeKuttaOrderConditions[4]
```

gives a list of the conditions in matrix form that a method of any number of stages must satisfy to be fourth order (see figure 5.5). The number of conditions increases exponentially with the order of the method as we can see with

```
Plus @@ Length /@ RungeKuttaOrderConditions[#] & /@ Range[15]
```

which returns the list

```
{1, 2, 4, 8, 17, 37, 85, 200, 486, 1205, 3047, 7813, 20299, 53272,
 141083}
```

Figure 5.5: The function `RungeKuttaOrderConditions` in Mathematica.

```

In[1]:= << NumericalMath`Butcher`
In[2]:= RungeKuttaOrderConditions[4, s]
Out[2]= {{Sum[b_i, {i, 1, s}] == 1}, {Sum[b_i c_i, {i, 1, s}] == 1/2},
{Sum[b_i Sum[a_{i,j}, {j, 1, s}] c_j, {i, 1, s}] == 1/6, Sum[b_i c_i^2, {i, 1, s}] == 1/3},
{Sum[b_i Sum[a_{i,j}, {j, 1, s}] Sum[a_{j,k}, {k, 1, s}] c_k, {i, 1, s}] == 1/24, Sum[b_i Sum[a_{i,j}, {j, 1, s}] c_j^2, {i, 1, s}] == 1/12},
{Sum[b_i c_i Sum[a_{i,j}, {j, 1, s}] c_j, {i, 1, s}] == 1/8, Sum[b_i c_i^3, {i, 1, s}] == 1/4}}

