Improving performance when solving high order and mixed order boundary value problems in ODEs

W.H. Enright and Min Hu

Abstract. Solving high order or mixed order boundary value problems by general purpose software often requires the system to be first converted to a larger equivalent first order system. The cost of solving such problems is generally $O(m^3)$ where m is the dimension of the equivalent first order system. In this paper, we show how to reduce this cost by exploiting the special structure the 'equivalent' first order system inherits from the original associated mixed-order system. This technique applies to a broad class of boundary value methods. We illustrate the potential benefits by considering in detail a general purpose Runge-Kutta method and a multiple shooting method.

Key Words: Runge-Kutta methods, boundary value problems

AMS Subject Classifications: 65L05, 65L10

1. Introduction

Most of the analysis and development of numerical methods for ordinary differential equations are devoted to the solution of the first order system

$$\begin{aligned}
y_1' &= f_1(x, y_1, y_2, \ldots, y_m)\\
y_2' &= f_2(x, y_1, y_2, \ldots, y_m)\\
&\vdots\\
y_{m-1}' &= f_{m-1}(x, y_1, y_2, \ldots, y_m)\\
y_m' &= f_m(x, y_1, y_2, \ldots, y_m).
\end{aligned} \tag{1}$$

For dealing with high order ordinary differential equations, it is assumed that

$$u^{(m)} = f(x, u, u', \ldots, u^{(m-1)}), \tag{2}$$

This work was supported by Natural Science and Engineering Research Council of Canada and the Information Technology Research Center of Ontario. Author address: Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, M5S 1A4. E-mail: enright@cs.toronto.edu (W.H. Enright); minhu@cs.toronto.edu (M. Hu)

is first converted to an equivalent first order form,

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
&\vdots \\
y_{m-1}' &= y_m \\
y_m' &= f(x, y_1, y_2, \dots, y_m).
\end{aligned}
\tag{3}
$$

Similarly, for more general ODEs of mixed order

$$
u_i^{(m_i)} = f_i(x, u_1, u_1', \dots, u_1^{(m_1-1)}, \dots, u_d, u_d', \dots, u_d^{(m_d-1)}), \quad i = 1, \dots, d \tag{4}
$$

where $m_1 + m_2 + \ldots + m_d = m$, it is assumed that this system is first converted to the equivalent first order system

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
&\vdots \\
y_{m_1}' &= f_1(x, y_1, y_2, \dots, y_m) \\
&\vdots \\
y_{m-1}' &= y_m \\
y_m' &= f_d(x, y_1, y_2, \dots, y_m).
\end{aligned}
\tag{5}
$$

Note that although (4) and (5) have the same true solution, when a numerical method is applied to (5), one obtains an approximation to the lower order derivatives of the solution of (4) in addition to the approximation to the solution of (4). Typically, unless one is very careful about the choice of error control strategies, the stepsize constraint imposed by accuracy of the lower order derivatives will be more severe than the stepsize constraint imposed by the accuracy of the solution only.

In addition the cost of the numerical method can be very sensitive to the dimension m of the system and the conversion of (4) to (5) significantly increases the dimension of the problem. This is particularly true for stiff initial value problems and for boundary value problems.

In the paper we will consider the numerical solution of two point boundary value problems with separated boundary conditions in ODEs by numerical method that require the mixed order system (4) to be first converted to the first order system (5).

Such methods will partition the underlying interval $[x_0, x_F]$ into N subintervals $x_0 < x_1 < \cdots < x_N = x_F$. The approximations $y_0, y_1, \dots, y_N$, where $y \in R^m$, are then generated by solving a underlying nonlinear system of equations by a modified Newton method. The dominant cost for computing each Newton iteration is proportional to Nm^3 (This is the cost associated with approximating the Newton Jacobian of the system.

We will consider how this cost can be reduced by exploiting the special structure that the Newton Jacobian has. The special structure arises from the special form of (5). In particular, we will consider in detail an implicit Runge-Kutta BVP method ([2], [3]) and a multiple shooting method.

It must be acknowledged that there are software package (eg. COLSYS, [1]) that can handle (4) directly and do not require the conversion to a first order systems.

