ELSEVIER

Journal of Computational and Applied Mathematics 120 (2000) 109–131

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Some comments on DAE theory for IRK methods and trajectory optimization

Neil Biehn[a], Stephen L. Campbell[b, *, 1], Laurent Jay[c], Tracey Westbrook[b]

[a] *Operations Research Program, North Carolina State University, Raleigh, NC 27695-7913, USA*

[b] *Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205, USA*

[c] *Department of Mathematics, 14 MacLean Hall, The University of Iowa, Iowa City, IA 52242-1419, USA*

Received 5 January 1999; received in revised form 12 November 1999

Abstract

It has been observed elsewhere in the literature that the activation of constraints in a trajectory optimization problem can lead to higher index DAEs. Several existing codes can handle a number of these constrained problems. In this note we will discuss why the situation is more complex than just saying a higher index DAE occurs. The discussion is in the context of a specific industrial code SOCS but the observations made here have relevance for a number of methods and have implications for what types of test problems a code should be tested on. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Implicit Runge–Kutta; Optimization; Differential algebraic equation

1. Introduction

A trajectory optimization problem typically takes the form

$$\min J(u) \tag{1a}$$

$$F(x', x, u, t) = 0, \tag{1b}$$

$$C(x, u, t) \geqslant 0, \tag{1c}$$

$$\text{Boundary Information.} \tag{1d}$$

* Corresponding author.

E-mail addresses: slc@math.ncsu.edu (S.L. Campbell), ljay@math.uiowa.edu, na.ljay@na-net.ornl.gov (L. Jay).

[1] Research supported in part by the National Science Foundation under DMS-9714811 and DMS-9802259.

PII: S0377-0427(00)00306-X

Here J is the quantity to be minimized, the cost function, u is control, (1b) are the dynamics equations and any equality constraints that hold for all (t,x), and (1c) are inequality path constraints.

Many modeling processes, such as those in constrained mechanics naturally lead to a model which is a system of differential algebraic equations (DAE) in (1b). See for example, [4–7,12,17]. However, even if (1b) is a system of ordinary differential equations (ODEs), DAEs will result when some of the constraints in (1c) are active. It is known that many numerical discretization schemes, such as the IRK methods discussed here, fail to converge or exhibit an order reduction when applied to DAEs. This order reduction depends not only on the index of the particular DAE but also on their structure. Having said this, a number of software packages have been developed and successfully applied to some of the optimization problems that arise in applications.

However, the situation is much more complex than merely stating that there is an order reduction and the index of the DAE is some number. In this paper we shall examine the significance of the presence of DAEs more carefully. Our discussion will be within the context of a particular industrial optimization package SOCS. However, the implications are of more general interest.

Section 2 will summarize some basic information about SOCS and describe the discretizations that it uses. Section 3 will develop some basic facts about these discretizations. Some of the order results in this section are known and some have not appeared elsewhere in the literature. Section 4 will discuss some preliminary implications of the analysis in Section 3. Academic test problems illustrating these ideas appear in Section 5. These test examples will show that the order estimates, even though they are sharp, do not tell the full story. Section 6 will discuss these differences. Related discussions that address different issues can be found in [1,8,9,15,16]. Finally, this paper is part of a larger investigation concerning the optimization of machine tool paths. In this application high precision is necessary. This is reflected in our choice of stopping tolerances.

2. SOCS

Sparse optimal control software (SOCS) is a software package developed at Boeing for the solution of optimal control problems that arise in industry [2,3]. SOCS can also solve related problems such as parameter identification. SOCS discretizes the dynamics and then solves the resulting nonlinear programming program (NLP) with a sequential quadratic programming (SQP) based algorithm. SOCS starts with an initial discretization mesh and then automatically generates finer and finer meshes until sufficient accuracy is obtained. The SOCS mesh refinement strategy is based on [3].

In our discussions it will be important to keep in mind what the goal of SOCS is. At a given grid level, the discrete problem is optimized. The termination of the grid refinement iteration, and hence of the overall calculation, is via a tolerance for an error in integrating the dynamics and satisfying the constraints. Thus the emphasis is not on the accuracy of the estimation of some theoretically optimal control. Rather, the emphasis is on assuring that the computed control will produce a trajectory and cost in the real-problem close to that of the computed trajectory and cost. Thus assuming our modeling equations are good, using the computed control in the actual process will result in a trajectory that will satisfy the constraints to high accuracy and provide close to the computed cost. This computed control may not be a "good" estimate of the optimal one. This is not a problem in many industrial examples for two reasons. One is that some of the parameters in the cost are picked for design purposes and could easily be changed somewhat. Secondly, for complex problems the

goal is to produce a control that does significantly better than current practice. We shall comment more on these ideas later.

SOCS currently uses two discretization schemes. On the initial course mesh it uses an integrator based on the trapezoidal rule. We shall refer to this as TR. Later as the mesh gets finer, SOCS switches to a higher order method based on Hermite–Simpson interpolation which we shall refer to as HS. Both of these methods are mathematically equivalent to implicit Runge–Kutta (IRK) methods known as the two- and three-stage Lobatto IIIA methods. There are other options in SOCS which will not be discussed here.

3. The IRK discretizations

A general s-stage IRK method is defined by its coefficients vectors $b = (b_i)_{i=1}^s$, $c = (c_i)_{i=1}^s$ and matrix $\mathscr{A} = (a_{ij})_{i,j=1}^s$. These coefficients are often written in a so-called Butcher tableau

$$\begin{array}{c|c} c & \mathscr{A} \\ \hline & b^{\mathrm{T}} \end{array}.$$

When used as an initial value solver, the method applied to the DAE

$$F(x', x, t) = 0 \tag{2}$$

proceeds as follows.

Let the discretization times be t_n. The estimate for $x(t)$ at time t_n is denoted x_n. Given x_{n-1} at t_{n-1} we solve for x_n at $t_n = t_{n-1} + h_{n-1}$ by solving the system

$$F\left(X_i', x_{n-1} + h_{n-1}\sum_{j=1}^{s} a_{ij}X_j', t_{n-1} + c_i h_{n-1}\right) = 0, \quad i = 1, \ldots, s \tag{3}$$

for the *stage derivatives* X_i'. If there are m equations in (2), then (3) consists of $s \cdot m$ equations in the s vector unknowns X_i'. Given the X_i', the value of x_n is given by

$$x_n = x_{n-1} + h_{n-1}\sum_{i=1}^{s} b_i X_i'.$$

For TR, HS we have $s = 2, 3$, respectively, and the Butcher tableaux are

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}, \qquad \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{5}{24} & \frac{1}{3} & -\frac{1}{24} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}.$$

In optimizing $J(u)$ with the constraint consisting of only the dynamics $x' = f(x, u, t)$ the control variable u is free and it may be natural to consider the possibility of discretizing u differently than the state variable x. However, if state constraints are active, then the process is a DAE and part

of u is no longer free. This raises the possibility that the numerical method on the DAE has been altered by the fact that the control u was thought of differently.

In a direct application of an IRK method the control is treated the same as the state variable x. For example,

$$F(x', x, u, t) = 0 \tag{4}$$

can be discretized using TR by solving the following set of equations:

$$F(X_1', x_{n-1}, u_{n-1}, t_{n-1}) = 0, \tag{5a}$$

$$F\left(X_2', x_{n-1} + \frac{h_{n-1}}{2}(X_1' + X_2'), u_{n-1} + \frac{h_{n-1}}{2}(U_1' + U_2'), t_n\right) = 0 \tag{5b}$$

with the updates

$$x_n = x_{n-1} + \frac{h_{n-1}}{2}(X_1' + X_2'), \tag{6a}$$

$$u_n = u_{n-1} + \frac{h_{n-1}}{2}(U_1' + U_2'). \tag{6b}$$

Note that when viewed as an ODE solver one has to solve (5) and then use (6) whereas a boundary value problem (BVP) approach has both (5) and (6) available at the same time.