In[3]:= RungeKuttaOrderConditions[4]
Out[3]= {{b.e == 1}, {b.c == 1/2},
{b.a.c == 1/6, b.c^2 == 1/3}, {b.a.a.c == 1/24,
b.a.c^2 == 1/12, b.(c a.c) == 1/8, b.c^3 == 1/4}}

```

5.6 Step Size Control for ERK Methods

Computing the local truncation error for a Runge-Kutta method is quite unwieldy and requires the analytic calculation of numerous partial derivatives, making it impractical for an efficient implementation. Instead, we are forced to use an approximation of the error. Suppose we have computed two different solutions: y , of order p , and z , of order $p + 1$. Then

$$y_n = y(t_n) + ch^{p+1} + O(h^{p+2}) \quad (5.132)$$

$$z_n = y(t_n) + O(h^{p+2}) \quad (5.133)$$

for some constant c that depends on the method but is independent of h . Subtracting,

$$ch^{p+1} = |y_n - z_n| \quad (5.134)$$

The idea is that we have some tolerance ϵ that we do not want to exceed, and when $|y_n - z_n|$ approaches ϵ , we choose a new step size that satisfies

$$\frac{|y_n - z_n|_{old}}{h_{old}^{p+1}} \approx c \approx \frac{|y_n - z_n|_{new}}{h_{new}^{p+1}} < \frac{\nu\epsilon}{h_{new}^{p+1}} \quad (5.135)$$

where ν is a safety fraction, say 50%, giving the level of comfort or closeness we want to allow the error estimate to approach ϵ . Then we choose any h_{new} satisfying

$$h_{new} < h_{old} \left(\frac{\nu\epsilon}{|y_n - z_n|_{old}} \right)^{\frac{1}{p+1}} \quad (5.136)$$

Again, when the error becomes significantly smaller than $\nu\epsilon$ the process can be reversed, and the step size increased. Or the step size can be adjusted at each calculation to equal the right hand side of 5.136.

The problem with this technique is that one needs to perform two calculations, one at each of two different orders. This could potentially double the computation time unless we can find an **embedded method** – that is, one that has a lower-level computation built into it as part of the higher level computation. If the original and embedded methods are given by

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} \quad (5.137)$$

and

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \hat{\mathbf{b}}^T \end{array} \quad (5.138)$$

respectively, then we use the shorter notation

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \\ \hline & \hat{\mathbf{b}}^T \end{array} \quad (5.139)$$

to describe them both together. For example, the forward Euler is embedded in the modified trapezoidal:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array} \quad (5.140)$$

One frequently implemented method is the six-stage **Runge-Kutta-Fehlberg** which has a 4-th order method embedded within a fifth-order method:

$$\begin{array}{c|cccccc} 0 & & & & & & \\ 1/4 & 1/4 & & & & & \\ 3/8 & 3/32 & 9/32 & & & & \\ 12/13 & 1932/2197 & -7200/2197 & 7296/2197 & & & \\ 1 & 439/216 & -8 & 3680/513 & -845/4104 & & \\ 1/2 & -8/27 & 2 & -3544/2565 & 1859/4104 & -11/40 & \\ \hline & 25/216 & 0 & 1408/2565 & 2197/4104 & -1/5 & 0 \\ \hline & 16/135 & 0 & 6656/12825 & 28561/56430 & -9/50 & 2/55 \end{array} \quad (5.141)$$

While one could use an arbitrary step size for the first step, some knowledge of the problem is still necessary to prevent a really bad choice. An automatic calculation of the first value of h is provided by the following algorithm, as presented in [10].

1. Input ϵ_{abs} and ϵ_{rel} , the acceptable absolute and relative tolerances, and calculate

$$\sigma = \epsilon_{abs} + |y_0|\epsilon_{rel} \quad (5.142)$$

$$d_0 = \|y_0\| \quad (5.143)$$

$$d_1 = \|f(t_0, y_0)\| \quad (5.144)$$

where $\|u\| = \sqrt{\frac{1}{n} \sum \frac{u_i^2}{\sigma}}$ (for scalar problems this becomes $\|u\| = |u|/\sqrt{\sigma}$).

2. Let $h_0 = 0.01(d_0/d_1)$ and calculate

$$y_1 = y_0 + h_0 f(t_0, y_0) \quad (5.145)$$

3. Calculate $f(t_0 + h_0, y_1)$ and let

$$d_2 = \frac{\|f(t_0 + h_0, y_1) - f(t_0, y_0)\|}{h_0} \quad (5.146)$$

4. Compute a new step size h_1 from

$$h_1^{p+1} \max(d_1, d_2) = 0.01 \quad (5.147)$$

5. Use the following initial step size:

$$h = \min(100h_0, h_1) \quad (5.148)$$

5.7 Implicit Runge-Kutta Methods

Implicit Runge-Kutta (IRK) methods are more complicated - the diagonal and supra-diagonal elements of the Butcher array may be nonzero. IRK's fall into several classes, including:

- **Gauss Methods** which are based on Gaussian quadrature², and have an order of accuracy of $2s$, where s is the number of stages. Gauss methods have the highest attainable order for a given number of stages. Gauss methods are A-Stable but do not have stiff decay.
- **Radau Methods** where one end of the interval is included in the iteration formula; the order of accuracy is $2s - 1$. Radau methods have stiff decay.
- **Lobatto methods**, in which the function is sampled at both ends of the interval; the order of accuracy is $2s - 2$. Lobatto methods are A-Stable but do not have stiff decay.

²see [13], for example, for a discussion of Gaussian quadrature.

Consider for example the following 2-stage method:

$$\begin{array}{c|cc} 0 & 1/4 & -1/4 \\ 2/3 & 1/4 & 5/12 \\ \hline & 1/4 & 3/4 \end{array} \quad (5.149)$$

This gives the following iteration formulas:

$$K_1 = f\left(t_{n-1}, y_{n-1} + \frac{h}{4}(K_1 - K_2)\right) \quad (5.150)$$

$$K_2 = f\left(t_{n-1} + \frac{2}{3}h, y_{n-1} + \frac{h}{12}(3K_1 + 5K_2)\right) \quad (5.151)$$

$$y_n = y_{n-1} + \frac{h}{4}(K_1 + 3K_2) \quad (5.152)$$

It is sufficient to assume that the differential equation is autonomous, that is, that $f(t, y) = f(y)$ only and does not depend explicitly on t . This is because it is always possible to increase the dimension of a differential system by adding an extra variable, whose derivative is 1. This variable is equal to t and including it makes the system autonomous. Hence it is sufficient to study just autonomous systems. We will treat a scalar autonomous differential equation; the corresponding derivations for the vector system are analogous. The corresponding autonomous iteration formulas are

$$K_1 = f\left(y_{n-1} + \frac{h}{4}(K_1 - K_2)\right) \quad (5.153)$$

$$K_2 = f\left(y_{n-1} + \frac{h}{12}(3K_1 + 5K_2)\right) \quad (5.154)$$

$$y_n = y_{n-1} + \frac{h}{4}(K_1 + 3K_2) \quad (5.155)$$

Expanding in a Taylor Series about y_{n-1} ,

$$K_1 = f(y_{n-1}) + \frac{h}{4}(K_1 - K_2)f_y + \frac{h^2}{32}(K_1 - K_2)f_{yy} + O(h^3) \quad (5.156)$$

$$K_2 = f(y_{n-1}) + \frac{h}{12}(3K_1 + 5K_2)f_y + \frac{h^2}{288}(3K_1 + 5K_2)f_{yy} + O(h^3) \quad (5.157)$$

Hence $K_1, K_2 = f(y_{n-1}) + O(h)$; substituting this back into 5.156 gives

$$K_1 = f(y_{n-1}) + O(h^2) \quad (5.158)$$

$$K_2 = f(y_{n-1}) + \frac{2}{3}hf_y f + O(h^2) \quad (5.159)$$

Substituting equations 5.158 and 5.159 back into the right-hand sides of equations 5.156 and 5.157 gives

$$K_1 = f(y_{n-1}) - \frac{1}{6}h^2f_y^2f + O(h^3) \quad (5.160)$$

$$K_2 = f(y_{n-1}) + \frac{2}{3}hf_y f + h^2\left(\frac{5}{18}f_y^2f + \frac{2}{9}f_{yy}f^2\right) + O(h^3) \quad (5.161)$$

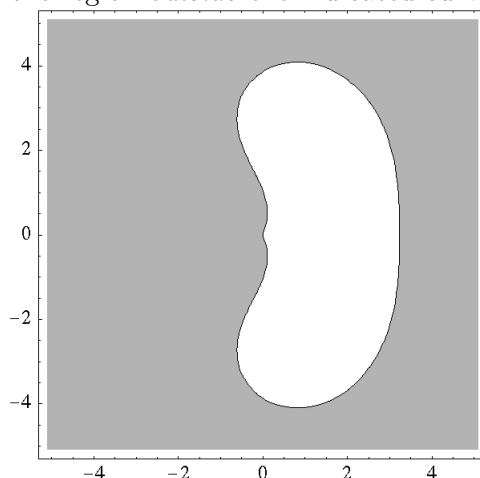
Hence the iteration formula (equation 5.155) gives

$$y_n = y_{n-1} + hf(y_{n-1}) + \frac{1}{2}h^2 f_y f + \frac{1}{6}h^3 (f_y^2 f + f_{yy} f^2) + O(h^4) \quad (5.162)$$

But since $y' = f$, $y'' = f_y f$, and $y''' = f_y^2 f^2 + f_{yy} f^2$, the exact expansion is identical. Thus the method is at least $O(h^3)$. (In fact the error is precisely $O(h^3)$ but this is left as an exercise).

Thus in contrast to explicit methods, implicit methods may have order higher than the number of stages; we shall see that they may, in fact have an order as high as twice the number of stages. The price we pay is that the method is implicit – some additional work, such as a Newton iteration – is required to implement the method.

Figure 5.6: Region of absolute stability for the method of equation 5.153. The region of absolute stability is the region *outside* the indicated curve.



Implicit methods can be absolutely stable in a large fraction of the negative real plane, making them potentially good stiff solvers. For example, the method 5.153 when applied to the test equation gives

$$K_1 = y_{n-1}\lambda + \frac{h}{4}(K_1 - K_2)\lambda \quad (5.163)$$

$$K_2 = y_{n-1}\lambda + \frac{h}{12}(3K_1 + 5K_2)\lambda \quad (5.164)$$

$$(5.165)$$

Solving the pair of equations for K_1 and K_2 gives

$$K_1 = \frac{6\lambda - 4h\lambda^2}{h^2\lambda^2 - 4h\lambda + 6}y_{n-1} \quad (5.166)$$

$$K_2 = \frac{6\lambda}{h^2\lambda^2 - 4h\lambda + 6}y_{n-1} \quad (5.167)$$

Hence absolute stability requires

$$\left| \frac{y_n}{y_{n-1}} \right| = \left| 1 + \frac{h}{4} \left(\frac{6\lambda - 4h\lambda^2}{h^2\lambda^2 - 4h\lambda + 6} + \frac{18\lambda}{h^2\lambda^2 - 4h\lambda + 6} \right) \right| \quad (5.168)$$

$$= \left| 1 + \frac{6z - z^2}{z^2 - 4z + 6} \right| \quad (5.169)$$

$$= \left| \frac{6 + 2z}{z^2 - 4z + 6} \right| \leq 1 \quad (5.170)$$

where $z = h\lambda$. The region of absolute stability is shown in figure 5.6

5.8 IRK Methods based on Collocation

In the method of **collocation**, given an initial value problem that we have solved numerically on the interval $[t_0, t_n]$ we seek a ν th order polynomial $P(t)$ such that

$$P(t_n) = y_n \quad (5.171)$$

$$P'(t_n)(t_n + c_j h) = f(t_n + c_j h, P(t_n + c_j h)), j = 1, 2, \dots, \nu \quad (5.172)$$

where the numbers c_1, \dots, c_n are called **collocation parameters**. Thus P satisfies the initial value problem defined by the numerical solution at t_n . A **collocation method** finds such a polynomial P and sets $y_{n+1} = P(t_{n+1})$.

Theorem 5.1. *The Implicit Runge-Kutta Method*

$$\frac{\mathbf{c} \mid \mathbf{A}}{\mathbf{b}^T} \quad (5.173)$$

and the collocation method of equation 5.171 are identical if

$$a_{ji} = \frac{1}{q_i(c_i)} \int_0^{c_j} q_i(s) ds \quad (5.174)$$

$$b_j = \frac{1}{q_j(c_j)} \int_0^1 q_j(s) ds \quad (5.175)$$

where

$$q_j(t) = \prod_{i=1, i \neq j}^{\nu} (t - c_i) \quad (5.176)$$

Proof. Define the ν th order polynomial

$$r(t) = \sum_{k=1}^{\nu} \frac{q_k((t - t_n)/h)}{q_k(c_k)} w_k \quad (5.177)$$

then

$$r(t_n + c_j h) = \sum_{k=1}^{\nu} \frac{q_k(c_j)}{q_k(c_k)} w_k = \sum_{k=1}^{\nu} \delta_{jk} w_k = w_j \quad (5.178)$$

If we choose

$$w_j = P'(t_n + c_j h), j = 1, 2, \dots, \nu \quad (5.179)$$

Then

$$r(t_n + c_j h) = w_j = P'(t_n + c_j h) \quad (5.180) \quad \Leftarrow$$

Since the two polynomials $r(t)$ and $P'(t)$ are both the same order ($\nu - 1$) and they coincide at ν points (order +1) then they must be identical polynomials. Hence we conclude that

$$r(t) = P'(t) \quad (5.181)$$

Then by equation 5.171

$$P'(t) = \sum_{k=1}^{\nu} \frac{q_k((t - t_n)/h)}{q_k(c_k)} P'(t_n + c_k h) \quad (5.182)$$

$$= \sum_{k=1}^{\nu} \frac{q_k((t - t_n)/h)}{q_k(c_k)} f(t_n + c_k h, P(t_n + c_k h)) \quad (5.183)$$

Integrating,

$$\int_{t_n}^t P'(s) ds = \sum_{k=1}^{\nu} f(t_n + c_k h, P(t_n + c_k h)) \int_{t_n}^t \frac{q_k((s - t_n)/h)}{q_k(c_k)} ds \quad (5.184)$$

$$P(t) - y_n = \sum_{k=1}^{\nu} f(t_n + c_k h, P(t_n + c_k h)) \int_0^{(t-t_n)/h} \frac{q_k(s)}{q_k(c_k)} h ds \quad \Leftarrow \quad (5.185)$$

If we let $t = t_n + c_j h$ in 5.185 then

$$P(t_n + c_j h) = y_n + h \sum_{k=1}^{\nu} f(t_n + c_k h, P(t_n + c_k h)) \int_0^{c_j} \frac{q_k(s)}{q_k(c_k)} ds \quad \Leftarrow \quad (5.186)$$

hence

$$P(t_n + c_j h) = y_n + \sum_{k=1}^{\nu} a_{j,k} f(t_n + c_k h, P(t_n + c_k h)) \quad \Leftarrow \quad (5.187)$$

Let

$$K_j = P(t_n + c_j h) = y_n + \sum_{k=1}^{\nu} a_{j,k} f(t_n + c_k h, K_k) \quad \Leftarrow \quad (5.188)$$

Then if we let $t = t_{n+1}$ in 5.185 then,

$$P(t_{n+1}) = y_n + h \sum_{k=1}^{\nu} f(t_n + c_k h, P(t_n + c_k h)) \int_0^1 \frac{q_k(s)}{q_k(c_k)} ds \quad \Leftarrow \quad (5.189)$$

$$= y_n + h \sum_{k=1}^{\nu} b_k f(t_n + c_k h, K_k) \quad \Leftarrow \quad (5.190)$$

which is a ν -stage implicit Runge-Kutta method. \square

The maximum order of any collocation method is equal to the number of stages. **Gauss-Legendre** numerical methods are ν -stage, order 2ν methods that are based on collocation, and hence they give the highest possible order for any ν -stage method.³

Example 5.3. Calculate the Butcher array for the $\nu = 1$ using $P_1(t) = t - 1/2$

Solution. The only root is at $c_1 = 1/2$, hence

$$q(t) = t - 1/2 \quad (5.191)$$

$$q_1(t) = 1 \quad (5.192)$$

$$a_{11} = \frac{1}{1} \int_0^{1/2} ds = \frac{1}{2} \quad (5.193)$$

$$b_1 = \frac{1}{1} \int_0^1 ds = 1 \quad (5.194)$$

Hence the Butcher Array is

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array} \quad \square \quad (5.195)$$

Example 5.4. Calculate the Butcher Array for the two-stage Gauss-Legendre method with $\nu = 2$ where $P_2(t) = t^2 - t + 1/6$

Solution. The roots of $6t^2 - 6t + 1 = 0$ are at

$$c_1, c_2 = \frac{6 \pm \sqrt{36 - 24}}{12} = \frac{1}{2} \pm \frac{\sqrt{3}}{6} \quad (5.196)$$

Hence

$$q(t) = \left(t - \frac{1}{2} - \frac{\sqrt{3}}{6} \right) \left(t - \frac{1}{2} + \frac{\sqrt{3}}{6} \right) \quad (5.197)$$

$$= t^2 - t + \frac{1}{2} \quad (5.198)$$

$$q_1(t) = t - \frac{1}{2} + \frac{\sqrt{3}}{6} \quad (5.199)$$

$$q_1(c_1) = \frac{1}{2} + \frac{\sqrt{3}}{6} - \frac{1}{2} + \frac{\sqrt{3}}{6} = \frac{\sqrt{3}}{3} \quad (5.200)$$

$$\frac{q_1(t)}{q_1(c_1)} = \sqrt{3}t - \frac{\sqrt{3}}{2} + \frac{1}{2} \quad (5.201)$$

$$a_{11} = \int_0^{1/2+\sqrt{3}/6} \left(\sqrt{3}s - \frac{\sqrt{3}}{2} + \frac{1}{2} \right) ds \quad (5.202)$$

$$= \left(\frac{\sqrt{3}}{2}s^2 + \frac{1-\sqrt{3}}{2}s \right) \Big|_0^{1/2+\sqrt{3}/6} \quad (5.203)$$

$$= \frac{\sqrt{3}}{2} \left(\frac{1}{2} + \frac{\sqrt{3}}{6} \right)^2 + \left(\frac{1-\sqrt{3}}{2} \right) \left(\frac{1}{2} + \frac{\sqrt{3}}{6} \right) = \frac{1}{4} \quad (5.204)$$

³For a proof of this statement see [13] section 3.4.

The remaining coefficients are left as an exercise. The result is

$$\frac{\begin{array}{c|cc} 1/2 - \sqrt{3}/6 & 1/4 & 1/4 - \sqrt{3}/6 \\ 1/2 + \sqrt{3}/6 & 1/4 + \sqrt{3}/6 & 1/4 \end{array}}{\begin{array}{c|cc} & 1/2 & 1/2 \end{array}} \quad \square \quad (5.205)$$

The method just computed is A-stable, because the region of stability is precisely the left-hand plane. It is also a fourth-order method, because it has two stages and was derived using the Gauss-Legendre method. The absolute stability condition gives

$$\left| \frac{y_{n+1}}{y_n} \right| = \left| \frac{1 + z/2 + (1 + \sqrt{3})z^2/12}{1 - z/2 + z^2/12} \right| < 1 \quad (5.206)$$

Because of the extra computation time needed for implicit methods they are more expensive than comparable order/stage explicit methods. On the other hand, they are much easier to derive and many of them are suitable for stiff problems. According to [13] the following three-stage sixth order method is “probably the largest that is consistent with reasonable implementation costs:”

$$\frac{\begin{array}{c|ccc} 1/2 - \sqrt{15}/10 & 5/36 & 2/9 - \sqrt{15}/15 & 5/36 - \sqrt{15}/30 \\ 1/2 & 5/36 + \sqrt{15}/24 & 2/9 & 5/36 - \sqrt{15}/24 \\ 1/2 + \sqrt{15}/10 & 5/36 + \sqrt{15}/30 & 2/9 + \sqrt{15}/15 & 5/36 \end{array}}{\begin{array}{c|ccc} & 5/18 & 4/9 & 5/18 \end{array}} \quad (5.207)$$

5.9 Páde Approximants and A-Stability

Definition 5.1. *Let*

$$\frac{\mathbf{c}}{\mathbf{b}^T} \Big| \frac{\mathbf{A}}{\mathbf{b}^T} \quad (5.208)$$

be a Runge Kutta method. Then

$$R(z) = \frac{y_n}{y_{n-1}} \quad (5.209)$$

*is called the **Stability Function** for the method. In terms of this definition, the requirement for absolute stability is that $|R(z)| \leq 1$.*

Theorem 5.2. *The stability function for any Runge-Kutta method $\frac{\mathbf{c}}{\mathbf{b}^T} \Big| \frac{\mathbf{A}}{\mathbf{b}^T}$ is*

$$R(z) = 1 + z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1} \quad (5.210)$$

$z = h\lambda$, and

$$\mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \quad (5.211)$$

Proof. When applied to the test equation $y' = \lambda y$ the general Runge-Kutta method \implies has

$$K_i = \lambda y_{n-1} + z \sum_{i=1}^{\nu} a_{ji} K_i, \quad = 1, 2, \dots, \nu \quad (5.212)$$

$$y_n = y_{n-1} + h \sum b_i K_i \quad (5.213)$$

where $z = h\lambda$. Let us define the following vector in \mathbb{R}^ν :

$$\xi = \begin{pmatrix} K_1 \\ \vdots \\ K_\nu \end{pmatrix} \quad (5.214)$$

Then we can rewrite equation 5.212 as

$$\xi = \lambda \mathbf{1} y_{n-1} + z \mathbf{A} \xi \quad \Leftarrow \quad (5.215)$$

Rearranging and solving for ξ , \Leftarrow

$$\xi - z \mathbf{A} \xi = \lambda \mathbf{1} y_{n-1} \quad (5.216)$$

$$(\mathbf{I} - z \mathbf{A}) \xi = \lambda \mathbf{1} y_{n-1} \quad (5.217)$$

$$\xi = \lambda (\mathbf{I} - z \mathbf{A})^{-1} \mathbf{1} y_{n-1} \quad (5.218)$$

hence \Leftarrow

$$y_n = y_{n-1} + h \sum_{j=1}^{\nu} b_j K_j \quad (5.219)$$

$$= y_{n-1} + h \mathbf{b}^T \xi \quad (5.220)$$

$$= (1 + z \mathbf{b}^T (\mathbf{I} - z \mathbf{A})^{-1} \mathbf{1}) y_{n-1} \quad (5.221)$$

which is equivalent to the desired result. \square

Theorem 5.3. *The stability function satisfies*

$$R(z) = \frac{\det(\mathbf{I} - z \mathbf{A} + z \mathbf{1} \mathbf{b}^T)}{\det(\mathbf{I} - z \mathbf{A})} \quad (5.222)$$

Proof. For the test problem $y' = \lambda y$, \Leftarrow

$$(\mathbf{I} - z \mathbf{A}) \xi = \lambda \mathbf{1} y_{n-1} \quad (5.223)$$

$$-h \mathbf{b}^T \xi + y_n = y_{n-1} \quad (5.224)$$

hence

$$\begin{pmatrix} \mathbf{I} - z \mathbf{A} & 0 \\ -h \mathbf{b}^T & 1 \end{pmatrix} \begin{pmatrix} \xi \\ y_n \end{pmatrix} = y_{n-1} \begin{pmatrix} \lambda \mathbf{1} \\ 1 \end{pmatrix} \quad (5.225)$$

The result follows from Cramer's Rule (solve for y_n). \square

Example 5.5. Find the stability function for the classical fourth-order Runge-Kutta method

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \quad (5.226)$$

Solution.

$$\mathbf{I} - z\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -z/2 & 1 & 0 & 0 \\ 0 & -z/2 & 1 & 0 \\ 0 & 0 & -z & 1 \end{pmatrix} \quad (5.227)$$

Therefore $\det(\mathbf{I} - z\mathbf{A}) = 1$ and

$$\mathbf{I} - z\mathbf{A} + z\mathbf{1b}^T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -z/2 & 1 & 0 & 0 \\ 0 & -z/2 & 1 & 0 \\ 0 & 0 & -z & 1 \end{pmatrix} + \frac{z}{6} \begin{pmatrix} 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \end{pmatrix} \quad (5.228)$$

$$= \begin{pmatrix} 1 + z/6 & z/3 & z/3 & z/6 \\ -z/3 & 1 + z/3 & z/3 & z/6 \\ z/6 & -z/6 & 1 + z/3 & z/6 \\ z/6 & z/3 & -2z/3 & 1 + z/6 \end{pmatrix} \quad (5.229)$$

$$\det(\mathbf{I} - z\mathbf{A} + z\mathbf{1b}^T) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} \quad (5.230)$$

Therefore

$$R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} \quad (5.231)$$

□

The formulas given by these theorems are examples of **Padé Approximants**. More specifically, a Padé Approximant is $P_{m/n}$ is a rational function

$$P_{m/n}(t) = \frac{P_m(t)}{P_n(t)} \quad (5.232)$$

where P_m and P_n are polynomials of degrees m and n , respectively. The Padé Approximant is the best approximation of function by a rational function of a given order; it is essentially the Rational-function generalization of a Taylor approximation.

We observe that not only is the result a rational function, it is a polynomial. This result is true in general for all explicit Runge-Kutta methods.

Theorem 5.4. *For every Runge-Kutta method there exists a rational function $P_{\nu/\nu}$ such that*

$$y_n = [P_{\nu/\nu}(z)]^n \quad (5.233)$$

Proof. Start with

$$R(z) = 1 + z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1} \quad (5.234)$$

which is a result we have already demonstrated. Then

$$(\mathbf{I} - z\mathbf{A})^{-1} = \frac{\text{adj}(\mathbf{I} - z\mathbf{A})}{\det(\mathbf{I} - z\mathbf{A})} \quad (5.235)$$

where $\text{adj}(\mathbf{M})$ is the adjunct matrix of \mathbf{M} ,

$$[\text{adj}(\mathbf{M})]_{ij} = (-1)^{i+j} \text{minor}(M_{ji}) \quad (5.236)$$

We observe that each entry in the adjunct matrix is linear in z , hence each of the principal minor determinants is a polynomial in z of order $\nu - 1$. Therefore

$$z\mathbf{b}^T \text{adj}(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1} \quad (5.237)$$

is a polynomial of degree ν . Hence $z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1}$ is the quotient of two polynomials, each of order ν , which immediately yields the desired result. \square

Corollary 5.1. *For an explicit Runge-Kutta method, $y_n = [P_\nu]^n$, a polynomial of order ν .*

Proof. If the method is explicit then \mathbf{A} is strictly lower triangular; hence $\det(\mathbf{I} - z\mathbf{A}) = 1$. \square

Corollary 5.2. *Explicit Runge-Kutta methods may not be A-stable.*

Proof. A non-constant polynomial cannot be uniformly bounded. \square

Theorem 5.5. *Let $R(z)$ be any rational functions. Then $|R(z)| < 1$ for all $z \in \mathbb{C}^-$ if and only if both of the following conditions hold:*

1. *All the poles of $R(z)$ have positive real parts; and*
2. *$|R(it)| \leq 1$ for all $t \in \mathbb{R}$.*

Theorem 5.6. *$R(z) = e^z + O(z^{\nu_1})$ for any Runge-Kutta method for order ν .*

Proof. By definition, $y_n = R(z)y_{n-1}$; for the test equation, $y(t_n) = y_n = e^z y_{n-1}$. \square

Theorem 5.7. *For any two integers m, n there is a Padé Approximant to the exponential for the stability function such that*

$$P_{m/n}(t) = \frac{P_{m,n}(t)}{Q_{m,n}(t)} \quad (5.238)$$

of order $m + n$, where

$$P_{m,n} = \sum_{k=0}^m \binom{m}{k} \frac{(m+n-k)!}{(m+n)!} z^k \quad (5.239)$$

$$Q_{m,n} = \sum_{k=0}^n \binom{n}{k} \frac{(m+n-k)!}{(m+n)!} (-z)^k = P_{n,m}(-z) \quad (5.240)$$

Chapter 6

Linear Multistep Methods for Initial Value Problems

6.1 Multistep Methods

Definition 6.1. A **Linear Multistep Method** for the initial value problem $y' = f(t, y), y(t_0) = y_0$ is any method of the form

$$\sum_{j=0}^k a_j y_{n-j} = h \sum_{j=0}^k b_j f_{n-j} \quad (6.1)$$

where $f_k = f(t_k, y_k)$, and $a_0, a_1, \dots, a_k, b_0, b_1, \dots, b_k$ are constants. If $b_0 = 0$ the method is explicit, and if $b_0 \neq 0$ the method is implicit.

These methods are linear in the function $f(t, y)$, as compared with Runge-Kutta methods, which were non-linear in f . The linearity of the method does not imply the linearity of y – in fact, there is no such restriction on y – but only the linearity of f in the method. They are called multistep methods because they depend on function (and possibly solution) values at up to k prior mesh points. We will also find the following notation helpful:

Definition 6.2. The **Characteristic Polynomials** of a linear multistep method are

$$\rho(x) = \sum_{j=0}^k a_j x^{k-j} \quad (6.2)$$

$$\sigma(x) = \sum_{j=0}^k b_j x^{k-j} \quad (6.3)$$

The **order** of a linear multistep method is particularly easy to calculate, as we will show in the following derivation. Define the **linear multistep operator**

corresponding to a method with coefficients $\{a_0, a_1, \dots, b_0, \dots\}$ as

$$\mathcal{L}u(t) = \sum_{j=0}^k [a_j u_{n-j} - hb_j u'_{n-j}] \quad (6.4)$$

for any arbitrary continuously differentiable function u with discretization u_n . If $y(t)$ is the exact solution of the differential equation $y' = f$ then

$$\mathcal{L}y(t) = \sum_{j=0}^k [a_j y_{n-j} - hb_j f_{n-j}] \quad (6.5)$$

Expanding the terms y_{n-j} and f_{n-j} right hand side in a Taylor series,

$$y_{n-j} = y_n - jhy'_n + \frac{(jh)^2}{2} y''_n + \dots + \frac{(-jh)^k}{k!} y_n^{(k)} + \dots \quad (6.6)$$

$$f_{n-j} = y'_n - jhy''_n + \frac{(jh)^2}{2} y'''_n + \dots + \frac{(-jh)^k}{k!} y_n^{(k+1)} + \dots \quad (6.7)$$

$$a_j y_{n-j} - hb_j f_{n-j} = a_0 y_n + (ja_1 - b_0) hy'_n + \left(\frac{a_2}{2} j^2 - jb_1\right) h^2 y''_n + \dots \quad (6.8)$$

$$+ (-1)^k \left(\frac{a_j j^k}{k!} - \frac{b_j j^{k-1}}{(k-1)!}\right) h^k y_n^{(k)} + \dots \quad (6.9)$$

$$(6.10)$$

Hence we have

$$\mathcal{L}y(t) = \sum_{j=0}^k \sum_{i=0}^{\infty} (-1)^i \left(\frac{a_j j^i}{i!} - \frac{b_j j^{i-1}}{(i-1)!}\right) h^i y_n^{(i)} \quad (6.11)$$

$$= C_0 y_n + C_1 h y'_n + C_2 h^2 y''_n + \dots \quad (6.12)$$

where

$$C_0 = \sum_{j=0}^k a_j \quad (6.13)$$

$$C_i = (-1)^i \left[\frac{1}{i!} \sum_{j=1}^k j^i a_j + \frac{1}{(i-1)!} \sum_{j=0}^k j^{(i-1)} b_j \right] \quad (6.14)$$

Hence the order of the method is p if

$$C_0 = C_1 = \dots = c_p = 0, \quad c_{p+1} \neq 0 \quad (6.15)$$

and therefore to obtain a method of order p , we need to set successive values of C to zero:

$$0 = a_0 + a_1 + \dots + a_k \quad (6.16)$$

$$0 = (a_1 + 2a_2 + \dots + ka_k) + (b_0 + b_1 + \dots + b_k) \quad (6.17)$$

$$0 = \frac{1}{2}(a_1 + 4a_2 + \dots + k^2 a_k) + (b_1 + 2b_2 + \dots + kb_k) \quad (6.18)$$

$$\vdots \quad (6.19)$$

Theorem 6.1. *A linear multistep method is **consistent** if it has order greater than or equal to 1. Thus*

$$0 = \sum_{j=0}^k a_j \quad (6.20)$$

$$0 = \sum_{j=1}^k j a_j + \sum_{j=0}^k b_j \quad (6.21)$$

In terms of the characteristic polynomial, the method is consistent if and only if

$$\rho(1) = 0 \quad (6.22)$$

$$\rho'(1) = \sigma(1) \quad (6.23)$$

The proof follows immediately from the definitions of the C_i and the characteristic polynomials.

6.2 The Root Condition for Linear Multistep Methods

If we apply the general k -step linear multistep method

$$\sum_{j=0}^k a_j y_{n-j} = h \sum_{j=0}^k b_j f_{n-j} \quad (6.24)$$

to the test equation $y' = \lambda y$, we obtain the difference equation

$$\sum_{j=0}^k a_j y_{n-j} = z \sum_{j=0}^k b_j y_{n-j} \quad (6.25)$$

where $z = h\lambda$. Solutions to this difference equation include r^k where r is any root of the characteristic polynomial

$$\phi(r) = \rho(r) - z\sigma(r) \quad (6.26)$$

where ρ and σ are as defined in equations 6.2 and 6.3. Because stability relates to what happens in the limit $h \rightarrow 0$ ($z = 0$ in the above equation) it turns out that only the roots of ρ matter.

Difference Equations

Any linear equation relating variables y_0, y_1, \dots of the form

$$\sum_{j=0}^k a_j y_j = g \quad (6.27)$$

is called an **(inhomogeneous) difference equation**. The corresponding **homogeneous difference equation** is

$$\sum_{j=0}^k a_j y_j = 0 \quad (6.28)$$

There are k linearly independent solutions to 6.28; the general solution of the homogeneous equation is the sum of these linearly independent terms. To find the terms of the homogeneous equation we substitute $y_j = r^j$ to obtain the **characteristic equation of the difference equation**

$$\sum_{j=0}^k a_j r^j = 0 \quad (6.29)$$

Corresponding to each root r of multiplicity s is a term

$$a_k = r^k (c_1 + c_2 k + c_3 k^2 + \dots + c_s k^{s-1}) \quad (6.30)$$

in the homogeneous solution. The solution to the inhomogeneous equation is the sum of the homogeneous solution and a particular solution to the full equation.

Example 6.1. Solve $a_{k+1} - 2a_k = 3^k + 5k$.

Solution. The homogeneous (linear) part of the equation is $a_{k+1} - 2a_k = 0$; the characteristic polynomial is $r^{k+1} - 2r^k = 0$, which has a single nonzero root of $r = 2$. (We ignore the zero roots because we always renormalize to $k = 0$ in the characteristic equation; if zero-valued roots still remained after this division by r^k we would have to include them in the solution). For a particular solution we guess $a_k = A(3)^k + 5Bk + C$ where A, B, C are unknown constants. We guess a polynomial to satisfy the $5k$ term and the exponential 3^k to satisfy the corresponding term in the difference equation. To determine the coefficients we substitute

$$3^k + 5k = a_{k+1} - a_k = (3^{k+1}A + 5B(k+1) + C) - 2(3^kA + 5Bk + C) \quad (6.31)$$

$$= 3^kA - 5Bk + 5B - C \quad (6.32)$$

Equating coefficients of linearly independent terms gives $A = 1$, $B = -1$, and $C = -5$, so the solution is $a_k = 2^k + 3^k - 5k - 5$. \square

Theorem 6.2 (First Root Condition). *The linear multistep method 6.1 is stable if all the roots r_i of the characteristic polynomial $\rho(r)$ satisfy*

$$|r_i| \leq 1 \quad (6.33)$$

and if $|r_i| = 1$ then r_i must be a simple root.

Any roots that violate the root condition are called **extraneous roots**. One of the objectives of designing a good method is to eliminate the extraneous roots.

A method is said to be **strongly stable** if all of the roots are within the unit circle, except possibly for a root at $r = 1$.

A method is said to be **weakly stable** if it is stable but not strongly stable.

The region of absolute stability is given by that part of the Complex plane bounded by

$$z = \frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})} \quad (6.34)$$

because $|e^{i\theta}| = 1$.

Theorem 6.3 (Second Root Condition). *The linear multistep method 6.1 is absolutely stable if all the roots r_i of the characteristic polynomial $\phi(r) = \rho(r) - z\sigma(r)$ satisfy*