The purpose of the paper is to show that a rather minor modification to methods for first order system can make them more efficient when solving problems that have the special structure exhibited in (4). In the next section we will introduce the modification to the Runge-Kutta BVP method MIRKDC and in section 3 we will describe the modification to a multiple shooting method. Similar modifications could be made to other methods designed for first order system.

2. Matrix multiplication in MIRK methods

When a MIRK Runge-Kutta formula is applied to a boundary value problem ([2], [3]), we need to solve nonlinear equations of the form $\Phi(Y) = 0$ for $Y = (y_0, y_1, \ldots, y_N)^T$. The Newton Jacobian matrix for this equation has the form ([3]),

$$\begin{bmatrix} L_0 & R_0 & 0 & \cdots & 0 & 0 \\ 0 & L_1 & R_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & L_{N-1} & R_{N-1} \\ B_a & 0 & 0 & \cdots & 0 & B_b \end{bmatrix} \tag{6}$$

where

$$R_i = I - h_i \sum_{r=1}^{s} \omega_r \frac{\partial k_r}{\partial y_{i+1}}$$

and

$$L_i = -I - h_i \sum_{r=1}^{s} \omega_r \frac{\partial k_r}{\partial y_{i+1}}$$

To determine the matrices L_i and R_i for each iteration, matrix multiplications by the Jacobian matrix $\partial F/\partial y$ are involved in computing

$$\frac{\partial k_r}{\partial y_i} = \left[\frac{\partial F}{\partial y}\right]\left((1-\gamma_r)I + h_i \sum_{j=1}^{s} \beta_{rj} \frac{\partial k_j}{\partial y_i}\right), \quad r = 1, \ldots, s, \tag{7}$$

and

$$\frac{\partial k_r}{\partial y_{i+1}} = \left[\frac{\partial F}{\partial y}\right]\left(\gamma_r I + h_i \sum_{j=1}^{s} \beta_{rj} \frac{\partial k_j}{\partial y_{i+1}}\right), \quad r = 1, \ldots, s, \tag{8}$$

where $F(x, y)$ is the vector valued function of dimension m defined by the right hand hand side of (5). These multiplications usually consume a large fraction of the total computation costs. The operation counts for the determination of each L_i or R_i is then $m^3s + m^2s^2 + ms^2/2$ where the term m^3s is associated with the matrix multiplications while $s^2m^2 + ms^2/2$ corresponds to the determination of the intermediate matrices required in (7) and (8).

For ODEs of the form (2), we have $\dfrac{\partial F}{\partial y} = J$, where

$$J = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ \frac{\partial f}{\partial y_1} & \frac{\partial f}{\partial y_2} & \frac{\partial f}{\partial y_3} & \cdots & \frac{\partial f}{\partial y_{m-1}} & \frac{\partial f}{\partial y_m} \end{bmatrix}, \tag{9}$$

while for the mixed order boundary value problem (4), the matrix corresponding to $\dfrac{\partial F}{\partial y} = J$ is

$$J = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \frac{\partial f_1}{\partial y_3} & \cdots & \frac{\partial f_1}{\partial y_{m-1}} & \frac{\partial f_1}{\partial y_m} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ \frac{\partial f_d}{\partial y_1} & \frac{\partial f_d}{\partial y_2} & \frac{\partial f_d}{\partial y_3} & \cdots & \frac{\partial f_d}{\partial y_{m-1}} & \frac{\partial f_d}{\partial y_m} \end{bmatrix}. \tag{10}$$

If we take advantage of the form of (9), in computing $J \cdot A$ for any matrix $A = (a_{i,j})_{n\times n}$, we have that

$$\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ \times & \times & \times & \cdots & \times & \times \end{bmatrix} \cdot A = \begin{bmatrix} a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,m-1} & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,m-1} & a_{3,m} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & a_{m-1,3} & \cdots & a_{m-1,m-1} & a_{m-1,m} \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,m-1} & a_{m,m} \\ \times & \times & \times & \cdots & \times & \times \end{bmatrix}$$

For the first $m-1$ rows of the product, the entries are only the shifts of the original rows of A. The computation of the last row requires m^2 operations. Similarly, for the matrix in (10), the computation cost for the product is dm^2 operations. When $d = m$, the computation cost is

m^3 operations. That is to say, if we exploit this observation, the total cost of determining each L_i or R_i will be $dm^2s + m^2s^2 + ms^2/2$ operations instead of the original $m^3s + m^2s^2 + ms^2/2$ operations.

