Elsevier Editorial System(tm) for Journal of Computational and Applied Mathematics
Manuscript Draft

Manuscript Number: CAM-D-17-01905

Title: Yet another code for Boundary Value Problems- Higher Derivative Method

Article Type: Research Paper

Section/Category: 65Lxx (Ordinary differential equations)

Keywords: Boundary Value Problems; Ordinary Differential Equations; Symbolic Computation; Higher Derivative Method

Corresponding Author: Dr. Venkat R Subramanian,

Corresponding Author's Institution: University of Washington

First Author: Jerry Chen

Order of Authors: Jerry Chen; Dayaram Sonawane; Kishalay Mitra; Venkat R Subramanian

Abstract: While there are many good existing solvers, in this paper, a discretization scheme using higher derivative method (HDM) with A-stability property is implemented for efficient solution of boundary value problems (BVPs) described by Ordinary Differential Equations (ODEs). The proposed approach uses nth derivative to get 2n order of accuracy at each node point in BVPs. The derivation of HDM schemes can be related to the stability function of the Implicit Runge-Kutta methods (IRK) and Padé approximation, and the coefficients of the HDM schemes can be found by Hermite collocation methods. Once the analytical derivatives are found, HDM scheme is applied for every node. Illustrative examples are included to demonstrate the applicability of HDM for the solution of BVPs. The algorithm is implemented and shared as an open access Maple® code.

Yet another code for Boundary Value Problems- Higher Derivative Method

Jerry Chen[a], Dayaram Sonawane[b], Kishalay Mitra[c], and Venkat R. Subramanian[a, d]

[a] Department of Chemical Engineering, University of Washington, Seattle, Washington 98195, USA

[b] Department of Instrumentation and Control, College of Engineering Pune, Pune, Maharashtra 411005, India

[c] Department of Chemical Engineering, Indian Institute of Technology Hyderabad, Kandi, Sangareddy, Telangana 502285, India

[d] Pacific Northwest National Laboratory, Richland, Washington 99354, USA

Abstract

While there are many good existing solvers, in this paper, a discretization scheme using higher derivative method (HDM) with *A*-stability property is implemented for efficient solution of boundary value problems (BVPs) described by Ordinary Differential Equations (ODEs). The proposed approach uses n^{th} derivative to get $2n$ order of accuracy at each node point in BVPs. The derivation of HDM schemes can be related to the stability function of the Implicit Runge-Kutta methods (IRK) and Padé approximation, and the coefficients of the HDM schemes can be found by Hermite collocation methods. Once the analytical derivatives are found, HDM scheme is applied for every node. Illustrative examples are included to demonstrate the applicability of HDM for the solution of BVPs. The algorithm is implemented and shared as an open access `Maple`® code.

Keywords: *Boundary Value Problems; Ordinary Differential Equations; Symbolic Computation; Higher Derivative Method*

I. Introduction

Boundary value problems (BVPs) appear in many domains including electrical circuits, control, economics, fluid flow, heat/mass transfer, electrochemical engineering, etc. The numerical methods for solution of BVPs are based on finite difference methods, or (multiple) shooting methods [1]. The finite difference methods are typically coupled with Implicit Runge-Kutta (IRK) methods or collocation methods within a particular mesh for higher order accuracy [2–5]. Many of the implementations use deferred correction or extrapolation methods to improve accuracy and

efficiency [6]. The semi-implicit and fully implicit RK methods proved to be the useful for the solutions of BVPs, but any IRK method involves the solution of a non-linear system of $n \times s$ equations; where n is the number of differential equations and s is the number of stages of IRK methods. Thus, it requires more computational effort per step including memory and CPU time when the number of discretization variables is increased especially for Newton's method for solving the resulting system of nonlinear algebraic equations. Since it has a maximum of s-order of accuracy at the internal collocation points, it is often times not accurate enough for control variables while solving optimal control problems. Without having the need for solving variables in the internal stages, mono-implicit forms have been used as well [7]. Mono implicit forms include variables only at mesh points.

Stability of the numerical discretization scheme is an important issue for solving ordinary differential equations (ODEs) in IVPs or BVPs. While developing robust discretization schemes, the aim is to ensure stability that can provide converged solutions while solving stiff set of differential equations characterized by rapidly decaying transients. Therefore, numerical methods that are "*A-stable*" are usually preferred. *A*-stable methods generally allow the use of larger step sizes because they guarantee that the rapidly decaying terms will continue to decrease for any step size used [8]. Implementations of *A*-stable single-step methods using higher derivatives are available in the literature. Gad et al. [9] proposed a generalized class of *A*-stable integration formulas using higher derivatives for IVPs, and this concept was first presented by Obrechkoff [10]. On the other hand, Gupta [4] utilized Boundary Value Runge-Kutta (BVRK) up to 8^{th} order while finding the solution of a nonlinear system of first order BVPs considering RK internal stages within a single step. *A*-stable property in IVPs also refers to the stability in BVPs [5,11,12].

In many of the popular codes, the numerical solution of a nonlinear BVP is generally accomplished by adopting linearization schemes leading to the fact that a BVP can have more than one solution [13]. Typically, n^{th} order BVPs are converted into n first order BVPs to apply IRK type of methods. Unlike IVPs, where a unique solution can be guaranteed with initial conditions of differential variables, BVPs sometimes have the situation that either no solution or multiple solutions exist even for the simple set of differential equations. This aspect poses challenges to develop efficient numerical methods that guarantee a unique/meaningful solution with high precision for the BVPs. Popular methods available today to solve first order BVPs include Lobatto-IIIA IRK algorithm used in `bvp4c` [14], Gauss-Legendre collocation used in `COLSYS` [15,16]

and COLNEW [17,18], and Mono-implicit Runge-Kutta (MIRK) methods with deferred correction used in TWPBVP developed by Cash [7,19] or in MIRKDC developed by Enright and Muir [20]. To obtain convergence, a good initial guess may be necessary. The initial guess involves two parts: (1) a proper initial mesh that reveals the behavior of the solution (2) the guess solutions at the mesh points. After obtaining convergence according to the initial mesh and the guess value, the codes adjust the mesh and obtain accurate numerical solutions with a modest number of mesh points.

While there are very good solvers for BVPs, a natural question is why do we need another BVP solver? It is indeed possible to attain higher order of accuracy without losing stability by using higher derivatives [21]. These methods have not been converted to production codes because of the need to find higher derivatives, and the stability is not guaranteed if numerical approximation is used for finding higher derivatives. With recent improvement in computer algebra system and symbolic mathematical software like Maple® [22], one can find the analytical form of derivatives enabling development of an efficient code for solving BVPs using higher order derivatives.

In this paper, the use of higher derivative method (HDM) with the property of *A*-stability for the solution of BVPs is presented. The use of n^{th} derivative leads us to formulate a $2n$ order HDM scheme. The HDM seems to require more work for computation of differentiation, but has the advantage that it needs lesser number of discretized variables with higher orders of accuracy ($2n$, where n is the use of the order of derivative) with *A*-stability. The HDM schemes only involve single-step, so the system is compact (the integration from one node point to another involves only variables at the two node points), and can be considered as a MIRK type of method (implicit in only one node point when the value is known at the previous node). The continuous solution (interpolation polynomial) accurate to the order $2n$ can be derived at any point in the spatial domain when constant step sizes are used. The code developed in this paper can solve many stiff and non-stiff BVPs without having the need to provide an initial guess in very few seconds to minutes. To the best of our knowledge, the application of higher derivative discretization scheme for the solution of BVPs has not been reported in the literature. Numerical tests on several test problems show the advantages of the proposed approach.

II. Higher Derivative Method

A BVP problem can be considered as a system of the first-order ODEs represented in the form of Eq. (1).

$$\frac{dy}{dx} = f(x,y) = \lambda y,\ x \in [a,b],\ y \in \mathrm{R}^n \tag{1}$$

With boundary conditions as $y(a) = \alpha$, $y(b) = \beta$ .The total number of boundary conditions matches the number of first order ODEs.

The concept of HDM scheme originates from the stability function of *A*-stable IRK methods, for example, Lobatto IIIA methods and Gauss–Legendre methods.

The (implicit) trapezoidal rule (second-order Lobatto IIIA method) is a second-order, single step, 2-stages method as given below.

$$y_{n+1} = y_n + \frac{h}{2}\left(f(y_n) + f(y_{n+1}) \right) \tag{2}$$

All Lobatto IIIA methods are collocation methods providing 2*s*-2 order of convergence at the terminal nodes, where *s* is the number of internal stages.

Similarly, the implicit midpoint method (single step, 1-stage) is a second-order Gauss–Legendre method as given below.

$$y_{n+1} = y_n + hf\left(\frac{y_n + y_{n+1}}{2} \right) \tag{3}$$

All Gauss–Legendre methods use the points of Gauss–Legendre quadrature as collocation points. If the method uses *s* stages, it has 2*s* order.

Both trapezoidal rule and implicit midpoint method share the same stability function $R(z)$ as given below.

$$R(z) = \frac{1 + 1/2\, z}{1 - 1/2\, z} \tag{4}$$

For IRK methods, $R(z)$ is a rational function with numerator and denominator of degree $\leq s$. Both of the methods are *A*-stable (which means if and only if $|R(z)| \leq 1$ whenever *z* is in the left half-plane, $\mathrm{Re}(z) \leq 0$.), but not *L*-stable (i.e. if it is A-stable and $\lim_{\mathrm{Re}(z) \to -\infty} R(z) = 0$).

It is observed that the stability function is the same as the diagonal Padé approximation to the exponential function ($\boldsymbol{P}_{n,n}(z)$) [23]. For example, the stability function for trapezoidal rule and implicit midpoint as Eq.(4) can be represented as the diagonal Padé approximant, $\boldsymbol{P}_{1,1}(z)$.

Birkhoff and Varga [24] pointed out that $\boldsymbol{P}_{n,n}(z)$ has the property that $\left|\boldsymbol{P}_{n,n}(z)\right| \le 1$ for $R(z) \le 0$, where n = 1, 2,…, which matches the IRK stability function. Ehle [25] showed that an *A*-stable single-step method can be written as $y_{n+1} = \boldsymbol{P}_{i,i}(\lambda h) y_n$. Therefore, Eq. (4) can also be represented as

$$y_{n+1} = \frac{1 + 1/2(\lambda h)}{1 - 1/2(\lambda h)} y_n \tag{5}$$

After applying $y_n' = f(y_n) = \lambda y_n$ and $y_{n+1}' = f(y_{n+1}) = \lambda y_{n+1}$, we can get Eq. (2), again. This is the second-order (uses first derivative) HDM scheme, and has the smallest truncation error among all the second order linear multistep methods with *A*-stability property [26].

In addition, it is also possible to derive HDM using Hermite polynomials or collocation. For example, if we assume *y* is a continuous function and taking *y* to be a second-order polynomial as

$$y = y_0 + ax + bx^2 \tag{6}$$

One can obtain the two constants by forcing the first derivative at $x = 0$ and $x = h$.

when $x = 0$

$$f(y_0) = a \tag{7}$$

when $x = h$

$$f(y_h) = a + 2bh \tag{8}$$

Therefore, by solving Eqs.(7)-(8) in terms of *a* and *b*, we can get the coefficients as

$$a = f(y_0) \tag{9}$$

$$b = \frac{f(y_h) - f(y_0)}{2h} \tag{10}$$

After substituting the coefficients into Eq. (6) and generalizing the formula from $x = x_n$ to $x = x_{n+1}$, one can get the second-order (the first derivative) HDM scheme as in Eq. (2). The coefficients directly correspond to the diagonal Padé approximation (having the same stability functions as Lobatto IIIA and Gauss–Legendre types of IRK methods). This approach can be used to derive higher order HDM schemes.

The second derivative method (SDM) or the 4th order HDM scheme can be derived from the stability function of the fourth-order Lobatto IIIA method with stages 0, 1/2 and 1 (single step, 3 stages) as

$$y_{int} = y_n + h\left(\frac{5}{24} f\left(y_n\right) + \frac{1}{3} f\left(y_{int}\right) - \frac{1}{24} f\left(y_{n+1}\right)\right) \tag{11}$$

$$y_{n+1} = y_n + h\left(\frac{1}{6} f\left(y_n\right) + \frac{2}{3} f\left(y_{int}\right) + \frac{1}{6} f\left(y_{n+1}\right)\right) \tag{12}$$

They require solving for both internal variables and terminal values simultaneously in each step, so the computational efforts increase as the number of internal stages increases. It should be noted that Lobatto IIIA scheme with 3-stages can be written in the mono-implicit form, but it is difficult to write the scheme in the mono-implicit form if the number of stages is greater than 3.

