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

Yu Zhang

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Variable-step Variable-order 3-stage Hermite-Birkhoff-obrechkoff Ode Solver of Order 4 to 14 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. B. Dionne

Dr. V. Leblanc

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–OBRECHKOFF ODE SOLVER OF ORDER 4 TO 14 WITH A C PROGRAM

By
Yu Zhang, B.Sc.
June 2007

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: 978-0-494-32493-6
Our file Notre référence
ISBN: 978-0-494-32493-6

Abstract

Variable-step variable-order 3-stage Hermite–Birkhoff–Obrechkoff methods of order 4 to 14, denoted by HBO(4-14)3, are constructed for solving nonstiff systems of first-order differential equations of the form $y' = f(x, y)$, $y(x_0) = y_0$. These methods use y' and y'' as in Obrechkoff's method. 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 Vandermonde-type systems. Fast algorithms are developed for solving these systems to obtain Hermite–Birkhoff interpolation polynomials in terms of generalized Lagrange basis functions. The new methods have larger regions of absolute stability than Adams–Bashforth–Moulton methods of comparable orders in PECE mode. The order and stepsize of these methods are controlled by four local error estimators. When programmed in Matlab, HBO(4-14)3 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 function evaluations, CPU time, and maximum global error. It is also superior to the variable-step 3-stage HBO(14)3 of order 14 on some problems. When programmed in C, HBO(4-14)3 is superior to the Dormand–Prince Runge–Kutta nested pair DP(8,7)13M in solving expensive equations over a long period of time. HBO(4-14)3 has been implemented in C. The C program for HBO(4-14)3 is an important contribution of this thesis.

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

I would like to express my deep appreciation to my parents who have supported and encouraged me during my graduate study.

I would also like to thank my friends who helped me and gave me suggestions when I met problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Idea about HBO(4-14)3

The 3-stage variable-step (VS) HBO(14)3 of order 14 constructed in [28] is extended to variable-step variable-order (VSVO) 3-stage Hermite–Birkhoff–Obrechkoff methods of order 4 to 14, denoted by HBO(4-14)3. Both of these methods are designed for solving nonstiff systems of first-order initial value problems (IVPs) of the form

$$y' = f(x, y), \qquad y(x_0) = y_0, \qquad \text{where } ' = \frac{d}{dx}. \tag{1}$$

HBO(4-14)3 is based on variable-order explicit multistep methods [32] and a 3-stage Runge–Kutta method of order 3. The method uses Hermite–Birkhoff interpolation polynomials, and the first and second order derivatives of y at step points as is done in Obrechkoff methods; The name of the method comes from the names of these two methods. Let $y'(x) = f(x, y)$ and

$$y''(x) = \frac{d}{dx} f(x, y(x)) = f_x(x, y(x)) + f_y(x, y(x)) f(x, y(x))$$

be the first and second order derivatives of $y(x)$, respectively. HBO methods are useful when the derivative y'' can be obtained recursively or symbolically. There are many such problems, for instance in dynamical systems [3], [33]. HBO(4-14)3 is self starting with HBO(4)3.

Multiple-step methods use information from the current point and several previous points to get the solution at the next point. HBO(4-14)3 belongs to multiple-step methods in that it requires values at previous points. It also belongs to one-step methods since it uses off-step points. Moreover, like Runge–Kutta methods, HBO(4-14)3 uses derivative evaluations at points between step points. Hermite-Birkhoff interpolation polynomials and Obrechkoff methods are linked by the values at off-step points which are obtained by means of predictors using values at previous points.

Milne [26] was perhaps the first to advocate the use of multiderivative, multistep Obrechkoff formulas for the numerical solution of differential equations. More recently, a new form of the classical Adams–Cowell methods and multiderivative, multistep methods was introduced by Huang and Innanen [19]. Some of these methods have a larger stability interval and a smaller truncation error than classical multistep methods.

In current scientific computation, there exists several methods to solve nonstiff systems of first-order differential equations. Variable-order, variable-step (VSVO) Adams–Bashforth–Moulton multistep methods of order 1 to 14 are widely used. They were implemented by Gear [14], [15]; Krogh [22]; and, up to order 13, by Shampine [34] in Matlab's `ode113` [2], [35]. The codes DVDQ of Krogh [23] and DIFSUB of Gear [15] prompted the recognition of the effectiveness of a variable-order, variable-step formulation of Adams methods. High-order solvers appear to be more efficient than lower-order ones when the equation is expensive to evaluate.

Multistep- and Runge–Kutta-type order conditions which are reorganized into linear Vandermonde-type systems are constructed by forcing a Taylor expansion of the numerical solution to agree with the exact solution. New fast algorithms are introduced to obtain the solution of these systems as generalized Lagrange basis functions.

The variable-order HBO(4-14)3 has the remarkable property that the norm of the principal local truncation error coefficient decreases rapidly as the order increases.

The competitiveness of HBO(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 [16, p. 187]). By taking the special structure of these systems into account,

in contrast with the 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.

Other HB methods of order 9, 10 and 11 have been studied in [27]. Three-stage HB(5-15)3 of order 5 to 15 and three-stage HBO(5-14)3 of order 5 to 14 can be found in [29] and [30], respectively.

1.2 Goals, Approaches and Results

The goal of this thesis is to show that HBO(4-14)3 has better performance than the multistep Adams–Bashforth–Moulton pairs ABM of orders 1 to 13 and the 13-stage Dormand–Prince Runge–Kutta nested pair DP(8,7)13M of orders 8 and 7 for solving some problems which are often used to test higher-order ODE solvers. Also, HBO(4-14)3 performs better than HBO(14)3 on some of these problems. The number of function evaluations, the CPU time, and the accuracy are the three key points to evaluate performance for HBO(4-14)3, ABM and DP(8,7)13M.

For both HBO(4-14)3 and DP(8,7)13M, three operations are needed to compare their performance. First, we implement HBO(4-14)3 and DP(8,7)13M in C. Second, we determine the number of function evaluations, the CPU time, and the maximum global error (MGE)used by each method in the problems considered. Here the CPU time is obtained by using Matlab's `polyfit`. Third, we compute the CPU percentage efficiency gain (CPU PEG). Some figures and tables are produced from the data collected through previous steps and used to compare the results. A similar approach is used to compare HBO(4-14)3 and ABM except that they are implemented in Matlab.

The results show that HBO(4-14)3 has smaller error constants and larger intervals of absolute stability than Adams–Bashforth–Moulton methods of comparable orders in PECE mode. When programmed in Matlab, HBO(4-14)3 requires fewer function evaluations, uses less CPU time, and has a higher accuracy than `ode113`. The VSVO HBO(4-14)3 was also compared with HBO(14)3 of order 14 [28] and shown to perform better on some problems. When programmed in C, HBO(4-14)3 is superior to DP(8,7)13M in solving costly equations over a long period of time.

Chapter 2 introduces the numerical methods for solving ordinary differential equations and concepts related to HBO(4-14)3. Chapter 3 explains in detail the implementation of the VSVO HBO(4-14)3. Chapter 4 describes the C program for HBO(4-14)3 and presents the numerical results obtained with the Matlab and C programs on several test problems. Chapter 5 summarizes the conclusions of the thesis, explains where improvement could be made and lists future work. Appendix A lists the algorithms, Appendix B describes Matlab programming, and Appendix C contains the C program.

Chapter 2

Solvers for Nonstiff ODEs

2.1 Different Types of Methods

Generally speaking, numerical schemes for solving ODEs are classified into two categories: one-step methods and multiple-step methods. With a one-step method consisting of a number of stages, such as Euler's and Runge-Kutta methods, the solution at the next point is constructed from its value at the current point and at intermediary points (called off-step points) for each step. In contrast, multiple-step methods use information from several previous points to get the solution at the next point. For example, Adams-Bashforth methods and Adams-Moulton methods are multiple-step methods. HBO(4-14)3 belongs to multiple-step methods because it requires values at previous points and to one-step methods because it uses off-step points.

2.2 Runge–Kutta Methods

Runge–Kutta methods were originally proposed and developed around 1900 by C. Runge and M. W. Kutta. There are important implicit and explicit iterative methods which are used to approximate the solutions of ordinary differential equations. The idea of Runge–Kutta methods is to allow multiple evaluations of the function f at each step by extending Euler's method. Runge–Kutta methods belong to one-step methods because they only need the information from the current point to compute

f at successive intermediary points, and to compute the estimate of the solution at the next point. Only one initial point, denoted by (x_0, y_0), is used to compute the next point denoted (x_1, y_1) or more generally, y_i is used to compute y_{i+1}. The stepsize for this method is constant. By using this method, we can get the approximations y_{n+1} for $y(x_{n+1})$ according to the previously given approximation y_n. The coefficients of the method are free parameters which satisfy a Taylor series expansion through some order in the time step h. Often, a Runge–Kutta method is concisely represented by its Butcher tableau. Traditionally, the classic fourth-order Runge–Kutta method has been the most commonly used ODE solver. It is usually called "RK4" or "the Runge–Kutta method", given by the following formulas:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

where

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + h/2, y_n + (h/2)k_1), \\ k_3 &= f(x_n + h/2, y_n + (h/2)k_2), \\ k_4 &= f(x_n + h, y_n + hk_3). \end{aligned}$$

Here, k_1, k_2, k_3 and k_4 are four slopes. In more detail, k_1 is the slope at the beginning, x_n, of the interval; k_2 is an approximate slope at the midpoint of the interval $(x_n, x_n + h)$. The new value $y_n + (h/2)k_1$ of y at the point $x_n + h/2$ is computed. Then k_3 is another approximate slope at the midpoint of the interval $(x_n, x_n + h)$. However, the difference from the previous one is that it uses the slope k_2 rather than k_1 to compute the new y-value $y_n + (h/2)k_2$. Finally, k_4 is the slope at the end of the interval $(x_n, x_n + h)$ where the y-value is determined by $y_n + hk_3$. The classic Runge–Kutta method uses an average of these four slopes to compute the y-value y_{n+1}, namely,

$$\text{Average slope} = \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

More weight is given to the slopes k_2 and k_3 which are the slopes at the midpoint of the interval $(x_n, x_n + h)$.

Explicit s-stage Runge–Kutta methods can be given by the formulas

$$\begin{aligned}
k_1 &= f(x_n, y_n) \\
k_2 &= f(x_n + c_2 h, y_n + a_{21} h k_1) \\
k_3 &= f(x_n + c_3 h, y_n + a_{31} h k_1 + a_{32} h k_2) \\
&\vdots \\
k_s &= f(x_n + c_s h, y_n + a_{s1} h k_1 + a_{s2} h k_2 + \cdots + a_{s,s-1} h k_{s-1}) \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i.
\end{aligned}$$

The coefficients in these formulas can be conveniently arranged in a Butcher tableau as given in Table 1.

Table 1: The Butcher tableau of an explicit Runge–Kutta method.

c_1					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
$\vdots$	$\vdots$		$\ddots$		
c_s	a_{s1}	a_{s2}	$\cdots$	$a_{s,s-1}$	
	b_1	b_2	$\cdots$	b_{s-1}	b_s

A Runge–Kutta method is explicit if the matrix $A = (a_{ij})$ is strickly lower triangular. It is semi-implicit if A is lower triangular, otherwise it is said to be implicit.

"Three-stage" means that the number of evaluations of the function f per step is equal to 3. Two common three-stage third-order Runge–Kutta methods are Heun's third-order formula and Kutta's third-order rule. The coefficients in Heun's third-order formula are arranged in the Butcher tableau given in Table 2. The coefficients in Kutta's third-order formula are arranged in the Butcher tableau given in Table 3. Order conditions for Runge–Kutta methods are generated through Taylor expansions.

Total derivatives are denoted as follows:

$$f^{(q)}(x, y(x)) \equiv \frac{d^q}{dx^q} f(x, y(x)), \quad q = 1, 2, \ldots, p-1.$$

Table 2: The Butcher tableau of Heun's third-order method.

0			
1/3	1/3		
2/3	0	2/3	
	1/4	0	3/4

Table 3: The Butcher tableau of Kutta's third-order method.

0			
1/2	1/2		
1	-1	2	
	1/6	2/3	1/6

As an example, let us consider the general third-order Runge–Kutta method. The coefficients in this method are as in Table 1. By setting $p = 3$, we can write the expansion of the exact solution in the form

$$y(x_{n+1}) = y(x_n) + hf(x_n, y_n) + \frac{1}{2}h^2 f^{(1)}(x_n, y_n) + \frac{1}{6}h^3 f^{(2)}(x_n, y_n) + O(h^4).$$

Let

$$F = f^{(1)}(x, y(x, y)) = f_x + ff_y,$$

$$G = f_{xx} + 2ff_{xy} + f^2 f_{yy}.$$

Thus,

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

As was described for explicit Runge–Kutta methods, k_1,k_2 and k_3 are of the form

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + c_2 h, y_n + a_{21} h k_1), \\ k_3 &= f(x_n + c_3 h, y_n + a_{31} h k_1 + a_{32} h k_2). \end{aligned} \tag{3}$$

Expanding k_2 in a Taylor series about the point (x_n, y_n), we obtain

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

and substituting f for k_1, and F and G and for the two bracketed expressions in the above expression for k_2, we get

$$k_2 = f + hc_2F + \frac{1}{2}h^2c_2^2G + O(h^3). \tag{4}$$

Using a similar strategy for k_3, we get

$$k_3 = f + hc_3F + h^2\left(c_2a_{32}Ff_y + \frac{1}{2}c_3^2G\right) + O(h^3). \tag{5}$$

Since

$$y_{n+1} - y_n = h\sum_{i=1}^{3} b_ik_i,$$

substituting equations (3), (4) and (5) into the above equation, we obtain the following equation:

$$y_{n+1} = y_n + h(b_1 + b_2 + b_3)f + h^2(b_2c_2 + b_3c_3)F + \frac{1}{2}h^3[2b_3c_2a_{32}Ff_y + (b_2c_2^2 + b_3c_3^2)G] + O(h^4). \tag{6}$$

The coefficients must satisfy the following constraints in order to match the expansions (2) and (6):

$$\begin{aligned} b_1 + b_2 + b_3 &= 1, \\ b_2c_2 + b_3c_3 &= 1/2, \\ b_2c_2^2 + b_3c_3^2 &= 1/3, \\ b_3c_3a_{32} &= 1/6, \end{aligned}$$

which are called order conditions for third-order explicit Rung–Kutta methods.

HBO(4-14)3 combines a 3-stage Runge–Kutta method with a multistep Obrechkoff method and uses Taylor expansion to get order conditions as Runge-Kutta methods.

2.3 FSAL Methods

First Same As Last (FSAL) methods are a special class of explicit Runge–Kutta methods. The reason why it is called First Same As Last is that the y value at the

Table 4: The Butcher tableau of a FSAL method.

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
$\vdots$			$\ddots$		
c_{s-1}	$a_{s-1,1}$	$a_{s-1,2}$	$\cdots$	$a_{s-1,s-2}$	
1	b_1	b_2		b_{s-2}	b_{s-1}
	b_1	b_2		b_{s-2}	b_{s-1}

end of one integration step is the same as the first y value at the next integration step. The Butcher tableau of a FSAL method can be found in Table 4.

The difference between FSAL and non-FSAL explicit Runge–Kutta methods is that the last formula evaluation of a FSAL method needs satisfy the following conditions:

$$a_{si} = b_i \quad \text{for} \quad i = 1, \ldots, s-1,$$
$$b_s = 0.$$

2.4 Dormand-Prince Pair of Order 8 and 7

The 13-stage Runge–Kutta pair DP(8,7)13M of Dormand-Prince is a FSAL scheme since its coefficients are given by a Butcher tableau like Table 4. The reason why this method is called DP(8,7)13M is that the coefficients set used to advance integration is of order 8 and the error estimation is of order 7. DP(8,7)13M was implemented in C and its performance was compared with HBO(4-14)3 also implemented in C.

2.5 ABM and PC Methods

The multistep Adams-Bashforth-Moulton (ABM) methods are different from one-step Runge–Kutta methods. The fourth-order ABM method uses the values y_i and $f(x_i, y_i)$ at the four points (x_{i-3}, y_{i-3}), (x_{i-2}, y_{i-2}), (x_{i-1}, y_{i-1}), (x_i, y_i) to generate

y_{i+1} and $f(x_{i+1}, y_{i+1})$ for $i = 3, 4, 5, \ldots$. So, it is a four-step method. Usually, multistep methods combine an explicit Adams–Bashforth method used as a predictor and an implicit Adams–Moulton method used as a corrector. The idea of a Predictor-Corrector scheme is that a numerical method for an ordinary differential equation generates an approximate solution step-by-step in discrete increments across the interval of integration, in effect producing a discrete sample of approximate values of the solution $y(x_i)$. At each step of a predictor-corrector method, the predictor supplies an initial guess for the next solution value. The corrector which is a more stable and more accurate method is then used to improve the initial guess. The corrector may also be iterated to improve the solution [15].

The following existence and uniqueness theorem is tantamount in numerical solutions of ODE's, since, otherwise, no numerical solution can be trusted, if it can be computed at all.

Let $f(x, y)$, where $f : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$, be continuous over the region $R = [a, b] \times \mathbb{R}^m$ and satisfy a Lipschitz condition over R with respect to the variable y [25]:

$$\|f(x, y) - f(x, z)\| < M\|y - z\|, \quad \text{for} \quad (x, y), (x, z) \in R,$$

where the constant M is called the Lipschitz constant for f. Then the initial value problem $y' = f(x, y)$ for $y(x_0) = y_0$ with $x_0 \in [a, b]$ admits a unique solution over $[a, b]$.

For $x_{i+1} = x_i + h$, with constant step h, the 4th-order Adams–Bashforth–Moulton method uses the following formulas to calculate y_{i+1}:

Predictor: $y_{i+1}^P = y_i^C + \frac{h}{24}\left(-9f_{i-3}^C + 37f_{i-2}^C - 59f_{i-1}^C + 55f_i^C\right)$,

Corrector: $y_{i+1}^C = y_i^C + \frac{h}{24}\left(f_{i-2}^C - 5f_{i-1}^C + 19f_i^C + 9f_{i+1}^P\right)$,

for $i = 3, 4, \ldots$. Here the predicted values of y_i and f_i are denoted by y_i^P and f_i^P. The corrected values of y_i and f_i are denoted by y_i^C and f_i^C. The predictor is an Adams–Bashforth method. The predictor uses the information at the points (x_{i-3}, f_{i-3}), (x_{i-2}, f_{i-2}), (x_{i-1}, f_{i-1}) and (x_i, f_i) to make the Lagrange polynomial approximation for $f(x, y(x))$. The information at the current point, x_i, and the three previous points, x_{i-1}, x_{i-2}, x_{i-3}, is used to predict the value y_{i+1}^P at the next point, x_{i+1}. The corrector

is an Adams–Moulton method which uses the value y_{i+1}^P from the predictor step. It is easy to notice that there are three separate processes occurring in a Predictor-Corrector method. The first process is the predictor step and we call it P. The second process is the evaluation of the function $f_{i+1}^P = f(x_{i+1}, y_{i+1}^P)$ and we call it E. The third step is the corrector and we call it C. In this case, the Predictor-Corrector is said to be in the Predictor-Evaluation-Corrector (PEC) mode. Usually, a final evaluation yields a better method than PEC; so the usual strategy is Predictor-Evaluation-Corrector-Evaluation (PECE).

In this thesis, ABM was implemented in `ode113` of Matlab and the performance of ode113 was compared with HBO(4-14)3 also implemented in Matlab.

2.6 Stability Analysis

An iteration scheme for approximating an ODE should be stable. Stability means that small perturbations in the initial conditions produce correspondingly small changes in the subsequent approximations. A linear test equation is usually used to verify the stability.

Numerical methods can be written in the general form

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\phi_f(y_{n+k}, y_{n+k-1}, \ldots, y_n, x_n; h), t$$

where the subscript f to ϕ indicates the dependence of ϕ on the function $f(x, y)$. Let us now apply the above equation, with its small nonvanishing parameter $h > 0$, to the linear test equation

$$y' = \lambda y, \tag{7}$$

where λ is the complex number

$$\lambda = \alpha + i\beta$$

in the left half space, that is, $\alpha < 0$.

The region of absolute stability, R, is the set of numbers

$$\widetilde{h} = h\lambda = h\alpha + ih\beta$$

in the complex plane for which the numerical solution y_n of (7) goes to zero as n goes to infinity. The boundary of R cuts the real axis at $\alpha < 0$ and at the origin. The interval $[\alpha, 0]$ is called the interval of absolute stability.

For a system of ODE, $y' = f(x, y)$ in $\mathbb{R}^m$, the m eigenvalues, $\lambda_1, \lambda_2, \ldots, \lambda_m$, of the $m \times m$ Jacobian matrix df/dy may be complex. When ordered by their real parts,

$$\alpha_m \leq \alpha_{m-1} \leq \cdots \leq \alpha_1 < 0,$$

the interval of absolute stability is determined by α_m.

For a fair comparison of the performance of different ODE solvers, the region of absolute stability is divided by the number of function evaluations per step, thus giving the scaled region of absolute stability.

2.7 VSVO Algorithms

Variable-step, variable-order, or VSVO, algorithms are constructed for varying the stepsize and order of specific methods. The VSVO algorithms are composed of the following: a family of methods; a starting procedure; a local error estimator; a strategy to decide when to change stepsize or order; and a technique for changing the stepsize or order.

For example, a family of methods can be composed of ABM methods of orders 1 to 13 or 14 in the PECE mode. Such a VSVO algorithm always starts with a one-step ABM pair. The value of the starting stepsize is evaluated and computed by the method itself.

The strategy of VSVO algorithms is the following: suppose the algorithm is working with a k-th order ABM method. Let E_k be the norm of the local error estimate for this k-th order ABM method at x_{n+1}, and let τ be the tolerance which is defined by the user. To accept the step from x_n to x_{n+1}, the following condition needs to be satisfied

$$E_k \leq \tau.$$

In PECE mode, this estimate is computed after the corrector (C) stage. If the above condition failed, the step from x_n to x_{n+1} is aborted and there is no need to access

the final evaluation (E) stage. Let us suppose the above condition is satisfied. Before going to the next step, we need to find out what should be the stepsize and order at the next step. Let h_k be the stepsize used with the k-th order method on the step from x_n to x_{n+1}. To choose the stepsize, we need to find the maximum stepsize $\widetilde{h}_k$ that could have been used on the just completed step from x_n to x_{n+1}, and use it as the stepsize at the next step. Three stepsizes h_{k-1}, h_k and h_{k+1} can be computed by using ABM methods of orders $k-1$, k and $k+1$ respectively. The greatest one will be the stepsize for the new step. Also, three error estimates E_{k-1}, E_k and E_{k+1} can be generated corresponding to the order $k-1$, k and $k+1$. The order k which produces the maximum stepsize will be the order for the new step. Note that it is possible to retain the same order at the next step, but with a different stepsize.

The technique for changing the stepsize is straightforward. The current stepsize is computed by a specific formula defined by the user. If the condition $E_k \leq \tau$ is satisfied, the step to x_{n+1} is accepted and a new stepsize will be computed according to the same formula. Otherwise, the step to x_{n+1} is rejected. We need to return to the previous step with a smaller stepsize by multiplying the current stepsize by a fraction smaller than one. If the ABM methods are expressed in terms of backward differences, the technique for changing the order is easy to understand. The order can be reduced from k to $k-1$ by removing the k-th backward difference. The order can be increased from k to $k+1$ by retaining all the back data at the completion of the step to form the $(k+1)$-th difference.

2.8 LU Factorization and Vandermonde Systems

Solving linear systems, $Ax = b$, is a central problem in scientific computation. Suppose we want to find the solution of the system $Ax = b$. If we can construct a unit lower triangular matrix L and an upper triangular matrix U, which decomposes A into the form $A = LU$, the solution of this system can be found by a two-step process

$$Ly = b \quad \text{followed by} \quad Ux = y,$$

which implies

$$Ax = LUx = Ly = b.$$

Once L and U are known, then we can solve the system $Ax = b$ via the triangular systems $Ly = b$ and $Ux = y$. In this case, the first thing is to find L and U.

Generally speaking, suppose $x \in \mathbb{R}^n$ with $x_k \neq 0$ and consider the vector

$$\tau = (0, \cdots, 0, \tau_{k+1}, \cdots, \tau_n)^T,$$

where

$$\tau_i = \frac{x_i}{x_k}, \qquad i = k+1, \cdots, n,$$

and k is the number of successive zeros in the top part of τ. We define the Gaussian transformation:

$$M_k = I - \tau e_k^T,$$

which, explicitly, is

$$M_k = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \\ 0 & & 1 & 0 & & 0 \\ 0 & & -\tau_{k+1} & 1 & & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\tau_n & 0 & \cdots & 1 \end{bmatrix}.$$

The matrix M_k is unit lower triangular. After computing $M_1, \ldots, M_{k-1}$, we have the upper triangular matrix U,

$$U = M_{k-1} \cdots M_1 A,$$

and the lower triangular matrix L,

$$L = M_1^{-1} \cdots M_{k-1}^{-1}.$$

The above process is called the LU factorization. Its computational complexity is of the order $O(n^3)$. A faster algorithm is required for HBO(4-14)3 to be competitive with DP(8,7)13M.

Suppose $x \in \mathbb{R}^{n+1}$. A matrix $V \in \mathbb{R}^{(n+1)\times(n+1)}$ of the form

$$V = V(x_0, \cdots, x_n) \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{bmatrix}$$

is said to be a Vandermonde matrix. The system $Vz = b$ is called a Vandermonde system. Define the lower bidiagonal matrix $L_k(\alpha) \in \mathbb{R}^{(n+1)\times(n+1)}$ by

$$L_k(\alpha) = \begin{bmatrix} I_k & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & -\alpha & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & -\alpha & 1 \end{bmatrix}$$

and the diagonal matrix D_k by

$$D_k = \text{diag}\,(1, \cdots, 1, x_{k+1} - x_0, \cdots, x_n - x_{n-k-1})$$

where k is the number of ones before $x_{k+1} - x_0$. We define the upper triangular matrix by

$$U^T = D_{n-1}^{-1} L_{n-1}(1) \cdots D_0^{-1} L_0(1)$$

and the unit lower triangular matrix by

$$L^T = L_0(x_0)^T \cdots L_{n-1}(x_{n-1})^T.$$

The solution to the system $Vz = b$ is given by

$$\begin{aligned} z &= V^{-1}b = U(Lb) \\ &= (L_0(1)^T D_0^{-1} \cdots L_{n-1}(1)^T D_{n-1}^{-1})(L_{n-1}(x_{n-1}) \cdots L_0(x_0)b). \end{aligned}$$

HBO(4-14)3 uses a similar idea but with different L_k. More details can be found in Sections 3.4 and 3.5. This factorization is of order $O(n^2)$ and makes HBO(4-14)3 competitive.