$$|r_i| \leq 1 \quad (6.35)$$

Theorem 6.4. *An explicit linear multistep method can not be A-stable.*

Theorem 6.5 (First Dahlquist Barrier). *The order p of a stable linear k -step multi-step method satisfies*

$$p \leq k + 2, \quad \text{if } k \text{ is even;} \quad (6.36)$$

$$p \leq k + 1, \quad \text{if } k \text{ is odd;} \quad (6.37)$$

$$p \leq k, \quad \text{if } b_k/a_k \leq 0 \text{ or the method is explicit.} \quad (6.38)$$

Theorem 6.6 (Second Dahlquist Barrier). *An A-stable linear-multistep method must be of order less than or equal to 2.*

6.3 Backward Difference Formula

The BDF formula is obtained by seeking a polynomial of the form

$$\begin{aligned} P(t) = & a_0 + a_1(t - t_n) \\ & + a_2(t - t_n)(t - t_{n-1}) \\ & + a_3(t - t_n)(t - t_{n-1})(t - t_{n-2}) \\ & \vdots \\ & + a_n(t - t_n)(t - t_{n-1}) \cdots (t - t_1) \end{aligned} \quad (6.39)$$

that interpolates the points

$$P(t_0) = f(t_0) \quad (6.40)$$

$$P(t_1) = f(t_1) \quad (6.41)$$

$$\vdots$$

$$P(t_{n-1}) = f(t_{n-1}) \quad (6.42)$$

$$P(t_n) = f(t_n) \quad (6.43)$$

We define the **backward difference operator** ∇ for an element f_n of a sequence as

$$\nabla f_n = f_n - f_{n-1} \quad (6.44)$$

$$\nabla^2 f_n = \nabla f_n - \nabla f_{n-1} = f_n - 2f_{n-1} + f_{n-2} \quad (6.45)$$

$$\nabla^3 f_n = \nabla^2 f_n - \nabla^2 f_{n-1} = f_n - 3f_{n-1} + 3f_{n-2} - f_{n-3} \quad (6.46)$$

$$\vdots$$

$$\nabla^k f_n = \nabla^{k-1} f_n - \nabla^{k-1} f_{n-1} \quad (6.47)$$

Letting $f_n = f(t_n)$ we have by substituting 6.43 into 6.39 that

$$f_n = a_0 \quad (6.48)$$

From 6.42 we get

$$f_{n-1} = f_n + a_1(t_{n-1} - t_n) \quad (6.49)$$

$$= f_n - a_1 h \quad (6.50)$$

$$a_1 = \frac{1}{h}(f_n - f_{n-1}) = \frac{1}{h}\nabla f_n \quad (6.51)$$

Substituting at $t = t_{n-2} = t_n - 2h$ gives

$$f_{n-2} = a_0 + a_1(t_{n-2} - t_n) + a_2(t_{n-2} - t_n)(t_{n-2} - t_{n-2}) \quad (6.52)$$

$$= f_n + \frac{1}{h}(f_n - f_{n-1})(-2h) + a_2(-2h)(-h) \quad (6.53)$$

$$= 2f_{n-1} - f_n + 2h^2 a_2 \quad (6.54)$$

$$a_2 = \frac{1}{2h^2}(f_n - 2f_{n-1} + f_{n-2}) = \frac{1}{2h^2}\nabla^2 f_n \quad (6.55)$$

Continuing the process we find in general that

$$a_k = \frac{1}{k!h^k}\nabla^k f_n \quad (6.56)$$

Next we define the polynomials Q_k by

$$Q_k(t) = \prod_{j=0}^{k-1} (t - t_{n-j}) \quad (6.57)$$

Using 6.56 and 6.57 in 6.39

$$P(t) = a_0 + a_1 Q_1(t) + a_2 Q_2(t) + \cdots + a_n Q_n(t) \quad (6.58)$$

$$= a_0 + \sum_{k=1}^n a_k Q_k(t) \quad (6.59)$$

$$= f_n + \sum_{k=1}^n \frac{\nabla^k f_n}{k! h^k} Q_k(t) \quad (6.60)$$

Define the parameter s , $-1 \leq s \leq 0$ in the interval $[t_{n-1}, t_n]$ by

$$t = t_n + sh \quad (6.61)$$

From equation 6.57,

$$Q_k(t) = \prod_{j=0}^{k-1} (t_n + sh - (t_n - jh)) \quad (6.62)$$

$$= \prod_{j=0}^{k-1} (j + s)h \quad (6.63)$$

$$= h^k \prod_{j=0}^{k-1} (s + j) \quad (6.64)$$

$$= h^k s(s+1)(s+2) \cdots (s+k-1) \quad (6.65)$$

Recall the definition of the binomial coefficient for n, m integers,

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = \frac{n(n-1)(n-2) \cdots (n-m+1)}{m!} \quad (6.66)$$

we can define, for any real number t , not necessarily integer,

$$\binom{t}{k} = \frac{t(t-1)(t-2) \cdots (t-m+1)}{k!} \quad (6.67)$$

Using this we calculate

$$\binom{-s}{k} = \frac{-s(-s-1)(-s-2) \cdots (-s-k+1)}{k!} \quad (6.68)$$

$$= \frac{(-1)^k}{k!} s(s+1)(s+2) \cdots (s+k-1) \quad (6.69)$$

$$= \frac{(-1)^k}{k! h^k} Q_k(t) \quad (6.70)$$

Using 6.70 in 6.60 we get

$$P(t) = f_n + \sum_{k=1}^n (-1)^k \binom{-s}{k} \nabla^k f_n \quad (6.71)$$

which is known as **Newton's Backward Difference Formula**.

6.4 Adams Methods

Definition 6.3. An **Adams Method** is a linear multistep method with $a_0 = 1, a_1 = -1$, and $a_k = 0$ for all $k > 1$. The explicit **Adams-Bashforth Methods** are given by

$$y_n = y_{n-1} + h(b_1 f_{n-1} + b_2 f_{n-2} + \cdots + b_k f_{n-k}) \quad (6.72)$$

while the implicit **Adams-Moulton Methods**

$$y_n = y_{n-1} + h(b_0 f_n + b_2 f_{n-1} + \cdots + b_k f_{n-k}) \quad (6.73)$$

Adams methods are derived by integrating the differential equation over two grid points

$$y_n = y_{n-1} + \int_{t_{n-1}}^{t_n} f(s, y(s)) ds \quad (6.74)$$

and approximating the integrand with an interpolating polynomial over the past several mesh points. The interpolating polynomial of choice is the **Newton backward difference formula** (BDF),

$$P_k(t_n) = f(t_n) + \sum_{j=1}^k (-1)^j \binom{-s}{j} \nabla^j f(t_n) \quad (6.75)$$

where

$$\binom{s}{k} = \frac{s(s-1)\cdots(s-k+1)}{k!} \quad (6.76)$$

$$t = t_n + sh \quad (6.77)$$

$$\nabla p_n = p_n - p_{n-1} \quad (6.78)$$

$$\nabla^k p_n = \nabla \left(\nabla^{k-1} p_n \right), \quad k \geq 2 \quad (6.79)$$

The resulting Adams method coefficients are then given by

$$b_j = (-1)^{j-1} \sum_{i=j-1}^{k-1} \binom{i}{j-1} (-1)^i \int_0^1 \binom{-s}{i} ds \quad (6.80)$$

Example 6.2. Derive the 3-stage Adams-Bashforth method.

Solution. The method will be given by

$$y_n = y_{n-1} + \int_{t_{n-1}}^{t_n} f(u, y(u)) du \quad (6.81)$$

where f is estimated using

$$\begin{aligned} f &\approx f_{n-1} + (-1) \binom{-s}{1} \nabla f_{n-1} + (-1)^2 \binom{-s}{2} \nabla^2 f_{n-1} \\ &\quad + (-1)^3 \binom{-s}{3} \nabla^3 f_{n-1} \end{aligned} \quad (6.82)$$

Since

$$(-1)^1 \binom{-s}{1} = (-1)(-s) = s \quad (6.83)$$

$$(-1)^2 \binom{-s}{2} = (-1)^2 \frac{(-s)(-s-1)}{2} = \frac{1}{2}(s^2 + s) \quad (6.84)$$

$$(-1)^3 \binom{-s}{3} = (-1)^3 \frac{(-s)(-s-1)(-s-2)}{6} = \frac{1}{6}(s^3 + 3s^2 + 2s) \quad (6.85)$$

Newton's backward difference formula becomes

$$f \approx f_{n-1} + s\nabla f_{n-1} + \frac{1}{2}(s^2 + s)\nabla^2 f_{n-1} + \frac{1}{6}(s^3 + 3s^2 + 2s)\nabla^3 f_{n-1} \quad (6.86)$$

Changing the variable of integration from t to $s = (t - t_{n-1})/h$, equation 6.81 becomes

$$y_n = y_{n-1} + \int_0^1 f(s, y(s)) ds \quad (6.87)$$

Substituting the approximation formula,

$$y_n = y_{n-1} + h \int_0^1 \left[f_{n-1} + s\nabla f_{n-1} + \frac{s^2 + s}{2}\nabla^2 f_{n-1} + \right. \quad (6.88)$$

$$\left. \frac{s^3 + 3s^2 + 2s}{6}\nabla^3 f_{n-1} \right] ds \quad (6.89)$$

$$= y_{n-1} + h \left[f_{n-1} + \frac{1}{2}\nabla f_{n-1} + \frac{5}{12}\nabla^2 f_{n-1} + \frac{3}{8}\nabla^3 f_{n-1} \right] \quad (6.90)$$

$$= y_{n-1} + h \left[f_{n-1} + \frac{1}{2}(f_{n-1} - f_{n-2}) + \frac{5}{12}(f_{n-1} - 2f_{n-2} + f_{n-3}) + \right. \quad (6.91)$$

$$\left. \frac{3}{8}(f_{n-1} - 3f_{n-2} + 3f_{n-3} - f_{n-4}) \right] \quad (6.92)$$

$$= y_{n-1} + h \left[\frac{55}{24}f_{n-1} - \frac{59}{24}f_{n-2} + \frac{37}{24}f_{n-3} - \frac{3}{8}f_{n-4} \right] \quad \square \quad (6.93)$$

Example 6.3. Derive the 3-step Adams-Moulton implicit formula.

Solution. The method is similar to the Adams-Bashforth derivation except for the following two modifications: the interpolation formula is evaluated at f_n instead of f_{n-1} ; and because the integration interval *precedes* the point where the interpolation

formula is evaluated, the limits of integration are $[-1, 0]$ rather than $[0, 1]$. Hence

$$y_n = y_{n-1} + h \int_{-1}^0 \left(f_n + s \nabla f_n + \frac{1}{2}(s^2 + 2) \nabla^2 f \right) ds \quad (6.94)$$

$$= y_{n-1} + h \left[s f_n + \frac{1}{2} s^2 \nabla f_n + \frac{1}{2} \left(\frac{1}{3} s^3 + \frac{1}{2} s^2 \right) \nabla^2 f_n \right]_{-1}^0 \quad (6.95)$$

$$= y_{n-1} + h \left[f_n - \frac{1}{2} \nabla f_n - \frac{1}{12} \nabla^2 f_n \right] \quad (6.96)$$

$$= y_{n-1} + h \left[f_n - \frac{1}{2}(f_n - f_{n-1}) - \frac{1}{12}(f_n - 2f_{n-1} + f_{n-2}) \right] \quad (6.97)$$

$$= y_{n-1} + h \left(\frac{5}{12} f_n + \frac{2}{3} f_{n-1} - \frac{1}{12} f_{n-2} \right) \quad \square \quad (6.98)$$

None of the Adams's methods are A-stable except for AM1 (the Forward Euler Method) and AM2 (Trapezoidal Rule). The outer boundaries of the regions of stability are plotted for several of these methods in figure ??, which are also listed in tables 6.1 and 6.2.

Table 6.1: Explicit Adams-Bashforth Methods.

Method	Formula for $y_n - y_{n-1}$
AB1	$h f_{n-1}$ (Euler's Method)
AB2	$\frac{h}{2} (3f_{n-1} - f_{n-2})$
AB3	$\frac{h}{12} (23f_{n-1} - 16f_{n-2} + 5f_{n-3})$
AB4	$\frac{h}{24} (55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4})$
AB5	$\frac{h}{720} (1901f_{n-1} - 2774f_{n-2} + 2616f_{n-3} - 1274f_{n-4} + 251f_{n-5})$
AB6	$\frac{h}{1440} (4277f_{n-1} - 7923f_{n-2} + 9982f_{n-3} - 7298f_{n-4} + 2877f_{n-5} - 475f_{n-6})$

6.5 BDF Methods

An alternative multistep method use backward differentiation. In these methods, the polynomial approximation is applied to y' rather than f . If $sh + t_n = t$ then

$$y'(t_n) = \frac{d}{dt} \sum_{k=0}^{\nu} (-1)^k \binom{-s}{k} \nabla^k y_n \Big|_{t=t_n} \quad (6.99)$$

Table 6.2: Implicit Adams-Moulton Methods.

Method	Formula for $y_n - y_{n-1}$
AM1	hf_n (Backward Euler's Method)
AM2	$\frac{h}{2}(f_n + f_{n-1})$ (Trapezoidal Rule)
AM3	$\frac{h}{12}(5f_n + 8f_{n-1} - f_{n-2})$
AM4	$\frac{h}{24}(9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3})$
AM5	$\frac{h}{720}(251f_n + 646f_{n-1} - 264f_{n-2} + 106f_{n-3} - 19f_{n-4})$
AM6	$\frac{h}{1440}(475f_n + 1427f_{n-1} - 798f_{n-2} + 482f_{n-3} - 173f_{n-4} + 27f_{n-5})$

Since $s = 0$ corresponds to $t = t_n$,

$$\left. \frac{d}{dt} \binom{-s}{k} \right|_{t=t_n} = \frac{1}{h} \left. \frac{d}{ds} \binom{-s}{k} \right|_{s=0} \iff \quad (6.100)$$

$$= \frac{1}{hk!} \left. \frac{d}{ds} [(-s)(-s-1)(-s-2)\cdots(-s-k+1)] \right|_{s=0} \quad (6.101)$$

$$= (-1)^k \frac{(k-1)!}{hk!} = (-1)^k \frac{1}{hk} \quad (6.102)$$

Hence

$$y'(t_n) = \sum_{k=0}^{\nu} \frac{1}{hk} \nabla^k y_n \quad (6.103)$$

Substituting this in the equation $y' = f$ gives us the **BDF formula** or **Gear's Method** is given by

$$\sum_{k=0}^{\nu} \frac{1}{k} \nabla^k y_n = hf(t_n, y_n) \quad (6.104)$$

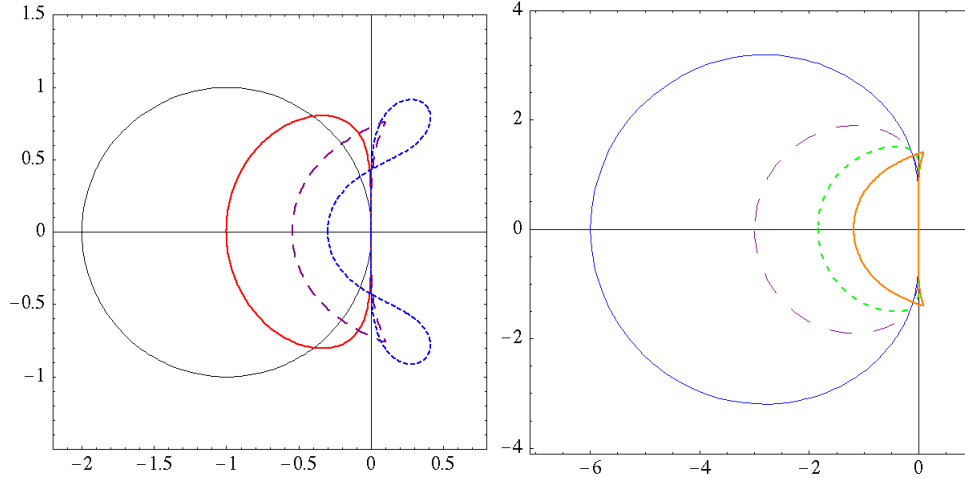
Thus there are constants $a_0, a_1, a_2, \dots, a_{\nu}$ and b_0 , with $a_0 = 1$, such that

$$\sum_{k=0}^{\nu} a_k y_{\nu-k} = hb_0 f(t_n, y_n) \quad (6.105)$$

The first several BDF methods using this formula are shown in Table 6.3, BDF methods are more stable than Adam's-Moulton methods and have the advantage of being explicit like Adams-Bashforth. Unfortunately they are unstable for all values of $\nu \geq 7$.

Example 6.4. Derive the BDF formula for $\nu = 2$.

Figure 6.1: Outer border of regions of stability for Adams Methods. Left: Adams-Bashforth methods AB1 (forward Euler, solid black); AB2 (Thick solid red); AB3 (Thick dashed purple); and AB4 (thick, short blue dashed). Right: Adams-Moulton Methods AM3 (thin, blue); AM4 (thin, long dashed purple); AM5 (short-dashed green); AM6 (thick, solid, orange).



Solution. From equation 6.104

$$hf_n = \sum_{k=1}^2 \frac{1}{k} \nabla^k y_k \quad (6.106)$$

$$= \nabla y_n + \frac{1}{2} \nabla^2 y_n \quad (6.107)$$

$$= y_n - y_{n-1} + \frac{1}{2}(y_n - 2y_{n-1} + y_{n-2}) \quad (6.108)$$

$$= \frac{3}{2}y_n - 2y_{n-1} + \frac{1}{2}y_{n-2} \quad (6.109)$$

$$2hf_n = 3y_n - 4y_{n-1} + y_{n-2} \quad \square \quad (6.110)$$

6.6 Nyström and Milne Methods

These methods consider the integral

$$y(t_n) = y(t_{n-2}) + \int_{t_{n-2}}^{t_n} f(t, y(t)) dt \quad (6.111)$$

Substitution of Newton's backward difference formula yields **Nyström's Method**

$$y_n = y_{n-2} + h \sum_{j=0}^{\nu-1} k_j \nabla^j f_{n-1} \quad (6.112)$$

Table 6.3: The first several Backward Differentiation Formulae.

Method	Formula for the method
BDF1	$hf_n = y_n - y_{n-1}$ (Backward Euler)
BDF2	$2hf_n = 3y_n - 4y_{n-1} + y_{n-2}$
BDF3	$6hf_n = 11y_n - 18y_{n-1} + 9y_{n-2} - 2y_{n-3}$
BDF4	$12hf_n = 25y_n - 48y_{n-1} + 36y_{n-2} - 16y_{n-3} + 3y_{n-4}$
BDF5	$60hf_n = 137y_n - 300y_{n-1} + 300y_{n-2} - 200y_{n-3} + 75y_{n-4} - 12y_{n-5}$
BDF6	$60hf_n = 147y_n - 360y_{n-1} + 450y_{n-2} - 400y_{n-3} + 225y_{n-4} - 72y_{n-5} + 10y_{n-6}$

where

$$k_j = (-1)^j \int_{-1}^1 \binom{-s}{j} ds \quad (6.113)$$

To obtain the **Milne-Simpson method** our interpolation includes f_n

$$y_n = y_{n-2} + h \sum_{j=0}^{\nu} k_j \nabla^j f_n \quad (6.114)$$

where

$$k_j = (-1)^j \int_{-1}^1 \binom{-s+1}{j} ds \quad (6.115)$$

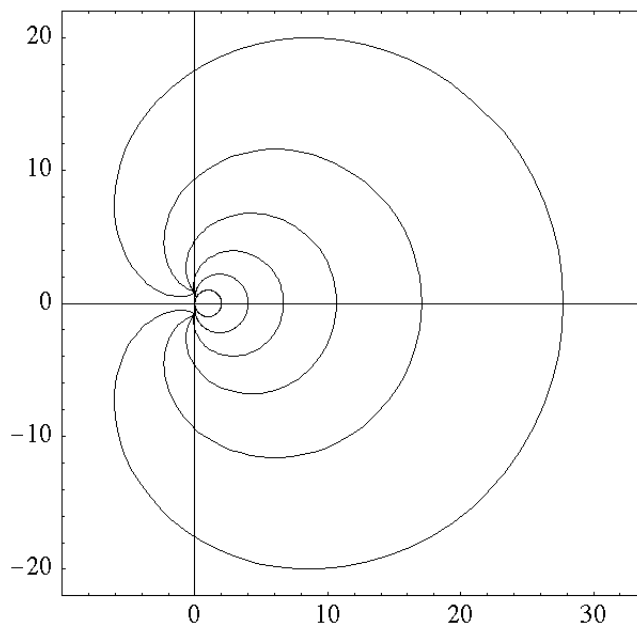
Table 6.4: Coefficients k_j for the Nyström Methods.

j	0	1	2	3	4	5	6	7	8
k_j	2	0	1/3	1/3	29/90	14/45	1139/3780	41/140	32377/113400

Table 6.5: Coefficients k_j for the Milne-Simpson Methods.

j	0	1	2	3	4	5	6	7	8
k_j	2	-2	1/3	0	-1/90	-1/90	-37/3780	-8/945	-119/16200

Figure 6.2: Boundaries of stability regions for BDF methods BDF1 (inside curve) through BDF6 (outside). The stability region is *outside* the bounded area shown.



6.7 Other Types of Multistep Methods

Enright's Second Derivative Method is based on the observation that a generalization of the method

$$y_n = y_{n-1} + hf_n \quad (6.116)$$

gives

$$y_n = y_{n-1} + hy'_n + \frac{1}{2}h^2y''_n \quad (6.117)$$

$$= y_{n-1} + hf_n + \frac{1}{2}h^2y''_n \quad (6.118)$$

by using an additional term in the Taylor expansion. To get an expression for y''_n we observe that if

$$y' = f(t, y) \quad (6.119)$$

then

$$y'' = f_t + f_y f \quad (6.120)$$

Hence

$$y_n = y_{n-1} + hf_n + \frac{1}{2}h^2g_n \quad (6.121)$$

Table 6.6: Milne-Simpson Methods

Method	Formula for y_n
MS0	$y_n = y_{n-2} + 2hf_n$ (Backward Euler Method, step size $2h$)
MS1	$y_n = y_{n-2} + 2hf_{n-1}$ (Midpoint Rule)
MS2	$3y_n = y_{n-2} + h(f_n + 4f_{n-1} + f_{n-2})$ (Milne's Method)
MS3	$90y_n = y_{n-2} + h(29f_n + 124f_{n-1} + 24f_{n-2} + 4f_{n-3} - f_{n-4})$

where $g = f_t + f_y f$. If we use a weighted combination of values at multiple grid points the result is **Enright's multistep second derivative method**, which is

$$\sum_{i=0}^{\nu} a_i y_{n+i} = h \sum_{i=0}^{\nu} b_i f_{n+i} + h^2 \sum_{i=0}^{\nu} c_i g_{n+i} \quad (6.122)$$

It can be shown by using a Taylor series approximation that this method is of order p if and only if

$$\sum_{j=0}^{\nu} a_j (j)^q = q \sum_{j=0}^{\nu} b_j (j)^{q-1} + q(q+1) \sum_{j=0}^{\nu} c_j (j)^{q-2}, \quad \forall j = 0, 1, 2, \dots, p \quad (6.123)$$

The following simplifications are usually made:

- Set $a_{\nu} = 1$, $a_{\nu-1} = -1$ and the rest of the $a_k = 0$. This ensures stability in a neighborhood of the origin.
- Set $c_i = 0$ for all $i < \nu$ to ensure stability at infinity.

The simplified method is then

$$y_n = y_{n-1} + h \sum_{j=0}^{\nu} b_j f_{n+j-\nu} + h^2 c_{\nu} g_n \quad (6.124)$$

Formulas for the first several methods are given in table 6.7.¹

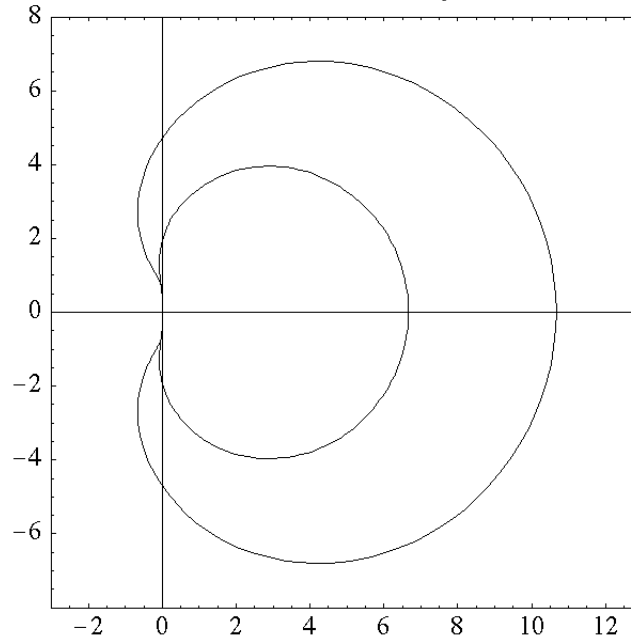
Blended Multistep Methods² attempt to take advantage of the features of both Adams methods, which are good at non-stiff problems, and BDF methods, which are good for stiff problems. The methods are blended in the sense that

$$[\text{Adams Method Expression}]^{(\nu+1)} = \gamma^{(\nu)} h f_y [\text{BDF Expression}]^{(k)} \quad (6.125)$$

¹See Enright, W.H., "Second Derivative Multistep Methods for Stiff Ordinary Differential Equations", SIAM Journal of Numerical Analysis, 11(2):321-331 (1974) for a table of values up through $\nu = 7$ (which is 9th order).

²See Skeel, R.D. and Kong, A.K., *Blended linear multistep methods*, ACM Transactions on Mathematical Software, 3:326-343 (1977).

Figure 6.3: Borders of stability region for Milne-Simpson MS3 and MS3 methods. The stable region is outside the indicated boundary.



where the superscripts denote the order of method and are not exponents are derivatives, γ is a free parameter, and hf_y is a weighting factor (for systems, we replace f_y with the Jacobian matrix). Weighting factors are chosen based on the observation that for nonstiff problems, $-hf_y$ is small, while for stiff problems, $-hf_y$ is large. Here the “Adams Method Expression” is

$$-y_{n+1} + y_n + h(b_\nu f_{n+1} + b_{\nu-1}f_n + \cdots + b_0 f_{n-\nu+1}) = 0 \quad (6.126)$$

and the BDF expression is

$$-(a_\nu y_{n+1} + a_{\nu-1}y_n + \cdots + a_0 y_{n-k+1}) + hf_{n+1} = 0 \quad (6.127)$$

For example, if we chose $\nu = 2$, this method gives

$$y_{n+1} = y_n + h \left(\frac{5}{12} f_{n+1} + \frac{8}{12} f_n - \frac{1}{12} f_{n-1} \right) \quad (6.128)$$

$$- \gamma^{(2)} hf_y \left(-\frac{3}{2} y_{n+1} + 2y_n - \frac{1}{2} y_{n-1} + hf_{n+1} \right) \quad (6.129)$$

For *autonomous linear equations*, we have $f = y' = f_y y$ and hence

- if $\gamma^{(2)} = 1/6$, cancellation of f_{n-1} and $f_y y_{n-1}$ terms leads to the $\nu - 1$ -step Enright method;
- if $\gamma^{(2)} = 1/8$, we obtain the ν -step Enright method.

Table 6.7: Enright Second Derivative Multistep Methods. The methods have order $\nu + 2$.

Method (ν)	Iteration Formula
1	$y_n = y_{n-1} + h \left(\frac{2}{3}f_n + \frac{1}{3}f_{n-1} \right) - \frac{1}{6}h^2g_n$
2	$y_n = y_{n-1} + h \left(\frac{29}{48}f_n + \frac{5}{12}f_{n-1} - \frac{1}{48}f_{n-2} \right) - \frac{1}{8}h^2g_n$
3	$y_n = y_{n-1} + h \left(\frac{307}{540}f_n + \frac{19}{40}f_{n-1} - \frac{1}{20}f_{n-2} + \frac{7}{1080}f_{n-3} \right) - \frac{19}{480}h^2g_n$
4	$y_n = y_{n-1} + h \left(\frac{3133}{4760}f_n + \frac{47}{90}f_{n-1} - \frac{41}{480}f_{n-2} + \frac{1}{45}f_{n-3} - \frac{17}{5760}f_{n-4} \right) - \frac{3}{32}h^2g_n$

6.8 Prediction-Correction (P(EC)ⁿE) Techniques

At each iteration step in an implicit linear multistep method such as

$$a_0\mathbf{y}_n + a_1\mathbf{y}_{n-1} + \cdots + a_k\mathbf{y}_{n-k} = h(b_0\mathbf{f}_n + b_1\mathbf{f}_{n-1} + \cdots + b_k\mathbf{f}_{n-k}) \quad (6.130)$$

where y and f are m -vectors (for systems, or scalars for a scalar equation), and $\mathbf{f}_p = \mathbf{f}(t_p, \mathbf{y}_p)$, we are faced with the problem of knowing what value of \mathbf{y}_n to use in the term in \mathbf{f}_n in the right hand side of the equation. More specifically, we have

$$a_0\mathbf{y}_n = hb_0\mathbf{f}(t_n, \mathbf{y}_n) - a_1\mathbf{y}_{n-1} - \cdots - a_k\mathbf{y}_{n-k} + h(+b_1\mathbf{f}_{n-1} + \cdots + b_k\mathbf{f}_{n-k}) \quad (6.131)$$

where \mathbf{y}_n appears on both sides of the equation.

When the system is not stiff, it is generally possible to find a step size h such that

$$\left\| hb_0 \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right\| \leq r < 1 \quad (6.132)$$

for some number r , in which case the method is a contraction (for scalar equations, $|hb_0f_y| \leq r$), the fixed-point theorem applies, and one can use fixed point iteration to find \mathbf{y}_n given a reasonably good starting guess. The general heuristic is to use an explicit method of the same order for the first guess. This is called the prediction (P) step. This first guess for y_n is used to estimate (E-step) the value of $f(t, y_n)$; then the implicit method is used to calculate a corrected (C-step) estimate of y_n . This corrected estimate is substituted back into the method for one final calculation (E-step). This type of method is often called a PEC, PECE, P(EC)ⁿ, or P(EC)ⁿE method, depending on the number of estimation - correction iterations. These **prediction-correction** are essentially explicit methods even though they use an implicit formula, because the method is based on an explicit evaluation of the function at the previous step, rather than using a solver to determine the volume.

If the equation is stiff, then a PECE method is not sufficient, because it is extremely expensive to find a value of h such that $|hb_0f_y| \leq r < 1$. Instead, the value

of y_n that is on both sides of the equation should be solved for using a Newton iteration as has been discussed previously in connection with the backward Euler method.

Algorithm 6.1. Prediction-Correction Method *To solve the initial value problem*

$$y' = f(t, y), y(t_0) = y_0$$

for the function $y(t)$, at each time step, let EM_n and IM_n represent predictions at time-step n using an explicit method (EM) and an implicit method (IM), respectively.

P Predict $y_n^{(0)} = EM_n$

Repeat n times ($n = 1$ for PEC and PECE methods)

E Estimate: $f_n^{(0)} = f(t_n, y_n^{(0)})$

C Correct: $y_n^{(1)} = IM(t_n, y_n^{(0)})$.

PEC methods stop here after the first iteration. P(EC) n methods stop here after n iterations.

E : Estimate: $f_n^{(1)} = f(t_n, y_n^{(1)})$ to be used in the next iteration. PECE and P(EC) n E methods stop here.

Chapter 7

Delay Differential Equations

7.1 Basic Theory

Definition 7.1 (Delay Differential Equation). *Let*

$$0 \leq \tau_1 < \tau_2 < \cdots < \tau_m \quad (7.1)$$

*be a set of m positive numbers representing the delays, the largest delay being denoted by τ_m . If y appears explicitly without any delay then $\tau_1 = 0$. Then we may define the scalar **delay differential equation (DDE)** as*

$$y'(t) = f(t, y(t - \tau_1), y(t - \tau_2), \dots, y(t - \tau_m)) \quad (7.2)$$

$$y(t) = g(t), \quad t_0 - \tau_m \leq t \leq t_0 \quad (7.3)$$

Theorem 7.1. *Let $D \in [t_0, t_0 + \beta) \times \mathbb{R}^m$ be convex; suppose that $f : D \mapsto \mathbb{R}$ is continuous on D ; and that $g(t) : [t_0 - \tau_m] \mapsto D$. Then if $r_1 \neq 0$ the DDE (equations 7.2, 7.3) has a unique solution on $[t_0, t_0 + \alpha)$, where $0 < \alpha \leq \beta$. Furthermore, if $\alpha < \beta$ then $y(t)$ approaches the boundary of D as $t \rightarrow \alpha$. If $\tau_1 = 0$ then the DDE has at most one solution.*

Proof. The proof is constructive. We consider two cases:

Case 1: $\tau_1 \geq 0$: For $t_0 \leq t \leq \tau_1 + t_0$, each $y(t - \tau_i) = g(t - \tau_i)$, hence we have an explicit expression for $f(t)$, viz.,

$$y'(t) = f(t, g(t - \tau_1), g(t - \tau_2), \dots, g(t - \tau_m)) \quad (7.4)$$

$$y(t_0) = g(t_0) \quad (7.5)$$

Hence we can integrate on (t_0, t) where $t < \tau_1 + t_0$,

$$y(t) = \int_{t_0}^t y'(t) dt = g(t_0) + \int_{t_0}^t f(x, g(x - \tau_1), \dots, g(x - \tau_m)) dx \quad (7.6)$$

This uniquely determines $y(t)$, $t_0 \leq t \leq \tau_1 + t_0$. So we can repeat the process on $[t_0 + \tau_1, t_0 + \min(\beta, 2\tau_1)]$. We can repeat this process until we reach β . This process is called the **method of steps**. Thus by construction, a solution exists.

Case 2: $\tau_1 = 0$: For $t_0 \leq t \leq t_0 + \tau_2$,

$$y'(t) = f(t, g(t), g(t - \tau_2), \dots, g(t - \tau_m)) \quad (7.7)$$

$$= f(t, y(t), g(t - \tau_2), \dots, g(t - \tau_m)) \quad (7.8)$$

$$y(0) = g(0) \quad (7.9)$$

By the fundamental theorem, If f is Lipschitz in y then at most one solution exists. □

Lemma 7.1 (Gronwall Inequality). *Suppose that $f, g : [a, b] \mapsto \mathbb{R}$ are continuous, non-negative functions such that*

$$f(t) \leq K + \int_a^t f(x)g(x)dx \quad (7.10)$$

for some non-negative constant K . Then

$$f(t) \leq K \exp\left(\int_a^t g(x)dx\right) \quad (7.11)$$

Proof. Define the functions

$$F(t) = K + \int_a^t f(x)g(x)dx \quad (7.12)$$

$$G(t) = \int_a^t g(x)dx \quad (7.13)$$

Then

$$F(a) = K \quad (7.14)$$

$$G(a) = 0 \quad (7.15)$$

$$G'(t) = g(t) \quad (7.16)$$

The assumption 7.10 becomes

$$f(t) \leq F(t) \quad (7.17)$$

hence by differentiating 7.12 we obtain

$$F'(t) = f(t)g(t) \leq F(t)g(t) \quad (7.18)$$

Multiplying both sides through by $\exp(-G(t))$,

$$F'(t)e^{-G(t)} \leq F(t)g(t)e^{-G(t)} \quad (7.19)$$

Differentiating $F(t)e^{-G(t)}$ and applying equations 7.16 and 7.18 gives

$$\frac{d}{dt}F(t)e^{-G(t)} = F'(t)e^{-G(t)} - F(t)G'(t)e^{-G(t)} \quad (7.20)$$

$$= (F'(t) - F(t)G'(t))e^{-G(t)} \quad (7.21)$$

$$= (F'(t) - F(t)g(t))e^{-G(t)} \leq 0 \quad (7.22)$$

Integrating the left hand side over $[a, t]$ gives

$$0 > \int_a^t \frac{d}{dt} F(x) e^{-G(x)} dx \quad (7.23)$$

$$> F(t) e^{-G(t)} - F(a) e^{-G(a)} \quad (7.24)$$

From equations 7.14 and 7.15,

$$K > F(t) e^{-G(t)} \quad (7.25)$$

and therefore $f(t) \leq F(t) \leq K e^{G(t)}$, which is equation 7.11. \square

Definition 7.2. (*Strong Lipschitz Condition*) Let f be defined on $[0, \beta) \times D^m$. Then it satisfies the **strong Lipschitz condition** if

$$|f(t, x_1, \dots, x_m) - f(t, \tilde{x}_1, \dots, \tilde{x}_m)| \leq K \max_{1 \leq j \leq m} |x_j - \tilde{x}_j| \quad (7.26)$$

Theorem 7.2 (Error Growth). Let f be continuous on $[t_0, t_0 + \beta) \times D^m$; suppose f satisfies a strong Lipschitz condition with Lipschitz constant K ; and suppose that y and \tilde{y} are solutions of

$$y'(t) = f(t, y(t - \tau_1), \dots, y(t - \tau_m)) \quad (7.27)$$

with corresponding continuous initial functions $\theta(t)$ and $\tilde{\theta}(t)$ on $[t_0 - r, t_0]$,

$$y(t) = \theta(t), \quad t_0 - r \leq t \leq t_0 \quad (7.28)$$

$$\tilde{y}(t) = \tilde{\theta}(t), \quad t_0 - r \leq t \leq t_0 \quad (7.29)$$

Then

$$|y(t) - \tilde{y}(t)| \leq \sup_{t_0 - r \leq x \leq t_0} |\theta(x) - \tilde{\theta}(x)| e^{Kt} \quad (7.30)$$

on $t_0 \leq t < t_0 + \beta$.

Proof. Let

$$v(t) = \sup_{t_0 - r \leq x \leq t} |y(x) - \tilde{y}(x)| \quad (7.31)$$

Then

$$y(t) = \theta(0) + \int_{t_0}^t f(x, \theta(x - \tau_1), \dots, \theta(x - \tau_m)) dx \quad (7.32)$$

$$\tilde{y}(t) = \tilde{\theta}(0) + \int_{t_0}^t f(x, \tilde{\theta}(x - \tau_1), \dots, \tilde{\theta}(x - \tau_m)) dx \quad (7.33)$$

so that

$$|y(t) - \tilde{y}(t)| \leq |\theta(t_0) - \tilde{\theta}(t_0)| + \int_{t_0}^t |f(x, \theta(x - \tau_1), \dots, \theta(x - \tau_m)) \quad (7.34)$$

$$- f(x, \tilde{\theta}(x - \tau_1), \dots, \tilde{\theta}(x - \tau_m))| dx \quad (7.35)$$

Since f is strongly Lipschitz,

$$|f(t, y_1, \dots, y_m) - f(t, \tilde{y}_1, \dots, \tilde{y}_m)| \leq K \max_{1 \leq j \leq m} |y_j - \tilde{y}_j| \quad (7.36)$$

and therefore

$$|y(t) - \tilde{y}(t)| \leq |\theta(t_0) - \tilde{\theta}(t_0)| + K \int_{t_0}^t \max_{1 \leq j \leq m} |\theta(x - \tau_j) - \tilde{\theta}(x - \tau_j)| dx \quad (7.37)$$

$$\leq v(t_0) + K \int_{t_0}^t v(x) dx \quad (7.38)$$

Applying the definition of $v(t)$ one more time,

$$v(t) \leq v(t_0) + K \int_{t_0}^t v(x) dx \quad (7.39)$$

By Gronwall's inequality,

$$v(t) \leq v(t_0) \exp \int_{t_0}^t v(x) dx \quad (7.40)$$

from which 7.30 follows immediately. □

7.2 Method Of Steps

Consider the DDE with a single delay,

$$y'(t) = f(t - \tau, y(t - \tau)) \quad (7.41)$$

$$y(t) = g(t), \quad t_0 - \tau \leq t \leq t_0 \quad (7.42)$$

where we will assume that $g(t_0 - \tau) = g(t_0)$. Then on the interval $[t_0, t_0 + \tau]$ we have

$$y(t) = g(t_0) + \int_{t_0}^t f(s - \tau, y(s - \tau)) ds \quad (7.43)$$

$$= g(t_0) + \int_{t_0}^t f(s - \tau, g(s - \tau)) ds \quad (7.44)$$

The right hand side is fully explicit and can be evaluated. This gives the solution on the first "step," $[t_0, t_0 + \tau]$. Next, assume we have already solved the DDE on $[t_0, t_0 + k\tau]$ for some integer $k > 0$. (This statement is immediately true for $k = 1$ once we have solved the first interval). Then on subsequent steps $[t_0 + k\tau, t_0 + (k + 1)\tau]$, for $k = 1, 2, \dots$,

$$y(t) = y(t_0 + k\tau) + \int_{t_0 + k\tau}^t f(s - \tau, y(s - \tau)) ds \quad (7.45)$$

Since the integrals are evaluated in order, i.e., the integral is solved on the interval $[t_0 + (k - 1)\tau, t_0 + k\tau]$ first, everything on the right-hand side of the equation is already known, so the integral can be evaluated.

Example 7.1. Solve the linear test equation

$$y' = \lambda y(t - 1) \quad (7.46)$$

$$y(t) = 1, \quad -1 \leq t \leq 0 \quad (7.47)$$

on the interval $[0, 3]$.

Solution. On $[0, 1]$,

$$y(t) = y(0) + \int_0^t \lambda y(s - 1) ds \quad (7.48)$$

$$= 1 + \lambda \int_0^t ds \quad (7.49)$$

$$= 1 + \lambda t \quad (7.50)$$

At $t = 1$ we have $y(1) = 1 + \lambda$, so on the interval $[1, 2]$, ←

$$y(t) = y(1) + \int_1^t \lambda y(s - 1) ds \quad (7.51)$$

$$= 1 + \lambda + \lambda \int_1^t (1 + \lambda s)|_{s \rightarrow s-1} ds \quad (7.52)$$

$$= 1 + \lambda + \lambda \int_1^t (1 + \lambda(s - 1)) ds \quad (7.53)$$

$$= 1 + \lambda + \lambda \left(-1 + t + \frac{\lambda}{2} - t\lambda + \frac{t^2\lambda}{2} \right) \quad (7.54)$$

$$= 1 + t\lambda + \frac{\lambda^2}{2} - t\lambda^2 + \frac{t^2\lambda^2}{2} \quad (7.55)$$

The next iteration gives the solution on $[2, 3]$ as ←

$$y(t) = y(2) + \int_2^t \left(1 + (s - 1)\lambda + \frac{\lambda^2}{2} - (s - 1)\lambda^2 + \frac{(s - 1)^2\lambda^2}{2} \right) ds \quad (7.56)$$

$$= 1 + \frac{\lambda^2}{2} - \frac{4\lambda^3}{3} + \frac{t^3\lambda^3}{6} + t^2 \left(\frac{\lambda^2}{2} - \lambda^3 \right) + t(\lambda - \lambda^2 + 2\lambda^3) \quad \square \quad (7.57)$$

This technique is repetitive (much like Picard's method) and easily implemented in Mathematica. The following code will solve the DDE

$$y' = \lambda y(t - \tau) \quad (7.58)$$

$$y(t) = g(t), \quad -\tau < t < 0 \quad (7.59)$$

It will print out the solution, one step at a time, for n steps:

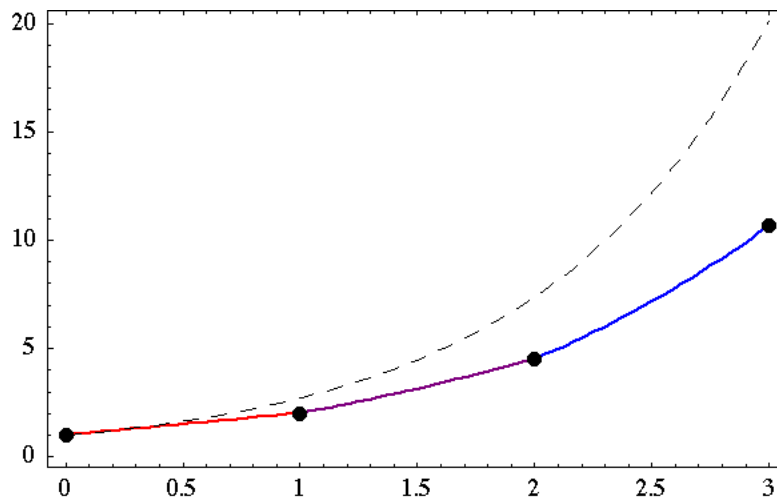
```
methodOfSteps[g_, t_, tau_, n_] :=
Module[{y0, y, s, i},
  y0 = g[0];
```