The fourth-order Gauss–Legendre method with stages $\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)$, $\left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)$ (single step, 2 stages) is given as

$$y_{int,1} = y_n + h\left(\frac{1}{4} f\left(y_{int,1}\right) + \left(\frac{1}{4} - \frac{\sqrt{3}}{6}\right) f\left(y_{int,2}\right)\right) \tag{13}$$

$$y_{int,2} = y_n + h\left(\left(\frac{1}{4} + \frac{\sqrt{3}}{6}\right) f\left(y_{int,1}\right) + \frac{1}{4} f\left(y_{int,2}\right)\right) \tag{14}$$

$$y_{n+1} = y_n + h\left(\frac{1}{2} f\left(y_{int,1}\right) + \frac{1}{2} f\left(y_{int,2}\right)\right) \tag{15}$$

Both, the fourth-order Lobatto IIIA method and the fourth-order Gauss–Legendre method share the same stability function given by

$$R(z) = \frac{1 + 1/2\, z + 1/12\, z^2}{1 - 1/2\, z + 1/12\, z^2} \tag{16}$$

For the second derivative method (SDM) as given in Eq.(17), one can either derive from Taylor series [21], Butcher's *n* stage IRK process of order 2*n* [25], Hermite interpolation formula [27] or the method of undetermined coefficients [28]. This method was first considered by Obrechkoff [10]. This formula is *A*-stable but not *L*-stable [29].

$$y_{n+1} = y_n + \frac{h}{2}\left(f\left(y_n\right) + f\left(y_{n+1}\right)\right) + \frac{h^2}{12}\left(f'\left(y_n\right) - f'\left(y_{n+1}\right)\right) \tag{17}$$

Similarly, collocation can be used to derive Eq.(17) starting from a fourth-order polynomial as Eq. (18).

$$y = y_0 + ax + bx^2 + cx^3 + dx^4 \tag{18}$$

One can obtain the four constants by forcing the first and second derivatives at $x = 0$ and $x = h$.

when $x = 0$

$$f\left(y_0\right) = a \text{ and } f'\left(y_0\right) = 2b \tag{19}$$

when $x = h$

$$f\left(y_h\right) = a + 2bh + 3ch^2 + 4dh^3 \text{ and } f'\left(y_h\right) = 2b + 6ch + 12dh^2 \tag{20}$$

Solving Eqs. (19)-(20) for *a*, *b*, *c*, and *d*, we can get

$$a = f\left(y_0\right) \tag{21}$$

$$b = \frac{f'\left(y_0\right)}{2} \tag{22}$$

$$c = \frac{-3f\left(y_0\right) + 3f\left(y_h\right) - 2hf'\left(y_0\right) - hf'\left(y_h\right)}{3h^2} \tag{23}$$

$$d = \frac{2f\left(y_0\right) - 2f\left(y_h\right) + hf'\left(y_0\right) + hf'\left(y_h\right)}{4h^3} \tag{24}$$

Eq. (17) is obtained by substituting Eqs. (21)-(24) into Eq. (18) and by generalizing the method for any node x_n to x_{n+1}..

One can realize that the HDM, Lobatto IIIA and Gauss–Legendre types of IRK methods share the same A-stability function/region and the form and coefficients are the same as the diagonal Padé approximation to the exponential function.

We can explore more of HDM's properties by looking into the *s*-stage IRK methods, which can be represented by Eq. (25) for the stages and Eq. (26) for the terminal respectively.

$$y_{n,i} = y_n + h\sum_{j=1}^{s} a_{ij} f\left(x_n + c_j h, y_{n,j}\right) \quad i = 1,...,s \tag{25}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{s} b_j f\left(x_n + c_j h, y_{n,j}\right) \tag{26}$$

Eq. (26) is well defined if all the stages have unique solutions for all λ in Eq. (1) with $\mathrm{Re}\left(\lambda\right) < 0$ [30]. When applied to Eq. (1), it yields $y_{n+1} = R\left(\lambda h\right) y_n$ with the stability function as

Eq. (27), and it can be summarized in a rational form, with numerator and denominator with degree $\leq s$.

$$R(z) = 1 + zb^T (I - zA)^{-1} \mathbf{1} = \frac{\det(I - zA + z\mathbf{1}b^T)}{\det(I - zA)} \tag{27}$$

where $b^T = (b_1, ..., b_s)$, $A = (a_{ij})_{i,j=1}^{s}$, $\mathbf{1} = (1, ..., 1)^T$

The IRK method is *A*-stable, if $|R(z)| \leq 1$ for all complex *z* with negative real part ($\mathrm{Re}(z) \leq 0$) [26]. It is *L*-stable if it is *A*-stable and $\lim_{\mathrm{Re}(z)\to-\infty} R(z) = 0$.

A-stability is well defined if the numerator and denominator of $R(z)$ in Eq. (27) have no common zeros in the left half of *z* plane. In particular, any *A*-stable IRK method represented as a Padé approximation to the exponential function is well defined [30]. So, one can write the correlation in a more explicit way as Eq. (28).

$$y_{n+1} = R(z)\, y_n = \boldsymbol{P}_{m,n}(z)\, y_n = \frac{\sum_{i=0}^{m} \frac{(m+n-i)!\,m!}{(m+n)!\,i!\,(m-i)!} z^i}{\sum_{i=0}^{n} \frac{(m+n-i)!\,n!}{(m+n)!\,i!\,(n-i)!} (-z)^i} y_n, \text{ and } z = \lambda h \tag{28}$$

Since the general solution of linearized first-order ODEs can be expressed as the exponential function, it makes sense to consider the Padé approximation (unique and rational) to $exp(z)$ of the order $p = m + n$, which has a property of $\boldsymbol{P}_{m,n}(z) - e^z = O(z^{p+1})$, $p \geq 0$ for *A*-stability.

Based on more general understanding of the stability function and its corresponding Padé approximation, one can not only take advantage of the diagonal Padé approximation with *A*-stability which is proved by Birkhoff and Varga [24] by using continued fraction expansions, but also use the first and the second sub-diagonal Padé approximations that have L-stability property as proved by Ehle [8,31].

The IRK methods which satisfy these conditions include those of order $2n$ which correspond to diagonal members of the Padé table [2]; the methods of order $2n-1$ which correspond to the first sub-diagonal Padé table [32]; the methods of order $2n-2$ which correspond to the second sub-diagonal Padé table [30]. The corresponding stability functions of IRK methods and their Padé approximation are listed in Table 1, where *s* is the number of stages.

Table 1. The stability functions of *A*-stable IRK methods and their Padé approximation

IRK Methods	Order for IRK	Stability function can express as...	Order for Padé	Stability
Gauss–Legendre	$2s$	Diagonal Padé $\boldsymbol{P}_{n,n}(z)$	$2n$	*A*-stable
Lobatto IIIA	$2s-2$			
Radau IIA	$2s-1$	The first sub-diagonal Padé $\boldsymbol{P}_{n-1,n}(z)$	$2n-1$	*L*-stable
Lobatto IIIC	$2s-2$	The second sub-diagonal Padé $\boldsymbol{P}_{n-2,n}(z)$	$2n-2$	*L*-stable

One can derive the HDM for any order either from the stability function of *A*-stable IRK methods by $y_{n+1}=R(z)y_n=\boldsymbol{P}_{m,n}(z)y_n$ or Hermite collocation methods. Eqs. (29)-(31) list the HDM methods, using the third derivative (6th order), the fourth derivative (8th order) and the fifth derivative (10th order), respectively.

$$y_{n+1}=y_n+\frac{h}{2}\left(f(y_n)+f(y_{n+1})\right)+\frac{h^2}{10}\left(f'(y_n)-f'(y_{n+1})\right)+\frac{h^3}{120}\left(f''(y_n)+f''(y_{n+1})\right) \tag{29}$$

$$\begin{aligned}y_{n+1}=y_n+&\frac{h}{2}\left(f(y_n)+f(y_{n+1})\right)+\frac{3h^2}{28}\left(f'(y_n)-f'(y_{n+1})\right)+\frac{h^3}{84}\left(f''(y_n)+f''(y_{n+1})\right)\\&+\frac{h^4}{1680}\left(f^{(3)}(y_n)-f^{(3)}(y_{n+1})\right)\end{aligned} \tag{30}$$

$$\begin{aligned}y_{n+1}=y_n+&\frac{h}{2}\left(f(y_n)+f(y_{n+1})\right)+\frac{h^2}{9}\left(f'(y_n)-f'(y_{n+1})\right)+\frac{h^3}{84}\left(f''(y_n)+f''(y_{n+1})\right)\\&+\frac{h^4}{1008}\left(f^{(3)}(y_n)-f^{(3)}(y_{n+1})\right)+\frac{h^5}{30240}\left(f^{(4)}(y_n)+f^{(4)}(y_{n+1})\right)\end{aligned} \tag{31}$$

L-stable family of HDM with properties similar to Radau IIA can also be derived by Hermite or polynomial collocation methods.

For example, if we take *y* to be a straight line passing through y_0.

$$y=y_0+ax \tag{32}$$

And equating the derivative at $x=h$

$$f\left(y_{n+1}\right)=a \tag{33}$$

By finding a, we get the Euler Backward method with first order accuracy.

$$y_{n+1}=y_n+hf\left(y_{n+1}\right) \tag{34}$$

Similarly, by considering y as a cubic profile passing through y_0.

$$y=y_0+ax+bx^2+cx^3 \tag{35}$$

And equating the first derivative at $x=0$, and first and second derivatives at $x=h$, we get when $x=0$

$$f\left(y_0\right)=a \tag{36}$$

when $x=h$

$$f\left(y_h\right)=a+2bh+3ch^2 \text{ and } f'\left(y_h\right)=2b+6ch \tag{37}$$

Solving a, b and c, and substituting back into Eq. (35), one can arrive at the third order (second derivative) HDM scheme with L-stability as in

$$y_{n+1}=y_n+h\left(\frac{1}{3}f\left(y_n\right)+\frac{2}{3}f\left(y_{n+1}\right)\right)+h^2\left(-\frac{1}{6}f'\left(y_{n+1}\right)\right) \tag{38}$$

From the first sub-diagonal Padé approximation, one can get the order $2n-1$ with A-stability (L-stability). HDM schemes using the third derivative (5th order), fourth derivative (7th order) and fifth derivative (9th order) are given below.

$$y_{n+1}=y_n+h\left(\frac{2}{5}f\left(y_n\right)+\frac{3}{5}f\left(y_{n+1}\right)\right)+h^2\left(\frac{1}{20}f'\left(y_n\right)-\frac{3}{20}f'\left(y_{n+1}\right)\right)+h^3\left(\frac{1}{60}f''\left(y_{n+1}\right)\right) \tag{39}$$

$$\begin{aligned} y_{n+1}=y_n&+h\left(\frac{3}{7}f\left(y_n\right)+\frac{4}{7}f\left(y_{n+1}\right)\right)+h^2\left(\frac{1}{14}f'\left(y_n\right)-\frac{2}{14}f'\left(y_{n+1}\right)\right)\\ &+h^3\left(\frac{1}{210}f''\left(y_n\right)+\frac{4}{210}f''\left(y_{n+1}\right)\right)+h^4\left(-\frac{1}{840}f^{(3)}\left(y_{n+1}\right)\right) \end{aligned} \tag{40}$$

$$\begin{aligned} y_{n+1}=y_n&+h\left(\frac{4}{9}f\left(y_n\right)+\frac{5}{9}f\left(y_{n+1}\right)\right)+h^2\left(\frac{3}{36}f'\left(y_n\right)-\frac{5}{36}f'\left(y_{n+1}\right)\right)\\ &+h^3\left(\frac{2}{252}f''\left(y_n\right)+\frac{5}{252}f''\left(y_{n+1}\right)\right)+h^4\left(\frac{1}{3024}f^{(3)}\left(y_n\right)-\frac{5}{3024}f^{(3)}\left(y_{n+1}\right)\right)\\ &+h^5\left(\frac{1}{15120}f^{(4)}\left(y_{n+1}\right)\right) \end{aligned} \tag{41}$$

In order to extend to higher orders, a general form is needed. The HDM schemes were mentioned in few papers, but in different integration forms [8,17,25,33], so we summarize the explicit form of HDM schemes in Eq.(42). The values for the coefficients for HDM scheme are listed in Table 2.

$$y_{j+1} = y_j + \sum_{i=1}^{n} h^i \left(\alpha_i y_j^{(i)} + (-1)^{i+1} \beta_i y_{j+1}^{(i)} \right), \; n = 1, 2, 3... \tag{42}$$

Table 2. Coefficients for HDM schemes

Padé approximants	The choice for α_i and β_i	Order	Stability
Diagonal Padé $\boldsymbol{P}_{n,n}(z)$	$\alpha_i = \beta_i = \dfrac{(2n-i)!(n)!}{(2n)!i!(n-i)!},\ i=1,2,...,n$	$2n$	A-stable
The first sub-diagonal Padé $\boldsymbol{P}_{n-1,n}(z)$	$\alpha_i = \dfrac{(2n-1-i)!(n-1)!}{(2n-1)!i!(n-1-i)!},\ i=1,2,...,n-1$ $\alpha_n = 0$ $\beta_i = \dfrac{(2n-1-i)!n!}{(2n-1)!i!(n-i)!},\ i=1,2,...,n$	$2n$-1	L-stable
The second sub-diagonal Padé $\boldsymbol{P}_{n-2,n}(z)$	$\alpha_i = \dfrac{(2n-2-i)!(n-2)!}{(2n-2)!i!(n-2-i)!},\ i=1,2,...,n-2$ $\alpha_n = \alpha_{n-1} = 0$ $\beta_i = \dfrac{(2n-2-i)!n!}{(2n-2)!i!(n-i)!},\ i=1,2,...,n$	$2n$-2	L-stable

III. HDM Code Development for BVPs

A `Maple`® code is developed that discretizes the BVPs based on the HDM scheme. It utilizes n^{th} order derivative to get $2n$ order of accuracy at each node point. The code contains two routines that can be called by the user: (1) Procedure `HDM` (2) Procedure `HDMadapt`. The `HDM` procedure returns the solutions based on a given mesh and order, while `HDMadapt` procedure returns the solutions with automatically adjusted mesh and order. For most of the problems we tested, the solutions can be attained by the default setting of the initial derivative (fifth derivative) and the initial number of elements (10 elements) within few seconds.

Procedure `HDM` takes the given ODEs, boundary conditions (BCs), range, absolute/relative tolerance, initial values of variables at each node, the number of elements (N), the size of the elements (spacing h_i) as the input. One can further provide both known and unknown parameters, for the procedure. The discretization scheme is given in Fig. 1. The `HDM` procedure begins with finding the ODE variables and their analytical forms of derivatives according to the input given

by the user. For instance, the procedure will compute the first to the fifth derivatives if the default value, five, (10^{th} order) is used. The HDM scheme is formed according to Eq. (42) and Table 2. The total number of HDM discretization equations is equal to the number of ODEs times the number of elements used. The total number of discretized variables is number of ODEs times the number of node points (= number of elements + 1). The value x_i is determined by the size of the elements (spacing h_i) as given in Eq. (43).