Chapter 3

VSVO HBO(4-14)3

3.1 General and Particular VS HBO(4-14)3

A Hermite–Birkhoff–Obrechkoff (HBO) method is a **general** variable-step HBO method if the backstep and off-step points are variable parameters. In order to perform the integration step from x_n to x_{n+1}, the predictor P_2, P_3, P_4 and the integration formula IF need to be computed as below:

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

$$y_{n+c_2} = y_n + h_{n+1}\left(a_{21}f_{n+c_1} + \sum_{j=1}^{\lfloor (p-3)/2 \rfloor} \beta_{2j} f_{n-j}\right) + h_{n+1}^2\left(\sum_{j=0}^{\lfloor (p-4)/2 \rfloor} \gamma_{2j} f'_{n-j}\right). \quad (8)$$

(P_3) A Hermite–Birkhoff polynomial of degree $(p-1)$ is used as predictor P_3 to obtain y_{n+c_3} to order $(p-2)$,

$$y_{n+c_3} = y_n + h_{n+1}\left(\sum_{j=1}^{2} a_{3j} f_{n+c_j} + \sum_{j=1}^{\lfloor (p-3)/2 \rfloor} \beta_{3j} f_{n-j}\right) + h_{n+1}^2\left(\sum_{j=0}^{\lfloor (p-4)/2 \rfloor} \gamma_{3j} f'_{n-j}\right). \quad (9)$$

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

$$y_{n+1} = y_n + h_{n+1}\left(\sum_{j=1}^{3} b_{1j} f_{n+c_j} + \sum_{j=1}^{\lfloor (p-3)/2 \rfloor} \beta_{1j} f_{n-j}\right) + h_{n+1}^2\left(\sum_{j=0}^{\lfloor (p-4)/2 \rfloor} \gamma_{1j} f'_{n-j}\right). \quad (10)$$

(P_4) A Hermite–Birkhoff polynomial of degree p is used as step control predictor P_4 to obtain $\widetilde{y}_{n+1}$ to order $(p-2)$,

$$\widetilde{y}_{n+1} = y_n + h_{n+1}\left(\sum_{j=1}^{2} a_{4j} f_{n+c_j} + a_{43} f_{n+1} + \sum_{j=1}^{\lfloor (p-3)/2 \rfloor} \beta_{4j} f_{n-j}\right) + h_{n+1}^2\left(\sum_{j=0}^{\lfloor (p-4)/2 \rfloor} \gamma_{4j} f'_{n-j}\right). \quad (11)$$

The "floor" of a real number q, denoted by $\lfloor q \rfloor$, is the largest integer smaller or equal to q. We note that f'_{n+1} is computed only once at x_{n+1}.

The off-step points of HBO(4-14)3 satisfy the following Runge–Kutta-type simplifying conditions:

$$c_i = \sum_{j=1}^{i-1} a_{ij} + \sum_{\ell=1}^{\lfloor (p-3)/2 \rfloor} \beta_{i\ell}, \qquad i = 2, 3. \quad (12)$$

For the general HBO(4-14)3, c_1 is zero for notational simplicity, and c_2 and c_3 are free parameters. They are called off-step points. We do not need backstep points for Runge–Kutta methods which are used in HBO(4-14)3. We only need off-step points which are obtained by removing the backstep points term from formulas (8), (9) and

(10). We then get the formulas for 3-stage Runge–Kutta methods of the form

$$y_{n+c_2} = y_n + h_{n+1}(a_{21}f_{n+c_1}),$$
$$y_{n+c_3} = y_n + h_{n+1}(\sum_{j=1}^{2} a_{3j}f_{n+c_j}),$$
$$y_{n+1} = y_n + h_{n+1}(\sum_{j=1}^{2} b_{1j}f_{n+c_j} + b_{13}f_{n+c_3}).$$

HBO(4-14)3 is said to be a **particular** variable-step method if the off-step points are fixed. After extensive numerical experimentation, we have obtained that HBO(4-14)3 is simple and uses less computation time when c_1, c_2, c_3 are assigned the values 0, 2/3 and 1 respectively, for simplicity and the addition of only one off-step point. We get a particular variable-step HBO(4-14)3.

From formula (8)–(11), we see that there are three evaluations per step. The first one is to compute y_{n+c_2} from (8), evaluate f_{n+c_2}, and plug it into (9). The second one is to compute y_{n+c_3} from (9), evaluate f_{n+c_3}, and plug it into IF. The last one is to compute y_{n+1} from (10) and evaluate f_{n+1} for P_4 and the next step. At the next step, we substitute the result f_{n+1} from the last step into (8) to compute y_{n+1+c_2}. That is why we choose c_3 to be 1.

3.2 Order Conditions for HBO(4-14)3

As in similar search for ODE solvers, we impose the following Runge–Kutta-type simplifying assumptions [8], [27], [29] on HBO(4-14)3:

$$\sum_{j=1}^{i-1} a_{ij}c_j^k + k!B_i(k+1) = \frac{1}{k+1}c_i^{k+1}, \qquad \begin{cases} i = 2,3, \\ k = 0,1,2,\ldots,p-3, \end{cases} \tag{13}$$

where

$$B_i(j) = \sum_{\ell=1}^{\lfloor (p-3)/2 \rfloor} \left[\beta_{i\ell}\frac{\eta_{\ell+1}^{j-1}}{(j-1)!}\right] + \sum_{\ell=1}^{\lfloor (p-4)/2 \rfloor} \left[\gamma_{i\ell}\frac{\eta_{\ell+1}^{j-2}}{(j-2)!}\right], \qquad \begin{cases} i = 2,3, \\ j = 0,1,2,\ldots,p, \end{cases} \tag{14}$$

with $\eta_{\ell+1}^{-2} = 0$ and $\eta_{\ell+1}^{-1} = 0$ by notation, and

$$\eta_j = -\frac{1}{h_{n+1}}(x_n - x_{n+1-j}) = -\frac{1}{h_{n+1}}\sum_{i=0}^{j-1} h_{n-i}, \qquad j = 2, 3, \ldots, 6. \tag{15}$$

Equation (15) will be frequently used in this thesis.

There remain two sets of equations to be solved:

$$\sum_{i=1}^{3} b_{1i}c_i^k + k!B_1(k+1) = \frac{1}{k+1}, \qquad k = 0, 1, \ldots, p-1, \tag{16}$$

$$\sum_{i=2}^{3} b_{1i}\left[\sum_{j=1}^{i-1} a_{ij}\frac{c_j^{p-2}}{(p-2)!} + B_i(p-1)\right] + B_1(p) = \frac{1}{p!}, \tag{17}$$

where

$$B_1(j) = \sum_{i=1}^{\lfloor (p-3)/2 \rfloor} \beta_{1i}\frac{\eta_{i+1}^{j-1}}{(j-1)!} + \sum_{i=1}^{\lfloor (p-4)/2 \rfloor} \gamma_{1i}\frac{\eta_{i+1}^{j-2}}{(j-2)!}, \qquad j = 1, 2, \ldots, p+1. \tag{18}$$

We note that equations (16), for $k = 0, 1, \ldots, p-2$, are multistep-type order conditions. On the other hand, equation (16) for $k = p-1$ and equation (17) are Runge–Kutta-type order conditions. The numbers $B_1(k)$, $B_2(k)$ and $B_3(k)$ are associated with IF, P_2 and P_3, respectively.

3.3 Vandermonde-type Formulation of HBO(p)3

3.3.1 Integration formula IF

The p-vector of the reordered coefficients of IF in (10),

$$\boldsymbol{u}^1 = [b_{11}, \gamma_{10}, b_{13}, b_{12}, \beta_{11}, \gamma_{11}, \beta_{12}, \gamma_{12}, \beta_{13}, \gamma_{13}, \ldots, \beta_{1,\lfloor (p-3)/2 \rfloor}, \gamma_{1,\lfloor (p-4)/2 \rfloor}]^T,$$

is the solution of the Vandermonde-type system of order conditions

$$M^1\boldsymbol{u}^1 = \boldsymbol{r}^1, \tag{19}$$

where

$$M^1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & \cdots & 1 & 0 \\ 0 & 1 & c_3 & c_2 & \eta_2 & 1 & \cdots & \eta_{\lfloor (p-3)/2 \rfloor + 1} & 1 \\ 0 & 0 & \frac{c_3^2}{2!} & \frac{c_2^2}{2!} & \frac{\eta_2^2}{2!} & \eta_2 & \cdots & \frac{\eta^2_{\lfloor (p-3)/2 \rfloor + 1}}{2!} & \eta_{\lfloor (p-4)/2 \rfloor + 1} \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & \frac{c_3^{p-1}}{(p-1)!} & \frac{c_2^{p-1}}{(p-1)!} & \frac{\eta_2^{p-1}}{(p-1)!} & \frac{\eta_2^{p-2}}{(p-2)!} & \cdots & \frac{\eta^{p-1}_{\lfloor (p-3)/2 \rfloor + 1}}{(p-1)!} & \frac{\eta^{p-2}_{\lfloor (p-4)/2 \rfloor + 1}}{(p-2)!} \end{bmatrix} \tag{20}$$

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

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

The leading error term of IF is

$$\left[b_{13} \frac{c_3^p}{p!} + b_{12} \frac{c_2^p}{p!} + \sum_{j=1}^{\lfloor (p-3)/2 \rfloor} \beta_{1j} \frac{\eta_{j+1}^p}{p!} + \sum_{j=1}^{\lfloor (p-4)/2 \rfloor} \gamma_{1j} \frac{\eta_{j+1}^{p-1}}{(p-1)!} - \frac{1}{(p+1)!} \right] h_{n+1}^{p+1} y_n^{(p+1)}.$$

The detailed structure of columns 5 to p of $M^1 \in \mathbb{R}^{p \times p}$ is as follows:

$$\begin{aligned} M^1(i,j) &= \eta_{\lfloor j/2 \rfloor}^{i-1}/(i-1)!, \\ M^1(i,j+1) &= \frac{d}{d\eta_{\lfloor j/2 \rfloor}} M^1(i,j), \end{aligned} \qquad \begin{cases} i = 1, 2, \ldots, p, \\ j = 5, 7, \ldots, 2\lfloor (p-1)/2 \rfloor + 1, \end{cases}$$

with $j + 1 \leq p$ in the second equation.

3.3.2 Predictor P_2

The $(p-2)$-vector of the reordered coefficients of P_2 in (8),

$$\boldsymbol{u}^2 = [a_{21}, \gamma_{20}, \beta_{21}, \gamma_{21}, \beta_{22}, \gamma_{22}, \beta_{23}, \gamma_{23}, \ldots, \beta_{2,\lfloor (p-3)/2 \rfloor}, \gamma_{2,\lfloor (p-4)/2 \rfloor}]^T,$$

is the solution of the system of order conditions

$$M^2 \boldsymbol{u}^2 = \boldsymbol{r}^2, \tag{21}$$

where

$$M^2 = \begin{bmatrix} 1 & 0 & 1 & 0 & \cdots & 1 & 0 \\ 0 & 1 & \eta_2 & 1 & \cdots & \eta_{\lfloor (p-3)/2 \rfloor + 1} & 1 \\ 0 & 0 & \frac{\eta_2^2}{2!} & \eta_2 & \cdots & \frac{\eta^2_{\lfloor (p-3)/2 \rfloor + 1}}{2!} & \eta_{\lfloor (p-4)/2 \rfloor + 1} \\ \vdots & & & & & & \vdots \\ 0 & 0 & \frac{\eta_2^{p-3}}{(p-3)!} & \frac{\eta_2^{p-4}}{(p-4)!} & \cdots & \frac{\eta^{p-3}_{\lfloor (p-3)/2 \rfloor + 1}}{(p-3)!} & \frac{\eta^{p-4}_{\lfloor (p-4)/2 \rfloor + 1}}{(p-4)!} \end{bmatrix} \tag{22}$$

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

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

The detailed structure of columns 3 to $p-2$ of $M^2 \in \mathbb{R}^{(p-2)\times(p-2)}$ is as follows:

$$\begin{aligned} M^2(i,j) &= \eta^{i-1}_{\lfloor j/2+1 \rfloor}/(i-1)!, \\ M^2(i,j+1) &= \frac{d}{d\eta_{\lfloor j/2+1 \rfloor}} M^2(i,j), \end{aligned} \qquad \begin{cases} i = 1, 2, \ldots, p-2, \\ j = 3, 5, \ldots, 2\lfloor (p-3)/2 \rfloor + 1, \end{cases}$$

with $j+1 \leq p-2$ in the second equation.

A truncated Taylor expansion of the right-hand side of (8) about x_n gives

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

where the coefficients are defined by

$$S_2(j) = M^2(j, 1 : p-2)\boldsymbol{u}^2 = r_2(j) = \frac{c_2^j}{j!}, \qquad j = 1, 2, \ldots, p-2, \tag{23}$$

$$S_2(j) = \sum_{i=1}^{\lfloor (p-3)/2 \rfloor} \beta_{2i} \frac{\eta_{i+1}^{j-1}}{(j-1)!} + \sum_{i=0}^{\lfloor (p-4)/2 \rfloor} \gamma_{2i} \frac{\eta_{i+1}^{j-2}}{(j-2)!}, \qquad j = p-1, p, p+1.$$

We see by (23) that P_2 is of order $p-2$. The leading term of the error of P_2 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 $(p-1)$-vector of the reordered coefficients of P_3 in (9),

$$\boldsymbol{u}^3 = [a_{31}, \gamma_{30}, a_{32}, \beta_{31}, \gamma_{31}, \beta_{32}, \gamma_{32}, \beta_{33}, \gamma_{33}, \ldots, \beta_{3,\lfloor (p-3)/2 \rfloor}, \gamma_{3,\lfloor (p-4)/2 \rfloor}]^T,$$

is the solution of the system of order conditions

$$M^3 \boldsymbol{u}^3 = \boldsymbol{r}^3, \tag{24}$$

where

$$M^3 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & \cdots & 1 & 0 \\ 0 & 1 & c_2 & \eta_2 & 1 & \cdots & \eta_{\lfloor (p-3)/2 \rfloor + 1} & 1 \\ 0 & 0 & \frac{c_2^2}{2!} & \frac{\eta_2^2}{2!} & \eta_2 & \cdots & \frac{\eta^2_{\lfloor (p-3)/2 \rfloor + 1}}{2!} & \eta_{\lfloor (p-4)/2 \rfloor + 1} \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \frac{c_2^{p-2}}{(p-2)!} & \frac{\eta_2^{p-2}}{(p-2)!} & \frac{\eta_2^{p-3}}{(p-3)!} & \cdots & \frac{\eta^{p-2}_{\lfloor (p-3)/2 \rfloor + 1}}{(p-2)!} & \frac{\eta^{p-3}_{\lfloor (p-4)/2 \rfloor + 1}}{(p-3)!} \end{bmatrix} \tag{25}$$

and $\boldsymbol{r}^3 = r_3(1:p-1)$ has components

$$r_3(i) = c_3^i/i!, \qquad i = 1, 2, \ldots, p-2, \tag{26}$$

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

We see by (26) that P_3 is of order $p-2$. The last equation of (24) corresponds to order condition (17).

The detailed structure of columns 4 to $p-1$ of $M^3 \in \mathbb{R}^{(p-1)\times(p-1)}$ is as follows:

$$\begin{aligned} M^3(i,j) &= \eta_{\lfloor j/2 \rfloor}^{i-1}/(i-1)!, \\ M^3(i,j+1) &= \frac{d}{d\eta_{\lfloor j/2 \rfloor}} M^3(i,j), \end{aligned} \qquad \begin{cases} i = 1, 2, \ldots, p-1, \\ j = 4, 6, \ldots, 2\lfloor (p-1)/2 \rfloor, \end{cases}$$

with $j+1 \leq p-1$ in the second equation.

A truncated Taylor expansion of the right-hand side of (9) about x_n gives

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

where the coefficients are given by

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

$$S_3(j) = a_{32}S_2(j-1) + \sum_{i=1}^{\lfloor (p-3)/2 \rfloor} \beta_{3i}\frac{\eta_{i+1}^{j-1}}{(j-1)!} + \sum_{i=1}^{\lfloor (p-4)/2 \rfloor} \gamma_{3i}\frac{\eta_{i+1}^{j-2}}{(j-2)!}, \quad j = p-1, p, p+1.$$

3.3.4 Step control predictor P_4

The p-vector of the reordered coefficients of P_4 in (11) is

$$\widetilde{\boldsymbol{u}}^4 = [a_{41}, \gamma_{40}, \beta_{41}, \gamma_{41}, \beta_{42}, \gamma_{42}, \ldots, \beta_{4,\lfloor (p-3)/2 \rfloor}, \gamma_{4,\lfloor (p-4)/2 \rfloor}, a_{43}, a_{42}]^T.$$

By setting $a_{43} = b_{13} + \omega_3$ and $a_{42} = b_{12} + \omega_2$, $\widetilde{\boldsymbol{u}}^4$ reduces to the $(p-2)$-vector $\boldsymbol{u}^4$ which is the solution of the system of order conditions

$$M^4\boldsymbol{u}^4 = \boldsymbol{r}^4, \tag{27}$$

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

$$r_4(i) = 1/i! - (b_{13} + \omega_3)\frac{c_3^{i-1}}{(i-1)!} - (b_{12} + \omega_2)\frac{c_2^{i-1}}{(i-1)!}, \qquad i = 1, 2, \ldots, p-2.$$

For arbitrary nonzero ω_3 and ω_2, P_4 yields $\widetilde{y}_{n+1}$ to order $(p-2)$. We found that a good experimental choice is $\omega_3 = -0.025$ and $\omega_2 = 0.029$.

The column ordering in matrices (20), (22), and (25) makes it easy to go from order p to order $p+1$ simply by adding a bottom row and a far-right column. This ordering differs from the one used in [28].

3.4 Construction of Elementary Matrices

Consider the matrices

$$M^\ell \in \mathbb{R}^{m_\ell \times m_\ell}, \qquad \ell = 1, 2, 3, 4, \tag{28}$$

of the Vandermonde-type systems (19), (21), (24), and (27), where

$$m_1 = p, \quad m_2 = p-2, \quad m_3 = p-1, \quad m_4 = p-2. \tag{29}$$

A fast solution to these systems in $O(m_\ell^2)$ operations will be achieved by decomposing $(M^\ell)^{-1}$ into the product of lower and upper bidiagonal matrices, one diagonal matrix and one upper tridiagonal matrix.

The purpose of this section is to construct symbolically elementary lower and upper bidiagonal matrixes depending on the parameters of HBO(p)3 which will be used in Section 3.5 to diagonalize M_ℓ, $\ell = 1, 2, 3$.

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 [22], Galimberti and Pereyra [13] and Björck and Elfving [4]. Pivoting is not needed in this decomposition because of the special structure of Vandermonde-type matrices.

3.4.1 Symbolic construction of lower bidiagonal matrices for M^ℓ, $\ell = 1, 2, 3$

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

$$L_k = \begin{bmatrix} I_{k-1} & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & & 0 \\ 0 & 1 & -\tau_{k+1} & & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & -\tau_m \end{bmatrix} \tag{30}$$

where

$$\tau_i = \frac{x_{i-1}}{x_i} = -L_k(i,i), \qquad i = k+1, k+2, \ldots, m, \tag{31}$$

zeros the last $(m-k)$ components, $x_{k+1}, \ldots, x_m$, of $\boldsymbol{x}$.

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

$$L^\ell M^\ell = L_{m_\ell - 1}^\ell \cdots L_4^\ell L_3^\ell M^\ell \tag{32}$$

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

Process 1 *At the k-th 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* $M^{\ell(k-1)}(k+1 : m_l, k)$ *since* $M^\ell(i,k) \neq 0$ *for* $i = k+1, k+2, \ldots, m_l$.

Following the Matlab notation, $A(a : b, c)$ denotes the vector containing the elements of the c-th column from row a to row b of matrix A.

Algorithm 1 in Appendix A describes this process.

3.4.2 Symbolic construction of initializing upper tridiagonal matrices U^ℓ_1 for M^ℓ, $\ell = 1, 2, 3$

The second step in diagonalizing M^ℓ transforms the first two rows of $L^\ell M^\ell$ by right multiplication by an upper tridiagonal matrix U^ℓ_1 such that

$$L^\ell M^\ell U^\ell_1(1 : 2, 1 : m_\ell) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{33}$$

The action of U^ℓ_1 amounts to take the divided difference of the columns of $L^\ell M^\ell$ whose first component is 1 (cf. [4]).

3.4.3 Symbolic construction of upper bidiagonal matrices for M^ℓ, $\ell = 1, 2, 3$

We construct upper bidiagonal matrices U^ℓ_k, $k = 2, 3, \ldots, m_\ell - 1$. With that, the product of $L^\ell M^\ell U^\ell_1$ is the divided differences of order k of the columns k to m_ℓ of the matrices on which they act.

Specifically, consider the two-row matrix:

$$G \equiv L^{\ell}M^{\ell}U_1^{\ell}\cdots U_{k-1}^{\ell}(k:k+1,1:m_{\ell})$$
$$= \begin{bmatrix} y_{k1} & \cdots & y_{k,k-1} & 1 & 1 & \cdots & 1 & 1 \\ y_{k+1,1} & \cdots & y_{k+1,k-1} & y_{k+1,k} & y_{k+1,k+1} & \cdots & y_{k+1,m_{\ell}-1} & y_{k+1,m_{\ell}} \end{bmatrix} \quad (34)$$

and define the upper bidiagonal matrix

$$U_k^{\ell} = \begin{bmatrix} I_{k-1} & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -\sigma_{k+1} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sigma_{k+1} & -\sigma_{k+2} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & 0 & \vdots \\ 0 & 0 & \cdots & 0 & \sigma_{m_{\ell}-2} & -\sigma_{m_{\ell}-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 & \sigma_{m_{\ell}-1} & -\sigma_{m_{\ell}} \\ 0 & 0 & 0 & \cdots & 0 & 0 & \sigma_{m_{\ell}} \end{bmatrix} \quad (35)$$

by means of the divisors

$$\sigma_i = \frac{1}{y_{2,i} - y_{2,i-1}} = U_k^{\ell}(i,i), \qquad i = k+1, k+2, \ldots, m_{\ell}. \quad (36)$$

Then, right multiplying (34) by U_k^{ℓ} returns G with $G_{1,j} = 0 for j = k+1, \ldots, m_{\ell}$ and $G_{2,j} = 1 for j = k+1, \ldots, m_{\ell}$. Then

$$L^{\ell}M^{\ell}U_1^{\ell}\cdots U_{k-1}^{\ell}U_k^{\ell}(k:k+1,1:m_{\ell})$$
$$= \begin{bmatrix} y_{k1} & \cdots & y_{k,k-1} & 1 & 0 & \cdots & 0 & 0 \\ y_{k+1,1} & \cdots & y_{k+1,k-1} & y_{k+1,k} & 1 & \cdots & 1 & 1 \end{bmatrix}. \quad (37)$$

Applying U_k^{ℓ}, $k = 2, 3, \ldots, m_{\ell}-1$, on the right of the upper triangular matrix $L^{\ell}M^{\ell}U_1^{\ell}$, yields the diagonal matrix

$$D^{\ell} = L^{\ell}M^{\ell}U^{\ell} = L_{m_{\ell}-1}^{\ell}\cdots L_4^{\ell}L_3^{\ell}M^{\ell}U_1^{\ell}U_2^{\ell}\cdots U_{m_{\ell}-1}^{\ell} \quad (38)$$

in $(m_{\ell} - 2)$ steps.

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

- $M^{\ell(k-1)} = (L^\ell M^\ell U_1^\ell) U_2^\ell \cdots U_{k-1}^\ell$ *is a diagonal 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* $M^{\ell(k-1)}(k+1, j) - M^{\ell(k-1)}(k+1, j-1) \neq 0$ *for* $j = k+1, k+2, \ldots, m_\ell - 1$.

Algorithm 2 in Appendix A describes this process.

3.5 Fast Solution of Vandermonde-type Systems

For L_k^ℓ and U_k^ℓ, the elementary matrix functions $\ell = 1, 2, 3, 4$ are constructed once at each integration step. The matrices L_k^ℓ are used by the fast Algorithm 3 listed in Appendix A to solve $M^\ell \boldsymbol{u}^\ell = \boldsymbol{r}^\ell$ for $\ell = 1, 2, 3, 4$, in (19), (21), (24) and (27), respectively.

We recall, from (29), that

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

and $M^4 = M^2$. 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, $k = 3, \ldots, m_\ell - 1$, of the form (30) defined by the multipliers where

$$\tau_i = \frac{i+2-k}{\mu_\ell(k)} = -L_k^\ell(i,i), \qquad i = k+1, k+2, \ldots, m_\ell, \tag{39}$$

and

$$\mu_\ell(k) = \begin{cases} M^\ell(2,k), & \text{if } M^\ell(1,k) = 1, \\ M^\ell(3,k), & \text{if } M^\ell(1,k) = 0, \end{cases} \qquad k = 3, 4, \ldots, m_\ell - 1.$$

Left multiplying M^ℓ by L_k^ℓ, $k = 3, \ldots, m_\ell - 1$, produces the upper triangular matrix $L^\ell M^\ell = L_{m_\ell - 1}^\ell \cdots L_3^\ell M^\ell$ of the form (32).

Second, we construct the $m_\ell \times m_\ell$ upper tridiagonal matrix U_1^ℓ which transforms M^ℓ into the matrix of first order divided differences $M^\ell U_1^\ell$ of the columns of M^ℓ whose first component is 1 and where the divisors are taken from the second row of M^ℓ. We note that $U_1^4 = U_1^2$ since $M^4 = M^2$. Table 5 lists the diagonal elements of U_1^ℓ. The

Table 5: The diagonal entries of U_1^ℓ.

i	$U_1^1(i,i)$	$U_1^2(i,i)$, $U_1^4(i,i)$	$U_1^3(i,i)$
1	1	1	1
2	1	1	1
3	$1/c_3$	$1/\eta_2$	$1/c_2$
4	$1/(c_2-c_3)$	1	$1/(\eta_2-c_2)$
5	$1/(\eta_2-c_2)$	$1/(\eta_3-\eta_2)$	1
6	1	1	$1/(\eta_3-\eta_2)$
7	$1/(\eta_3-\eta_2)$	$1/(\eta_4-\eta_3)$	1
8	1	1	$1/(\eta_4-\eta_3)$
9	$1/(\eta_4-\eta_3)$	$1/(\eta_5-\eta_4)$	1
10	1	1	$1/(\eta_5-\eta_4)$
11	$1/(\eta_5-\eta_4)$	$1/(\eta_6-\eta_5)$	1
12	1		$1/(\eta_6-\eta_5)$
13	$1/(\eta_6-\eta_5)$		

nonzero non-diagonal elements of U_1^ℓ satisfy the following three equations:

$$\begin{aligned} U_1^1(i-2,i) &= -U_1^1(i,i), \qquad i=3,7,9,\ldots,13,\\ U_1^1(3,4) &= -U_1^1(4,4),\\ U_1^1(4,5) &= -U_1^1(5,5), \end{aligned} \tag{40}$$

$$U_1^2(i-2,i) = -U_1^2(i,i), \qquad i=3,5,7,\ldots,11, \tag{41}$$

$$\begin{aligned} U_1^3(i-2,i) &= -U_1^3(i,i), \qquad i=3,6,8,\ldots,12,\\ U_1^3(3,4) &= -U_1^3(4,4). \end{aligned} \tag{42}$$

The remaining elements of U_1^ℓ are zero. The first two rows of $M^\ell U_1^\ell$ are as in (33).