SOCS does not, at this time, directly apply the IRK formulation but rather approaches the IRK discretization as a collocation. It also assumes that the system of DAEs is semi-explicit:

$$x' = f(x, u, t), \tag{7a}$$

$$0 = g(x, u, t). \tag{7b}$$

For TR, SOCS tries to minimize the cost and satisfy the constraints

$$x_n - x_{n-1} - \frac{h_{n-1}}{2}(f(x_n, u_n, t_n) + f(x_{n-1}, u_{n-1}, t_{n-1})) = 0 \tag{8a}$$

along with

$$g(x_n, u_n, t_n) = 0 \tag{8b}$$

for $n = 1, 2, \ldots$.

In contrast TR, viewed as an IRK, when applied to (7) gives the set of equations:

$$X_1' = f(x_{n-1}, u_{n-1}, t_{n-1}), \tag{9a}$$

$$0 = g(x_{n-1}, u_{n-1}, t_{n-1}), \tag{9b}$$

$$X_2' = f\left(x_{n-1} + \frac{h_{n-1}}{2}(X_1' + X_2'), u_{n-1} + \frac{h_{n-1}}{2}(U_1' + U_2'), t_n\right), \tag{9c}$$

$$0 = g\left(x_{n-1} + \frac{h_{n-1}}{2}(X_1' + X_2'), u_{n-1} + \frac{h_{n-1}}{2}(U_1' + U_2'), t_n\right), \tag{9d}$$

$$x_n = x_{n-1} + \frac{h_{n-1}}{2}(X_1' + X_2'), \tag{9e}$$

$$u_n = u_{n-1} + \frac{h_{n-1}}{2}(U_1' + U_2'). \tag{9f}$$

Note that if (9e) and (9f) are substituted into (9c) and (9d) and then (9a) and (9c) are then used to eliminate X_1', X_2' in (9e), we wind up with (8). Thus, the IRK representation is mathematically equivalent to the collocation form. However, it has some redundant variables. Note that U_1', U_2' appear only in the expression $u_{n-1} + (h_{n-1}/2)(U_1' + U_2')$ which is equal to u_n. This does not alter our observations but it could impact on actually using the IRK formulations. Similar comments hold for HS [1]. SOCS implements HS as a collocation and not directly as an IRK.

It is known that DAEs pose problems for IRK methods and a reduction of the order of convergence may occur. Most of the existing theory deals with DAEs in Hessenberg form. For index two Hessenberg DAEs

$$y' = f(y, u), \tag{10a}$$

$$0 = g(y) \tag{10b}$$

with $g_y f_u$ invertible in a neighborhood of the solution it is known that:

Theorem 1 (Hairer and Wanner [20] and Jay [21]). *Suppose that the s-stage Lobatto IIIA method is being applied to the index two Hessenberg DAE* (10). *Then for consistent initial conditions at* t_0 *the local error can be estimated by*

$$y_1 - y(t_0 + h_0) = \mathrm{O}(h_0^{2s-1}),$$

$$u_1 - u(t_0 + h_0) = \mathrm{O}(h_0^{s}).$$

The global error is

$$y_n - y(t_n) = \mathrm{O}(h^{2s-2}),$$

$$u_n - u(t_n) = \mathrm{O}(h^{s-1}),$$

where $h := \max_n h_n$. *For s even and fixed stepsizes* $h_n = h$ *this last estimate can be improved to* $u_n - u(t_n) = \mathrm{O}(h^s)$.

As a corollary we obtain:

Theorem 2. *Given consistent initial values at* t_0 *for the index two Hessenberg DAE* (10) *we have*

(1) *For TR the local error is* $y_1 - y(t_0 + h_0) = \mathrm{O}(h_0^3), u_1 - u(t_0 + h_0) = \mathrm{O}(h_0^2)$ *and the global error is* $y_n - y(t_n) = \mathrm{O}(h^2), u_n - u(t_n) = \mathrm{O}(h)$ *where* $h := \max_n h_n$. *For fixed stepsizes* $h_n = h$ *we have* $u_n - u(t_n) = \mathrm{O}(h^2)$.
(2) *For HS the local error is* $y_1 - y(t_0 + h_0) = \mathrm{O}(h_0^5), u_1 - u(t_0 + h_0) = \mathrm{O}(h_0^3)$ *and the global error is* $y_n - y(t_n) = \mathrm{O}(h^4), u_n - u(t_n) = \mathrm{O}(h^2)$ *where* $h := \max_n h_n$.

We shall see that we also need results for index three Hessenberg systems

$$y' = f(y,z), \tag{11a}$$

$$z' = k(y,z,u), \tag{11b}$$

$$0 = g(y), \tag{11c}$$

where $g_y f_z k_u$ is invertible in a neighborhood of the solution. The following next two results have not appeared in the literature. Their proof is rather long and technical and is outlined in the appendix.

Theorem 3. *Suppose that the s-stage Lobatto IIIA method is being applied to the index three Hessenberg DAE* (11). *Then for consistent initial conditions at* t_0 *the local error can be estimated by*

$$y_1 - y(t_0 + h_0) = \mathrm{O}(h_0^{\max(s+1,2s-3)}), \tag{12a}$$

$$z_1 - z(t_0 + h_0) = \mathrm{O}(h_0^{s}), \tag{12b}$$

$$u_1 - u(t_0 + h_0) = \mathrm{O}(h_0^{s-1}). \tag{12c}$$

For $s \geqslant 4$ *the global error is*

$$y_n - y(t_n) = \mathrm{O}(h^{2s-5}), \tag{13a}$$

$$z_n - z(t_n) = \mathrm{O}(h^{s-1}), \tag{13b}$$

$$u_n - u(t_n) = \mathrm{O}(h^{s-3}), \tag{13c}$$

where $h := \max_n h_n$. *For* $s = 2,3$ (TR, HS) *there is generally no global convergence.*

One important case is when u, which is often the control, enters linearly in (11b).

Theorem 4. *In addition to the assumptions of Theorem* 3 *suppose that* (11*b*) *is linear in u. Then the assumption* $s \geqslant 4$ *can be relaxed to* $s \geqslant 3$ *and the global error in y given by* (13*a*) *can be improved to* $y_n - y(t_n) = \mathrm{O}(h^{2s-4})$. *In this case, for HS the global errors in* y,z,u *are* $\mathrm{O}(h^2), \mathrm{O}(h^2), \mathrm{O}(1)$, *respectively.*

Numerical experiments indicate that the results in Theorems 1–4 are sharp. In fact, the conclusions of Theorems 1–4 also hold for time-varying systems. Note that there is generally no global convergence of TR (the case $s = 2$) when applied to index three Hessenberg DAEs.

One might then conclude that it would not make sense to use HS on a problem which might be index three along a portion of the trajectory since the control would not be computed. However, this need not be the case for a code which uses strategies similar to those in SOCS. We begin to examine this question in Sections 5 and 6. First, we will examine the linear constant coefficient case more carefully since these results will be used later in our discussion of the numerical examples.