Once the mesh is established, it can be substituted into the HDM scheme and boundary conditions. This becomes the equation set for the solution of BVPs by the Newton-Raphson method. Based on testing different linear algebra solvers, the sparse solver was found to be the most robust and efficient in `Maple`® and the same is used in this paper to handle the banded nature of the Jacobian generated by the single-step discretization scheme. The compact and mono implicit form of the method means that at any point *i*, there is a need to find the derivatives only with respect to the variables in the nodes *i* and *i* + 1, with boundary conditions handled separately. The code stores only non-zero values, and calculates the Jacobian matrix for ODEs only for the non-zero elements according to the HDM discretization scheme. However, for unknown parameters, all the entries are calculated because the pattern cannot be predetermined.

$$x_i = Range\left(left\right) + \sum_{j}^{i} h_j \tag{43}$$

The solutions at discretization points and the solutions of unknown variables as well as the root-mean-square error at each node point are returned. In the default setting, the 10^{th} order HDM scheme (finding 5^{th} derivative by using $\boldsymbol{P}_{5,5}\left(z\right)$ with *A*-stability), and the 9^{th} order HDM scheme ($\boldsymbol{P}_{4,5}\left(z\right)$ with *L*-stability) are used to find the local error. The error is defined as given in Eq. (44) by using the HDM scheme from Eq. (42) and the coefficients from Table 2, where the coefficients of diagonal Padé are α_1 and α_2, and the same for the first sub-diagonal Padé are β_1 and β_2. The flow diagram of `HDM` procedure is given in Fig. 2.

$$Error = RootMeanSquare\left[\sum_{i=1}^{n} h^i \left(\left(\alpha_{1,i} - \alpha_{2,i}\right) y_j^{(i)} + \left(-1\right)^{i+1} \left(\beta_{1,i} - \beta_{2,i}\right) y_{j+1}^{(i)}\right)\right],\ n = 1, 2, 3... \tag{44}$$

Procedure `HDMadapt` dynamically adjusts the mesh size, and finds the solutions with proper order of accuracy. It does not require the input of the size of elements (spacing *h*) and initial values of variables at each node point. The `HDMadapt` procedure includes the `HDM` procedure and

adapts the mesh automatically based on the local error returned from the `HDM` procedure.

To arrive at the accuracy of the solutions i.e. assuring that the error is smaller than the tolerance, one can increase the node points or the order of HDM scheme. A proper mesh density is required for the resolution of plotting purpose, but if the mesh is too dense, it will result in a large system of algebraic equations to be solved, which is computationally expensive. Therefore, finding a good balance of node points and order is crucial while solving BVPs. On the other hand, for some stiff BVPs (e.g. boundary layers problems), which characterize rapid transients in a narrow region, it is very hard to determine a proper mesh and approximate numerical solutions at the same time. However, since the mesh and the order can be adjusted, the `HDMadapt` procedure can handle most of the tough problems. For most of the problems tested, reasonable results (for tolerance < 1e-6) were achieved without adapting the meshes and with $N \leq 10$ elements.

Since some initial guesses may not provide a converged solution, a "try/catch" kind of logical strategy is used in the `HDM` procedure. If the `HDM` procedure fails due to improper initial conditions, the number of elements (*N*) will be doubled; but only if the number of elements is greater than the maximum elements (assumed 200 by default for efficiency of Newton-Raphson), the derivative will be increased by one (two orders more accurate) till the maximum derivative is reached (assumed to be 9 by default, an arbitrary number).

The mesh will be adjusted by the norm of the local error (because several unknown variables are present at the same node point). Two actions are taken to make the mesh adaptable: insertion and removal of node points. The general mechanism of node point insertion/removal is given in Fig. 3. The adaptive strategy is given in Fig. 4 and `HDMadapt` procedure is explained in Fig. 5.

A second-order two-point BVP can be converted into 2 first-order ODEs with two boundary conditions (BCs).

$$\frac{d^2y}{dx^2} = f(x,y) \qquad BCs: \; y(a)=\alpha,\; y(b)=\beta$$

$$\Rightarrow \quad \frac{dy_1}{dx} = y_2, \quad \frac{dy_2}{dx} = f(x, y_1, y_2) \qquad BCs: \; y_1(a)=\alpha,\; y_1(b)=\beta$$

y_1 [0] y_1 [1] y_1 [2] y_1 [N-2] y_1 [N-1] y_1 [N]

y_2 [0] y_2 [1] y_2 [2] y_2 [N-2] y_2 [N-1] y_2 [N]

2(N+1) unknown variables

N elements with spacing h_1 to h_N

h_1 h_2 …… h_{N-1} h_N

x a $a+h_1$ $a+h_1+h_2$ $a+h_1+\ldots+h_{N-1}$ b

BCs $y_1(a)=\alpha$ $y_1(b)=\beta$ ***2 equations***

For simplicity, the second order HDM method is illustrated here.

$$\begin{cases}
h_1 & y_1[1] = y_1[0] + h_1 f\left(\frac{y_1[0]+y_1[1]}{2}\right) & y_2[1] = y_2[0] + h_1 f\left(\frac{y_2[0]+y_2[1]}{2}\right) \\
h_2 & y_1[2] = y_1[1] + h_2 f\left(\frac{y_1[1]+y_1[2]}{2}\right) & y_2[2] = y_2[1] + h_2 f\left(\frac{y_2[1]+y_2[2]}{2}\right) \\
\vdots & \vdots & \vdots \\
h_{N-1} & y_1[N-1] = y_1[N-2] + h_{N-1} f\left(\frac{y_1[N-2]+y_1[N-1]}{2}\right) & y_2[N-1] = y_2[N-2] + h_{N-1} f\left(\frac{y_2[N-2]+y_2[N-1]}{2}\right) \\
h_N & y_1[N] = y_1[N-1] + h_N f\left(\frac{y_1[N-1]+y_1[N]}{2}\right) & y_2[N] = y_2[N-1] + h_N f\left(\frac{y_2[N-1]+y_2[N]}{2}\right)
\end{cases}$$