For given p and ℓ, U_1^ℓ is an $m_\ell \times m_\ell$ upper tridiagonal matrix whose elements are given by Table 5, (40), (41) and (42).

Third, the elimination procedure of subsection 3.4.3 is used to construct $m_\ell \times m_\ell$ upper bidiagonal matrices U_k^ℓ, $k=2,\ldots,m_\ell-1$, of the form (35) where the divisors are given by

$$\sigma_i = \frac{k}{\mu_\ell(i)-\mu_\ell(i-k)} = U_k^\ell(i,i), \qquad i=k+1,k+2,\ldots,m_\ell. \tag{43}$$

Right multiplying $L^\ell M^\ell$ by U_k^ℓ, $k = 1, \ldots, m_\ell - 1$, produces the diagonal matrix

$$D^\ell = L^\ell_{m_\ell - 1} L^\ell_{m_\ell - 2} \cdots L^\ell_3 M^\ell U^\ell_1 U^\ell_2 \cdots U^\ell_{m_\ell - 1},$$

where

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

and

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

Finally, M^ℓ is decomposed into the product of elementary matrices:

$$M^\ell = \left(L^\ell_{m_\ell - 1} L^\ell_{m_\ell - 2} \cdots L^\ell_3\right)^{-1} D^\ell \left(U^\ell_1 U^\ell_2 \cdots U^\ell_{m_\ell - 1}\right)^{-1}$$

and the solution of $M^\ell \boldsymbol{u}^\ell = \boldsymbol{r}^\ell$ is

$$\boldsymbol{u}^\ell = U^\ell_1 U^\ell_2 \cdots U^\ell_{m_\ell - 1} (D^\ell)^{-1} L^\ell_{m_\ell - 1} L^\ell_{m_\ell - 2} \cdots L^\ell_3 \boldsymbol{r}^\ell, \tag{44}$$

where fast computation goes from right to left.

Process 3 *Procedure* (44) *is implemented in the following two steps:*

Step 1 Algorithm 3 in Appendix A overwrites $\boldsymbol{r}^\ell = r_\ell(1 : m_\ell)$ *with* $U^\ell_2 \cdots U^\ell_{m_\ell - 1} (D^\ell)^{-1} L^\ell_{m_\ell - 1} L^\ell_{m_\ell - 2} \cdots L^\ell_3 \boldsymbol{r}^\ell$ *in* $O(m_\ell^2)$ *operations. The input is* $M = M^\ell$; $m = m_\ell$; $\boldsymbol{r} = \boldsymbol{r}^\ell$; $L_k = L^\ell_k$, $k = 3, 4, \ldots, m_\ell - 1$; $U_k = U^\ell_k$, $k = 2, \ldots, m_\ell - 1$; *and* $D = D^\ell$.

Step 2 For each value of ℓ, *one of the following three computations is performed:*

Case 1 ($\ell = 1$) *The following iteration overwrites* $\boldsymbol{r}^1 = r_1(1 : m_1)$ *with* $U^1_1 \boldsymbol{r}^1$:

$$\begin{aligned}
&r_1(3) = r_1(3) U^1_1(3,3), \\
&r_1(4) = r_1(4) U^1_1(4,4), \\
&r_1(i) = r_1(i) U^1_1(i,i), && i = 5, 7, \ldots, \lfloor (m_1 + 1)/2 \rfloor 2 - 1, \\
&r_1(1) = r_1(1) - r_1(3), \\
&r_1(3) = r_1(3) - r_1(4), \\
&r_1(4) = r_1(4) - r_1(5) && \text{if } m_1 \geqq 5, \\
&r_1(i) = r_1(i) - r_1(i+2), && i = 5, 7, \ldots, \lfloor (m_1 + 1)/2 \rfloor 2 - 3.
\end{aligned}$$

Case 2 ($\ell = 2$) The following iteration overwrites $\boldsymbol{r}^2 = r_2(1 : m_2)$ with $U_1^2 \boldsymbol{r}^2$:

$$\begin{aligned} r_2(i) &= r_2(i) U_1^2(i,i), && i = 3, 5, 7, \ldots, \lfloor (m_2+1)/2 \rfloor 2 - 1, \\ r_2(i) &= r_2(i) - r_2(i+2), && i = 1, 3, 5, \ldots, \lfloor (m_2+1)/2 \rfloor 2 - 3. \end{aligned}$$

Case 3 ($\ell = 3$) The following iteration overwrites $\boldsymbol{r}^3 = r_3(1 : m_3)$ with $U_1^3 \boldsymbol{r}^3$:

$$\begin{aligned} & r_3(3) = r_3(3) U_1^3(3,3), \\ & r_3(i) = r_3(i) U_1^3(i,i), && i = 4, 6, 8, \ldots, \lfloor m_3/2 \rfloor 2, \\ & r_3(1) = r_3(1) - r_3(3), \\ & r_3(3) = r_3(3) - r_3(4) && \text{if } m_3 \geqq 4, \\ & r_3(i) = r_3(i) - r_3(i+2), && i = 4, 6, 8, \ldots, \lfloor m_3/2 \rfloor 2 - 2. \end{aligned}$$

3.6 Algorithm to Advance Integration from x_n to x_{n+1}

A variant of the procedure described in [34] 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 $\widetilde{y}_{n+1,q} := \widetilde{y}_{n+1}$ is the value obtained by the step control predictor P_4 of order $q = p - 2$.

- The stepsize h_{n+1} is obtained by the formula (see [20]):

$$h_{n+1} = \min \left\{ h_{\max}, \beta\, h_n \left(\frac{\text{tolerance}}{\mathrm{E}} \right)^{1/\kappa}, 4\, h_n \right\}, \tag{45}$$

  with $\kappa = p - 1$ and safety factor $\beta = 0.81$.

- 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 (19), (21), (24) and (27).

- The step to x_{n+1} is accepted if 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 step control predictors of order $\rho = q \pm 1$ and $q-2$,

$$\widetilde{y}_{n+1} = y_n + h_{n+1}\left[\sum_{j=1}^{2} a_{4j} f_{n+c_j} + a_{43} f_{n+1} + \sum_{j=1}^{\lfloor(\rho-1)/2\rfloor} \beta_{4j} f_{n-j}\right] + h_{n+1}^2\left[\sum_{j=0}^{\lfloor(\rho-2)/2\rfloor} \gamma_{4j} f'_{n-j}\right],$$

and similar to P_4, are used to produce three values $\widetilde{y}_{n+1,\rho} for \rho = q \pm 1$ and $q-2$. These values are used to control the order and stepsize by means of the following three maximum norms,

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

which estimate the local error at x_{n+1} had the step to x_{n+1} been taken at orders $q \pm 1$ and $q-2$. These three quantities are combined with E to select the order and step size as follows 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. When the order q of P_4 is 12, E_{+1} is not available; Thus, the order is lowered if

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

When $q = 2$, 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, $\kappa_{\text{new}} = \kappa_{\text{old}} - 1$ and $\text{E} = \text{E}_{-1}$. The stepsize h_{n+1} is then controlled by formula (45).

3.7 Regions of Absolute Stability of HBO(4-14)3

In this section, the region of absolute stability of HBO(4-14)3 is obtained and is compared with the region of absolute stability of ABM methods. This comparison is useful for the evaluation of numerical performance.

The unscaled regions of absolute stability, R, of HBO(p)3, $p = 4, 5 \ldots, 14$, shown in Fig. 1 are obtained by applying the root condition (see [18, pp. 256–257]) to the characteristic equation

$$\sum_{j=0}^{\lfloor (p-1)/2 \rfloor} \gamma_j r^j = 0, \tag{46}$$

which comes from an application of HBO(p)3 with constant h to the linear test equation

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

Let ABM(4-13) denote the family of ABM methods, ABM($p, p-1$), with predictor of order $p-1$ and corrector of order p in PECE mode. Table 6 lists the scaled abscissae of absolute stability, $\alpha/4$ and $\alpha/2$, of HBO(4-14)3 and ABM(4-13) [34, p. 135–140], respectively. It is seen that HBO(p)3 has a larger scaled interval of absolute stability than ABM($p, p-1$) for $p > 7$.

The principal error term of HBO(p)3 is of the form

$$\left[\delta_1 \{f^p\} + \delta_2 (p)\{\{f^{p-2}\}f\} + \delta_3 \{{}_2f^{p-1}\}_2 + \delta_4 \{{}_3f^{p-2}\}_3\right] h^{p+1}, \tag{47}$$

where $\{f^p\}, \{\{f^{p-2}\}f\}, \{{}_2f^{p-1}\}_2, \{{}_3f^{p-2}\}_3$ are elementary differentials defined in [7], [24] and [17] and the principal local truncation error coefficients (PLTC) are $[\delta_1, \delta_2, \delta_3, \delta_4]$. One may attempt to reduce the l_2–norm of the PLTC denoted by |PLTC|| by varying the parameters c_2 and c_3 in HBO(4-14)3.

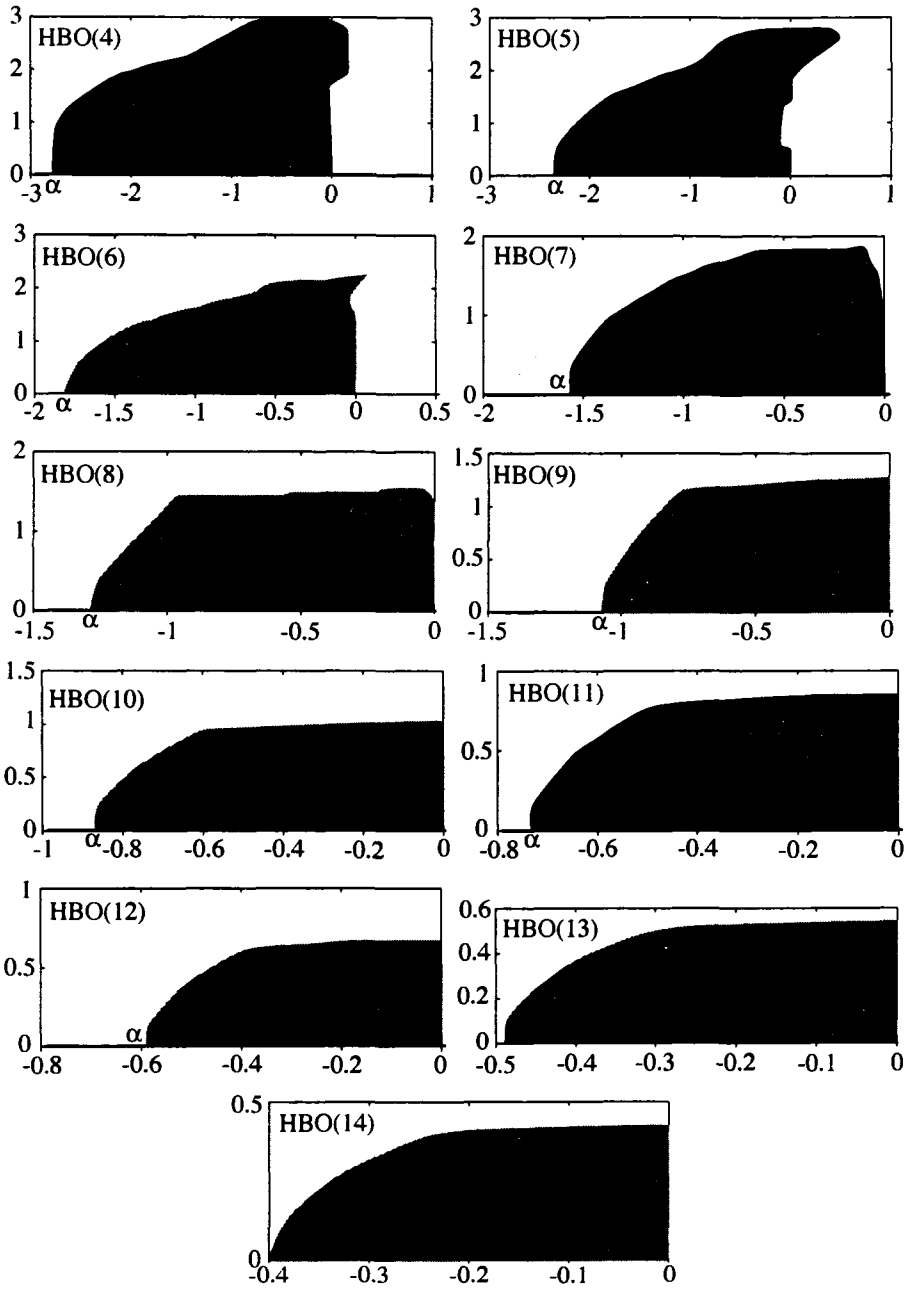

Figure 1: Unscaled regions of absolute stability of HBO(4-14)3.

Table 6: For order p, the table lists the scaled abscissae of absolute stability for HBO(p)3 and ABM($p, p-1$).

Order	$\alpha/4$	$\alpha/2$
p	HBO(p)3	ABM($p, p-1$)
4	−0.68	−0.97
5	−0.57	−0.70
6	−0.45	−0.52
7	−0.39	−0.39
8	−0.32	−0.30
9	−0.27	−0.22
10	−0.21	−0.17
11	−0.18	−0.13
12	−0.14	−0.11
13	−0.12	−0.03
14	−0.10	

The PLTC of ABM($p, p-1$) are $[\beta_k C_p*, C_{p+1}]$ [25, p. 107].

Table 7 lists the principal local truncation error coefficients (PLTC) of HBO(p)3. Also listed are the scaled norms $4 \times \|\text{PLTC}\|_2$ and $2 \times \|\text{PLTC}\|_2$ for HBO(p)3 and ABM($p, p-1$), respectively, the former rapidly decreasing and being smaller that the latter.

Table 7: For given order p, the table lists the principal local truncation error coefficients (PLTC) of HBO(p)3 and the scaled norms $4 \times \|\text{PLTC}\|_2$ and $2 \times \|\text{PLTC}\|_2$ for HBO(p)3 and ABM($p, p-1$), respectively.

	PLTC of HBO(p)3				$4 \times \|\text{PLTC}\|_2$	$2 \times \|\text{PLTC}\|_2$
p	δ_1	δ_2	δ_3	δ_4	HBO(p)3	ABM($p, p-1$)
4	$\frac{1}{4320}$	$-\frac{1}{480}$	$-\frac{1}{1080}$	$\frac{1}{120}$	4.72e-02	2.86e-01
5	$\frac{1}{21600}$	$-\frac{91}{108000}$	$-\frac{208}{870979}$	$\frac{235}{55816}$	2.38e-02	2.44e-01
6	$\frac{1}{181440}$	$-\frac{18}{67333}$	$-\frac{35}{991971}$	$\frac{79}{49268}$	9.08e-03	2.18e-01
7	$\frac{1}{1814400}$	$-\frac{646}{5542035}$	$-\frac{3}{638978}$	$\frac{78}{95609}$	4.60e-03	2.00e-01
8	$-\frac{1}{3277626}$	$-\frac{13}{299659}$	$\frac{4}{1781889}$	$\frac{75}{216103}$	1.96e-03	1.83e-01
9	$-\frac{1}{3799685}$	$-\frac{19}{968072}$	$\frac{7}{3004423}$	$\frac{16}{90575}$	1.00e-03	1.75e-01
10	$-\frac{1}{6833273}$	$-\frac{7}{887862}$	$\frac{5}{3400686}$	$\frac{40}{507317}$	4.44e-04	1.65e-01
11	$-\frac{1}{12746889}$	$-\frac{5}{1371479}$	$\frac{1}{1142050}$	$\frac{15}{374021}$	2.27e-04	1.57e-01
12	$-\frac{1}{27152737}$	$-\frac{5}{3263239}$	$\frac{1}{2216075}$	$\frac{11}{598260}$	1.04e-04	1.51e-01
13	$-\frac{1}{54600439}$	$-\frac{1}{1391224}$	$\frac{1}{4095617}$	$\frac{13}{1391300}$	5.28e-05	1.45e-01
14	$-\frac{1}{120066847}$	$-\frac{1}{3212565}$	$\frac{1}{8337285}$	$\frac{1}{229499}$	2.46e-05	

Chapter 4

Numerical Results

4.1 Description of the C Program for HBO(4-14)3

HBO(4-14)3 is implemented in C in the program `HBO4_14_3Ed1_d2prgsim.cpp`. The main lines of the calling hierarchy for this program are `HBO4d1d2initstepsprg()` which calls `HBO4d1d2varopstep()` to do the initial steps with HBO(4). Then the program goes to the main loop and does the following four calls until the end (one integration step for each iteration): `HBO4_14_3Ed1_d2Qc()` to calculate the IF coefficients; `HBO2_12d1d2prd1()` to calculate the coefficients of P_2; `HBO2_12d12prd2()` to calculate the coefficients of P_3; `HBO4_14d12_4SC()` to calculate the coefficients of the four local error estimators in order to control the stepsize and order.

The code of the C program is listed in Appendix C.

4.2 Problems Used for Comparison

We list standard test problems for ODE solvers. Many of these problems have been used to test HBO(4-14)3 and HBO(14)3. In this thesis, we report on the numerical performance of

- HBO(4-14)3 (programmed in Matlab) and Matlab's `ode113`,
- HBO(4-14)3 and HBO(14)3 (both programmed in Matlab) and

- HBO(4-14)3 and DP(8,7) (both programmed in C),

on the starred problems.

***CUBICWAVE** (CUBIC) Nonlinear wave equation in deep water [6].

***BRUSSELATOR** (BRUS) This reaction-diffusion partial differential equation is to be solved by the method of lines. Thus it is transformed into a system of ordinary differential equations along characteristics. [17, p. 248–249].

***ARENSTORF'S ORBITS** (AREN) Equations for the restricted three-body problem [1] and [17, p. 129–130].

***THE PLEIADES** (PLEI) A celestial mechanics problem for seven stars in the plane [17, p. 245–246].

***RESTRICTED 3-BODY PROBLEM** (R-3-body) Equations of motion of a restricted three-body problem [34, p. 246–247].

***EULER'S EQUATION** (EULER) Equations of motion for a rigid body without external force [34, p. 242–243].

The 25 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:

A1 $y' = -y$, $y(0) = 1$ (Exponentially decreasing).

A2 $y' = -y^3/2$, $y(0) = 1$ (A special case of the Riccati equation [10, p. 73]).

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

A4 $y' = \frac{y}{4}\left(1 - \frac{y}{20}\right)$, $y(0) = 1$ (A logistic curve [10, p. 97]).

A5 $y' = \frac{y-x}{y+x}$, $y(0) = 4$ (A spiral curve [10, p. 38]).

SMALL SYSTEMS:

***B1** The growth of two conflicting populations [10, p. 102].

***B2** A linear chemical reaction [12, p. 175].

B3 A nonlinear chemical reaction [12, p. 177].

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

B5 Euler's equations of motion for a rigid body without external force. [23].

MODERATE SYSTEMS:

***C1** A radioactive decay chain.

C2 Another radioactive decay chain.

C3 Derived from a parabolic partial differential equation.

C4 As in C3 except with 51 equations.

C5 Five-body problems: the motion of 5 outer planets about the sun [37, p. 102–103].

ORBIT EQUATIONS:

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

***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 [10, p. 4, 69].

***E2** Derived from Van der Pol's equation [20] and [10, p. 358, 531].

E3 Derived from Duffing's equation [10, p. 390].

E4 Derived from the falling body equation [10, p. 60].

***E5** Derived from a linear pursuit equation [10, p. 117].

These equations have been scaled so that $x_0 = 0$ and $x_f = 20$.

4.3 Comparing HBO(4-14)3 vs. ode113

The numerical performance of HBO(4-14)3 in Matlab and Matlab's `ode113` is compared on the following problems: (1) Arenstorf's orbits; (2) the Brusselator and the Pleiades; (3) Euler's equation and the restricted three-body problem; and (4) the following nonstiff DETEST problems: growth problem B1 of two conflicting populations, two-body problems D1–D5, and Van der Pol's equation E2 with $\epsilon = 1$.

HBO(4-14)3 is selfstarting with HBO(4)3. The initial stepsize, h_1, is chosen by a method similar to steps (a) and (b) of [17, p. 169].

Computations were performed on a Mac with a dual 2.5 GHz PowerPC G5 and 4 GB DDR SSRAM running under Mac OS X Version 10.4.6 and Matlab Version 7.0.4.352 (R14) Service Pack 2.

4.3.1 CPU against maximum global error

The maximum global error (MGE) was obtained from the errors at each integration step. These errors were calculated from the numerical value y_{n+1} obtained with HBO(4-14)3 and the "exact solution" obtained with Matlab's `ode113` under stringent tolerance 5×10^{-14}.

The CPU time was obtained from the curves which fit, in a least-squares sense, the data $(\log_{10}(|\text{MGE}|), \log_{10}(\text{CPU}))$ using Matlab's `polyfit`.

In Figs. 2 and 3, CPU (horizontal axis) is plotted versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for 12 problems considered in this thesis.

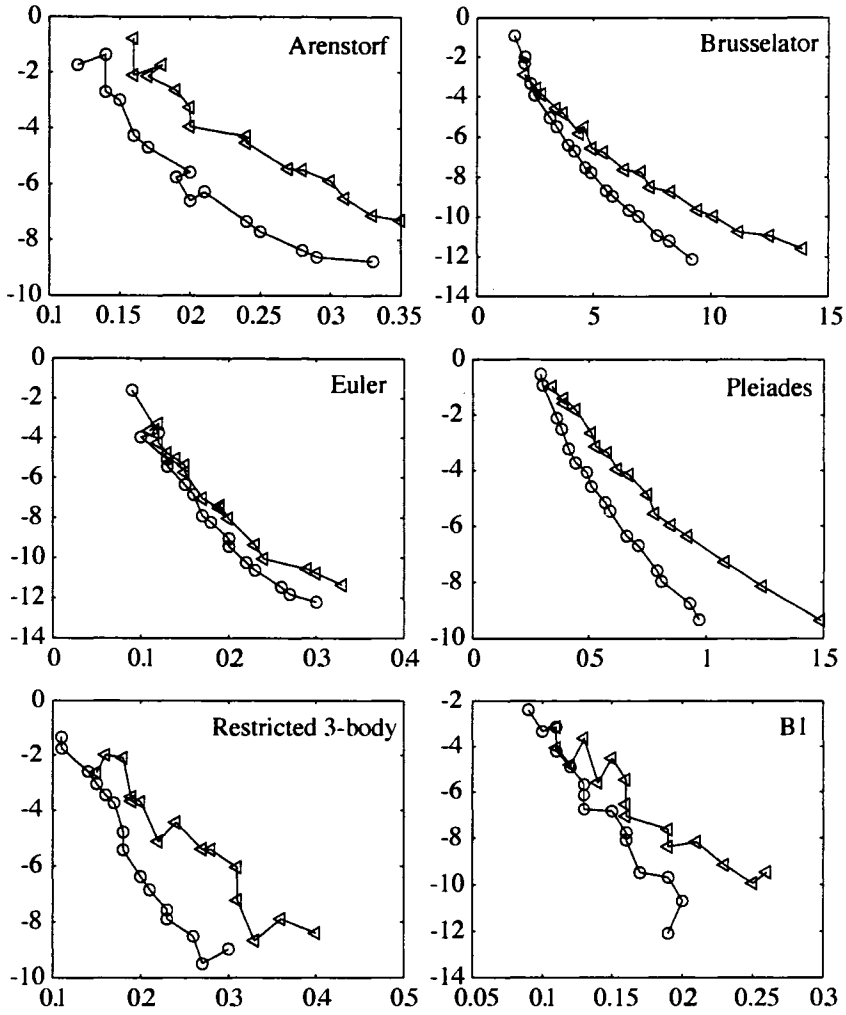

Figure 2: CPU (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for Arenstorf, Brusselator, Euler, Pleiades, the restricted three-body and B1.
∘ HBO(4-14)3
▷ `ode113`

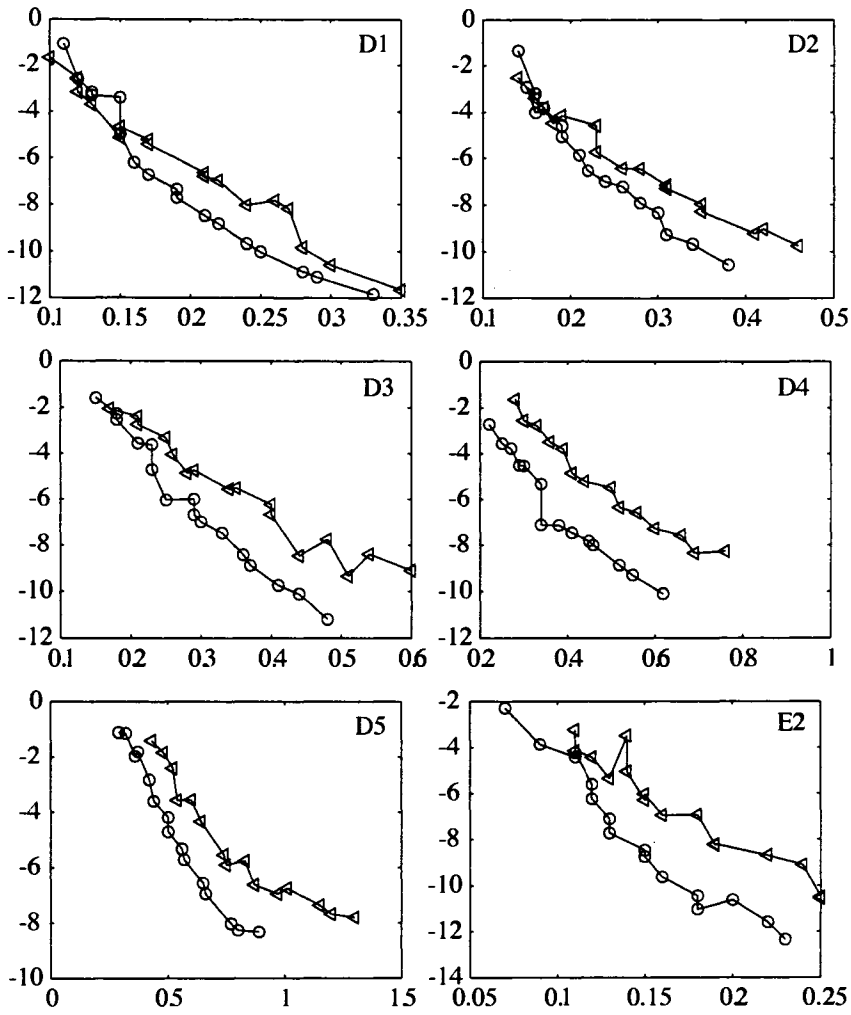

Figure 3: CPU (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for D1 to D5 and E2.
∘ HBO(4-14)3
▷ `ode113`

Table 8: CPU percentage efficiency gain (CPU PEG) of HBO(4-14)3 over `ode113` for the listed problems.

