uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yi Li

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Department of Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Variable-step Variable-order 3-stage Hermite-Birkhoff Ode Solver of Order 5 to 15 with a C++ Program

TITRE DE LA THÈSE / TITLE OF THESIS

Dr. Rémi Vaillancourt

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Dr. Yves Bourgault

Dr. Benoit Dionne

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

VARIABLE-STEP VARIABLE-ORDER 3-STAGE HERMITE–BIRKHOFF ODE SOLVER OF ORDER 5 TO 15 WITH A C++ PROGRAM

By
Yi Li, B.Sc.
January 2008

A Thesis
submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of
Master's of Science in Systems Science

Your file Votre référence
ISBN: 978-0-494-50901-2
Our file Notre référence
ISBN: 978-0-494-50901-2

Abstract

Variable-step variable-order 3-stage Hermite–Birkhoff (HB) methods HB(p)3 of order $p = 5$ to 15 are constructed for solving nonstiff differential equations. Forcing a Taylor expansion of the numerical solution to agree with an expansion of the true solution leads to multistep and Runge–Kutta type order conditions which are reorganized into linear confluent Vandermonde-type systems of HB type. Fast algorithms are developed for solving these systems in $O(p^2)$ operations to obtain HB interpolation polynomials in terms of generalized Lagrange basis functions. The order and stepsize of these methods are controlled by four local error estimators. These methods, when programmed in Matlab, are superior to Matlab's `ode113` in solving several problems often used to test higher order ODE solvers on the basis of the number of steps, CPU time, and maximum global error. On the other hand, HB(5-15)3 are programmed in object-oriented C++ and the Dormand–Prince 13-stage nested Runge–Kutta pair DP(8,7)13M are programmed in C. DP(8,7) is found to use less CPU time, have smaller maximum global error but require a larger number of function evaluations than HB(5-15)3. However, for expensive equations, such as the Cubicwave, HB(5-15)3 is superior. In the C++ program, array and matrix are considered to be new objects. Algorithms, testing programs and new objects are structured separately as header files.

Acknowledgements

I would like to express my deep appreciation to my supervisor, Dr. Rémi Vaillancourt, for his academic and financial support, understanding and guidance during the completion of this thesis. His constant encouragement, suggestions and ideas have been invaluable to this work.

I would also like to thank Dr. Truong Nguyen-Ba for his insightful suggestions and technical help which shaped my thesis.

Dedication

To my friends ....

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Literature Review and Motivation

From extensive computational experience accumulated over the years in solving ordinary differential equations (ODEs), it has been observed that the key to achieve high efficiency in predictor-corrector algorithms is the capacity to vary automatically not only the stepsize but also the order and hence the stepnumber of the employed methods. Algorithms with such a capacity are known as variable-step, variable-order (VSVO) algorithms.

There is a large variety of VSVO methods designed to solve nonstiff and stiff systems of first-order ODEs. This introduction intends to put some perspective among the several approaches and methods. Gear advocated a quasi-constant stepsize implementation in DIFSUB [23]. This software works with a constant stepsize until a change of stepsize is necessary or clearly advantageous. Then a continuous extension is used to get approximations to the solution at previous points in an equally spaced mesh to modify the stepsize, because constant mesh spacing is very efficient and simple to implement when solving stiff problems. Finally, in some methods, the actual mesh size is chosen by the code as done in Matlab's `ode113`. `Ode113` method is equivalent to a (Predictor-Evaluation-Corrector-Evaluation) PECE Adams formula while the DIFSUB and DASSL are equivalent to Adams–Moulton formulas. In this thesis a fully variable stepsize method is implemented where the actual mesh are chosen by

the code as for a PECE Adams formula.

A fundamental issue in the implementation of a method is the choice of the formula used and how they are implemented. A Lagrangian formulation is often used to compute the coefficients of a method efficiently. Pros and cons about different formulations are discussed by Gear for the Nordsieck form [37], Krogh for modified divided differences [28], and Brayton *et al.* [6] for Lagrangian form. The Lagrangian form has the virtue of simplicity. It should be pointed out that the CPU time and the number of iterations depends on the order of the formula. Krogh [28] was concerned about the effects of roundoffs under stringent tolerances, which is a central concern of this thesis, and he concluded that the use of divided differences is a good way to minimize the effect of roundoffs. Working with this form does involve manipulation of vectors of the size equal to the number of equations. These manipulations can be largely vectorized nowadays. Sofroniou and Spaletta [44] have carefully used vector operations to develop extrapolation solvers that might be used to achieve extraordinary accuracies in Mathematica.

The code DVDQ [29] and `ode113` [42, 2] implement Adams–Bashforth–Moulton multistep methods in PECE modes. On the other hand, DIFSUB and DASSL implement Adams–Moulton formulas. Although these codes predict with Adams–Bashforth formulas, they iterate to completion so that, in principle, it does not matter what predictor was used. Extrapolation is another method to achieve high accuracy. Deuflhard [15] is responsible for drawing attention to the value of this approach, but the codes of [25, Section II.9] are probably responsible for spreading its use. Indeed, NDSolve of Mathematica is based on those codes. Hairer *et al.* [25] consider extrapolation to be the best way of solving problems very accurately. It is the reason that extrapolation is used in Mathematica—they want to provide users with the accuracy they require. Extrapolation can be viewed as a variable-order Runge–Kuttta scheme. Following an important comparison of Enright and Hull [19] on fixed order Runge–Kutta codes and extrapolation codes, Shampine and Baca [40] argue that RK of a fixed order (7,8) pair is more efficient than extrapolation at accuracies common in scientific computation. Finally, Taylor series can be very attractive in VSVO implementation. It has been an excellent choice in astronomical calculations [3]. For general introduction one can

read the work of Corliss and Chang [13]. Lastly, an interpolant [18, 12] for approximating the solution between mesh points deserve important consideration depending on the global smoothness desired. It is worth remarking that a fundamental element in the development of the Matlab ODE Suite was to provide solvers with an event location facility. An event occurs when the value of a function of dependent variables or their derivatives increases and decreases through a given level. Indeed, this is the main reason why the Suite does not contain an extrapolation code nor a high order Runge–Kutta pair. In retrospect, one should acknowledge that Fehlberg's (7,8) pair is the one that drew attention to the value of high order RK pair.

General linear methods for solving nonstiff systems of first-order ordinary differential equations of the form

$$y' = f(x, y), \qquad y(x_0) = y_0, \tag{1}$$

can be seen as multistep methods with off-step points or Runge–Kutta methods with backstep points. General linear methods, like multistep methods, use information prior to the last step, and, like Runge–Kutta methods, they use derivative evaluations at points partway through the current step. The link between the two types of methods is that values at off-step points are obtained by means of predictors which use values at previous points. It has been noted in [9] that general linear methods incorporate function evaluations at off-step points in order to reduce the number of backsteps without lowering the order. It was found experimentally that increasing the number of backstep points is more efficient in increasing the accuracy of HB methods than increasing the number of off-step points. It was also found that increased speed is generally achieved by higher order HB methods.

HB methods of order 9, 10 and 11 have been studied in [32]. An efficient HB Obrechkoff 3-stage 6-step method of order 14 using $(d/dx)f(x, y(x))$ has been studied in [33]. Three and four stage HBO methods are found in [36, 45] and [34], respectively.

In order to solve relatively expensive nonstiff ordinary differential equations, we will formulate a new 3-stage variable-step variable-order (VSVO) general linear methods of order $p = 5, 6, \ldots, 15$ which use the values y_n, y_{n-1} and f_{n-j} for $j = 0, 1, \ldots, p-$

4 and program the algorithm in the interpreter language Matlab and an object-oriented language C++. Since these methods use Hermite–Birkhoff (HB) interpolation polynomials, they will be called HB(p)3 methods and the family of such methods will be designated by HB(5-15)3. We deal here only with algorithms for nonstiff initial value problems.

1.2 Thesis Objective

The objectives of this thesis are:

- To formulate algorithms for the HB(p)3 method for non-stiff ordinary differential equations;
- To fully derive the order condition for the HB(p)3 method based on the formulation of the algorithm;
- To program the algorithm using the Matlab and C++ programming languages;
- To compare the performance of the proposed algorithm HB(p)3 with ODE113 in Matlab and with DP(8,7)13M in C++.

1.3 Thesis Organization

In Chapter 2 we will present the background and a review of the literature on numerical methods. In Chapter 3, we will introduce a new family of general HB(p)3 methods of order $p = 5, 6, \ldots, 15$. Order conditions are listed in Section 3.2. In Section 3.3 general HB(p)3 are represented in terms of Vandermonde-type systems. In Section 3.4 elementary matrices are constructed symbolically as functions of the parameters of the methods in view of factoring the coefficient matrices of Vandermonde-type systems. In Section 3.5 a family of particular variable-step HB(5-15)3 is defined by fixing the off-step points. Fast solutions of Vandermonde-type systems are considered in Section 3.6. The implementation of HB(5-15)3 and numerical results are found in Chapter 4. Section 4.1 considers the regions of absolute stability and principal local

truncation coefficients of constant step HB(5-15)3. Section 4.2 deals with the step and order control. Numerical testing of HB(5-15)3 programmed in Matlab and C++ is done in Sections 4.3–4.4, respectively, according to three criteria. Section 4.5 briefly describe the C++ program used for the testing. Appendix A lists the algorithms. Appendix B describes the Matlab programming for Matlab users.

The comparison of the performance is made between HB(5-15)3 and ODE113 in Matlab, and between HB(5-15)3 and DP(8,7) in C++. The performance of HB(5-15)3 and DP(8,7)13M is compared on several problems often used to test higher order ODE solvers in [35].

1.4 Thesis Contribution

The contribution of this thesis includes the following items:

- The derivation of the order conditions from the formulation of the algorithm for the HB(5-15)3 method;
- The implementation of HB(5-15)3 in Matlab and C++, and DP(8,7) in C++, including the regions of absolute stability, principal error terms, step control and order control;
- A comparison of the numerical results between HB(5-15)3 and `ODE113` in Matlab, and HB(5-15)3 and DP(8,7) in C++;
- The writing up of a 4509-line C++ programming code for HB(5-15) (available on demand and in `www.site.uottawa.ca/~remi/YILIprogram`).

Chapter 2

Background and Notation

2.1 Numerical Methods

In this section, different concepts related to differential equations will be briefly reviewed. First-order systems are fundamental because higher-order systems can be reduced to first-order systems and since most computer programs solves first order systems. The general solution and particular solution of linear systems with constant coefficients will also be illustrated. Furthermore, I will give the formula for the Lagrange interpolating polynomials.

First-Order Systems

First-order system of ordinary differential equations can be written in vector form as $y' = f(x, y)$, where $y = [y^1, y^2, \ldots, y^m]^T$ and $f = [f^1, f^2, \ldots, f^m]^T$. If each f^t $(t = 1, 2, \ldots, m)$ depends on $y^1, y^2, \ldots, y^m$, the system is coupled.

Before trying to solve a differential equation numerically, we need to know that the solution exists and is unique. This is provided by the following theorem.

If $f(x, y)$ is continuous in $x \in \mathbb{R}$ and Lipschitz continuous in $y \in \mathbb{R}^m$ in a region $D = [a, b] \times \mathbb{R}^m$, that is, $\|f(x, y) - f(x, y^*)\| \leq L\|y - y^*\|$ for every (x, y) and (x, y^*) in D, then there exists a unique solution $y(x)$ of the system.

If $f(x, y)$ is differentiable with respect to y, then it is Lipschitz continuous in y, since we have $f(x, y) - f(x, y^*) = J(x, \zeta)(y - y^*)$, where $J(x, \zeta)$ is the Jacobian of f with respect to y. Here ζ_i, is on the line segment from (x, y) to (x, y^*) and the ith

row of the Jacobian is evaluated at ζ_i (see [31, p. 6]).

The general solution of such system of m equations contains m arbitrary constants. A unique solution is obtained if we impose m initial conditions. Then the system together with these conditions constitutes an initial value problem. The system in vector form is denoted by $y' = f(x, y)$, $y(a) = \eta$, where $\eta = [\eta^1, \eta^2, \ldots, \eta^m]^T$.

High-order Systems

When we seek the numerical solutions of a qth-order initial value equation, it is standard to reduce it to a system of first-order equation.

Following Lambert's notation [31, p. 7], which is needed in the program used in this thesis, a qth-order m-dimensional system of ordinary differential equations in the form of

$$y^{(q)} = \varphi(x, y^{(0)}, y^{(1)}, \ldots, y^{(q-1)}) \tag{2}$$

can be written as a first-order system of dimension qm by the following substitutions:

$$\begin{aligned} Y_1 &:= y && (\equiv y^{(0)}) \\ Y_2 &:= Y_1' && (\equiv y^{(1)}) \\ &\vdots \\ Y_q &:= Y_{q-1}' && (\equiv y^{(q-1)}) \end{aligned}$$

to get the system of first-order equations:

$$\begin{aligned} Y_1' &:= Y_2 \\ Y_2' &:= Y_3 \\ &\vdots \\ Y_{q-1}' &:= Y_q \\ Y_q' &= \varphi(x, Y_1, Y_2, \ldots, Y_q). \end{aligned}$$

The initial value problem (2) together with the initial conditions $y^{(r)} = \eta_{r+1}$ for $r = 0, 1, \ldots, q-1$ can thus be written in the form

$$Y' = F(x, Y), \qquad Y(a) = \chi,$$

where $\chi := [\eta_1, \eta_2, \ldots, \eta_q]^T$.

Linear Systems with Constant Coefficients

A first-order system $y' = f(x, y)$ is said to be linear and non-homogeneous with constant coefficients if $f(x, y) = Ay + \varphi(x)$, where A is a constant $m \times m$ matrix. The general solution (in complex form) of such a system is

$$y(x) = \sum_{t=1}^{m} \kappa_t \, e^{\lambda_t x} c_t + \psi(x),$$

where $\lambda_t \in \mathbb{R}$ or $\mathbb{C}$ and $c_t \in \mathbb{R}^m$ or $\mathbb{C}^m$ are eigenvalues and eigenvectors or generalized eigenvectors of A, respectively, κ_t are arbitrary constants, and $\psi(x)$ is a particular solution.

In the stability analysis of numerical methods for ODE's, we need to consider the solutions of finite difference equations. A k-th order system of linear difference equations with constant coefficients is of the form

$$\sum_{j=1}^{k} \gamma_j y_{n+j} = \varphi_n, \qquad n = n_0, n_0 + 1, \ldots,$$

where γ_j are scalar constants. The solution of such a system is a sequence y_n of vectors. In particular, if r_1 is a zero of multiplicity μ of the characteristic equation

$$\sum_{j=1}^{k} \gamma_j r^j = 0,$$

then

$$y_n = \sum_{j=1}^{\mu} n^{j-1} {r_1}^n d_{1j} + \sum_{t=\mu+1}^{k} {r_t}^n d_t + \psi_n,$$

where d_{1j} and d_t are arbitrary vectors and ψ_n is a particular solution.

The difference equation associated with a numerical method for ODE's is obtained by applying this method to the test equation $y' = e^{\lambda x}$. The stability of this method requires that the roots of the characteristic equation all have moduli smaller or equal to 1 and multiple zeros have moduli strictly smaller than one. This is called the "root condition". This condition plays a central role in the stability of ODE solvers.

Although there are many examples where exact solutions are known, the instances where we can find an exact solution are very limited. It is necessary to resort to either approximate or numerical solutions.

The Lagrange interpolation formula

There exists a unique polynomial of degree at most q, $I_q(x)$, which interpolates $q+1$ distinct data points (x_{n-j}, F_{n-j}), $j = 0, 1, \ldots, q$. We are interested in the representation of a polynomial $I_q(x)$ to compute $I_{q+1}(x)$ from $I_q(x)$ easily.

An easy interpolation formula uses the Lagrange basis

$$L_{q,j}(x) = \prod_{i=0, i\neq j}^{q} \frac{x - x_{n-i}}{x_{n-j} - x_{n-i}},$$

where $L_{q,j}(x)$ is a polynomial in x of degree q, such that

$$L_{q,j}(x_{n-i}) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{otherwise,} \end{cases} \qquad i = 1, 2, \ldots, q.$$

The Lagrange interpolation polynomial $I_q(x)$ can be written in the form

$$I_q(x) = \sum_{j=0}^{q} L_{q,j}(x) F_{n-j}.$$

The Lagrange formula is simple, but it suffers from a serious disadvantage. When we need to add a further data point (x_{n-q-1}, F_{n-q-1}), with q replaced by $q+1$, we get an expression for $I_{q+1}(x)$, but there is no easy way we can generate $I_{q+1}(x)$ directly from $I_q(x)$.