2N equations

Fig. 1. Discretization scheme for HDM codes

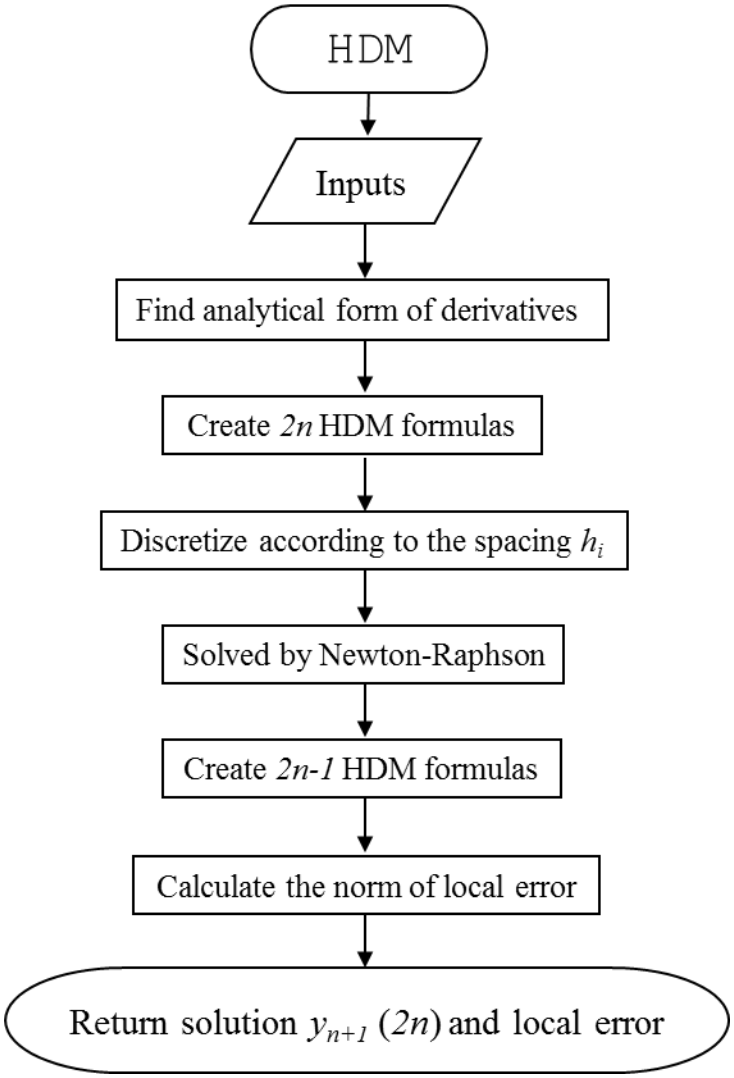


Fig. 2. The flow diagram of HDM procedure

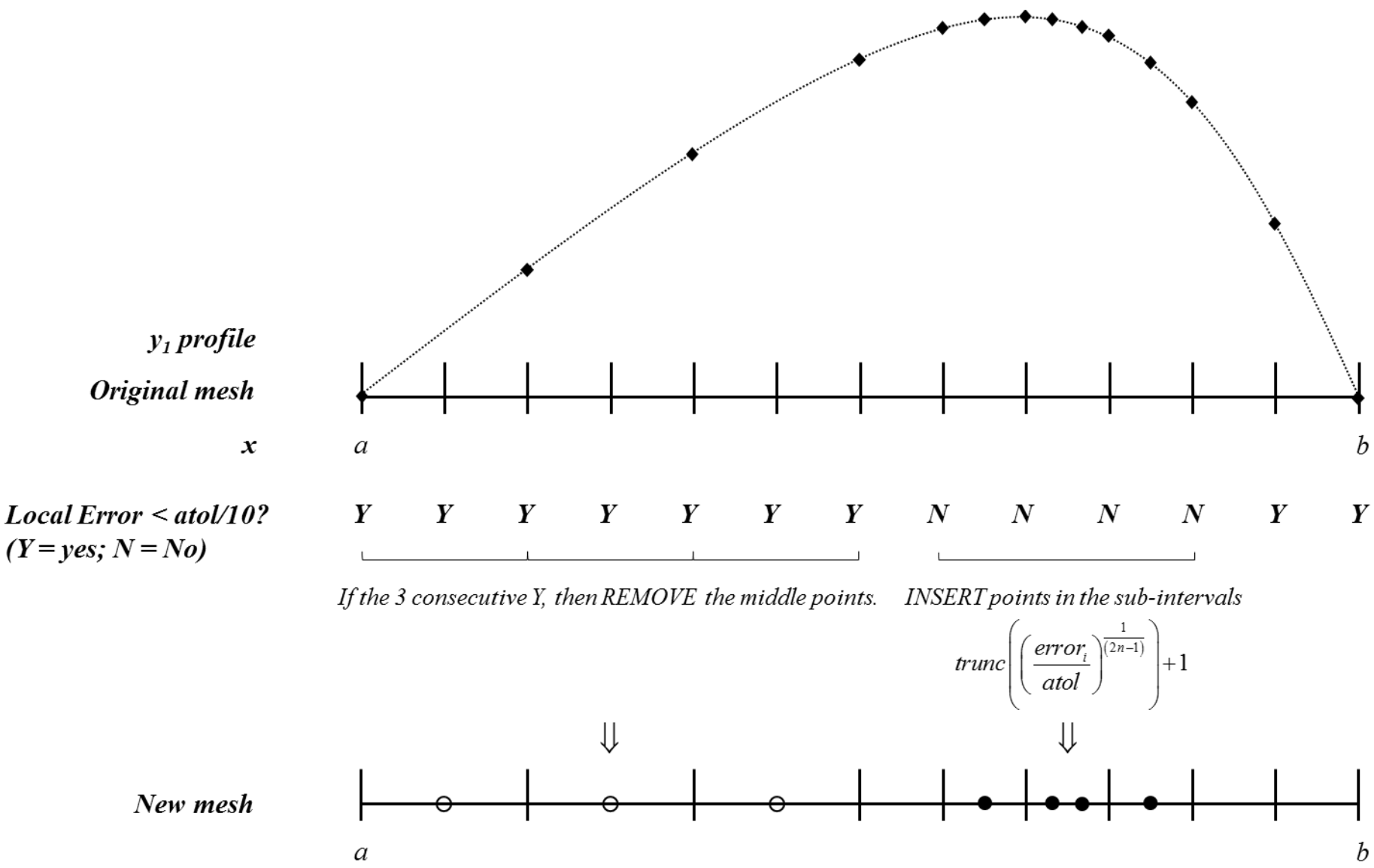


Fig. 3.The adaptive mesh mechanism used by the proposed code

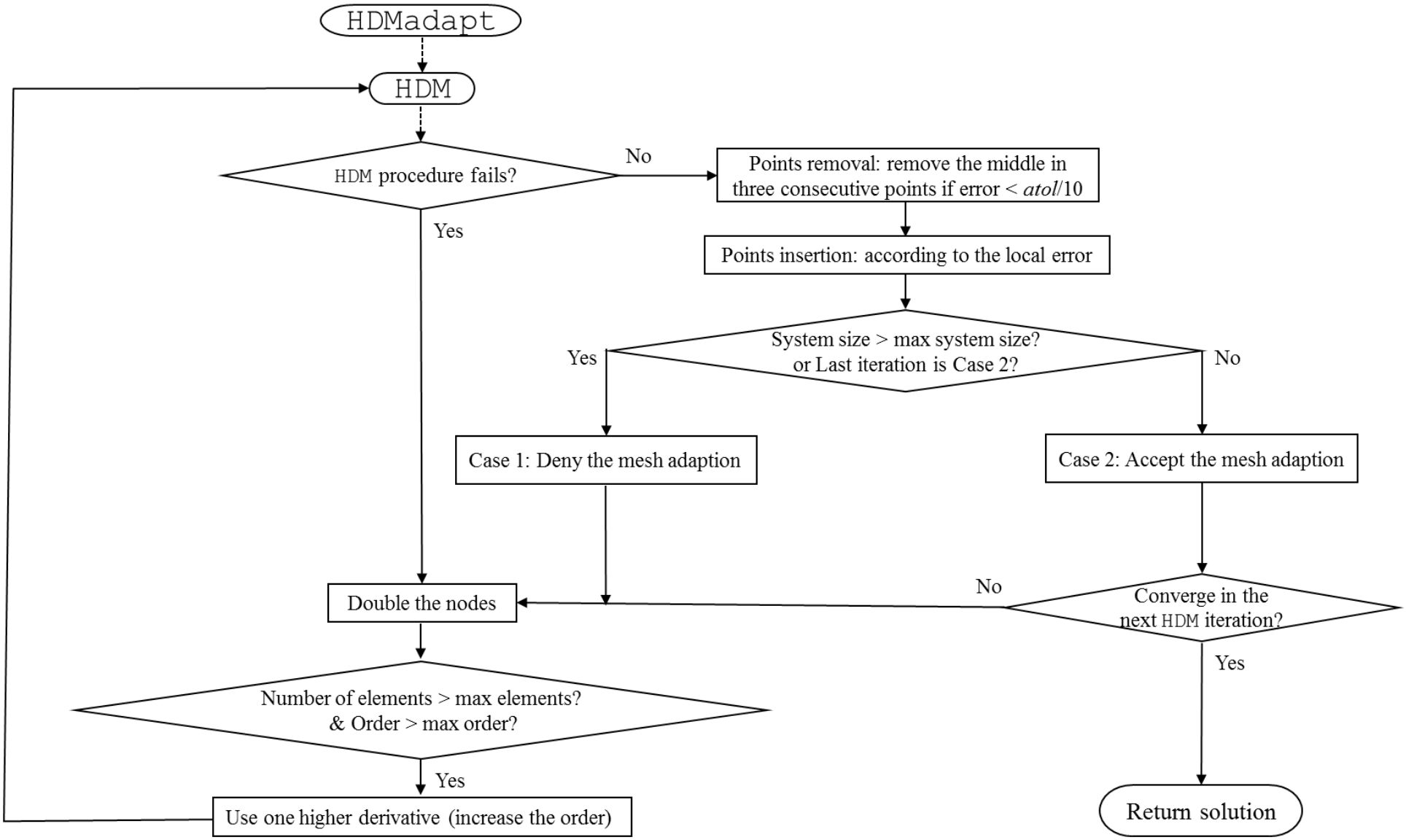


Fig. 4. The adaptive mesh/order strategy

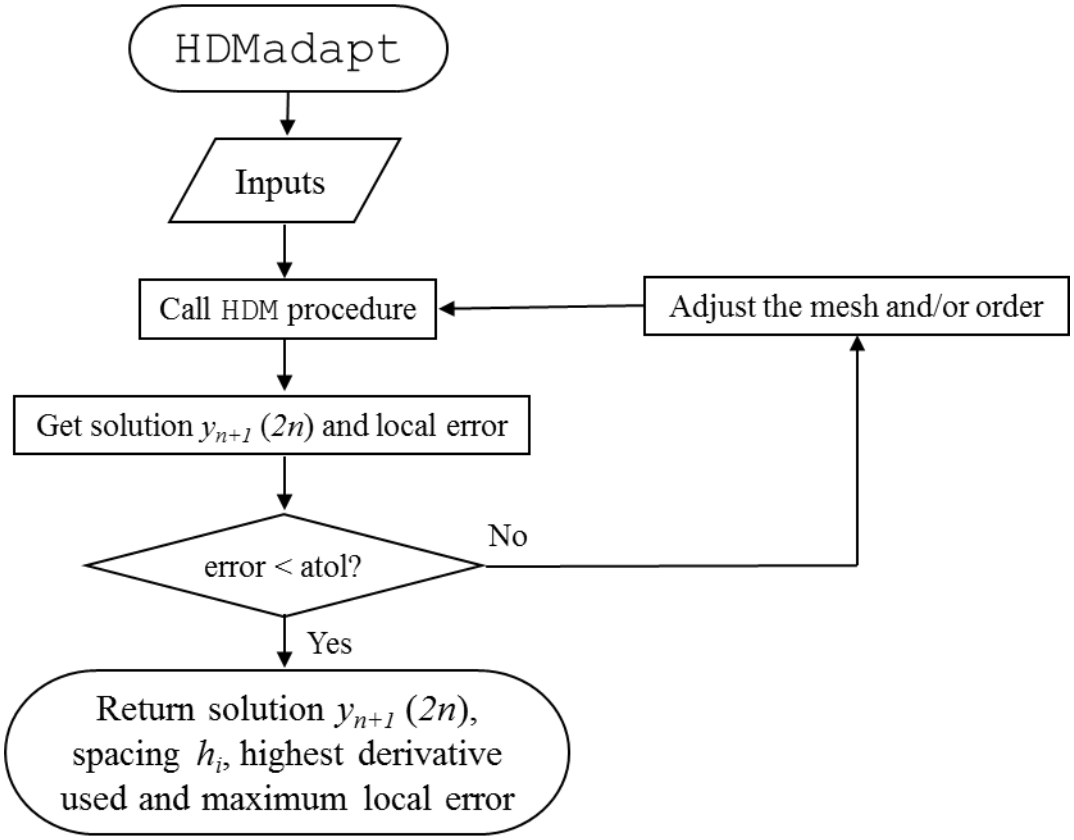


Fig. 5. The flow diagram for HDMadapt procedure

IV. Examples

1. HDM Error Analysis

The error analysis is demonstrated by a simple 2-point BVP as given below

$$y'' = 0.1y \tag{45}$$

with boundary conditions: $y(0) = 1.0$, $y(1) = 0.0$

The analysis is done by choosing different sizes of spacing (h), where the discretization points are equally distributed. In this analysis, 10 elements are chosen (11 node points). All the solutions at the nodes are compared with the values calculated by the analytical solution as given in Eq. (46). The solutions are generated by the HDM code, and the error is defined by the maximum absolute value of differences between the numerical and analytical solutions at each node.

$$y(x) = \frac{e^{-\frac{x}{\sqrt{10}}} - e^{\frac{x-2}{\sqrt{10}}}}{1 - e^{-\frac{2}{\sqrt{10}}}} \tag{46}$$

Plotting the negative log value of the error and the negative log value of the spacing as the coordinates, the slope of the line indicates the numerical order of convergence of the HDM. The linear regression results for the 2^{nd} to 12^{th} order are studied as shown in Table 3 and Fig. 6, and the order (the slope of the regression) obtained is as expected ($2n$), where n is the highest derivative used.

Table 3. The regression results of error analysis for different HDM schemes.

Highest derivative used	order	Regression for y
1	2^{nd}	$-0.774145 + 2.175016h$
2	4^{th}	$-2.794195 + 4.059814h$
3	6^{th}	$-5.543270 + 6.004168h$
4	8^{th}	$-8.830802 + 7.973144h$
5	10^{th}	$-12.547187 + 9.953897h$
6	12^{th}	$-16.632225 + 11.924494h$

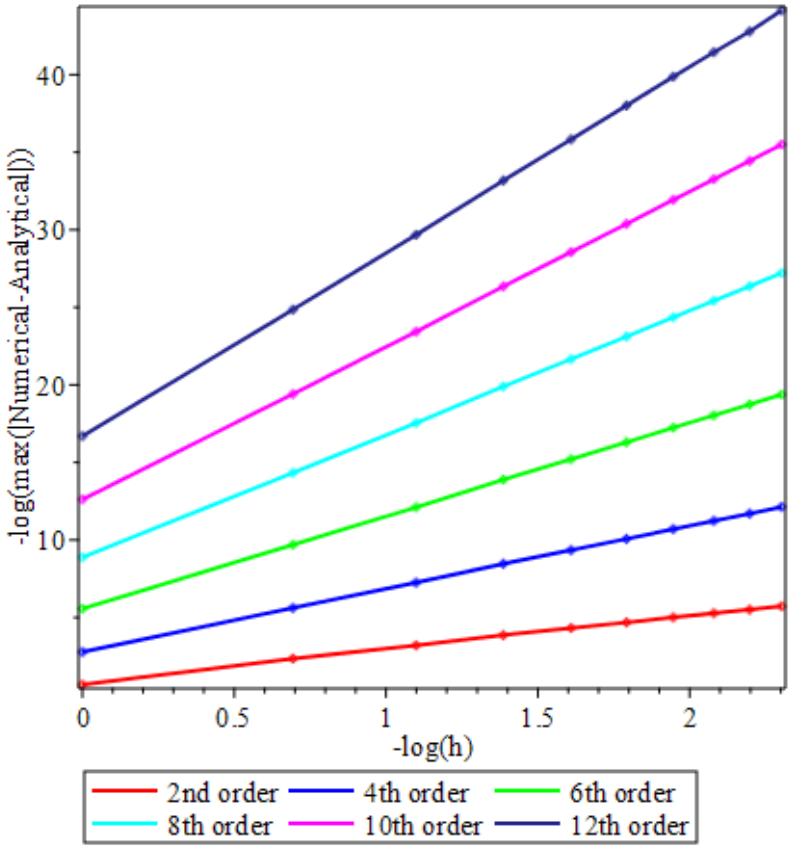


Fig. 6. Error analysis for different HDM schemes for a test BVP.

2. Cash's problem: A singularly perturbed two-point BVP

This second order singularly perturbed two-point BVP as given in Eq. (47) is adopted from Jeff Cash's collection of 35 test problems. This type of problem includes a small, positive parameter (ε) multiplying the highest derivative term in the system of differential equations. The stiffness of the problem increases while with value of the parameter becomes smaller which makes the BVP difficult to solve.

$$\varepsilon y'' - y = 0 \tag{47}$$

with boundary conditions: $y(0) = 1.0$, $y(1) = 0.0$.

The analytical solution for this problem is shown as in Eq. (48).

$$y(x) = \frac{e^{-\frac{x}{\sqrt{\varepsilon}}} - e^{\frac{x-2}{\sqrt{\varepsilon}}}}{1 - e^{\frac{-2}{\sqrt{\varepsilon}}}} \tag{48}$$

To begin with, this second order problem is converted into two first order ODEs and their boundary conditions are $y_1(0)=1.0$, $y_1(1)=0.0$.

$$y_1' = y_2 \tag{49}$$

$$y_2' = \frac{1}{\varepsilon} y_1 \tag{50}$$

A tolerance value of 1e-6, fifth derivative and initial number of elements of 10 are used as the default inputs. Table 4 and Fig. 7 show the results for different values of parameter (ε), 1e0, 1e-1, 1e-2, 1e-3, and 1e-4, respectively. Decreasing the value of ε increases the stiffness of the ODE. The final number of elements used and Root Mean Squared Error (RMSE) are shown in Table 4. This example is computed in Maple® with 15 digits of accuracy.

Table 4. The results for various values of ε in Cash's problem

ε	Initial		Final		RMSE with analytical
	highest derivative	elements	highest derivative	elements	
10^{0}	5	10	5	10	$4.63190516467528\times10^{-16}$
10^{-1}	5	10	5	10	$1.37972329241640\times10^{-15}$
10^{-2}	5	10	5	10	$1.49610391565853\times10^{-11}$
10^{-3}	5	10	5	20	$4.37571658468702\times10^{-11}$
10^{-4}	5	10	5	36	$2.91072853070729\times10^{-12}$

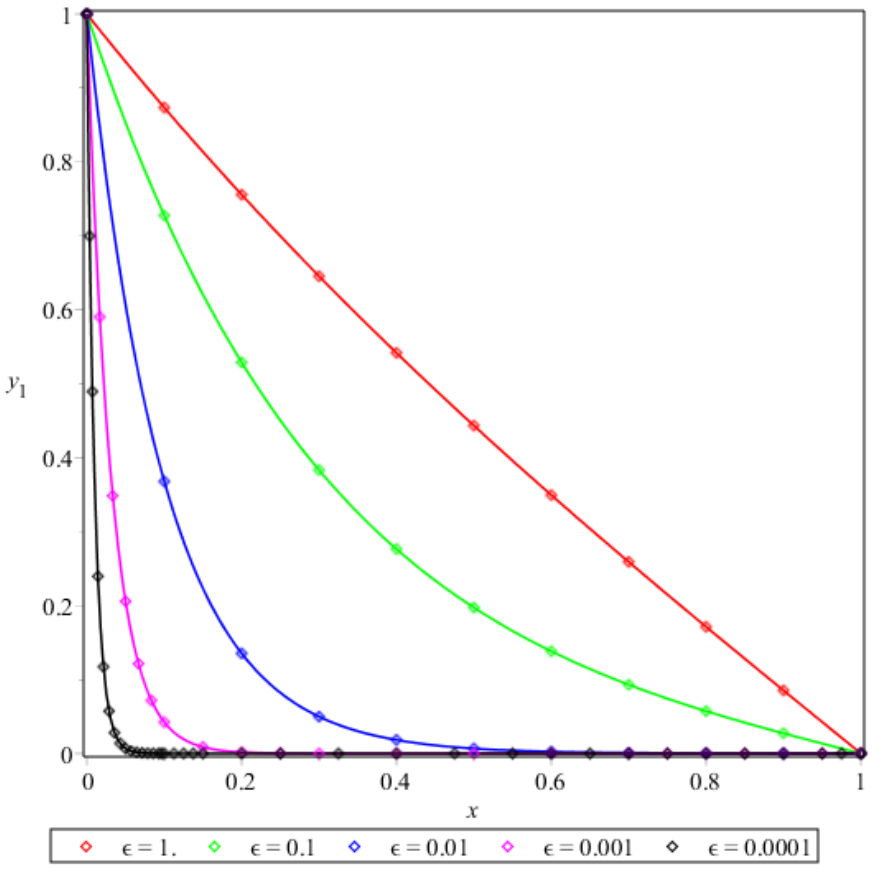


Fig. 7. The results of *y* for Cash's problem (line: analytical solutions; dots: numerical solutions)

All the 35 test problems provided on Cash's website were tested by the proposed HDM approach. The problem set can be found on Jeff Cash's website (http://wwwf.imperial.ac.uk/~jcash/BVP_software/PROBLEMS.PDF) and `HDMadapt` procedure can successfully solve all the listed problems with given perturbation parameters. The complete list of test results and the `Maple`® code can be found at the corresponding author's website (http://depts.washington.edu/maple/HDM.html).