Problem	CPU PEG	Problem	CPU PEG
Arenstorf	42%	D1	12%
Brusselator	34%	D2	23%
Euler	13%	D3	37%
Pleiades	45%	D4	51%
Restricted 3-body	43%	D5	49%
B1	26%	E2	32%

4.3.2 CPU PEG of HBO(4-14)3 against `ode113`

The CPU percentage efficiency gain (CPU PEG) is defined by the formula (cf. Sharp [36]),

$$(\text{CPU PEG})_i = 100\left(\frac{\sum_j \text{CPU}_{2,ij}}{\sum_j \text{CPU}_{1,ij}} - 1\right), \tag{48}$$

where $\text{CPU}_{1,ij}$ and $\text{CPU}_{2,ij}$ are the CPU of methods 1 and 2, respectively, associated with problem i, and $j = -\log_{10}(|\text{MGE}|)$.

The CPU PEG for six problems is listed in Table 8.

It is seen from Figs. 2 and 3, and Table 8 that the new VSVO HBO(4-14)3 compares favorably to Matlab's `ode113` on the basis of the MGE Efficiency Curves against the CPU for the problems considered.

4.3.3 Maximum global error against tolerance

Table 9 lists several numerical data related to the step control for several problems considered on the time interval $[0, t_f]$ with set tolerance (TOL). The data listed are the CPU time (CPU), the number of function evaluations (NFE), the number of failed attempts (REJ) and the maximum global error (MGE) of HBO(4-14)3 and `ode113`. For HBO(4-14)3, NFE splits into 75% and 25% between f and f', respectively.

For equivalent MGE, the boldface data in Table 9 show that HBO(4-14)3 uses less CPU time and fewer function evaluations than `ode113` for tolerances 10^{-07} and

Table 9: For each problem, time interval $[0, t_f]$ and tolerance (TOL), the table lists the CPU time (CPU), the number of functions evaluations (NFE), the number of failed attempts (REJ) and the maximum global error (MGE). The left column is for HBO(4-14)3 and the right column is for `ode113`.

		HBO(4-14)3 and ode113							
Problem	TOL	CPU		NFE		REJ		MGE	
BRUS	10^{-04}	3.12	2.57	860	736	1	5	9.13e-06	2.64e-04
$t_f = 7.5$	10^{-07}	**5.55**	4.95	**1620**	1359	2	4	**2.17e-09**	2.74e-07
	10^{-10}	9.19	**9.84**	2780	**2419**	1	4	7.25e-13	**2.22e-10**
EULR	10^{-04}	0.12	0.09	636	485	7	4	1.79e-04	2.89e-03
$t_f \approx 52.2$	10^{-07}	0.17	0.15	1088	921	0	4	1.26e-08	3.99e-06
	10^{-10}	0.26	0.24	1860	1637	0	4	3.51e-12	8.92e-11
AREN	10^{-04}	0.12	0.15	612	491	13	20	1.87e-02	2.22e-01
$t_f \approx 17.1$	10^{-07}	**0.20**	0.19	**1072**	927	8	18	**2.76e-06**	2.28e-03
	10^{-10}	0.28	**0.27**	1756	**1637**	0	18	4.45e-09	**3.29e-06**
R-3-body	10^{-04}	0.11	0.11	628	513	15	20	4.73e-02	4.15e 00
$t_f \approx 6.2$	10^{-07}	**0.18**	0.18	**1084**	959	8	18	**1.59e-05**	7.99e-03
	10^{-10}	0.26	**0.27**	1780	**1632**	0	21	3.15e-09	**4.03e-06**
PLEI	10^{-04}	0.36	0.31	964	765	19	22	7.81e-03	4.99e-01
$t_f = 3$	10^{-07}	**0.57**	0.53	**1584**	1428	6	23	**7.21e-06**	7.22e-04
	10^{-10}	0.93	**0.85**	2660	**2408**	0	23	1.79e-09	**1.09e-06**
B1	10^{-04}	0.10	0.09	468	377	7	12	4.78e-04	5.26e-03
$t_f = 20$	10^{-07}	**0.13**	0.15	**800**	686	9	11	**1.74e-07**	3.05e-05
	10^{-10}	0.20	**0.19**	1332	**1208**	12	11	2.02e-11	**2.30e-08**
E2	10^{-04}	0.11	0.11	572	448	12	15	3.84e-05	6.02e-04
$t_f = 20$	10^{-07}	**0.15**	0.15	**996**	842	18	11	**3.44e-09**	5.22e-07
	10^{-10}	0.22	**0.22**	1704	**1437**	30	10	2.61e-12	**2.07e-09**
D1	10^{-04}	0.12	0.10	524	450	7	7	2.56e-03	2.11e-02
$t_f = 16\pi$	10^{-07}	**0.16**	0.15	**868**	886	0	5	**6.25e-07**	7.48e-06
	10^{-10}	0.24	**0.22**	1480	**1480**	0	3	2.08e-10	**1.06e-07**
D2	10^{-04}	0.16	0.12	752	577	12	12	6.09e-04	4.97e-02
$t_f = 16\pi$	10^{-07}	0.21	0.18	1312	1156	8	15	1.43e-06	3.34e-05
	10^{-10}	0.31	0.31	2152	2035	0	12	5.29e-10	6.89e-08
D3	10^{-04}	0.18	0.14	992	785	18	26	5.42e-03	4.32e-01
$t_f = 16\pi$	10^{-07}	**0.29**	0.25	**1704**	1530	8	25	**1.01e-06**	4.75e-04
	10^{-10}	0.41	**0.40**	2840	**2688**	0	27	1.77e-10	**5.98e-07**
D4	10^{-04}	0.22	0.17	1316	1102	28	41	6.43e-04	2.81e 00
$t_f = 16\pi$	10^{-07}	**0.34**	0.30	**2284**	1992	18	37	**4.48e-06**	2.80e-03
	10^{-10}	0.52	**0.50**	3736	**3463**	0	36	1.33e-09	**3.29e-06**
D5	10^{-04}	0.29	0.26	1972	1662	48	77	7.92e-02	4.56e 00
$t_f = 16\pi$	10^{-07}	**0.50**	0.45	**3424**	2970	33	61	**6.40e-05**	4.00e-02
	10^{-10}	0.77	**0.74**	5544	**5099**	0	56	9.45e-09	**2.91e-06**

10^{-10}, respectively. It is seen that HBO(4-14)3 controls better the stepsize for given tolerance and maximum global error than `ode113` even though the step control is problem dependent. There is a compromise between a sharper step control and the number of failed attempts.

4.4 Comparing HBO(4-14)3 vs. HBO(14)3

In this section, the VSVO HBO(4-14)3 is compared with HBO(14)3 of order 14 [28], both in Matlab, in solving Van der Pol's equation for $\epsilon = 2, 3, \ldots, 6$ [17, pp. 111–115] and the DETEST problems: B2, C1, E1 and E5. We show that variable-order HBO(14)3 performs better on the problems considered. The starting values for HBO(14)3 are obtained with DP(5,4)7FM [11].

4.4.1 CPU against MGE for HBO(4-14)3 and HBO(14)3

It is seen from Fig. 4 that, for large values of ϵ, HBO(4-14)3 is performing better than HBO(14)3 on the basis of CPU versus MGE. In Fig. 5 we have that HBO(4-14)3 performs better than HBO(14)3 on the basis of CPU versus MGE.

4.4.2 CPU PEG of HBO(4-14)3 against HBO(14)3

Table 10 lists the CPU percentage efficiency gain of HBO(4-14)3 against HBO(14)3 for Van der Pol's equation. One observes that HBO(4-14)3 performs better than HBO(14)3 as ϵ gets larger.

Table 11 lists the CPU percentage efficiency gain of HBO(4-14)3 against HBO(14)3 for problems B2, C1, E1 and E5 for which HBO(4-14)3 performs better than HBO(14)3. For the other problems, HBO(14)3 is as good as HBO(4-14)3. Note that the l_2 norm of the truncation error in HBO(14)3 is very small (see Table 7).

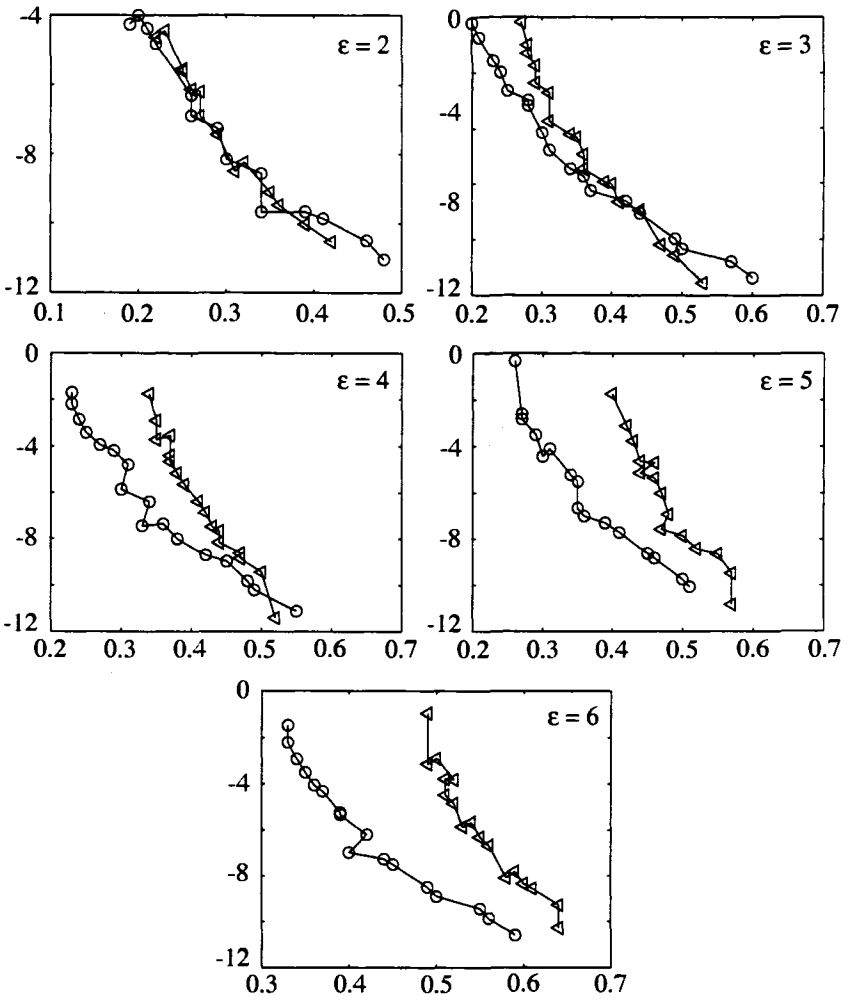

Figure 4: For $\epsilon = 2, \ldots, 6$, CPU (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for Van der Pol's equation.
∘ HBO(4-14)3
▷ HBO(14)3

Table 10: For given ϵ, CPU percentage efficiency gain (CPU PEG) of HBO(4-14)3 over HBO(14)3 for Van der Pol's equation.

ϵ	2	3	4	5	6
CPU PEG	1%	6%	17%	26%	27%

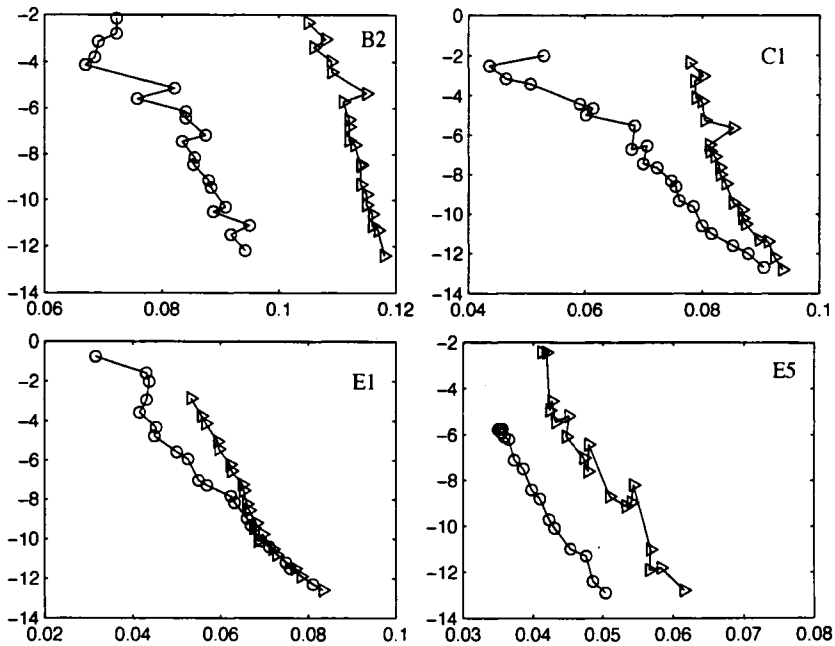

Figure 5: CPU (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for B2, C1, E1 and E5.
∘ HBO(4-14)3
▷ HBO(14)3

Table 11: CPU percentage efficiency gain (CPU PEG) of HBO(4-14)3 over HBO(14)3 for the listed problems.

Problem	CPU PEG
B2	37%
E1	12%
C1	24%
E5	28%

4.5 Comparing HBO(4-14)3 in C vs. DP(8,7)13M

The numerical performance of HBO(4-14)3 and DP(8,7)13M, both in C, is compared on the following problems: (1) Arenstorf's orbits; (2) the Brusselator; (3) Cubicwave; and (4) the two-body problems D1–D5. For problems whose derivative evaluations are expensive, HBO(4-14)3 wins over DP(8,7)13M.

4.5.1 CPU against maximum global error

The maximum global error (MGE) is computed as it was done in Section 4.3.1. The "exact solutions" of D1–D5 were obtained with Newton's method. The "exact solutions" for Arenstorf, Brusselator, and Cubicwave were obtained with DP(8,7)13M using the same tolerance as HBO(4-14)3. Matlab's `polyfit` was used to obtain the CPU time in a least-squares sense for the data $(\log_{10}(|\text{MGE}|), \log_{10}(\text{CPU}))$.

In Fig. 6, CPU (horizontal axis) is plotted versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for the listed problems.

4.5.2 CPU PEG of HBO(4-14)3 against DP(8,7)13M

The CPU PEG for eight problems is listed in Table 12, over the interval $[10^{-4}, 10^{-11}]$ for D1–D4, $[10^{-4}, 10^{-10}]$ for D5, $[10^{-4}, 10^{-12}]$ for Arenstorf, and $[10^{-3}, 10^{-12}]$ for Brusselator and Cubicwave. It is clear that HBO(4-14)3 uses much less CPU time than DP(8,7)13M for the expensive Cubicwave problem. DP(8,7)13M wins over HBO(4-14)3 for the simple problems: D1-D5 and Arenstorf. For the Brusselator, DP(8,7)13M uses less CPU time when the tolerance is between $[10^{-4}, 10^{-9}]$. However, HBO(4-14)3 takes less CPU time when the tolerance is smaller than 10^{-9}.

4.5.3 Maximum global error against tolerance

In general, HBO(4-14)3 is superior to DP(8,7)13M on costly problems such as Cubicwave. But DP(8,7)13M works better than HBO(4-14)3 on simple problems such as D1–D5 and Arenstorf.

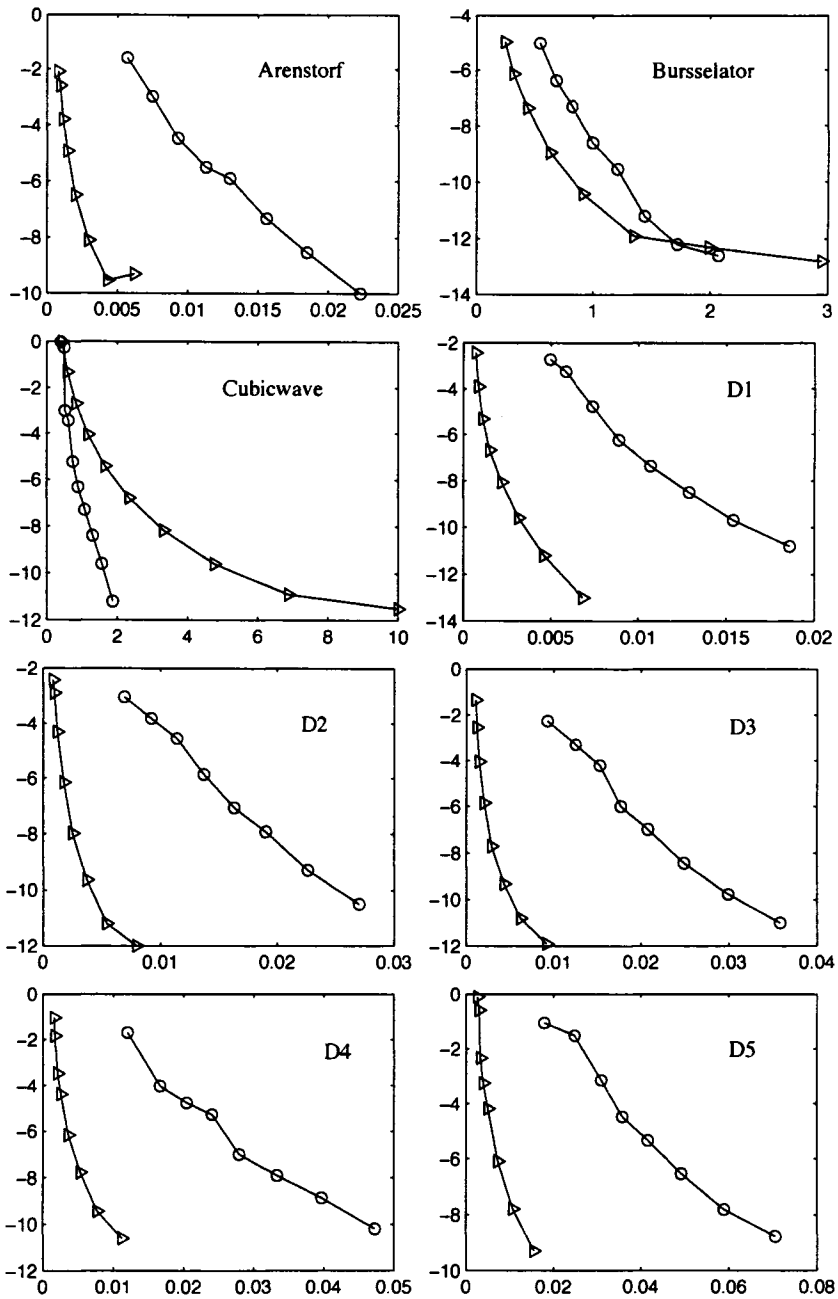

Figure 6: CPU (horizontal axis) versus $\log_{10}(|\text{MGE}|)$ (vertical axis) for Arenstorf, Brusselator, Cubicwave and D1 to D5.
○ HBO(4-14)3
▷ DP(8,7)13M

Table 12: CPU percentage efficiency gain (CPU PEG) of HBO(4-14)3 over DP(8,7)13M for the listed problems.

Problem	CPU PEG
Arenstorf	-83.80%
Brusselator	-42.90%
Cubicwave	157.86%
D1	-83.08%
D2	-86.30%
D3	-86.05%
D4	-83.74%
D5	-84.02%

Table 13: For each problem, time interval $[0, t_f]$ and tolerance (TOL), the table lists the CPU time (CPU), the number of functions evaluations (NFE), the number of failed attempts (REJ) and the maximum global error (MGE). The left column is for HBO(4-14)3 and the right column is for DP(8,7)13M.

		HBO(4-14)3 and DP(8,7)13M							
Problem	TOL	CPU		NFE		REJ		MGE	
BRUS	10^{-04}	5.29e-01	2.44e-01	872	1066	3	2	9.45e-06	1.05e-05
$t_f = 7.5$	10^{-07}	9.82e-01	6.30e-01	1608	2769	2	0	2.60e-09	1.14e-09
	10^{-10}	1.70e+00	1.99e+00	2764	8749	0	0	5.80e-13	5.38e-13
AREN	10^{-04}	5.75e-03	8.50e-04	616	988	14	16	2.55e-02	8.19e-03
$t_f \approx 17.1$	10^{-07}	1.12e-02	1.50e-03	1064	1846	8	6	3.18e-06	1.17e-05
	10^{-10}	1.85e-02	4.20e-03	1744	5382	0	0	2.94e-09	2.87e-10
CUBIC	10^{-04}	4.80e-01	5.80e-01	4496	6604	83	0	5.59e-01	4.78e-02
WAVE	10^{-07}	7.50e-01	1.65e+00	6372	18889	0	0	5.92e-06	3.81e-06
$t_f = 10^6$	10^{-10}	1.30e+00	4.75e+00	11020	54327	0	0	3.87e-09	2.43e-10
D1	10^{-04}	4.95e-03	6.50e-04	528	819	8	0	1.92e-03	3.74e-03
$t_f = 16\pi$	10^{-07}	8.90e-03	1.55e-03	856	2171	0	0	5.75e-07	2.11e-07
	10^{-10}	1.55e-02	4.55e-03	1468	6565	0	0	2.09e-10	7.06e-12
D2	10^{-04}	6.90e-03	8.50e-04	732	1131	10	14	9.19e-04	3.72e-03
$t_f = 16\pi$	10^{-07}	1.36e-02	1.75e-03	1300	2457	8	0	1.39e-06	6.91e-07
	10^{-10}	2.26e-02	5.40e-03	2140	7761	0	0	5.18e-10	5.76e-12
D3	10^{-04}	9.15e-03	1.00e-03	992	1417	19	18	5.47e-03	4.51e-02
$t_f = 16\pi$	10^{-07}	1.80e-02	2.00e-03	1692	2860	8	0	1.03e-06	1.41e-06
	10^{-10}	3.01e-02	6.25e-03	2828	8983	0	0	1.75e-10	1.77e-11
D4	10^{-04}	1.20e-02	1.60e-03	1304	2314	27	48	2.01e-02	9.08e-02
$t_f = 16\pi$	10^{-07}	2.40e-02	2.45e-03	2280	3458	19	0	5.37e-06	4.15e-05
	10^{-10}	3.96e-02	7.65e-03	3724	10946	0	0	1.36e-09	3.73e-10
D5	10^{-04}	1.80e-02	2.65e-03	1968	3978	48	96	8.82e-02	7.85e-01
$t_f = 16\pi$	10^{-07}	3.57e-02	3.90e-03	3384	5603	29	38	3.33e-05	5.58e-04
	10^{-10}	5.89e-02	1.06e-02	5532	15145	0	0	1.56e-08	1.63e-08

Chapter 5

Conclusion

A fast, variable-step, variable-order, 3-stage, Hermite–Birkhoff–Obrechkoff method of order 4 to 14 was constructed by solving Vandermonde-type systems satisfying multistep- and Runge–Kutta-type order conditions. The stepsize and order are controlled by four local error estimators. This method, in its vectorized Lagrange form, was tested on the Brusselator; Euler's equation; Arenstorf's orbits; the restricted three-body problem; the Pleiades; and the following nonstiff DETEST problems: two-body problems of class D, the growth problem of two conflicting populations of class B and Van der Pol's equation of class E.

When programmed in Matlab, the new method was found to have lower global error, use less CPU time and require fewer function evaluations than Matlab's `ode113`. VSVO HBO(4-14)3 was also compared with HBO(14)3 of order 14, both in Matlab, on the Van der Pol's equation and some DETEST problems: B2, C1, E1 and E5. We showed that HBO(4-14)3 performs better on these problems and similar problems than HBO(14)3.

When programmed in C, HBO(4-14)3 wins over DP(8,7)13M on expensive problems such as the Brusselator and Cubicwave over a long period of time. But DP(8,7)13M uses less CPU time than HBO(4-14)3 for simple problems such as D1–D5 and Arenstorf.

In a future work, the C program will be further optimized to reduce CPU time.

Appendix A

Algorithms

Algorithm 1 ***This algorithm constructs lower bidiagonal matrices*** L_k ***as functions of*** c_2, c_3 ***and*** η_j, $j = 2:6$.

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 ***as functions of*** c_2, c_3 ***and*** η_j, $j = 2:6$.

For $k = 2 : m-1$, do the following iteration:

For $j = m : -1 : k+1$, 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 overwrites*** $\boldsymbol{r} = r(1:m)$ ***with*** $U_2 \cdots U_{m-1} D^{-1} L_{m-1} L_{m-2} \cdots L_3 \boldsymbol{r}$ ***in*** $O(m^2)$ ***operations for IF,*** $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ ***and*** $\boldsymbol{P}_4$.

Given $[\eta_2, \eta_3, \ldots, \eta_6]$ and $\boldsymbol{r} = r(1:m)$, the following algorithm overwrites $\boldsymbol{r}$ with $U_2 \cdots U_{m-1} D^{-1} L_{m-1} L_{m-2} \cdots L_3 \boldsymbol{r}$.

Step (1) The following iteration overwrites $\boldsymbol{r} = r(1:m)$ with $L_{m-1} L_{m-2} \cdots L_3 \boldsymbol{r}$:

for $k = 3 : m-1$, compute

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

Step (2) The following iteration overwrites $\boldsymbol{r} = r(1:m)$ with $U_2 U_3 \cdots U_{m-1} D^{-1} \boldsymbol{r}$:

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

For $k = m-1 : -1 : 2$, compute

$$\begin{aligned} r(i) &= r(i) U_k(i,i), && i = k+1 : m, \\ r(i) &= r(i) - r(i+1), && i = k : m-1. \end{aligned}$$

Algorithm 3 uses minimum storage since the solution is obtained by successively transforming the right-hand side into the solution vector. This is an advantage compared to generating $m \times m$ triangular matrices as an intermediate result at each integration step.

Appendix B

Matlab Programming

Algorithm 3 which solves systems IF, P_2 and P_3 was programmed in C and compiled by the Matlab **mex** command into **mex** files.

Algorithm 3 which solves the P_4 system and three additional similar systems to produce four local error estimators $\widetilde{y}_{n+1,p-j}$, $j = 1, 4$ of order $p - j$ was programmed in C and compiled by the Matlab **mex** command into a **mex** file.

At runtime, the data of differential equations were the input. Then, at each integration step until completion of the integration, the previous **mex** files were called and run to calculate the values of the coefficients of IF, P_2, P_3, P_4 and three other step control predictors. CPU time and number of evaluations of $f(x, y)$ and $f'(x, y)$ for the runtime of Algorithm 3 were recorded.

Matlab's `ode113` can be run with appropriate tolerance for comparison with HBO(4-14)3.

The elementary matrix functions L_k^ℓ and U_k^ℓ of η_j, for $j = 2, 3, \ldots, 6$ and $\ell = 1, 2, 3, 4$, are constructed with Algorithms 1 and 2. These algorithms are not needed at runtime since these matrix functions are already implemented in the Matlab **mex** files mentioned above.