Table 1
Local error of TR method applied to $Nx' + x = f(t)$ with index 4

	x_1	x_2	x_3	x_4
h^5	$\frac{1}{120}(f_2^{(6)} - f_3^{(7)} + f_4^{(8)})$	$\frac{1}{120}(f_3^{(6)} - f_4^{(7)})$	$\frac{1}{120} f_4^{(6)}$	0
h^4	$\frac{1}{24}(f_4^{(7)} - f_3^{(6)}) + \frac{1}{40} f_2^{(5)}$	$\frac{-1}{24} f_4^{(6)} + \frac{1}{40} f_3^{(5)}$	$\frac{1}{40} f_4^{(5)}$	0
h^3	$\frac{1}{6} f_4^{(6)} + \frac{-2}{15} f_3^{(5)} + \frac{1}{12} f_2^{(4)}$	$\frac{1}{12} f_3^{(4)} - \frac{2}{15} f_4^{(5)}$	$\frac{1}{12} f_4^{(4)}$	0
h^2	$\frac{13}{30} f_4^{(5)} + \frac{1}{6} f_2^{(3)} - \frac{1}{3} f_3^{(4)}$	$\frac{-1}{3} f_4^{(4)} + \frac{1}{6} f_3^{(3)}$	$\frac{1}{6} f_4^{(3)}$	0
h	$\frac{1}{3}(2f_4^{(4)} - f_3^{(3)})$	$\frac{-1}{3} f_4^{(3)}$	0	0
1	$\frac{2}{3} f_4^{(3)}$	0	0	0

It is well known [10] that through linear time invariant coordinate changes one may reduce the solvable system

$$Ex' + Fx = g(t) \tag{14}$$

with E, F constant matrices to a decoupled system consisting of ODEs and systems of the form

$$Nx' + x = f(t), \tag{15}$$

where N is an elementary nilpotent Jordan block. The local and global error for a numerical scheme applied to (14) are the minimum of the local and global error for the subsystems. The most common situations in optimization when constraints are active are index two, three, or four systems. For the index two system (15) is

$$x_2' + x_1 = f_1(t), \tag{16a}$$

$$x_2 = f_2(t). \tag{16b}$$

The solution of $Nx' + x = f(t)$ is

$$x = \sum_{i=0}^{k-1} (-N)^i f^{(i)}(t).$$

In the case of (16) we get $x_2 = f_2(t)$, $x_1 = f_1(t) - f_2'(t)$. Note that (16) is not a system in the form of (10) since there is no dependence on t in (10). This is not important from a general viewpoint of method order but we will see that it can lead to an overestimate of the order reduction which is important in understanding particular problems.

Using MAPLE one can compute the local error for each state variable. In Tables 1 and 2 the x_i are the components of x and the f_i are the components of f. In the local error calculation the derivatives are evaluated at the grid point and the exact error is given. The index three case is given by deleting the x_1 column and reducing all subscripts by 1. Similarly, the index two case is given by deleting the x_1, x_2 columns and reducing all subscripts by 2. There are two key things to note about Tables 1 and 2. One is that the order reduction in the local error depends on whether certain derivatives of $f_3(t), f_4(t)$ vanish. This will be commented on later. The second is that unlike the previous theorems the tables do not give error estimates but are the exact error coefficients. Thus they not only give order bounds on the error but, at least for the special case in which they apply, they also guarantee a certain size to the local error.

Table 2
Local error of HS method applied to $Nx' + x = f(t)$ with index 4

	x_1	x_2	x_3	x_4
h^5	$\frac{1}{120}(f_4^{(8)} - f_3^{(7)} + f_2^{(6)})$	$\frac{1}{120}(f_3^{(6)} - f_4^{(7)})$	$\frac{1}{120} f_4^{(6)}$	0
h^4	$\frac{1}{24}(f_4^{(7)} - f_3^{(6)}) + \frac{1}{96} f_2^{(5)}$	$\frac{-1}{24} f_4^{(6)} + \frac{1}{96} f_3^{(5)}$	$\frac{1}{96} f_4^{(5)}$	0
h^3	$\frac{1}{48} f_2^{(4)} - \frac{19}{240} f_3^{(5)} + \frac{1}{6} f_4^{(6)}$	$\frac{1}{48} f_3^{(4)} - \frac{19}{240} f_4^{(5)}$	$\frac{1}{48} f_4^{(4)}$	0
h^2	$\frac{-1}{8} f_3^{(4)} + \frac{7}{20} f_4^{(5)}$	$\frac{-1}{8} f_4^{(4)}$	0	0
h	$\frac{1}{2} f_4^{(4)}$	0	0	0
1	0	0	0	0

This is the local error assuming exact function evaluations. Understanding the global error is more complex. Part of the global error is given by the accumulation of the exact local errors given in the previous two tables. In general, the order results for time-varying DAEs of indices 2 and 3 are given by Theorems 1–4. However, sometimes a better order is possible. To see when this occurs it suffices to consider (15) with N a Jordan block. Suppose that N is an index three Jordan block. We consider constant stepsizes $h_n = h$. Again using MAPLE on the HS equations we see that in the recursion for the x_3 variable we get a formula with h^{-1} in it. Thus, ignoring function evaluation and roundoff errors, the error recursion for x_3 at step n takes the form

$$\varepsilon_{G,n}^{x_3} = \varepsilon_{G,n-1}^{x_3} + \varepsilon_{L,n}^{x_3} + \frac{\varepsilon_{G,n-1}^{x}}{h}, \tag{17}$$

where $\varepsilon_{G/L,n}^{x_i}$ is the global (G) or local (L) error for the components x_i at step n. If ε_G^x is $O(h^2)$, then the far right term in (17) is an $O(h)$ term which is added on each time step. For fixed stepsizes, assuming $t_{\text{end}} - t_0 = O(1)$ we have $O(1/h)$ steps and the global error in x_3 is $O(1)$ as stated in Theorem 4 for u. The argument is more complicated but the same conclusion holds for variable stepsizes. However, as shown in Table 1, it is possible for x to have global order higher than 2 provided the forcing function f is a low degree polynomial. In this case there will be some convergence in x_3. This effect is not very important in most problems where f is an input or forcing function since most forcing terms are not low degree polynomials. However, it can be a factor in our setting here where f may come from a constraint. Linear constraints are common in applications. Thus this effect could be important in choosing good test problems.

There is another error component that is potentially equally important. We shall loosely call this evaluation error. This arises because the functions are not evaluated exactly. In fact, in practice they are often the results of interpolated table lookup. Also, the solution of the nonlinear integration equations is not done exactly so that there is numerical error, for example, in satisfying (8a). These two types of errors enter in different ways but have similar effect in terms of error analysis. There is another way in which they are quite different philosophically and we will discuss that shortly. Looking at (8a) we can think of the error in satisfying the constraints as adding an ε to the right-hand side. On the other hand, an error ε in evaluating f would add an $h\varepsilon$ to (8a). In the solving of the IRK equations there is a division by a power of h. For HS this is h^2 in some components. Thus to the local error we need to add a term $\tilde{M}\varepsilon h^{-k}$ where the value of k depends on the index, how ε

enters the equations, and which variables are considered. This additional local error term will also accumulate from one step to another.

Regardless of the source of evaluation error, the global error for the methods considered here when applied to system (15) is of the form

$$Mh^r + \tilde{M}\frac{\varepsilon}{h^k} + [\text{Error in initial conditions}]. \tag{18}$$

The value of r depends on the method, the index, and which variables are considered while the value of k depends on the index, which variables are considered, and what type of evaluation error ε is. Thus, it is possible to get extra order reduction when the $\tilde{M}\varepsilon h^{-k}$ term starts to become significant.

4. Implications for trajectory optimization

Many physical systems are modeled by a second-order system $x'' = f(x', x, u, t)$. Even when the original model is unconstrained, constraints may become active during the optimization process. When constraints are active we have a DAE and all or part of the control u becomes determined and should thus be considered as part of the state variable. A velocity constraint $g(x', x, t) = 0$ leads to DAEs of index at least two while a position constraint $g(x, t) = 0$ leads to DAEs of index at least three. In most complex systems the control u will only appear in some of the dynamics equations. If the constraints are in terms of variables x_i for which the x_i'' equation does not involve u, then the index can be even strictly higher than three [11].