3. Troesch's problem

This is an inherently unstable, difficult, nonlinear, two-point BVP formulated by Weibel [34] and Troesch [35] that describes the confinement of a place column by radiation pressure. Increasing ε increases the stiffness of the ODE. Due to the difficulties involved, several researchers have explored different approaches for solving this problem. For example, Roberts and Shipman [36] used a combination of perturbation technique, the parallel shooting method, and the

continuation method to accurately solve this problem, when $\varepsilon < 5$. Jones [37] used a modified Newton method instead to solve this problem for handling cases when $\varepsilon \geq 5$, but it requires higher number of iterations. Miele et al. [38] used modified quasi-linearization method by dividing the interval of integration into a number of sub-intervals. Although less iterations are used, high numbers of integration steps were required in this case. Chiou and Na [39] used the method of transformation groups to solve this problem; however, the accuracy became a concern in their approach. Vazquez-Leal et al. [40] provided a general solution to the problem by using the homotopy perturbation method. The uses of Sinc-collocation method [41], Sinc-Galerkin method [42] and Christove rational functions [43] for solving this problem are also noteworthy.

The problem is shown in Eq. (51) and it is subjected to the boundary conditions: $y(0) = 0.0$, $y(1) = 1.0$.

$$y'' = \varepsilon \sinh(\varepsilon y), \ \varepsilon > 1 \tag{51}$$

To obtain the solution of the problem, first it has been rewritten in the form of an equivalent first order system as given in Eqs. (52)-(53) with the boundary conditions: $y_1(0) = 0.0$, $y_1(1) = 1.0$.

$$y_1' = y_2 \tag{52}$$

$$y_2' = \varepsilon \sinh(\varepsilon y_1) \tag{53}$$

A tolerance value of 1e-6, fifth derivative, and 10 elements were chosen as the input. Fig. 8 shows the results obtained by the proposed approach and its comparison with the results of Chiou and Na [39] and iterative methods when $\varepsilon = 8$. The final number of the nodes used in the proposed approach is 23 and the proposed mesh adaption strategy works well.

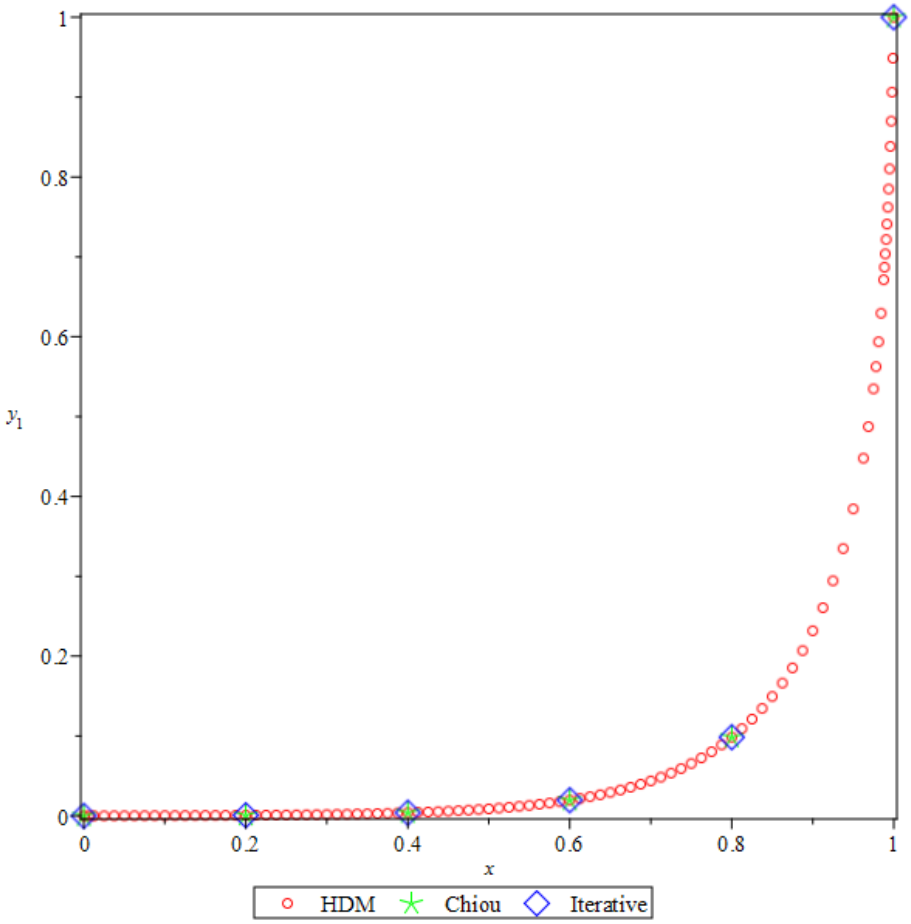


Fig. 8. Results for Troesch's problem

4. Unknown parameter problem

This problem is adopted form Example 1.4 in the book written by Ascher et al. [17] which describes momentum transfer and the heat transfer of fluid injection in a long vertical channel as in Eqs. (54)-(56).

$$f'''-R\left[\left(f'\right)^2-ff''\right]+RA=0 \tag{54}$$

$$h''+Rfh'+1=0 \tag{55}$$

$$\theta''+Pf\theta'=0 \tag{56}$$

with boundary conditions:

when $x = 0$, $f(0)=f'(0)=h(0)=\theta(0)=0$; when $x = 1$, $f(1)=\theta(1)=1$ and $f'(1)=h(1)=0$.

The variables f and h describe the two potential functions, θ is the temperature distribution function, R is the Reynolds number and P is the Peclet number (e.g. take $P = 0.7R$), and A is the undetermined constant. It is difficult to solve this problem numerically not only because of the unknown parameter (A) but also because the Reynolds number determines the stiffness of the problem. Stiffness increases with the Reynolds number (a rapidly changing boundary layer develops near $x = 0$).

In this example, a value of $R = 100$ (i.e. $P = 70$) is chosen, and the nonlinear third-order ODEs are transformed into equivalent first-order ODEs as

$$y_1' = y_2,\ y_2' = y_3,\ \text{and}\ y_3' = -RA + R\left(y_2^2 - y_1 y_3\right) \tag{57}$$

$$y_4' = y_5,\ \text{and}\ y_5' = -1 - R y_1 y_5 \tag{58}$$

$$y_6' = y_7,\ \text{and}\ y_7 = -P y_1 y_7 \tag{59}$$

with $y_1(0) = y_2(0) = y_4(0) = y_6(0) = 0$ and $y_1(1) - 1 = y_2(1) = y_4(1) = y_6(1) - 1 = 0$.

With the default input values, the procedure `HDMadapt` ends up using 42 elements, and 10^{th} order HDM. The unknown parameter A was found to be 2.76063141405118. The result of $y_2 = f'$ is given in Fig. 9. The code for this problem is presented in the Appendix A.

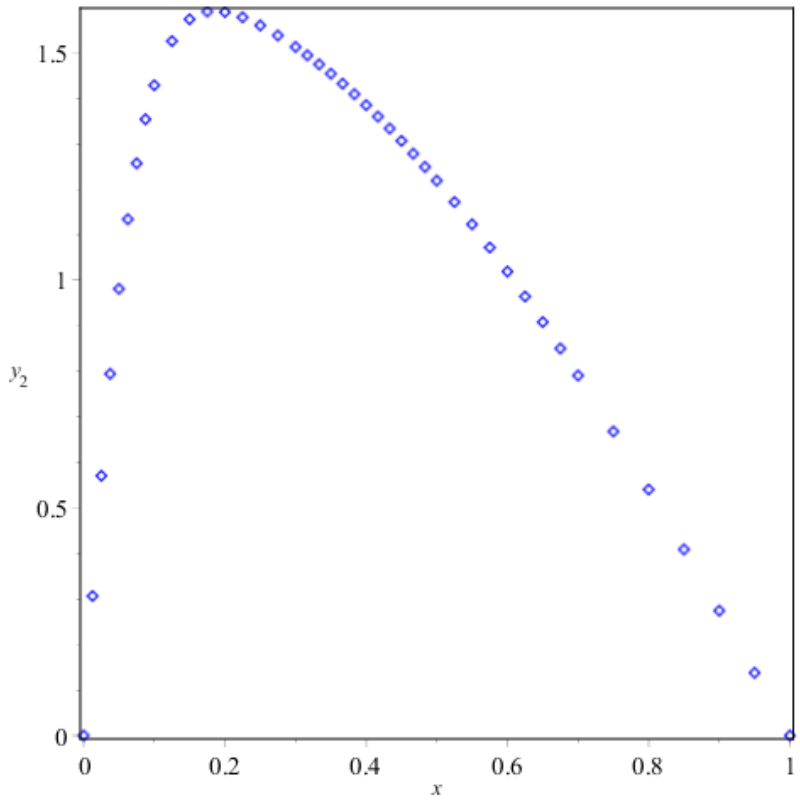


Fig. 9. The first derivative of the fluid potential in a long vertical channel predicted by the HDM scheme.

5. Parameter estimation and optimal control

When parameter estimation or optimal control is performed in a sequential approach, there is a need to solve the same model multiple times. In order to reduce the effort of repeated discretization for a fixed mesh, one can take advantage of the symbolic form of equations and Jacobian to discretize the problem only once. i.e. the discretized HDM scheme can be derived as a function of parameters, and the equation set and the analytical Jacobian are computed only once (as done in `HDMpars` procedure). The discretized HDM set and the Jacobian can be repeatedly called later in Newton-Raphson iterations (`HDMpars2` procedure), so the effort can be greatly reduced in sequential estimation or control problems. Similarly, the error of the HDM scheme for a given problem for different values of the system parameters can be arrived in the same way. This approach can also be used as a continuation strategy for difficult problems. The code for this problem is presented in Appendix B.

A simple model given as Eq. (60) for $\varepsilon = [1e+2, 1e+1, 1e+0, 1e-1, 1e-2]$, respectively, is solved and plotted in Fig. 10. This model describes diffusion with a second-order reaction in a rectangular catalyst pellet.

$$y'' = \varepsilon^2 y^2 \tag{60}$$

with boundary conditions: $y'(0) = 0$ and $y(1) = 1$

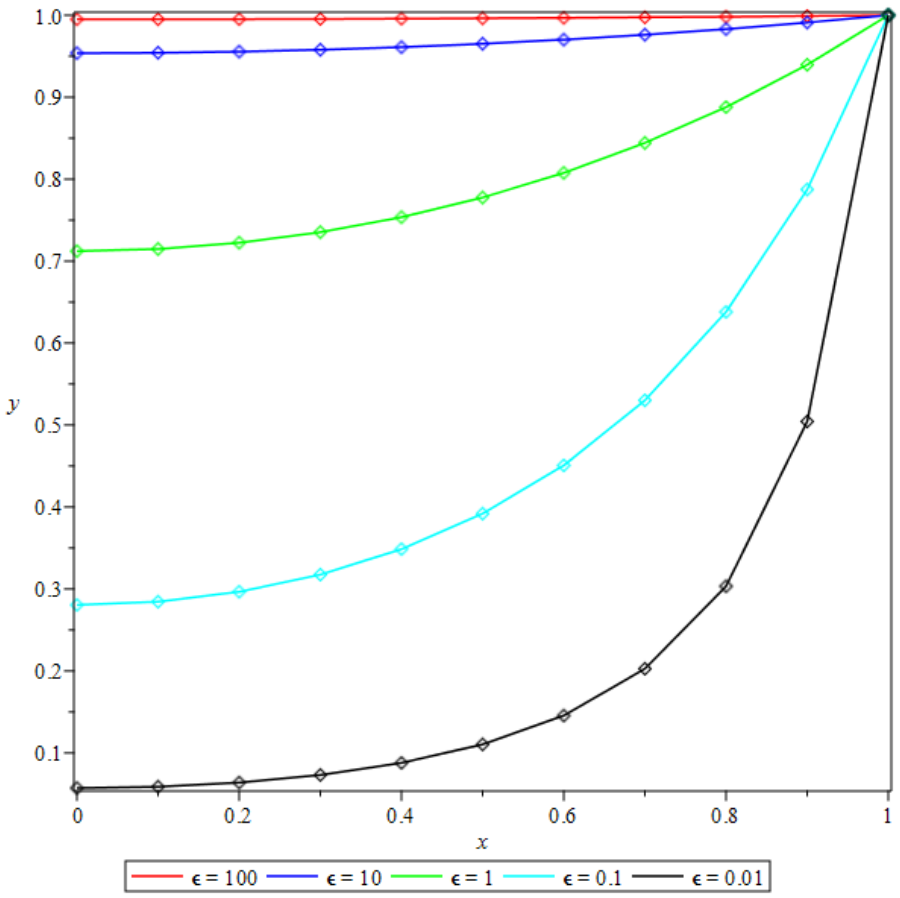


Fig. 10. HDM simulation of diffusion and reaction in a rectangular catalyst pellet for different values of the Thiele modulus. Equations were created only once, but solved for different values of the parameters without recalculating the Jacobian.