Appendix C

The C PROGRAM

```
{\small{
\begin{verbatim}
#include <stdio.h> #include <iostream> #include <math.h>

#include <fstream>
#include <time.h>
#include <string>

using namespace std;

//#include "odeD.cpp"
//#include "odeBr.cpp"
//#include "odeAr.cpp"
#include "odeCu.cpp"

static double* HB04d1d2varopstep(double RELTOL,double
TOL,std::string fname, double t, double* y, double* inpf, double h, int
k,int j, double* tvals, double** yvals, double** fvals, double**
f2vals, int nbpos, int nbcmp1, double tnew, double* ynew, double* fnew,
double* f2new, double stsize, double erk) {
    double tc=t;
    double stfac=0.9 ;
    double p = 1.0/3.0;
    // A: c_i
    double A[] ={0.0,2.0/3.0,1.0};
    // E = coeffs(sol y) - coeffs(SCP y)
   double Ebv1_3[] = {0.0,1.0/20.0, -29.0/500.0 , 1.0/125.0};
   double Ebv4[]   = {0.0, -17.0/1500.0};
   double B[4][4]={{0.0,0.0,0.0,0.0},{0.0,0.0,0.0,1.0/12.0},
   {0.0,0.0,9.0/4.0,9.0/16.0},{0.0,2.0/3.0,-5.0/4.0,17.0/48.0} };
```

```
   double bv4[]= {0.0,2.0/9.0,-1.0,1.0/24.0};
   int index1, index2;
   double** col1_3=new double*[nbcmp1];
   double* col4=new double[nbcmp1];
//zeros col1_3, col4
for(index1=0;index1<nbcmp1;index1++) {
    col1_3[index1]=new double[4];
   for(index2=0;index2<=3;index2++)
    col1_3[index1][index2]=0.0;
    col4[index1]=0.0;
}
double eps= 2.220446049250313e-16 ;
double hmin = 16*eps  ;

int jpnbpos=j+nbpos;

for(index1=1;index1<nbcmp1;index1++) {
    col1_3[index1][3] =fvals[index1][jpnbpos];
    col4[index1] =f2vals[index1][jpnbpos];
} double hA[]={0.0,0.0,0.0};

double hB[4][4]={{0.0,0.0,0.0,0.0},{0.0,0.0,0.0,0.0},
{0.0,0.0,0.0,0.0}, {0.0,0.0,0.0,0.0}};

double h2bv4[]={0.0,0.0,0.0,0.0};

for(index1=1;index1<=3;index1++) {
    for(index2=1;index2<=3;index2++)
     hB[index1][index2]=h*B[index1][index2]   ;
     h2bv4[index1]=h*h*bv4[index1] ;
}

for(index1=1;index1<=2;index1++) hA[index1] = h * A[index1];
double temp; double *temp1=new double[nbcmp1];

//col1_3(:,2) = feval(fname,t+hA(1), y + col1_3*hB(:,1) + col4*h2bv4(1)  );
for(index1=1;index1<nbcmp1;index1++)
temp1[index1]=y[index1]+(col1_3[index1][1]*hB[1][1]
+col1_3[index1][2]*hB[2][1]+col1_3[index1][3]*hB[3][1])
+col4[index1]*h2bv4[1];

double* temp2=new double[nbcmp1]; FirstD(fname,t +
h*A[1],temp1,nbcmp1,temp2); for(index1=1;index1<nbcmp1;index1++)
col1_3[index1][2]=temp2[index1];

//col1_3(:,1) = feval(fname,t+hA(2), y + col1_3*hB(:,2) + col4*h2bv4(2)  );
```

```
for(index1=1;index1<nbcmp1;index1++)
temp1[index1]=y[index1]+(col1_3[index1][1]*hB[1][2]
+col1_3[index1][2]*hB[2][2]+col1_3[index1][3]*hB[3][2])
+col4[index1]*h2bv4[2];

FirstD(fname,t+h*A[2],temp1,nbcmp1,temp2);
for(index1=1;index1<nbcmp1;index1++)
col1_3[index1][1]=temp2[index1];

//ynew = y  + col1_3*hB(:,3) + col4*h2bv4(3) ;
for(index1=1;index1<nbcmp1;index1++)
ynew[index1]=y[index1]+(col1_3[index1][1]*hB[1][3]
+col1_3[index1][2]*hB[2][3]+col1_3[index1][3]*hB[3][3])
+col4[index1]*h2bv4[3];

tnew = t + h ;

//[fnew, f2new] = feval(fname2, tnew, ynew) ;

FirstD(fname,tnew,ynew,nbcmp1,fnew);
SecondD(fname,tnew,ynew,nbcmp1,f2new);
for(index1=0;index1<nbcmp1;index1++)
col1_3[index1][1]=fnew[index1] ;

//erk = norm(h* col1_3 * Ebv1_3 + h*h *col4 * Ebv4 )   ;
for(index1=1;index1<nbcmp1;index1++)
temp1[index1]=h*(col1_3[index1][1]*Ebv1_3[1]+col1_3[index1][2]*Ebv1_3[2]
+col1_3[index1][3]*Ebv1_3[3])+h*h*col4[index1]*Ebv4[1];
//erk=norm(temp1)
temp=0.0; for(index1=1;index1<nbcmp1;index1++)
  temp= temp+ temp1[index1]*temp1[index1];
erk = sqrt(temp) ;

// (step size formula : h =hold, stsize = hnew)
   if (erk != 0.0)

//     ( step cntrl)
    stsize = h *stfac *pow((TOL/erk ),p)  ;
   else
    stsize = 2*h ;

// (end of step size formula : h =hold, stsize = hnew)

//  constraints that stsize must satisfy :
//    (a) h( next step) = max (h( next step), hmin, 0.5* h(last step))
//    (b) h( next step) = min (h( next step), hmax, 4 * h(last step))
```

```
   double hmax = 10.0 ;

   stsize = max ( stsize, 0.5* h ) ;
   stsize = max ( stsize, hmin ) ;
   stsize = min( stsize, hmax) ;
   stsize = min( stsize, 4*h) ;

double* result=new double[3];
result[0]=tnew;
result[1]=stsize;
result[2]=erk;

return result;
}

static double*  HBO4d1d2initstepsprg(double RELTOL,double
TOL,std::string fname,double t0,double tend,double* y0,int
nbinitst,double* tvals,double** yvals,double** fvals,double**
f2vals,int nbcmp1,double* res,int j, double h) {

double eps = 2.220446049250313e-16 ;
double hmax=0.1*fabs(tend-t0);
double stfac=0.9 ;

int nbeps=4;
double hmin = 16.0*eps ;
int index1,index2;

int nbpos=10;
double TOL111=TOL/500.0;

double* RESULT=new double[3]; double tem1=0.0; double tem2=0.0;
double stsize=0.0; double* temp1=new double[nbcmp1];double* ynew=new
double[nbcmp1]; double* fnew=new double[nbcmp1]; double* f2new=new
double[nbcmp1];

int jpnbposp1; double prevh; double* yc=new double[nbcmp1];

double tc = t0; double tnew=0.0;

for(index1=1;index1<nbcmp1;index1++) yc[index1] =y0[index1];

double* fc=new double [nbcmp1]; FirstD(fname,tc,yc,nbcmp1,fc);
double* f2c=new double[nbcmp1]; SecondD(fname,tc,yc,nbcmp1,f2c);
```

```
// p 169, Hairer: first guess for step size:

for(index1=1;index1<nbcmp1;index1++) {
    tem1=tem1+y0[index1]*y0[index1];
    tem2=tem2+fc[index1]*fc[index1];
}

double  sc = sqrt(tem1)*RELTOL+ TOL111 ;
//d0 =p168norm(y0) = norm(y0)/(sqrt(nbcmp)*sc)
int nbcmp=nbcmp1-1;
double d0 = sqrt(tem1)/(sqrt((double)nbcmp)*sc);
//d1 =p168norm(fc) = norm(fc)/(sqrt(nbcmp)*sc)
double d1 = sqrt(tem2)/(sqrt((double)nbcmp)*sc) ;

double h0=0.01*(d0/d1) ;

if ((d0 < 0.00001)||(d1 < 0.00001))
    h0= 1e-6;

double* y1=new double[nbcmp1]; y1[0]=0.0;
//  (if (d0 < 0.00001)||(d1 < 0.00001) )
//  perform one expl. Euler step
for(index1=1;index1<nbcmp1;index1++)
  y1[index1]=y0[index1]+h0*fc[index1];

double *fc1 =new double[nbcmp1]; FirstD(fname,tc+h0,y1,nbcmp1,fc1) ;

//d2 = (norm(fc1-fc)/(sqrt(nbcmp)*sc))/h0 ;
tem1=0.0; for(index1=1;index1<nbcmp1;index1++) {
    temp1[index1]=fc1[index1]-fc[index1];
    tem1 = tem1 + temp1[index1]*temp1[index1];
} double d2 = (sqrt(tem1)/(sqrt((double)nbcmp1-1.0)*sc))/h0 ;

double  h1=pow( 0.01/max(d1,d2), 1.0/5.0 ) ;

if ( max(d1,d2) <= pow( 10.0,-15.0 ) )
h1=max( pow(10.0,-6.0), h0*pow(10.0,-3.0) );

h=min(100*h0, h1) ;

j=0; int jpnbpos=j+nbpos;

tvals[jpnbpos] = tc; for(index1=1;index1<nbcmp1;index1++) {
    yvals[index1][jpnbpos] =  yc[index1];
    fvals[index1][jpnbpos] = fc[index1];
    f2vals[index1][jpnbpos] = f2c[index1];
```

```
}
int k=1111 ;

//----------- iterations for harder problems --------------
int iter=0;

double erk=1111.0 ; double hold; double HBO4initsteph1;

double hnew=0.0;

while ( fabs(erk) > TOL111 ) {

   hold=h;

//calling HBO4d1d2varopstep
RESULT =HBO4d1d2varopstep(RELTOL,
TOL111,fname,tc,yc,fc,h,k,j,tvals,yvals,fvals,f2vals,nbpos,nbcmp1,tnew,ynew,fnew,f2new,hnew,e

hnew=RESULT[1]; erk=RESULT[2];

// TOL/1000  step size:
// formula : err1/TOL/1000 = \frac{ C* hold^3/ft3 der3(y0)}{  C* hnew^3/ft3
der3(y0) }

   h=hnew ;
   HBO4initsteph1=h ;
   iter=iter+1;
}
// (end while)

j=0 ; int nreje=0;

// init param:

double p = 1.0/3.0;
// A: c_i
double A[] = {0.0,2.0/3.0, 1.0};
double hA[]={0.0,0.0,0.0};

double hB[4][4]={{0.0,0.0,0.0,0.0},{0.0,0.0,0.0,0.0},
{0.0,0.0,0.0,0.0},{0.0,0.0,0.0,0.0}};

double h2bv4[]={0.0,0.0,0.0,0.0};

double Ebv1_3[] = {0.0, 1.0/20.0,-29.0/500.0 , 1.0/125.0};
```

```
double Ebv4[]   = {0.0, -17.0/1500.0};

double B[4][4]={{0.0,0.0,0.0,0.0} ,{0.0, 0.0,0.0,1.0/12.0},
{0.0,0.0,9.0/4.0,9.0/16.0}, {0.0,2.0/3.0,-5.0/4.0,17.0/48.0} };
double bv4[]= {0.0,2.0/9.0, -1.0, 1.0/24.0 } ;

//zero col1_3,col4
double** col1_3=new double*[nbcmp1];
double* col4=newdouble[nbcmp1];

for(index1=0;index1<nbcmp1;index1++) {
    col1_3[index1]=new double[4];
    for(index2=0;index2<=3;index2++)
     col1_3[index1][index2]=0.0;

    col4[index1]=0.0;
}

//( MAIN LOOP  1 integration step/ iteration)

while ( j  < nbinitst ) {

     h=hnew;

    if((tc+h) > tend)
       h=tend-tc ;

    jpnbpos=j+nbpos;

    for(index1=1;index1<nbcmp1;index1++)
    {
        col1_3[index1][3] =fvals[index1][jpnbpos];
        col4[index1] =f2vals[index1][jpnbpos];
    }

    for(index1=0;index1<=3;index1++)
    {
        for(index2=0;index2<=3;index2++)
         hB[index1][index2]=h*B[index1][index2];

        h2bv4[index1]=h*h*bv4[index1] ;
    }

    for(index1=0;index1<=2;index1++)
     hA[index1] = h * A[index1];
```

```
//col1_3(:,2) = feval(fname,tc+hA(1), yc + col1_3*hB(:,1) +
col4*h2bv4(1) ); for(index1=1;index1<nbcmp1;index1++)
temp1[index1]=yc[index1]+(col1_3[index1][1]*hB[1][1]
+col1_3[index1][2]*hB[2][1]+col1_3[index1][3]*hB[3][1])
+col4[index1]*h2bv4[1]; double* temp2=new double[nbcmp1];
FirstD(fname,tc + h*A[1],temp1,nbcmp1,temp2);
for(index1=1;index1<nbcmp1;index1++)
col1_3[index1][2]=temp2[index1];

//col1_3(:,1)=feval(fname,tc+hA(2), yc + col1_3*hB(:,2) + col4*h2bv4(2));
for(index1=1;index1<nbcmp1;index1++)
temp1[index1]=yc[index1]+(col1_3[index1][1]*hB[1][2]
+col1_3[index1][2]*hB[2][2]+col1_3[index1][3]*hB[3][2])
+col4[index1]*h2bv4[2];

FirstD(fname,tc+h*A[2],temp1,nbcmp1,temp2);
for(index1=1;index1<nbcmp1;index1++)
col1_3[index1][1]=temp2[index1];

//ynew = yc  + col1_3*hB(:,3) + col4*h2bv4(3) ;
for(index1=1;index1<nbcmp1;index1++)
ynew[index1]=yc[index1]+(col1_3[index1][1]*hB[1][3]
+col1_3[index1][2]*hB[2][3]+col1_3[index1][3]*hB[3][3])
+col4[index1]*h2bv4[3];

tnew = tc + h ;

FirstD(fname,tnew,ynew,nbcmp1,fnew);
SecondD(fname,tnew,ynew,nbcmp1,f2new);
for(index1=0;index1<nbcmp1;index1++)
col1_3[index1][1]=fnew[index1] ;

// erk = norm(h* col1_3 * Ebv1_3 + h*h *col4 * Ebv4 ) ;

for(index1=1;index1<nbcmp1;index1++)
temp1[index1]=h*(col1_3[index1][1]*Ebv1_3[1]
+col1_3[index1][2]*Ebv1_3[2]+col1_3[index1][3]*Ebv1_3[3])
+h*h*col4[index1]*Ebv4[1];
//erk=norm(temp1)
tem1=0.0; for(index1=1;index1<nbcmp1;index1++)
  tem1= tem1+ temp1[index1]*temp1[index1];
erk = sqrt(tem1) ;

//   ode45 hmin:  eps = 2.220446049250313e-16
//    eps = 2.220446049250313e-16 ;
//    hmin = 16*eps*abs(tc);    stephmin2=hmin
```

```
//  PART :  STEP SIZE

//  (step size formula : h =hold, stsize = hnew)
   if (erk != 0.0)
   stsize = h *stfac *pow((TOL/erk ),p)  ;
   else
   stsize = 2*h ;

//  (end of step size formula : h =hold, stsize = hnew)

//  constraints that stsize must satisfy :
//     (a) h( next step) = max (h( next step), hmin, 0.5* h(last step))
//     (b) h( next step) = min (h( next step), hmax, 4 * h(last step))

   stsize = max ( stsize, 0.5* h ) ;
   stsize = max ( stsize, hmin ) ;
   stsize = min( stsize, hmax) ;
   stsize = min( stsize, 4*h) ;

  tc=tnew;
  for(index1=1;index1<nbcmp1;index1++)
  {
      yc[index1]=ynew[index1];
  }

  jpnbposp1=j+nbpos+1;

  for(index1=1;index1<nbcmp1;index1++)
  {
   fvals[index1][jpnbposp1] = fnew[index1];
   f2vals[index1][jpnbposp1]= f2new[index1];
  }

  hnew=stsize ;

//  (END OF INTEG STEP)

//  (yvals,tvals,.. UPDATE STEP)

   tvals[jpnbposp1] =  tc;

   for(index1=1;index1<nbcmp1;index1++)
   yvals[index1][jpnbposp1] = yc[index1];

   prevh = h ;
   j = j+1 ;
```

```
//return to stop temporarily the process of doing 1 step:

}
// ( END OF MAIN LOOP  1 integration step/ iteration)

// print out statistics
// with index j-1,  yvals(n+1+nbpos, . )  cont solution
int n=j-1;
int nb_f_eval=(n+1+nreje)*nbeps ;
int NFE=nb_f_eval;
res[0]=0.0; res[1]=(double)NFE;
// nb succes step:
res[2]=(double)n+1.0 ;
res[3]=(double)nreje;
double * result=new
double[2];
result[0]=(double)j;
result[1]=h;
return result;
}

static void HBO2_12d12prd1(double *xtab,double *parout) {

       int i,n, idx = 0;
       double  lop[17][17], U[17][17],A_MAT2col[17];
       double invdiag[17];
       for(i=0;i<17;i++)
           invdiag[i]=0.0;

       double b_v[17], b_v2[17];

       double  RHS[17];

       double x, xkm11, xkm10,  xkm9 , xkm8 , xkm7 , xkm6 ,
              xkm5 , xkm4 ,xkm3 , xkm2 ;
       double xkm1 , xk1, xk2,xk3, c2,  c3  ;
       double hl , hm1 , hm2 , hm3 , hm4 , hm5 , hm6 ,
              hm7 , hm8 , hm9 ,hm10 , hm11  ;

       double ft[19];

       double  c22 , c23 , c24 , c25 , c26 ,c27 ;
       double  c28 , c29 , c210 , c211 , c212 ;
       double  c213 , c214 , c215 ;
       double rmultip,  lhs2ord, lhs2ordp1, lhs2ordtot  ;
```

```
    int  nm1, nm2, nm3, k , lim1, np12, ic;

   //  -----  HB013_3E2yp1c.c  calc. : ---

    x    = xtab[1] ;xkm11= xtab[2] ;xkm10= xtab[3] ;
    xkm9 = xtab[4] ;xkm8 = xtab[5] ;xkm7 = xtab[6] ;
    xkm6 = xtab[7] ;xkm5 = xtab[8] ;xkm4 = xtab[9] ;
    xkm3 = xtab[10] ;xkm2 = xtab[11] ;xkm1 = xtab[12] ;
    xk1  = xtab[13] ;xk2  = xtab[14] ;xk3  = xtab[15] ;
    c2   = xtab[16] ;c3   = xtab[17] ;

    n= (int)xtab[40] ;

    nm1=n-1;
    nm2=n-2;
    nm3=n-3;
    hl = (x-xk1)/c2 ;

hm11= (xkm11-xk1)/hl;hm10= (xkm10-xk1)/hl;
hm9 = (xkm9-xk1)/hl;hm8 = (xkm8-xk1)/hl;
hm7 = (xkm7-xk1)/hl;hm6 = (xkm6-xk1)/hl;
hm5 = (xkm5-xk1)/hl;hm4 = (xkm4-xk1)/hl;
hm3 = (xkm3-xk1)/hl;hm2 = (xkm2-xk1)/hl;   h
m1 = (xkm1-xk1)/hl;

    ft[1]=1.0;ft[2]=2.0 ;ft[3]= 6.0 ;
    ft[4]=24.0;ft[5]=120.0;ft[6]=720.0;
    ft[7]=5040.0;ft[8]=40320.0;ft[9]=362880.0;
    ft[10]=3628800.0;ft[11]=39916800.0;ft[12]=479001600.0;
    ft[13]=6.227020800000000e+09  ;ft[14]=8.717829120000000e+10;
    ft[15]=1.307674368000000e+12;ft[16]=2.092278988800000e+13;
    ft[17]=3.556874280960000e+14 ;ft[18]=0.0;

     c2 =    6.666666666666666e-01;c22 =     2.222222222222222e-01;
     c23 =     4.938271604938271e-02;c24 =     8.230452674897118e-03;
     c25 =     1.097393689986282e-03;c26 =     1.219326322206980e-04;
     c27 =     1.161263164006648e-05;c28 =     9.677193033388730e-07;
     c29 =     7.168291135843504e-08;c210 =     4.778860757229003e-09;
     c211 =     2.896279246805456e-10;c212 =     1.609044026003031e-11;
     c213 =     8.251507825656569e-13;c214 =     3.929289440788842e-14;
     c215 =     1.746350862572819e-15;

    A_MAT2col[0]=0.0; A_MAT2col[1]=0.0; A_MAT2col[2]=0.0;
    A_MAT2col[3] =hm1 ;A_MAT2col[4] =hm1 ;A_MAT2col[5] =hm2 ;
    A_MAT2col[6]=hm2 ;A_MAT2col[7] =hm3 ;A_MAT2col[8] =hm3 ;
    A_MAT2col[9] =hm4 ;A_MAT2col[10]=hm;A_MAT2col[11]=hm5 ;
```

```
      A_MAT2col[12]=hm5 ;A_MAT2col[13]=0.0; A_MAT2col[14]=0.0;
      A_MAT2col[15]=0.0;A_MAT2col[16]=0.0;

       RHS[0]= 0.0;RHS[1]=  c2  ; RHS[2]=  c22  ; RHS[3]=  c23  ;
       RHS[4]=  c24  ;RHS[5]=  c25  ;RHS[6]=  c26  ;RHS[7]=  c27  ;
       RHS[8]=  c28  ;RHS[9] = c29  ;RHS[10]= c210  ;RHS[11]= c211 ;
       RHS[12]= c212 ;RHS[13]= 0.0;RHS[14]= 0.0;RHS[15]= 0.0;RHS[16]= 0.0;

   for (idx = 1; idx <= n; idx++)
   {
      invdiag[idx]=1.0e0;
   }

   for (idx = 4; idx <= n; idx++)
   {
          rmultip =-A_MAT2col[idx-1]/(idx-1) ;
          invdiag[idx]= invdiag[idx-1]*rmultip  ;
   }

  lop[3][4] = -3.0e0/hm1; lop[3][5] = -4.0e0/hm1; lop[3][6] = -5.0e0/hm1;
  lop[3][7] = -6.0e0/hm1; lop[3][8] = -7.0e0/hm1; lop[3][9] = -8.0e0/hm1;
  lop[3][10]= -9.0e0/hm1; lop[3][11]= -10.0e0/hm1; lop[3][12]= -11.0e0/hm1;
  lop[4][5] = -3.0e0/hm1; lop[4][6] = -4.0e0/hm1; lop[4][7] = -5.0e0/hm1;
  lop[4][8] = -6.0e0/hm1; lop[4][9] = -7.0e0/hm1; lop[4][10]= -8.0e0/hm1;
  lop[4][11]= -9.0e0/hm1; lop[4][12]= -10.0e0/hm1; lop[5][6] = -3.0e0/hm2;
  lop[5][7] = -4.0e0/hm2; lop[5][8] = -5.0e0/hm2; lop[5][9] = -6.0e0/hm2;
  lop[5][10]= -7.0e0/hm2; lop[5][11]= -8.0e0/hm2; lop[5][12]= -9.0e0/hm2;
  lop[6][7] = -3.0e0/hm2; lop[6][8] = -4.0e0/hm2; lop[6][9] = -5.0e0/hm2;
  lop[6][10]= -6.0e0/hm2; lop[6][11]= -7.0e0/hm2; lop[6][12]= -8.0e0/hm2;
  lop[7][8] = -3.0e0/hm3; lop[7][9] = -4.0e0/hm3; lop[7][10]= -5.0e0/hm3;
  lop[7][11]= -6.0e0/hm3; lop[7][12]= -7.0e0/hm3; lop[8][9] = -3.0e0/hm3;
  lop[8][10]= -4.0e0/hm3; lop[8][11]= -5.0e0/hm3; lop[8][12]= -6.0e0/hm3;
  lop[9][10]= -3.0e0/hm4; lop[9][11]= -4.0e0/hm4; lop[9][12]= -5.0e0/hm4;
  lop[10][11]= -3.0e0/hm4 ;lop[10][12]= -4.0e0/hm4 ;lop[11][12]= -3.0e0/hm5;

U[1][3]=  1.0e0/hm1; U[1][5]=1.0e0/(hm2-hm1);
U[1][7]1.0e0/(hm3-hm2);U[1][9]=1.0e0/(hm4-hm3);
U[1][11]=1.0e0/(hm5-hm4);U[2][3]=2.0e0/hm1;
U[2][4]=2.0e0/hm1;U[2][5]=  2.0e0/(hm2-hm1);
U[2][6]=2.0e0/(hm2-hm1);U[2][7]= 2.0e0/(hm3-hm2);
U[2][8]=2.0e0/(hm3-hm2);U[2][9]= 2.0e0/(hm4-hm3);
U[2][10]=2.0e0/(hm4-hm3);U[2][11]=2.0e0/(hm5-hm4);
U[2][12]=2.0e0/(hm5-hm4);U[3][4]=  3.0e0/hm1;
U[3][5]=3.0e0/hm2;U[3][6]= 3.0e0/(hm2-hm1);
U[3][7]=3.0e0/(hm3-hm1);U[3][8]=3.0e0/(hm3-hm2);
U[3][9]=3.0e0/(hm4-hm2);U[3][10]=3.0e0/(hm4-hm3);
```

```
U[3][11]=3.0e0/(hm5-hm3);U[3][12]=3.0e0/(hm5-hm4);
U[4][5]=4.0e0/hm2;U[4][6]= 4.0e0/hm2;
U[4][7]=4.0e0/(hm3-hm1);U[4][8]= 4.0e0/(hm3-hm1);
U[4][9]=4.0e0/(hm4-hm2);U[4][10]= 4.0e0/(hm4-hm2);
U[4][11]=4.0e0/(hm5-hm3);U[4][12]= 4.0e0/(hm5-hm3);
U[5][6]=5.0e0/hm2;U[5][7]= 5.0e0/hm3;
U[5][8]=5.0e0/(hm3-hm1);U[5][9]= 5.0e0/(hm4-hm1);
U[5][10]=5.0e0/(hm4-hm2);U[5][11]=5.0e0/(hm5-hm2);
U[5][12]=5.0e0/(hm5-hm3);U[6][7]= 6.0e0/hm3;
U[6][8]=6.0e0/hm3;U[6][9]= 6.0e0/(hm4-hm1);
U[6][10]=6.0e0/(hm4-hm1);U[6][11]=6.0e0/(hm5-hm2);
U[6][12]=6.0e0/(hm5-hm2);U[7][8]= 7.0e0/hm3;
U[7][9]=7.0e0/hm4;U[7][10]=7.0e0/(hm4-hm1);
U[7][11]=7.0e0/(hm5-hm1);U[7][12]=7.0e0/(hm5-hm2);
U[8][9]=8.0e0/hm4;U[8][10]=8.0e0/hm4;
U[8][11]=8.0e0/(hm5-hm1);U[8][12]=8.0e0/(hm5-hm1);
U[9][10]=9.0e0/hm4;U[9][11]=9.0e0/hm5;
U[9][12]=9.0e0/(hm5-hm1);U[10][11]=10.0e0/hm5;
U[10][12]=10.0e0/hm5;U[11][12]=11.0e0/hm5;

  for (i = 1; i <= n; i++)
   {
      b_v2[i]=RHS[i] ;
   }

  for (k = 3; k <= nm1; k++)
   {
      for (idx = n ; idx >= k+1 ; idx--)
      {
         b_v2[ idx]=b_v2[ idx-1]+lop[k][idx]*b_v2[ idx] ;
      }
   }

  for (k = 1; k <= n; k++)
   {
      b_v2[k]= invdiag[k]* b_v2[k] ;
   }

  for (k = nm1; k >= 2 ; k--)
   {
      for (idx = k+1 ; idx <= n ; idx++)
      {
          b_v2[idx]= U[k][idx] *b_v2[idx] ;
      }

      for (idx = k ; idx <= nm1 ; idx++)
```

```
      {
          b_v2[idx]= b_v2[idx]-b_v2[idx+1] ;
      }
   }

     np12=(n+1)/2 ;
     lim1=np12*2-1;

     for (idx = 3 ; idx <= lim1 ; idx +=2 )
      {
          b_v2[idx]= U[1][idx] *b_v2[idx] ;
      }

      for (idx = 1 ; idx <=lim1-2 ; idx +=2 )
      {
          b_v2[idx]= b_v2[idx]-b_v2[idx+2] ;
      }

for (idx = 1 ; idx <= 12 ; idx +=1 )
    {
      b_v[idx]=0.0;
     }

ic=1 ;

     for (idx = 1 ; idx <= n ; idx +=2 )
      {
       b_v[ic]=b_v2[idx];
       ic=ic+1;
      }

     ic=7;
     for (idx = 2 ; idx <= n ; idx +=2 )
      {
      b_v[ic]=b_v2[idx];
      ic=ic+1;
     }

lhs2ord=0;
     for (idx = 3 ; idx <= n ; idx +=2 )
      {
      lhs2ord= lhs2ord+ b_v2[idx]*pow(A_MAT2col[idx],n) ;
     }

lhs2ordp1= lhs2ord/ft[n] ;
```

```
lhs2ord=0;

     for (idx = 4 ; idx <= n ; idx +=2 )
      {
      lhs2ord= lhs2ord+ b_v2[idx]*pow(A_MAT2col[idx],n-1) ;
     }

lhs2ord= lhs2ord/ft[n-1] ;

lhs2ordtot=lhs2ordp1+lhs2ord ;

    for (idx = 1 ; idx <= 13 ; idx +=1 )
      {
          parout[idx]=0.0;
     }

  parout[1] = lhs2ordtot ;

  for (idx = 2; idx <= 13 ; idx++)
   {
       parout[idx]=b_v[idx-1] ;
   }
return;
}

static void HB02_12d12prd2(double *xtab,double *parout  ) {

       int i, n,idx = 0;

       double  lop[17][17], U[17][17],A_MAT2col[17] ;
       double invdiag[17];
       for(i=0;i<17;i++)
           invdiag[i]=0.0;

       double b_v[17], b_v2[17];

       double  RHS[17];

       double x, xkm11, xkm10,  xkm9 , xkm8 , xkm7 , xkm6 ,
              xkm5 , xkm4 ,xkm3 , xkm2 ;
       double xkm1 , xk1, xk2,xk3, c2,  c3;
       double hl ,h0,h1, h2, hm1 , hm2 , hm3 , hm4 , hm5 ,
              hm6 ,  hm7 , hm8, hm9 , hm10 , hm11  ;
       double ft2, ft3, ft4 , ft5 ,ft6 ,ft7 ,ft8 , ft9 , ft10 ,
              ft11 , ft12, ft13 ,  ft14 ,  ft15 ,  ft16 , ft17  ;
       double ift[19];
```

```
double multip;
double LP20, lhs2 ,rynnlhs, b3, b2;
int  nm1, nm2, nm3, k, ndiv2, ic, lim1;

x    = xtab[1] ;xkm11= xtab[2] ;xkm10= xtab[3] ;
xkm9 = xtab[4] ;xkm8 = xtab[5] ;xkm7 = xtab[6] ;
xkm6 = xtab[7] ;xkm5 = xtab[8] ;xkm4 = xtab[9] ;
xkm3 = xtab[10] ;xkm2 = xtab[11] ;xkm1 = xtab[12] ;
xk1  = xtab[13] ;xk2  = xtab[14] ;xk3  = xtab[15] ;
c2   = xtab[16] ;c3   = xtab[17] ;LP20   = xtab[18] ;
lhs2    = xtab[19] ; b3   = xtab[20] ;b2   = xtab[21] ;
n    = (int)xtab[40]  ;

xk3 = x ;
hl = (xk3-xk1)/c3 ;

hm11= (xkm11-xk1)/hl;hm10= (xkm10-xk1)/hl;
hm9 = (xkm9-xk1)/hl;hm8 = (xkm8-xk1)/hl;
hm7 = (xkm7-xk1)/hl;hm6 = (xkm6-xk1)/hl;
hm5 = (xkm5-xk1)/hl;hm4 = (xkm4-xk1)/hl;
hm3 = (xkm3-xk1)/hl;hm2 = (xkm2-xk1)/hl;
hm1 = (xkm1-xk1)/hl;h0 = xk1-xk1;
h1 = (xk2 - xk1)/hl ; h2 = (xk3 - xk1)/hl ;

ft2=2.0 ;ft3= 6.0 ;ft4 =    24.0;ft5 =   120.0;
ft6 =   720.0;ft7 =        5040.0;ft8 =       40320.0;
ft9 =      362880.0;ft10 =     3628800.0;ft11 =    39916800.0;
ft12 =   479001600.0;ft13 =    6.227020800000000e+09  ;
ft14 =     8.717829120000000e+10;ft15 =     1.307674368000000e+12;
ft16 =     2.092278988800000e+13;ft17 =     3.556874280960000e+14 ;

ift[0]=0.0;ift[1]=0.0;ift[2] =5.000000000000000e-01  ;
ift[3] =1.666666666666667e-01 ;ift[4] =4.166666666666666e-02 ;
ift[5] =8.333333333333333e-03 ;ift[6] =1.388888888888889e-03 ;
ift[7] =1.984126984126984e-04 ;ift[8] =2.480158730158730e-05 ;
ift[9] =2.755731922398589e-06 ;ift[10]=2.755731922398589e-07;
ift[11]=2.505210838544172e-08 ;ift[12]=2.087675698786810e-09;
ift[13]=1.605904383682161e-10 ;ift[14]=1.147074559772972e-11;
ift[15]=7.647163731819816e-13 ;ift[16]=4.779477332387385e-14;
ift[17]=2.811457254345521e-15 ;ift[18]=0.0;

rynnlhs= ( ift[n+1]-(b2*lhs2+LP20 ))/b3 ;

nm1=n-1;
nm2=n-2;
nm3=n-3;
```

```
      A_MAT2col[0]=0.0;  ; A_MAT2col[1] =0.0 ; A_MAT2col[2] =0.0 ;
      A_MAT2col[3]=c2  ;  A_MAT2col[4] =hm1 ; A_MAT2col[5] =hm1 ;
      A_MAT2col[6] =hm2 ; A_MAT2col[7] =hm2 ; A_MAT2col[8] =hm3 ;
      A_MAT2col[9] =hm3 ;A_MAT2col[10] =hm4 ;A_MAT2col[11]=hm4 ;
      A_MAT2col[12]=hm5 ; A_MAT2col[13]=hm5 ; A_MAT2col[14]=0.0 ;
      A_MAT2col[15]=0.0 ; A_MAT2col[16]=0.0 ;

       RHS[0]=  0.0  ; RHS[1]=  1.0  ; RHS[2]=  ift[2]  ;
       RHS[3]=  ift[3]  ; RHS[4]=  ift[4]  ; RHS[5]=  ift[5]  ;
       RHS[6]=  ift[6]  ; RHS[7]=  ift[7]  ; RHS[8]=  ift[8]  ;
       RHS[9]=  ift[9]  ; RHS[10]= ift[10]  ; RHS[11]= ift[11] ;
       RHS[12]= ift[12] ;RHS[13]= rynnlhs    ;RHS[14]= 0.0 ;
       RHS[15]= 0.0 ; RHS[16]= 0.0    ;