In this thesis, a generalized form of Lagrange interpolation polynomials is used. More details will be presented in Section 3.6.2

2.2 Multistep Methods

Linear k-step methods are of the form

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j},$$

where α_j and β_j are constants subject to the conditions $\alpha_k = 1$ and $|\alpha_0| + \beta_0| \neq 0$. Adams multistep methods are of the form

$$y_{n+k} - y_{n+k-1} = h \sum_{j=0}^{k} \beta_j f_{n+j}.$$

Adams methods of order 1 to 13 or 14 are frequently used; beyond order 14, they lack stability. The region of absolute stability shrinks as the order increases. Since the accuracy limits the stepsize at low order, the low order multistep methods are often replaced by predictor-corrector methods with larger regions of absolute stability.

The algorithm of multistep methods starts with the one-step pair for a given tolerance with no additional starting values required. The stepsize and order are allowed to change. This ends up with the order building up rapidly over the initial few steps.

Simple error estimators are needed to monitor the stepsize and order of VSVO multistep methods. Given that the order is accepted, let $E_{k,n+1}$ denote the norm of the local error estimate at x_{n+1}, where k is the order of the method and let τ be a user-defined tolerance. If $E_{k,n+1} \leq \tau$, the step is be accepted. Otherwise the step will be redone with a smaller step size. Other predictors will monitor the order.

2.3 Runge–Kutta Methods

The technique for deriving Runge–Kutta Methods consists in matching the Taylor expansion of y_{n+1} with the Taylor expansion of the exact solution $y(x_{n+1})$.

The 3-stage explicit Runge–Kutta Method is of the form:

$$\begin{aligned} y_{n+1} &= y_n + h(b_1 k_1 + b_2 k_2 + b_3 k_3), \\ k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + hc_2, y_n + hc_2 k_1), \\ k_3 &= f(x_n + hc_3, y_n + ha_{31} k_1 + ha_{32} k_2). \end{aligned} \tag{3}$$

where the usual simplifying assumptions $c_i = \sum_j a_{ij}$ are used; Thus $a_{31} = c_3 - a_{32}$.

The Taylor expansion of $y(x_{n+1})$ is

$$y(x_{n+1}) = y(x_n) + hf + \frac{1}{2}h^2F + \frac{1}{6}h^3(Ff_y + G) + O(h^4), \tag{4}$$

where the elementary differentials F and G are $F = f_x + k_1 f_y$ and $G = f_{xx} + 2k_1 f_{xy} + k_1^2 f_{yy}$, also denoted by $\{f\}$ and $\{f^2\}$ in the context of Butcher's rooted trees [31, p. 153–154, 166].

We denote by $\widetilde{y}_{n+1}$ the value of y at x_{n+1} generated by the Runge–Kutta method. In order to use the local truncation error $y(x_{n+1}) - y_{n+1} = T_{n+1}$, we need an expansion for y_{n+1} similar to (4). Expanding the k_i given by (3), we have $k_1 = f$ and

$$k_2 = f + hc_2(f_x + k_1 f_y) + \frac{1}{2}h^2 {c_2}^2(f_{xx} + 2k_1 f_{xy} + {k_1}^2 f_{yy}) + O(h^3). \tag{5}$$

We rewrite k_2 in the form

$$k_2 = f + hc_2F + \frac{1}{2}h^2 {c_2}^2 G + O(h^3). \tag{6}$$

If we substitute the expansion for k_1 and k_2 into k_3 and expand, we obtain

$$\begin{aligned} k_3 = f &+ h\{c_3 f_x + [(c_3 - a_{32})k_1 + a_{32}k_2]f_y\} \\ &+ \frac{1}{2}h^2\{{c_3}^2 f_{xx} + 2c_3[(c_3 - a_{32})k_1 + a_{32}k_2]f_{xy} + [(c_3 - a_{32})k_1 + a_{32}k_2]^2 f_{yy}\} \\ &+ O(h^3) \\ &= f + hc_3F + h^2(c_2 a_{32} F f_y + \frac{1}{2}{c_3}^2 G) + O(h^3). \end{aligned} \tag{7}$$

On substituting (6) and (7) into (3) and using the localizing assumption, which stipulates that past values are exact, we obtain the following expansion for y_{n+1}:

$$\begin{aligned} y_{n+1} = y(x_n) &+ h(b_1 + b_2 + b_3)f + h^2(b_2c_2 + b_3c_3)F \\ &+ \frac{1}{2}h^3[2b_3c_2a_{32}Ff_y + (b_2{c_2}^2 + b_3{c_3}^2)G] + O(h^4). \end{aligned} \tag{8}$$

Then we need to match the expansions of (4) and (8).

One-Stage.

When we set $b_2 = b_3 = 0$, (3) is a one-stage method. Then (8) reduces to

$$\widetilde{y}_{n+1} = y(x_n) + hb_1 f + O(h^4).$$

Matching with (4) yields $b_1 = 1$, then $T_{n+1} = O(h^2)$. There exists only one explicit one-stage Runge–Kutta method of order 1, namely Euler's Rule.

Two-Stage.

When we set $b_3 = 0$, we have a two-stage method and (8) becomes

$$\widetilde{y}_{n+1} = y(x_n) + h(b_1 + b_2)f + h^2 b_2 c_2 F + \frac{1}{2} h^3 b_2 {c_2}^2 G + O(h^4).$$

If we match the coefficients of this expression with those of (4), we have $b_1 + b_2 = 1$ and $b_2 c_2 = 1/2$. The result is a Runge–Kutta method of order 2.

Because the two equations contains three unknowns, there exists a one-parameter family of solutions of explicit two-stage Runge–Kutta methods of order 2. No member of this family can achieve order higher than 2.

Three-Stage. We can achieve order 3 if we satisfy the following order conditions:

$$\begin{aligned} b_1 + b_2 + b_3 &= 1, \\ b_2 c_2 + b_3 c_3 &= \frac{1}{2}, \\ b_2 {c_2}^2 + b_3 {c_3}^2 &= \frac{1}{3}, \\ b_3 a_{32} c_2 &= \frac{1}{6}. \end{aligned} \tag{9}$$

That results from matching (4) and (8) up to $O(h^4)$. Since those four equations have six unknowns, there exists a two-parameter family of solutions. No member of this family can achieve order higher than three.

The four elementary differentials of order four, defined in Table 1,

$$\{f^3\}, \{f\{f\}_2, \{{}_2f^2\}_2, \{{}_3f\}_3,$$

are used in the formulation of the principal error term in RK3.

2.4 The colon ":" notation

The colon ":" notation will be used to specify a column or row of a matrix. If $A \in \mathbb{R}^{m \times n}$, then $A(k,:)$ designates the kth row,

$$A(k,:) = [a_{k1}, a_{k2}, \ldots, a_{kn}].$$

The kth column is specified by

$$A(:,k) = \begin{bmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{mk} \end{bmatrix}.$$

If the integers p, q, and r satisfy $1 \leq p \leq q \leq n$ and $1 \leq r \leq m$, then

$$A(r, p:q) = [a_{rp}, \ldots, a_{rq}] \in \mathbb{R}^{1\times(q-p+1)}.$$

Similarly, if $1 \leq p \leq q \leq m$ and $1 \leq c \leq n$, then

$$A(p:q, c) = \begin{bmatrix} a_{pc} \\ \vdots \\ a_{qc} \end{bmatrix} \in \mathbb{R}^{q-p+1}.$$

This notation is extended to designate a rectangular part of a matrix A as $A(p:q, r:s)$ from row p to row q and column r to column s.

The colon notation is extensively used in numerical literature. See [24, pages 7–8 and 19]. It is also systematically used in Matlab.

Chapter 3

VSVO HB(5-15)3

In this chapter, following [35] we construct general and particular variable-step HB(p)3 of variable order p, and fast algorithms for solving Vandermonde-type matrices.

3.1 General Variable Step HB(p)3 of Order p

The following terminology will be useful. An HB(p)3 method is said to be a **general** variable-step HB method if its backstep and off-step points are variable parameters. If the off-step points are fixed, the method is said to be a **particular** variable-step method. If the stepsize is constant, and hence the backsteps and off-steps are fixed parameters, the method is said to be a **constant-step** method.

A general 3-stage HB(p)3 of order $p = 5, 6, \ldots, 15$ requires the following four formulas to perform the integration step from x_n to x_{n+1}, where, for simplicity, the offstep point $c_1 = 0$ is used in the summations.

A Hermite–Birkhoff polynomial of degree $p-2$ is used as predictor P_2 to obtain y_{n+c_2} to order $p-2$,

$$y_{n+c_2} = \alpha_{20} y_n + \alpha_{21} y_{n-1} + h_{n+1} \left(a_{21} f_n + \sum_{j=1}^{p-4} \beta_{2j} f_{n-j} \right). \tag{10}$$

A Hermite–Birkhoff polynomial of degree $p-1$ is used as predictor P_3 to obtain

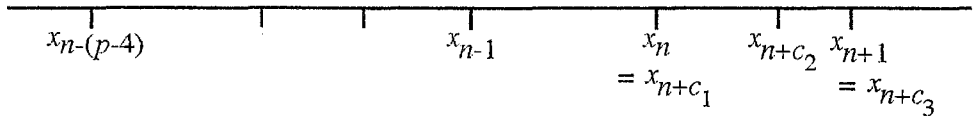

Figure 1: The $(p-3)$ backstep and 3 offstep points.

y_{n+c_3} to order $p-2$,

$$y_{n+c_3} = \alpha_{30} y_n + \alpha_{31} y_{n-1} + h_{n+1} \left(\sum_{j=1}^{2} a_{3j} f_{n+c_j} + \sum_{j=1}^{p-4} \beta_{3j} f_{n-j} \right). \tag{11}$$

A Hermite–Birkhoff polynomial of degree p is used as integration formula IF to obtain y_{n+1} to order p:

$$y_{n+1} = \alpha_0 y_n + \alpha_1 y_{n-1} + h_{n+1} \left(\sum_{j=1}^{3} b_j f_{n+c_j} + \sum_{j=1}^{p-4} \beta_j f_{n-j} \right). \tag{12}$$

An Adams–Moulton corrector of order $p-2$ is used as P_4 to control the stepsize, h_{n+2}, and obtain $\widetilde{y}_{n+1}$ to order $p-2$,

$$\widetilde{y}_{n+1} = y_n + h_{n+1} \left(a_{41} f_n + a_{43} f_{n+1} + \sum_{j=1}^{p-4} \beta_{4j} f_{n-j} \right). \tag{13}$$

For the 3-stage $(p-3)$-step methods (that is, a 3-stage Runge–Kutta method combined with a $(p-3)$-step method of order p) considered in this thesis (see Fig. 1), the off-step points satisfy the following Runge–Kutta type simplifying conditions:

$$c_i = \sum_{j=1}^{i-1} a_{ij} + B_i(1), \qquad i = 2, 3, \tag{14}$$

where

$$B_i(j) = \alpha_{i1} \frac{\eta_2^j}{j!} + \sum_{\ell=1}^{p-4} \left[\beta_{i\ell} \frac{\eta_{\ell+1}^{j-1}}{(j-1)!} \right], \qquad j = 1, 2, \ldots, p, \quad i = 2, 3. \tag{15}$$

and

$$\eta_j = -\frac{1}{h_{n+1}} (x_n - x_{n+1-j}) = -\frac{1}{h_{n+1}} \sum_{i=0}^{j-1} h_{n-i}, \qquad j = 2, 3, \ldots, k+1. \tag{16}$$

In the sequel, η_j will be frequently used without explicit reference to (16). The strategy of the 3-stage Runge–Kutta Methods is applied on P2, P3, IF and P4 to specify the coefficients.

3.2 Order Conditions for HB(p)3

By subtracting the Taylor series of the numerical solution y_1 produced by the formulae of HB(5-15)3 from the Taylor series of the true solution $y(x_0+h)$, we obtain a series whose first few terms can be written as follows:

$$
\begin{aligned}
y(x_0+h) - y_1 &= h\left(1 - [b_1 + b_2 + b_3 + B_1(1)]\right) f \\
&+ h^2 \left\{ \frac{1}{2!} - [b_2 c_2 + b_3 c_3 + B_1(2)] \right\} \{f\} \\
&+ h^3 \left\{ \frac{1}{6} - \left(\sum_{i=2}^{s} b_i \left[\sum_{j=1}^{i-1} a_{ij} c_j + B_j(2) \right] + B_1(3) \right) \right\} \{{}_2 f\}_2 \\
&+ h^3 \left\{ \frac{1}{6} - \left(b_2 \frac{{c_2}^2}{2} + b_3 \frac{{c_3}^2}{2} + B_1(3) \right) \right\} \{f^2\} \\
&+ h^4 \left\{ \frac{1}{4!} - \sum_{i=2}^{s} b_i \left[\sum_{j=1}^{i-1} a_{ij} \left[\sum_{k=1}^{j-1} a_{jk} c_k + B_j(2) \right] + B_i(3) \right] + B_1(4) \right\} \{{}_3 f\}_3 \qquad (17) \\
&+ h^4 \left\{ \frac{1}{4!} - \sum_{i=2}^{s} b_i \left[\sum_{j=1}^{i-1} a_{ij} \frac{{c_j}^2}{2!} + B_i(3) \right] + B_1(4) \right\} \{{}_2 f\}_2 \\
&+ h^4 \times 3 \times \left\{ \frac{1}{4!} - \left(\frac{1}{3} \sum_{i=2}^{s} b_i c_i \left[\sum_{j=1}^{i-1} a_{ij} c_j + B_i(2) \right] + B_1(4) \right) \right\} \{f\{f\}\} \\
&+ h^4 \left\{ \frac{1}{4!} - \left[\sum_{i=2}^{s} \frac{b_i {c_i}^3}{3!} + B_1(4) \right] \right\} \{f^3\} + \ldots,
\end{aligned}
$$

where s is the number of stages and f, $\{f\}$, $\{{}_2 f\}_2$, $\ldots$, are elementary differentials defined in [8], [25] and [31]. The elementary differentials used in this thesis are listed in Tables 1–4.

In other words, by forcing a Taylor expansion of the numerical solution produced by the formulae of HB(5-15)3 to agree with an expansion of the true solution, we obtain Runge–Kutta-type order conditions that must be satisfied by general HB(p)3 methods of order $p = 5, \ldots, 15$. (see [8]). These order conditions can be obtained from (17) (or, see [8], with added backstep expression).

Next, we impose the following simplifying assumptions:

$$\sum_{j=0}^{1} \alpha_{ij} = 1, \qquad i = 2, 3, \tag{18}$$

$$\sum_{j=1}^{i-1} a_{ij} c_j^k + k! B_i(k+1) = \frac{1}{k+1} c_i^{k+1}, \qquad \begin{cases} i = 2, 3, \\ k = 0, 1, 2, \ldots, p-3. \end{cases} \tag{19}$$

Because of these simplifying assumptions, all RK-type order conditions associated with elementary differentials of order $r = 1, 2, \ldots, p-1$ are equivalent to the order condition

$$\sum_{i=1}^{3} b_i c_i^{r-1} + (r-1)! B_1(r) = \frac{1}{r}, \tag{20}$$

associated with the elementary differentials f for $r = 1$ and $\{f^{r-1}\}$ otherwise. We note that for the simplifying assumption (20), the predictors are of order at least $p-2$. Thus we obtain the following multistep- and RK-type order conditions that must be satisfied by the general HB(p)3 method of order $p = 5, \ldots, 15$:

$$\sum_{i=0}^{1} \alpha_i = 1, \tag{21}$$

$$\sum_{i=1}^{3} b_i c_i^k + k! B_1(k+1) = \frac{1}{k+1}, \qquad k = 0, 1, \ldots, p-1, \tag{22}$$

$$\sum_{i=2}^{3} b_i \left[\sum_{j=1}^{i-1} a_{ij} \frac{c_j^{p-2}}{(p-2)!} + B_i(p-1) \right] + B_1(p) = \frac{1}{p!}, \tag{23}$$

where the backstep parts, $B_1(j)$, are defined by

$$B_1(j) = \alpha_1 \frac{\eta_2^j}{j!} + \sum_{i=1}^{p-4} \beta_i \frac{\eta_{i+1}^{j-1}}{(j-1)!}, \qquad j = 1, \ldots, p+1. \tag{24}$$

Similar order conditions have been used in [9] where Butcher has derived general linear methods analogous to RK3.