V. Conclusions

The HDM method is a single step *A*-stable method that can greatly reduce the size of the discretized system and eventually the computational time while solving BVPs. In addition, solutions obtained at all the node points have the same order of accuracy. The HDM scheme is

generic enough to extend to any desired order. With an efficient Newton-Raphson method based on a sparse linear solver and the adaptive meshing strategy, the HDM scheme can handle many BVPs efficiently.

V. Acknowledgements

The authors would like to thank the Department of Energy (DOE) for providing partial financial support for this work, through the Advanced Research Projects Agency (ARPA-E) award number DE-AR0000275, along with the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Vehicle Technologies of the DOE through the Advanced Battery Material Research (BMR) Program (Battery500 consortium). The authors would also like to thank the Clean Energy Institute (CEI) at the University of Washington (UW) and the Washington Research Foundation (WRF) for their partial monetary support during this work. The corresponding author Venkat Subramanian would like to dedicate this code to professors Ernst Hairer, Jeff Cash, Linda Petzold and Uri Ascher for their open-source codes RADAU (and the classical yellow books), MEBDFI, DASSL and COLDAE and for being an inspiration for this work.

VI. References

[1] J.R. Cash, D.R. Moore, A high order method for the numerical solution of two-point boundary value problems, BIT Numer. Math. 20 (1980) 44–52.

[2] J.C. Butcher, Implicit Runge-Kutta processes, Math. Comput. 18 (1964) 50–64.

[3] W.H. Enright, P.H. Muir, Efficient classes of Runge-Kutta methods for two-point boundary value problems, Computing. 37 (1986) 315–334. =

[4] S. Gupta, An adaptive boundary value Runge–Kutta solver for first order boundary value problems, SIAM J. Numer. Anal. 22 (1985) 114–126.

[5] R. Weiss, The application of implicit Runge-Kutta and collection methods to boundary-value problems, Math. Comput. 28 (1974) 449–464.

[6] H. Keller, Numerical Solution of Two Point Boundary Value Problems, Society for Industrial and Applied Mathematics, 1976.

[7] J.R. Cash, M.P. Garcia, D.R. Moore, Mono-implicit Runge–Kutta formulae for the numerical solution of second order nonlinear two-point boundary value problems, J. Comput. Appl. Math. 143 (2002) 275–289.

[8] B. Ehle, *A*-stable methods and Padé approximations to the exponential, SIAM J. Math. Anal. 4 (1973) 671–680.

[9] E. Gad, M. Nakhla, R. Achar, Y. Zhou, *A*-Stable and *L*-Stable high-order integration methods for solving stiff differential equations, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 28 (2009) 1359–1372.

[10] N. Obrechkoff, Sur le quadrature mecaniques, Spis. Bulg. Akad Nauk. Bulg. Acad. Sci. 65 (1942) 191–289.

[11] J.R. Cash, A. Singhal, High order methods for the numerical solution of two-point boundary value problems, BIT Numer. Math. 22 (1982) 183–199.

[12] A.B. White, Numerical solution of two-point boundary-value problems (PhD Thesis), California Institute of Technology, 1974.

[13] L.F. Shampine, I. Gladwell, S. Thompson, Solving ODEs with MATLAB, Cambridge University Press, 2003.

[14] J. Kierzenka, L.F. Shampine, A BVP Solver Based on Residual Control and the Maltab PSE, ACM Trans Math Softw. 27 (2001) 299–316.

[15] U.M. Ascher, J. Christiansen, R.D. Russell, COLSYS -- A collocation code for boundary-value problems, in: Proceedings of a Working Conference on Codes for Boundary-Value Problems in Ordinary Differential Equations, Springer-Verlag, London, UK, 1979: pp. 164–185.

[16] U. Ascher, J. Christiansen, R.D. Russell, Collocation software for boundary-value ODEs, ACM Trans Math Softw. 7 (1981) 209–222.

[17] U. Ascher, R. Mattheij, R. Russell, Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, Society for Industrial and Applied Mathematics, 1995.

[18] G. Bader, U. Ascher, A new basis implementation for a mixed order boundary value ode solver, SIAM J. Sci. Stat. Comput. 8 (1987) 483–500.

[19] J. Cash, M. Wright, A Deferred Correction Method for Nonlinear Two-Point Boundary Value Problems: Implementation and Numerical Evaluation, SIAM J. Sci. Stat. Comput. 12 (1991) 971–989.

[20] W. Enright, P. Muir, Runge–Kutta Software with Defect Control for Boundary Value ODEs, SIAM J. Sci. Comput. 17 (1996) 479–497.

[21] W.E. Milne, Numerical Solution of Differential Equations, John Wiley & Sons, 1953.

[22] Maple, Maplesoft, https://www.maplesoft.com/.

[23] E. Hairer, G. Wanner, Stability function of Implicit RK-Methods, in: Solving Ordinary Differential Equations II, Springer, Berlin, Heidelberg, 1996: pp. 40–50.

[24] G. Birkhoff, R.S. Varga, Discretization errors for well-Set Cauchy problems. I., J. Math. Phys. 44 (1965) 1–23.

[25] B.L. Ehle, High order *A*-stable methods for the numerical solution of systems of D.E.'s, BIT Numer. Math. 8 (1968) 276–278.

[26] G.G. Dahlquist, A special stability problem for linear multistep methods, BIT Numer. Math. 3 (1963) 27–43.

[27] T. Fort, Finite differences and difference equations in the real domain, Clarendon Press, 1948.

[28] F.B. Hildebrand, Introduction to Numerical Analysis, McGraw-Hill, 1974.

[29] W. Enright, Second derivative multistep methods for stiff ordinary differential equations, SIAM J. Numer. Anal. 11 (1974) 321–331.

[30] F.H. Chipman, *A*-stable Runge-Kutta processes, BIT Numer. Math. 11 (1971) 384–388.

[31] B.L. Ehle, Z. Picel, Two-parameter, arbitrary order, exponential approximations for stiff equations, Math. Comput. 29 (1975) 501–511.

[32] B.L. Ehle, On Padé approximations to the exponential function and *A*-stable methods for the numerical solution of initial value problems, University of Waterloo, 1969.

[33] P.M. Hummel, C.L. Seebeck, A generalization of Taylor's expansion, Am. Math. Mon. 56 (1949) 243–247.

[34] E.S. Weibel, On the confinement of a plasma by magnetostatic fields, Phys. Fluids. 2 (1959) 52–56.

[35] B.. Troesch, A simple approach to a sensitive two-point boundary value problem, J. Comput. Phys. 21 (1976) 279–290.

[36] S.. Roberts, J.. Shipman, On the closed form solution of Troesch's problem, J. Comput. Phys. 21 (1976) 291–304.

[37] D.. Jones, Solution of Troesch's, and other, two point boundary value problems by shooting techniques, J. Comput. Phys. 12 (1973) 429–434.

[38] A. Miele, A. Aggarwal, J.. Tietze, Solution of two-point boundary-value problems with Jacobian matrix characterized by large positive eigenvalues, J. Comput. Phys. 15 (1974) 117–133.

[39] J.P. Chiou, T.Y. Na, On the solution of Troesch's nonlinear two-point boundary value problem using an initial value method, J. Comput. Phys. 19 (1975) 311–316.

[40] H. Vazquez-Leal, Y. Khan, G. Fernández-Anaya, A. Herrera-May, A. Sarmiento-Reyes, U. Filobello-Nino, Jimenez-Fernández, V.-M. Jimenez-Fernández, D. Pereyra-Díaz, A General Solution for Troesch's Problem, Math. Probl. Eng. (2012).

[41] M. El-Gamel, Numerical solution of Troesch's problem by Sinc-collocation method (2013).

[42] M. Nabati, M. Jalalvand, Solution of Troesch's problem through double exponential Sinc-Galerkin method, Comput. Methods Differ. Equ. 5 (2017) 141–157.

[43] A. Saadatmandi, T. Abdolahi-Niasar, Numerical solution of Troesch's problem using Christov rational functions, Comput. Methods Differ. Equ. 3 (2015) 247–257.

Appendix A: Package HDM, solving boundary value problems in Maple®

A.1 Introduction

The package HDM solves boundary value problems (BVPs) using higher derivative methods (HDM) in Maple®. In this appendix, we explain how to solve BVPs using this package. HDM can numerically solve BVPs of ordinary differential equations (ODEs) of the form shown in Eq. (61).

$$\frac{dy}{dx} = f\left(x, y\right),\ x \in \left[a, b\right],\ y \in \mathrm{R}^s \tag{61}$$

With boundary conditions as $y\left(a\right) = \alpha$, $y\left(b\right) = \beta$.

A.2 A BVP Example

This ***unknown parameter problem*** is taken from ***Example 4*** in this paper which describes momentum and heat transfer of fluid injection in a long vertical channel. The ODEs and BCs are as Eqs. (57)-(59).

The statements in red in the following text are Maple commands and are to be executed on Maple command prompt.

Reset the program to clear the memory from previous execution command.

```
> restart:
```

Read the txt file which contains the HDM solver for BVPs.

```
> read("HDM.txt"):
```

Declare the precision for the entire Maple® sheet.

```
> Digits:=15;
```

Enter the first-order ODEs into `EqODEs` list.

>

```
EqODEs:=[diff(y1(x),x)=y2(x),diff(y2(x),x)=y3(x),diff(y3(x),x)=-
R*A+R*(y2(x)^2-y1(x)*y3(x)),diff(y4(x),x)=y5(x),diff(y5(x),x)=-
1-R*y1(x)*y5(x),diff(y6(x),x)=y7(x),diff(y7(x),x)=-
P*y1(x)*y7(x)];
```

Define the left boundary condition (`bc1`), and the right boundary condition (`bc2`). One should collect all the terms in one side.

```
> bc1:=evalf([y1(x),y2(x),y4(x),y6(x)]);
> bc2:=evalf([y1(x)-1,y2(x),y4(x),y6(x)-1]);
```

Define the range (`bc1` to `bc2`) of this BVP.

```
> Range:=[0.,1.];
```

List any known parameters in the list.

```
> pars:=[P=70,R=100];
```

List any unknown parameters in the list. When there is no unknown parameter, use [ ].

```
> unknownpars:=[A];
```

Define the initial derivative in `nder` (default is 5 for 10^{th} order) and the number of the nodes in `nele` (default is 10 and distributed evenly across the range provided by the user). The code adapts to increase the order. For many problems, 10^{th} order method with 10 elements are sufficient.

```
> nder:=5;nele:=10;
```

Define the absolute and relative tolerance for the local error. The error calculation is done based on the norm of both the 9^{th} and 10^{th} order simulation results.

```
> atol:=1e-10;rtol:=atol/100;
```

Call `HDMadapt` procedure, input all the information entered above and save the solution in `sol`.

`HDMadapt` procedure does not need the initial guess for the mesh.

```
> sol:=
HDMadapt(EqODEs,bc1,bc2,pars,unknownpars,nder,nele,Range,atol,rtol):
```

Present some details of the solution.

```
> sol[1][(nops(sol[3])+1)*nops(EqODEs)+1..nops(sol[1])]; #
unknownpars
```

$$[A = 2.76063141405118]$$

```
> sol[2]; # local error
```

$$[0.516017280511727\ 10^{-13}, 0.280404478021445\ 10^{-13}, 0.134899333565169\ 10^{-13},$$
$$0.843380403070125\ 10^{-14}, 0.167535225524744\ 10^{-13},$$
$$0.180234435328161\ 10^{-13}, 0.130081862677236\ 10^{-13},$$
$$0.451621256032576\ 10^{-14}, 0.749079157248931\ 10^{-11},$$
$$0.122411259557231\ 10^{-10}, 0.609079772435872\ 10^{-11},$$
$$0.179853244150479\ 10^{-11}, 0.424861368893361\ 10^{-11},$$
$$0.347122731398200\ 10^{-11}, 0.122317276291983\ 10^{-11},$$
$$0.857871045290069\ 10^{-12}, 0.186057114444594\ 10^{-13},$$
$$0.174392022266724\ 10^{-13}, 0.122639604223559\ 10^{-13},$$
$$0.626123370917862\ 10^{-14}, 0.145217247840687\ 10^{-14},$$
$$0.142198120732110\ 10^{-14}, 0.253269624925408\ 10^{-14},$$
$$0.247082309631126\ 10^{-14}, 0.185661718412170\ 10^{-14},$$
$$0.113883175163602\ 10^{-14}, 0.553480170974474\ 10^{-15},$$
$$0.171223827678051\ 10^{-15}, 0.331492786706419\ 10^{-14},$$
$$0.681872237775073\ 10^{-14}, 0.513329216422696\ 10^{-14},$$
$$0.284943314986258\ 10^{-14}, 0.131819560402305\ 10^{-14},$$
$$0.533280411666936\ 10^{-15}, 0.193552603916320\ 10^{-15},$$
$$0.640536040213990\ 10^{-16}, 0.114863358153041\ 10^{-13},$$
$$0.826247508702572\ 10^{-15}, 0.681890398190881\ 10^{-16},$$
$$0.797287234807799\ 10^{-15}, 0.839249720850315\ 10^{-16},$$
$$0.433065969235651\ 10^{-15}]$$

```
> sol[3]; # element spacing
```

```
[ 0.0125000000000000  , 0.0125000000000000  , 0.0125000000000000  ,
    0.0125000000000000  , 0.0125000000000000  , 0.0125000000000000  ,
    0.0125000000000000  , 0.0125000000000000  , 0.0250000000000000  ,
    0.0250000000000000  , 0.0250000000000000  , 0.0250000000000000  ,
    0.0250000000000000  , 0.0250000000000000  , 0.0250000000000000  ,

    0.0250000000000000  , 0.0166666666666667  , 0.0166666666666667  ,
    0.0166666666666667  , 0.0166666666666667  , 0.0166666666666667  ,
    0.0166666666666667  , 0.0166666666666667  , 0.0166666666666667  ,
    0.0166666666666667  , 0.0166666666666667  , 0.0166666666666667  ,
    0.0166666666666667  , 0.0250000000000000  , 0.0250000000000000  ,

    0.0250000000000000  , 0.0250000000000000  , 0.0250000000000000  ,
    0.0250000000000000  , 0.0250000000000000  , 0.0250000000000000  ,
    0.0500000000000000  , 0.0500000000000000  , 0.0500000000000000  ,
    0.0500000000000000  , 0.0500000000000000  , 0.0500000000000000  ]
```

```
> sol[4]; # final order
```

$$5$$

```
> sol[5]; # Maximum local RMSE
```

$$0.122411259557231 \quad 10^{-10}$$

Store the dimension of the solution (after adjusting the mesh) to NN.