   for (idx = 1; idx <= n; idx++)
   {
      invdiag[idx]=1.0e0;
   }
   for (idx = 4; idx <= n; idx++)
   {
         multip =-A_MAT2col[idx-1]/(idx-1) ;
         invdiag[idx]= invdiag[idx-1]*multip  ;
   }

  lop[3][4] = -3/c2  ;lop[3][5] = -4/c2  ;lop[3][6] = -5/c2  ;
  lop[3][7] = -6/c2  ;lop[3][8] = -7/c2  ;lop[3][9] = -8/c2  ;
  lop[3][10]= -9/c2  ;lop[3][11]= -10/c2  ;lop[3][12]= -11/c2  ;
  lop[3][13]= -12/c2  ;lop[4][5] = -3/hm1  ;lop[4][6] = -4/hm1  ;
  lop[4][7] = -5/hm1  ;lop[4][8] = -6/hm1  ;lop[4][9] = -7/hm1  ;
  lop[4][10]= -8/hm1  ;lop[4][11]= -9/hm1  ;lop[4][12]= -10/hm1  ;
  lop[4][13]= -11/hm1 ;lop[5][6] = -3/hm1  ;lop[5][7] = -4/hm1  ;
  lop[5][8] = -5/hm1  ;lop[5][9] = -6/hm1  ;lop[5][10]= -7/hm1  ;
  lop[5][11]= -8/hm1  ;lop[5][12]= -9/hm1  ;lop[5][13]= -10/hm1  ;
  lop[6][7] = -3/hm2  ;lop[6][8] = -4/hm2  ;lop[6][9] = -5/hm2  ;
  lop[6][10]= -6/hm2  ;lop[6][11]= -7/hm2  ;lop[6][12]= -8/hm2  ;
  lop[6][13]= -9/hm2  ;lop[7][8] = -3/hm2  ;lop[7][9] = -4/hm2  ;
  lop[7][10]= -5/hm2  ;lop[7][11]= -6/hm2  ;lop[7][12]= -7/hm2  ;
  lop[7][13]= -8/hm2  ;lop[8][9] = -3/hm3  ;lop[8][10]= -4/hm3  ;
  lop[8][11]= -5/hm3  ;lop[8][12]= -6/hm3  ;lop[8][13]= -7/hm3  ;
  lop[9][10]= -3/hm3  ;lop[9][11]= -4/hm3  ;lop[9][12]= -5/hm3  ;
  lop[9][13]= -6/hm3  ;lop[10][11]= -3/hm4  ;lop[10][12]= -4/hm4  ;
  lop[10][13]= -5/hm4  ;lop[11][12]= -3/hm4  ;lop[11][13]= -4/hm4  ;
  lop[12][13]= -3/hm5  ;

U[1][3]=  1/c2; U[1][4]=  1/(hm1-c2); U[1][6]=  1/(hm2-hm1);
```

```
U[1][8]=  1/(hm3-hm2); U[1][10]= 1/(hm4-hm3); U[1][12]= 1/(hm5-hm4);
U[2][3]=  2/c2; U[2][4]=  2/hm1; U[2][5]=  2/(hm1-c2);
U[2][6]=2/(hm2-hm1); U[2][7]=  2/(hm2-hm1); U[2][8]=  2/(hm3-hm2);
U[2][9]= 2/(hm3-hm2); U[2][10]= 2/(hm4-hm3); U[2][11]= 2/(hm4-hm3);
U[2][12]= 2/(hm5-hm4); U[2][13]= 2/(hm5-hm4); U[3][4]=  3/hm1;
U[3][5]=  3/hm1; U[3][6]=  3/(hm2-c2) ; U[3][7]= 3/(hm2-hm1);
U[3][8]=  3/(hm3-hm1); U[3][9]=  3/(hm3-hm2); U[3][10]= 3/(hm4-hm2);
U[3][11]= 3/(hm4-hm3); U[3][12]= 3/(hm5-hm3); U[3][13]= 3/(hm5-hm4);
U[4][5]=  4/hm1 ; U[4][6]=  4/hm2 ; U[4][7]=  4/(hm2-c2);
U[4][8]=4/(hm3-hm1); U[4][9]=  4/(hm3-hm1); U[4][10]= 4/(hm4-hm2);
U[4][11]= 4/(hm4-hm2); U[4][12]= 4/(hm5-hm3); U[4][13]= 4/(hm5-hm3);
U[5][6]=  5/hm2; U[5][7]=  5/hm2; U[5][8]=  5/(hm3-c2);
U[5][9]=5/(hm3-hm1); U[5][10]= 5/(hm4-hm1); U[5][11]= 5/(hm4-hm2);
U[5][12]= 5/(hm5-hm2); U[5][13]= 5/(hm5-hm3);U[6][7]= 6/hm2;
U[6][8]=  6/hm3; U[6][9]=  6/(hm3-c2); U[6][10]= 6/(hm4-hm1);
U[6][11]=6/(hm4-hm1); U[6][12]=6/(hm5-hm2); U[6][13]=6/(hm5-hm2);
U[7][8]=  7/hm3; U[7][9]=  7/hm3; U[7][10]= 7/(hm4-c2);
U[7][11]=7/(hm4-hm1); U[7][12]=7/(hm5-hm1); U[7][13]=7/(hm5-hm2);
U[8][9]= 8/hm3; U[8][10]=8/hm4; U[8][11]=8/(hm4-c2);
U[8][12]=8/(hm5-hm1); U[8][13]=8/(hm5-hm1); U[9][10]=9/hm4;
U[9][11]=9/hm4; U[9][12]=9/(hm5-c2); U[9][13]=9/(hm5-hm1);
U[10][11]=10/hm4; U[10][12]=10/hm5; U[10][13]=10/(hm5-c2);
U[11][12]=11/hm5; U[11][13]=11/hm5; U[12][13]=12/hm5;

  for (i = 1; i <= n-1; i++)
   {
      b_v2[i]=RHS[i] ;
   }
   b_v2[n] = rynnlhs ;

  for (k = 3; k <= nm1; k++)
   {
      for (idx = n ; idx >= k+1 ; idx--)
      {
         b_v2[ idx]=b_v2[ idx-1]+lop[k][idx]*b_v2[ idx] ;
      }
   }

   for (k = 1; k <= n; k++)
   {
      b_v2[k]= invdiag[k]* b_v2[k] ;
   }

  for (k = nm1 ; k >= 2 ; k--)
   {
      for (idx = k+1 ; idx <= n ; idx++)
```

```
      {
          b_v2[idx]= U[k][idx] *b_v2[idx] ;
      }

      for (idx = k ; idx <= nm1 ; idx++)
      {
          b_v2[idx]= b_v2[idx]-b_v2[idx+1] ;
      }
   }

    b_v2[3]= U[1][ 3] *b_v2[3] ;

  ndiv2=n/2;
  lim1=ndiv2*2;

  for (idx = 4 ; idx <= lim1 ; idx +=2 )
    {
        b_v2[idx]= U[1][idx] *b_v2[idx] ;
    }

  b_v2[1]= b_v2[1]-b_v2[3 ] ;

  if (n >= 4) b_v2[3]= b_v2[3]-b_v2[4 ] ;

  for (idx = 4 ; idx <= lim1-2 ; idx +=2 )
    {
     b_v2[idx]= b_v2[ idx]-b_v2[ idx+2] ;
    }

  for (idx = 1 ; idx <= 13 ; idx +=1 )
    {
    b_v[idx]=0.0;
    }

b_v[1]= b_v2[3] ;
b_v[2] =b_v2[1];

ic=3 ; for (idx = 4 ; idx <= n ; idx +=2 )
  {
   b_v[ic]=b_v2[idx];
   ic=ic+1;
  }

ic=8; b_v[ic]=b_v2[2]; ic=ic+1;

for (idx = 5 ; idx <= n ; idx +=2 )
```

```
  {
   b_v[ic]=b_v2[idx] ;
   ic=ic+1;
  }
   for (idx = 1; idx <= 13 ; idx++)
   {
       parout[idx]=b_v[idx] ;
   }
return;
}

static void HB04_14_3Ed1_d2Qc(double* xtab,double* b_v) {

       int i, idx = 0;
       double  lop[17][17], U[17][17],A_MAT2col[17] ;
       double RHS[17], invdiag[17];
       for(i=0;i<17;i++)
       invdiag[i]=0.0;

       double b_v2[17];
       double LP20n,LP20np1,LP20ntot ;

       double xpos, xkm11, xkm10,  xkm9 , xkm8 , xkm7 , xkm6 ,
              xkm5 , xkm4 , xkm3 , xkm2 ;
       double xkm1 , xk1, xk2,xk3, c2,  c3  ;

       double hl , hm1 , hm2 , hm3 , hm4 , hm5 , hm6 ,
              hm7 , hm8 , hm9 , hm10 , hm11  ;

       double ft[19];
       double rift2,rift3,rift4,rift5,rift6,rift7,rift8,rift9,
              rift10, rift11,rift12,rift13,rift14,rift15,rift16,rift17;
       double  multip ;
       int n,  nm1, nm2,nm3, k, lim1, np12, ic;

       xpos = xtab[1] ;xkm11= xtab[2] ;xkm10= xtab[3] ;
       xkm9 = xtab[4] ;xkm8 = xtab[5] ;xkm7 = xtab[6] ;
       xkm6 = xtab[7] ;xkm5 = xtab[8] ;xkm4 = xtab[9] ;
       xkm3 = xtab[10] ;xkm2 = xtab[11] ;xkm1 = xtab[12] ;
       xk1  = xtab[13] ;xk2  = xtab[14] ;xk3  = xtab[15] ;
       c2   = xtab[16] ;c3   = xtab[17] ;
       n   = (int)xtab[40] ;
       hl = (xpos-xk1) ;

       hm11= (xkm11-xk1)/hl;hm10= (xkm10-xk1)/hl;
       hm9 = (xkm9-xk1)/hl;hm8 = (xkm8-xk1)/hl;
```

```
    hm7 = (xkm7-xk1)/hl;hm6 = (xkm6-xk1)/hl;
    hm5 = (xkm5-xk1)/hl;hm4 = (xkm4-xk1)/hl;
    hm3 = (xkm3-xk1)/hl;hm2 = (xkm2-xk1)/hl;
    hm1 = (xkm1-xk1)/hl;

    ft[0]=0.0;ft[1]=1.0;ft[2]=2.0;ft[3]= 6.0;
    ft[4] =24.0 ;ft[5] =120.0 ; ft[6] =720.0 ;
    ft[7] =5040.0 ;ft[8] =40320.0 ;ft[9] =362880.0 ;
    ft[10] =3628800.0 ;ft[11] =39916800.0 ;
    ft[12] =479001600.0 ;ft[13] =6.227020800000000e+09 ;
    ft[14] =8.717829120000000e+10 ;ft[15] =1.307674368000000e+12 ;
    ft[16] =2.092278988800000e+13 ;ft[17] =3.556874280960000e+14 ;
    ft[18]=0.0;

    rift2 =5.000000000000000e-01;rift3 =1.666666666666667e-01;
    rift4 =4.166666666666666e-02;rift5 =8.333333333333333e-03;
    rift6 =1.388888888888889e-03;rift7 =1.984126984126984e-04;
    rift8 =2.480158730158730e-05;rift9 =2.755731922398589e-06;
    rift10=2.755731922398589e-07;rift11=2.505210838544172e-08;
    rift12=2.087675698786810e-09;rift13=1.605904383682161e-10;
    rift14=1.147074559772972e-11;rift15=7.647163731819816e-13;
    rift16=4.779477332387385e-14;rift17=2.811457254345521e-15;

    A_MAT2col[0] =0.0 ;A_MAT2col[1] =0.0 ;A_MAT2col[2] =0.0 ;
    A_MAT2col[3] =c3 ;A_MAT2col[4] =c2 ;A_MAT2col[5] =hm1 ;
    A_MAT2col[6] =hm1 ;A_MAT2col[7] =hm2 ; A_MAT2col[8] =hm2 ;
    A_MAT2col[9] =hm3 ;A_MAT2col[10]=hm3 ;A_MAT2col[11]=hm4 ;
    A_MAT2col[12]=hm4 ;A_MAT2col[13]=hm5 ;A_MAT2col[14]=hm5 ;
    A_MAT2col[15]=0.0 ;A_MAT2col[16]=0.0 ;

     RHS[0]=  0.0  ;RHS[1]=  1.0  ;RHS[2]=  rift2  ;
     RHS[3]=  rift3  ;RHS[4]=  rift4  ;RHS[5]=  rift5  ;
     RHS[6]=  rift6  ;RHS[7]=  rift7  ;RHS[8]=  rift8  ;
     RHS[9]=  rift9  ;RHS[10]= rift10  ;RHS[11]= rift11 ;
     RHS[12]= rift12 ;RHS[13]= rift13 ;RHS[14]= rift14 ;
     RHS[15]= 0.0 ; RHS[16]= 0.0 ;

    nm1=n-1 ;
    nm2=n-2 ;
    nm3=n-3 ;

for (idx = 1; idx <= n; idx++)
{
   invdiag[idx]=1.0e0;
}
```

```
for (idx = 4; idx <= n; idx++)
{
       multip =-A_MAT2col[idx-1]/(idx-1) ;

       invdiag[idx]= invdiag[idx-1]*multip  ;
}

    lop[3][4] = -3.0e0/c3  ;lop[3][5] = -4.0e0/c3  ;
    lop[3][6] = -5.0e0/c3  ;lop[3][7] = -6.0e0/c3  ;
    lop[3][8] = -7.0e0/c3  ;lop[3][9] = -8.0e0/c3  ;
    lop[3][10]= -9.0e0/c3  ;lop[3][11]= -10.0e0/c3  ;
    lop[3][12]= -11.0e0/c3 ;lop[3][13]= -12.0e0/c3  ;
    lop[3][14]= -13.0e0/c3 ;lop[4][5] = -3.0e0/c2  ;
    lop[4][6] = -4.0e0/c2  ;lop[4][7] = -5.0e0/c2  ;
    lop[4][8] = -6.0e0/c2  ;lop[4][9] = -7.0e0/c2  ;
    lop[4][10]= -8.0e0/c2  ;lop[4][11]= -9.0e0/c2  ;
    lop[4][12]= -10.0e0/c2 ;lop[4][13]= -11.0e0/c2  ;
    lop[4][14]= -12.0e0/c2 ;lop[5][6] = -3.0e0/hm1  ;
    lop[5][7] = -4.0e0/hm1 ;lop[5][8] = -5.0e0/hm1  ;
    lop[5][9] = -6.0e0/hm1 ;lop[5][10]= -7.0e0/hm1 ;
    lop[5][11]= -8.0e0/hm1 ;lop[5][12]= -9.0e0/hm1 ;
    lop[5][13]= -10.0/hm1  ;lop[5][14]= -11.0/hm1  ;
    lop[6][7] = -3.0/hm1   ;lop[6][8] = -4.0/hm1  ;
    lop[6][9] = -5.0/hm1   ;lop[6][10]= -6.0/hm1  ;
    lop[6][11]= -7.0/hm1   ;lop[6][12]= -8.0/hm1  ;
    lop[6][13]= -9.0/hm1   ;lop[6][14]= -10.0/hm1  ;
    lop[7][8] = -3.0/hm2   ;lop[7][9] = -4.0/hm2  ;
    lop[7][10]= -5.0/hm2   ;lop[7][11]= -6.0/hm2   ;
    lop[7][12]= -7.0/hm2   ;lop[7][13]= -8.0/hm2   ;
    lop[7][14]= -9.0/hm2   ;lop[8][9] = -3.0/hm2  ;
    lop[8][10]= -4.0e0/hm2 ;lop[8][11]= -5.0e0/hm2  ;
    lop[8][12]= -6.0e0/hm2 ;lop[8][13]= -7.0e0/hm2 ;
    lop[8][14]= -8.0e0/hm2 ;lop[9][10]= -3.0e0/hm3  ;
    lop[9][11]= -4.0e0/hm3 ;lop[9][12]= -5.0e0/hm3  ;
    lop[9][13]= -6.0e0/hm3 ;lop[9][14]= -7.0e0/hm3 ;
    lop[10][11]= -3.0e0/hm3;lop[10][12]= -4.0e0/hm3;
    lop[10][13]= -5.0e0/hm3;lop[10][14]= -6.0e0/hm3;
    lop[11][12]= -3.0e0/hm4;lop[11][13]= -4.0e0/hm4  ;
    lop[11][14]= -5.0e0/hm4;lop[12][13]= -3.0e0/hm4  ;
    lop[12][14]= -4.0e0/hm4;lop[13][14]= -3.0e0/hm5  ;

    U[1][3]=  1.0e0/c3;U[1][4]=  1.0e0/(c2-c3);
    U[1][5]=  1.0e0/(hm1-c2);U[1][7]=  1.0e0/(hm2-hm1);
    U[1][9]= 1.0e0/(hm3-hm2);U[1][11]= 1.0e0/(hm4-hm3);
    U[1][13]= 1.0e0/(hm5-hm4);U[2][3]=  2.0e0/c3;
    U[2][4]=  2.0e0/c2;U[2][5]=  2.0e0/(hm1-c3);
```

```
      U[2][6]=  2.0e0/(hm1-c2);U[2][7]=  2.0e0/(hm2-hm1);
      U[2][8]=  2.0e0/(hm2-hm1);U[2][9]=  2.0e0/(hm3-hm2);
      U[2][10]= 2.0e0/(hm3-hm2);U[2][11]= 2.0e0/(hm4-hm3);
      U[2][12]= 2.0e0/(hm4-hm3);U[2][13]= 2.0e0/(hm5-hm4);
      U[2][14]= 2.0e0/(hm5-hm4);U[3][4]=  3.0e0/c2;
      U[3][5]=  3.0e0/hm1 ;U[3][6]=  3.0e0/(hm1-c3);
      U[3][7]=  3.0e0/(hm2-c2);U[3][8]=  3.0e0/(hm2-hm1);
      U[3][9]=  3.0e0/(hm3-hm1);U[3][10]= 3.0e0/(hm3-hm2);
      U[3][11]= 3.0e0/(hm4-hm2);U[3][12]= 3.0e0/(hm4-hm3);
      U[3][13]= 3.0e0/(hm5-hm3);U[3][14]= 3.0e0/(hm5-hm4);
      U[4][5]= 4.0e0/hm1 ;U[4][6]= 4.0e0/hm1;
      U[4][7]= 4.0e0/(hm2-c3);U[4][8]= 4.0e0/(hm2-c2);
      U[4][9]= 4.0e0/(hm3-hm1);U[4][10]=4.0e0/(hm3-hm1);
      U[4][11]=4.0e0/(hm4-hm2);U[4][12]=4.0e0/(hm4-hm2);
      U[4][13]=4.0e0/(hm5-hm3);U[4][14]=4.0e0/(hm5-hm3);
      U[5][6]= 5.0e0/hm1;U[5][7]= 5.0e0/hm2;
      U[5][8]= 5.0e0/(hm2-c3);U[5][9]= 5.0e0/(hm3-c2);
      U[5][10]=5.0e0/(hm3-hm1);U[5][11]=5.0e0/(hm4-hm1);
      U[5][12]=5.0e0/(hm4-hm2);U[5][13]=5.0e0/(hm5-hm2);
      U[5][14]=5.0e0/(hm5-hm3);U[6][7]= 6.0e0/hm2;
      U[6][8]= 6.0e0/hm2;U[6][9]= 6.0e0/(hm3-c3);
      U[6][10]=6.0e0/(hm3-c2);U[6][11]=6.0e0/(hm4-hm1);
      U[6][12]=6.0e0/(hm4-hm1);U[6][13]=6.0e0/(hm5-hm2);
      U[6][14]=6.0e0/(hm5-hm2);U[7][8]= 7.0e0/hm2;
      U[7][9]= 7.0e0/hm3;U[7][10]=7.0e0/(hm3-c3);
      U[7][11]=7.0e0/(hm4-c2);U[7][12]=7.0e0/(hm4-hm1);
      U[7][13]=7.0e0/(hm5-hm1);U[7][14]=7.0e0/(hm5-hm2);
      U[8][9]= 8.0e0/hm3;U[8][10]=8.0e0/hm3;
      U[8][11]=8.0e0/(hm4-c3);U[8][12]=8.0e0/(hm4-c2);
      U[8][13]=8.0e0/(hm5-hm1);U[8][14]=8.0e0/(hm5-hm1);
      U[9][10]=9.0e0/hm3;U[9][11]=9.0e0/hm4;
      U[9][12]=9.0e0/(hm4-c3);U[9][13]=9.0e0/(hm5-c2);
      U[9][14]=9.0e0/(hm5-hm1);U[10][11]=10.0e0/hm4;
      U[10][12]=10.0e0/hm4;U[10][13]=10.0e0/(hm5-c3);
      U[10][14]=10.0e0/(hm5-c2);U[11][12]=11.0e0/hm4;
      U[11][13]=11.0e0/hm5;U[11][14]=11.0e0/(hm5-c3);
      U[12][13]=12.0e0/hm5;U[12][14]=12.0e0/hm5;
      U[13][14]=13.0e0/hm5;

for (i = 1; i <= n; i++)
 {
    b_v2[i]=RHS[i] ;
 }

for (k = 3; k <= nm1; k++)
 {
```

```
      for (idx = n ; idx >= k+1 ; idx--)
      {
         b_v2[ idx]=b_v2[ idx-1]+lop[k][idx]*b_v2[ idx] ;
      }
   }

   for (k = 1; k <= n; k++)
   {
      b_v2[k]= invdiag[k]* b_v2[k] ;
   }

  for (k = n-1; k >= 2 ; k--)
   {
      for (idx = k+1 ; idx <= n ; idx++)
      {
          b_v2[idx]= U[k][idx] *b_v2[idx] ;
      }

      for (idx = k ; idx <= n-1 ; idx++)
      {
          b_v2[idx]= b_v2[idx]-b_v2[idx+1] ;
      }
   }

    b_v2[3]= U[1][ 3] *b_v2[3] ;
    b_v2[4]= U[1][ 4] *b_v2[4] ;

  np12=(n+1)/2 ;
  lim1=np12*2-1;

  for (idx = 5 ; idx <= lim1 ; idx +=2 )
    {
      b_v2[idx]= U[1][idx] *b_v2[idx] ;
    }

    b_v2[1]= b_v2[1]-b_v2[3 ] ;
    b_v2[3]= b_v2[3]-b_v2[4 ] ;

  if (n >= 5) b_v2[4]= b_v2[4]-b_v2[5 ] ;

  for (idx = 5 ; idx <= lim1-2 ; idx +=2 )
    {
      b_v2[idx]= b_v2[ idx]-b_v2[ idx+2] ;
    }

for (idx = 1 ; idx <= 15 ; idx +=1 )
```

```
    {
      b_v[idx]=0.0;
     }

b_v[1]=b_v2[3] ; b_v[2]=b_v2[4]; b_v[3]=b_v2[1];

ic=4 ;
for (idx = 5 ; idx <= n ; idx +=2 )
    {
     b_v[ic]=b_v2[idx];
     ic=ic+1;
    }

ic=9; b_v[ic]=b_v2[2]; ic=ic+1;

for (idx = 6 ; idx <= n ; idx +=2 )
    {
     b_v[ic]=b_v2[idx];
     ic=ic+1;
    }

LP20n=0;

for (idx = 5 ; idx <= n ; idx +=2 )
    {
     LP20n= LP20n+ b_v2[idx]*pow(A_MAT2col[idx],(double)n-1.0) ;
    }

LP20n= LP20n/ft[n-1] ; LP20np1=LP20n ;LP20n=0;

for (idx = 6 ; idx <= n ; idx +=2 )
    {
     LP20n= LP20n+ b_v2[idx]*pow(A_MAT2col[idx],n-2) ;
    }

LP20n= LP20n/ft[n-2] ; LP20ntot=LP20np1+LP20n ;
b_v[15]=LP20ntot ;

return;
}

static void HB04_14d12_4SC(double *xtab,double *parout) {

       int i,n, idx = 0;
       double  lop[17][17], U[17][17], orbv[5][17], A_MAT2col[17] ;
       double A_MAT2coln[17], A_MAT2colnm1[17] ;
```

```
double invdiag[17];
for(i=0;i<17;i++)
invdiag[i]=0.0;

double b_v[17], b_v2[17], DLbv[17];
double  RHS[17];

double x, xkm11, xkm10,  xkm9 , xkm8 , xkm7 , xkm6 ,
       xkm5 , xkm4 ,xkm3 , xkm2 ;
double xkm1 , xk1, xk2,xk3, c2,  c3  ;
double hl ,h0,h1, h2, hm1 , hm2 , hm3 , hm4 , hm5 , hm6 ,
       hm7 , hm8, hm9 , hm10 , hm11  ;
double ft2, ft3, ft4 , ft5 ,ft6 ,ft7 ,ft8 , ft9 , ft10 ,
       ft11 , ft12, ft13 ,  ft14 ,  ft15 ,  ft16 , ft17  ;
double ift2,ift3,ift4,ift5,ift6,ift7,ift8,ift9,ift10,ift11,
       ift12,ift13, ift14,ift15,ift16,ift17;
double c32,c33,c34,c35,c36,c37,c38,c39,c310,c311,c312,c313, c314;
double c22,c23,c24,c25,c26,c27,c28,c29,c210,c211,c212,c213,c214,c215;
double h;
double multip;
double LP2015, lhs215, b3, b2, b1 ;
double  stfac, tc, hmin ;
int nm1, nm2, nm3, k, meth_n, nlim, nmax,nm12,lim1, ic, jdx, spcc ;

x      = xtab[1] ;xkm11  = xtab[2] ;xkm10  = xtab[3] ;
xkm9   = xtab[4] ;xkm8   = xtab[5] ;xkm7   = xtab[6] ;
xkm6   = xtab[7] ;xkm5   = xtab[8] ;xkm4   = xtab[9] ;
xkm3   = xtab[10] ;xkm2   = xtab[11] ;xkm1   = xtab[12] ;
xk1    = xtab[13] ;xk2    = xtab[14] ;xk3    = xtab[15] ;
c2     = xtab[16] ;c3     = xtab[17] ;LP2015 = xtab[18] ;
lhs215 = xtab[19] ;b3     = xtab[20] ;b2     = xtab[21] ;
b1     = xtab[22] ;

meth_n=(int)xtab[40]  ;

n= meth_n+1  ;
if( n>14 )
    n=14;

nlim=meth_n+1 ;

stfac=0.9 ;
tc=xk1 ;
hmin = 3.552713678800501e-15*fabs( xk1);
```

```
xk3 = x ;

hl = (xk3-xk1) ;
h=hl;

hm11= (xkm11-xk1)/hl;hm10= (xkm10-xk1)/hl;
hm9 = (xkm9-xk1)/hl;hm8 = (xkm8-xk1)/hl;
hm7 = (xkm7-xk1)/hl;hm6 = (xkm6-xk1)/hl;
hm5 = (xkm5-xk1)/hl;hm4 = (xkm4-xk1)/hl;
hm3 = (xkm3-xk1)/hl;hm2 = (xkm2-xk1)/hl;
hm1 = (xkm1-xk1)/hl;h0 = xk1-xk1;
h1 = (xk2 - xk1)/hl ;h2 = (xk3 - xk1)/hl ;

ft2=2.0 ;ft3= 6.0 ;ft4 =    24.0;
ft5 =   120.0;ft6 =   720.0;ft7 =        5040.0;
ft8 =       40320.0;ft9 =      362880.0;
ft10 =     3628800.0;ft11 =    39916800.0;
ft12 =   479001600.0;ft13 =    6.227020800000000e+09  ;
ft14 =     8.717829120000000e+10;ft15 =     1.307674368000000e+12;
ft16 =     2.092278988800000e+13;ft17 =     3.556874280960000e+14 ;
ift2 =     5.000000000000000e-01  ;ift3 =     1.666666666666667e-01 ;
ift4 =     4.166666666666666e-02 ;ift5 =     8.333333333333333e-03 ;
ift6 =     1.388888888888889e-03 ;ift7 =     1.984126984126984e-04 ;
ift8 =     2.480158730158730e-05 ;ift9 =     2.755731922398589e-06 ;
ift10 =     2.755731922398589e-07;ift11 =     2.505210838544172e-08;
ift12 =     2.087675698786810e-09;ift13 =     1.605904383682161e-10;
ift14 =     1.147074559772972e-11;ift15 =     7.647163731819816e-13;
ift16 =     4.779477332387385e-14;ift17 =     2.811457254345521e-15;

c32= c3*c3/2 ;c33= c32*c3/3  ;c34= c33*c3/4  ;
c35= c34*c3/5  ;c36= c35*c3/6  ;c37= c36*c3/7  ;
c38= c37*c3/8  ;c39= c38*c3/9  ;c310=c39*c3/10  ;
c311=c310*c3/11  ;c312=c311*c3/12  ;c313=c312*c3/13  ;
c314=c313*c3/14  ;c22= c2*c2/2 ;c23= c22*c2/3  ;
c24= c23*c2/4  ; c25= c24*c2/5  ;c26= c25*c2/6  ;
c27= c26*c2/7  ;c28= c27*c2/8  ;c29= c28*c2/9  ;
c210=c29*c2/10  ;c211=c210*c2/11  ;c212=c211*c2/12  ;
c213=c212*c2/13  ;c214=c213*c2/14  ;c215=c214*c2/15  ;

nm1=n-1;nm2=n-2;nm3=n-3;

RHS[0]= 0.0;RHS[1]=  1.0  ;RHS[2]=  ift2  ;
RHS[3]=  ift3  ;RHS[4]=  ift4  ;RHS[5]=  ift5  ;
RHS[6]=  ift6  ;RHS[7]=  ift7  ;RHS[8]=  ift8  ;
RHS[9]=  ift9  ;RHS[10]= ift10  ;RHS[11]= ift11 ;
RHS[12]= ift12 ;RHS[13]= b3-0.025  ;RHS[14]= b2+0.029 ;
```

```
     RHS[15]= 0.0  ;RHS[16]= 0.0 ;
for (idx = 1; idx <=n ; idx++)
{
   invdiag[idx]=1.0e0;
}

A_MAT2col[0] =0.0 ;A_MAT2col[1] =0.0 ;A_MAT2col[2] =0.0;
A_MAT2col[3] =hm1 ;A_MAT2col[4] =hm1 ;A_MAT2col[5] =hm2 ;
A_MAT2col[6]=hm2 ;A_MAT2col[7] =hm3 ;A_MAT2col[8] =hm3 ;
A_MAT2col[9] =hm4 ;A_MAT2col[10]=hm4 ;A_MAT2col[11]=hm5  ;
A_MAT2col[12]=hm5  ;A_MAT2col[13]=c3  ;A_MAT2col[14]=c2  ;
A_MAT2col[15]=0.0  ;    A_MAT2col[16]=0.0  ;

for (idx = 4; idx <= n-2 ; idx++)
{
       multip =-A_MAT2col[idx-1]/(idx-1) ;
       invdiag[idx]= invdiag[idx-1]*multip  ;
}

lop[3][4] = -3/hm1  ;lop[3][5] = -4/hm1  ;
lop[3][6] = -5/hm1  ;lop[3][7] = -6/hm1  ;
lop[3][8] = -7/hm1  ;lop[3][9] = -8/hm1  ;
lop[3][10]= -9/hm1  ;lop[3][11]= -10/hm1  ;
lop[3][12]= -11/hm1  ;lop[3][13]= -12/hm1  ;
lop[4][5] = -3/hm1  ;lop[4][6] = -4/hm1  ;
lop[4][7] = -5/hm1  ;lop[4][8] = -6/hm1  ;
lop[4][9] = -7/hm1  ;lop[4][10]= -8/hm1  ;
lop[4][11]= -9/hm1  ;lop[4][12]= -10/hm1  ;
lop[4][13]= -11/hm1  ;lop[5][6] = -3/hm2  ;
lop[5][7] = -4/hm2  ;lop[5][8] = -5/hm2  ;
lop[5][9] = -6/hm2  ;lop[5][10]= -7/hm2  ;
lop[5][11]= -8/hm2  ;lop[5][12]= -9/hm2  ;
lop[5][13]= -10/hm2  ;lop[6][7] = -3/hm2  ;
lop[6][8] = -4/hm2  ;lop[6][9] = -5/hm2  ;
lop[6][10]= -6/hm2  ;lop[6][11]= -7/hm2  ;
lop[6][12]= -8/hm2  ;lop[6][13]= -9/hm2  ;
lop[7][8] = -3/hm3  ;lop[7][9] = -4/hm3  ;
lop[7][10]= -5/hm3  ;lop[7][11]= -6/hm3  ;
lop[7][12]= -7/hm3  ;lop[7][13]= -8/hm3  ;
lop[8][9] = -3/hm3  ;lop[8][10]= -4/hm3  ;
lop[8][11]= -5/hm3  ;lop[8][12]= -6/hm3  ;
lop[8][13]= -7/hm3  ;lop[9][10]= -3/hm4  ;
lop[9][11]= -4/hm4  ;lop[9][12]= -5/hm4  ;
lop[9][13]= -6/hm4  ;lop[10][11]= -3/hm4  ;
lop[10][12]= -4/hm4  ;lop[10][13]= -5/hm4  ;
lop[11][12]= -3/hm5  ;lop[11][13]= -4/hm5  ;
```

```
  lop[12][13]= -3/hm6  ;

U[1][3]=  1/hm1; U[1][5]=  1/(hm2-hm1); U[1][7]=  1/(hm3-hm2);
U[1][9]=  1/(hm4-hm3); U[1][11]= 1/(hm5-hm4); U[1][13]= 1/(hm6-hm5);
U[2][3]=  2/hm1; U[2][4]=  2/hm1; U[2][5]=  2/(hm2-hm1);
U[2][6]=2/(hm2-hm1); U[2][7]=  2/(hm3-hm2); U[2][8]=  2/(hm3-hm2);
U[2][9]= 2/(hm4-hm3); U[2][10]= 2/(hm4-hm3); U[2][11]= 2/(hm5-hm4);
U[2][12]= 2/(hm5-hm4); U[2][13]= 2/(hm6-hm5); U[3][4]=  3/hm1;
U[3][5]=  3/hm2 ; U[3][6]=  3/(hm2-hm1); U[3][7]= 3/(hm3-hm1);
U[3][8]=  3/(hm3-hm2); U[3][9]=  3/(hm4-hm2); U[3][10]= 3/(hm4-hm3);
U[3][11]= 3/(hm5-hm3); U[3][12]= 3/(hm5-hm4); U[3][13]= 3/(hm6-hm4);
U[4][5]=  4/hm2 ; U[4][6]=  4/hm2; U[4][7]=  4/(hm3-hm1);
U[4][8]=4/(hm3-hm1); U[4][9]=  4/(hm4-hm2); U[4][10]= 4/(hm4-hm2);
U[4][11]= 4/(hm5-hm3); U[4][12]= 4/(hm5-hm3); U[4][13]= 4/(hm6-hm4);
U[5][6]=  5/hm2; U[5][7]=  5/hm3; U[5][8]=  5/(hm3-hm1);
U[5][9]=5/(hm4-hm1); U[5][10]= 5/(hm4-hm2); U[5][11]= 5/(hm5-hm2);
U[5][12]= 5/(hm5-hm3); U[5][13]= 5/(hm6-hm3); U[6][7]=  6/hm3;
U[6][8]=  6/hm3; U[6][9]=  6/(hm4-hm1); U[6][10]= 6/(hm4-hm1);
U[6][11]= 6/(hm5-hm2); U[6][12]= 6/(hm5-hm2); U[6][13]= 6/(hm6-hm3);
U[7][8]=  7/hm3; U[7][9]=  7/hm4; U[7][10]= 7/(hm4-hm1);
U[7][11]=7/(hm5-hm1); U[7][12]= 7/(hm5-hm2); U[7][13]= 7/(hm6-hm2);
U[8][9]=  8/hm4; U[8][10]= 8/hm4; U[8][11]= 8/(hm5-hm1);
U[8][12]=8/(hm5-hm1); U[8][13]= 8/(hm6-hm2); U[9][10]=9/hm4;
U[9][11]=9/hm5; U[9][12]=9/(hm5-hm1); U[9][13]=9/(hm6-hm1);
U[10][11]=10/hm5; U[10][12]=10/hm5; U[10][13]=10/(hm6-hm1);
U[11][12]=11/hm5; U[11][13]=11/hm6; U[12][13]=12/hm6;

  for (i = 1; i <= 14 ; i++)
   {
      b_v2[i]=RHS[i] ;
   }

   A_MAT2colnm1[0] = 0.0 ;A_MAT2colnm1[1] = 1.0 ;
   A_MAT2colnm1[2] = c3 ;A_MAT2colnm1[3] = c32 ;
   A_MAT2colnm1[4] = c33 ;A_MAT2colnm1[5] = c34 ;
   A_MAT2colnm1[6] = c35 ;A_MAT2colnm1[7] = c36 ;
   A_MAT2colnm1[8] = c37 ;A_MAT2colnm1[9] = c38 ;
   A_MAT2colnm1[10]= c39 ;A_MAT2colnm1[11]= c310 ;
   A_MAT2colnm1[12]= c311 ;A_MAT2colnm1[13] =0.0 ;
   A_MAT2colnm1[14]= 0.0 ;A_MAT2colnm1[15]= 0.0 ;
   A_MAT2colnm1[16]= 0.0 ;A_MAT2coln[0] = 0.0 ;
   A_MAT2coln[1] = 1.0 ;A_MAT2coln[2] = c2 ;
   A_MAT2coln[3] = c22 ;A_MAT2coln[4] = c23 ;
   A_MAT2coln[5] = c24 ;A_MAT2coln[6] = c25 ;
   A_MAT2coln[7] = c26 ;A_MAT2coln[8] = c27 ;
   A_MAT2coln[9] = c28 ;A_MAT2coln[10]= c29 ;
```

```
   A_MAT2coln[11]= c210 ;A_MAT2coln[12]= c211 ;
   A_MAT2coln[13] =0.0 ;A_MAT2coln[14]= 0.0 ;
   A_MAT2coln[15]= 0.0 ;A_MAT2coln[16]= 0.0 ;

  for (k = 12; k >= 1 ; k--)
   {
    b_v2[k]= b_v2[k]-A_MAT2colnm1[k]* b_v2[13] ;
   }

  for (k = 12; k >= 1 ; k--)
   {
    b_v2[k]= b_v2[k]-A_MAT2coln[k]* b_v2[14] ;
   }

  for (k = 3; k <= 11; k++)
   {
      for (idx = 12  ; idx >= k+1 ; idx--)
      {
         b_v2[idx]=b_v2[idx-1]+lop[k][idx]*b_v2[idx] ;
      }
   }

   for (k = 1; k <= 12 ; k++)
   {
      b_v2[k]= invdiag[k]* b_v2[k] ;
   }

   for (k = 1; k <= 14 ; k++)
   {
     DLbv[k]=b_v2[k];
   }

   ic=0;

   for (idx = 1; idx <= 4; idx++)
      {
      for (jdx = 1; jdx <= 15; jdx++)
         {
          orbv[idx][jdx]=0.0 ;
         }
      }

     spcc=0;

nmax=nlim; if (nmax > 14) nmax=14 ;
```

```
for (n = nlim-3; n <= nmax ; n++)
   {
    for (k = 1; k <= 14 ; k++)
      {
       b_v2[k]= DLbv[k];
      }

   for (k = n-3; k >= 2 ; k-=1 )
      {
      for (idx = k+1; idx <= n-2; idx++)
         {
          b_v2[idx]= U[k][ idx] *b_v2[idx] ;
         }
      for (idx = k; idx <= n-3; idx++)
         {
          b_v2[idx]= b_v2[ idx]-b_v2[ idx+1] ;
          }
      }

     nm12=(n-1)/2 ;
     lim1=nm12*2-1;