```

y = y0 + Integrate[g[s - tau], {s, 0, t} ];
Print["y(t)=", y, " on [0, ", tau, " ]"];
For[i = 1, i n - 1, i++,
  a = i*tau;
  b = a + tau;
  y0 = y /. t -> a;
  y = y0 + Integrate[y /. {t -> s - 1}, {s, a, t} ];
  y = Expand[y];
  Print["y(t)=", y, " on [", a, ", ", b, " ]"];
]
]

```

Figure 7.1: Solution found in example for 7.1 with $\lambda = 1$. The dots show the endpoints of each step. The solution of the initial value problem $y' = \lambda y, y(1) = 1$ is indicated by the dashed lines.



To run the program one must first define the function $g(t)$. For example, to solve the DDE

$$y' = g(t - \tau) \quad (7.60)$$

$$y(t) = \lambda e^t, \quad -\tau < t < 0 \quad (7.61)$$

the following must be typed into mathematica:

```

g[t_]:=λExp[t]
methodOfSteps[g, τ, 1, 5]

```

The corresponding output that will be printed is:

$$\begin{aligned}
y(t) &= \lambda + \frac{(-1 + e^t) \lambda}{e} \quad \text{on } [0, 1] \\
y(t) &= \lambda - \frac{\lambda}{e} + e^{-2+t} \lambda + t \lambda - \frac{t \lambda}{e} \quad \text{on } [1, 2] \\
y(t) &= 2 \lambda - \frac{2 \lambda}{e} + e^{-3+t} \lambda + \frac{t^2 \lambda}{2} - \frac{t^2 \lambda}{2e} \quad \text{on } [2, 3] \\
y(t) &= e^{-4+t} \lambda + \frac{5 t \lambda}{2} - \frac{5 t \lambda}{2e} - \frac{t^2 \lambda}{2} + \frac{t^2 \lambda}{2e} + \frac{t^3 \lambda}{6} - \frac{t^3 \lambda}{6e} \\
&\quad \text{on } [3, 4] \\
y(t) &= 5 \lambda - \frac{5 \lambda}{e} + e^{-5+t} \lambda - \frac{19 t \lambda}{6} + \frac{19 t \lambda}{6e} + 2 t^2 \lambda - \\
&\quad \frac{2 t^2 \lambda}{e} - \frac{t^3 \lambda}{3} + \frac{t^3 \lambda}{3e} + \frac{t^4 \lambda}{24} - \frac{t^4 \lambda}{24e} \quad \text{on } [4, 5]
\end{aligned}$$

While the solution in the first example behaves in a manner that is similar (albeit numerically different) from the IVP with $y(0) = g(0)$ the following slightly different example behaves quite differently.

Example 7.2. Use Mathematica to solve the DDE

$$y'(t) = -y(1-t) \quad (7.62)$$

$$y(t) = 1, \quad -\tau < t < 0 \quad (7.63)$$

on the interval $[0, 10]$ and plot the results.

Solution. We can (almost) trivially modify our previous code to solve the new differential equation, which has a minus sign on the right hand side.

```

methodOfSteps2[g_, t_, tau_, n_] :=
Module[{y0, y, s, i},
  y0 = g[0];
  y = y0 - Integrate[g[s - tau], {s, 0, t}];
  Print["y(t)=", y, " on [0, ", tau, "]];
  For[i = 1, i < n - 1, i++,
    a = i*tau;
    b = a + tau;
    y0 = y /. t -> a;
    y = y0 - Integrate[y /. {t -> s - 1}, {s, a, t}];
    y = Expand[y];
    Print["y(t)=", y, " on [", a, ", ", b, "]];
  ]
]

```

The calling sequence is

```
g[t_] := 1; methodOfSteps2[g, t, 1, 10];
```

which prints the following output:

$$\begin{aligned}
 y(t) &= 1 - t \text{ on } [0, 1] \\
 y(t) &= \frac{3}{2} - 2t + \frac{t^2}{2} \text{ on } [1, 2] \\
 y(t) &= \frac{17}{6} - 4t + \frac{3t^2}{2} - \frac{t^3}{6} \text{ on } [2, 3] \\
 y(t) &= \frac{149}{24} - \frac{17t}{2} + \frac{15t^2}{4} - \frac{2t^3}{3} + \frac{t^4}{24} \text{ on } [3, 4] \\
 y(t) &= \frac{1769}{120} - \frac{115t}{6} + \frac{109t^2}{12} - 2t^3 + \frac{5t^4}{24} - \frac{t^5}{120} \text{ on } [4, 5] \\
 y(t) &= \frac{26239}{720} - \frac{1085t}{24} + \frac{1061t^2}{48} - \frac{197t^3}{36} + \frac{35t^4}{48} - \frac{t^5}{20} + \frac{t^6}{720} \text{ on } [5, 6] \\
 y(t) &= \frac{463609}{5040} - \frac{13201t}{120} + \frac{13081t^2}{240} - \frac{521t^3}{36} + \frac{107t^4}{48} - \frac{t^5}{5} + \frac{7t^6}{720} - \frac{t^7}{5040} \text{ on } [6, 7] \\
 y(t) &= \frac{3157891}{13440} - \frac{39371t}{144} + \frac{39227t^2}{288} - \\
 &\quad \frac{27227t^3}{720} + \frac{3685t^4}{576} - \frac{487t^5}{720} + \frac{7t^6}{160} - \frac{t^7}{630} + \frac{t^8}{40320} \text{ on } [7, 8] \\
 y(t) &= \frac{43896157}{72576} - \frac{1158379t}{1680} + \frac{1156699t^2}{3360} - \frac{212753t^3}{2160} + \\
 &\quad \frac{51193t^4}{2880} - \frac{1511t^5}{720} + \frac{701t^6}{4320} - \frac{t^7}{126} + \frac{t^8}{4480} - \frac{t^9}{362880} \text{ on } [8, 9] \\
 y(t) &= \frac{5681592251}{3628800} - \frac{23615939t}{13440} + \frac{23602499t^2}{26880} - \frac{7761511t^3}{30240} + \frac{279533t^4}{5760} - \\
 &\quad \frac{89269t^5}{14400} + \frac{1873t^6}{3456} - \frac{323t^7}{10080} + \frac{11t^8}{8960} - \frac{t^9}{36288} + \frac{t^{10}}{3628800} \text{ on } [9, 10]
 \end{aligned}$$

To automatically plot the results instead of printing them out, we modify our code block as follows.

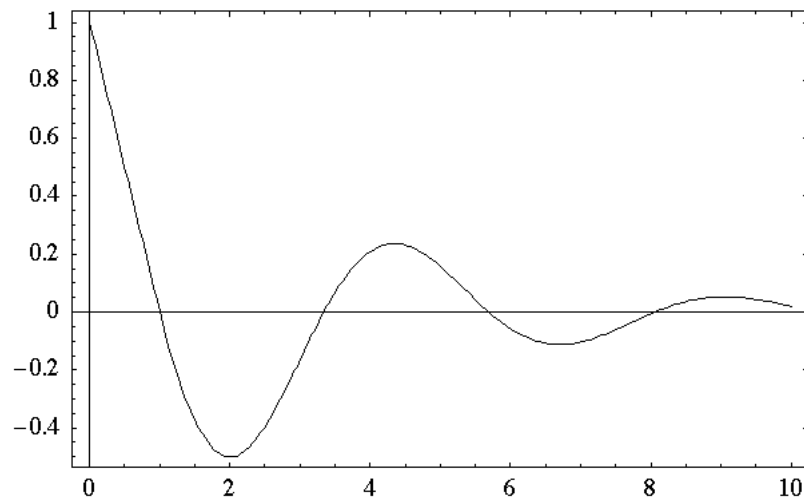
```

methodOfSteps2p[g_, t_, tau_, n_] :=
Module[{y0, y, s, i, p = {}},
  y0 = g[0];
  y = y0 - Integrate[g[s - tau], {s, 0, t}];
  AppendTo[p, Plot[y, t, 0, tau, DisplayFunction -> Identity]];
  For[i = 1, i < n - 1, i++,
    a = i*tau;
    b = a + tau;
    y0 = y /. {t -> a};
    y = y0 - Integrate[y /. {t -> s - 1}, {s, a, t}];
    y = Expand[y];
    AppendTo[p, Plot[y, {t, a, b}, DisplayFunction -> Identity]];
  ];
  Return[Show[p, DisplayFunction -> $DisplayFunction]];
]

```

The plot is then generated with

```
methodOfSteps2p[g, τ, 1, 10]
```



We observe that the solution is a decaying oscillation, whereas the corresponding initial value problem without delay would give a pure exponential decay. Thus the two solutions are qualitatively different.

7.3 Numerical Implementation of Method of Steps

In this section we illustrate the numerical solution of a delay differential equation using the method of steps via several examples. These examples illustrate that the solution of the DDE can vary radically by minor changes in the initial data or the parameters of the DDE, such as the size of the delay.

We will begin with the example from the previous section:

$$y'(t) = -y(1 - t) \quad (7.64)$$

$$y(t) = 1, -\tau < t < 0 \quad (7.65)$$

One key drawback of the method of steps is that the step size must be an integral divisor of the delay. Thus for some integer n , we must have $nh = \tau$. So long as there is only a single delay and the delay is not a function of time, we can get around this by automatically setting the mesh size based on the number of points we want to have in the mesh. For a fixed step size, this would give us $h = n/\tau$ with n an input parameter; for a variable step size, we would be limited to multiples of this step.

Algorithm 7.1. Method of Steps *To solve the delay differential equation*

$$y'(t) = f(t, y(t), y(t - \tau)) \quad (7.66)$$

$$y(t) = g(t), t_0 - \tau < t < t_0 \quad (7.67)$$

on the interval $(t_0, K\tau)$, $K \in \mathbb{Z}^+$.

1. Input n , number of points in the mesh; set $h = \tau/n$.
2. For $i = 0, \dots, K - 1$,
 - (a) Set $p(t) = y(t - \tau)$ (either as a function or an array of data to index into).
 - (b) Solve the following IVP using any solver you choose:

$$y'(t) = f(t, y(t), p(t)) \quad (7.68)$$

$$y(t_0 + i\tau) = p(t_0) \quad (7.69)$$

on the interval $(t_0 + i\tau, t_0 + (i + 1)\tau)$.

We choose to store our initial data as a set of points evaluated at the delayed times $t_0 - \tau, t_0 - \tau + h, t_0 - \tau + 2h, \dots, t_0$ in an array. The conversion between the array index i and the time t is given by the formula

$$i = 1 + n(1 + t/\tau) \quad (7.70)$$

This formula works fine so long as t falls on a mesh point; if not, we have two choices: interpolation (more accurate); and rounding (simpler). Since this is an illustrative implementation, we choose the simpler method, and force the value of i to be an integer in the following code:

```
index[t_, tau_, n_] := Module[{} ,
  If[t < -tau , Return[$Failed]];
  Return[Floor[1 + n(1 + t/tau)]];
];
```

The function `index[t, tau, n]` returns the value of i corresponding to a given value of t , based on the mesh size n and delay τ . Assuming that our initial data is stored in the array `data`, to get the value of $f(t - \tau)$ would call

```
data[[index[t-tau, tau, n]]]
```

and to implement our function $f(t) = -y(t - \tau)$

```
f[t_] := -1.0*data[[index[t - tau, tau, n]]];
```

The function `steps[n, tmax]` then solves the differential equation using the method of steps on a mesh of size `n` mesh points per τ on an interval (t_0, t_{max}) and returns a plot of the results.

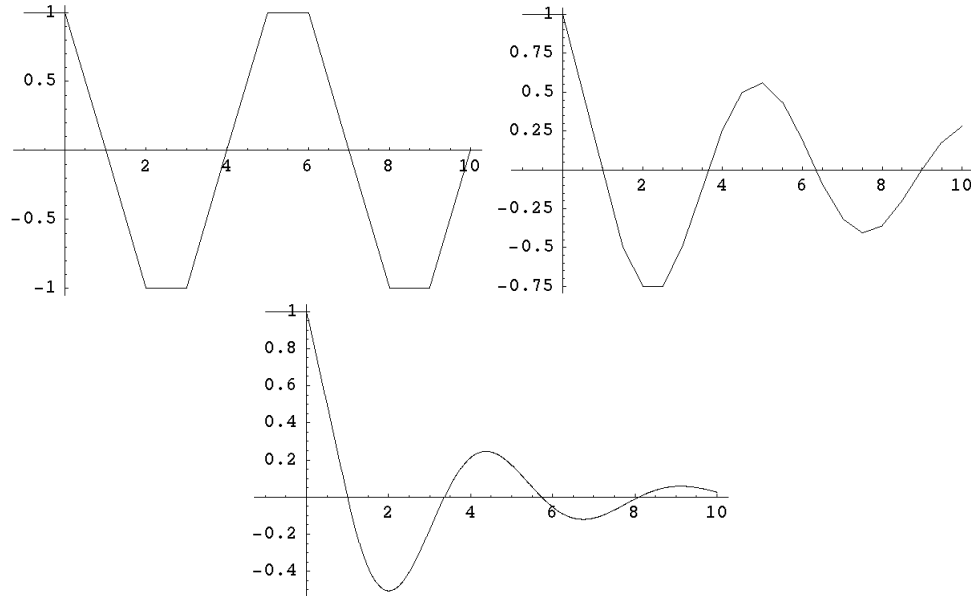
```
steps[n_, tmax_, tau_] := Block[
  { times, data, results, f, t0, y0, h, k},
  (** define initial data  $g(t)=1, t_0 - \tau < t < t_0$  **)
  data = Table[1, {n + 1}];
  times = Table[-tau + i/n, {i, 0., n, 1.0}];
  results = {times, data} // Transpose;
  (** define right hand side of DDE **)
  f[time_] := -1.0*data[[index[time - tau, tau, n]]];
  (** set up parameters for Euler Integration **)
  h = tau/n;
  t0 = Last[times];
  y0 = Last[data];
  (** Each iteration of the For loop Solves the DDE
      on  $(t_0 - (k - 1)\tau, t_0 + k\tau)$  **)
  For[k = 1, k ≤ tmax/tau, k++,
    Block[{t, i, y}, (** Block defines local variables **)
      t = t0; y = y0;
      For[i = 1, i ≤ n, i++, (** Euler Integration **)
        y = y + h*f[t];
        t = t + h;
        AppendTo[results, {t, y}];
        AppendTo[data, y];
      ];
      t0 = t; y0 = y;
    ];
  ];
  Return[ListPlot[results, PlotJoined -> True]]
];
```

We run this program three times with $\tau = 1$ on the interval $(0, 10)$ using $n = 1$, $n = 2$, and $n = 100$,

```
p1=steps[1,10,1]
p2=steps[2,10,1]
p3=steps[100,10,1]
```

Figure 7.2 shows the results of images p1, p2, and p3.

Figure 7.2: Numerical integration of $y' = -y(t - 1)$, $y(t) = 1, -1 < t < 0$, on the interval $(0, 10)$ for three different step sizes using the Forward Euler Method combined with the Method of Steps.



Next we consider the delay differential equation¹

$$y'(t) = \frac{cy(t - \tau)}{1 + y(t - \tau)^m} - \lambda y(t) \quad (7.71)$$

$$y(t) = 1, -\tau < t < 0 \quad (7.72)$$

with $c = 0.11$, $\lambda = 0.21$, and $m = 10$. The only difference in the implementation is the definition of the function `f[t]`:

```
f[t_] := Module[{
  λ = 0.11, c = 0.21, m = 10, i, ydelay, ynow, v},
  i = index[t - tau, tau, n];
  ydelay = data[[i]];
  ynow = Last[data];
  v = c*ydelay/(1 + ydelaym) - λ*ynow;
  Return[v];
];
```

This function makes the assumption that whenever `f[t]` is called the current value of $y(t)$ is stored the last element of the list array `data`. In the illustrations shown in figure 7.3 we have also modified the call to `ListPlot` to also display the value of `tau`,

¹This equation is derived from a model of respiration, M.C.Mackey and L.Glass (1977) "Oscillation and chaos in physiological control systems," *Science*, **197**:287-289.