Equations (22) and (23) are derived by induction on the order p. First, setting

$p = 3$ in equation (22) and (23), we have the four conditions:

$$\begin{aligned} b_1 + b_2 + b_3 &= 1, \\ b_2 c_2 + b_3 c_3 &= \frac{1}{2}, \\ b_2 {c_2}^2 + b_3 {c_3}^2 &= \frac{1}{3}, \\ b_3 a_{32} c_2 &= \frac{1}{6}, \end{aligned}$$

which are the order conditions (9) of RK3. Thus the order condition is true for $p = 3$, since there is no backstep in this case.

Suppose that the order conditions of HB(p)3 of order p, for $p \geq 3$, are of the form:

$$\sum_{i=1}^{3} b_i c_i^k + k! B_1(k+1) = \frac{1}{k+1}, \qquad k = 0, 1, \ldots, p-1, \tag{25}$$

$$\sum_{i=2}^{3} b_i \left[\sum_{j=1}^{i-1} a_{ij} \frac{c_j^{p-2}}{(p-2)!} + B_i(p-1) \right] + B_1(p) = \frac{1}{p!}, \tag{26}$$

where the backstep parts, $B_1(j)$, are defined by

$$B_1(j) = \alpha_1 \frac{\eta_2^j}{j!} + \sum_{i=1}^{p-4} \beta_i \frac{\eta_{i+1}^{j-1}}{(j-1)!}, \qquad j = 1, \ldots, p+1. \tag{27}$$

Since all predictors are of order at least $p-2$, a method of order $p+1$ cannot have more than the four additional order conditions listed in (28) to (31). Other order conditions associated with elementary differentials of order $(p+1)$ are equivalent to these four order conditions; otherwise, the Runge–Kutta method of order four would have more than four order conditions.

The four order conditions associated with the four elementary differentials of order

$p+1$ are:

$$\sum_{i=2}^{3} b_i c_i^p + p!B_1(p+1) = \frac{1}{p+1}, \tag{28}$$

$$\sum_{i=2}^{3} b_i \frac{c_i}{p}\left[\sum_{j=1}^{i-1} a_{ij}\frac{c_j^{p-2}}{(p-2)!} + B_i(p-1)\right] + B_1(p+1) = \frac{1}{(p+1)!}, \tag{29}$$

$$\sum_{i=2}^{3} b_i \left[\sum_{j=1}^{i-1} a_{ij}\frac{c_j^{p-1}}{(p-1)!} + B_i(p)\right] + B_1(p+1) = \frac{1}{(p+1)!}, \tag{30}$$

$$\sum_{i=2}^{3} b_i \left[\sum_{j=1}^{i-1} a_{ij}\left[\sum_{k=1}^{j-1} a_{jk}\frac{c_k^{p-2}}{(p-2)!} + B_j(p-1)\right] + B_i(p)\right] + B_1(p+1) = \frac{1}{(p+1)!}. \tag{31}$$

Order conditions similar to the above four conditions have been used in [10] where Butcher derived methods which are analogous to RK4. The four elementary differentials of order $(p+1)$,

$$\{f^p\}, \quad \{\{f^{p-2}\}f\}, \quad \{{}_2f^{p-1}\}_2, \quad \{{}_3f^{p-2}\}_3$$

are used for the principal error term of HB(p)3. More details will be presented in Section 4.1.

We consider now the case where the order of all the predictors is increased to at least $(p-1)$. Equation (29) is then equivalent to equation (28) while equation (31) is equivalent to equation (30). In other words, if HB($p+1$) satisfies (28) and (30), then (29) and (31) follow as well, respectively.

Hence, the order conditions (22)–(23) and (28)–(31) will be satisfied if the following order conditions are satisfied:

$$\sum_{i=1}^{3} b_i c_i^k + k!B_1(k+1) = \frac{1}{k+1}, \qquad k = 0, 1, \ldots, p, \tag{32}$$

$$\sum_{i=2}^{3} b_i \left[\sum_{j=1}^{i-1} a_{ij}\frac{c_j^{p-1}}{(p-1)!} + B_i(p)\right] + B_1(p+1) = \frac{1}{(p+1)!}. \tag{33}$$

Hence, the order conditions (22) and (23) are proved.

We consider an example of deriving order conditions for HB(p)3 of order $p = 4$ to 6. By forcing a Taylor expansion of the numerical solution produced by the formulae

of HB of order 6 to agree with an expansion of the true solution, we obtain Runge–Kutta-type order conditions that must be satisfied by the general HB(6) method of order 6 with predictors of order at least 1. Since the assumption on the order of the predictors is that of the Runge–Kutta method, this HB(6) is simply an RK6 method with backstep expressions. These order conditions correspond to elementary differentials listed in Table 1, where $r(i)$ is the order of elementary differentials $\mathbf{F}^{(i)}$, $\alpha(i)$, $\beta(i)$ and $\gamma(i)$ defined in [8].

Consider now the case where the orders of predictors P2 and P3 are increased to at least 2. This implies that the order conditions listed in Table 1 are not independent. The order conditions corresponding to $i = 3, 5, 7, 9, 11, 13, 15, 16, 18, 20, 22, 24, 25, 27, 29, 31, 32, 33, 35, 36$ are equivalent to the order conditions corresponding to $i = 4, 6, 8, 10, 12, 14, 17, 17, 19, 21, 23, 26, 26, 28, 30, 34, 34, 34, 37, 37$, respectively, and hence can be discarded. The resulting elementary differentials of order 1 to 6 for HB with predictors of order at least 2 are listed in Table 2. In this table, the elementary differentials corresponding to $i = 1, 2, \ldots, 8$ can be used to write down the order conditions for HB(4)3.

Consider now the case where the order of predictors P2 and P3 are increased to at least 3. This implies that some order conditions listed in table 2 are not independent. The order conditions corresponding to $i = 6, 10, 14, 19, 23, 28, 34$ are equivalent to the order conditions corresponding to $i = 8, 12, 17, 21, 26, 30, 37$, respectively; hence these can be discarded. The resulting elementary differentials of orders 1 to 6 for HB with predictors of order at least 3 are listed in Table 3. In this table, the elementary differentials corresponding to $i = 1, 2, \ldots, 17$ can be used to write down the order conditions for HB(5)3.

Consider now the case where the order of predictors P2 and P3 are increased to at least 4. This implies that some order conditions listed in table 3 are not independent. The order conditions corresponding to $i = 12, 21, 30$ are equivalent to the order conditions corresponding to $i = 17, 26, 37$, respectively; hence these can be discarded. The resulting elementary differentials of order 1 to 6 for HB with predictors of order at least 4 are listed in Table 4. These elementary differentials can be used to write down the order conditions for HB(6)3.

Table 1: Elementary differentials of order 1 to 6 for HB with predictors of order ≥ 1.

i	$r(i)$	$\mathbf{F}^{(i)}$	$\phi^0(i)$	$\alpha(i)$	$\beta(i)$	$\gamma(i)$
1	1	f	$\sum_i b_i$	1	1	1
2	2	$\{f\}$	$\sum_i b_i c_i$	1	1	2
3	3	$\{_2f\}_2$	$\sum_{ij} b_i a_{ij} c_j$	1	2	6
4	3	$\{f^2\}$	$\sum_i b_i c_i^2$	1	1	3
5	4	$\{_3f\}_3$	$\sum_{ijk} b_i a_{ij} a_{jk} c_k$	1	6	24
6	4	$\{_2f^2\}_2$	$\sum_{ij} b_i a_{ij} c_j^2$	1	3	12
7	4	$\{\{f\}f\}$	$\sum_{ij} b_i c_i a_{ij} c_j$	3	6	8
8	4	$\{f^3\}$	$\sum_i b_i c_i^3$	1	1	4
9	5	$\{_4f\}_4$	$\sum_{ijkl} b_i a_{ij} a_{jk} a_{kl} c_l$	1	24	120
10	5	$\{_3f^2\}_3$	$\sum_{ijk} b_i a_{ij} a_{jk} c_k^2$	1	12	60
11	5	$\{_2\{f\}f\}_2$	$\sum_{ijk} b_i a_{ij} c_j a_{jk} c_k$	3	24	40
12	5	$\{_2f^3\}_2$	$\sum_{ij} b_i a_{ij} c_j^3$	1	4	20
13	5	$\{\{_2f\}_2f\}$	$\sum_{ijk} b_i c_i a_{ij} a_{jk} c_k$	4	24	30
14	5	$\{\{f^2\}f\}$	$\sum_{ij} b_i c_i a_{ij} c_j^2$	4	12	15
15	5	$\{\{f\}^2\}$	$\sum_i b_i (\sum_j a_{ij} c_j)^2$	3	12	20
16	5	$\{\{f\}f^2\}$	$\sum_{ij} b_i c_i^2 a_{ij} c_j$	6	12	10
17	5	$\{f^4\}$	$\sum_i b_i c_i^4$	1	1	5
18	6	$\{_5f\}_5$	$\sum_{ijklm} b_i a_{ij} a_{jk} a_{kl} a_{lm} c_m$	1	120	720
19	6	$\{_4f^2\}_4$	$\sum_{ijkl} b_i a_{ij} a_{jk} a_{kl} c_l^2$	1	60	360
20	6	$\{_3\{f\}f\}_3$	$\sum_{ijkl} b_i a_{ij} a_{jk} c_k a_{kl} c_l$	3	120	240
21	6	$\{_3f^3\}_3$	$\sum_{ijk} b_i a_{ij} a_{jk} c_k^3$	1	20	120
22	6	$\{_2\{_2f\}_2f\}_2$	$\sum_{ijkl} b_i a_{ij} c_j a_{jk} a_{kl} c_l$	4	120	180
23	6	$\{_2\{f^2\}f\}_2$	$\sum_{ijk} b_i a_{ij} c_j a_{jk} c_k^2$	4	60	90
24	6	$\{_2\{f\}^2\}_2$	$\sum_i b_i a_{ij} (\sum_k a_{jk} c_k)^2$	3	60	120
25	6	$\{_2\{f\}f^2\}_2$	$\sum_{ijk} b_i a_{ij} c_j^2 a_{jk} c_k$	6	60	60
26	6	$\{_2f^4\}_2$	$\sum_{ij} b_i a_{ij} c_j^4$	1	5	30
27	6	$\{\{_3f\}_3f\}$	$\sum_{ijkl} b_i c_i a_{ij} a_{jk} a_{kl} c_l$	5	120	144
28	6	$\{\{_2f^2\}_2f\}$	$\sum_{ijk} b_i c_i a_{ij} a_{jk} c_k^2$	5	60	72
29	6	$\{\{\{f\}f\}f\}$	$\sum_{ijk} b_i c_i a_{ij} c_j a_{jk} c_k$	15	120	48
30	6	$\{\{f^3\}f\}$	$\sum_{ij} b_i c_i a_{ij} c_j^3$	5	20	24
31	6	$\{\{_2f\}_2\{f\}\}$	$\sum_i b_i (\sum_{jk} a_{ij} a_{jk} c_k)(\sum_j a_{ij} c_j)$	10	120	72
32	6	$\{\{f^2\}\{f\}\}$	$\sum_i b_i (\sum_j a_{ij} c_j^2)(\sum_j a_{ij} c_j)$	10	60	36
33	6	$\{\{_2f\}_2f^2\}$	$\sum_{ijk} b_i c_i^2 a_{ij} a_{jk} c_k$	10	60	36
34	6	$\{\{f^2\}f^2\}$	$\sum_{ij} b_i c_i^2 a_{ij} c_j^2$	10	30	18
35	6	$\{\{f\}^2f\}$	$\sum_i b_i c_i (\sum_j a_{ij} c_j)^2$	15	60	24
36	6	$\{\{f\}f^3\}$	$\sum_{ijk} b_i c_i^3 a_{ij} c_j$	10	20	12
37	6	$\{f^5\}$	$\sum_i b_i c_i^5$	1	1	6

Table 2: Elementary differentials of order 1 to 6 for HB with predictors of order ≥ 2.

i	$r(i)$	$\mathbf{F}^{(i)}$	$\phi^0(i)$	$\alpha(i)$	$\beta(i)$	$\gamma(i)$
1	1	f	$\sum_i b_i$	1	1	1
2	2	$\{f\}$	$\sum_i b_i c_i$	1	1	2
4	3	$\{f^2\}$	$\sum_i b_i c_i^2$	1	1	3
6	4	$\{{}_2f^2\}_2$	$\sum_{ij} b_i a_{ij} c_j^2$	1	3	12
8	4	$\{f^3\}$	$\sum_i b_i c_i^3$	1	1	4
10	5	$\{{}_3f^2\}_3$	$\sum_{ijk} b_i a_{ij} a_{jk} c_k^2$	1	12	60
12	5	$\{{}_2f^3\}_2$	$\sum_{ij} b_i a_{ij} c_j^3$	1	4	20
14	5	$\{\{f^2\}f\}$	$\sum_{ij} b_i c_i a_{ij} c_j^2$	4	12	15
17	5	$\{f^4\}$	$\sum_i b_i c_i^4$	1	1	5
19	6	$\{{}_4f^2\}_4$	$\sum_{ijkl} b_i a_{ij} a_{jk} a_{kl} c_l^2$	1	60	360
21	6	$\{{}_3f^3\}_3$	$\sum_{ijk} b_i a_{ij} a_{jk} c_k^3$	1	20	120
23	6	$\{{}_2\{f^2\}f\}_2$	$\sum_{ijk} b_i a_{ij} c_j a_{jk} c_k^2$	4	60	90
26	6	$\{{}_2f^4\}_2$	$\sum_{ij} b_i a_{ij} c_j^4$	1	5	30
28	6	$\{\{{}_2f^2\}_2f\}$	$\sum_{ijk} b_i c_i a_{ij} a_{jk} c_k^2$	5	60	72
30	6	$\{\{f^3\}f\}$	$\sum_{ij} b_i c_i a_{ij} c_j^3$	5	20	24
34	6	$\{\{f^2\}f^2\}$	$\sum_{ij} b_i c_i^2 a_{ij} c_j^2$	10	30	18
37	6	$\{f^5\}$	$\sum_i b_i c_i^5$	1	1	6

Table 3: Elementary differentials of order 1 to 6 for HB with predictors of order ≥ 3.

i	$r(i)$	$\mathbf{F}^{(i)}$	$\phi^0(i)$	$\alpha(i)$	$\beta(i)$	$\gamma(i)$
1	1	f	$\sum_i b_i$	1	1	1
2	2	$\{f\}$	$\sum_i b_i c_i$	1	1	2
4	3	$\{f^2\}$	$\sum_i b_i c_i^2$	1	1	3
8	4	$\{f^3\}$	$\sum_i b_i c_i^3$	1	1	4
12	5	$\{{}_2f^3\}_2$	$\sum_{ij} b_i a_{ij} c_j^3$	1	4	20
17	5	$\{f^4\}$	$\sum_i b_i c_i^4$	1	1	5
21	6	$\{{}_3f^3\}_3$	$\sum_{ijk} b_i a_{ij} a_{jk} c_k^3$	1	20	120
26	6	$\{{}_2f^4\}_2$	$\sum_{ij} b_i a_{ij} c_j^4$	1	5	30
30	6	$\{\{f^3\}f\}$	$\sum_{ij} b_i c_i a_{ij} c_j^3$	5	20	24
37	6	$\{f^5\}$	$\sum_i b_i c_i^5$	1	1	6

Table 4: Elementary differentials of order 1 to 6 for HB with predictors of order ≥ 4.

i	$r(i)$	$\mathbf{F}^{(i)}$	$\phi^0(i)$	$\alpha(i)$	$\beta(i)$	$\gamma(i)$
1	1	f	$\sum_i b_i$	1	1	1
2	2	$\{f\}$	$\sum_i b_i c_i$	1	1	2
4	3	$\{f^2\}$	$\sum_i b_i c_i^2$	1	1	3
8	4	$\{f^3\}$	$\sum_i b_i c_i^3$	1	1	4
17	5	$\{f^4\}$	$\sum_i b_i c_i^4$	1	1	5
26	6	$\{{}_2f^4\}_2$	$\sum_{ij} b_i a_{ij} c_j^4$	1	5	30
37	6	$\{f^5\}$	$\sum_i b_i c_i^5$	1	1	6

3.3 Vandermonde-type Formulation of HB(p)3

In this section, the equations for the integration formula and the predictors are derived from their assumed forms (10)–(13) and their Taylor expansions. Moreover, the purpose of this section is not to derive a practical algorithm to solve these equations in terms of all the parameters since the c_j appear nonlinearly. Later, for particular variable-step HB(p) methods, the c_j will be fixed and a flow chart outlining the order in which the parameters are obtained will be given in section 3.5.