Suppose the cost for this set up is $\alpha_0\%$ of the total work. The maximum saving we can expect to achieve by adopting the above approach

$$\alpha_0\left[1 - \frac{dm^2s + m^2s^2 + ms^2/2}{m^3s + m^2s^2 + ms^2/2}\right]\% = \alpha_0\frac{m(m-d)}{m^2 + ms + s/2}\%. \tag{11}$$

For example, if we take $\alpha_0 = 60$, $s = 5, m = 8, d = 1$, the expected saving is 32%. In extensive testing and profiling of the MIRKDC program Muir [6] has observed that over a wide range of problems and accuracy requirements the time required to determine the L_i and R_i is generally close to 60% of the total time required to solve the problem.

3. Matrix multiplication in multiple shooting methods

The Newton Jacobian matrix for solving boundary value problems by multiple shooting codes ([1]) is of the form

$$\begin{bmatrix} -Y_1(x_2) & I & 0 & \cdots & 0 & 0 \\ 0 & -Y_2(x_3) & I & \cdots & 0 & 0 \\ 0 & 0 & -Y_3(x_4) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -Y_{N-1}(x_N) & I \\ B_a & 0 & 0 & \cdots & 0 & B_bY_N(b) \end{bmatrix} \tag{12}$$

where I is order m identity matrix and the $Y_i(x_{i+1})$ are solutions of the matrix IVP

$$Y_i' = \frac{\partial F}{\partial y}\, Y_i = J\, Y_i, \qquad Y_i(x_i) = I \quad \text{over } [x_i, x_{i+1}]. \tag{13}$$

To approximate $Y_i(x_{i+1})$ multiple shooting codes will generally apply an IVP method to (13) over the range $[x_i, x_{i+1}]$. The cost of this integration is generally dominated by the cost associated with evaluating the right hand side of (13) and this involves a matrix product of the same form as that considered in the previous section. The cost of computing each block, $Y_i(x_{i+1})$, using standard matrix multiplication is proportional to m^3 while the expected percentage savings if our new approach is adopted is

$$\alpha_1\left(1 - \frac{d}{m}\right)\%, \tag{14}$$

where $\alpha_1\%$ is the percentage of the total time involved in performing these matrix multiplications in the original multiple shooting method.

Table 1: Fourth order MIRKDC ($s = 3$)

		CPU time		Saving	
		Original	Modified	Expected	Actual
Problem 1		15.95	8.89	38%	44%
Problem 2	$\gamma = 0.0$	1.08	0.95	22%	12%
	$\gamma = 10.0$	9.50	8.00	22%	16%
Problem 3	$\epsilon = 1.0$	0.64	0.56	26%	13%
	$\epsilon = 0.1$	1.42	1.13	26%	20%

In preliminary experiments with a multiple shooting method (before it was modified to adopt our technique) we determined that over a range of problems α_1 was observed to vary between 45% and 55%. Taking $\alpha_1 = 50$ in (14) should therefore result in a reasonable estimate of the expected saving that can be realised when our approach is adopted in a multiple shooting code.

4. Numerical results

In this section, we present the numerical testing for the MIRKDC code [3] and a multiple shooting code based on BVPMS of the IMSL library ([4], [5]), by only modifying the matrix products associated with (7) (8) and with the numerical solution of (13) respectively. We have included three test problems to illustrate that the predicted saving can be realised through these minor changes. The computations were performed on a Sun Sparc 4 workstation. The tolerance for all problems is 10^{-5}. The statistics are given in tables 1 — 3 respectively, CPU time is in seconds. The expected saving is computed according to (11) and (14) with $\alpha_0 = 60$ and $\alpha_1 = 50$ respectively. Note that the multiple shooting method could not solve problem 2 for any of the prescribed values of the parameters γ. In all examples, the computed solutions agreed (to within round-off) with those produced by the unmodified methods.