We have seen that one effect of the existence of the system being higher index is that there can be order reduction in the discretization. What are the effects of this order reduction on the optimization? One effect, reduced order in computation of the controls has been discussed elsewhere. We will also discuss this shortly but first we wish to examine what the effect is on computational effort.

The order parameter p appears in a key way in the mesh refinement strategy of SOCS. This strategy is described in detail in [3]. Basically, the strategy is as follows. The user specifies a local error tolerance ε_{TOL} which will terminate the optimization. Initially, a uniform grid with a user specified number of grid points is used. At each iteration the number of grid points is increased from N grid points to between $1.2N$ and $2N - 1$ grid points. On each grid update between 0 and 5 uniformly spaced grid points are added between each pair of grid points and a grid point is added at a linearly interpolated estimate of any events. The actual number of grid points chosen and the selection of which subintervals to place them is calculated by a discrete optimization problem. This minimization problem tries to minimize the new integration error after the grid points are added with the restriction that there is no need to add additional points in a subinterval once the local truncation error for that subinterval is estimated to be safely below ε_{TOL}. This current error is computed by an integration of the dynamics not utilizing constraints with a mesh that halves the current mesh. The order parameter p appears in the estimate of the effect of adding additional meshpoints to any subinterval.

What is the effect of a wrong estimate of the order on the SOCS code? Suppose that a constraint is active on part of an interval. If the actual order is lower than the order parameter of SOCS, it will overestimate how much effect a stepsize reduction will have on the error when the system is a DAE. Thus it will underestimate how much to subdivide the interval to reduce the error. The code may work but with reduced efficiency since it will put fewer points than needed into regions where order

reduction occurs. This is the problem of making suboptimal mesh selections. Eventually, mesh points go where they are needed but the code may put off placing them where most needed. Since the NLP problem size grows with the size of the mesh this can increase significantly the computational effort.

There is another effect that can be even more important. Meshpoints are only added if they are needed to reach the termination criterion which for SOCS is based on an estimate of the local truncation error. Thus, one typically sees that early on, SOCS adds a maximum number of $N-1$ points and then adds a lower percentage of points towards the end. If the order is lower than the order parameter of SOCS, SOCS may add too few points and will not add them where they are most needed. This can lead to a series of extra iterations with grids which were not increased enough precisely when the grid is large and the iterations are the most expensive. We will see this effect on the examples to follow.

The theoretical analysis shows that the order reduction can be less for some simpler problems with simple forcing functions. This might sound like a desirable effect, but it means that when testing a code on simpler problems, say with bound constraints, one might not see the effect of order reduction due to reduced r in (18) and thereby get a misleading interpretation of code performance.

There is also the question of order reduction due to the second term in (18). This is a more difficult problem since a smaller stepsize may not improve things. Working with this term is not easy. Many codes, such as SOCS, have a number of tolerances with interlinked limits on how they can be set. Resetting these tolerances in the presence of reduced order due to $k > 0$ can be a difficult and code dependent task.

As seen from above there is a reduced accuracy in the computation of the control u. In fact, on some constraints, it is possible for the approximation to only be an O(1) approximation of the optimal solution. Surprisingly this need not always be as bad as it sounds. This is due to the philosophy behind the termination strategy of SOCS and will be discussed in greater detail after presenting some computational examples.

5. Computational examples

To illustrate how the previous observations can actually affect the performance of an optimization package we will consider the following simple problem which has the same structure as many problems in mechanics:

$$J(u) = \frac{1}{2}\int_0^3 x^2 + qu^2 \,\mathrm{d}t, \tag{19a}$$

$$x'' = u, \tag{19b}$$

$$x \geqslant g(t), \tag{19c}$$

$$x(0) = 5, \tag{19d}$$

$$x'(0) = 0. \tag{19e}$$

Table 3
Standard mesh refinement, g_4 constraint

GRID	NPT	NFG	NFE	NRHS	ERRODE	ERROBJ	CPU
1	10	15	372	7068	0.29E − 02	0.47E − 06	0.70
2	19	4	74	2738	0.18E − 03	0.59E − 06	0.35
3	37	4	74	5402	0.50E − 04	0.30E − 05	0.69
4	73	4	74	10730	0.20E − 05	0.27E − 07	1.9
5	127	4	74	18722	0.67E − 06	0.25E − 09	4.6
6	152	3	59	17877	0.18E − 06	0.28E − 09	4.5
7	182	3	59	21417	0.15E − 06	0.40E − 09	6.4
8	218	3	59	25665	0.44E − 07	0.67E − 10	7.6
	218	40	845	109619			26.80

This example, simple though it is, turns out to be capable of illustrating a number of facts that are important. Note that when (19c) is active, (19b), (19c) is an index three DAE in (x, u). If the factor q is too large, then the constraint need not be active. We take $q = 10^{-3}$ which is typical of q values when this term in the cost is used to regularize the calculations. We consider three possible constraints $g(t)$ initially. The first one is no constraint ($g(t) = -\infty$). The other two are

$$g_2(t) = 16 - 4(t-4)^2, \quad g_4(t) = 15 - (t-4)^4.$$

The functions g_2 and g_4 are defined so that they are of comparable size and are active for approximately the same length of time.

To simplify our discussion HS is used at every iteration and 10 initial grid points are chosen uniformly. NPT is the number of grid points. The number of function and gradient calls were the same after the first iteration and are called NFG (these are the NFC and NGC of SOCS). NFE is the number of function evaluations. After the first iteration there was always one Hessian call. NRHS is the number of right-hand sides evaluated. ERRODE is the computed error in the integration and it is used as the stopping criterion. This is consistent with the SOCS strategy of emphasizing accurately computing what the candidate control does as opposed to worrying about how close it is to the optimal control. CPU time is in seconds.

Table 3 gives the results from SOCS on this problem.

Fig. 1 shows the control. The small jump at the right end is an anomaly due to the value of u at the endpoint not being fully constrained by the dynamics. Fig. 2 shows the state trajectory and the constraint. The constraint is active for approximately $2.2 \leqslant t \leqslant 3$.

The standard SOCS strategy is based on the order parameter being equal to $p = 4$. We reset this parameter to $p = 2$. The result is given in Table 4.

Table 4 shows reduced iterations and CPU time. This improvement is because of the DAE behavior giving reduced order and not to this just being a better choice. To illustrate this we consider the no constraint case. Tables 5 and 6 show that there is no improvement in the modified case for the unconstrained problem and in fact the grid is slightly larger for the order parameter set to $p = 2$. Note that in Table 3 there are several iterations starting with the fifth iteration where the termination criterion is not met and yet a minimal number of points are being added and those that are added are not added where they will be most needed.

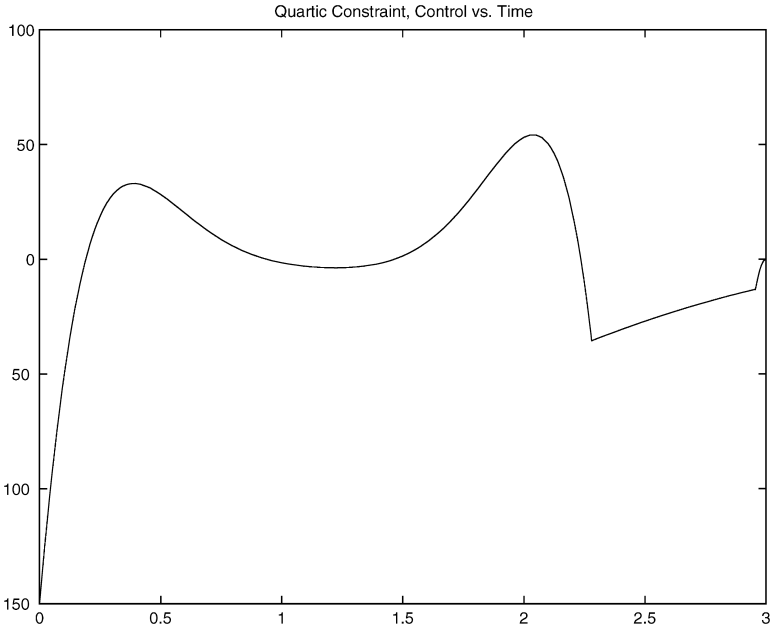

Fig. 1. Control with g_4 constraint.