3.3.1 Integration formula IF

To find the IF coefficients which satisfy the order condition (22), we consider the $(p+1)$-vector of reordered coefficients of the integration formula IF in (12),

$$\boldsymbol{u}^1 = [\alpha_0, b_1, b_2, b_3, \beta_1, \beta_2, \ldots, \beta_{p-4}, \alpha_1]^T,$$

which is the solution of the confluent Vandermonde-type system of order conditions (21) and (22):

$$M^1 \boldsymbol{u}^1 = \boldsymbol{r}^1, \tag{34}$$

where

$$M^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & \cdots & 1 & \eta_2 \\ 0 & 0 & c_2 & c_3 & \eta_2 & \cdots & \eta_{p-3} & \frac{\eta_2^2}{2!} \\ 0 & 0 & \frac{c_2^2}{2!} & \frac{c_3^2}{2!} & \frac{\eta_2^2}{2!} & \cdots & \frac{\eta_{p-3}^2}{2!} & \frac{\eta_2^3}{3!} \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \frac{c_2^{p-1}}{(p-1)!} & \frac{c_3^{p-1}}{(p-1)!} & \frac{\eta_2^{p-1}}{(p-1)!} & \cdots & \frac{\eta_{p-3}^{p-1}}{(p-1)!} & \frac{\eta_2^p}{p!} \end{bmatrix}, \tag{35}$$

and $\boldsymbol{r}^1 = r_1(1:p+1)$ has components

$$r_1(i) = 1/(i-1)!, \qquad i = 1, 2, \ldots, p+1.$$

The leading error term of IF, which is the first deleted coefficient in the Taylor expansion of y_{n+1}, is

$$\left[\alpha_1 \frac{\eta_2^{p+1}}{(p+1)!} + b_2 \frac{c_2^p}{p!} + b_3 \frac{c_3^p}{p!} + \sum_{j=1}^{p-4} \beta_j \frac{\eta_{j+1}^p}{p!} - \frac{1}{(p+1)!}\right] h_{n+1}^{p+1} y_n^{p+1}.$$

3.3.2 Predictor P_2

The $(p-1)$-vector of reordered coefficients of predictor P_2 in (10),

$$\boldsymbol{u}^2 = [\alpha_{20}, a_{21}, \beta_{21}, \beta_{22}, \ldots, \beta_{2,p-4}, \alpha_{21}]^T,$$

is the solution of the system of order conditions (19):

$$M^2 \boldsymbol{u}^2 = \boldsymbol{r}^2, \tag{36}$$

where

$$M^2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 1 & \cdots & 1 & \eta_2 \\ 0 & 0 & \eta_2 & \cdots & \eta_{p-3} & \frac{\eta_2^2}{2!} \\ 0 & 0 & \frac{\eta_2^2}{2!} & \cdots & \frac{\eta_{p-3}^2}{2!} & \frac{\eta_2^3}{3!} \\ \vdots & & & & & \vdots \\ 0 & 0 & \frac{\eta_2^{p-3}}{(p-3)!} & \cdots & \frac{\eta_{p-3}^{p-3}}{(p-3)!} & \frac{\eta_2^{p-2}}{(p-2)!} \end{bmatrix} \tag{37}$$

and $\boldsymbol{r}^2 = r_2(1:p-1)$ has components

$$r_2(i) = c_2^{i-1}/(i-1)!, \qquad i = 1, 2, \ldots, p-1.$$

A truncated Taylor expansion of the right-hand side of (10) about x_n gives

$$\sum_{j=0}^{p+1} S_2(j) h_{n+1}^j y_n^{(j)}$$

with coefficients

$$S_2(j) = M^2(j+1, 1, \ldots, p-1)\boldsymbol{u}^2 = r_2(j+1) = \frac{c_2^j}{j!}, \qquad j = 0, 1, \ldots, p-2,$$

$$S_2(j) = \alpha_{21}\frac{\eta_2^j}{j!} + \sum_{i=1}^{p-4} \beta_{2i}\frac{\eta_{i+1}^{j-1}}{(j-1)!}, \qquad j = p-1, p, p+1.$$

We note that P_2 is of order $p-2$ since it satisfies the order conditions

$$\sum_{i=0}^{1} \alpha_{2i} = 1,$$

$$S_2(j) = c_2^j/j!, \qquad j = 1, \ldots, p-2,$$

and its leading error term is

$$\left[S_2(p-1) - \frac{c_2^{p-1}}{(p-1)!}\right] h_{n+1}^{p-1} y_n^{(p-1)}.$$

3.3.3 Predictor P_3

The p-vector of reordered coefficients of predictor P_3 in (11),

$$\boldsymbol{u}^3 = [\alpha_{30}, a_{31}, a_{32}, \beta_{31}, \beta_{32}, \ldots, \beta_{3,p-4}, \alpha_{31}]^T,$$

is the solution of the system of order conditions (18) and (19) with $i = 3$, and (23):

$$M^3\boldsymbol{u}^3 = \boldsymbol{r}^3, \tag{38}$$

where

$$M^3 = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 1 & 1 & \cdots & 1 & \eta_2 \\ 0 & 0 & c_2 & \eta_2 & \cdots & \eta_{p-3} & \frac{\eta_2^2}{2!} \\ 0 & 0 & \frac{c_2^2}{2!} & \frac{\eta_2^2}{2!} & \cdots & \frac{\eta_{p-3}^2}{2!} & \frac{\eta_2^3}{3!} \\ \vdots & & & & & & \vdots \\ 0 & 0 & \frac{c_2^{p-2}}{(p-2)!} & \frac{\eta_2^{p-2}}{(p-2)!} & \cdots & \frac{\eta_{p-3}^{p-2}}{(p-2)!} & \frac{\eta_2^{p-1}}{(p-1)!} \end{bmatrix}. \tag{39}$$

The first $p-1$ components of $\boldsymbol{r}^3 = r_3(1:p)$ are

$$r_3(i) = c_3^{i-1}/(i-1)!, \qquad i = 1, 2, \ldots, p-1,$$

and the pth component is

$$r_3(p) = \frac{1}{b_{13}} \left[\frac{1}{p!} - b_{12} S_2(p-1) - B_1(p) \right].$$

A truncated Taylor expansion of the right-hand side of (11) about x_n gives

$$\sum_{j=0}^{p+1} S_3(j) h_{n+1}^j y_n^{(j)}$$

with coefficients

$$S_3(j) = M^3(j+1, 1, \ldots, p)\boldsymbol{u}^3 = r_3(j+1) = \frac{c_3^j}{j!}, \qquad j = 0, 1, \ldots, p-2,$$

$$S_3(j) = \alpha_{31} \frac{\eta_2^j}{j!} + a_{32} S_2(j-1) + \sum_{i=1}^{p-4} \beta_{3i} \frac{\eta_{i+1}^{j-1}}{(j-1)!}, \qquad j = p-1, p, p+1.$$

3.3.4 Step control predictor P_4

The $(p-2)$-vector of reordered coefficients of P_4 in (13),

$$\boldsymbol{u}^4 = [a_{41}, a_{43}, \beta_{41}, \beta_{42}, \ldots, \beta_{4,p-4}]^T,$$

is the solution of the system of order conditions:

$$M^4 \boldsymbol{u}^4 = \boldsymbol{r}^4, \tag{40}$$

where

$$M^4 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & c_3 & \eta_2 & \cdots & \eta_{p-3} \\ 0 & \frac{c_3^2}{2!} & \frac{\eta_2^2}{2!} & \cdots & \frac{\eta_{p-3}^2}{2!} \\ \vdots & & & & \vdots \\ 0 & \frac{c_3^{p-3}}{(p-3)!} & \frac{\eta_2^{p-3}}{(p-3)!} & \cdots & \frac{\eta_{p-3}^{p-3}}{(p-3)!} \end{bmatrix} \tag{41}$$

and $\boldsymbol{r}^4 = r_4(1:p-2)$ has components

$$r_4(i) = 1/i!, \qquad i = 1, 2, \ldots, p-2.$$

The solutions $\boldsymbol{u}^\ell$, $\ell = 1, 2, 3, 4$, form generalized Lagrange basis functions for representing the HB interpolation polynomials.

3.4 Symbolic Construction of Elementary Matrices

Consider the matrices

$$M^\ell \in \mathbb{R}^{m_\ell \times m_\ell}, \qquad \ell = 1, 2, 3, 4, \tag{42}$$

of the Vandermonde-type systems (34), (36), (38), and (40), where

$$m_1 = p+1, \quad m_2 = p-1, \quad m_3 = p, \quad m_4 = p-2, \tag{43}$$

and p is the order of the method.

The purpose of this section is to construct symbolically elementary lower and upper bidiagonal matrices as functions of the parameters of HP(p). These functions will be used in Subsection 3.4.1 to factor each M^ℓ, $\ell = 1, 2, 3$, into a diagonal+last-column matrix, W^ℓ, which will be further diagonalized by a Gaussian elimination. This decomposition will lead to a fast solution of the systems $M^\ell \boldsymbol{u}^\ell = \boldsymbol{r}^\ell$ in $O(p^2)$ operations.

Since the Vandermonde-type matrices M^ℓ can be decomposed into the product of a diagonal matrix containing reciprocals of factorials and a confluent Vandermonde matrix, the factorizations used in this thesis hold following the approach of Björck

and Pereyra [5], Krogh [28], Galimberti and Pereyra [21] and Björck and Elfving [4]. Pivoting is not needed in this decomposition because of the special structure of Vandermonde-type matrices. The competitiveness of HB(p)3 of order p comes from the surprisingly stable fast solution of ill-conditioned Vandermonde-type linear systems in $O(p^2)$ operations (see [24, p. 187]) by taking the special structure of these systems into account, in contrast with Gaussian elimination with pivoting in $O(p^3)$ operations, which requires more storage, is slower and, at times, leads to unstable solutions when the stepsize is very small.

3.4.1 Construction of lower bidiagonal matrices

We first describe the zeroing process of a general vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$ with no zero elements. The lower bidiagonal matrix

$$L_k = \begin{bmatrix} I_{k-1} & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & & 0 \\ 0 & 1 & -\tau_{k+1} & & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & -\tau_m \end{bmatrix} \tag{44}$$

defined by the multipliers

$$\tau_i = \frac{x_{i-1}}{x_i} = -L_k(i,i), \qquad i = k+1, k+2, \ldots, m, \tag{45}$$

zeros the last $(m-k)$ components, $x_{k+1}, \ldots, x_m$, of $\boldsymbol{x}$. This zeroing process will be applied recursively on M^ℓ as follows.

For $k = 3, 4, \ldots, m_\ell - 1$, left multiplying $T_k^\ell = L_{k-1}^\ell \cdots L_4^\ell L_3^\ell M^\ell$ by L_k^ℓ zeros the last $(m_l - k)$ components of the kth column of T_k^ℓ. Thus we obtain the upper triangular matrix

$$L^\ell M^\ell = L_{m_\ell - 1}^\ell \cdots L_4^\ell L_3^\ell M^\ell \tag{46}$$

in $(m_\ell - 3)$ matrix operations. We note that L^ℓ does not change the first two rows of M^ℓ.

Process 1 *At the kth step, starting with* $k = 3$,

- $M^{\ell(k-1)} = L^{\ell}_{k-1}L^{\ell}_{k-2}\cdots L^{\ell}_{3}M^{\ell}$ *is an upper triangular matrix in columns* 1 *to* $k-1$.
- *The multipliers in* L^{ℓ}_{k} *are obtained from* $M^{\ell(k-1)}(k+1,\ldots,m_l,k)$ *since* $M^{\ell}(i,k) \neq 0$ *for* $i = k+1, k+2, \ldots, m_l$.

Algorithm 1 in Appendix A describes this process.

3.4.2 Construction of upper bidiagonal matrices

For each matrix $L^{\ell}M^{\ell}$, $\ell = 1,2,3$, we construct recursively upper bidiagonal matrices $U^{\ell}_{2}, U^{\ell}_{3}, \ldots, U^{\ell}_{m_\ell-2}$ such that the upper triangular matrix $U^{\ell} = U^{\ell}_{2}U^{\ell}_{3}\cdots U^{\ell}_{m_\ell-2}$ transforms $L^{\ell}M^{\ell}$ into a matrix $W^{\ell} = L^{\ell}M^{\ell}U^{\ell}$ with nonzero diagonal elements, $W^{\ell}(i,i) \neq 0$, $i = 1,2,\ldots,m_\ell$, nonzero $W^{\ell}(1:m_\ell, m_\ell) \neq 0$ in the last column and zero elsewhere. We call such a matrix a "diagonal+last-column" matrix.

We describe the zeroing process of the upper bidiagonal matrix U^{ℓ}_{k} on the two-row matrix

$$L^{\ell}M^{\ell}U^{\ell}_{2}U^{\ell}_{3}\cdots U^{\ell}_{k-1}(k:k+1, 1:m_\ell) = \begin{bmatrix} y_{k,1} & \cdots & y_{k,k-1} & 1 & 1 & \cdots & 1 & y_{k,m_\ell} \\ y_{k+1,1} & \cdots & y_{k+1,k-1} & y_{k+1,k} & y_{k+1,k+1} & \cdots & y_{k+1,m_\ell-1} & y_{k+1,m_\ell} \end{bmatrix}. \quad (47)$$

The divisors

$$\sigma_i = \frac{k-1}{\mu_\ell(i) - \mu_\ell(i-k+1)} = U^{\ell}_{k}(i,i), \qquad i = k+1, k+2, \ldots, m_\ell - 1, \quad (48)$$

where $\mu_\ell(k) = M^{\ell}(3,k)$, define the upper bidiagonal matrix

$$U^{\ell}_{k} = \begin{bmatrix} I_{k-1} & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & -\sigma_{k+1} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sigma_{k+1} & -\sigma_{k+2} & \cdots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_{m_\ell-2} & -\sigma_{m_\ell-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \sigma_{m_\ell-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}. \quad (49)$$

Right-multiplying (47) by U_k^ℓ zeros the 1's in position $k, \ldots, m_\ell - 1$ in the first row and puts 1's in position $k+1, \ldots, m_\ell - 1$ in the second row:

$$L^\ell M^\ell U_2^\ell U_3^\ell \cdots U_{k-1}^\ell U_k^\ell (k : k+1, 1 : m_\ell) = \begin{bmatrix} y_{k1} & \cdots & y_{k,k-1} & 1 & 0 & \cdots & 0 & y_{k,m_\ell} \\ y_{k+1,1} & \cdots & y_{k+1,k-1} & y_{k+1,k} & 1 & \cdots & 1 & y_{k+1,m_\ell} \end{bmatrix}. \quad (50)$$

Thus, $U^\ell = U_2^\ell U_3^\ell \cdots U_{m_\ell-2}^\ell$ transforms the upper triangular matrix $L^\ell M^\ell$ into the diagonal+last-column matrix

$$W^\ell = L^\ell M^\ell U_2^\ell U_3^\ell \cdots U_{m_\ell-2}^\ell \quad (51)$$

in $(m_\ell - 3)$ steps.

Process 2 *At the kth step, starting with* $k = 2$,

- $M^{\ell(k-1)} = L^\ell M^\ell U_2^\ell U_3^\ell \cdots U_{k-1}^\ell$ *is a diagonal+last-column matrix in rows* 1 *to* $k-1$.
- *The divisors in* U_k^ℓ *are obtained from* $M^{\ell(k-1)}(k+1, k+1 : m_\ell)$ *since* $M^{\ell(k-1)}(k+1, j) - M^{\ell(k-1)}(k+1, j-1) \neq 0$, $j = k+1, k+2, \ldots, m_\ell - 1$.

Algorithm 2 in Appendix A describes this process.

3.5 Particular Variable-step HB(p)3

The general HB(p)3 methods obtained in Chapter 4 contain one free coefficient, c_2, and depends on h_{n+1} and the previous nodes, $x_n, x_{n-1}, \ldots, x_{n-(p-4)}$, which determine $\eta_2, \ldots, \eta_{p-3}$ in (16).

For simplicity and to reduce computation, only one intermediary point, c_2, is introduced while $c_1 = 0$ and $c_3 = 1$ are taken at the ends of the interval of integration, thus reducing the cost per step in the implementation of a particular variable-step HB(p)3, for $p = 5, 6, \ldots, 15$. After some numerical experimentation on the choice of c_2, the following three coefficients were chosen as

$$c_1 = 0, \quad c_2 = \frac{2}{3}, \quad c_3 = 1. \quad (52)$$

We also denote this family of particular HB(p)3 by HB(5-15)3.