```
ListPlot[results, PlotJoined -> True, PlotLabel -> ‘‘tau’’ <>
ToString[tau], Frame->True, AspectRatio-> 0.3]
```

Figure 7.3 illustrates the results when this DDE is solved using $\tau = 1, 4, 7, 15, 20$, showing a constant steady state, damped oscillations, stable regular oscillations, irregular oscillations, and chaotic oscillations. The point to be gained from this is that even with constant initial conditions ($g(t) = 1$) the behavior of the solution can change radically with different size delays.

7.4 Runge-Kutta Methods for Delay Differential Equations

Delay differential equations can be solved using Runge-Kutta methods. Suppose that we have an equation with a single delay that we wish to solve,

$$y'(t) = f(t, y(t), y(t - \tau)) \quad (7.73)$$

$$y(t) = g(t), \quad t_0 - \tau < t < t_0 \quad (7.74)$$

It is convenient to use a fixed step size such that

$$\tau = kh \quad (7.75)$$

for some integer k . Then the corresponding Runge-Kutta method is

$$K_i^{(n)} = y_n + h \sum_{j=1}^{\nu} a_{ij} f(t_j + c_j h, K_j^{(n)}, \gamma_j^{(n)}) \quad (7.76)$$

$$\gamma_j^{(n)} = \begin{cases} \gamma_j^{(n)} = g(t_n + c_j h - \tau) & \text{if } n < \nu \\ K_j^{(n-\nu)} & \text{if } n \geq \nu \end{cases} \quad (7.77)$$

$$y_n = y_{n-1} + \sum_{j=1}^{\nu} b_j f(t_{n-1} + c_j h, K_j^{(n-1)}, \gamma_j^{(n-1)}) \quad (7.78)$$

This is equivalent to solving a different set of ordinary differential equations on each interval: on the interval $[t_0, t_0 + \tau]$, solve

$$u'(t) = f(t, u(t), g(t - \tau)) \quad (7.79)$$

On the interval $[t_0 + \tau, t_0 + 2\tau]$, solve

$$u'(t) = f(t, u(t), v(t)) \quad (7.80)$$

$$v'(t) = f(t - \tau, v(t), g(t - 2\tau)) \quad (7.81)$$

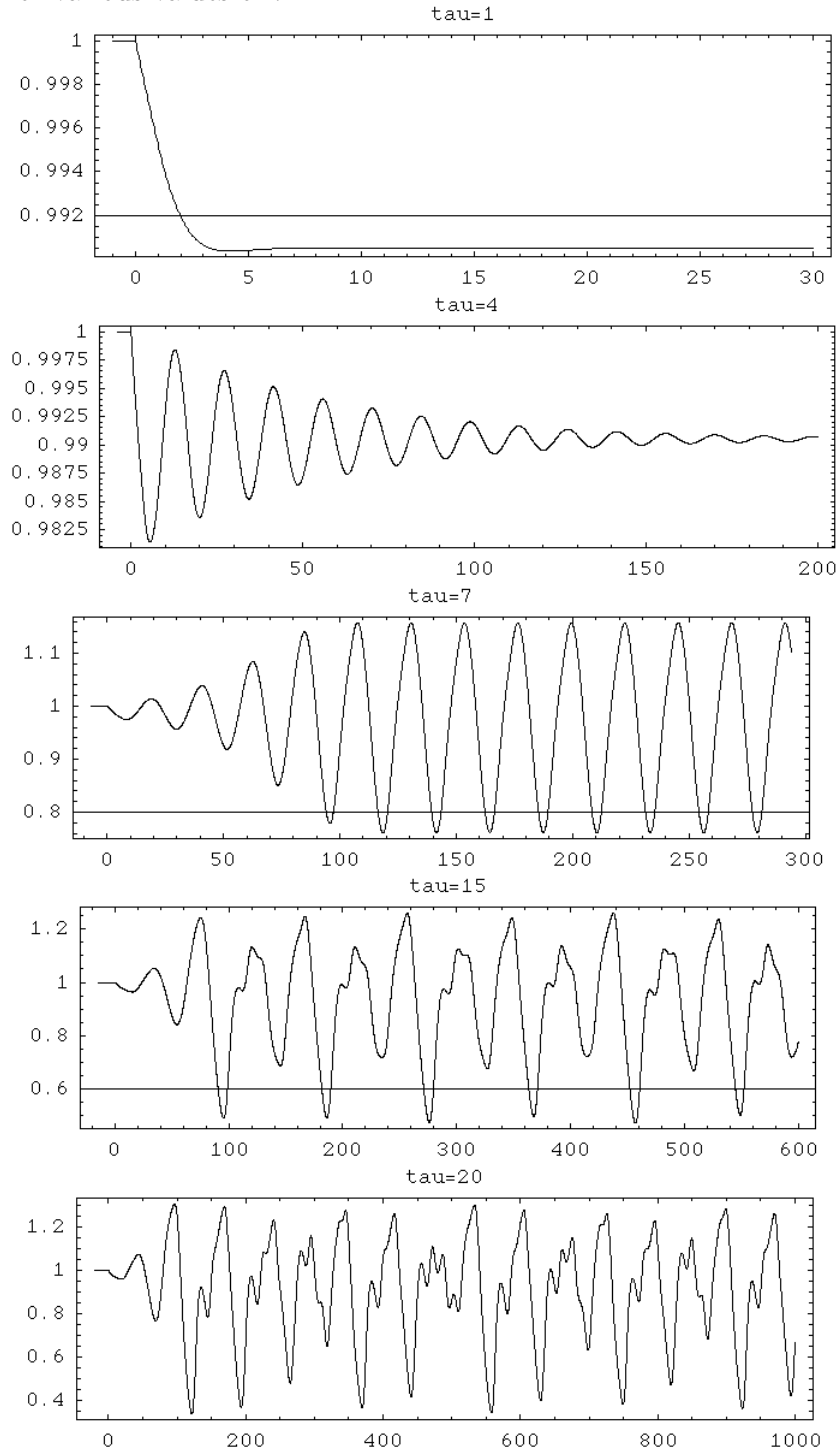
On the interval $[t_0 + 2\tau, t_0 + 3\tau]$, solve

$$u'(t) = f(t, u(t), v(t)) \quad (7.82)$$

$$v'(t) = f(t - \tau, v(t), w(t)) \quad (7.83)$$

$$w'(t) = f(t - 2\tau, w(t), g(t - 3\tau)) \quad (7.84)$$

Figure 7.3: Solution of $y' = cy(t - \tau)/(1 + [y(t - \tau)]^{10}) - \lambda y(t)$ with $c = 0.21$ and $b = 0.11$ for various values of τ .



and so forth.

This method will converge for DDE's with the same order as the corresponding RK method for IVPs does (i.e., the RK method with the same coefficients). To determine the stability of the method for solving delay differential equations we introduce the following generalization of the test equation to include delays,

$$y'(t) = \lambda y(t) + \mu y(t - 1) \quad (7.85)$$

for some constants λ and μ . The substitution $y(t) = e^{\gamma t}$ yields the "characteristic equation,"

$$\gamma - \lambda - \mu e^{-\gamma} = 0 \quad (7.86)$$

Theorem 7.3. *The solutions of the test equation 7.85 remain stable as $t \rightarrow \infty$ if all the roots of the characteristic equation remain in the left-half of the complex plane.*

The region of stability is very difficult to calculate and we will not consider it further here.²

7.5 Continuous Methods

One of the restrictions of the method of steps is that we must always have an integral number of mesh-points within each delay interval; otherwise there will be an error due to the overlap between the mesh and the delay. When the delay is time-dependent, or there are multiple delays, the problem becomes more complicated. For constant delays, the method of steps is generally sufficient and it is not necessary to resort to continuous methods. Instead, one can just adjust the step size to match the interval.

Any numerical method can be extended to a continuous form by replacing the coefficients with functions that depend on a parameter so long as we enforce appropriate continuity conditions on the mesh. This allows us to evaluate the method at points in-between mesh points. Clearly there can be many possible extensions of any given method; ideally we would like an extension that matches the desired solution as well as possible. This implies that we need some kind of first-guess to a solution before we can even compute a continuous extension, and need to do some sort of prediction-correction iteration before we can proceed from one step to the next.

An additional problem is the propagation of discontinuity. For a method of order p to work it is necessary to include among the mesh all discontinuities of

²See Hayes, N.D. "Roots of the Transcendental Equation Associate with a Certain Difference-Differential Equation" *Journal of the London Mathematical Society*, 25:226-232 (1950); C.A.H. Paul and C.T.H. Baker, "Stability Boundaries Revisited - Runge-Kutta Methods for Delay Differential Equations," *Numerical Analysis Report No. 205*, University of Manchester, Dept. of Mathematics (1991); R. Bellman and K.L. Cooke "Differential-Difference Equations," *Academic Press* (1963).

$y, y', y'', \dots, y^{(p+1)}$; however, we may not even know where all the discontinuities are. For a system with time-dependent delay,

$$y' = f(t, y(t), y(t - \tau(t, y))) \quad (7.87)$$

$$y(t) = g(t), t \leq t_0 \quad (7.88)$$

Discontinuities can arise because

- The initial function $g(t)$ may have discontinuities.
- The initial function $g(t)$ may not link smoothly to the solution at $t = t_0$. This will lead to a discontinuity in y'' at the next step, y''' at the following step, and so on.
- Discontinuities in the derivatives of f, τ , and g .

In particular, one can construct pathological examples where the distance between discontinuities gets smaller and smaller, approaching zero as $t \rightarrow \xi$ for some number ξ . This vanishing delay can lead one to require infinitely small grid spacing, which is not possible. The subject of discontinuity is subtle and non-trivial. We will not consider the details here (see [3] for details) except to point out that when the delay is not constant they can arise in unexpected places. When the delay is constant, the functions f and g are smooth and the solution matches g at $t = 0$ then the only discontinuities correspond to multiples of the delay.

Definition 7.3. *Let*

$$y_{n+1} = \sum_{i=1}^k \alpha_{n,i} y_{n+1-i} + h_{n+1} \phi(y_n, \dots, y_{n-k+1}) \quad (7.89)$$

be a general multistep method for the initial value problem $y' = f(t, y), y(t_0) = y_0$. Then a **continuous extension of the method** is a piecewise polynomial interpolant $\eta(t)$

$$\eta(t_n + \theta h_{n+1}) = \sum_{i=1}^{j_n+i_n+1} \beta_{n,i}(\theta) y_{n+j_n-i+1} + h_{n+1} \phi(y_{n+j_n}, \dots, y_{n-i_n}) \quad (7.90)$$

$$\eta(t_n) = y_n \quad (7.91)$$

$$\eta(t_{n+1}) = y_{n+1} \quad (7.92)$$

that is computed on an interval $I = [t_{n-i_n}, t_{n+j_n+1}]$, where $[t_n, t_{n+1}] \subset I$.

A continuous Runge-Kutta method, for example, is

$$y(t_n + \theta h) = y_n + h \sum_{i=1}^2 b_i(\theta) f(t_n, Y_{ni}) \quad (7.93)$$

$$Y_{ni} = y_n + h \sum_{j=1}^2 a_{ij} Y_{nj} \quad (7.94)$$

The order conditions for the continuous extensions of the Runge-Kutta 4-stage method, for example, become (compare with 5.128 through 5.131). For first order:

$$\sum_i b_i(\theta) = \theta \quad (7.95)$$

For second order,

$$\sum_i b_i(\theta)c_i = \frac{1}{2}\theta^2 \quad (7.96)$$

For third order,

$$\sum_i b_i(\theta)c_i^2 = \frac{1}{3}\theta^3 \quad (7.97)$$

$$\sum_{i,j} b_i(\theta)a_{ij}c_j = \frac{1}{6}\theta^3 \quad (7.98)$$

For fourth order, the following conditions must also be satisfied:

$$\sum_i b_i(\theta)c_i^3 = \frac{1}{4}\theta^4 \quad (7.99)$$

$$\sum_{i,j} b_i(\theta)c_i a_{ij}c_j = \frac{1}{8}\theta^4 \quad (7.100)$$

$$\sum_{i,j} b_i(\theta)a_{ij}c_j^2 = \frac{1}{12}\theta^4 \quad (7.101)$$

$$\sum_{i,j,k} b_i(\theta)a_{ij}a_{jk}c_k = \frac{1}{24}\theta^4 \quad (7.102)$$

A second order interpolant for the traditional Runge-Kutta 4-stage method is given by

$$b_1(\theta) = \left(-\frac{1}{2}\theta + \frac{2}{3}\right)\theta \quad (7.103)$$

$$b_2(\theta) = \frac{1}{3}\theta \quad (7.104)$$

$$b_3(\theta) = \frac{1}{3}\theta \quad (7.105)$$

$$b_4(\theta) = \left(\frac{1}{2}\theta - \frac{1}{3}\right)\theta \quad (7.106)$$

A third order interpolant is given by

$$b_1(\theta) = \left(\frac{2}{3}\theta^2 - \frac{3}{2}\theta + 1\right)\theta \quad (7.107)$$

$$b_2(\theta) = \left(-\frac{2}{3}\theta + 1\right)\theta^2 \quad (7.108)$$

$$b_3(\theta) = \left(-\frac{2}{3}\theta + 1\right) \theta^2 \quad (7.109)$$

$$b_4(\theta) = \left(\frac{2}{3}\theta - \frac{1}{2}\right) \theta^2 \quad (7.110)$$

Details on implementing these methods are given in the reports by Paul and Baker.³

Algorithm 7.2. Continuous Extension to Method Steps To solve the delay differential equation

$$y'(t) = f(t, y(t), y(t - \tau(t, y))) \quad (7.111)$$

$$y(t) = g(t), t_0 - \tau < t < t_0 \quad (7.112)$$

on (t_0, t_f)

1. Locate all discontinuity points $\xi_1, \xi_2, \dots, \xi_m < t_f$.
2. Set $\xi_0 = t_0, \xi_{m+1} = t_f$
3. Solve

$$y' = f(t, y(t), g(t - \tau(t))) \quad (7.113)$$

$$y(x_{i_0}) = g(\xi_0) \quad (7.114)$$

on (ξ_0, ξ_1) using any ODE solver.

4. For $i = 1, \dots, m$
 - (a) Compute the continuous extension $\eta(t)$ on (ξ_{i-1}, ξ_i)
 - (b) Solve the equation

$$y' = f(t, y(t), \eta(t - \tau(t))) \quad (7.115)$$

$$y(x_{i_0}) = \eta(\xi_0) \quad (7.116)$$

on (ξ_i, ξ_{i+1})

³C.A.H. Paul & C.T.H. Baker, "Explicit Runge-Kutta Methods for the Numerical Solution of Singular Delay Differential Equations," Numerical Analysis Report 212, April 1992, University of Manchester, Dept. of Mathematics, <http://citeseer.ist.psu.edu/37968.html>

Chapter 8

Boundary Value Problems

8.1 Shooting

In this section the **method of shooting** will be illustrated with an example. Suppose we are given the boundary value problem

$$y'' + \frac{1}{t}y'(t) - \frac{1}{t^2}y = 0 \quad (8.1)$$

$$y(1) = y(2) = 1 \quad (8.2)$$

It is easily verified that an exact solution is

$$y(t) = \frac{2 + t^2}{3t} \quad (8.3)$$

Our goal is to find this solution numerically. We will do this by converting the the problem into an initial value problem. Let $u = y, v = y'$. Then

$$u' = v, \quad u(1) = 1 \quad (8.4)$$

$$v' = \frac{1}{t^2}u - \frac{1}{t}v, \quad v(1) = c \quad (8.5)$$

where c is the unknown initial condition for $v(1) = y'(1)$. We can estimate a first guess at c using 8.7 and the given boundary data with a forward approximation to the derivative:

$$c^{(0)} = v(1)^{(0)} = u'(1) \approx \frac{u(2) - u(1)}{2 - 1} = 0 \quad (8.6)$$

The estimate is very rough and could be quite far from the actual value. Next we will use our Forward Euler method (we could replace this with any other IVP method, but we will stick to Forward Euler because it is the simplest, and works quite well for illustrating the problem) to solve the first IVP approximation to our BVP:

$$u' = v, \quad u(1) = 1 \quad (8.7)$$

$$v' = \frac{1}{t^2}u - \frac{1}{t}v, \quad v(1)^{(0)} = c^{(0)} = 0 \quad (8.8)$$

We will modify the Euler program slightly to take the number of mesh points and the two boundary points – rather than the step size – as parameter. This is because we want to force the problem to stop at the second boundary point. The following code will return a single Euler solution for the variable u (we don't really care about v since it was not in the original problem).

```
EulerIVP[f_, ya_, a_, b_, n_] :=
Module[{ t, y, h, solution},
  h = (b - a)/n;
  solution = {{a, ya[[1]]}};
  y = ya;
  For[t = a, t < b, t += h,
    y = y + h*f[t, y];
    AppendTo[solution, {t + h, y[[1]]}];
  ];
  Return[solution];
]
```

Suppose we want to estimate the solution using a mesh with 100 intervals. Then our first guess would be obtained with

```
f[t_, u_, v_] := {v, u/(t*t) - v/t}
EulerIVP[f, 1, 0, 1, 2., 100]
```

which will return

```
{{1, 1}, {1.01, 1}, {1.02, 1.0001}, ..., {2., 1.25041}}
```

where the intervening values have been omitted. The very last value is what we are interested in: our first guess gives $u(2)^{(0)} = 1.25041$. Our goal is to iterate on this value: if we have two guesses at c and the corresponding two guess at $u(2)$, we can use linear interpolation to make a third guess:

$$m^{(i)} = \frac{u(b)^{(i)} - u(b)^{(i-1)}}{c^{(i)} - c^{(i-1)}} \quad (8.9)$$

$$c^{(i+1)} = c^{(i)} + \frac{u(b) - u(b)^{(i)}}{m} \quad (8.10)$$

The problem is that we don't have a good second estimate for c , so we pick one arbitrarily, namely

$$c^{(1)} = c^{(0)} + 1 = 1 \quad (8.11)$$

Then

```
EulerIVP[f, 1, 1, 1, 2., 100]
```

returns

```
{{1, 1}, {1.01, 1.01}, {1.02, 1.02}, ..., {2., 2}}
```

Thus using $u(b)^{(0)} = 1.25041$, $u(b)^{(1)} = 2$, with our values for $c^{(0)}$ and $c^{(1)}$ we calculate

$$m^{(1)} = \frac{2 - 1.25041}{1 - 0} = 0.74959 \quad (8.12)$$

$$c^{(2)} = 1 + \frac{1 - 2}{0.74959} = -0.33406 \quad (8.13)$$

Next

```
EulerIVP[f, 1, -0.33406, 1, 2., 100]
```

gives

```
{{1, 1}, {1.01, 0.9966594}, {1.02, 0.993452206}, ...,
 {2., 1.0000059174753426}}
```

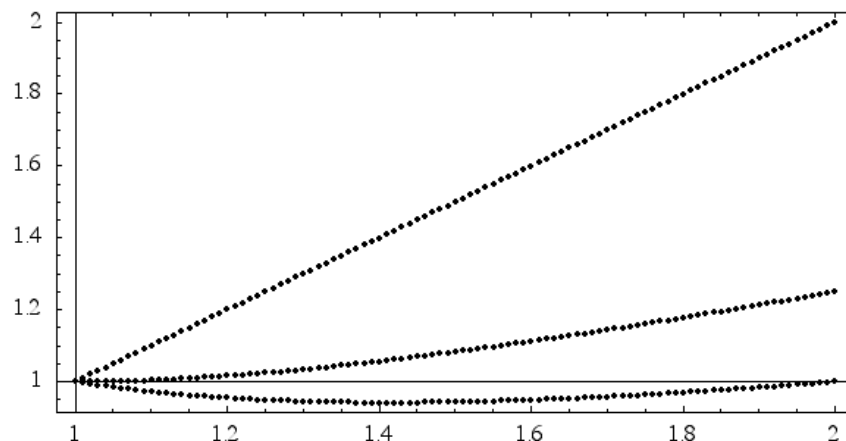
which shows convergence at the boundary value to better than one part in 10^5 . The three estimates are illustrated in figure 8.1.

Now we put it all together and automate the shooting process.