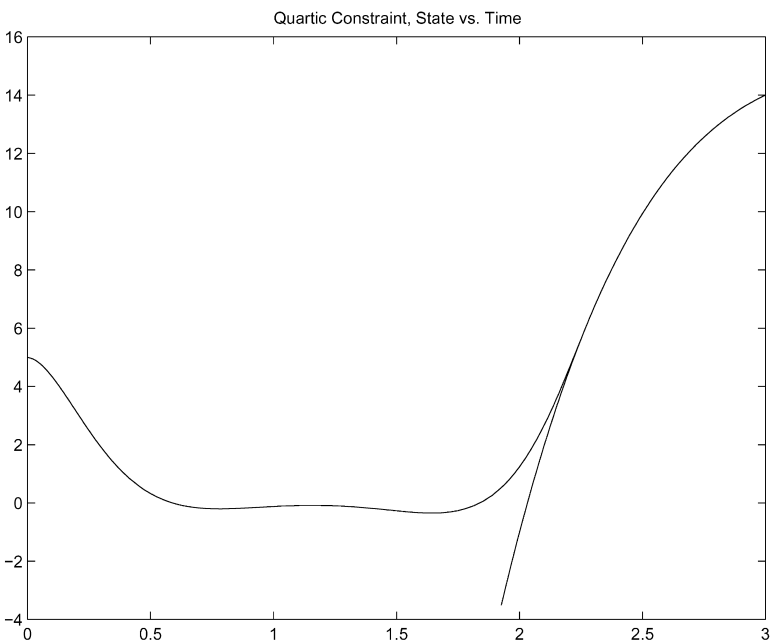

Fig. 2. State with g_4 constraint.

Table 4
$p = 2$ mesh refinement, g_4 constraint

GRID	NPT	NFG	NFE	NRHS	ERRODE	ERROBJ	CPU
1	10	15	372	7068	0.29E − 02	0.47E − 06	0.78
2	19	4	74	2738	0.22E − 03	0.59E − 06	0.39
3	37	4	74	5402	0.30E − 04	0.11E − 05	0.89
4	73	4	74	10730	0.49E − 05	0.26E − 06	2.1
5	137	4	74	20202	0.70E − 06	0.89E − 10	8.2
6	164	3	59	19293	0.83E − 07	0.57E − 09	7.3
	164	34	727	65433			19.63

Table 5
Standard mesh refinement, no constraint

GRID	NPT	NFG	NFE	NRHS	ERRODE	ERROBJ	CPU
1	10	11	172	3268	0.89E − 02	0.58E − 13	0.41
2	19	4	74	2738	0.15E − 04	0.32E − 12	0.24
3	37	4	74	5402	0.10E − 05	0.15E − 14	0.34
4	65	4	74	9546	0.17E − 06	0.94E − 14	0.51
5	78	3	59	9145	0.29E − 06	0.11E − 12	0.50
6	93	3	59	10915	0.70E − 07	0.25E − 13	0.58
	93	29	512	41014			2.59

Table 6
$p = 2$ mesh refinement, no constraint

GRID	NPT	NFG	NFE	NRHS	ERRODE	ERROBJ	CPU
1	10	11	172	3268	0.89E − 02	0.58E − 13	0.44
2	19	4	74	2738	0.24E − 04	0.36E − 12	0.28
3	37	4	74	5402	0.82E − 06	0.13E − 13	0.40
4	65	3	59	7611	0.48E − 06	0.46E − 13	0.52
5	83	3	59	9735	0.49E − 06	0.46E − 13	0.63
6	99	3	59	11623	0.25E − 07	0.16E − 12	0.74
	99	28	497	40377			3.01

It is instructive to examine the grid strategy. Fig. 3 shows the additional grid points added at each iteration for the noconstraint problem. The initial grid is at 1.0 and each successive grid addition is 0.1 below.

We see that the mesh refinement strategy is placing grid points at the left-hand edge where there are the faster transients.

Consider now what happens with the g_4 constraint. The grid additions are in Fig. 4. There is reduced order taking place in the right third of the interval. The mesh refinement strategy is placing points near the event around $t = 2.1$ but delays putting the needed points in the right third because it

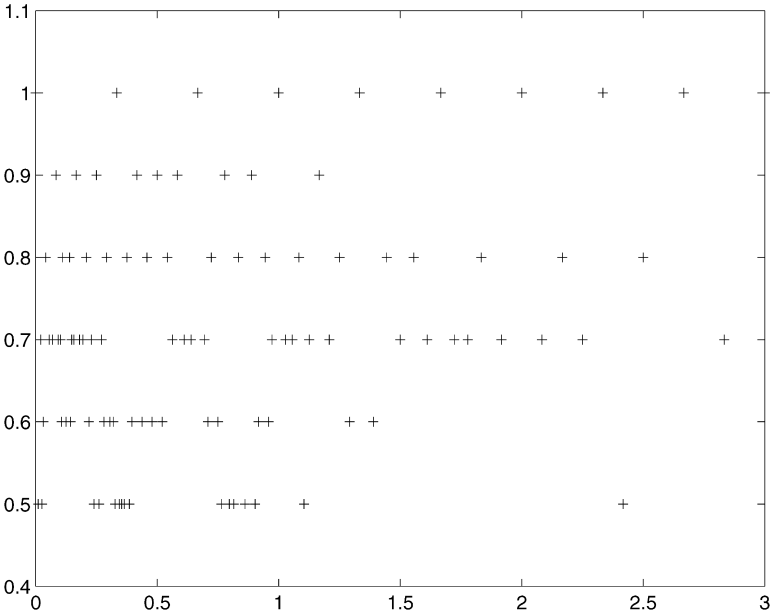

Fig. 3. Grid additions with no constraint.

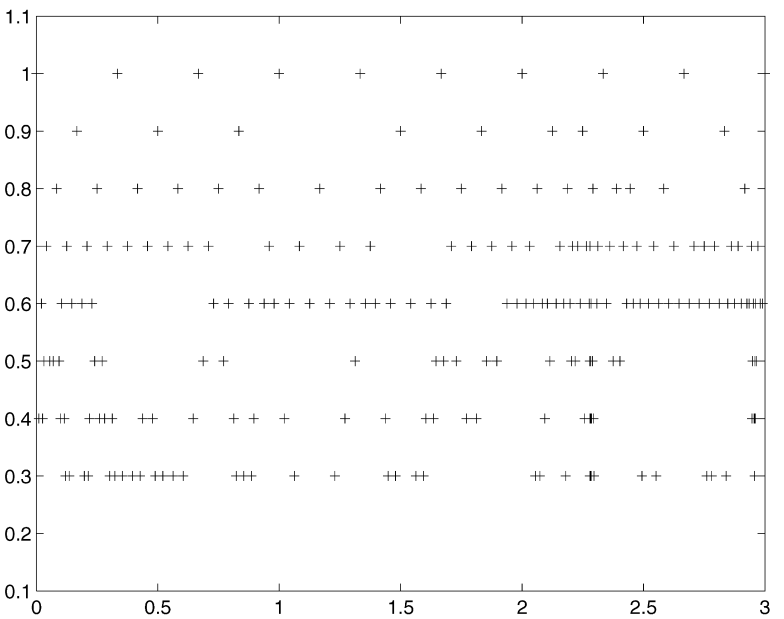

Fig. 4. Grid additions with g_4, standard mesh refinement.