The remaining of this thesis is concerned with the family of particular variable-step HB(p)3 with coefficients c_j given in (52).

The procedure to advance integration from x_n to x_{n+1} is as follows.

(a) The order p is obtained by the procedure of Chapter 4. Then, the stepsize, h_{n+1}, is obtained by formula (57) of Section 4.2 with $\kappa = p-1$, that will be given below.

(b) The numbers $\eta_2, \ldots, \eta_{p-4}$, defined in (16), are calculated.

(c) The coefficients of integration formula IF, predictors P_2, P_3 and step control predictor P_4 are obtained successively as solutions of systems (34), (36), (38) and (40).

(d) The values y_{n+c_2}, y_{n+c_3}, y_{n+1}, and $\widetilde{y}_{n+1}$ are obtained by formulas (10)–(13).

(e) The step is accepted if $|y_{n+1} - \widetilde{y}_{n+1}|$ is smaller than the chosen tolerance and the program goes to (a) with n replaced by $n+1$. Otherwise the program returns to (a) with the same order p and the smaller stepsize $0.7\, h_{n+1}$.

3.6 Fast Solution of Vandemonde-type Systems

Symbolic software can be conveniently used to construct the elementary matrix functions L_k^ℓ and U_k^ℓ, $\ell = 1, 2, 3$, only once as functions of η_j, for $j = 2, 3, \ldots, p-3$. Symbolic Algorithms 1 and 2 in Appendix A do this only once to produce diagonal+last-column matrices in $O(m^3)$ operations. These elementary matrix functions are permanently incorporated in, and used by the fast C++ Algorithms 3 and 4, in Appendix A, to solve systems (34), (36), (38) and (40) at each integration step in $O(m^2)$ operations.

3.6.1 Solution of $M^\ell \boldsymbol{u}^\ell = \boldsymbol{r}^\ell$, $\ell = 1, 2, 3$

We let

$$m_1 = p+1, \quad m_2 = p-1, \quad m_3 = p, \quad m_4 = p-2,$$

as defined in (43).

First, the elimination procedure of subsection 3.4.1 is applied to M^ℓ to construct $m_\ell \times m_\ell$ lower bidiagonal matrices L_k^ℓ, $k = 3, \ldots, m_\ell - 1$, with multipliers

$$\tau_i = \frac{i+1-k}{M^\ell(3,k)} = -L_k^\ell(i,i), \qquad i = k+1, k+2, \ldots, m_\ell. \tag{53}$$

The matrix $L^\ell = L_{m_\ell-1}^\ell \cdots L_4^\ell L_3^\ell$ transforms the coefficient matrix M^ℓ into the upper triangular matrix $L^\ell M^\ell$ of the form (46).

Second, the elimination procedure of subsection 3.4.2 is used to construct $m_\ell \times m_\ell$ upper bidiagonal matrices U_k^ℓ, $k = 2, \ldots, m_\ell - 2$, with multipliers

$$\sigma_i = \frac{k-1}{M^\ell(3,i) - M^\ell(3,i-k+1)} = U_k^\ell(i,i), \qquad i = k+1, k+2, \ldots, m_\ell - 1. \tag{54}$$

The right-product of the U_k^ℓ will transform $L^\ell M^\ell$ into a diagonal+last-column matrix W^ℓ of the form (51).

Finally, a factored Gaussian elimination, $L_{m_\ell+1}^\ell L_{m_\ell}^\ell$, diagonalizes W^ℓ as follows. First, $W^\ell(m_\ell, m_\ell)$ is set to 1 by the diagonal matrix $L_{m_\ell}^\ell$:

$$L_{m_\ell}^\ell(i,i) = 1, \qquad i = 1, \ldots, m_\ell - 1,$$
$$L_{m_\ell}^\ell(m_\ell, m_\ell) = 1/W^\ell(m_\ell, m_\ell).$$

Then the non-diagonal entries in the last column of $L_{m_\ell}^\ell W^\ell$ are zeroed by the unit diagonal+last-column matrix $L_{m_\ell+1}^\ell$ whose last column has top $m_\ell - 1$ entries

$$L_{m_\ell+1}^\ell(1 : m_\ell - 1, m_\ell) - (L_{m_\ell}^\ell W^\ell)(1 : m_\ell - 1, m_\ell).$$

This procedure transforms M^ℓ into the diagonal matrix

$$D^\ell = L_{m_\ell+1}^\ell L_{m_\ell}^\ell \cdots L_3^\ell M^\ell U_2^\ell U_3^\ell \cdots U_{m_\ell-2}^\ell,$$

where

$$D^\ell(i,i) = 1, \qquad i = 1, 2, 3, m_\ell,$$

and

$$D^\ell(i,i) = \frac{(i-2)!}{[-M^\ell(3,3)]\,[-M^\ell(3,4)] \cdots [-M^\ell(3,i-1)]}, \qquad i = 4, 5, \ldots, m_\ell - 1.$$

Thus we have the following factorization of M^ℓ into the product of elementary matrices:

$$M^\ell = \left(L^\ell_{m_\ell+1} L^\ell_{m_\ell} \cdots L^\ell_3\right)^{-1} D^\ell \left(U^\ell_2 U^\ell_3 \cdots U^\ell_{m_\ell-2}\right)^{-1},$$

and the solution is

$$\boldsymbol{u}^\ell = U^\ell_2 U^\ell_3 \cdots U^\ell_{m_\ell-2} (D^\ell)^{-1} L^\ell_{m_\ell+1} L^\ell_{m_\ell} \cdots L^\ell_3 \boldsymbol{r}^\ell, \tag{55}$$

where fast computation goes from right to left.

Procedure (55) is implemented in Algorithm 3 in Appendix A in $O(m_\ell^2)$ operations. The input is $M = M^\ell$; $m = m_\ell$; $\boldsymbol{r} = \boldsymbol{r}^\ell$; $L_k = L^\ell_k$, $k = 3, 4, \ldots, m_\ell - 1$; $U_k = U^\ell_k$, $k = 2, 3, \ldots, m_\ell - 2$; and $D = D^\ell$. The output is $\boldsymbol{u} = \boldsymbol{u}^\ell$;

3.6.2 Solution of $M^4\boldsymbol{u}^4 = \boldsymbol{r}^4$

The algorithm to solve the system $M^4\boldsymbol{u}^4 = \boldsymbol{r}^4$ in $O(m_4^2)$ operations is similar to the algorithm for the primal system of [5, p. 896] and is described in Algorithm 4 in Appendix A. The input is $M = M^4$; $m = m_4$; $\boldsymbol{r} = \boldsymbol{r}^4$ and the output is $\boldsymbol{u} = \boldsymbol{u}^4$.

Remark 1 *Formulae* (10)–(13) *can be put in matrix form. For instance,* (12) *can be written as*

$$y_{n+1} = F^1 \boldsymbol{u}^1.$$

where

$$F^1 = \left[y_n, h_{n+1}f_n, h_{n+1}f_{n+c_2}, h_{n+1}f_{n+c_3}, h_{n+1}f_{n-1}, h_{n+1}f_{n-2}, \ldots, h_{n+1}f_{n-(p-4)}, y_{n-1}\right],$$

and

$$\boldsymbol{u}^1 = [\alpha_{10}, b_{11}, b_{12}, b_{13}, \beta_{11}, \beta_{12}, \ldots, \beta_{1,p-4}, \alpha_{11}]^T,$$

It is interesting to note the three decomposition forms of the system $F\boldsymbol{u}$*:*

$$\begin{aligned} &F(UD^{-1}L\boldsymbol{r}) && \textit{(generalized Lagrange interpolation)},\\ &(FUD^{-1})L\boldsymbol{r} && \textit{(Krogh's modified divided differences)},\\ &(FUD^{-1}L)\boldsymbol{r} && \textit{(Nordsieck's formulation)}. \end{aligned}$$

The first form is used in this thesis, the second form for Vandermonde systems is found in [28], and the third form is found in [37].

Chapter 4

Implementation and Numerical Results

4.1 Regions of Absolute Stability and Principal Error Term

To obtain the regions of absolute stability, R, of these methods, we apply predictors P_2, P_3 and integration formula IF of each method, with constant h, to the linear test equation

$$y' = \lambda y, \qquad y_0 = 1.$$

This gives the difference equation and the corresponding characteristic equation

$$\sum_{j=0}^{p-3} \gamma_j y_{n+j} = 0, \qquad \sum_{j=0}^{p-3} \gamma_j r^j = 0, \tag{56}$$

respectively, where $p-3$ is the number of steps of the method. A complex number λh is in R if the $p-3$ roots of the characteristic equation satisfy the root condition $|r_s| \leq 1$ and the multiple roots satisfy $|r_s| < 1$. The method used in [35] to find R is similar to the one used for k-step multistep methods (see [25, pp. 256–257]).

The upper part of the regions of absolute stability, R, of HB(5-15)3 are shown in grey in Fig. 2 taken from [35]. The region R is symmetric with respect to the real axis.

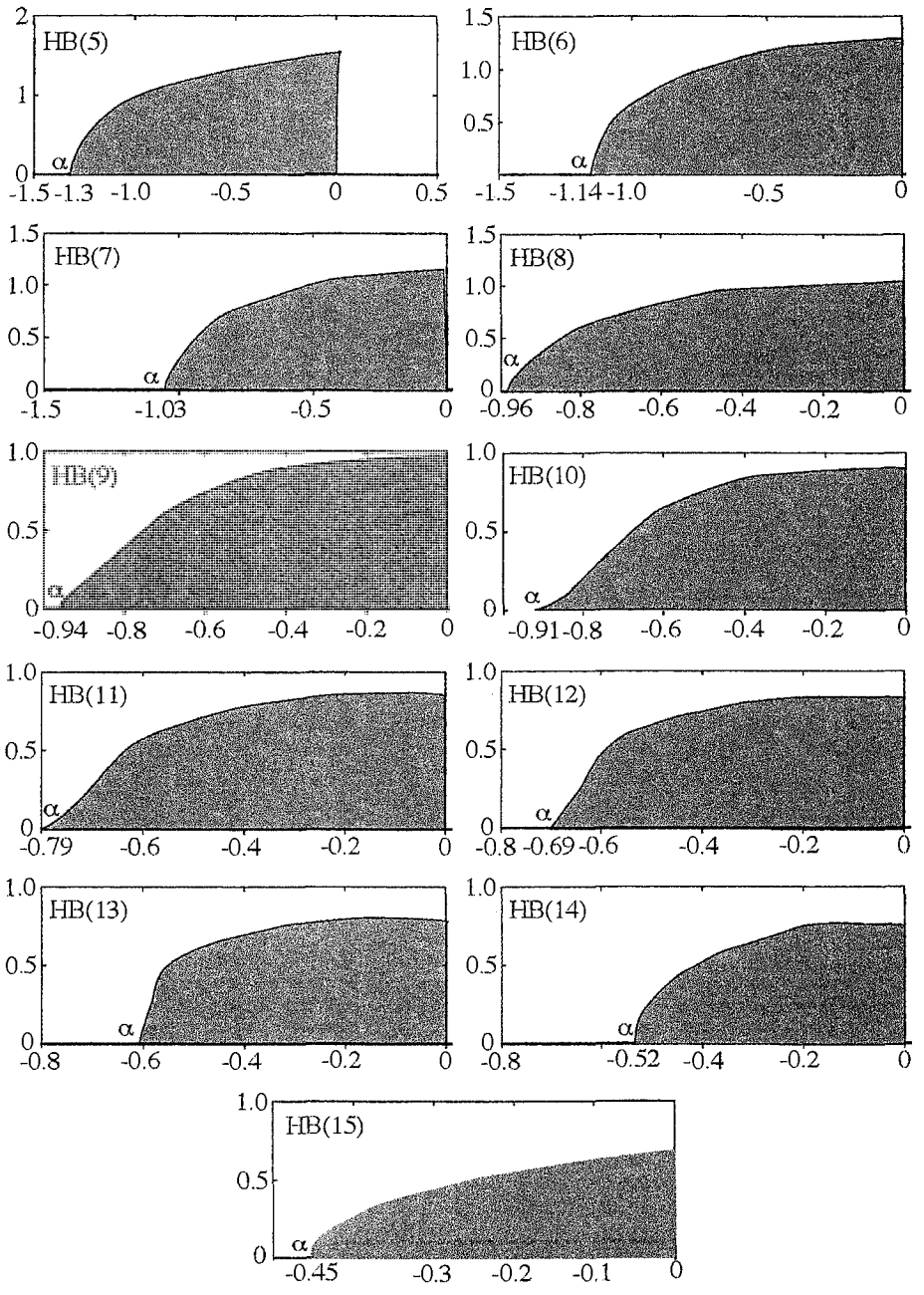

Figure 2: Regions of absolute stability, R, of HB(5-15)3.

Table 5: For given order p, the table lists the abscissa of absolute stability, α, and the norm $\|\text{PLTC}\|_2$ for HB(p)3 and ABM($p, p-1$), respectively.

	α		$\|\text{PLTC}\|_2$	
p	HB(p)3	ABM($p, p-1$)	HB(p)3	ABM($p, p-1$)
5	−1.30	−1.40	1.31e-02	1.04e-01
6	−1.14	−1.03	7.85e-03	9.56e-02
7	−1.03	−0.78	5.37e-03	8.89e-02
8	−0.96	−0.60	3.97e-03	8.37e-02
9	−0.94	−0.44	3.09e-03	7.94e-02
10	−0.91	−0.33	2.49e-03	7.58e-02
11	−0.79	−0.26	2.06e-03	7.28e-02
12	−0.69	−0.21	1.75e-03	7.01e-02
13	−0.60	−0.06	1.50e-03	6.78e-02
14	−0.52		1.31e-03	
15	−0.45		1.16e-03	

Let ABM($p, p-1$) denote the ABM method with predictor of order $p-1$ and corrector of order p in PECE mode [41, p. 135–140]. The intervals of absolute stability $(\alpha, 0)$ of HB(p)3 and ABM($p, p-1$) are listed in the left part of Table 5. It is seen that HB methods have larger intervals of absolute stability than ABM methods of comparable order.

The principal error term of HB(5-15)3 is of the form

$$\left[\delta_1 \{f^p\} + \delta_2 p \{\{f^{p-2}\} f\} + \delta_3 \{{}_2 f^{p-1}\}_2 + \delta_4 \{{}_3 f^{p-2}\}_3\right] h^{p+1}.$$

where $\{f^p\}, \{\{f^{p-2}\} f\}, \{{}_2 f^{p-1}\}_2, \{{}_3 f^{p-2}\}_3$ are elementary differentials defined in [8], [30] and [25]. The principal local truncation coefficients (PLTC), $\delta_1, \delta_2, \delta_3$ and δ_4, of the principal error term are listed in Table 6. The norms $\|\text{PLTC}\|_2$ for HB(5-15)3 and ABM methods of order $p = 5, \ldots, 13$ are listed in the right part of Table 5. It is observed that the norm $\|\text{PLTC}\|_2$ for HB methods are between 3% to 12% of the norm $\|\text{PLTC}\|_2$ for ABM methods of comparable order.

Table 6: For each order p, the table lists the principal local truncation coefficients for HB(5-15)3.

p	δ_1	δ_2	δ_3	δ_4
5	$\frac{122167958642}{593736279000119}$	$-\frac{1759218604442}{949978046398679}$	$-\frac{63281244764}{61509369910607}$	$\frac{23456248059221}{2533274790395869}$
6	$\frac{59}{873180}$	$-\frac{3229}{3492720}$	$-\frac{4688352}{11418240917}$	$\frac{8854480}{1597004107}$
7	$\frac{85471927}{3076068117697}$	$-\frac{707023}{1303219439}$	$-\frac{468962}{2354249605}$	$\frac{3854824}{1015639297}$
8	$\frac{2354423}{179914245043}$	$-\frac{587270}{1673247147}$	$-\frac{189737}{1753207927}$	$\frac{3686879}{1313789481}$
9	$\frac{336355}{49883522711}$	$-\frac{68883}{283875427}$	$-\frac{948607}{15000368616}$	$\frac{2708477}{1240785068}$
10	$\frac{179086}{48448766529}$	$-\frac{1873166}{10637173139}$	$-\frac{232193}{5984151746}$	$\frac{1604661}{911571649}$
11	$\frac{116718}{55190680283}$	$-\frac{172071}{1297436831}$	$-\frac{244227}{9931242032}$	$\frac{1821263}{1248757818}$
12	$\frac{62473}{50220577404}$	$-\frac{136176}{1323940543}$	$-\frac{235381}{14807436217}$	$\frac{4230511}{3428202290}$
13	$\frac{170880}{230209966259}$	$-\frac{686019}{8397578576}$	$-\frac{172405}{16640440694}$	$\frac{5242881}{4937467859}$
14	$\frac{123146}{278016455347}$	$-\frac{613889}{9276690181}$	$-\frac{300125}{44620746959}$	$\frac{2397797}{2588362967}$
15	$\frac{53849}{207525400761}$	$-\frac{448503}{8228865958}$	$-\frac{82398}{19247139919}$	$\frac{778718}{952540877}$

4.2 Controlling Stepsize and Order

A variant of the procedure described in [41] is used to control the stepsize and order of our VSVO HB methods.