```
Shoot[f_, ua_, ub_, a_, b_, n_, MaxIterations_:5, tolerance_:10-6]
:= Module[{c, ya, ubObserved, ubObservedOld, cold, m},
c = (ub - ua)/(b - a); (* first guess *)
ya = {ua, c};
s = EulerIVP[f, ya, a, b, n]; (* first shot *)
ubObserved = Last[Last[s]];
cold = c;
c = c + 1; (* second guess *)
Do[
ubObservedOld = ubObserved;
ya = ua, c;
s = EulerIVP[f, ya, a, b, n]; (* next shot *)
ubObserved = Last[Last[s]];
If[Abs[ubObserved - ubObservedOld] < tolerance, Break[]];
m = (ubObserved - ubObservedOld)/(c - cold) (* slope *)
cold = c;
c = cold + (ub - ubObserved)/m; (* next guess at c *) ,
{MaxIterations}
];
Return[s];
];
```

The problem with this implementation is that it can miss solutions, if the BVP has multiple solutions. To fix this we will have to replace the linear slope correction with a full Newton root finder. At this point it is not yet clear what we are finding the root of, so we will need to develop the theory somewhat first before we can return to the shooting algorithm.

Figure 8.1: Three estimates to the solution using shooting, top to bottom curves.



8.2 Basic Theory of Boundary Value Problems

In vector form the basic **two point boundary value problem** is given by

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \quad (8.14)$$

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0} \quad (8.15)$$

for some function (generally nonlinear) $\mathbf{g}(\mathbf{u}, \mathbf{v})$. When the boundary condition is linear then for some given data \mathbf{d}

$$B_a \mathbf{y}(a) + B_b \mathbf{y}(b) = \mathbf{d} \quad (8.16)$$

for some square matrices B_a and B_b . In general, however, we will define

$$B_a = \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \quad (8.17)$$

$$B_b = \frac{\partial \mathbf{g}}{\partial \mathbf{v}} \quad (8.18)$$

for both linear and nonlinear boundary conditions.

Example 8.1. For the boundary value problem considered in section 8.1

$$\frac{d}{dt} \begin{pmatrix} u \\ v \end{pmatrix} = \mathbf{y}' = \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} u & v \\ \frac{u}{t^2} & -\frac{v}{t} \end{pmatrix} \quad (8.19)$$

we have linear boundary conditions

$$B_a = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}; \quad B_b = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}; \quad \mathbf{d} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad (8.20)$$

because

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} u(a) \\ v(a) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u(b) \\ v(b) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \square \quad (8.21)$$

We assume that f is bounded and Lipschitz; hence there is a unique solution to the initial value problem

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}) \quad (8.22)$$

$$\mathbf{y}(a) = \mathbf{c} \quad (8.23)$$

for some unknown vector \mathbf{c} , which we will denote by $\mathbf{y}(t, y(t); \mathbf{c})$. Substituting this back into equation 8.15 gives

$$\mathbf{g}(\mathbf{c}, \mathbf{y}(b, y(b); \mathbf{c})) = \mathbf{0} \quad (8.24)$$

This is a nonlinear equation in n unknowns (n is the dimension of y), which, in general, may have any number (or no) solutions and hence is a very difficult problem to solve.

We will limit our study to linear differential equations, which have the form

$$\mathbf{y}'(t) = A(t)\mathbf{y}(t) + \mathbf{q}(t) \quad (8.25)$$

for some square matrix A and some vector \mathbf{q} . For example, the problem studied in the previous section can be written

$$\frac{d}{dt} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1/t^2 & -1/t \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \quad (8.26)$$

Note that the matrix does not have to be constant nor does it have to be linear in t for the system to be linear. It is only linearity in the dependent variables (e.g., u and v) that matters for the problem to be considered linear.

A **fundamental matrix** $Y(t)$ of the differential equation is the square matrix that satisfies

$$Y'(t) = A(t)Y(t) \quad (8.27)$$

The columns of the fundamental matrix are the linearly independent solutions to the homogeneous equation. In terms of the fundamental matrix, the general solution of the differential equation 8.25 is

$$\mathbf{y}(t) = Y(t) \left[\mathbf{c} + \int_a^t Y^{-1}(s)\mathbf{q}(s)ds \right] \quad (8.28)$$

where c is determined by the boundary conditions.

Example 8.2. Find the fundamental matrix and general solution of the problem studied in the previous section.

Solution. The fundamental matrix is

$$Y(t) = \begin{pmatrix} \frac{t^2 + 1}{t^2 - 1} & \frac{t^2 - 1}{2t^2} \\ \frac{2t}{t^2 - 1} & \frac{2t}{t^2 + 1} \end{pmatrix} \quad (8.29)$$

and hence, since $q = 0$,

$$y(t) = c_1 \left(\frac{t^2 + 1}{\frac{t^2 - 1}{2t^2}} \right) + c_2 \left(\frac{t^2 - 1}{\frac{2t}{t^2 + 1}} \right) \quad (8.30)$$

where the parameter vector $\mathbf{c} = (c_1 \ c_2)^T$ is determined by the boundary conditions, equation 8.16. \square

Substituting 8.16 into 8.28,

$$\mathbf{y}(a) = Y(a)\mathbf{c} \quad (8.31)$$

$$\mathbf{y}(b) = Y(b)\mathbf{c} - Y(b) \int_a^b Y^{-1}(s)\mathbf{q}(s)ds \quad (8.32)$$

$$\mathbf{d} = B_a\mathbf{y}(a) + B_b\mathbf{y}(b) \quad (8.33)$$

$$= (B_aY(a) + B_bY(b))\mathbf{c} - B_bY(b) \int_a^b Y^{-1}(s)\mathbf{q}(s)ds \quad (8.34)$$

If we define the matrix

$$Q = B_aY(a) + B_bY(b) \quad (8.35)$$

then

$$\mathbf{c} = Q^{-1} \left[\mathbf{d} - B_bY(b) \int_a^b Y^{-1}(s)\mathbf{q}(s)ds \right] \quad (8.36)$$

Example 8.3. Solve the boundary value problem considered in section 8.1 using this formalism.

Solution. We have $a = 1$ and $b = 2$, so

$$Y(a) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \quad Y(b) = \begin{pmatrix} 5/4 & 3/4 \\ 3/8 & 5/8 \end{pmatrix} \quad (8.37)$$

and hence (see example 8.1),

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 5/4 & 3/4 \\ 3/8 & 5/8 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 5/4 & 3/4 \end{pmatrix} \quad (8.38)$$

Hence

$$Q^{-1} = \begin{pmatrix} 1 & 0 \\ -5/3 & 4/3 \end{pmatrix} \quad (8.39)$$

and therefore (since $\mathbf{q} = 0$ for this problem)

$$\mathbf{c} = Q^{-1}\mathbf{d} = \begin{pmatrix} 1 & 0 \\ -5/3 & 4/3 \end{pmatrix} \begin{pmatrix} 1 \\ -1/3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1/3 \end{pmatrix} \quad (8.40)$$

The solution of the differential equation is then (we only state the first component since that is all we are really interested in, as it originated in a 2nd order equation):

$$u(t) = \frac{t^2 + 1}{2t} - \frac{1}{3} \frac{t^2 - 1}{2t} = \frac{t^2 + 2}{3t} \quad \square \quad (8.41)$$

Theorem 8.1. *If the matrix $Q = B_a Y(a) + B_b Y(b)$ is invertible then the solution to*

$$\mathbf{y}' = A(t)\mathbf{y} + \mathbf{q}(t) \quad (8.42)$$

$$B_a \mathbf{y}(a) + B_b \mathbf{y}(b) = \mathbf{d} \quad (8.43)$$

exists and is given by

$$\mathbf{y}(t) = Y(t) \left[\mathbf{c} + \int_a^t Y^{-1}(s) \mathbf{q}(s) ds \right] \quad (8.44)$$

$$\mathbf{c} = Q^{-1} \left[\mathbf{d} - B_b Y(b) \int_a^b Y^{-1}(s) \mathbf{q}(s) ds \right] \quad (8.45)$$

If we define the matrix $\Phi(t) = Y(t)Q^{-1}$ then

$$\mathbf{y}(t) = \Phi(t) + \int_a^b G(t, s) \mathbf{q}(s) ds \quad (8.46)$$

where $G(t, s)$ is the **Green's function** of the differential equation

$$G(t, s) = \begin{cases} \Phi(t) B_a \phi(a) \Phi^{-1}(s), & s \leq t \\ -\Phi(t) B_b \Phi(b) \Phi^{-1}(s), & s > t \end{cases} \quad (8.47)$$

8.3 Shooting Revisited

We return to solving the general boundary value problem

$$\mathbf{y}' = \mathbf{y}(t, \mathbf{y}) \quad (8.48)$$

$$\mathbf{g}(\mathbf{y}(a; \mathbf{c}), \mathbf{y}(b; \mathbf{c})) = \mathbf{0} \quad (8.49)$$

where we have used the notation $\mathbf{y}(t, \mathbf{c})$ to indicate the solution at t that satisfies the initial condition $\mathbf{y}(a; \mathbf{c}) = \mathbf{c}$. Then we define the function

$$\mathbf{H}(\mathbf{c}) = \mathbf{g}(\mathbf{c}, \mathbf{y}(b, \mathbf{c})) = \mathbf{0} \quad (8.50)$$

To solve this equation for \mathbf{c} we can iterate using Newton's method

$$\mathbf{c}^{i+1} = \mathbf{c}^i - \left(\frac{\partial \mathbf{H}}{\partial \mathbf{c}} \right)^{-1} \mathbf{H}(\mathbf{c}^i) \quad (8.51)$$

Using the notation of 8.17, where $\mathbf{g} = \mathbf{g}(\mathbf{u}, \mathbf{v})$, by the chain rule

$$\frac{\partial \mathbf{H}}{\partial \mathbf{c}} = \frac{\partial \mathbf{H}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{c}} + \frac{\partial \mathbf{H}}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{c}} \quad (8.52)$$

$$= B_a Y(a) + B_b Y(b) \quad (8.53)$$

$$= Q \quad (8.54)$$

where Y is the fundamental matrix. This makes the Newton iteration formul

$$\mathbf{c}^{i+1} = \mathbf{c}^i - Q^{-1}\mathbf{H}(\mathbf{c}^i) \quad (8.55)$$

Since inverting a matrix is expensive, it is usually more reasonable to solve the linear system

$$Q\Delta^{i+1} = \mathbf{H}(\mathbf{c}^i) \quad (8.56)$$

for Δ and then perform the update

$$\mathbf{c}^{i+1} = \mathbf{c}^i + \Delta^{i+1} \quad (8.57)$$

Algorithm 8.1. Shooting

To solve the boundary value problem

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$$

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = 0$$

1. Input: \mathbf{f} , \mathbf{f}_y , $\mathbf{g}(\mathbf{u}, \mathbf{v})$, \mathbf{g}_u , \mathbf{g}_v , $\mathbf{c}^{(0)}$, and a convergence tolerance;
2. Repeat the following until the desired tolerance in \mathbf{c} is reached
 - (a) Solve the initial value problem $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ with the initial value at a with $\mathbf{y}(\mathbf{c}^i)$ obtaining a mesh of solution values $\mathbf{y}_0^{(i)}, \dots, \mathbf{y}_N^{(i)}$.
 - (b) Form the vector $\mathbf{H} = \mathbf{g}(\mathbf{c}^{(i)}, \mathbf{y}_N^{(i)})$
 - (c) Integrate the fundamental matrix Y_n on the same mesh using $A_j = \mathbf{f}_y$.
 - (d) Calculate Q and solve the linear system $Q\Delta = \mathbf{H}(\mathbf{c}^{(i)})$.
 - (e) Update $\mathbf{c}^{(i)} = \mathbf{c}^{(i-1)} + \Delta$.
3. Solve the initial value problem for $\mathbf{y}(0) = \mathbf{c}$, where \mathbf{c} is the final iteration in the preceding step.

Shooting has a number of problems, among them

- Roundoff error;
- Propagation error – for linear systems $\approx e^{L(b-a)}$ where $L = \max \|A\|$;
- It assumes that the solution to the IVP exists globally, which might not be true.

and consequently this “simple” shooting algorithm is rarely used.

One way to improve shooting is called **multiple shooting**. In this technique simple shooting is applied separately on each interval of the mesh. Let \mathbf{y}_n be the *exact* solution of the initial value problem

$$\mathbf{y}'_n = \mathbf{f}(t, \mathbf{y}_n) \quad (8.58)$$

$$\mathbf{y}_n(t_{n-1}) = \mathbf{c}_{n-1} \quad (8.59)$$

on the interval $[t_{n-1}, t_n]$. Assuming that each of these exact solutions is known then the *exact* solution of

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \quad (8.60)$$

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0} \quad (8.61)$$

on $[a, b]$ satisfies

$$\mathbf{y}(t) = \begin{cases} \mathbf{y}_1(t, \mathbf{c}_0) & \text{on } [t_0, t_1] \\ \mathbf{y}_2(t, \mathbf{c}_1) & \text{on } [t_1, t_2] \\ \vdots & \\ \mathbf{y}_N(t, \mathbf{c}_{N-1}) & \text{on } [t_{N-1}, t_N] \end{cases} \quad (8.62)$$

if the following **patching conditions**, which ensure continuity on the entire interval $[a, b]$, are met:

$$\begin{cases} \mathbf{y}_1(t_1; \mathbf{c}_0) = \mathbf{c}_1 \\ \mathbf{y}_2(t_2; \mathbf{c}_1) = \mathbf{c}_2 \\ \vdots \\ \mathbf{y}_{N-1}(t_{N-1}; \mathbf{c}_{N-2}) = \mathbf{c}_{N-1} \\ \mathbf{g}(\mathbf{c}_0, \mathbf{y}_N(b, \mathbf{c}_{N-1})) = \mathbf{0} \end{cases} \quad (8.63)$$

The patching conditions give a vector \mathbf{c} of Nm coefficients, where

$$\mathbf{c} = (\mathbf{c}_0^T \quad \mathbf{c}_1^T \quad \cdots \quad \mathbf{c}_{N-1}^T)^T \quad (8.64)$$

which satisfy the condition 8.63. As before, we will write the condition 8.63 as

$$\mathbf{H}(\mathbf{c}) = \mathbf{0}, \quad (8.65)$$

which is a nonlinear equation that needs to be solved for \mathbf{c} .

For the linear BVP,

$$\mathbf{y}' = A(t)\mathbf{y} + \mathbf{q}(t) \quad (8.66)$$

$$B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{d} \quad (8.67)$$

for some initial data vector \mathbf{d} . If $Y_n(t)$ is a fundamental matrix ¹, i.e.,

$$Y_n' = A(t)Y_n \quad (8.68)$$

$$Y_n(t_{n-1}) = I \quad (8.69)$$

(and in particular, $Y_1 = Y$) then

$$\mathbf{y}_n(t; \mathbf{c}_{n-1}) = Y_n(t)\mathbf{c}_{n-1} + \mathbf{v}_n(t) \quad (8.70)$$

where \mathbf{v}_n is a particular solution satisfying

$$\mathbf{v}_n' = A(t)\mathbf{v}_n + \mathbf{q}(t) \quad (8.71)$$

The patching conditions then become

$$\begin{cases} \mathbf{c}_1 - Y_1(t_1)\mathbf{c}_0 = \mathbf{v}_1(t_1) \\ \mathbf{c}_2 - Y_2(t_2)\mathbf{c}_1 = \mathbf{v}_2(t_2) \\ \vdots \\ \mathbf{c}_{N-1} - Y_{N-1}(t_{N-1})\mathbf{c}_{N-2} = \mathbf{v}_{N-1}(t_{N-1}) \\ B_a Y(a)\mathbf{c}_0 + B_b Y_N(b)\mathbf{c}_{N-1} = \mathbf{d} - B_b \mathbf{v}_N(b) \end{cases} \quad (8.72)$$

which we can write in matrix form as $\mathbf{A}\mathbf{c} = \mathbf{r}$,

$$\begin{pmatrix} -Y_1(t_1) & I & & & & \\ & -Y_2(t_2) & I & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & -Y_{N-1}(t_{N-1}) & I \\ B_a & & & & & B_b Y_N(b) \end{pmatrix} \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{N-2} \\ \mathbf{c}_{N-1} \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1(t_1) \\ \mathbf{v}_2(t_2) \\ \vdots \\ \mathbf{v}_{N-1}(t_{N-1}) \\ \mathbf{d} - B_b \mathbf{v}_N(b) \end{pmatrix} \quad (8.73)$$

While this matrix is very large $mN \times mN$ the system can still be inverted in linear time because the matrix is sparse (mostly zeroes).

8.4 One-step Methods

Initial value problems are inherently “directional,” in the sense that some points occur “before” or “after” other values. The value at t_n needs to be known before we can calculate t_{n+1} , and so forth, and so we say that y_n occurs before y_{n+1} . This is natural to our intuitive feeling of the dependent variable t as time. Boundary value problems, however, do not have a “before” or an “after” – all points “occur” at once. This is again in line with our intuition, because the boundary typically represents a spatial rather than a temporal domain (although boundary value problems in a

¹The columns of the fundamental matrix are the linearly independent solutions to the differential equation, so the matrix at the endpoints is a constant linearly independent matrix. By a suitable orthogonalization process the solutions can be made orthogonal and the resulting fundamental matrix will be the identity at the end points.

temporal domain can also be solved). Even though we solve an IVP when we apply the shooting algorithm, we get the entire solution “at once,” so to speak, because we are iterating on the final value of the IVP solution, and we do not know whether we will accept or reject the solution until we have *all* of the mesh points.

We consider the two-point boundary value problem

$$y' = f(t, y) \quad (8.74)$$

$$g(y(a), y(b)) = 0 \quad (8.75)$$

where these equations may be either scalar or vector. A numerical solution will be a set of values

$$y_0, y_1, \dots, y_N \quad (8.76)$$

on a mesh

$$a = t_0 < t_1 < \dots < t_N = b \quad (8.77)$$

We seek a finite-difference method to solve this problem. Since there is no before and no after, there is no advantage to using explicit methods over implicit methods, nor is there any advantage to using multistep methods because they are biased in one direction. It makes the most sense to use a symmetric method, one that treats points in the “forward direction” and “reverse direction” in the same manner, such as the midpoint and trapezoidal methods.

The **midpoint method** is given by

$$y_n - y_{n-1} = hf \left(t_{n-1/2}, \frac{1}{2}(y_n + y_{n-1}) \right) \quad (8.78)$$

and the **trapezoidal method** by

$$y_n - y_{n-1} = \frac{h}{2} [f(t_n, y_n) + f(t_{n-1}, y_{n-1})] \quad (8.79)$$

along with the boundary condition

$$g(y_0, y_N) = 0 \quad (8.80)$$

Both methods are implicit. If the system has dimension m ($m = 1$ for a scalar equation) then this gives us a system of $m(N + 1)$ nonlinear algebraic equations for the function values at the mesh points.

Given an initial guess y_n^0 at a mesh point we can define a sequence of iterates y^0, y^1, \dots via **quasi-linearization** as

$$(y^{k+1}(t))' = f(t, y^k(t)) + f_y(t, y^k(t))(y^{k+1}(t) - y^k(t)) \quad (8.81)$$

$$= f(t, y^k(t)) + A(t)(y^{k+1}(t) - y^k(t)) \quad (8.82)$$

$$= A(t)y^{k+1}(t) + q(t) \quad (8.83)$$

where $A(t) = f_y(t, y^k(t))$ and

$$q(t) = f(t, y^k(t)) - A(t)y^k(t) \quad (8.84)$$

For the boundary condition we apply the chain rule

$$0 = g + g_u(y^k(a), y^k(b))\Delta y(a) + g_v(y^k(a), y^k(b))\Delta y(b) \quad (8.85)$$

$$= g + B_a^k(y^{k+1}(a) - y^k(a)) + B_b^k(y^{k+1}(b) - y^k(b)) \quad (8.86)$$

where

$$B_a^k = \frac{\partial g}{\partial u}(y^k(a), y^k(b)); \quad B_b^k = \frac{\partial g}{\partial v}(y^k(a), y^k(b)) \quad (8.87)$$

If we define

$$d = -g(y^k(a), y^k(b)) + B_0 y^k(a) + B_b y^k(b) \quad (8.88)$$

then

$$B_a y^{k+1}(a) + B_b y^{k+1}(b) = d \quad (8.89)$$

Writing $y = y^{k+1}$ gives

$$y' = A(t)y + q(t) \quad (8.90)$$

$$B_a y(a) + B_b y(b) = d \quad (8.91)$$

Example 8.4. Calculate $A(t)$ and $q(t)$ for the linear example 8.1, where

$$y' = \frac{d}{dt} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} v \\ u/t^2 - v/t \end{pmatrix} \quad (8.92)$$

$$B_a = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}; \quad B_b = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad (8.93)$$

Solution. Since

$$f(t, u, v) = \begin{pmatrix} v \\ u/t^2 - v/t \end{pmatrix} \quad (8.94)$$

the Jacobian is given by

$$A(t) = \frac{df}{dy} = \frac{\partial(f_1, f_2)}{\partial(u, v)} = \begin{pmatrix} \partial f_1 / \partial u & \partial f_1 / \partial v \\ \partial f_2 / \partial u & \partial f_2 / \partial v \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1/t^2 & -1/t \end{pmatrix} \quad (8.95)$$

where the f_1 and f_2 denote the first and second vector components of f . Similarly,

$$q(t) = f^k - Ay^k \quad (8.96)$$

$$= \begin{pmatrix} v^k \\ u^k/t^2 - v^k/t \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ 1/t^2 & -1/t \end{pmatrix} \begin{pmatrix} u^k \\ v^k \end{pmatrix} \quad (8.97)$$

$$= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (8.98)$$

as expected. □

For equation 8.90, we have $f(t, y) = A(t)Y + q$, so that the midpoint method gives

$$\frac{y_n - y_{n-1}}{h} = f\left(t_{n-1/2}, \frac{1}{2}(y_n + y_{n-1})\right) \quad (8.99)$$

$$= A(t_{n-1/2})\frac{1}{2}(y_n + y_{n-1}) + q(t_{n-1/2}) \quad (8.100)$$

$$B_a y_0 + B_b y_N = d \quad (8.101)$$

which we can write as the sparse system (where each element of the matrix is really a matrix)

$$\begin{pmatrix} S_1 & R_1 & 0 & \dots & 0 \\ 0 & S_2 & R_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & S_N & R_N \\ B_a & 0 & \dots & 0 & B_b \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ B_a \end{pmatrix} = \begin{pmatrix} q(t_{1/2}) \\ q(t_{3/2}) \\ \vdots \\ q(t_{N-1/2}) \\ d \end{pmatrix} \quad (8.102)$$

where

$$S_n = -\frac{1}{h}I - \frac{1}{2}A(t_{n-1/2}) \quad (8.103)$$

$$R_n = \frac{1}{h}I - \frac{1}{2}A(t_{n-1/2}) \quad (8.104)$$

The quasilinearization algorithm using the midpoint method is summarized in algorithm 9.2. This method can be used for nonlinear problems as well as linear problems because we have quasilinearized the system and then used a linear approximation at each iteration.

Algorithm 8.2. Quasilinearization with the Midpoint Method

To solve the boundary value problem

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$$

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = 0$$

1. Input f , f_y (the Jacobian), $g(u, v)$, the Jacobians g_u , g_v , a mesh definition; an initial guess $y^0(t)$; and a convergence tolerance ϵ

2. For $k = 0, 1, \dots$ until $\|y_n^{k+1} - y_n^k\| < \epsilon$, repeat the following

(a) Calculate each of the S_n and R_n using

$$A(t_{n-1/2}) = f_y \left(t_{n-1/2}, \frac{y_n^k + y_{n-1}^k}{2} \right) \quad (8.105)$$

$$q(t_{n-1/2}) = f \left(t_{n-1/2}, \frac{y_n^k + y_{n-1}^k}{2} \right) - \frac{y_n^k - y_{n-1}^k}{h} \quad (8.106)$$

(b) Calculate

$$B_a = g_u(y_0^k, y_N^k) \quad (8.107)$$

$$B_b = g_v(y_0^k, y_N^k) \quad (8.108)$$

$$d = -g(y_0^k, y_N^k) \quad (8.109)$$

(c) Solve 8.102

Chapter 9

Differential Algebraic Equations

9.1 Concepts

A differential algebraic equation¹ (DAE) is a system of differential equations and algebraic constraints. The most general form is

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0 \quad (9.1)$$

where the Jacobian matrix $\mathbf{F}_{\mathbf{y}'}$ may be singular.² For example, the DAE with $\mathbf{y} = (u, v, w)^T$ described by the system of scalar equations

$$u' = v \quad (9.2)$$

$$v' = w \quad (9.3)$$

$$0 = ue^w + we^u \quad (9.4)$$

has

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \begin{pmatrix} u' - v \\ v' - w \\ ue^w + we^u \end{pmatrix} \quad (9.5)$$

The Jacobian is

$$\frac{\partial \mathbf{F}}{\partial \mathbf{y}'} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (9.6)$$

Under certain conditions any DAE can be reduced to a system of ordinary differential equations by repeatedly differentiating the constraints. The idea is to get an explicit equation for each of the derivatives, of the form $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$. For example, if we differentiate 9.4 we get

$$0 = u'e^w + uw'e^w + w'e^u + wu'e^u \quad (9.7)$$

$$= u'(e^w + we^u) + w'(ue^w + e^u) \quad (9.8)$$

$$w' = -\frac{u'(e^w + we^u)}{ue^w + e^u} = -\frac{v(e^w + we^u)}{ue^w + e^u} \quad (9.9)$$

¹The material in this and the following sections is based on [5].

²In general, throughout this chapter we will be informal with notation and may omit the boldface even though the referenced variable may be a vector or matrix as in the Jacobian $F_{\mathbf{y}'}$

so the original DAE is equivalent to the ODE

$$u' = v \quad (9.10)$$

$$v' = w \quad (9.11)$$

$$w' = -\frac{v(e^w + we^u)}{ue^w + e^u} \quad (9.12)$$

which gives explicit expressions for each of the derivatives in terms of the variables. This type of DAE is said to be of index 1 because a single differentiation was sufficient to convert the equation to a system of ODEs.

Definition 9.1. *The index of a DAE is the minimum number of times all or part of the DAE must be differentiated with respect to t to be able to determine \mathbf{y}' explicitly as a function of t and \mathbf{y} .*

Example 9.1. *Show that the equations for the unit pendulum*

$$y'' = g - T \cos \theta \quad (9.13)$$

$$x'' = -T \sin \theta \quad (9.14)$$

$$1 = x^2 + y^2 \quad (9.15)$$

where T is the unknown tension in the pendulum rope, form an index-3 system.

Solution. We can rewrite this as a first-order system by defining cartesian velocity components $u = x'$ and $v = y'$, and by observing that because of the constraint

$$y = \cos \theta \quad (9.16)$$

$$x = \sin \theta \quad (9.17)$$

Hence the first order DAE becomes

$$x' = u \quad (9.18)$$

$$y' = v \quad (9.19)$$

$$u' = -Tx \quad (9.20)$$

$$v' = g - Ty \quad (9.21)$$

$$0 = x^2 + y^2 - 1 \quad (9.22)$$

where T is an unknown variable. To get a differential equation for T we must differentiate the constraint three times, as follows. First, we obtain

$$0 = xx' + yy' \quad (9.23)$$

$$0 = xu + yv \quad (9.24)$$

Differentiating a second time,

$$0 = xu' + x'u + yv' + y'v \quad (9.25)$$

$$0 = -x^2T + u^2 + yg - Ty^2 + v^2 \quad (9.26)$$

Since $x^2 + y^2 = 1$,

$$T = u^2 + v^2 + yg \quad (9.27)$$

Differentiating a third time gives an equation for T' ,

$$T' = 2uu' + 2vv' + gy' \quad (9.28)$$

$$= -2uTx + 2vg - 2vTy + gv \quad (9.29)$$

$$= -2T(ux + vy) + 3gv \quad (9.30)$$

$$= 3gv \quad (9.31)$$

Since the DAE can be reduced to an explicit ODE in 3 differentiations, it is an index-3 DAE. \square

Equation 9.1 is sometimes referred to as a **fully implicit DAE**. It becomes a **semi-implicit DAE** if we can express the constraint explicitly:

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0 \quad (9.32)$$

$$\mathbf{g}(t, \mathbf{y}, \mathbf{y}') = 0 \quad (9.33)$$

where the Jacobian $\mathbf{F}_{\mathbf{y}'}$ is non-singular. If we can express all the derivatives explicitly then we have a **semi-explicit DAE**:

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}, \mathbf{y}') \quad (9.34)$$

$$0 = g(t, \mathbf{y}, \mathbf{y}') \quad (9.35)$$

A **linear time-varying DAE** can be expressed as

$$A(t)\mathbf{y}'(t) + B(t)\mathbf{y}(t) = f(t) \quad (9.36)$$

where the matrix $A(t)$ is singular for all t (if it were not singular the equation would reduce to linear system of ordinary differential equations). If the matrices are composed of constants we call the system a **Linear constant coefficient DAE**, and write it as

$$A\mathbf{y}'(t) + B\mathbf{y}(t) = f(t) \quad (9.37)$$

It is convenient to write the linear equations in block form, e.g., as

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \mathbf{y}' \end{pmatrix} + \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f}(t) \\ \mathbf{g}(t) \end{pmatrix} \quad (9.38)$$

where the A_{ij} and B_{ij} are matrices, and the A_{2j} are singular. Expanding,

$$A_{11}\mathbf{x}' + A_{12}\mathbf{y}' + B_{11}\mathbf{x} + B_{12}\mathbf{y} = \mathbf{f}(t) \quad (9.39)$$

$$A_{21}\mathbf{x}' + A_{22}\mathbf{y}' + B_{21}\mathbf{x} + B_{22}\mathbf{y} = \mathbf{g}(t) \quad (9.40)$$

The system is semi-explicit if it has the form

$$A_{11}\mathbf{x}' + B_{11}\mathbf{x} + B_{12}\mathbf{y} = \mathbf{f}(t) \quad (9.41)$$

$$B_{21}\mathbf{x} + B_{22}\mathbf{y} = \mathbf{g}(t) \quad (9.42)$$

Definition 9.2. The function $\phi(t)$ is a **solution** of 9.1 on an interval I if

$$\mathbf{F}(t, \phi(t), \phi'(t)) = 0 \quad (9.43)$$

for all $t \in I$. We call the DAE **solvable on I** if a solution (or family of solutions that depend on a parameter) exists on I .

While this construction described earlier suggests that one can solve a DAE by converting it to an ODE, this is not always the best method, since differentiation can be expensive and knowing how to differentiate to get the desired result is a difficult problem. However, the concepts of solvability and the definition of the Index of the DAE are closely related.

9.2 Linear Differential Algebraic Equations with Constant Coefficients

Our focus will be primarily on linear equations of the form

$$A(t)\mathbf{y}' + B(t)\mathbf{y} = \mathbf{f}(t) \quad (9.44)$$

Definition 9.3. A linear system is said to be in **standard canonical form** if it can be partitioned into the form

$$\begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \mathbf{y}' \end{pmatrix} + \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f}(t) \\ \mathbf{g}(t) \end{pmatrix} \quad (9.45)$$

where N is a strictly³ triangular matrix. For linear time-varying systems, both N and C depend on t . Expanding,

$$\mathbf{x}' + C\mathbf{x} = \mathbf{f}(t) \quad (9.46)$$

$$N\mathbf{y}' + \mathbf{y} = \mathbf{g}(t) \quad (9.47)$$

We begin with the linear constant coefficient DAE (9.37).

Definition 9.4 (Matrix Pencil). Let A, B be square matrices and $\lambda \in \mathbb{C}$. Then the **matrix pencil** of A and B is given by

$$P = \lambda A + B \quad (9.48)$$

P is said to be a **regular pencil** if $\det \lambda A + B \neq 0$.

Definition 9.5 (Nilpotent Matrix). A matrix N is said have a **Nilpotency k** if k is the smallest integer such that

$$N^k = 0 \quad (9.49)$$

³A strictly triangular matrix is a triangular matrix with all zeros on the diagonal.

Theorem 9.1. Let $Ay' + By = f$ be a linear constant coefficient DAE with regular pencil $\lambda A + B$. Then there exists nonsingular matrices P, Q , such that

$$PAQ\mathbf{u}' + PBQ\mathbf{u} = P\mathbf{f} \quad (9.50)$$

where $\mathbf{u} = Q^{-1}\mathbf{y}$,

$$PAQ = \begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix}; \quad PBQ = \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \quad (9.51)$$

and N is a matrix of nilpotency k , for some integer $k \geq 0$. If $N = 0$ define $k = 1$; if A is nonsingular, $PAQ = I$ and $k = 0$. If $\det(\lambda A + B)$ is constant then $PAQ = n$, $PBQ = I$.

Definition 9.6. The **index of the pencil** $\lambda A + B$ is the degree of nilpotency in equation 9.51.

Theorem 9.2. Let $Ay' + By = f$ be a linear constant coefficient DAE with regular pencil $\lambda A + B$. Then the index of the pencil equals the index of the DAE.

This theorem says that there exist nonsingular matrices P, Q , such that we can write

$$\begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix} \begin{pmatrix} \mathbf{u}' \\ \mathbf{v}' \end{pmatrix} + \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{g} \\ \mathbf{h} \end{pmatrix} \quad (9.52)$$

where

$$Q\mathbf{y} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}, \quad P\mathbf{f} = \begin{pmatrix} \mathbf{g} \\ \mathbf{h} \end{pmatrix} \quad (9.53)$$

Then

$$\mathbf{u}' + C\mathbf{u} = \mathbf{g} \quad (9.54)$$

$$N\mathbf{v}' + \mathbf{v} = \mathbf{h} \quad (9.55)$$

and N is k -nilpotent, where k is the index of the DAE. Equation 9.54 has a unique solution that is fully determined by its initial conditions (this is the existence theorem for ordinary differential equations). Equation 9.55 also has a unique solution, but does not require initial conditions, because the matrix N is nilpotent ($N^k=0$). To see this let D be the derivative operator; then 9.55 becomes

$$ND\mathbf{v} + \mathbf{v} = \mathbf{h} \quad (9.56)$$

$$(ND + I)\mathbf{v} = \mathbf{h} \quad (9.57)$$

$$\mathbf{v} = (ND + I)^{-1}\mathbf{h} \quad (9.58)$$

But since

$$(I + ND)^{-1} = I - ND + N^2D^2 - N^3D^3 + N^4D^4 + \dots + N^{k-1}D^{k-1} \quad (9.59)$$

we conclude that

$$\mathbf{v} = (I - ND + \cdots + N^{k-1}D^{k-1})\mathbf{h} \quad (9.60)$$

$$= \mathbf{h} - N\mathbf{h}' + N^2\mathbf{h}'' - \cdots + N^{k-1}\mathbf{h}^{(k-1)} \quad (9.61)$$

In other words, equation 9.55 is not a differential equation at all, despite the presence of the derivative, but is an algebraic equation for \mathbf{v} . This is sometimes referred to as a hidden constraint of the DAE.

Two key points emerge from the preceding discussion. First, we see that DAE's may have hidden constraints, namely, part of the solution may be fully determined independently of any initial conditions. Second, too many initial conditions can over-determine the system. If too many initial conditions are provided, that is, more than are necessary to determine \mathbf{u} , then there may not be any solution at all.

Theorem 9.3. *Let $Ay' + By = f$ be a solvable linear constant coefficient DAE with index $k \geq 1$. Then $(A + \lambda B)^{-1}$ has a pole of order k at $\lambda = 0$, and $(A + \lambda B)^{-1}A$ has pole of order $k - 1$ at $\lambda = 0$.*

Proof. We will give the proof for the first statement. The second is left as an exercise.

$$I = Q^{-1}Q \quad (9.62)$$

$$= Q^{-1}(A + \lambda B)^{-1}(A + \lambda B)Q \quad (9.63)$$

$$= Q^{-1}(A + \lambda B)^{-1}P^{-1}P(A + \lambda B)Q \quad (9.64)$$

$$= Q^{-1}(A + \lambda B)^{-1}P^{-1}(PAQ + \lambda PBQ) \quad (9.65)$$

Hence

$$Q^{-1}(A + \lambda B)^{-1}P^{-1} = (PAQ + \lambda PBQ)^{-1} \quad (9.66)$$

$$(A + \lambda B)^{-1} = Q(PAQ + \lambda PBQ)^{-1}P \quad (9.67)$$

$$= Q \begin{pmatrix} I + \lambda C & 0 \\ 0 & N + \lambda I \end{pmatrix}^{-1} P \quad (9.68)$$

$$= Q \begin{pmatrix} (I + \lambda C)^{-1} & 0 \\ 0 & (N + \lambda I)^{-1} \end{pmatrix} P \quad (9.69)$$

Since (this can be verified by multiplying out the product, using the fact that $N^k = 0$)

$$I = \frac{1}{\lambda} \left[I - \frac{1}{\lambda}N + \frac{1}{\lambda^2}N^2 + \cdots + \frac{(-1)^{k-1}}{\lambda^{k-1}}N^{k-1} \right] [N + \lambda I] \quad (9.70)$$

we know that

$$(N + \lambda I)^{-1} = \frac{1}{\lambda} \sum_{i=0}^{k-1} \frac{-1^i}{\lambda^i} N^i \quad (9.71)$$

Hence $A + \lambda B$ has elements that are proportional to $1/\lambda^k$. This gives a pole of order k when $\lambda \rightarrow 0$. \square

9.3 General Linear Differential Algebraic Equations

In the general linear system the matrices A and B may be functions of t ,

$$A(t)y'(t) + B(t)y(t) = f(t) \quad (9.72)$$

Then theorem 9.1 is still valid, but theorem 9.2 is not. In general, the index of the pencil and the index of the DAE are not the same!

Definition 9.7. Let $A(t)y' + B(t)y = f$ be a linear time-varying DAE with pencil $\lambda A(t) + B(t)$. We say the DAE is a **regular DAE** if the pencil is regular for all values of t .

Definition 9.8. Let $A(t)y' + B(t)y = f$ be a linear time-varying DAE with pencil $\lambda A(t) + B(t)$. Then the **local index of the pencil** is the index of the pencil at a given time t .

Theorem 9.4. Let $A(t)y' + B(t)y = f$ be a solvable linear time-varying DAE with pencil $\lambda A(t) + B(t)$. Then the following are equivalent:

1. $A(t)y' + B(t)y = f$ is an ordinary differential equation (system).
2. The index of the DAE $A(t)y' + B(t)y = f$ is zero.
3. The local index of the pencil $\lambda A(t) + B(t)$ is zero for all t .
4. The local index of the pencil is zero for some $t = t_0$.

Theorem 9.5. Let $A(t)y' + B(t)y = f$ be a solvable linear time-varying DAE with pencil $\lambda A(t) + B(t)$. Then the index of the pencil is one if and only if the index of the DAE is one.

Definition 9.9. Let $A(t)u'(t) + B(t)u(t) = f$ be a solvable linear time-varying DAE. An **analytically equivalent** transformation of the DAE is given by

1. Pre-multiply the DAE by a non-singular matrix $P(t)$,

$$PAu' + PBu = Pf \quad (9.73)$$

2. Make the change of variables $u = Q(t)y$, for some non-singular matrix $Q(t)$

$$PA(Qy' + Q'y) + PBQy = Pf \quad (9.74)$$

$$PAQy' + (PAQ' + PBQ)y = Pf \quad (9.75)$$

Theorem 9.6. An analytically equivalent transformation preserves the following properties of a solvable linear DAE:

1. The local index of the pencil is zero
2. The local index of the pencil is one

3. The local index of the pencil is ≥ 2

The following theorem effectively says that there are three broad classes of DAE's: implicit ODE's (index 0 DAEs); index 1 DAEs; and index 2 DAEs.

Theorem 9.7. *Let $A(t)\mathbf{y}'(t) + B(t)\mathbf{y}(t) = \mathbf{f}(t)$ be a solvable linear time-varying DAE with matrix pencil $\lambda A(t) + B(t)$ with index > 2 on some interval I . Then it is possible to find an analytically equivalent transformation of the DAE on some open subinterval of I with local index of 2.*

Theorem 9.8. *Let $A(t)\mathbf{y}'(t) + B(t)\mathbf{y}(t) = \mathbf{f}(t)$ be a linear time-varying DAE where $A(t)$ and $B(t)$ are real and analytic. Then the DAE is solvable if and only if it is analytically equivalent to a system in standard canonical form using a real analytic coordinate change.*

Theorem 9.9. *Let $A(t)\mathbf{u}'(t) + B(t)\mathbf{u}(t) = \mathbf{f}(t)$ be a solvable linear time-varying DAE. Then it is analytically equivalent to*

$$\mathbf{x}' + C\mathbf{y}' = \mathbf{f}(t) \quad (9.76)$$

$$N\mathbf{y}' + \mathbf{y} = \mathbf{g}(t) \quad (9.77)$$

where N is nilpotent and C is a matrix.

9.4 Hessenberg Forms for Linear and Nonlinear DAEs

Recall that a matrix A is said to be a **Hessenberg Matrix** if $A_{i,j} = 0$ for $i > j + 1$, i.e., only the upper triangular, diagonal, and sub-diagonal elements are non-zero:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & & a_{2,n-1} & a_{2,n} \\ 0 & a_{3,2} & a_{3,3} & & \vdots & \vdots \\ 0 & 0 & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & a_{n,n-1} & a_{n,n} \end{pmatrix} \quad (9.78)$$

Hessenberg forms arise commonly in physical problems due to the natural symmetry of the problems. Thus a great deal of current research is based on solving problems of this form.

Definition 9.10. *Let $A(t)\mathbf{y}'(t) + B(t)\mathbf{y}(t) = \mathbf{f}(t)$ be a linear time varying DAE. Then it is in **Hessenberg Size n Form** if it can be written in block form as*

$$\begin{pmatrix} I & 0 & \cdots & 0 \\ 0 & I & \ddots & \vdots \\ \vdots & \ddots & \ddots & \\ 0 & & I & 0 \\ 0 & \cdots & \cdots & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{y}' \end{pmatrix} + \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1,n-1} & B_{1,n} \\ B_{21} & B_{22} & \cdots & B_{2,n-1} & 0 \\ 0 & B_{32} & \cdots & B_{3,n-1} & \vdots \\ 0 & 0 & \ddots & \vdots & \\ 0 & \cdots & 0 & B_{n,n-1} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f}(t) \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{g}(t) \end{pmatrix} \quad (9.79)$$

where $B_{n,n-1}B_{n-1,n-2}\cdots B_{21}$ is nonsingular

Example 9.2. Write the Hessenberg forms of size 2 and 3 in block form.

Solution. The size 2 form:

$$\mathbf{x}' + B_{11}\mathbf{x} + B_{12}\mathbf{y} = \mathbf{f}(t) \tag{9.80}$$

$$B_{21}\mathbf{x} = \mathbf{g}(t) \tag{9.81}$$

The size 3 form:

$$\mathbf{x}' + B_{11}\mathbf{x} + B_{12}\mathbf{y} + B_{13}\mathbf{z} = \mathbf{f}(t) \tag{9.82}$$

$$\mathbf{y}' + B_{21}\mathbf{x} + B_{22}\mathbf{y} = \mathbf{g}(t) \tag{9.83}$$

$$B_{32}\mathbf{y} = \mathbf{h}(t) \tag{9.84}$$

Theorem 9.10. A linear differential algebraic equation in **Hessenberg form of size** n is solvable and has local index n .

For the general, nonlinear DAE

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0 \tag{9.85}$$

we say the DAE is in Hessenberg form of size n if it can be written explicitly as

$$y_1' = F_1(t, y_1, y_2, \dots, y_n) \tag{9.86}$$

$$y_2' = F_2(t, y_1, y_2, \dots, y_{n-1}) \tag{9.87}$$

$$\vdots \tag{9.88}$$

$$y_i' = F_i(t, y_{i-1}, y_i, \dots, y_{n-1}, t) \tag{9.89}$$

$$\vdots \tag{9.90}$$

$$0 = 0F_n(t, y_{n-1}) \tag{9.91}$$

and the product

$$\frac{\partial F_n}{\partial x_{n-1}} \frac{\partial F_{n-1}}{\partial x_{n-2}} \cdots \frac{\partial F_2}{\partial x_1} \frac{\partial F_1}{\partial x_n} \tag{9.92}$$

is nonsingular.

The Hessenberg form of size 1 is defined as

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}, \mathbf{x}) \tag{9.93}$$

$$0 = \mathbf{g}(t, \mathbf{y}, \mathbf{x}) \tag{9.94}$$

where the Jacobian $\partial\mathbf{g}/\partial\mathbf{x}$ is nonsingular. In principle, one could solve the constraint for \mathbf{x} ⁴ and reduce the DAE to an ordinary differential equation in \mathbf{y} . However, it will turn out that uniqueness of the solution is not guaranteed.

⁴This is a consequence of the implicit function theorem and depends on the nonsingularity of the Jacobian.

Example 9.3. Show that $(y')^2 = t^2$, $y(0) = 0$, is a Hessenberg form of size 1 and that its solution is not unique.

Solution. To write the equation in Hessenberg form we add the new variable $x = y'$. Then our system reads as

$$y' = f(t, y, x) = x \quad (9.95)$$

$$0 = g(t, y, x) = x^2 - t^2 \quad (9.96)$$

Since $\partial g/\partial x = 2x$ is not identically zero the Jacobian is nonsingular. To find the solutions we observe that any solution to either of the following equations satisfies the original equation:

$$y' = t \quad (9.97)$$

$$y' = -t \quad (9.98)$$

Hence both $y = t^2/2$ and $y = -t^2/2$ are solutions. Hence

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} t \\ t^2/2 \end{pmatrix} \text{ or } \begin{pmatrix} -t \\ -t^2/2 \end{pmatrix} \quad \square \quad (9.99)$$

The Hessenberg form of size 2 can be written as

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}, \mathbf{x}) \quad (9.100)$$

$$0 = \mathbf{g}(t, \mathbf{y}) \quad (9.101)$$

where the product of Jacobians

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \quad (9.102)$$

is nonsingular

The Hessenberg form of size 3 is

$$\mathbf{x}' = \mathbf{f}(t, \mathbf{x}, \mathbf{y}, \mathbf{z}) \quad (9.103)$$

$$\mathbf{y}' = \mathbf{g}(t, \mathbf{x}, \mathbf{y}) \quad (9.104)$$

$$0 = \mathbf{h}(t, \mathbf{y}) \quad (9.105)$$

where the product of Jacobians

$$\frac{\partial \mathbf{h}}{\partial \mathbf{y}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \frac{\partial \mathbf{f}}{\partial \mathbf{z}} \quad (9.106)$$

is nonsingular

Example 9.4. Show that pendulum equations discussed in example 9.1 is an Hessenberg Index-3 DAE.

Solution. Write the pendulum equations from the earlier example as

$$p' = u \quad (9.107)$$

$$q' = v \quad (9.108)$$

$$u' = -Tp \quad (9.109)$$

$$v' = g - Tq \quad (9.110)$$

$$0 = p^2 + q^2 - 1 \quad (9.111)$$

and make the associations $\mathbf{x} = (u, v)^T$, $\mathbf{y} = (p, q)^T$, and $\mathbf{z} = (T)$. Then

$$\mathbf{f}(t, \mathbf{x}, \mathbf{y}, \mathbf{z}) = \begin{pmatrix} -zy_1 \\ g - zy_2 \end{pmatrix} \quad (9.112)$$

$$\mathbf{g}(t, \mathbf{x}, \mathbf{y}, \mathbf{z}) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (9.113)$$

$$\mathbf{h}(t, \mathbf{y}) = y_1^2 + y_2^2 - 1 \quad (9.114)$$

and

$$\frac{\partial \mathbf{h}}{\partial \mathbf{y}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = (2y_1, 2y_2) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -y_1 \\ -y_2 \end{pmatrix} \quad (9.115)$$

$$= -2(y_1^2 + y_2^2) = -2(p^2 + q^2) = -2 \neq 0 \quad \square \quad (9.116)$$

9.5 Implementation: A Detailed Example

Implementing a general solver for the differential algebraic equation $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$ can be quite complicated. If we use a method that approximates \mathbf{y}' with

$$\mathbf{y}'_n = \phi_n(\dots, \mathbf{y}_{n+1}, \mathbf{y}_n, \mathbf{y}_{n-1}, \dots) \quad (9.117)$$

then we end up with a system of m nonlinear equations for the \mathbf{y}_n , where $m = \dim \mathbf{y}$:

$$\mathbf{F}(t_n, \mathbf{y}_n, \phi_n(\dots, \mathbf{y}_n, \dots)) = 0 \quad (9.118)$$

For, example, with the Backward Euler Method we have

$$\phi_n = \frac{1}{h}(\mathbf{y}_n - \mathbf{y}_{n-1}) \quad (9.119)$$

which gives us the nonlinear system

$$\mathbf{F}\left(t_n, \mathbf{y}_n, \frac{1}{h}(\mathbf{y}_n - \mathbf{y}_{n-1})\right) = 0 \quad (9.120)$$

Let us consider an implementation of the index-1 system that describes enzymatic conversion according to the system of chemical reactions



In this system the input (or *substrate*) A is converted to B (called the *product*) as a result of interactions with an enzyme E . First, we have that A combines with E to form a *molecular complex* that we call X , at a rate that is proportional to the concentrations of both A and E , with constant of proportionality (called the *rate constant*) α . Let A , B , E , and X denote the total concentrations of each of these chemicals. Then for each molecule of X created, one molecule each of A and E are consumed, e.g.,

$$X' = \alpha AE + \text{other reactions affecting } X \quad (9.124)$$

$$A' = -\alpha AE + \text{other reactions affecting } A \quad (9.125)$$

The second reaction tells us that X can be converted back to A and E at a rate β ,

$$X' = -\beta X + \text{other reactions affecting } X \quad (9.126)$$

$$A' = \beta X + \text{other reactions affecting } A \quad (9.127)$$

The third reaction tells us that X can also be converted to B and E at the same time:

$$X' = -\gamma X + \text{other reactions affecting } X \quad (9.128)$$

$$B' = \gamma X + \text{other reactions affecting } B \quad (9.129)$$

Finally, we observe that E and X are really different forms of the same molecule, i.e., X is the same thing as E with A bound to it, so we have a conservation law

$$E + X = 1 \quad (9.130)$$

Putting it all together gives us the following DAE:

$$A' = -\alpha AE + \beta X \quad (9.131)$$

$$B' = \gamma X \quad (9.132)$$

$$X' = \alpha AE - (\gamma + \beta)X \quad (9.133)$$

$$0 = E + X - 1 \quad (9.134)$$

This particular system is semi-explicit, that is, we can clearly tell which equations involve the explicit derivatives, and which equation involves the constraint. Thus there are (at least) three ways we can go about solving this system numerically.

1. We could differentiate 9.134 and solve the fully explicit system of four ODE's:

$$A' = -\alpha AE + \beta X \quad (9.135)$$

$$B' = \gamma X \quad (9.136)$$

$$X' = \alpha AE - (\gamma + \beta)X \quad (9.137)$$

$$E' = -\alpha AE + (\gamma + \beta)X \quad (9.138)$$

This method has the disadvantage that it obscures the constraint, i.e., by looking at the equations it is not at all obvious that $E + X = 1$. In fact, this system satisfies any constraint $E + X = C$ for any constant C , which you can see by adding the last two equations together to obtain $d/dt(E + X) = 0$.

2. We could solve 9.157 for E and eliminate E completely from the system, giving a system of three explicit ODE's:

$$A' = -\alpha A(1 - X) + \beta X \quad (9.139)$$

$$B' = \gamma X \quad (9.140)$$

$$X' = \alpha A(1 - X) - (\gamma + \beta)X \quad (9.141)$$

Then the value of E can be computed from the constraint $E = 1 - X$ after X is known. This is the method that is generally used to solve a semi-explicit system where we are able to solve for the constraints and reduce the system to an ODE.

3. We can treat the system as a full implicit DAE as in 9.143 and use Newton's method to extract \mathbf{y}_n at each step. This is the the only procedure that can be used for a fully implicit system, that is, a DAE where it is not clear how to separate the constraints from the ODEs, and so we will illustrate the procedure in detail.

Algorithm 9.1. Fully Implicit DAE with Backward Euler

To solve the fully implicit differential algebraic equation

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') \quad (9.142)$$

At each time step t_n , solve

$$\mathbf{F}\left(t_n, \mathbf{y}_n, \frac{1}{h}(\mathbf{y}_n - \mathbf{y}_{n-1})\right) = 0 \quad (9.143)$$

for \mathbf{y}_n using Newton's method

$$\mathbf{y}_n^{k+1} = \mathbf{y}_n^k - J^{-1} \mathbf{F}\left(t_n, \mathbf{y}_n, \frac{1}{h}(\mathbf{y}_n - \mathbf{y}_{n-1})\right) \quad (9.144)$$

where the Jacobian J is

$$J = \frac{1}{h} \frac{\partial \mathbf{F}}{\partial \mathbf{y}'} + \frac{\partial \mathbf{F}}{\partial \mathbf{y}} \quad (9.145)$$

In general it is more efficient to calculate updates by solving

$$J \Delta \mathbf{y} = \mathbf{F} \quad (9.146)$$

for $\Delta \mathbf{y}$, e.g., by Gaussian elimination on a sparse system, and updating

$$\mathbf{y}_n^{k+1} = \mathbf{y}_n^k + \Delta \mathbf{y} \quad (9.147)$$

instead of inverting J .

Algorithm 9.2. Semi-explicit Index-1 DAE with Backward Euler

To solve the semi-implicit index-1 DAE

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}, \mathbf{z}) \quad (9.148)$$

$$0 = \mathbf{g}(t, \mathbf{y}, \mathbf{z}) \quad (9.149)$$

where $\mathbf{g}_{\mathbf{z}}$ is non-singular, at each time step t_n solve the constraint for \mathbf{z}_n

$$\mathbf{z} = \mathbf{h}(t_n, \mathbf{y}_n) \quad (9.150)$$

and substitute the result back into the ODE,

$$\mathbf{y}'_n = \mathbf{f}(t_n, \mathbf{y}_n, \mathbf{h}(t_n, \mathbf{y}_n)) \quad (9.151)$$

$$(9.152)$$

This produces an ordinary differential system that can be discretized using backwards-Euler (or any other method of choice) via

$$\frac{1}{h}(\mathbf{y}_n - \mathbf{y}_{n-1}) = \mathbf{f}(t_n, \mathbf{y}_n, \mathbf{h}(t_n, \mathbf{y}_n)) \quad (9.153)$$

Solve for \mathbf{y}_n using a Newton root-finder, and then proceed to the next mesh point.

Let us proceed by implementing an equation-specific solution for the implicit DAE

$$A' = -\alpha AE + \beta X \quad (9.154)$$

$$B' = \gamma X \quad (9.155)$$

$$X' = \alpha AE - (\gamma + \beta)X \quad (9.156)$$

$$0 = E + X - 1 \quad (9.157)$$

Using the backward Euler method, the iteration formulas are

$$0 = F_1(A_n, B_n, X_n, E_n) = \frac{1}{h}(A_n - A_{n-1}) + \alpha A_n E_n - \beta X_n \quad (9.158)$$

$$0 = F_2(A_n, B_n, X_n, E_n) = \frac{1}{h}(B_n - B_{n-1}) - \gamma X_n \quad (9.159)$$

$$0 = F_3(A_n, B_n, X_n, E_n) = \frac{1}{h}(X_n - X_{n-1}) - \alpha A_n E_n + (\gamma + \beta)X_n \quad (9.160)$$

$$0 = F_4(A_n, B_n, X_n, E_n) = E_n + X_n - 1 \quad (9.161)$$

We need to solve this explicitly at each time-step for the values of A_n, B_n, E_n, X_n ;

to do this using Newton's method we must compute the Jacobian matrix is

$$J = \begin{pmatrix} \partial F_1/\partial A_n & \partial F_1/\partial B_n & \partial F_1/\partial X_n & \partial F_1/\partial E_n \\ \partial F_2/\partial A_n & \partial F_2/\partial B_n & \partial F_2/\partial X_n & \partial F_2/\partial E_n \\ \partial F_3/\partial A_n & \partial F_3/\partial B_n & \partial F_3/\partial X_n & \partial F_3/\partial E_n \\ \partial F_4/\partial A_n & \partial F_4/\partial B_n & \partial F_4/\partial X_n & \partial F_4/\partial E_n \end{pmatrix} \quad (9.162)$$

$$= \begin{pmatrix} 1/h + \alpha E_n & 0 & -\beta & \alpha A_n \\ 0 & 1/h & -\gamma & 0 \\ -\alpha E_n & 0 & 1/h + (\gamma + \beta) & -\alpha A_n \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad (9.163)$$

We can implement the Jacobian calculation in Mathematica as follows