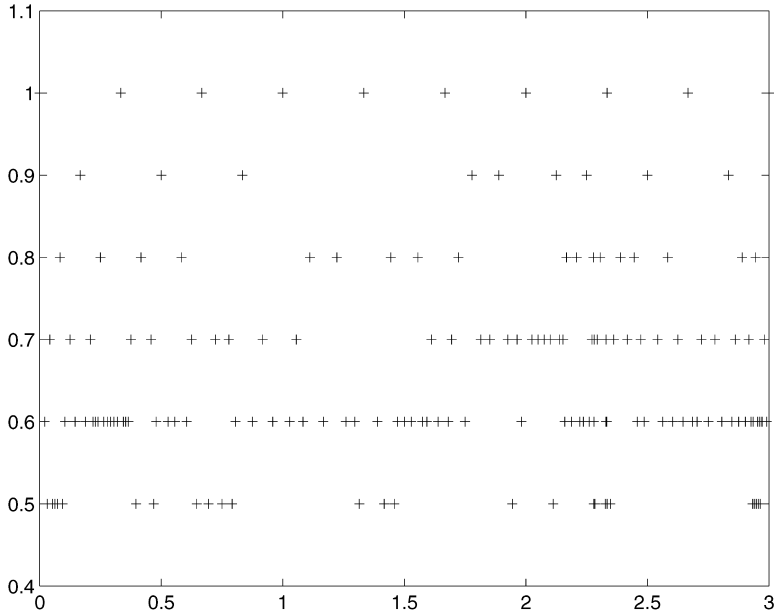

Fig. 5. Grid additions with g_4, modified mesh refinement.

Table 7
Statistics as a function of the order parameter p in SOCS

p	NPT	Iter.
1	231	9
2	137	5
3	261	9
4	292	10

overestimates what the effect of the added points will be. There are also several iterations that add fewer points than allowed.

Fig. 5 shows the grid additions with the mesh strategy modified by assuming that the order is really 2 rather than 4. We see that while a few more grid points than needed are placed in the left two-thirds, there are more points placed early on in the right third where they are needed.

The best choice of the order parameter p also depends on how long constraints are active and other factors. Ideally, this parameter should be dynamically estimated by SOCS. We close this part of the discussion with one final example. This is the same optimal control problem (19) but on the interval $[0,4]$ and constraint (19c) is $x \geqslant 11 - 16e^{(1-t)/2}$. Table 7 gives the mesh size and number of iterations for this problem for various values of the order parameter p.

Table 7 shows that on this problem whose optimum solution is unconstrained for half the time and index three for the other half the choice of $p = 2$ was clearly best.

We have then that the presence of DAEs is causing order reduction and hence there are inefficiencies in the mesh selection. But it is important for us to understand where this order reduction

Table 8
Standard mesh refinement with order parameter $p = 4$, g_2 constraint

GRID	NPT	NFG	NFE	NRHS	ERRODE	ERROBJ	CPU
1	10	16	247	4693	0.29E − 02	0.71E − 03	0.58
2	19	4	74	2738	0.17E − 03	0.30E − 05	0.34
3	37	4	74	5402	0.34E − 04	0.12E − 05	0.73
4	73	4	74	10730	0.30E − 05	0.18E − 07	2.0
5	128	4	74	18870	0.43E − 06	0.42E − 08	5.5
6	153	5	89	27145	0.23E − 06	0.17E − 09	9.4
7	183	3	59	21535	0.24E − 06	0.10E − 10	6.2
8	219	3	59	25783	0.17E − 07	0.15E − 10	7.0
	219	43	750	116896			31.79

Table 9
Order parameter $p = 2$ for mesh refinement, g_2 constraint

GRID	NPT	NFG	NFE	NRHS	ERRODE	ERROBJ	CPU
1	10	16	247	4693	0.29E − 02	0.71E − 03	0.64
2.	19	4	74	2738	0.16E − 03	0.25E − 05	0.44
3	37	4	74	5402	0.22E − 04	0.12E − 05	1.1
4	73	4	74	10730	0.54E − 05	0.61E − 08	3.6
5	134	4	74	19758	0.99E − 06	0.15E − 08	6.7
6	160	3	59	18821	0.11E − 06	0.41E − 10	10.
7	192	3	59	22597	0.28E − 07	0.23E − 10	7.1
	192	38	661	84739			29.68

is coming from since this has important consequences on more complex problems. If the order reduction is due to a reduced value of r in (18), then the codes taking smaller stepsizes will still eventually lead to a reasonable solution although not in an efficient manner. However, if the order reduction is due in part to the h^{-k} value, then taking smaller stepsizes is generally not going to lead to any improvement.

We now turn to briefly examining what is causing the order reduction in our specific example. To see which terms in (18) are important, we consider our test problem with the quadratic forcing function $g_2(t)$. Table 8 gives the result with the standard mesh refinement strategy with order parameter $p = 4$ while in Table 9 the order parameter $p = 2$ mesh strategy is used.

According to the analysis of Section 3, since $g_2^{(4)} = 0$, and the constrained DAE is exactly the system studied in Section 3, we do not expect there to be order reduction effects due to truncation error. However, the numerical results show that the g_2 case is closer to the g_4 case than the no constraint case. (Note that there is one more iteration in Table 9 than in Table 4 but that is because the iteration just misses the 10^{-7} tolerance at the end of iteration 6 in Table 9 and has to take one more iteration.)

This suggests that on this problem with these tolerances that the order reduction might be due in part to the εh^{-k} term.

5.1. Its not that simple

We now turn to an issue which is very important for trajectory optimization codes. Namely do they get a reasonable answer? Let us look more carefully at the computational example. According to the theory when the constraint is active and it is not a cubic polynomial, HS should give an O(1) error. Since this example is linear with constant coefficients and has fourth degree polynomial as forcing function, O(1) is the actual size of the error and not an upper bound. Yet looking at Fig. 1 we see that SOCS appears to be computing the control. Examining the computed solution and the stepsizes selected one finds that in fact the answer for the control is correct with the error between $O(h)$ and $O(h^2)$.

This is not due to some weakness in the theory. We have taken a computed value at say $t=2.5$ and integrated forward the index three DAE using HS implemented as an IRK. The computed value for the control was, in fact, wrong by a term which was O(1) in size. Thus using the HS discretization SOCS has found a correct value for the control but the HS discretization cannot correctly integrate the DAE. We have looked at an even higher index version of the test problem using $x''' = u$ and constraints with all derivatives nonzero. The theory and numerical tests show that HS implemented as an IRK gives an error that grows as a negative power of h. However, SOCS again found an approximation to the control although it was not quite as accurate as in the index three case.

These observations should not be construed as implying that one does not have to worry about the discretizations or the DAE theory. We have seen constrained problems where the codes failed. We have already noted the effects of order reduction. Rather understanding what is happening here is important not only in terms of how to apply DAE theory but also in the implementation of different algorithms so as to reliably solve larger classes of problems. Convergence of optimal control approximations has been widely studied. However, this theory does not generally apply to our situation. For example, in [13,14] an assumption is made which, in our terminology, means that the DAE is at most index two when constraints are active.

The key turns out to be how SOCS is implemented. The HS method is not actually used to first integrate the constrained problem and then evaluate the cost functional as is done with some other approaches which integrate the dyamics in order to reduce the problem size. Rather SOCS proceeds as follows. The constraints and discretization are only required to hold to specified tolerances. The default, and the one we have used, is the square root of machine precision which is $1.4832 \cdot 10^{-8}$ in our case.

In addition, although the theoretical solution is unconstrained up to a particular time and then constrained after that, this is not true for the numerical solution. While it does not show up on the graphs there is usually a small amount of constraint chatter around the time the constraint becomes active. That is, for a few time steps after a constraint should be theoretically active the constraint alternates between being active and inactive. The time steps in this region are typically 10^{-4}–10^{-3}.