- The program computes the maximum norm

$$\mathrm{E} = \|y_{n+1} - \widetilde{y}_{n+1,q}\|_\infty,$$

where y_{n+1} is the value obtained by the integration formula IF of order p and $\widetilde{y}_{n+1,q} := \widetilde{y}_{n+1}$ is the value obtained by the step control predictor P_4 of order $q = p - 2$. In other word, the numerical solution is obtained from the the lower order estimator.

- The stepsize h_{n+1} is obtained by the formula (see [26]):

$$h_{n+1} = \min\left\{h_{\max}, \beta\, h_n \left(\frac{\text{tolerance}}{\mathrm{E}}\right)^{1/\kappa}, 4\, h_n\right\}, \tag{57}$$

where $h_{\max}$ is the largest admissible step determined by the user or, by default, by the software, tolerance is chosen by the user, $\kappa = p - 1$ and $\beta = 0.81$ is a conservative safety factor.

- The coefficients of integration formula IF, predictors P_2, P_3 and step control predictor P_4 are obtained successively as solutions of the linear systems (34), (36), (38) and (40).

- The step to x_{n+1} is accepted if $\mathrm{E} \leq$ tolerance, else it is rejected and the program returns to the previous step with smaller step $0.7\, h_{n+1}$.

- If the step to x_{n+1} is successful, besides P_4, three other Adams–Moulton step control predictors,

$$\widetilde{y}_{n+1,\rho} = y_n + h_{n+1}\left(a_{41} f_n + a_{43} f_{n+c_3} + \sum_{j=1}^{\rho-2} \beta_{4j} f_{n-j}\right), \tag{58}$$

of order $\rho = q \pm 1$ and $q-2$ are used to produce the three values $\widetilde{y}_{n+1,\rho}$, used to control the order and stepsize by means of the following three maximum norms:

$$\mathrm{E}_{\pm 1} = \|y_{n+1} - \widetilde{y}_{n+1,q\pm 1}\|_\infty, \quad \mathrm{E}_{-2} = \|y_{n+1} - \widetilde{y}_{n+1,q-2}\|_\infty,$$

which estimate the local error at x_{n+1} had the step to x_{n+1} been taken at orders $q \pm 1$ and $q-2$, respectively. These three quantities are formed with E so that much of the order and stepsize selection can be done by using the following rules. The lowest satisfactory order is used. Thus, the order is lowered if

$$\mathrm{E}_{-1} \leq \min\{\mathrm{E}, \mathrm{E}_{+1}\} \quad \text{or} \quad \mathrm{E} \geq \max\{\mathrm{E}_{-1}, \mathrm{E}_{-2}\}.$$

The order is raised only if the following stronger conditions,

$$\mathrm{E}_{+1} < \mathrm{E} < \max\{\mathrm{E}_{-1}, \mathrm{E}_{-2}\},$$

are satisfied. Otherwise, the same order is kept.

- When the order q of P_4 is 13, E_{+1} is not available; Thus, the order is lowered if

$$\mathrm{E} \geq \max\{\mathrm{E}_{-1}, \mathrm{E}_{-2}\}.$$

- When $q = 3$, the order is raised only if

$$\mathrm{E}_{+1} < \mathrm{E}.$$

- After selecting the order to be used, κ and E are reassigned according to the selected order. For example, if the order is to be lowered in the next step, $\kappa_{n+1} = \kappa_n - 1$ and $\mathrm{E} = \mathrm{E}_{-1}$. The stepsize h_{n+1} is then controlled by formula (57).

4.3 HB(5-15)3 against `ODE113` in Matlab

We list below standard test problems for ODE solvers. Many of these problems have been used to test HB(5-15)3. In this section, we report on the numerical performance of HB(5-15)3 (programmed in Matlab) and Matlab's `ode113` on most of the starred problems. In the next section, we report on the numerical performance of HB(5-15)3 and DP(8,7) (both programmed in C++) on four of the starred problems.

***CUBICWAVE** (CUBIC) Nonlinear wave equation in deep water [7].

***BRUSSELATOR** (BRUS) This reaction-diffusion partial differential equation is to be solved by the method of lines[1]. Thus the equation is transformed into a system of ordinary differential equations. [25, p. 248–249].

ARENSTORF'S ORBITS (AREN) Equations for the restricted three-body problem [25, p. 129–130].

THE PLEIADES (PLEI) A celestial mechanics problem for seven stars in the plane [25, p. 245–246].

RESTRICTED 3-BODY PROBLEM (R-3-BODY) Equations of motion of a restricted three-body problem [41, p. 246–247].

EULER'S EQUATION (EULER) Equations of motion for a rigid body without external force [41, p. 242–243].

The DETEST problems are divided into the following five classes, each containing five equations and are scaled so that $x_0 = 0$ and $x_f = 20$.

SINGLE EQUATIONS:

A3 $y' = -y\cos x, \quad y(0) = 1$ (An oscillatory problem).

SMALL SYSTEMS:

B1 The growth of two conflicting populations [14, p. 102].

B4 The integral surface of a torus [16, p. 9].

ORBIT EQUATIONS:

***D1** Two-body problem with eccentricity $\epsilon = 0.1$.

[1]For a first-order partial differential equation (PDE), the method of lines discovers a line along which the PDE becomes an ordinary differential equation (ODE). Once the ODE is found, it can be solved along the this line and transformed into a solution for the original PDE. If the line is a characteristic line or characteristic, the method is called method of characteristics.

D2 As in D1 except with eccentricity $\epsilon = 0.3$.

D3 As in D1 except with eccentricity $\epsilon = 0.5$.

D4 As in D1 except with eccentricity $\epsilon = 0.7$.

***D5** As in D1 except with eccentricity $\epsilon = 0.9$.

HIGHER ORDER EQUATIONS:

E1 Derived from Bessel's equation of order 1/2 with origin shifted one unit to the left [14, p. 4, 69].

E2 Derived from Van der Pol's equation [14, p. 358, 531].

CPU time, the number of function evaluations and the maximum global error are compared on the above starred problems to test HB(5-15)3 as a VSVO solver on problems which require many function evaluations. Usually large stepsizes can be used to solve easier problems with fewer function evaluations. HB(5-15)3 wins on expensive problems. That is why we construct it.

The three starting values for HB(5-15)3 were obtained by Dormand–Prince pair DP(5,4)7M (see [17]) with initial stepsize, h_1, chosen by a method similar to steps (a) and (b) of [25, p. 169].

Computations were performed on a System with a dual 1.66 GHz Laptop 2 GB DDR2 running under Windows Vista Home Premium and Matlab Version 6.5. Algorithms 3 and 4 were written in C and made into system-dependent Matlab `mex` files for speed.

4.3.1 CPU time against maximum global error

As a first comparison, the Maximum Global Error (MGE) has been plotted for the methods treated here. MGE is taken to be $\max_n\{|y_{n+1} - y(t_{n+1})|_\infty\}$ of the difference between the numerical and the reference solution at every integration step. The reference solution is obtained by means of DP(8.7) at stringent tolerance. In Figs. 3

to 5 the horizontal axis is total CPU time (CPU) in seconds for a given tolerance and the vertical axis is

$$\log_{10}(|\text{MGE}|). \tag{59}$$

4.3.2 CPU percentage efficiency gain of HB(5-15)3 against `ode113`

The CPU percentage efficiency gain (CPU PEG) is defined by formula (cf. Sharp [43]),

$$(\text{CPU PEG})_i = 100\left[\frac{\sum_j \text{CPU}_{2,ij}}{\sum_j \text{CPU}_{1,ij}} - 1\right], \tag{60}$$

where $\text{CPU}_{1,ij}$ and $\text{CPU}_{2,ij}$ are the CPU time of methods 1 and 2, respectively, associated with problem i, and $j = -\log_{10}(|\text{MGE}|)$. For each problem i, the CPU time is the execution time of 600 runs and we divide the execution time by 600 to obtain the CPU time for each run.

The CPU PEG for the problems considered in this thesis is listed in Table 7.

Table 7: CPU percentage efficiency gain (CPU PEG) of HB(5-15)3 over `ode113` for the listed problems.

Harder problems	CPU PEG	Harder problems	CPU PEG
Arenstorf	67%	D1	37%
Brusselator	36%	D2	41%
Euler	26%	D3	52%
Pleiades	57%	D4	82%
Restricted 3 body	47%	D5	49%
B1	43%	E2	58%
Easier problems		Easier problems	
A3	31%	E1	41%
B4	19%		

It is seen from Figs. 3, 4, 5 and Table 7 that the new VSVO HB(5-15)3 compare favorably with Matlab's `ode113` on the basis of the MGE Efficiency Curves against

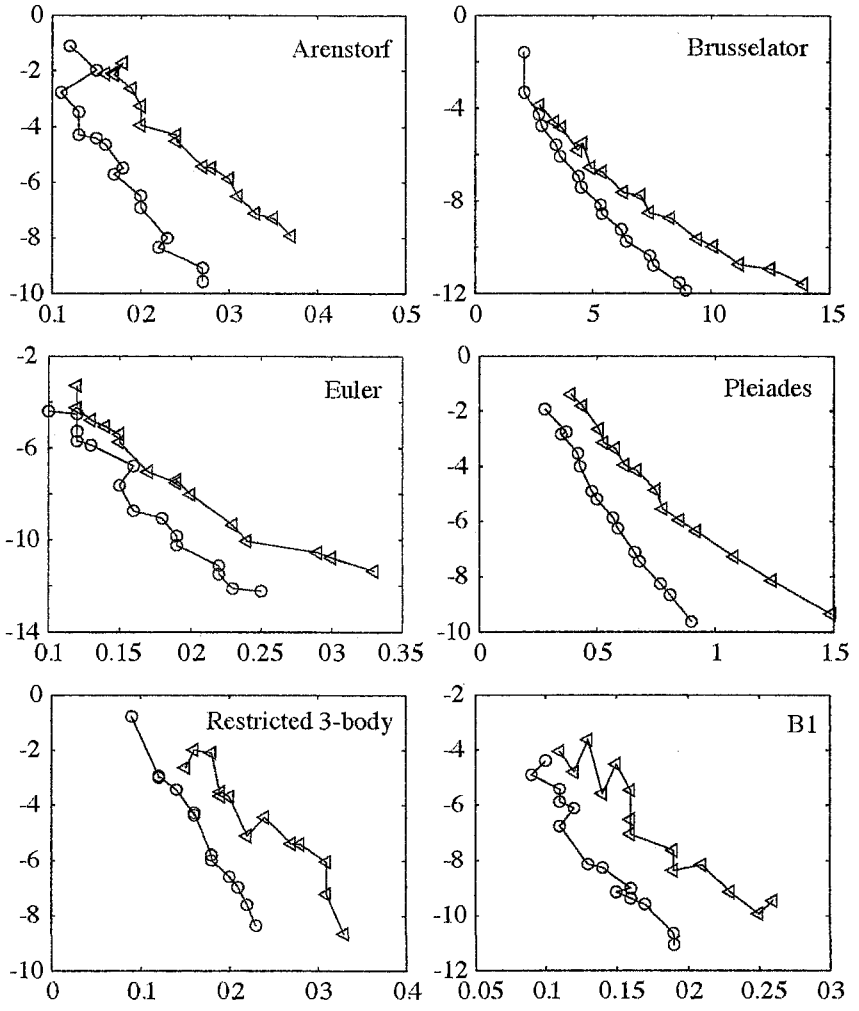

Figure 3: CPU time in seconds (horizontal axis) versus $\log_{10}$ (|MGE|) (vertical axis) for Arenstorf, Brusselator, Euler, Pleiades, restricted three-body and B1 in Matlab.

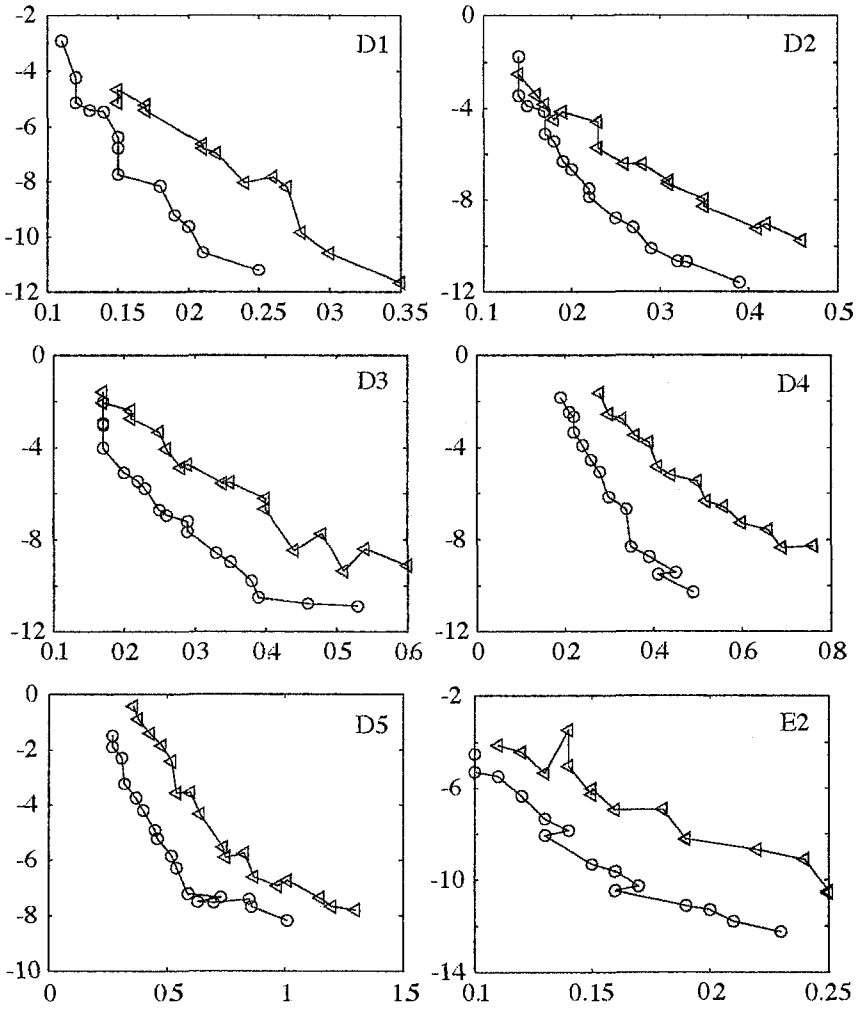

Figure 4: CPU time in seconds (horizontal axis) versus $\log_{10}$ (|MGE|) (vertical axis) for D1 to D5 and E2 in Matlab.

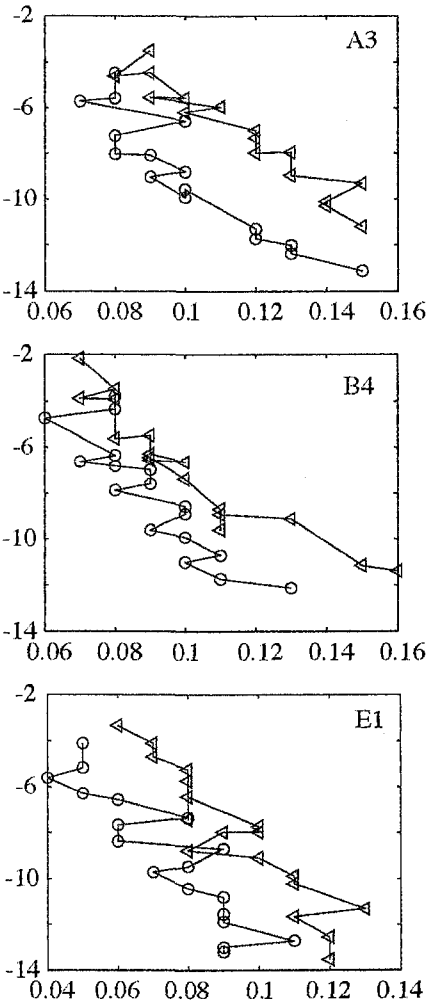

Figure 5: CPU time in seconds (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for A3, B4 and E1 in Matlab.

the CPU time for the problems in hand. Figure 5 illustrate the fact that the behavior of ODE solvers is highly problem dependent (see [25], page 253).