```
jac[f_, vars_] := Module[dF,
  If[Length[f] ≠ Length[vars], Return[$Failed]];
  dF[func_] := Map[D[func, #] &, vars];
  Return[Map[dF, f]]
]
```

and then we can calculate the Jacobian with

```
F={ (An-An-1)/h + (α*An*en-β*Xn),
  (Bn-Bn-1)/h - γ*Xn,
  (Xn-Xn-1)/h - (α*An*en - (β + γ)*Xn),
  en+Xn-1 };
jac[F, {An, Bn, Xn, en}]
```

Note that because the symbol E is reserved in Mathematica for the exponential e , we have replaced the symbol for the enzyme with a lower case e .

At each time step we need to apply the iteration formula and invert it using Newton's formula, e.g.,

```
nextStep[h_, {ALast_, BLast_, XLast_, eLast_}] :=
Module[F, J, A, X, e, B, Δ, IJ, IJF, ε=10-12, An, Xn, en, Bn,
F={ (An-ALast)/h + (α*An*en-β*Xn),
  (Bn-BLast)/h - γ*Xn,
  (Xn-XLast)/h - (α*An*en - (β + γ)*Xn),
  en+Xn-1 };
J= jac[F, {An, Bn, Xn, en}];
IJ = Inverse[J];
IJF = IJ.F;
{A, B, X, e} = ALast, BLast, XLast, eLast;
For [i = 0, i 5, i++,
  Δ = -IJF /. {An -> A, Bn -> B, Xn -> X, en -> e};
  {A, B, X, e} += Δ;
  If[Abs[Δ] < ε, Break[]];
];
Return[A, B, X, e]
]
```

In practice it is better not to compute the inverse, but instead to use a linear solver to compute the update, however, this program is for illustrative purposes and the inverse is sufficient. Finally, we can implement the solver. Of course we also need to give values to the parameters α, β, γ .

```
{An, Bn, Xn, en} = {1, 0, 0, 1};
h = 0.01;
α=10; β=γ=1;
data = {{0, An, Bn, Xn, en}};
For[t = 0, t < 3, t += h,
  {An, Bn, Xn, en} = nextStep[h, {An, Bn, Xn, en}}];
data = Append[data, {t+h, An, Bn, Xn, en}}];
];
```

This will generate a table of values of ordered sets of values $\{t_0, A_0, B_0, X_0, e_0\}, \{t_1, A_1, B_1, X_1, e_1\}, \dots$ at each of the mesh points. To see the data we could type

```
TableForm[data]
```

Mathematica would respond with (only some of the data is shown):

0	1	0	0	1
0.01	0.916713	0.0824626	0.000824626	0.917537
0.02	0.84639	0.151273	0.00233735	0.0848727
⋮	etc			

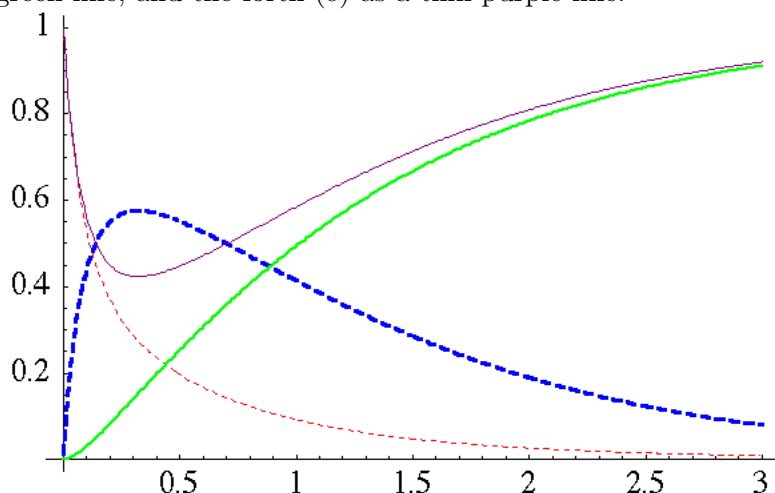
To plot all the curves on a single graph using `ListPlot` we write the function `plotDataTable`,

```
plotDataTable[d_, styles_] := Module[{nvars, makedata, plt, data,
  plots},
  nvars = Length[d[[1]]] - 1;
  data = Transpose[d];
  makedata[i_] := {data[[1]], data[[i + 1]]};
  plt[i_] := ListPlot[makedata[i]
    Transpose, PlotJoined -> True, PlotStyle -> styles[[i]],
    DisplayFunction -> Identity];
  plots = plt /@ Range[nvars];
  Show[plots, DisplayFunction -> $DisplayFunction];
]
```

The to actually generate the plot,