In the actual implementation of SOCS, HS is applied to the dynamics and the control is discretized separately. However, as noted earlier the resulting problem, when constraints are active, is equivalent to applying the IRK formulation to the constrained problem. In problems where there are multiple controls, part of the control variable will remain unconstrained, free for optimization, and part will

become part of the DAE state variable. However, that does not occur in the single input test problem discussed here. Given computed values of x, u the dynamics are integrated by HS at a halved stepsize and the local truncation error is estimated. Constraints are not used in this phase. It is done using the computed control u.

We have experimented with a number of other tolerance settings but we limit our discussion here to the default settings in SOCS. Similar observations on the approximation hold if the mesh refinements are made uniform. In this case the overall optimization stops when the local error estimate is below 10^{-7} at all grid points. The interior logic of SOCS prevents considering finer error tolerances.

In order to be specific we shall make a number of assumptions which are stronger than necessary.

Suppose that the process is modeled by a second-order dynamical system $x'' = f(x', x, u, t)$ where f is smooth in x', x, u, t and t is in a finite interval which for simplicity is assumed to be fixed. Suppose that the optimization problem has a unique solution x, u. Furthermore, suppose that there is a partition of the t interval into several subintervals. On the interior of the ith subinterval x, u is a smooth solution of either the original dynamics or a solvable DAE

$$x'' = f(x', x, u, t), \tag{20a}$$

$$0 = C^{[i]}(x', x, t). \tag{20b}$$

Here we assume that $[C_x^{[i]}, C_{x'}^{[i]}]$ is full row rank, $C^{[i]}$ is a subset of the constraints C, and (20) is considered a DAE in x and a subset of u. Finally, we suppose in the final solution that u is continuous and piecewise smooth. We call such an optimization problem *nice*.

Suppose that we have a nice optimization problem. Suppose that we have run the SOCS algorithm and that it has generated numerical estimates for x, u and a final mesh $\mathscr{M}$. Note that SOCS has selected among those sequences that satisfy the inequality constraints to the tolerance. In particular at points with constraint chatter the inequality constraint holds to tolerance but the equality does not hold within tolerance. Let us examine what the true solution looks like. Since SOCS places grid points very close to where constraints change in effect what happens right at these changes is not important here in our discussion. Let us take a closer look inside a subinterval i.

The mesh refinement has continued until the local error estimate is below 10^{-7} everywhere. Thus we expect that it will be significantly lower than that in most places. In fact, where the constraint is active the stepsize is around 0.02 or smaller. Note that $(0.02)^5 = 3.2 \cdot 10^{-9}$. There are other IRK methods that will converge for some DAEs when HS does not. We mention here the 3-stage Lobatto IIIC whose Butcher tableau is

$$\begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

This method integrates successfully index three DAEs in Hessenberg form (11) but with reduced order in the control. From [24] we have convergence of Lobatto IIIC for the variables y, z, u with orders $2, 2, 1$, respectively. Suppose then that we look at the Lobatto IIIC solution of the constrained DAE on the mesh $\mathscr{M}$. It will provide estimates of the control and state. Furthermore, when the local error is checked the constraint is ignored and the integration is done with local order $O(h^5)$. Finally

note that the difference between Mh^5 being $0.8 \cdot 10^{-7}$ and $0.4 \cdot 10^{-7}$ is only about a 10% reduction. Thus, we have the Lobatto IIIC solution is also feasible for meshes very close to the mesh $\mathscr{M}$.

We are not asserting that the Lobatto IIIC solution is the one chosen. We have integrated the constrained DAE with a fixed step Lobatto IIIC method with $h = 0.02$ and compared this with the results of using SOCS where the initial grid was $h = 0.02$ and SOCS was forced to quit after one iteration. SOCS found the control to higher accuracy than Lobatto IIIC. Rather we are pointing out that for discretization tolerances of approximately the same size as the termination tolerance, that it is possible for a number of different numerical methods to produce approximations that are feasible. And a good approximation of the theoretical optimal solution will be close to optimal for fine enough meshes. The optimization process can choose one of these other approximations rather than the one that would come from a straightforward integration of the constrained dynamics by HS. For example, the values of the exact solution on a given subinterval are often either feasible or only a small perturbation from being feasible.

6. Conclusion

We have seen that the theory for IRK methods applied to DAEs interrelates with a trajectory optimization code like SOCS in several ways. In some cases, such as with mesh refinement, the DAE theory correctly describes what is wrong and how to fix it. For other topics, such as convergence of the control estimates, the DAE theory does not tell the full story and further work is needed.

The ideas and results presented in this paper raise a number of interesting questions. One is the role that the cost function plays in the observations we have made. It is known from computational experience that SOCS often performs better when there is some control weighting as in the examples used here. It has also been observed that there can be an interaction between the cost and a DAE process which can result in variational equations of lower index than the original DAE process [11]. These and other questions are under investigation.

Appendix

We outline the main points for a proof of Theorem 3. This result was derived in 1995 but was not submitted for publication since Lobatto IIIA methods are clearly not optimal for the solution of index three Hessenberg DAEs. For this class of problems other families of IRK methods were shown to have better convergence properties [18–20,22–24]. A complete proof of Theorem 3 would be extremely long and is out of the scope of this appendix. Most technical details contained in the thesis [23] and in the article [22] will not be developed here. A proof can be given following the four usual steps when proving convergence of collocation methods and more generally of IRK methods applied to higher index DAEs, see [19–24]: first we show local existence and uniqueness of the IRK solution; secondly we study the influence of perturbations in the nonlinear system of equations; thirdly we study the local error; finally we prove global convergence by studying the error propagation. The ideas for a convergence proof of Lobatto IIIA methods applied to index three Hessenberg DAEs are mainly a mixture of those developed in [21] for the same methods but applied to index two Hessenberg DAEs, of those developed in [22] to prove convergence of

collocation methods for index three Hessenberg DAEs, and of those developed in [23] for IRK methods with an invertible RK matrix applied to index three Hessenberg DAEs, a proof outlined in [24].

For index three Hessenberg DAEs (11), the direct application of an s-stage IRK method reads

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i f(Y_i, Z_i),$$

$$z_1 = z_0 + h\sum_{i=1}^{s} b_i k(Y_i, Z_i, U_i),$$

where the internal stages Y_i, Z_i, U_i must satisfy

$$Y_i = y_0 + h\sum_{j=1}^{s} a_{ij} f(Y_j, Z_j), \tag{A.1a}$$

$$Z_i = z_0 + h\sum_{j=1}^{s} a_{ij} k(Y_j, Z_j, U_j), \tag{A.1b}$$

$$0 = g(Y_i). \tag{A.1c}$$

Various definitions for the numerical solution u_1 of the u-component are possible depending on the IRK method. The Lobatto IIIA coefficients satisfy the properties

H1: $a_{1j} = 0$ for $j = 1, \ldots, s$;
H2: the submatrix $\tilde{A} := (a_{ij})_{i,j\geqslant 2}^{s}$ is invertible;
H3: $b_i = a_{si}$ for $i = 1, \ldots, s$, i.e., the method is stiffly accurate.