4.3.3 Maximum global error against tolerance

Table 8 lists several numerical results related to the step control for the problems in hand on the time interval $[0, t_f]$ with set tolerance (TOL), namely, CPU time (CPU), number of function evaluations (NFE), the number of failed attempts (REJ)[2] and the maximum global error (MGE) of HB(5-15)3 and `ode113`.

For equivalent MGE, the boldface data in Table 8 show that HB(5-15)3 uses less CPU time and fewer function evaluations than `ode113` for tolerances 10^{-07} and 10^{-10}, respectively. It is seen that HB(5-15)3 controls better the stepsize for given tolerance and maximum global error than `ode113` even though the step control is problem dependent.

Consider the example of problem AREN in Table 8. The MGEs of the two algorithms are considered the same, when the MGE of HB(5-15)3 is **3.2e-06** and the MGE of ode113 is **3.3e-06**. The NFE of HB(5-15)3 is **1203** and of ode113 is **1637**. Thus HB(5-15)3 takes less NFE for the same MGE than ode113. The CPU time of HB(5-15)3 is **0.16** while the CPU time of ode113 is **0.24**. In other words, HB(5-15)3 takes less CPU time than ode113 for the same MGE.

4.4 HB(5-15)3 against DP(8,7) in C++

In this section, the numerical performance of HB(5-15)3 and DP(8,7) in C++ is compared on two harder problems: the Brusselator and the Cubicwave, and on two easier problems: the DETEST two-body problems D1 and D5 [26].

[2]Softwares, for example, Matlab, usually have the option to find the number of failed attempts, that is, the number of steps for which the local error exceeded the tolerance [26].

Table 8: For each problem, time interval $[0, t_f]$ and tolerance (TOL), the table lists the CPU time in seconds (CPU), number of function evaluations (NFE), number of failed attempts (REJ) and maximum global error (MGE) for the HB(5-15)3 method in the left column and the `ode113` in the right column, respectively.

		HB(5-15)3 and `ode113`							
Problem	TOL	CPU		NFE		REJ		MGE	
BRUS	10^{-04}	2.73	2.57	702	736	2	5	5.5e-05	2.6e-04
$t_f = 7.5$	10^{-07}	**5.34**	3.69	**1500**	1021	2	4	**6.8e-09**	1.5e-05
	10^{-10}	9.89	**9.84**	3018	**2419**	0	4	1.0e-12	**2.2e-10**
EULR	10^{-04}	0.12	0.09	708	485	11	4	3.0e-05	2.9e-03
	10^{-07}	**0.16**	0.15	**1203**	921	0	4	**1.8e-09**	4.0e-06
$t_f \approx 52.2$	10^{-10}	0.23	**0.24**	2001	**1637**	0	4	7.6e-13	**8.9e-11**
AREN	10^{-04}	0.15	0.15	615	491	11	20	1.1e-02	2.2e-01
	10^{-07}	**0.18**	0.18	**1161**	927	4	18	**3.2e-06**	2.3e-03
$t_f \approx 17.1$	10^{-10}	0.27	**0.27**	1917	**1637**	0	18	2.7e-10	**3.3e-06**
R3p	10^{-04}	0.12	0.11	630	513	11	20	1.1e-03	4.2e 00
	10^{-07}	**0.18**	0.18	**1176**	959	3	18	**1.6e-06**	8.0e-03
$t_f \approx 6.19$	10^{-10}	0.25	**0.27**	1941	**1632**	0	21	7.9e-09	**4.0e-06**
PLEI	10^{-04}	0.37	0.31	984	765	13	22	1.8e-03	5.0e-01
	10^{-07}	**0.57**	0.53	**1734**	1428	1	23	**1.3e-06**	7.2e-04
$t_f = 3$	10^{-10}	0.90	**0.85**	2886	**2408**	0	23	2.3e-10	**1.1e-06**
B1	10^{-04}	0.09	0.09	459	377	6	12	1.3e-05	5.3e-03
	10^{-07}	**0.13**	0.11	**834**	686	6	11	**2.6e-08**	3.1e-05
$t_f = 20$	10^{-10}	0.19	**0.19**	1374	**1208**	8	11	2.3e-11	**2.3e-08**
E2	10^{-04}	0.10	0.11	576	448	12	15	5.1e-06	6.0e-04
	10^{-07}	**0.15**	0.15	**999**	842	4	11	**4.8e-10**	5.2e-07
$t_f = 20$	10^{-10}	0.21	**0.22**	1629	**1437**	0	10	1.6e-12	**2.1e-09**
D1	10^{-04}	0.14	0.10	549	450	5	7	3.5e-03	2.1e-02
	10^{-07}	**0.15**	0.15	**954**	886	0	5	**4.2e-07**	7.5e-06
$t_f = 16\pi$	10^{-10}	0.21	**0.22**	1587	**1480**	0	3	2.7e-11	**1.1e-07**
D2	10^{-04}	0.15	0.12	789	577	12	12	9.8e-04	5.0e-02
	10^{-07}	**0.19**	0.18	**1419**	1156	4	15	**4.9e-07**	3.3e-05
$t_f = 16\pi$	10^{-10}	0.29	**0.31**	2340	**2035**	0	12	7.8e-11	**6.9e-08**
D3	10^{-04}	0.17	0.14	1020	785	16	26	1.2e-03	4.3e-01
	10^{-07}	**0.25**	0.25	**1890**	1530	8	25	**2.0e-07**	4.8e-04
$t_f = 16\pi$	10^{-10}	0.38	**0.40**	3090	**2688**	0	27	1.7e-10	**6.0e-07**
D4	10^{-04}	0.22	0.17	1365	1102	24	41	2.1e-03	2.8e 00
	10^{-07}	**0.30**	0.30	**2487**	1992	8	37	**7.0e-07**	2.8e-03
$t_f = 16\pi$	10^{-10}	0.49	**0.50**	4095	**3463**	0	36	5.1e-11	**3.3e-06**
D5	10^{-04}	0.27	0.26	1950	1662	32	77	3.3e-02	4.6e 00
	10^{-07}	**0.45**	0.43	**3669**	2970	8	61	**1.3e-05**	4.0e-02
$t_f = 16\pi$	10^{-10}	0.70	**0.74**	6081	**5099**	0	56	2.9e-08	**2.9e-06**
A3	10^{-04}	0.08	0.08	267	210	6	9	2.6e-06	1.9e-03
	10^{-07}	**0.10**	0.11	**435**	398	4	7	**1.6e-09**	1.1e-06
$t_f = 20$	10^{-10}	0.13	**0.13**	696	**611**	1	4	9.6e-13	**1.1e-09**

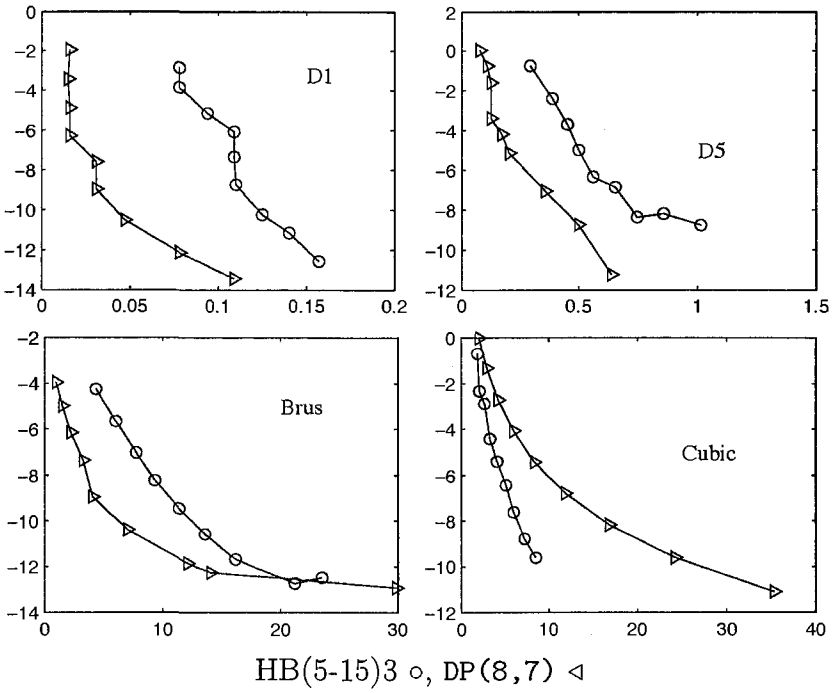

Figure 6: CPU time in seconds (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for D1, D5, Brusselator and Cubicwave.

4.4.1 CPU time against maximum global error

In Fig. 6, the CPU time in seconds (horizontal axis) is plotted against the common logarithm of the Maximum Global Error (MGE) (vertical axis),

$$\log_{10}(|\text{MGE}|), \tag{61}$$

for four problems. By comparing the two curves for the Cubicwave in Fig. 6, HB(5-15)3 wins for such an expensive problem since the curve for HB(5-15)3 lies to the left of the one for DP(8,7). However, DP(8,7) works better than HB(5-15)3 for easy problems such as D1 and D5, because in HB(5-15) many coefficients are calculated and the stepsizes have to be changed. Thus, HB(5-15) takes more CPU time than DP(8,7). At small MGE, which corresponds to stringent tolerance, HB(5-15)3 is better than DP(8,7) for the Brusselator.

Table 9 lists several numerical results related to the step control for the problems in hand on the time interval $[0, t_f]$ with set tolerance (TOL), namely, CPU time

Table 9: For each problem, time interval $[0, t_f]$ and tolerance (TOL), the table lists CPU time (CPU), number of function evaluations (NFE), number of failed attempts (REJ) and maximum global error (MGE) for the HB(5-15)3 methods in the left column and the `DP(8,7)` in the right column, respectively in C++.

		HB(5-15)3 and DP(8,7)							
Problem	TOL	CPU		NFE		REJ		MGE	
BRUS	10^{-04}	7.8e-02	1.6e-02	969	1066	8	2	1.4e-04	3.6e-04
	10^{-07}	1.1e-01	1.6e-02	1806	2769	5	0	4.6e-08	2.6e-08
$t_f = 7.5$	10^{-10}	1.4e-01	6.3e-02	3021	8749	3	0	7.0e-12	7.2e-13
CUBIC	10^{-04}	2.7e+00	2.9e+00	3702	6604	19	0	4.7e-03	4.8e-02
	10^{-07}	5.4e+00	8.5e+00	7101	18889	1	0	4.0e-06	3.8e-06
$t_f = 1\,000\,000$	10^{-10}	9.6e+00	2.4e+01	12027	54327	0	0	1.6e-09	2.4e-10
D1	10^{-04}	7.8e-02	1.6e-02	231	351	2	0	1.4e-04	3.6e-04
	10^{-07}	1.1e-01	1.6e-02	399	897	0	0	4.6e-08	2.6e-08
$t_f = 20$	10^{-10}	1.4e-01	6.3e-02	657	2652	0	0	7.0e-12	2.0e-13
D5	10^{-04}	1.6e-01	4.7e-02	888	1638	15	36	4.0e-03	1.8e-01
	10^{-07}	2.8e-01	7.9e-02	1620	2444	3	15	4.4e-07	6.4e-05
$t_f = 20$	10^{-10}	4.1e-01	1.7e-01	2655	6552	0	0	1.4e-09	1.8e-09

(CPU), number of function evaluations (NFE), number of failed attempts (REJ) and maximum global error (MGE) of HB(5-15)3 and `DP(8,7)`. These results agree with the curves in Fig. 6.

4.4.2 CPU percentage efficiency gain of HB(5-15)3 against DP(8,7)

Table 10 lists the CPU PEG for the four problems considered in this section. The CPU PEG is positive for Cubicwave, so HB(5-15)3 wins for this expensive equation. However, CPU PEG is negative for problem D1, D5 and Brusselator, i.e. DP(8,7) is better than HB(5-15)3 in these cases.

Table 10: CPU percentage efficiency gain (CPU PEG) of HB(5-15)3 over `DP(8,7)` for the listed problems in C++.

Problem	CPU PEG
D1	-75%
D5	-54%
Brusselator	-43%
Cubicwave	144%

4.5 The C++ Program

The structure of the C++ program for HB(5-15)3 is briefly described in this section. The listing of my 4509-line C++ program is available on demand.

The main part of HB(5-15)3 is programmed in `HB515main.h`. It consists of two parts: `DP5isteps.h` and `HB515mainsub.h`. `DP5isteps.h` does the initial step whereas `HB515mainsub.h` calculates four coefficients: the `IF` coefficient by `HB515IF`, the `P2` coefficient by `HB515P2`, the `P3` coefficient by `HB515P3`, and the coefficient of four local error estimators to control the stepsize and order by `HB515P4`. The C++ program for HB(5-15)3 will process from the top down to the bottom following the flow chart shown in Fig. 7.

The main program `main.cpp` will call `problem.h` which consists of four testing problems:

- HB515D1.h,
- HB515D5.h,
- HB515cubw.h,
- HB515brus.h,

When a testing problem is selected, the main method of HB(5-15)3 starts. The C++ program for HB(5-15)3 uses the most fundamental and important characteristic of the C++ programming language, that is, it is object-oriented. By using objects, the

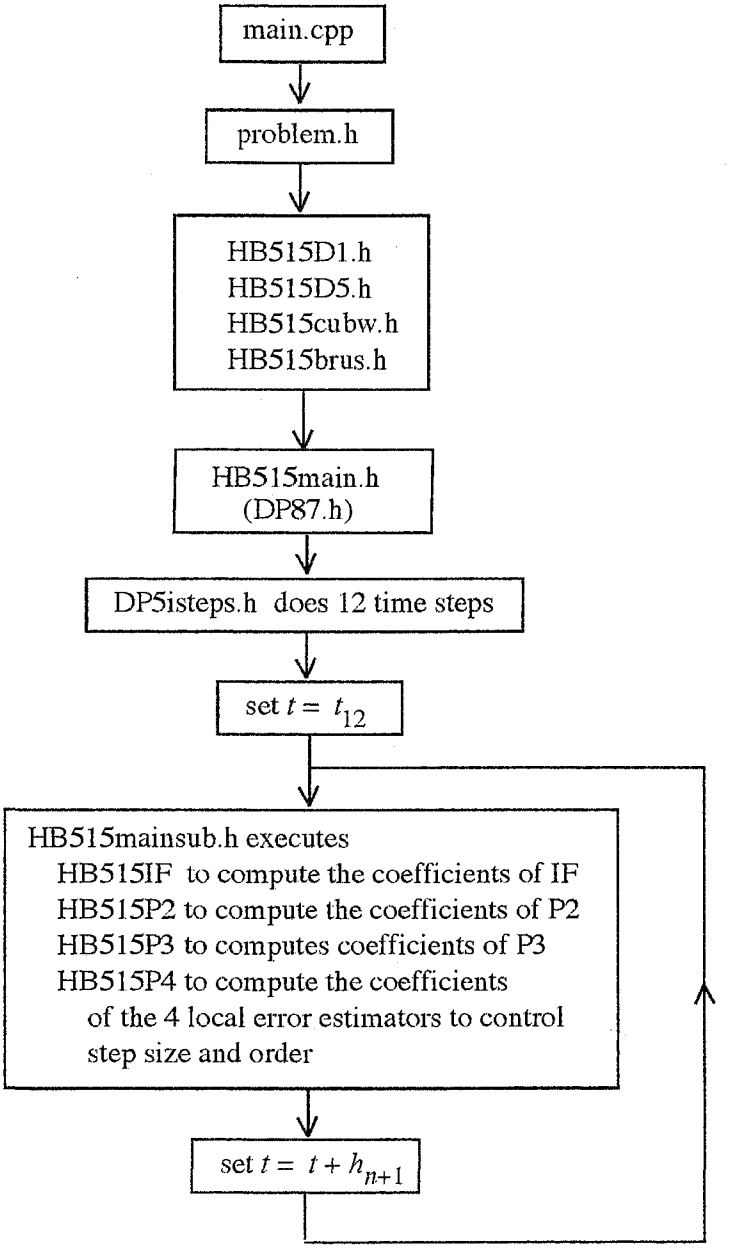

Figure 7: Chart of function calls and subroutines

program has a good structure and it is easy to understand as well. Memory release and template are two other points that I will explain in detail in the following.