Problem 1. (Linear problem) [7]

$$u^{(8)} = 914u^{(7)} - 12649u^{(5)} + 44136u^{(3)} - 32400u$$

the corresponding first order system form,

$$\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
&\vdots \\
y_7' &= y_8 \\
y_8' &= 914y_7 - 12649y_5 + 44136y_3 - 32400y_1
\end{aligned}$$

Table 2: Sixth order MIRKDC ($s = 5$)

		CPU time		Saving	
		Original	Modified	Expected	Actual
Problem 1		3.82	2.39	32%	34%
Problem 2	$\gamma = 0.0$	1.48	1.34	17%	10%
	$\gamma = 10.0$	12.75	10.64	17%	17%
Problem 3	$\epsilon = 1.0$	0.89	0.73	21%	18%
	$\epsilon = 0.1$	1.51	1.21	21%	20%

Problem 2. (Swirling flow I family) [1]

$$\begin{aligned} &f''' + 2ff'' - (f')^2 + g^2 = \gamma \\ &g'' + 2fg' - 2f'g = 0 \end{aligned} \qquad 0 < x < \infty$$

the corresponding first order system form,

$$\begin{aligned} &y_1' = y_2, \quad y_2' = y_3, \quad y_3' = \gamma^2 - 2y_1y_3 + y_2^2 - y_4^2, \\ &y_4' = y_5, \quad y_5' = 2y_2y_4 - 2y_1y_5, \end{aligned}$$

$$\begin{aligned} &y_1(0) = y_2(0) = 0, \quad y_4(0) = 1, \\ &y_2(\infty) = 0, \qquad y_4(\infty) = \gamma. \end{aligned}$$

No closed form solution exists in general.

Problem 3. (Swirling flow III family) [1]

$$\begin{aligned} &\epsilon f'''' + ff''' + gg' = 0, \\ &\epsilon g'' + fg' - f'g = 0, \end{aligned} \qquad 0 < x < 1$$

with

$$\begin{aligned} &f(0) = f(1) = f'(0) = f'(1) = 0, \\ &g(0) = \Omega_0, \qquad g(1) = \Omega_1. \end{aligned}$$

the corresponding first order system form,

$$\begin{aligned} &y_1' = y_2, \quad \epsilon y_2' = y_1y_4 - y_2y_3, \quad y_3' = y_4, \\ &y_4' = y_5, \quad y_5' = y_6, \quad y_6' = -y_3y_6 - y_1y_2, \end{aligned}$$

Table 3: Multiple shooting code

		CPU time		Saving	
		Original	Modified	Expected	Actual
Problem 1		2.72	1.70	44%	38%
Problem 2	$\gamma = 0.0$				
	$\gamma = 10.0$				
Problem 3	$\epsilon = 1.0$	0.98	0.70	33%	29%
	$\epsilon = 0.1$	2.18	1.55	33%	29%

$$y_3(0) = y_4(0) = y_3(1) = y_4(1) = 0,$$
$$y_1(0) = \Omega_0, \qquad y_1(1) = \Omega_1.$$

No closed form solution exists in general.

Conclusions:

We have shown that minor modification to existing boundary value software (developed for first order systems) can lead to significant and predictable reductions in computing time when the problem being solved was originally a mixed order system. These modifications are easy to implement and apply to mixed order systems with no restriction on the maximum order.

References

[1] U. Ascher, R.M.M. Mattheij and R. Russell, *Numerical solutions of boundary value problems for ordinary differential equations*, Prentice Hall, New Jersey, 1988.

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

[3] W.H. Enright and P.H. Muir, *A Runge-Kutta type boundary value ODE solver with defect control*, Rep. 267, Dept. of Computer Science, Univ. of Toronto, Canada, 1993.

[4] W.H. Enright and T.F.Fairgrieve, *Performance of the boundary value solver BVPMS*, Contributed paper presented at the first IMSL users meeting, Orlando, February 1988.

[5] *IMSL User's Manual.* International Mathematical and Statistical Libraries, Houston, 1987.

[6] P.H. Muir, *Private communication*, 1994.

[7] M.R. Scott and H.A. Watts, *SUPORT - a computer code for two-point boundary value problems via orthonormalization*, Sandia Labs Report 75-0198, Albuquerque, New Mexico, June 1975.