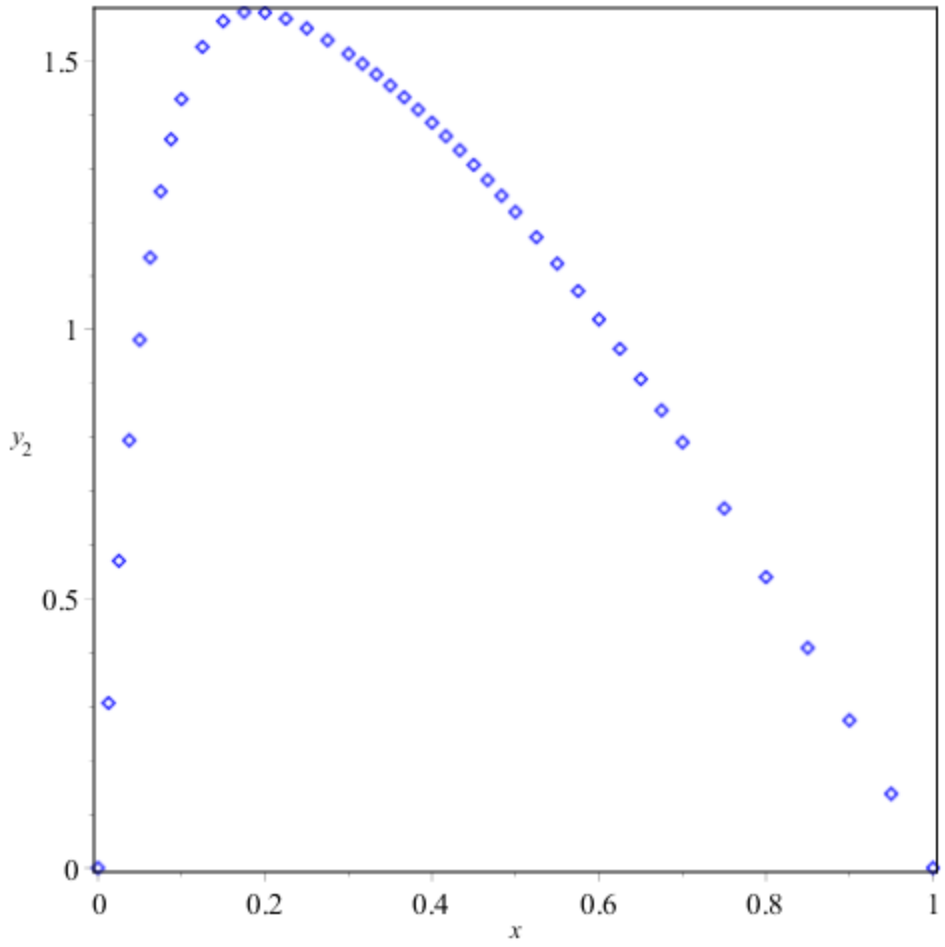
```
> NN:=nops(sol[3])+1;
```

$$NN := 43$$

Plot the interested variable (the $\underline{a}^{th}$ ODE variable will be sol[1][i+NN***(a-1)***])

```
> xx:=Vector(NN):

> xx[1]:=Range[1]:
> for i from 1 to nops(sol[3]) do xx[i+1]:=xx[i]+sol[3][i]: od:
>
plot([seq([xx[i],rhs(sol[1][i+NN*1])],i=1..NN)],axes=boxed,style
=point);
```


1.5
1
0.5
0
y_2
0
0.2
0.4
0.6
0.8
1
x

Appendix B: Procedure HDMpars, HDMpars2 in Maple®

B.1 Introduction

HDMpars generates the HDM equation set, Jacobian for the Newton-Raphson procedure, and the error formulas for the HDM as a function of the parameters of interest; HDMpars2 performs Newton-Raphson iterations.

A.2 A BVP Example

The functions are the same as in Appendix A.

```
> restart:
> read("HDM.txt");
> Digits:=15;
> EqODEs:=[diff(y1(x),x)=y2(x),diff(y2(x),x)=1/epsilon*y1(x)^2];
> bc1:=evalf([y2(x)]);
> bc2:=evalf([y1(x)-1]);
> Range:=[0.,1.];
> unknownpars:=[];
> nder:=5;nele:=10;
```

Define the element size of each element.

```
> h1:=Vector(nele,[seq((Range[2]-Range[1])/nele,i=1..nele)]):
```

Define the symbolic parameters

```
> sympars:=[epsilon];
```

Generate the HDM equation set, Jacobian for Newton-Raphson, and the error formulas for the HDM and store them into Gen.

```
>
Gen:=HDMpars(EqODEs,bc1,bc2,sympars,unknownpars,nder,nele,Range,
h1,atol,rtol):
```

Give the initial conditions for the variables at the node points

```
> ic0:=[seq(0.5,i=1..(nele+1)*nops(EqODEs)),seq(2,i=1..nops(unknownpars))];
```

List of values of parameters

```
> p:=[100,10,1,1e-1,1e-2,1e-3];
```

Substitute the parameter individually and execute the Newton-Raphson iteration. The solutions and the error at each node points will be returned. Note that one can use sol[1] as ic0 for sol[2] for continuation strategy.

```
> for i from 1 to 5 do
  pars:=[epsilon=p[i]];
  sol[i]:=HDMpars2(Gen,ic0,pars);
end do:
```

Plot the solutions obtained. The plot will be the same as Fig. 10.

```
> xx:=Vector(nele+1):
> xx[1]:=Range[1]:
> for i from 1 to nele do xx[i+1]:=xx[i]+h1[i]: od:
> clr:=[red,blue,green,cyan,black]:
> for i from 1 to 5 do
pp[i]:=plot([seq([xx[j],sol[i][1][j]],j=1..nele+1)],axes=boxed,color=clr[i],legend=[epsilon=p[i]]):
end do:
> plots:-display(seq(pp[i],i=1..5),style=pointline,labels=[x,y]);
```