First of all, objects are necessary in the C++ program of HB(5-15)3 because there are many array and matrix operations and C++ does not have a build-in type for matrix at all. The new objects, *array* and *matrix*, are created for easy operation. They will be called and take operations in the same way as simple type variables.

Secondly, I have to use the memory release technique of C++. Matrix is implemented by using a two-dimensional array as data structure. In order to make the matrix class applicable to various dimensions, it is necessary to use memory dynamic allocation and de-allocation. In this way, one can dynamically allocate and de-allocate memory by using the C++ operators `new` and `delete` to avoid consuming vast amounts of virtual memory and destroying performance through page swapping.

Thirdly, I make use of templates in the program. Templates are very useful when implementing generic constructs which can be used with any arbitrary type. C++ templates provide a way to re-use source code as opposed to inheritance which provide a way to re-use object code. Matrix can be used in other data types and objects. In this C++ program, most data types are "double" (precision) in the matrix. However, for the sake of re-usability, flexibility and compatibility, the design of the template is necessary.

One deficiency in the object-oriented C++ program for HB(5-15)3 is that the structure design requires more operations and takes more CPU time to establish arrays and matrices as can be seen by comparing the CPU time used by the non-object-oriented program in [35]. Improve performance of the C++ program is a future research project.

Chapter 5

Conclusion

A family of variable-step variable-order 3-stage Hermite–Birkhoff (HB) methods of orders 5 to 15 was constructed by solving generalized confluent Vandermonde systems containing Runge–Kutta type order conditions. The order and stepsize of these methods are controlled by four local error estimators.

These methods, in their vectorized Lagrange form, were tested on the Brusselator, Euler's equation, Arenstorf's orbits, the restricted three-body problem, the Pleiades, and the following nonstiff DETEST problems: an oscillatory problem A3, two-body problems D1–D5, the growth problem B1 of two conflicting populations, the integral surface of a torus B4, Bessel's equation of order 1/2 with the origin shifted one unit to the left E1, and Van der Pol's equation E2 with $\epsilon = 1$. The new methods were found generally to have larger regions of absolute stability, lower global error, use less CPU time and fewer function evaluations for the problems in hand than Matlab's `ode113`.

HB(5-15)3 was also tested against DP(8,7) on D1, D5, the Brusselator, and the Cubicwave. HB(5-15)3, which is aimed at solving expensive equations such as Cubicwave, uses less CPU time and requires fewer function evaluations than DP(8,7). However, DP(8,7) works better on easy problems such as D1, D5, and on the Brusselator at relaxed tolerance.

The HB(5-15)3 method has been written in object-oriented C++ and tested on typical problems.

Appendix A

Algorithms

Algorithm 1 ***This algorithm constructs lower bidiagonal matrices*** L_k ***(applied to*** IF, P_2 ***and*** P_3***) as functions of*** c_2, c_3 ***and*** η_j, $j = 2, 3, \ldots, p-3$.

For $k = 3 : m-1$, do the following iteration:

For $i = m : -1 : k+1$, do the following two steps:

Step (1) $L_k(i,i) := -M^\ell(i-1,k)/M^\ell(i,k)$.

Step (2) For $j = k : m$, compute:

$$M^\ell(i,j) := M^\ell(i-1,j) + M^\ell(i,j)L_k(i,i).$$

Algorithm 2 ***This algorithm constructs upper bidiagonal matrices*** U_k ***(applied to*** IF, P_2 ***and*** P_3***) as functions of*** c_2, c_3 ***and*** η_j, $j = 2, 3, \ldots, p-3$.

For $k = 2 : m-2$, do the following iteration:

For $j = m-1 : -1 : k+1$, do the following two steps:

Step (1) $U_k(j,j) := 1/[M^\ell(k+1,j) - M^\ell(k+1,j-1)]$.

Step (2) for $i = k : j$, compute

$$M^\ell(i,j) := (M^\ell(i,j) - M^\ell(i,j-1))U_k(j,j).$$

Algorithm 3 ***This algorithm solves the systems for*** IF, P_2 ***and*** P_3 ***in*** $O(m^2)$ ***operations***

Given $[\eta_2, \eta_3, \ldots, \eta_{p-3}]$ and $\boldsymbol{r} = r(1:m)$, the following algorithm overwrites $\boldsymbol{r}$ with the solution $\boldsymbol{u} = u(1:m)$ of the system $M\boldsymbol{u} = \boldsymbol{r}$.

Step (1) The following iteration overwrites $\boldsymbol{r} = r(1:m)$

with $L_{m-1}L_{m-2}\cdots L_3\boldsymbol{r}$:

for $k = 3, 4, \ldots, m-1$, compute

$$r(i) := r(i-1) + r(i)L_k(i,i), \qquad i = m, m-1, \ldots, k+1.$$

Step (2) First put

$$G(1:m) := M(1:m, m).$$

We obtain the coefficients of the last two row transformations, L_m and L_{m+1}, by means of the recursion:

for $k = 3, 4, \ldots, m-1$, compute

$$G(i) := G(i-1) + G(i)L_k(i,i), \qquad i = m, m-1, \ldots, k+1.$$

Step (3) The following computation overwrites the newly obtained $\boldsymbol{r}$ with $L_{m+1}L_m\boldsymbol{r}$:

$$r(m) := r(m)/G(m),$$

and for $k = m-1, m-2, \ldots, 1$, compute

$$r(k) := r(k) - G(k)r(m).$$

Step (4) The following iteration overwrites $\boldsymbol{r} = r(1:m)$

with $U_2U_3\cdots U_{m-2}D^{-1}\boldsymbol{r}$:

$$r(i) := r(i)/D(i,i), \qquad i = 1, 2, \ldots, m.$$

For $k = m-2, m-3, \ldots, 2$, compute

$$r(i) := r(i)U_k(i,i), \qquad i = k+1, k+2, \ldots, m-1,$$
$$r(i) := r(i) - r(i+1), \qquad i = k, k+1, \ldots, m-2.$$

Algorithm 4 ***This algorithm solves the system for the step control predictor*** P_4 ***in*** $O(m^2)$ ***operations***

Given $[\eta_2, \eta_3, \ldots, \eta_{p-3}]$ and $\boldsymbol{r} = r(1:m)$, the following algorithm overwrites $\boldsymbol{r}$ with the solution $\boldsymbol{u} = u(1:m)$ of the system $M\boldsymbol{u} = \boldsymbol{r}$.

Step (1) for $k = 2, 3, \ldots, m-1$, compute

$$r(i) := r(i-1) - r(i)\frac{i+1-k}{M^4(2,k)}, \qquad i = m, m-1, \ldots, k+1.$$

Step (2) compute

$$r(i) := r(i), \qquad i = 1, 2.$$
$$r(i) := r(i)\frac{[-M^4(2,2)]\,[-M^4(2,3)]\cdots[-M^4(2,i-1)]}{(i-1)!}, \qquad i = 3, 4, \ldots, m.$$

For $k = m-1, m-2, \ldots, 1$, compute

$$r(i) := r(i)\frac{k}{M^4(2,i+1) - M^4(2,i-k+1)}, \qquad i = k+1, k+2, \ldots, m,$$
$$r(i) := r(i) - r(i+1), \qquad i = k, k+1, \ldots, m-1.$$

Appendix B

Matlab Programming

Algorithm 3 which solves systems IF, P_2 and P_3 were programmed as subroutines in C, say, `IFsub`, `P2sub` and, `P3sub`

Algorithm 4 which solves the P_4 system was programmed in C as subroutines in C, say, `P4sub`.

A calling program in C, say, `IFP` which calls `IFsub`, `P2sub`, `P3sub` and, `P4sub` was compiled together with the four subroutines above by the Matlab `mex` command into `mex` files, say, `IFP.macmex`.

At runtime, the data of differential equations were input. Then, `IFP.macmex` was called and run to calculate the values of the coefficients of IF, P_2, P_3 and P_4 at each integration step until completion of the integration.

As an option, CPU time and NFE of function $f(x, y)$ in (1) at the runtime of Algorithms 3 and 4 can be recorded.

As another option, MGE can also be run. Matlab's `ode113` can be run with appropriate tolerance for comparison with HB(p)3.

The elementary matrices L_k^ℓ and U_k^ℓ, $\ell = 1, 2, 3, 4$, are constructed by Algorithms 1 and 2 as functions of η_j, for $j = 2, 3, \ldots, p-3$. These algorithms are not needed at runtime since these matrix functions are already implemented in the four subroutines `IFsub`, `P2sub`, `P3sub` and, `P4sub` which are compiled together with the calling program into Matlab `mex` file `IFP.macmex`.

Bibliography

[1] R. F. Arenstorf, *Periodic solutions of the restricted three-body problem representing analytic continuations of Keplerian elliptic motions*, Amer. J. Math., **LXXXV** (1963), pp. 27–35.

[2] R. Ashino, M. Nagase, and R. Vaillancourt, *Behind and beyond the* MATLAB *ODE suite*, Comput. Math. Applic., **40** (2000) pp. 491–512.

[3] R. Barrio, F. Blesa and M. Lara, *VSVO formulation of the Taylor method for the numerical solution of ODEs*, Comput. Math. Applic., **50** (2005), pp. 93–111.

[4] A. Björck and T. Elfving, *Algorithms for confluent Vandermonde systems*, Numer. Math., **21** (1973), pp. 130–137.

[5] A. Björck and V. Pereyra, *Solution of Vandermonde systems of equations*, Math. Comp. , **24** (1970), pp. 893–903.

[6] R. K. Brayton, F. G. Gustavson and G.D. Hachtel, *A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulas*, Proc. IEEE, **60** (1972), pp. 98–108.

[7] P. J. Bryant, *Nonlinear wave groups in deep water*, manuscript.

[8] J. C. Butcher, *Coefficients for the study of Runge–Kutta integration processes*, J. Aust. Math. Soc., **3** (1963), pp. 185–201.

[9] J. C. Butcher, *A modified multistep method for the numerical integration of ordinary differential equations*, J. Assoc. Comput. Mach., **12** (1965), pp. 124–135.

[10] J. C. Butcher, *A multistep generalization of Runge–Kutta methods with four or five stages*, J. Assoc. Comput. Mach., **14** (1967), pp. 84–99.

[11] J. C. Butcher, *The numerical analysis of ordinary differential equations*, Chapter 4, Wiley, Chichester, 1987.

[12] M. Calvé and R. Vaillancourt, *Interpolants for Runge–Kutta pairs of order four and five*, Computing, **45** (1990), pp. 383–388.

[13] G. F. Corliss and Y. F. Chang, *Solving ordinary differential equations using Taylor series*, ACM Trans. Math. Software, **8**(2) (1982), pp. 114–144.

[14] H. T. Davi, *Introduction to nonlinear differential and integral equations*, Dover, New York, 1962.

[15] P. Deuflhard, *Recent progress in extrapolation methods for ordinary differential equations*, SIAM Rev., **27** (1985), pp. 505–535.

[16] J. W. Daniel and R. E. Moore, *Computation and theory in ordinary differential equations*, Freeman, San Francisco, 1970.

[17] J. R. Dormand and P. J. Prince, *A reconsideration of some embedded Runge–Kutta formulae*, J. Comput. Appl. Math., **15** (1986), pp. 203–211.

[18] W. H. Enright, *Continuous numerical methods for ODEs with defect control*, J. Comput. Appl. Math., **125** (2000), pp. 159-170.

[19] W. H. Enright and T. E. Hull, *The test results on initial value methods for non-stiff ordinary differential equations*, SIAM J. Numer. Anal., **13** (1976), pp. 944–961.

[20] A. A. Frost and R. G. Pearson, *Kinetics and mechanism*, rev. ed., Wiley, New York, 1961.

[21] G. Galimberti and V. Pereyra, *Solving confluent Vandermonde systems of Hermite type*, Numer. Math, **18** (1971), pp. 44–60.

[22] C. W. Gear, *The numerical integration of ordinary differential equations*, Math. Comp., **21** (1967), pp. 146–156.

[23] C. W. Gear, *Numerical initial value problems in ordinary differential equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[24] G. H. Golub and C.F. Van Loan, *Matrix computations*, 3rd edition, The Johns Hopkins University Press, Baltimore, MD, 1996.

[25] E. Hairer, S. P. Nørsett and G. Wanner, *Solving ordinary differential equations I. Nonstiff problems*, Section III.8, Springer-Verlag. Berlin, 1987.

[26] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal., **9** (1972), pp. 603–637.

[27] F. T. Krogh, *VODQ/SVDQ/DVDQ—variable order integrators for the numerical solution of ordinary differential equations*, TU Doc. No. CP-2308, NPO-11643, May 1969, Jet Propulsion Laboratory, Pasadena, CA.

[28] F. T. Krogh, *Changing stepsize in the integration of differential equations using modified divided differences*, in Proc. Conf. on the Numerical Solution of Ordinary Differential Equations, University of Texas at Austin 1972 (Ed. D.G. Bettis), Lecture Notes in Mathematics No. 362, Springer-Verlag, Berlin, 22–71, 1974.

[29] F. T. Krogh, *Variable order integrators for the numerical solution of ordinary differential equations.* Jet Propulsion Laboratory Tech. Memo., California Institute of Technology, Pasadena, 1969.

[30] J. D. Lambert, *Computational methods in ordinary differential equations*, Ch. 5, London, Wiley, 1973.

[31] J. D. Lambert, *Numerical methods for ordinary differential systems*, London, Wiley, 1991.

[32] T. Nguyen-Ba and R. Vaillancourt, *Hermite–Birkhoff differential equation solvers*, Scientific Proceedings of Riga Technical University, 5-th series: Computer Science, 46-th thematic issue, **21** (2004), pp. 47–64.

[33] T. Nguyen-Ba and R. Vaillancourt, *Hermite–Birkhoff–Obrechkoff 3-stage 6-step ODE solver of order 14*, Can. Appl. Math. Quarterly, **13**(2) (2005), pp. 151–181.

[34] T. Nguyen-Ba, H.Yagoub, S. J. Desjardins and R. Vaillancourt, *Variable-step variable-order 4-stage Hermite-Birkhoff-Obrechkoff ODE solver of order 5 to 14*, Scientific Proceedings of Riga Technical University, in series "Computer Science" **30** (48) (2006), pp. 53–80.

[35] T. Nguyen-Ba, H. Yagoub, Y. Li and R. Vaillancourt, *Variable-step variable-order 3-stage Hermite-Birkhoff ODE solver of order 5 to 15*, Can. Appl. Math. Quarterly, 14(1) (2006), pp. 43-69.

[36] T. Nguyen-Ba, H.Yagoub, Y. Zhang and R. Vaillancourt, *Variable-step variable-order 3-stage Hermite-Birkhoff-Obrechkoff ODE solver of order 4 to 14*, Can. Appl. Math. Quarterly, 14(4) (2006), pp. 413-437.

[37] A. Nordsieck, *On numerical integration of ordinary differential equations*, Math. Comp., **16** (1962), pp. 22–49.

[38] L. R. Petzold, *A description of DASSL: A differential/algebraic system solver*, in Proceedings of IMACS World Congress, Montréal, Canada, 1982.

[39] P. J. Prince and J. R. Dormand, *High order embedded Runge–Kutta formulae*, J. Comput. Appl. Math., **7**(1) (1981), pp. 67–75.

[40] L. F. Shampine and L. S. Baca, *Fixed versus variable order Runge–Kutta*, ACM Trans. Math. Software, **12**(1) (1986), pp. 1–23.

[41] L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, Freeman, San Francisco, 1975.

[42] L. F. Shampine and M. W. Reichelt, *The Matlab ODE suite*, SIAM J. Sc. Comp., **18**(1) (1997), pp. 1–22.

[43] P. W. Sharp, *Numerical comparison of explicit Runge–Kutta pairs of orders four through eight*, Trans. on Mathematical Software, **17** (1991), pp. 387–409.

[44] M. Sofroniou and G. Spaletta, *Precise numerical computation*, J. Log. Algebr. Program, **64**(1) (2005), pp. 113-134.

[45] Y. Zhang, *Variable-step variable-order 3-stage Hermite–Birkhoff–Obrechkoff ODE Solver of order 4 to 14 with a C program*, M.Sc. Thesis, University of Ottawa, Ottawa ON K1N 6N5, June 2007.

[46] J. A. Zonneveld, *Automatic numerical integration*, Mathematical Centre Tracts No. 8, Mathematisch Centrum, Amsterdam (1964).