  for (idx = 3; idx <= lim1 ; idx+=2)
     {
      b_v2[idx]= U[1][idx] *b_v2[idx] ;
      }
  for (idx = 1; idx <= lim1-2 ; idx+=2)
     {
     b_v2[idx]= b_v2[ idx]-b_v2[ idx+2] ;
     }
  for (idx = 1; idx <= 14 ; idx+=1)
     {
     b_v[idx]=0.0;
     }

   b_v[1]=b_v2[13] ;
   b_v[2]=b_v2[14];

   ic=3 ;

  for (idx = 1; idx <= n-2  ; idx+=2)
     {
      b_v[ic]=b_v2[idx];
      ic=ic+1;
      }
```

```
ic=9 ;

if (n >= 4) b_v[ic]=b_v2[2];

ic=ic+1;
for (idx = 4; idx <= n-2  ; idx+=2)
     {
      b_v[ic]=b_v2[idx];
      ic=ic+1;
     }

spcc=spcc+1;

for (idx = 1; idx <= 14  ; idx+=1)
     {
      orbv[spcc][idx]=b_v[idx] ;
     }
   }

for (idx = 1; idx <= 4  ; idx+=1)
     {
     for (jdx = 1; jdx <= 14  ; jdx+=1)
         {
          parout[(idx-1)*14+jdx]=  orbv[idx][ jdx]  ;
          }
     }

return;
}

static void  HB04_14_3Ed1_d2prgsim(double RELTOL,double
TOL,std::string fname,double t0,double tend,double* y0,int
nbcmp1,double* tvals,double** yvals,double** fvals,double**
f2vals,double* res) {

double hmax = 0.1*fabs(tend-t0); double stfac=0.81 ; int nbinitst =
3; int nbeps=4 ; int iniordk=4 ; int ordnew; int jpnbpos; int
m6jd,m5jd,m4jd,m3jd,m2jd,m1jd,m0jd;
double xkm14,xkm13,xkm12,xkm11,xkm10,xkm9,xkm8,xkm7,
       xkm6,xkm5,xkm4,xkm3,xkm2,xkm1; double xk1,xk2,xk3,xk4;
double tc; double b1,b2,b3,b4;
double am1201,am2201,am3201,am4201,am5201;
double alp203,am1202,am2202,am3202,am4202,am5202;
double LP2014,yk1v,ykm1v,lhs214; double temp;
double err_estkp1,err_est,err_estkm1,err_estkm2;
double hold,tnew,stsize,prevh;
```

```
double* temp1=new double[nbcmp1]; double* temp2=new double[nbcmp1];;
double* yk1=new double[nbcmp1]; double* y_jpc2=new double[nbcmp1];
double* y_jpc3=new double[nbcmp1]; double* ynew=new double[nbcmp1];

double* Qbv1_8=new double[9]; double* Qbv9_14=new double[7]; double*
hQbv1_8=new double[9]; double* h2Qbv9_14=new double[7]; double*
bv1_8=new double[9]; double* bv9_14=new double[7]; double*
hbv1_8=new double[9]; double* h2bv9_14=new double[7]; double*
SCPbv1_8k=new double[9]; double* SCPbv9_14k=new double[7]; double*
hSCPbv1_8k=new double[9]; double* h2SCPbv9_14k=new double[7];
double* SCPbv1_8km2=new double[9]; double* SCPbv9_14km2=new
double[7]; double* SCPbv1_8km1=new double[9]; double*
SCPbv9_14km1=new double[7]; double* SCPbv1_8kp1=new double[9];
double* SCPbv9_14kp1=new double[7]; double* hSCPbv1_8km2=new
double[9]; double* h2SCPbv9_14km2=new double[7]; double*
hSCPbv1_8km1=new double[9]; double* h2SCPbv9_14km1=new double[7];
double* hSCPbv1_8kp1=new double[9]; double* h2SCPbv9_14kp1=new
double[7];

int index1,index2;

double* y_est=new double[nbcmp1];

double* scpbv=new double[60];

for(index1=0;index1<60;index1++) scpbv[index1]=0.0;

double* y_estkm2=new double[nbcmp1]; double* y_estkm1=new
double[nbcmp1]; double* y_estkp1=new double[nbcmp1];

double* b_v=new double[17];

for(index1=0;index1<=16;index1++)
b_v[index1]=0.0;

double* parout = new double[17];

for(index1=0;index1<=16;index1++)
parout[index1]=0.0;

double eps = 2.220446049250313e-16 ;
int q= 1; int nbpos = 10;
tc=t0;

double* yc=new double[nbcmp1];
```

```
for(index1=1;index1<nbcmp1;index1++)
yc[index1]=y0[index1];

int k=1111;
int j=0;
int oc=3 ;
int jpnbposp1;
double h=0.0;

double* RESULT=
HB04d1d2initstepsprg(RELTOL,TOL,fname,t0,tend,y0,nbinitst,tvals,yvals,
                     fvals,f2vals,nbcmp1,res,j,h);
j =(int)RESULT[0];
h =RESULT[1];

int HB04_14prg_j=j;
double HB04_14prg_tvalsjpnbpos  =tvals[j+nbpos];

//yc=yvals(:,j+nbpos);
for(index1=1;index1<nbcmp1;index1++) {
    yc[index1]=yvals[index1][j+nbpos];
}

tc = tvals[j+nbpos] ; double hv =(4.0/5.0)* h;

for(index1=4;index1<=12;index1++) {
    tvals[j+nbpos-index1]=tc-(double)index1*h;
}
double MAXGE = -9999.0; oc=1111 ; int abmhbnfe=nbinitst*2 ;

double t2 = t0;

double* y2=new double[nbcmp1];

for(index1=1;index1<nbcmp1;index1++) y2[index1]=y0[index1];

int nreje=0; tc = tvals[j+nbpos]; double hnew = h ;

// INIT PARAM.
double c1=0.0,c2=2.0/3.0,c3=1.0,c4=1.0;
double c_i[] = {0.0,2.0/3.0,1.0};
double hc_i[] ={0.0,h*(2.0/3.0),h*1.0};


double compeps = 2.220446049250313e-16 ;
double hmin = 16.0*compeps*fabs(tc);
```

```
double ft2= 2.0; double ft3= 6.0; double ft4 = 24.0 ; double ft5 =
120.0 ; double ft6 = 720.0 ; double ft7 = 5040.0 ; double ft8 =
40320.0; double ft9 = 362880.0 ; double ft10 = 3628800.0 ; double
ft11 = 39916800.0 ; double ft12 = 479001600.0 ; double ft13 =
6.227020800000000e+09 ; double ft14 =     8.717829120000000e+10 ;
double ft15 =     1.307674368000000e+12 ; double ft16 =
2.092278988800000e+13 ; double ft17 =     3.556874280960000e+14 ;

double ift2 =     5.000000000000000e-01  ; double ift3 =
1.666666666666667e-01 ; double ift4 =     4.166666666666666e-02 ;
double ift5 =     8.333333333333333e-03 ; double ift6 =
1.388888888888889e-03 ; double ift7 =     1.984126984126984e-04 ;
double ift8 =     2.480158730158730e-05 ; double ift9 =
2.755731922398589e-06 ; double ift10 =     2.755731922398589e-07;
double ift11 =     2.505210838544172e-08; double ift12 =
2.087675698786810e-09; double ift13 =     1.605904383682161e-10;
double ift14 =     1.147074559772972e-11; double ift15 =
7.647163731819816e-13; double ift16 =     4.779477332387385e-14;
double ift17 =     2.811457254345521e-15;

double c22= c2*c2/2.0 ; double c23= c22*c2/3.0  ; double c24=
c23*c2/4.0  ; double c25= c24*c2/5.0  ; double c26= c25*c2/6.0  ;
double c27= c26*c2/7.0  ; double c28= c27*c2/8.0  ; double c29=
c28*c2/9.0  ; double c210=c29*c2/10.0  ; double c211=c210*c2/11.0  ;
double c212=c211*c2/12.0  ; double c213=c212*c2/13.0  ; double
c214=c213*c2/14.0  ;

double c32= c3*c3/2.0 ; double c33= c32*c3/3.0  ; double c34=
c33*c3/4.0  ; double c35= c34*c3/5.0  ; double c36= c35*c3/6.0  ;
double c37= c36*c3/7.0  ; double c38= c37*c3/8.0  ; double c39=
c38*c3/9.0  ; double c310=c39*c3/10.0  ; double c311=c310*c3/11.0  ;
double c312=c311*c3/12.0  ; double c313=c312*c3/13.0  ; double
c314=c313*c3/14.0  ;

// col1_2(:,1)=  yvals(m0jd ,:); ...

double** col1_8=new double*[nbcmp1];
for(index1=0;index1<nbcmp1;index1++) {
    col1_8[index1]=new double[9];

    for(index2=0;index2<=8;index2++)
     col1_8[index1][index2]=0.0;
}

double** col9_14=new double*[nbcmp1];
```

```
for(index1=0;index1<nbcmp1;index1++) {
    col9_14[index1]=new double[7];

   for(index2=0;index2<=6;index2++)
        col9_14[index1][index2]=0.0;
}

int ordk=iniordk ; double  erk= 0.0; int  nbrep=0;

double* xtab=new double[41];
//zeros xtab

for(index1=0;index1<41;index1++) xtab[index1]=0.0;

double* fnewk2=new double[nbcmp1]; double* fnewk3=new
double[nbcmp1]; double* fnew=new double[nbcmp1]; double* f2new=new
double[nbcmp1];

//( MAIN LOOP  1 integration step/ iteration)

while( tvals[j+nbpos] < tend )
{
   hmin = 16*eps*fabs(tc);
   h=hnew;

   if((tc+h) > tend)
       h=tend-tc ;

// ------- (MAIN INTEG STEP) --------------

err_est=9999.0;

// LOOP REPEAT THE STEP UNTIL SUCCESS ----------------------
  while ( err_est > TOL  )
{
jpnbpos=j+nbpos;

m6jd=jpnbpos-6  ;m5jd=jpnbpos-5  ;m4jd=jpnbpos-4  ;m3jd=jpnbpos-3  ;
m2jd=jpnbpos-2  ;m1jd=jpnbpos-1  ;m0jd=jpnbpos  ;

xkm14 =0.0 ;xkm13 = 1e-11  ;xkm12 = 2e-11  ;xkm11 = 3e-11  ;xkm10 =
4e-11  ; xkm9 =  5e-11  ;xkm8 =  6e-11  ;xkm7 =  7e-11  ; xkm6 =
tvals[m6jd] ;xkm5 = tvals[m5jd] ; xkm4 = tvals[m4jd] ; xkm3 =
tvals[m3jd] ; xkm2 = tvals[m2jd] ; xkm1 = tvals[m1jd] ; xk1 =
tvals[m0jd] ; xk2 = tvals[jpnbpos] + c2* h; tc=xk1 ; xk3 =
tvals[jpnbpos] + c3*h; xk4 = tvals[jpnbpos] + c4*h;
```

```
if(ordk < 4)
   ordk=4 ;

if( ordk > 14)
   ordk=14 ;

xtab[1] = xk3   ; xtab[2] = xkm11 ; xtab[3] = xkm10 ; xtab[4] = xkm9
; xtab[5] = xkm8  ; xtab[6] = xkm7  ; xtab[7] = xkm6  ; xtab[8] =
xkm5  ; xtab[9] = xkm4  ; xtab[10] =xkm3  ; xtab[11] =xkm2  ;
xtab[12] =xkm1  ; xtab[13] =xk1   ; xtab[14] =xk2   ; xtab[15] =xk3
; xtab[16] =c2    ; xtab[17] =c3    ; xtab[40] =(double)ordk  ;

HB04_14_3Ed1_d2Qc( xtab,b_v);

// der1:
  b3   = b_v[1] ;
  b2   = b_v[2]   ;
  b1   = b_v[3]   ;
  b4=0.0 ;

// ( alp(20,2) = acf(20,1 ) = b1 )

  am1201   = b_v[4] ;am2201   = b_v[5] ;
  am3201   = b_v[6] ;am4201   = b_v[7] ;
  am5201   = b_v[8] ;
// der2:
  alp203   = b_v[9]  ;am1202   = b_v[10] ;
  am2202   = b_v[11] ;am3202   = b_v[12] ;
  am4202   = b_v[13] ;am5202   = b_v[14] ;
  LP2014   = b_v[15] ;
  yk1v = (exp(-xk1)) ;
  ykm1v = (exp(-xkm1)) ;

for(index1=1;index1<=8;index1++)
{
  Qbv1_8[index1]=b_v[index1];
  hQbv1_8[index1]=h*b_v[index1];
}

for(index1=1;index1<=6;index1++)
{
  Qbv9_14[index1]=b_v[index1+8];
  h2Qbv9_14[index1]=h*h*b_v[index1+8];
}
```

```
for(index1=1;index1<nbcmp1;index1++) {
    yk1[index1]= yvals[index1][m0jd] ;}

for(index1=1;index1<nbcmp1;index1++) {
    col1_8[index1][3]=fvals[index1][m0jd];
    col1_8[index1][4]=fvals[index1][m1jd];
    col1_8[index1][5]=fvals[index1][m2jd];
    col1_8[index1][6]=fvals[index1][m3jd];
    col1_8[index1][7]=fvals[index1][m4jd];
    col1_8[index1][8]=fvals[index1][m5jd];
    col9_14[index1][1]=f2vals[index1][m0jd];
    col9_14[index1][2]=f2vals[index1][m1jd];
    col9_14[index1][3]=f2vals[index1][m2jd];
    col9_14[index1][4]=f2vals[index1][m3jd];
    col9_14[index1][5]=f2vals[index1][m4jd];
    col9_14[index1][6]=f2vals[index1][m5jd];
}

     xtab[20] =  b3    ; xtab[21] =  b2      ;
     xtab[22] =  b1      ;xtab[23] =  am1201    ;
     xtab[24] =  am2201    ;xtab[25] =  am3201    ;
     xtab[26] =  am4201    ;xtab[27] =  am5201    ;
     xtab[28] =  alp203    ;xtab[29] =  am1202    ;
     xtab[30] =  am2202    ;xtab[31] =  am3202    ;
     xtab[32] =  am4202    ;xtab[33] =  am5202    ;
     xtab[36] = TOL    ; xtab[37] = (double)nreje    ;

// PART 2:   HBO2_12d12prd1csim

        xtab[1] = xk2 ;
        xtab[40] =(double)ordk-2.0    ;

        HBO2_12d12prd1( xtab,b_v) ;

        lhs214= b_v[1]  ;

         bv1_8[1]=0.0;bv1_8[2]=0.0;
         hbv1_8[1]=0.0;hbv1_8[2]=0.0;

         for(index1=3;index1<=8;index1++)
         {
          bv1_8[index1]=b_v[index1-1];
          hbv1_8[index1]=h*bv1_8[index1];
         }

         for(index1=1;index1<=6;index1++)
```

```
        {
          bv9_14[index1]=b_v[index1+7];
          h2bv9_14[index1]=h*h*bv9_14[index1];
      }

  // y_jpc2 = yk1 + col1_8*hbv1_8 + col9_14*h2bv9_14  ;
    for(index1=1;index1<nbcmp1;index1++)

y_jpc2[index1]=yk1[index1]+(col1_8[index1][1]*hbv1_8[1]+
               col1_8[index1][2]*hbv1_8[2]+col1_8[index1][3]*hbv1_8[3]
               +col1_8[index1][4]*hbv1_8[4]+col1_8[index1][5]*hbv1_8[5]
               +col1_8[index1][6]*hbv1_8[6]+col1_8[index1][7]*hbv1_8[7]
               +col1_8[index1][8]*hbv1_8[8])+(col9_14[index1][1]*h2bv9_14[1]

+col9_14[index1][2]*h2bv9_14[2]+col9_14[index1][3]*h2bv9_14[3]

+col9_14[index1][4]*h2bv9_14[4]+col9_14[index1][5]*h2bv9_14[5]
               +col9_14[index1][6]*h2bv9_14[6]);

       FirstD(fname,xk2,y_jpc2,nbcmp1,fnewk2);

       for(index1=1;index1<nbcmp1;index1++)
        col1_8[index1][2]=fnewk2[index1];

//   PART (Calculate 2nd of 2 predictors)
//      (at point tc + c3*h)

        xtab[1] = xk3  ;xtab[18] = LP2014 ;
        xtab[19] = lhs214 ;xtab[40] =(double)ordk-1.0    ;

  HBO2_12d12prd2( xtab,parout) ;

         bv1_8[1]=0.0;
         hbv1_8[1]=0.0;
         for(index1=2;index1<=8;index1++)
         {  bv1_8[index1]=parout[index1-1];
            hbv1_8[index1]=h*bv1_8[index1];
         }
         for(index1=1;index1<=6;index1++)
         {  bv9_14[index1]=parout[index1+7];
            h2bv9_14[index1]=h*h*bv9_14[index1];
         }

     // y_jpc3 = yk1 + col1_8*hbv1_8 + col9_14*h2bv9_14  ;
    for(index1=1;index1<nbcmp1;index1++)
```

```
y_jpc3[index1]=yk1[index1]+(col1_8[index1][1]*hbv1_8[1]
      +col1_8[index1][2]*hbv1_8[2]+col1_8[index1][3]*hbv1_8[3]
      +col1_8[index1][4]*hbv1_8[4]+col1_8[index1][5]*hbv1_8[5]
      +col1_8[index1][6]*hbv1_8[6]+col1_8[index1][7]*hbv1_8[7]
      +col1_8[index1][8]*hbv1_8[8])+(col9_14[index1][1]*h2bv9_14[1]
      +col9_14[index1][2]*h2bv9_14[2]+col9_14[index1][3]*h2bv9_14[3]
      +col9_14[index1][4]*h2bv9_14[4]+col9_14[index1][5]*h2bv9_14[5]
      +col9_14[index1][6]*h2bv9_14[6]);

      FirstD(fname,xk3,y_jpc3,nbcmp1,fnewk3);

      for(index1=1;index1<nbcmp1;index1++)
        col1_8[index1][1]=fnewk3[index1];

     err_estkp1=0.0;err_est = 0.0 ;err_estkm1=0.0;err_estkm2=0.0;

// PART: YNEW CALC. &  STEP SIZE SELECTION

        xtab[40] =(double)ordk ;

    HBO4_14d12_4SC( xtab,scpbv ) ;

        for(index1=1;index1<=8;index1++)
        {
          SCPbv1_8k[index1]=scpbv[index1+28];
          hSCPbv1_8k[index1]=h*SCPbv1_8k[index1];
        }
        for(index1=1;index1<=6;index1++)
        {
          SCPbv9_14k[index1]=scpbv[index1+36];
          h2SCPbv9_14k[index1]=h*h*SCPbv9_14k[index1];
        }

    // y_est = yk1 + col1_8*hSCPbv1_8k + col9_14*h2SCPbv9_14k  ;
    for(index1=1;index1<nbcmp1;index1++)

y_est[index1]=yk1[index1]+(col1_8[index1][1]*hSCPbv1_8k[1]+
col1_8[index1][2]*hSCPbv1_8k[2]+col1_8[index1][3]*hSCPbv1_8k[3]
+col1_8[index1][4]*hSCPbv1_8k[4]+col1_8[index1][5]*hSCPbv1_8k[5]
+col1_8[index1][6]*hSCPbv1_8k[6]+col1_8[index1][7]*hSCPbv1_8k[7]
+col1_8[index1][8]*hSCPbv1_8k[8])+(col9_14[index1][1]*h2SCPbv9_14k[1]
+col9_14[index1][2]*h2SCPbv9_14k[2]+col9_14[index1][3]*h2SCPbv9_14k[3]
+col9_14[index1][4]*h2SCPbv9_14k[4]+col9_14[index1][5]*h2SCPbv9_14k[5]
             +col9_14[index1][6]*h2SCPbv9_14k[6]);

  // ynew = yk1+ col1_8*hQbv1_8 + col9_14*h2Qbv9_14  ;
```

```
   for(index1=1;index1<nbcmp1;index1++)

ynew[index1]=yk1[index1]+(col1_8[index1][1]*hQbv1_8[1]
             +col1_8[index1][2]*hQbv1_8[2]+col1_8[index1][3]*hQbv1_8[3]
             +col1_8[index1][4]*hQbv1_8[4]+col1_8[index1][5]*hQbv1_8[5]
             +col1_8[index1][6]*hQbv1_8[6]+col1_8[index1][7]*hQbv1_8[7]
             +col1_8[index1][8]*hQbv1_8[8])+(col9_14[index1][1]*h2Qbv9_14[1]

+col9_14[index1][2]*h2Qbv9_14[2]+col9_14[index1][3]*h2Qbv9_14[3]

+col9_14[index1][4]*h2Qbv9_14[4]+col9_14[index1][5]*h2Qbv9_14[5]
             +col9_14[index1][6]*h2Qbv9_14[6]);

// PART:  STEP SIZE, ORDER SELECTION
err_est=fabs(y_est[1]-ynew[1]); for(index1=2;index1<nbcmp1;index1++)
{
    temp=fabs(y_est[index1]-ynew[index1]);
    if( err_est < temp  )
    err_est=temp;
}

hold=h;

if (err_est >  TOL) {

      h =  max(  hold*0.7*stfac *pow( (TOL/err_est),(1.0/((double)ordk-1.0))
), hmin) ;

      nreje=nreje+1;
}

}
// end  LOOP REPEAT UNTIL SUCCESS    while ( err_est > TOL )

         tnew = tc + h;
         FirstD(fname,tnew,ynew,nbcmp1,fnew);
         SecondD(fname,tnew,ynew,nbcmp1,f2new);

         for(index1=1;index1<nbcmp1;index1++)
          col1_8[index1][1]=fnew[index1];

err_estkp1=0.0; err_est = 0.0 ; err_estkm1=0.0; err_estkm2=0.0;

for(index1=1;index1<=8;index1++) {
    SCPbv1_8km2[index1]=scpbv[index1];
hSCPbv1_8km2[index1]=h*SCPbv1_8km2[index1];
```

```
    SCPbv1_8km1[index1]=scpbv[index1+14]; hSCPbv1_8km1[index1]  =h*
SCPbv1_8km1[index1];
    SCPbv1_8k[index1]=scpbv[index1+28]; hSCPbv1_8k[index1]=h*
SCPbv1_8k[index1]  ;
    SCPbv1_8kp1[index1]=scpbv[index1+42]; hSCPbv1_8kp1[index1]=h*
SCPbv1_8kp1[index1]  ;
}

for(index1=1;index1<=6;index1++) {
    SCPbv9_14km2[index1]=scpbv[index1+8];
h2SCPbv9_14km2[index1]=h*h*SCPbv9_14km2[index1];
    SCPbv9_14km1[index1]=scpbv[index1+22];
h2SCPbv9_14km1[index1]=h*h*SCPbv9_14km1[index1];
    SCPbv9_14k[index1]=scpbv[index1+36];
h2SCPbv9_14k[index1]=h*h*SCPbv9_14k[index1] ;
    SCPbv9_14kp1[index1]=scpbv[index1+50];
h2SCPbv9_14kp1[index1]=h*h*SCPbv9_14kp1[index1] ; }

// y_estkm2 = yk1 + col1_8*hSCPbv1_8km2 + col9_14*h2SCPbv9_14km2  ;
for(index1=1;index1<nbcmp1;index1++)
y_estkm2[index1]=yk1[index1]+(col1_8[index1][1]*hSCPbv1_8km2[1]
                +col1_8[index1][2]*hSCPbv1_8km2[2]
                +col1_8[index1][3]*hSCPbv1_8km2[3]
                +col1_8[index1][4]*hSCPbv1_8km2[4]
                +col1_8[index1][5]*hSCPbv1_8km2[5]
                +col1_8[index1][6]*hSCPbv1_8km2[6]
                +col1_8[index1][7]*hSCPbv1_8km2[7]
                +col1_8[index1][8]*hSCPbv1_8km2[8])
                +(col9_14[index1][1]*h2SCPbv9_14km2[1]
                +col9_14[index1][2]*h2SCPbv9_14km2[2]
                +col9_14[index1][3]*h2SCPbv9_14km2[3]
                +col9_14[index1][4]*h2SCPbv9_14km2[4]
                +col9_14[index1][5]*h2SCPbv9_14km2[5]
                +col9_14[index1][6]*h2SCPbv9_14km2[6]);

// y_estkm1 = yk1 + col1_8*hSCPbv1_8km1 + col9_14*h2SCPbv9_14km1  ;
for(index1=1;index1<nbcmp1;index1++)
y_estkm1[index1]=yk1[index1]+(col1_8[index1][1]*hSCPbv1_8km1[1]
                +col1_8[index1][2]*hSCPbv1_8km1[2]
                +col1_8[index1][3]*hSCPbv1_8km1[3]
                +col1_8[index1][4]*hSCPbv1_8km1[4]
                +col1_8[index1][5]*hSCPbv1_8km1[5]
                +col1_8[index1][6]*hSCPbv1_8km1[6]
                +col1_8[index1][7]*hSCPbv1_8km1[7]
                +col1_8[index1][8]*hSCPbv1_8km1[8])
                +(col9_14[index1][1]*h2SCPbv9_14km1[1]
```

```
                +col9_14[index1][2]*h2SCPbv9_14km1[2]
                +col9_14[index1][3]*h2SCPbv9_14km1[3]
                +col9_14[index1][4]*h2SCPbv9_14km1[4]
                +col9_14[index1][5]*h2SCPbv9_14km1[5]
                +col9_14[index1][6]*h2SCPbv9_14km1[6]);

// y_est = yk1 + col1_8*hSCPbv1_8k + col9_14*h2SCPbv9_14k  ;
for(index1=1;index1<nbcmp1;index1++)
y_est[index1]=yk1[index1]+(col1_8[index1][1]*hSCPbv1_8k[1]
+col1_8[index1][2]*hSCPbv1_8k[2]+col1_8[index1][3]*hSCPbv1_8k[3]
+col1_8[index1][4]*hSCPbv1_8k[4]+col1_8[index1][5]*hSCPbv1_8k[5]
+col1_8[index1][6]*hSCPbv1_8k[6]+col1_8[index1][7]*hSCPbv1_8k[7]
              +col1_8[index1][8]*hSCPbv1_8k[8])
              +(col9_14[index1][1]*h2SCPbv9_14k[1]
              +col9_14[index1][2]*h2SCPbv9_14k[2]
              +col9_14[index1][3]*h2SCPbv9_14k[3]
              +col9_14[index1][4]*h2SCPbv9_14k[4]
              +col9_14[index1][5]*h2SCPbv9_14k[5]
              +col9_14[index1][6]*h2SCPbv9_14k[6]);

//y_estkp1 = yk1 + col1_8*hSCPbv1_8kp1 + col9_14*h2SCPbv9_14kp1  ;
for(index1=1;index1<nbcmp1;index1++)
y_estkp1[index1]=yk1[index1]+(col1_8[index1][1]*hSCPbv1_8kp1[1]
                +col1_8[index1][2]*hSCPbv1_8kp1[2]
                +col1_8[index1][3]*hSCPbv1_8kp1[3]
                +col1_8[index1][4]*hSCPbv1_8kp1[4]
                +col1_8[index1][5]*hSCPbv1_8kp1[5]
                +col1_8[index1][6]*hSCPbv1_8kp1[6]
                +col1_8[index1][7]*hSCPbv1_8kp1[7]
                +col1_8[index1][8]*hSCPbv1_8kp1[8])
                +(col9_14[index1][1]*h2SCPbv9_14kp1[1]
                +col9_14[index1][2]*h2SCPbv9_14kp1[2]
                +col9_14[index1][3]*h2SCPbv9_14kp1[3]
                +col9_14[index1][4]*h2SCPbv9_14kp1[4]
                +col9_14[index1][5]*h2SCPbv9_14kp1[5]
                +col9_14[index1][6]*h2SCPbv9_14kp1[6]);

    temp=0.0;
    err_estkp1=fabs(y_estkp1[1]-ynew[1]);
    err_est=fabs(y_est[1]-ynew[1]);
    err_estkm1=fabs(y_estkm1[1]-ynew[1]);
    err_estkm2=fabs(y_estkm2[1]-ynew[1]);

    for(index1=2;index1<nbcmp1;index1++)
    {
        temp=fabs(y_estkp1[index1]-ynew[index1]);
```

```
        if( err_estkp1 < temp  )
        err_estkp1=temp;

        temp=fabs(y_est[index1]-ynew[index1]);
        if( err_est < temp  )
        err_est=temp;

        temp=fabs(y_estkm1[index1]-ynew[index1]);
        if( err_estkm1 < temp  )
        err_estkm1=temp;

        temp=fabs(y_estkm2[index1]-ynew[index1]);
        if( err_estkm2 < temp  )
        err_estkm2=temp;
    }

ordnew=ordk;

erk=err_est ;

//lower order
if( (ordk > 4)&&(err_estkm1 <= min(err_est, err_estkp1))||(err_est
>= max(err_estkm1, err_estkm2)) ) {   ordnew=ordk-1;
    erk=err_estkm1 ;
}
// raise order
if( (ordk < 14) && (err_estkp1 < err_est)&&(err_est <
max(err_estkm1, err_estkm2)) ) {   ordnew=ordk+1;
    erk=err_estkp1 ;
}
//  (erk=erk, abm ordr = ordnew-2)
//  (step size formula : h =hold, stsize = hnew)

if (erk != 0.0)

// ACT    (HB../HB11 step cntrl)
      stsize = h * stfac *pow( (TOL/erk ),(1.0/((double)ordnew-1.0 )) ) ;
   else
      stsize = 2*h ;

//  (end of step size formula : h =hold, stsize = hnew)

//  constraints that stsize must satisfy :
//     (a) h( next step) = max (h( next step), hmin, 0.5* h(last step))
//     (b) h( next step) = min (h( next step), hmax, 4 * h(last step))
```

```
   stsize = max ( stsize, 0.5* h ) ;
   stsize = max ( stsize, hmin ) ;
   stsize = min( stsize, hmax) ;
   stsize = min( stsize, 4*h) ;

tc=tnew; for(index1=1;index1<nbcmp1;index1++)
yc[index1]=ynew[index1] ;

jpnbposp1=j+nbpos+1;

for(index1=1;index1<nbcmp1;index1++)
  fvals[index1][jpnbposp1]=fnew[index1];

for(index1=1;index1<nbcmp1;index1++)
  f2vals[index1][jpnbposp1]=f2new[index1];

hnew=stsize ;

//  (END OF INTEG STEP)
//  (yvals,tvals,.. UPDATE STEP)

   ordk=ordnew ;

   tvals[jpnbposp1] =  tc;

   for(index1=1;index1<nbcmp1;index1++)
     yvals[index1][jpnbposp1] =yc[index1];

   prevh = h ;

   j = j+1 ;
}
// ( END OF MAIN LOOP  1 integration step/ iteration)

// print out statistics
// with index j-1,  yvals(n+1+nbpos, . )  cont solution
int n=j-1;

int NFE = nbinitst*4 +(n+1+nreje-nbinitst)*nbeps;

res[1]=(double)NFE;
// nb succes step:
res[2]=(double)n+1 ; res[3]=(double)nreje ;

int HBO14prog_nsuccst=n+1; int HBO14prog_nreje=nreje;
```

```
delete temp1; delete temp2; delete yk1; delete y_jpc2; delete
y_jpc3; delete  ynew; delete  y_estkm2; delete  y_estkm1; delete
y_estkp1; delete  scpbv; delete  b_v; delete  parout; delete  yc;
for(index1=0;index1<nbcmp1;index1++) {
    delete col1_8[index1];
    delete col9_14[index1];
} delete  col1_8; delete  col9_14; delete  xtab; delete  fnewk2;
delete  fnewk3; delete  fnew; delete  f2new; delete Qbv1_8; delete
Qbv9_14; delete hQbv1_8; delete h2Qbv9_14; delete bv1_8; delete
bv9_14; delete hbv1_8; delete h2bv9_14; delete SCPbv1_8k; delete
SCPbv9_14k; delete hSCPbv1_8k; delete h2SCPbv9_14k; delete
SCPbv1_8km2; delete SCPbv9_14km2; delete SCPbv1_8km1; delete
SCPbv9_14km1; delete SCPbv1_8kp1; delete SCPbv9_14kp1; delete
hSCPbv1_8km2; delete h2SCPbv9_14km2; delete hSCPbv1_8km1; delete
h2SCPbv9_14km1; delete hSCPbv1_8kp1; delete h2SCPbv9_14kp1;

delete RESULT;

return ; }
```