Attachment- Code for HDM

Click here to view linked References

```
HDMadapt:=proc(EqODEs,bc1,bc2,pars,xpars,nder,nele,Range,atol,rtol)
    local hnew,N,ics,maxiter,ErrMax,k,sol,ErrList,ErrMin,x0,i,
          x0old,OKpts,Rmvindx,Wntindx,x0wnt,hwnt,ErrListwnt,hpredict,Nsub,ndernew,
          maxnele,maxnder,maxsize,cnt,rmv,nODEs,Ntot,NN,adaptflag,catchflag,nWntindx,nxpars;

    # Copyright: Dr. Venkat R.Subramanian and MAPLE group members at UW.
    # Original code is written by V. R. Subramanian (vsubram@uw.edu).
    # Last modified by Jerry Chen (jerrysyc@uw.edu) on 2017/07/28

    optimize;

    # Use uniform mesh as a start
    hnew:=[seq((Range[2]-Range[1])/nele,i=1..nele)]:
    N:=nele;
    #ics:=ic0;
    ndernew:=nder;
    nODEs:=nops(EqODEs);
    nxpars:=nops(unknownpars);
    ics:=[seq(0.5,i=1..(N+1)*nODEs+nxpars)];

    # maxiter for adapt
    maxiter:= 20;
    ErrMax:=10;
    maxnele:=200;
    maxnder:=9;
    maxsize:=500; # maximum size of system


    # loop till we find the converged answer
    for k to maxiter while ErrMax > atol or ndernew < maxnder+1 do

       try
         sol:=HDM(EqODEs,bc1,bc2,pars,xpars,ndernew,N,Range,ics,hnew,atol,rtol);

         # store the norm error as a list
         ErrList:=op(sol[2]);

         # find the min and max for the judging criteria
         ErrMax:=max(ErrList);
```

```
    ErrMin:=min(ErrList);

    catchflag:= 0;

catch:
    print("HDM fail, increase points to...", 2*N);
    N:=2*N;
    hnew:=[seq((Range[2]-Range[1])/N,i=1..N)]:
    ics:=[seq(0.5,i=1..(N+1)*nODEs+nxpars)];
    if ndernew <= maxnder and N > maxnele then ndernew:= ndernew+1; end if;
    catchflag:= 1;
end try:

  # confirm to find the converged answer
  if ErrMax < atol then break; end;

  # if the situation is really bad (error is really big) or cannot find the answer => double the nodes
  if ErrMin > 0.1 and catchflag = 0 then
      print("ErrMin > 0.1, double the mesh to...",2*N);
      hnew:=[seq(seq(hnew[i]/2,j=1..2),i=1..N)];
      NN:=[seq(op([1,seq(2,i=1..N)]),j=1..nODEs)];
      ics:=[seq(seq(seq(map(rhs,sol[1])[(k-1)*(N+1)+i],j=1..NN[i]),i=1..N+1),k=1..nODEs),
                seq(rhs(sol[1][nODEs*(N+1)+i]),i=1..nxpars)];
      N:=2*N;

  # minimize the error by insertion local points
  elif catchflag = 0 then

      # initial mesh points before add and removal
      x0:=Vector(N+1,datatype=float[8]);
      x0[1]:=Range[1]:
      for i from 2 to N+1 do x0[i]:=x0[i-1]+hnew[i-1]:od:
      x0old:=[seq(x0[i],i=1..N+1)]; #print("initial mesh points before add and removal",x0old);

      # Find the index of the norm ErrList which is less then (atol/10)
      OKpts:=[seq(`if`(ErrList[i]<atol/10,i,NULL),i=1..N)];

      # Based on OKpts, find the which index should be removed
      # Removal rules: in a grouping of 3, remove 2 (4) point from 1,2,3,(4,5)
      cnt:=1;
      for i from 1 to nops(OKpts)-2 do
```

```
   if OKpts[i+2]=OKpts[i]+2 then
      rmv[cnt]:=i+1;
      i:=i+1;
      cnt:=cnt+1;
   end if;
end do;
Rmvindx:=[seq(rmv[i],i=1..cnt-1)];
Wntindx:=remove(has,[seq(i,i=1..N)],Rmvindx);
nWntindx:= nops(Wntindx);

# New nodes and h after removal
x0wnt:=[x0[1],seq(x0[i+1],i=Wntindx)];
hwnt:=[seq(x0wnt[i+1]-x0wnt[i],i=1..nWntindx)];
ErrListwnt:=[seq(ErrList[i],i=Wntindx)];

# Find the new h based on the 2n-1 accuracy
## Nsub = number of insertion points for subintervals
## hpredict = predict spacing
for i from 1 to nWntindx do hpredict[i]:=hwnt[i]*(atol/ErrListwnt[i])^(1/(2*nder-1));od;
for i from 1 to nWntindx do Nsub[i]:=trunc(hwnt[i]/hpredict[i])+1; od;
Ntot:=add(Nsub[i],i=1..nWntindx);

# update the info for the new mesh
if (Ntot > maxnele or Ntot*nODEs > maxsize) or adaptflag = 1 then
   print("case1 - double the nodes to...",2*N);
   hnew:=[seq(seq(hnew[i]/2,j=1..2),i=1..N)];
   NN:=[seq(op([1,seq(2,i=1..N)]),j=1..nODEs)];
   if adaptflag = 1 then
      ics:=[seq(seq(seq(map(rhs,sol[1])[(k-1)*(N+1)+i],j=1..NN[i]),i=1..N+1),k=1..nODEs),
            seq(rhs(sol[1][nODEs*(N+1)+i]),i=1..nxpars)];
   else
      ics:=[seq(0.5,i=1..(2*N+1)*nODEs+nxpars)];
   end if:
   if ndernew <= maxnder and N > maxnele then ndernew:= ndernew+1; end if:
   N:=2*N;
   adaptflag:= 0;
else
   print("case2 - insert/remove points based on 2n-1 error to...",Ntot);
   hnew:=[seq(seq(hwnt[i]/Nsub[i],j=1..Nsub[i]),i=1..nWntindx)];
   NN:=[seq(op([1,seq(Nsub[i],i=1..nWntindx)]),j=1..nODEs)];
   ics:=[seq(seq(seq(map(rhs,sol[1])[(k-1)*(nWntindx+1)+i],j=1..NN[i]),i=1..nWntindx+1),k=1..nODEs),
```

```
                    seq(rhs(sol[1][nODEs*(nWntindx+1)+i]),i=1..nxpars)];
            N:=Ntot;
            adaptflag:= 1;
        end if;

    end if; # End with decision if ErrMin > 0.1

end do; # close the loop for maxiter

# Return
[sol[1],([seq(evalf(ErrList[i]),i=1..N)]),hnew,ndernew,ErrMax];

end proc:


HDM:=proc(eqodes,bc1,bc2,pars,xpars,nder,N,rng,ic,hh,atol,rtol)
    local node,odevars,vars,f,i,F,x0,pade_alpha,pade_beta,alpha,beta,
        alpha1,beta1,YY,Eqq,Eqs,varsNR,iter,guess,sol,Err,nxpars;

    # Copyright: Dr. Venkat R.Subramanian and MAPLE group members at UW.
    # Original code is written by V. R. Subramanian (vsubram@uw.edu).
    # Last modified by Jerry Chen (jerrysyc@uw.edu) on 2017/08/03

    optimize;

    # Define the variables
    node:=nops(eqodes);
    odevars:=select(type,map(op,map(lhs,EqODEs)),'function');
    vars:=[seq(odevars[i]=Y||i[j],i=1..node)]; # i=vars, j=nodes

    # Find the derivatives
    f[1]:=Vector(node,[seq(rhs(eqodes[i]),i=1..node)]);

    for i from 2 to nder do
      f[i]:=simplify(subs(eqodes,Vector(node,[seq(diff(f[i-1][j],x),j=1..node)])));
    end do:

    for i from 1 to nder do
      F[i]:=unapply(subs(op(pars),vars,f[i]),j,x);
    od:
```

```
# Initialize the mesh
x0:=Vector(N+1,datatype=float[8]):
x0[1]:=rng[1]:
for i from 1 to N do x0[i+1]:=x0[i]+hh[i]: od:

# Pade coefficients
pade_alpha:=unapply((m+n-ii)!*(m)!/((m+n)!*ii!*(m-ii)!),m,n,ii):
pade_beta:=unapply((m+n-ii)!*(n)!/((m+n)!*ii!*(n-ii)!),m,n,ii):

for i from 1 to nder do
   alpha[i]:= pade_alpha(nder,nder,i);
   beta[i]:= alpha[i];
   beta1[i]:= pade_beta(nder-1,nder,i);
end do:

for i from 1 to nder-1 do
   alpha1[i]:=pade_alpha(nder-1,nder,i);
end do:
alpha1[nder]:=0;

# HDM formula
YY:=unapply(Vector(node,[seq(Y||i[j],i=1..node)]),j);
for i from 1 to N do
   Eqq[i]:=eval(YY(i-1)-YY(i)+add(hh[i]^j*(alpha[j]*F[j](i-1,x0[i])+(-1)^(j+1)*beta[j]*F[j](i,x0[i+1]))
            ,j=1..nder));
od;

Eqs:=evalf([seq(seq(Eqq[i][j],i=1..N),j=1..node),
              op(subs(op(pars),vars,j=0,x=rng[1],bc1)),op(subs(op(pars),vars,j=N,x=rng[2],bc2))]);

# vars for NR
varsNR:=[seq(seq(Y||j[i],i=0..N),j=1..node),op(xpars)]:

# NR
iter:=15;
nxpars:=nops(xpars);
guess:=[seq(varsNR[i]=ic[i],i=1..node*(N+1)+nxpars)];
sol:=SNewton(Eqs,guess,atol,rtol,iter,N,nxpars):

# Error (2n HDM formula - (2n-1) HDM formula) - Note: not consider error at the first point, x0
for i from 1 to N do
```

```
        Err[i]:=evalf(LinearAlgebra:-Norm(subs(sol,add(hh[i]^j*((alpha[j]-alpha1[j])*F[j](i-1,x0[i])
                    +(-1)^(j+1)*(beta[j]-beta1[j])*F[j](i,x0[i+1])),j=1..nder)))/sqrt(node));
    end do;

    # Return
    [sol,([seq(evalf(Err[i]),i=1..N)])];

end proc:


SNewton := proc (Eqs, ics, atol, rtol, iter, nele, nxpars)
    local Jac,N,scale,vars,Eqs1,yold,jac,i,LL,j,F,k,errL,dy,ynew,rhsics,N1,bcindex;

    # Copyright: Dr. Venkat R.Subramanian and MAPLE group members at UW.
    # Original code is written by V. R. Subramanian (vsubram@uw.edu).
    # Last modified by Jerry Chen (jerrysyc@uw.edu) on 2017/08/03

    optimize;

    # Initialization
    N:=nops(Eqs);
    errL := 10;

    # Input check
    if nops(Eqs) <> nops(ics) then ERROR("Check the number of equations and variables"); end;
    if nops(remove(has,indets(Eqs,name),map(lhs,ics))) <> 0 then
        ERROR("Variables' name don't match");
    end;

    # Scale based on the ics to force all y caculate from 0 to 1.
    rhsics:=map(rhs,ics);
    vars:=[seq(lhs(ics[i])=y[i],i=1..N)];
    yold :=evalf(Vector([seq(rhsics[i],i=1..N)]));

    # Create equation set
    Eqs1:=subs(vars,Eqs);
    F := unapply(`<,>`(seq(Eqs1[i],i = 1 .. N)),y);

    # Create Jacobian
    jac := Matrix(1 .. N,1 .. N,storage = sparse);
    N1:=(N-nxpars)/(nele+1);
```

```
for k from 1 to N1 do
   for i from 1 to nele do
      for j from 1 to N1 do
         jac[i+(k-1)*nele,i+(j-1)*(nele+1)]:=diff(Eqs1[i+(k-1)*(nele)],y[i+(j-1)*(nele+1)]);
         jac[i+(k-1)*nele,i+1+(j-1)*(nele+1)]:=diff(Eqs1[i+(k-1)*(nele)],y[i+1+(j-1)*(nele+1)]);
      od:
   od:
od:
bcindex:=[seq(1+(i-1)*(nele+1),i=1..N1),seq(nele+1+(i-1)*(nele+1),i=1..N1)];
for i from N1*nele+1 to N-nxpars do
   for j from 1 to nops(bcindex) do
      jac[i,bcindex[j]]:=diff(Eqs1[i],y[bcindex[j]]);
   od:
od:
for j from N-nxpars+1 to N do
   for i from 1 to N do
      jac[i,j]:=diff(Eqs1[i],y[j]):
      jac[j,i]:=diff(Eqs1[j],y[i]):
   od:
od:
Jac:=unapply(jac,y);

# Iterate until the y converges less then the tolerence
for k to iter while errL > 1 do

   # Avoid version caused "SparseDirect" failed happens in Maple classic worksheet
   dy := LinearAlgebra:-LinearSolve(-Jac(yold),F(yold),method=SparseDirect);

   if add(`if`(type((dy[i]),numeric),0,1),i=1..N)>0 then ERROR("Cannot converge"); end if;
   ynew := yold + dy;
   errL := LinearAlgebra:-Norm(dy,2)/LinearAlgebra:-Norm(Vector([seq(atol+rtol*ynew[i],i=1..N)]),2);

   # update yold to next iteration
   yold := ynew;

end do;

if iter < k then ERROR("Exceed max NR iteration: %1",iter) end if;

# Return
[seq(lhs(ics[i])=ynew[i],i=1..N)]; # with name
```

```
end proc:


HDMpars:=proc(eqodes,bc1,bc2,pars,xpars,nder,nele,rng,hh,atol,rtol)
    local node,odevars,vars,f,i,F,x0,pade_alpha,pade_beta,alpha,
        beta,alpha1,beta1,YY,Eqq,Eqs,varsNR,guess,sol,Err,nxpars,
        N,errL,Eqs1,jac,N1,k,j,bcindex,Jac,LErr;

    # Copyright: Dr. Venkat R.Subramanian and MAPLE group members at UW.
    # Original code is written by V. R. Subramanian (vsubram@uw.edu).
    # Last modified by Jerry Chen (jerrysyc@uw.edu) on 2017/08/11

    optimize;

    # Define the variables
    node:=nops(eqodes);
    odevars:=select(type,map(op,map(lhs,EqODEs)),'function');
    vars:=[seq(odevars[i]=Y||i[j],i=1..node)]; # i=vars, j=nodes

    # Find the derivatives
    f[1]:=Vector(node,[seq(rhs(eqodes[i]),i=1..node)]);

    for i from 2 to nder do
       f[i]:=simplify(subs(eqodes,Vector(node,[seq(diff(f[i-1][j],x),j=1..node)])));
    end do:

    for i from 1 to nder do
       F[i]:=unapply(subs(vars,f[i]),j,x);
    od:

    # Initialize the mesh
    x0:=Vector(nele+1,datatype=float[8]):
    x0[1]:=rng[1]:
    for i from 1 to nele do x0[i+1]:=x0[i]+hh[i]: od:

    # Pade coefficients
    pade_alpha:=unapply((m+n-ii)!*(m)!/((m+n)!*ii!*(m-ii)!),m,n,ii):
    pade_beta:=unapply((m+n-ii)!*(n)!/((m+n)!*ii!*(n-ii)!),m,n,ii):
```

```
for i from 1 to nder do
   alpha[i]:= pade_alpha(nder,nder,i);
   beta[i]:= alpha[i];
   beta1[i]:= pade_beta(nder-1,nder,i);
end do:

for i from 1 to nder-1 do
   alpha1[i]:=pade_alpha(nder-1,nder,i);
end do:
alpha1[nder]:=0;

# HDM formula
YY:=unapply(Vector(node,[seq(Y||i[j],i=1..node)]),j);
for i from 1 to nele do
   Eqq[i]:=eval(YY(i-1)-YY(i)+add(hh[i]^j*(alpha[j]*F[j](i-1,x0[i])
                  +(-1)^(j+1)*beta[j]*F[j](i,x0[i+1])),j=1..nder));
od;

Eqs:=evalf([seq(seq(Eqq[i][j],i=1..nele),j=1..node),op(subs(vars,j=0,x=rng[1],bc1)),
            op(subs(vars,j=nele,x=rng[2],bc2))]);

# vars for NR
varsNR:=[seq(seq(Y||j[i],i=0..nele),j=1..node),op(xpars)]:

# NR
nxpars:=nops(xpars);
N:=nops(Eqs);
vars:=[seq(varsNR[i]=y[i],i=1..N)];

# Local Error
LErr:=Vector([seq(evalf(LinearAlgebra:-Norm(subs(vars,add(hh[i]^j*((alpha[j]-alpha1[j])*F[j](i-1,x0[i])
             +(-1)^(j+1)*(beta[j]-beta1[j])*F[j](i,x0[i+1])),j=1..nder)))/sqrt(node)),i=1..nele)]);

# Create equation set
Eqs1:=subs(vars,Eqs);

# Create Jacobian
jac := Matrix(1 .. N,1 .. N,storage = sparse);
N1:=(N-nxpars)/(nele+1);
for k from 1 to N1 do
   for i from 1 to nele do
```

```
            for j from 1 to N1 do
               jac[i+(k-1)*nele,i+(j-1)*(nele+1)]:=diff(Eqs1[i+(k-1)*(nele)],y[i+(j-1)*(nele+1)]);
               jac[i+(k-1)*nele,i+1+(j-1)*(nele+1)]:=diff(Eqs1[i+(k-1)*(nele)],y[i+1+(j-1)*(nele+1)]);
            od:
         od:
      od:
      bcindex:=[seq(1+(i-1)*(nele+1),i=1..N1),seq(nele+1+(i-1)*(nele+1),i=1..N1)];
      for i from N1*nele+1 to N-nxpars do
         for j from 1 to nops(bcindex) do
            jac[i,bcindex[j]]:=diff(Eqs1[i],y[bcindex[j]]);
         od:
      od:
      for j from N-nxpars+1 to N do
         for i from 1 to N do
            jac[i,j]:=diff(Eqs1[i],y[j]):
            jac[j,i]:=diff(Eqs1[j],y[i]):
         od:
      od:

      [Eqs1,jac,LErr,node];

end proc:


HDMpars2 := proc (Gen,ic0,pars)
      local Jac,N,scale,vars,Eqs1,yold,jac,i,LL,j,F,k,errL,dy,ynew,rhsics,N1,bcindex,iter,Lerr,node;

      # Copyright: Dr. Venkat R.Subramanian and MAPLE group members at UW.
      # Original code is written by V. R. Subramanian (vsubram@uw.edu).
      # Last modified by Jerry Chen (jerrysyc@uw.edu) on 2017/08/11

      optimize;

      # Initialization
      N:=nops(Gen[1]);
      errL := 10;
      iter:=15;
      node:=Gen[4];

      F := unapply(`<,>`(seq(subs(op(pars),Gen[1][i]),i = 1 .. N)),y);
      Jac:=unapply(subs(op(pars),Gen[2]),y);
```

```
    # Scale based on the ics to force all y caculate from 0 to 1.
    rhsics:=ic0;      #map(rhs,ics);
    yold:=evalf(Vector([seq(rhsics[i],i=1..N)]));

    # Iterate until the y converges less then the tolerence
    for k to iter while errL > 1 do

      # Avoid version caused "SparseDirect" failed happens in Maple classic worksheet
      dy := LinearAlgebra:-LinearSolve(-Jac(yold),F(yold),method=SparseDirect);

      if add(`if`(type((dy[i]),numeric),0,1),i=1..N)>0 then ERROR("Cannot converge"); end if;
      ynew := yold + dy;
      errL := LinearAlgebra:-Norm(dy,2)/LinearAlgebra:-Norm(Vector([seq(atol+rtol*ynew[i],i=1..N)]),2);

      # update yold to next iteration
      yold := ynew;

    end do;

    if iter < k then ERROR("Exceed max NR iteration: %1",iter) end if;

    Lerr:=subs(seq(y[i]=ynew[i],i=1..N),op(pars),Gen[3]);

    # Return
    [ynew,Lerr]; # without name

end proc:
```