```
plotDataTable[data,
  {{Red, Dashing[ {.006, 0.008} ]},
  {Blue, Thickness[.006], Dashing[ {0.01} ]},
  {Green, Thickness[.005]},
  Purple}]
```

Each curve will be plotted in its own style: the first column of data (A) as a thin, dashed, red line; the second (B) as a thick blue dashed line; the third (X) as a thick green line; and the fourth (e) as a thin purple line:



The figure shows how all of the substrate (A , thin, red, dashed curve) is eventually converted to the product (B , thick green). The concentration of enzyme dips, and the concentration of intermediate (X , thick blue dashes) while the conversion is taking place, but as the process reaches completion they return to their steady state values.

Unfortunately, the simple process of writing

$$0 = F\left(t_n, y_n, \frac{y_n - y_{n-1}}{h}\right) \quad (9.164)$$

(or using any other discretization method besides backwards Euler) does not always work. For some equations, it is even possible to find any discretization formula at all. Consider, for example, the index-2 differential-algebraic equation

$$x' - ty' = 0 \quad (9.165)$$

$$x - ty = f(t) \quad (9.166)$$

The exact solution is

$$x = f(t) - tf'(t) \quad (9.167)$$

$$y = -f'(t) \quad (9.168)$$

which may be verified by substitution. Backward Euler gives the formulas

$$\frac{x_n - x_{n-1}}{h} - t_n \frac{y_n - y_{n-1}}{h} = 0 \quad (9.169)$$

$$x_n - t_n y_n = f(t_n) \quad (9.170)$$

There is no solution for (x_n, y_n) in terms of the variables at earlier times. This can be seen by rewriting the system as

$$x_n - t_n y_n = h(x_{n-1} - t_n y_{n-1}) \quad (9.171)$$

$$x_n - t_n y_n = f(t_n) \quad (9.172)$$

or in matrix form

$$\begin{pmatrix} 1 & -t_n \\ 1 & -t_n \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} h(x_{n-1} - t_n y_{n-1}) \\ f(t_n) \end{pmatrix} \quad (9.173)$$

The matrix of coefficients is singular hence there is no solution.

9.6 BDF Methods for DAEs

While the numerical solution of initial value problems for ordinary differential equations has a long history going back over one hundred years, the numerical solution of DAEs is mostly still an open research issue (see [5] for the more common methods) with significant interest only dating back to the 1970's. Since many of the scientific problems of interest turn out to be stiff, early research focused first on BDF methods and later on some RK methods that are known to be better for stiff problems. The first DAE algorithms published were BDF methods.⁵ The solution of linear and index-1 systems are now understood but methods for higher index systems still form the basis of much current methods.

In general variable step sizes can cause problems with DAEs. For example, it is not possible to solve an index-3 system using implicit Euler with variable step-size because of the following theorem because it will have $O(1)$ errors.

Theorem 9.11. *The global error for a k -step BDF applied to an index- ν system is $O(h^n)$ where $n = \min(k, k - \nu + 2)$*

Furthermore, all multistep (as well as Runge-Kutta) methods are unstable for higher index (≥ 3) systems, and in some cases fail for specific index-1 and index-2 systems. Some convergence results for semi-implicit and fully-implicit index-1 systems, semi-explicit index-2 systems, and index-3 systems in Hessenberg form have been worked out.

We can write the general k -step BDF method for a scalar ODE (see 6.105) as

$$\sum_{j=0}^k a_j y_{k-j} = hb_0 D y_n \quad (9.174)$$

where D is the differential operator $Dy = y'$. Then the general BDF method becomes

$$0 = \mathbf{F}(t_n, \mathbf{y}_n, D\mathbf{y}_n) \quad (9.175)$$

⁵C.W.Gear (1971) "Simultaneous Numerical Solution of Differential-Algebraic Equations," *IEEE Transactions on Circuit Theory*, 18:89-95

where

$$D\mathbf{y}_n = \frac{1}{hb_0} \sum_{j=0}^k a_j \mathbf{y}_{k-j} \quad (9.176)$$

Theorem 9.12. *Let*

$$A\mathbf{y}' + B\mathbf{y} = \mathbf{f}(t) \quad (9.177)$$

be a linear constant coefficient DAE of index- ν with regular pencil $\lambda A + B$, and let 9.174 be a k -step (where $k < 7$) constant step-size BDF method. Then the BDF method converges for this system to order $O(h^k)$ after $(\nu - 1)k + 1$ steps.

Proof. The BDF method for the linear equation is

$$AD\mathbf{y}_n + B\mathbf{y}_n = \mathbf{f}_n(t) \quad (9.178)$$

where $D\mathbf{y}_n$ is given by 9.176. Since the pencil is regular there are nonsingular matrices P and Q , where

$$PAQ = \begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix}; \quad PBQ = \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \quad (9.179)$$

with N a nilpotent matrix of nilpotency ν , so that

$$PAQD\mathbf{y}_n + PBQ\mathbf{y}_n = P\mathbf{f}_n(t) \quad (9.180)$$

$$\begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix} D\mathbf{y}_n + \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \mathbf{y}_n = P\mathbf{f}_n(t) \quad (9.181)$$

Partitioning

$$\mathbf{y}_n = \begin{pmatrix} \mathbf{u}_n \\ \mathbf{v}_n \end{pmatrix}; \quad P\mathbf{f}_n(t) = \begin{pmatrix} \mathbf{g}_n(t) \\ \mathbf{h}_n(t) \end{pmatrix} \quad (9.182)$$

Multiplying out the matrix expressions,

$$D\mathbf{u}_n + C\mathbf{u}_n = \mathbf{g}_n(t) \quad (9.183)$$

$$ND\mathbf{v}_n + \mathbf{v}_n = \mathbf{h}_n(t) \quad (9.184)$$

Equation 9.183 is a pure ODE expression for the BDF method that we already know is convergent of order- k . Equation 9.184 can be rewritten as

$$(I + ND)\mathbf{v}_n = \mathbf{h}_n(t) \quad (9.185)$$

From equation 9.59

$$(I + ND)^{-1} = \sum_{j=0}^{\nu-1} (-1)^j N^j D^j \quad (9.186)$$

and therefore

$$\mathbf{v}_n = \sum_{j=0}^{\nu-1} (-1)^j N^j D^j \mathbf{h}_n(t) \quad (9.187)$$

For the BDF method 9.176,

$$D\mathbf{y}_n = \mathbf{y}'_n + O(h^k) \quad (9.188)$$

hence

$$D^j \mathbf{v}_n = \mathbf{v}_n^{(j)} + O(h^k) \quad (9.189)$$

where the superscript $\mathbf{v}^{(j)}$ represents the j -th derivative, and therefore

$$\mathbf{v}_n = \sum_{j=0}^{\nu-1} (-1)^j N^j(\mathbf{h}_n^{(j)}(t) + O(h^k)) \quad (9.190)$$

Therefore the method converges with order h^k . □

For proofs of the following results see [5].

Theorem 9.13. *Let $\mathbf{F}(t, \mathbf{y}, \mathbf{y}')$ be a fully-implicit index-1 system. Then its numerical solution by the k -step ($k < 7$) BDF with fixed step-size converges to $O(h^k)$ if all initial values are correct to $O(h^k)$ and the Newton iteration is solved at each step to $O(h^{k+1})$ accuracy.*

Theorem 9.14. *Let*

$$\mathbf{f}(t, \mathbf{y}, \mathbf{y}', \mathbf{z}) = 0 \quad (9.191)$$

$$\mathbf{g}(t, \mathbf{y}, \mathbf{z}) = 0 \quad (9.192)$$

be a solvable index-2 DAE such that \mathbf{f} and \mathbf{g} are continuously differentiable (as many times as necessary), that $\partial\mathbf{f}/\partial\mathbf{y}'$ is defined and bounded, and that $\partial\mathbf{g}/\partial\mathbf{y}$ has constant rank. Then the k -step ($k < 7$) BDF method applied to this DAE converges with order $O(h^k)$ if the initial values are all accurate to $O(h^k)$ and the Newton iteration is performed at each step to $O(h^{k+1})$.

Theorem 9.15. *Let*

$$\mathbf{x}' = \mathbf{F}(t, \mathbf{x}, \mathbf{y}, \mathbf{z}) \quad (9.193)$$

$$\mathbf{y}' = \mathbf{G}(t, \mathbf{x}, \mathbf{y}) \quad (9.194)$$

$$0 = \mathbf{H}(t, \mathbf{y}) \quad (9.195)$$

be an index-three Hessenberg system. Then the k -step ($k < 7$) BDF with constant step size converges to $O(h^k)$ after $k+1$ steps if the starting values are consistent to $O(h^{k+1})$ and the Newton iteration at each step is performed to $O(h^{k+2})$ when $k \geq 2$, and to $O(h^{k+3})$ when $k = 1$.

9.7 Runge-Kutta Methods for DAEs

Runge-Kutta methods have the advantage that it is possible to construct higher-order A-stable methods for differential-algebraic equations. Furthermore, they are useful for problems that are highly discontinuous because multistep methods such as BDF need to be completely restarted after each discontinuity and are not completely accurate for the first few iterations. In fact RK methods are sometimes used as starter methods for these multistep methods. On the other hand, RK methods have multiple function evaluations, which means they can be inefficient to implement: a completely general s -stage method requires s^2 evaluations at each mesh point. Furthermore, implicit Runge-Kutta methods suffer from *order reduction*, a phenomenon in which the order of the method for the algebraic constraint is lower than the order of the method for the differential system. We will only study RK methods applied to the simplest system, namely, index-1 linear systems, and will generalize our results without proof to more complicated systems.⁶

Recall that we wrote the general Runge-Kutta method (equation 5.99) for the initial value problem $y' = f(t, y), y(t_0) = y_0$ as

$$Y_i = y_{n-1} + h \sum_{j=1}^s a_{ij} f(t_{n-1} + c_j h, Y_j) \quad (9.196)$$

$$y_n = y_{n-1} + h \sum_{i=1}^s b_i f(t_{n-1} + c_i h, Y_i) \quad (9.197)$$

This can be reformulated by replacing $f(t_{n-1} + c_j h, Y_j)$ with Y'_j ,

$$Y_i = y_{n-1} + h \sum_{j=1}^s a_{ij} Y'_j \quad (9.198)$$

$$y_n = y_{n-1} + h \sum_{i=1}^s b_i Y'_i \quad (9.199)$$

where Y'_j represents the numerical estimate of Y' at $t_{n-1} + c_j h$. Then the discretization of

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0 \quad (9.200)$$

becomes

$$\mathbf{F} \left(t_{n-1} + c_i h, \mathbf{y}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{Y}'_j, \mathbf{Y}'_i \right) = 0 \quad (9.201)$$

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h \sum_{i=1}^s b_i \mathbf{Y}'_i \quad (9.202)$$

⁶See [5][Chapter 4] for a more complete discussion. The material in this section roughly follows the paper by L.R. Petzold (1986) "Order results for implicit Runge-Kutta methods applied to differential/algebraic systems," *SIAM Journal of Numerical Analysis*, 23(4):837-852. For details on the proofs that are sketched and further examples the reader should refer to Petzold's paper.

Definition 9.11. Let $y_n = y_{n-1} + h\psi(t_{n-1}, y_{n-1})$ be a numerical method. Then we define the **local error** as

$$d_n = y(t_{n-1}) + h\psi(t_{n-1}, y(t_{n-1})) - y(t_n) \quad (9.203)$$

The local error defined above is just h times the local truncation error we found earlier - if a method has LTE $O(h^p)$ then it has local error $O(h^{p+1})$. (See definition 4.5.)

For the Runge-Kutta method 9.202,

$$\psi = \sum_{i=1}^s b_i \mathbf{Y}'_i \quad (9.204)$$

where the \mathbf{Y}'_i are determined by solving the system of algebraic equations 9.201. If the equation is linear and solvable (hence with regular pencil)

$$A\mathbf{y}' + B\mathbf{y} = \mathbf{f}(t) \quad (9.205)$$

then the RK method becomes

$$A\mathbf{Y}'_i + B \left(\mathbf{y}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{Y}'_j \right) = \mathbf{f}(t_{n-1} + c_i h) \quad (9.206)$$

along with equation 9.202.

We have previously seen that when the linear system is solvable then there exist matrices P and Q such that it can be transformed into canonical form with

$$PAQ = \begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix}; \quad PBQ = \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \quad (9.207)$$

where

$$N = \text{diag}(N_1, \dots, N_K) \quad (9.208)$$

$$N_i = \text{subdiagonal}(1, \dots, 1) \quad (9.209)$$

Here N is nilpotent of nilpotency k , where k is the index of if the DAE; hence for index-1 systems, we have $N = 0$.

Definition 9.12. If $PAQ = N$ and $PBQ = I$ with N nilpotent then the system is called **completely singular**.

Definition 9.13. If N has the form subdiagonal $1, \dots, 1$ for a completely singular system then the system is called a **canonical completely singular system**.

Premultiplying the linear system by P gives

$$PAQQ^{-1}\mathbf{Y}'_i + PBQQ^{-1} \left(\mathbf{y}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{Y}'_j \right) = P\mathbf{f}(t_{n-1} + c_i h) \quad (9.210)$$

we can transform to canonical form

$$\begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix} \tilde{\mathbf{Y}}'_i + \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \left(\tilde{\mathbf{y}}_{n-1} + h \sum_{j=1}^s a_{ij} \tilde{\mathbf{Y}}'_j \right) = \mathbf{g}(t_{n-1} + c_i h) \quad (9.211)$$

$$\tilde{\mathbf{y}}_n = \tilde{\mathbf{y}}_{n-1} + h \sum_{i=1}^s b_i \tilde{\mathbf{Y}}'_i \quad (9.212)$$

where

$$\tilde{\mathbf{Y}}'_i = Q^{-1} \mathbf{Y}'_i, \quad \tilde{\mathbf{y}}_n = Q^{-1} \mathbf{y}_n, \quad \mathbf{g}(t) = P \mathbf{y}(t) \quad (9.213)$$

Partitioning the solutions as

$$\tilde{\mathbf{Y}} = \begin{pmatrix} \mathbf{U} \\ \mathbf{V} \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}, \quad \tilde{\mathbf{g}} = \begin{pmatrix} \mathbf{p} \\ \mathbf{q} \end{pmatrix} \quad (9.214)$$

and multiplying out the matrix expression in equation 9.211,

$$\mathbf{U}'_i + C \left(\mathbf{u}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{U}'_j \right) = \mathbf{p}(t_{n-1} + c_i h) \quad (9.215)$$

$$N \mathbf{V}'_i + \mathbf{v}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{V}'_j = \mathbf{q}(t_{n-1} + c_i h) \quad (9.216)$$

Equation 9.215 is an ordinary differential equation that has been discretized using an s-stage Runge-Kutta method, which has properties that we have already studied. Equation 9.216 contains the “singular” part of the system, and because the differential and singular parts of the system are completely decoupled, it suffices to just study the canonical singular subsystem.

Let $v_{n,p}, q_p$ denote the p^{th} component of the corresponding vectors \mathbf{v}_n, \mathbf{q} , let $V_{j,p}$ be the j^{th} stage derivative for the p^{th} component of \mathbf{V}_j , and denote the RK matrix (a_{ij}) by \mathcal{A} . Then for the index-1 system (with $N = 0$) equation 9.216 becomes

$$v_{n-1,p} + h \mathcal{A} \mathbf{V}' = q_p(t_{n-1} + c_i h) \quad (9.217)$$

where \mathbf{V} is the vector of stage derivatives corresponding to component p . Equation 9.217 must hold for all values of h , and in the limit $h \rightarrow 0$,

$$v_{n-1,p} = q_p(t_{n-1}) \quad (9.218)$$

Omitting the index p to simplify the notation and assuming that the matrix \mathcal{A} is nonsingular,

$$\mathbf{V}' = \frac{1}{h} \mathcal{A}^{-1} [(q(t_{n-1} + c_i h) - v_{n-1})]_{i=1, \dots, s} \quad (9.219)$$

$$= \frac{1}{h} \mathcal{A}^{-1} [(q(t_{n-1} + c_i h) - q(t_{n-1}))]_{i=1, \dots, s} \quad (9.220)$$

where $[w(i)]_{i=1,\dots,s}$ represents the vector of components indexed by i . Define

$$\mathbf{G} = \frac{1}{h} \begin{pmatrix} g(t_{n-1} + c_1 h) - g(t_{n-1}) \\ g(t_{n-1} + c_2 h) - g(t_{n-1}) \\ \vdots \\ g(t_{n-1} + c_s h) - g(t_{n-1}) \end{pmatrix} \quad (9.221)$$

so that

$$\mathbf{V}' = \mathcal{A}^{-1} \mathbf{G} \quad (9.222)$$

Then from equation 9.212,

$$v_n = v_{n-1} + h \mathbf{b}^T \mathbf{V}' \quad (9.223)$$

$$= g(t_{n-1}) + h \mathbf{b}^T \mathcal{A}^{-1} \mathbf{G} \quad (9.224)$$

The local error is thus

$$d_n = g(t_{n-1}) + h \mathbf{b}^T \mathcal{A}^{-1} \mathbf{G} - g(t_n) \quad (9.225)$$

$$= - \left(hg' + \frac{h^2}{2} g'' + \frac{h^3}{6} g''' + \dots \right) \quad (9.226)$$

$$+ \mathbf{b}^T \mathcal{A}^{-1} \begin{pmatrix} c_1 hg' + \frac{c_1^2 h^2}{2} g'' + \frac{c_1^3 h^3}{g} g''' + \dots \\ \vdots \\ c_s hg' + \frac{c_s^2 h^2}{2} g'' + \frac{c_s^3 h^3}{g} g''' + \dots \end{pmatrix} \quad (9.227)$$

Definition 9.14. The algebraic order of an IRK method is equal to k if $d_n = O(h^{k+1})$ for all components.

Theorem 9.16. The algebraic order of an IRK method equal to k if and only if

$$\mathbf{b}^T \mathcal{A}^{-1} \mathbf{c}^j = 1, \text{ for } j = 1, \dots, k \quad (9.228)$$

where

$$\mathbf{c}^j = (c_1^j, \dots, c_s^j)^T \quad (9.229)$$

We state the remaining results without proof.

Definition 9.15. A perturbation $(z_n, \delta_0, \dots, \delta^{s+1})$ of the method for y_n ,

$$\mathbf{F} \left(t_{n-1} + c_i h, \mathbf{y}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{Y}'_j, \mathbf{Y}'_i \right) = 0 \quad (9.230)$$

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h \sum_{i=1}^s b_i \mathbf{Y}'_i \quad (9.231)$$

is given by

$$\mathbf{F} \left(t_{n-1} + c_i h, \mathbf{z}_{n-1} + h \sum_{j=1}^s a_{ij} \mathbf{z}'_j + \delta_n^{(i)}, \mathbf{z}'_i \right) = 0 \quad (9.232)$$

$$\mathbf{z}_n = \mathbf{z}_{n-1} + h \sum_{i=1}^s b_i \mathbf{z}'_i + \delta_n^{s+1} \quad (9.233)$$

where $z_0 = y_0 + \delta_0^{s+1}$.

Definition 9.16. Let $(z_n, \delta_0, \dots, \delta^{s+1})$ be a perturbation of the IRK method y_n for a particular differential algebraic equation $F(t, y, y') = 0$, with $|\delta_n^{(i)}| \leq \Delta, i = 1, \dots, s+1$, for some number Δ . Then if $|z_n - y_n| \leq K_0 \Delta$ for $0 < h \leq h_0$, for some constants K_0 and h_0 that depend only on the method and the differential algebraic equation, then we say the method is **strictly stable for the DAE**.

Theorem 9.17. An IRK method with \mathcal{A} nonsingular is strictly stable for a linear constant-coefficient index-1 differential algebraic equation iff and only iff

$$|1 - \mathbf{b}^T \mathcal{A}^{-1} \mathbf{1}| < 1 \quad (9.234)$$

where $\mathbf{1}$ is as defined in equation 5.211.

Recall from that the stability function $R(z) = y_n/y_{n-1}$ satisfies

$$R(z) = 1 + z \mathbf{b}^T (\mathbf{I} - z \mathcal{A})^{-1} \mathbf{1} \quad (9.235)$$

Taking the limit as $|z| \rightarrow \infty$,

$$\lim_{|z| \rightarrow \infty} R(z) = 1 - \mathbf{b}^T \mathcal{A}^{-1} \mathbf{1} \quad (9.236)$$

From theorem 9.17 implies that an implicit Runge-Kutta method is stable when

$$r = \lim_{|z| \rightarrow \infty} |R(z)| < 1 \quad (9.237)$$

Definition 9.17. k_c is called the **constant coefficient order** of an implicit Runge-Kutta method if the method converges with global error $O(h^{k_c})$ for all linear constant coefficient index-one systems.

Theorem 9.18. Let $\mathcal{A}, \mathbf{b}, \mathbf{c}$ be an implicit Runge-Kutta method that is strictly stable, with constant-coefficient order k_c , algebraic order k_a , and differential order k_d . Then

$$k_c = \min(k_a + 1, k_d) \quad (9.238)$$

Example 9.5. *The IRK method*

$$\begin{array}{c|cc} (3 + \sqrt{3})/6 & (3 + \sqrt{3})/6 & 0 \\ (3 - \sqrt{3})/6 & -(\sqrt{3})/3 & (3 + \sqrt{3})/6 \\ \hline & 1/2 & 1/2 \end{array} \quad (9.239)$$

has

$$5 = \frac{1 + \sqrt{3}}{2 + \sqrt{3}} \quad (9.240)$$

hence it is stable, and

$$k_d = 3, \quad k_a = 1 \quad (9.241)$$

hence the overall order of the method is $k_c = 2$. \square

For a full discussion of Runge-Kutta methods applied to higher index and non-linear problems the student is referred to [5].

Chapter 10

Appendix on Analytic Methods (Math 280 in a Nutshell)

This chapter is included for reference only; the level of detail is less than the rest of the notes and only a summary of results is provided.

First Order Linear Equations

Equations of the form $y' + p(t)y = q(t)$ have the solution

$$y(t) = \frac{1}{\mu(t)} \left(C + \int \mu(s)q(s)ds \right)$$

where

$$\mu(t) = \exp \left(\int_t p(s)ds \right)$$

Exact Equations

An differential equation

$$M(t, y)dt + N(t, y)dy = 0$$

is exact if

$$\frac{\partial M}{\partial y} = \frac{\partial N}{\partial t}$$

in which case the solution is a $\phi(t) = C$ where

$$M = \frac{\partial \phi}{\partial t}, N = \frac{\partial \phi}{\partial y}$$

$$\phi(t, y) = \int Mdt + \int \left(N - \int \frac{\partial M}{\partial y} dt \right) dy$$

Integrating Factors

An integrating factor μ for the differential equation

$$M(t, y)dt + N(t, y)dy = 0$$

satisfies

$$\frac{\partial(\mu(t, y)M(t, y))}{\partial y} = \frac{\partial(\mu(t, y)N(t, y))}{\partial t}$$

If

$$P(t, y) = \frac{M_y - N_t}{N}$$

is only a function of t (and not of y) then $\mu(t) = e^{\int P(t)dt}$ is an integrating factor. If

$$Q(t, y) = \frac{N_t - M_y}{M}$$

is only a function of y (and not of t) then $\mu(t) = e^{\int Q(t)dt}$ is an integrating factor.

Homogeneous Equations

An equation is homogeneous if has the form

$$y' = f(y/t)$$

To solve a homogeneous equation, make the substitution $y = tz$ and rearrange the equation; the result is separable:

$$\frac{dz}{F(z) - z} = \frac{dt}{t}$$

Bernoulli Equations

A Bernoulli equation has the form

$$y'(t) + p(t)y = q(t)y^n$$

for some number n . To solve a Bernoulli equation, make the substitution

$$u = y^{1-n}$$

The resulting equation is linear and

$$y(t) = \left[\frac{1}{\mu} \left(C + \int \mu(t)(1-n)q(t)dt \right) \right]^{1/(1-n)}$$

where

$$\mu(t) = \exp \left((1-n) \int p(t)dt \right)$$

Second Order Homogeneous Linear Equation with Constant Coefficients

To solve the differential equation

$$ay'' + by' + cy = 0$$

find the roots of the characteristic equation

$$ar^2 + br + c = 0$$

If the roots (real or complex) are distinct, then

$$y = Ae^{r_1t} + Be^{r_2t}$$

If the roots are repeated then

$$y = (A + Bt)e^{rt}$$

Method of Undetermined Coefficients

To solve the differential equation

$$ay'' + by' + cy = f(t)$$

where $f(t)$ is a polynomial, exponential, or trigonometric function, or any product thereof, the solution is

$$y = y_H + y_P$$

where y_H is the complete solution of the homogeneous equation

$$ay'' + by' + cy = 0$$

To find y_P make an educated guess based on the form of $f(t)$. The educated guess should be the product

$$y_P = P(t)S(t)E(t)$$

where $P(t)$ is a polynomial of the same order as in $f(t)$. $S(t) = r^n(A \sin rt + B \cos rt)$ is present only if there are trig functions in rt in $f(t)$, and n is the multiplicity of r as a root of the characteristic equation ($n = 0$ if r is not a root). $E(t) = r^n e^{rt}$ is present only if there is an exponential in rt in $f(t)$. If $f(t) = f_1(t) + f_2(t) + \dots$ then solve each of the equations

$$ay'' + by' + cy = f_i(t)$$

separately and add all of the particular solutions together to get the complete particular solution.

General Non-homogeneous Linear Equation with Constant Coefficients

To solve

$$ay'' + by' + cy = f(t)$$

where a, b, c are constants for a general function $f(t)$, the solution is

$$y = Ae^{r_1 t} + Be^{r_2 t} \int_t e^{r_2 - r_1 s} ds + \frac{e^{r_1 t}}{a} \int_t e^{r_2 - r_1 s} \int_s e^{-r_2 u} f(u) du ds$$

where r_1 and r_2 are the roots of $ar^2 + br + c = 0$.

An alternative method is to factor the equation into the form

$$(D - r_1)(D - r_2)y = f(t)$$

and make the substitution

$$z = (D - r_2)y$$

This reduces the second order equation in y to a first order linear equation in z . Solve the equation

$$(D - r_1)z = f(t)$$

for z , then solve the equation

$$(D - r_2)y = z$$

for y once z is known.

Method of Reduction of Order

If one solution y_1 is known for the differential equation

$$y'' + p(t)y' + q(t)y = 0$$

then a second solution is given by

$$y_2(t) = y_1(t) \int \frac{W(y_1, y_2)(t)}{y_1(t)^2} dt$$

where the Wronskian is given by Abel's formula

$$W(y_1, y_2)(t) = C \exp\left(-\int p(s) ds\right)$$

Method of Variation of Parameters

To find a particular solution to

$$y'' + p(t)y' + q(t)y = r(t)$$

when a pair of linearly independent solutions to the homogeneous equation

$$y'' + p(t)y' + q(t)y = 0$$

are already known,

$$y_p = -y_1(t) \int_t \frac{y_2(s)r(s)}{W(y_1, y_2)(s)} ds + y_2(t) \int_t \frac{y_1(s)r(s)}{W(y_1, y_2)(s)} ds$$

Power Series Solution

To solve

$$y'' + p(t)y' + q(t)y = g(t)$$

expand y , p , q and g in power series about ordinary (non-singular) points and determine the coefficients by applying linear independence to the powers of t .

To solve

$$a(t)y'' + b(t)y' + c(t)y = g(t)$$

about a point t_0 where $a(t) = 0$ but the limits $b(t)/a(t)$ and $c(t)/a(t)$ exist as $t \rightarrow 0$ (a regular singularity), solve the indicial equation

$$r(r - 1) + rp_0 + q_0 = 0$$

for r where $p_0 = \lim_{t \rightarrow 0} b(t)/a(t)$ and $q_0 = \lim_{t \rightarrow 0} c(t)/a(t)$. Then one solution to the homogeneous equation is

$$y(t) = (t - t_0)^r \sum_{k=0}^{\infty} c_k (t - t_0)^k$$

for some unknown coefficients c_k . Determine the coefficients by linear independence of the powers of t . The second solution is found by reduction of order and the particular solution by variation of parameters.

Bibliography

- [1] Ascher, Uri M., Mattheij, Robert M. M., and Russell, Robert D. (1995) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. SIAM.
- [2] Ascher, Uri M, and Petzold, Linda R. (1998) *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM.
- [3] Bellen, Alfredo, and Zennaro, Marino (2003) *Numerical Methods for Delay Differential Equations*. Oxford.
- [4] Boyce, William E., and Diprima, Richard C. (2004) *Elementary Differential Equations*. Wiley.
- [5] Brenan, K.E., Campbell, S.L., and Petzold, L.R. (1996) *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* SIAM.
- [6] Burden, Richard L., and Faires, J. Douglas (2001) *Numerical Analysis, 7th Edition*. Brooks/Cole.
- [7] Butcher, J.C. (2003) *Numerical Methods for Ordinary Differential Equations*. Wiley.
- [8] Driver, Rodney D. (1978) *Introduction to Ordinary Differential Equations*. Harper and Row.
- [9] Gear, G. William (1971) *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall.
- [10] Hairer, E, Norsett, S. P, and Wanner, G. (1987) *Solving Ordinary Differential Equation I: Nonstiff Problems*. Springer-Verlag.
- [11] Hairer, E. and Wanner, G. (2005) *Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems. Second Edition*. Springer-Verlag.
- [12] Hurewicz, Witold (1990) *Lectures on Ordinary Differential Equations*. Dover. Reprint of 1958 MIT Press Edition.

- [13] Iserles, Arieh (1996) *A First Course in the Numerical Analysis of Differential Equations*. Cambridge.
- [14] Lambert, J.D. (1991) *Numerical Methods for Ordinary Differential Systems*. Wiley.
- [15] Seydel, Rudiger (1988) *From Equilibrium to Chaos: Practical Bifurcation and Stability Analysis*. Elsevier
- [16] Shampine, Lawrence F. (1994) *Numerical Solution of Ordinary Differential Equations*. Chapman and Hall.