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), 27–35.

[2] R. Ashino, M. Nagase, and R. Vaillancourt, *Behind and beyond the* MATLAB *ODE Suite*, Comput. Math. Applic., **40** (2000), 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), 93–111.

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

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

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

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

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

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

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

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

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

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

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

[15] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

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

[17] E. Hairer, S. P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problems*, Springer-Verlag, Berlin, 1993.

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

[19] T. Y. Huang and K. Innanen, *A survey of multiderivative multistep integrators*, Astronomical J., **112**(3) (1996), 1254–1262.

[20] 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), 603–637.

[21] 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.

[22] 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.

[23] 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)

[24] J. D. Lambert, *Computational Methods in Ordinary Differential Equations*, Ch. 5, Wiley, London, 1973.

[25] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems*, Wiley, Chichester UK, 1991.

[26] W. E. Milne, *A note on the numerical integration of differential equations*, J. Res. Nat. Bur. Standards, **43** (1949), 537–542.

[27] 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), 47–64.

[28] 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), 151–181.

[29] 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) 43–69.

[30] 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) 415–439.

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

[32] N. Obrechkoff, *Neue Quadraturformeln*, Abh. Preuss. Akad. Wiss. Math. Nat. Kl., No. 4, (1940), 1–20.

[33] E. Rabe, *Determination and survey of periodic Trojan orbits in the restricted problem of three bodies*, Astronomical J., **66**(9) (November 1961) 500–513.

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

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

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

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