For IRK methods satisfying H3 the numerical solution for the u-component is taken as the value of the last internal stage, i.e., $u_1 = U_s$. From H1 we have $Y_1 = y_0$, $Z_1 = z_0$, and U_1 is taken as a given value u_0. When $g(y_0) = 0$ it follows directly from H1 that $g(Y_1) = 0$ is automatically satisfied. We also need the simplifying assumptions

$$B(p)\colon \sum_{i=1}^{s} b_i c_i^{k-1} = \frac{1}{k} \quad \text{for } k = 1, \ldots, p,$$

$$C(q)\colon \sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{c_i^k}{k} \quad \text{for } i = 1, \ldots, s,\ k = 1, \ldots, q.$$

The s-stage Lobatto IIIA method satisfies $B(2s-2)$ and $C(s)$, hence it is also a collocation method. We start with the local existence and uniqueness of the IRK solution:

Theorem A.1. *Suppose that* (η, ζ, v) *satisfy*

$$g(\eta) = 0,$$

$$(g_y f)(\eta, \zeta) = \mathrm{O}(h^2),$$

$$(g_{yy}(f, f) + g_y f_y f + g_y f_z k)(\eta, \zeta, v) = \mathrm{O}(h)$$

and that the IRK method satisfies the assumptions $s \geqslant 2$, H1, H2, *and* $C(2)$. *Then for* $h \leqslant h_0$ *there exists a locally unique solution to* (A.1) *satisfying* $U_1 = v$ *with* (y_0, z_0) *replaced by* (η, ζ). *It satisfies* $Y_1 = \eta$, $Z_1 = \zeta$ *and*

$$Y_i - \eta = \mathrm{O}(h), \quad Z_i - \zeta = \mathrm{O}(h), \quad U_i - v = \mathrm{O}(h) \quad \textit{for } i = 2, \ldots, s.$$

A proof can be given by adapting the one given in [23, Theorem III.2.1] for IRK methods having an invertible RK matrix. The condition $g(\eta) = 0$ is necessary since $g(Y_1) = 0$ must be satisfied. For methods satisfying H3 we have $y_1 = Y_s$, and since $g(Y_s) = 0$ we thus have $g(y_1) = 0$. Whenever η is the value from a previous step given by an IRK method satisfying H3 the condition $g(\eta) = 0$ is naturally satisfied.

The second step is to analyze the influence of perturbations to the IRK solution. The main difference with [22, Theorem 3.2] is the fact that perturbations in the u-component must be considered and that they affect all components:

Theorem A.2. *In addition to the assumptions of Theorem* A.1 *suppose that* H3, $B(p), C(s)$ *hold and that the values* $(\hat{\eta}, \hat{\zeta}, \hat{v})$ *satisfy*

$$g(\hat{\eta}) = 0, \quad \Delta\eta = \mathrm{O}(h), \quad \Delta\zeta = \mathrm{O}(h^{\kappa}) \quad \textit{for } \kappa \geqslant 2, \quad \Delta v = \mathrm{O}(h^{\gamma}) \quad \textit{for } \gamma \geqslant 1.$$

Then we have

$$\begin{aligned}
\Delta y_1 &= P_y \Delta\eta + h f_z P_z \Delta\zeta \\
&\quad + \mathrm{O}(h\|\Delta\eta\| + h^2\|P_z\Delta\zeta\| + h^{m+2}\|Q_z\Delta\zeta\| + \|Q_z\Delta\zeta\|^2 \\
&\qquad + h^{m+3}\|\Delta v\| + h^2\|\Delta v\|^2),
\end{aligned}$$

$$\begin{aligned}
hP_{z,1}\Delta z_1 &= hP_z\Delta\zeta \\
&\quad + \mathrm{O}(h\|Q_y\Delta\eta\| + h^2\|P_y\Delta\eta\| + h^2\|P_z\Delta\zeta\| + h^{\ell+2}\|Q_z\Delta\zeta\| \\
&\qquad + \|Q_z\Delta\zeta\|^2 + h^{\ell+3}\|\Delta v\| + h^2\|\Delta v\|^2),
\end{aligned}$$

$$\begin{aligned}
hQ_{z,1}\Delta z_1 &= R(\infty) hQ_z\Delta\zeta \\
&\quad + \mathrm{O}(\|Q_y\Delta\eta\| + h\|P_y\Delta\eta\| + h^2\|\Delta\zeta\| + h^3\|\Delta v\|),
\end{aligned}$$

$$\begin{aligned}
h^2\Delta u_1 &= R(\infty) h^2\Delta v \\
&\quad + \mathrm{O}(\|Q_y\Delta\eta\| + h\|P_y\Delta\eta\| + h\|Q_z\Delta\zeta\| + h^2\|P_z\Delta\zeta\| + h^3\|\Delta v\|),
\end{aligned}$$

where $m = \min(\kappa - 2, \gamma - 1, s - 3, \max(p - s - 1, 0))$, $\ell = \min(\kappa - 2, \gamma - 1,\ s - 3,\ p - s)$. *The linear operator* S *and the projectors* Q_y, P_y, Q_z, P_z *are defined by*

$$S := k_u (g_y f_z k_u)^{-1} g_y,$$

$$Q_y := f_z S, \quad P_y := I - Q_y, \quad Q_z := S f_z, \quad P_z := I - Q_z.$$

The arguments of $P_{z,1}$ *and* $Q_{z,1}$ *are* (y_1, z_1, u_1). *When no arguments are mentioned they are given by* (η, ζ, v). *The notation* Δ *is self-explicit, for example we have* $\Delta\eta = \eta - \hat{\eta}$, $\Delta\zeta = \zeta - \hat{\zeta}$, *etc. The*

expression $R(\infty)$ is the value of the stability function of the IRK method evaluated at ∞. When the function k in (11) *is linear in u we have above $m = \max(p-s-1, 0)$, $\ell = p-s$, the quantity $\|Q_z \Delta\zeta\|^2$ is multiplied by a factor h, and there is no term $\|\Delta v\|^2$.*

The proof follows the lines of the proof of [22, Theorem 3.2]. From $g(\eta) = 0 = g(\hat{\eta})$ the expression $Q_y \Delta\eta$ satisfies $\|Q_y \Delta\eta\| = h\|P_y \Delta\eta\|$ and can thus be neglected. We stress the point that we need to consider the method both as an IRK method and as a collocation method in the proof. The main difference in the proof comes from the estimate $\tilde{\theta}''(t) = \mathrm{O}(h^{s-2})$ and the presence of $\|\Delta v\|$ in the estimates.

For given consistent initial values (y_0, z_0, u_0) we now take a look at estimates for the local error of the IRK solution

$$\delta y_{h_0}(t_0) = y_1 - y(t_0 + h_0),$$

$$\delta z_{h_0}(t_0) = z_1 - z(t_0 + h_0),$$

$$\delta u_{h_0}(t_0) = u_1 - u(t_0 + h_0).$$

Theorem A.3. *Assume that $s \geqslant 2$ and that* H1–H3, $B(p), C(s)$ *hold. Then we have*

$$\delta y_{h_0}(t_0) = \mathrm{O}(h_0^{p+1}), \quad \delta z_{h_0}(t_0) = \mathrm{O}(h_0^s),$$

$$P_z(t_0 + h_0)\delta z_{h_0}(t_0) = \mathrm{O}(h_0^{\min(p, \max(s, 2s-4))+1}), \quad \delta u_{h_0}(t_0) = \mathrm{O}(h_0^{s-1}).$$

If in addition the function k in (11) *is linear in u we have $P_z(t_0 + h_0)\delta z_{h_0}(t_0) = \mathrm{O}(h_0^{p+1})$.*

Compared to [22, Theorem 4.1] the main difference in the proof comes from the estimates $\theta'(t) = \mathrm{O}(h_0^{s-1})$ and $\theta''(t) = \mathrm{O}(h_0^{s-2})$ where $0 = g(Y(t)) + \theta(t)$ with $Y(t)$ being the collocation polynomial for the y-component on the interval $[t_0, t_0 + h_0]$.

Finally, we can give a global convergence result.

Theorem A.4. *Consider the index three Hessenberg DAEs* (11) *with consistent initial values and an IRK method satisfying assumptions* H1–H3, $B(p), C(s)$, *and $s \geqslant 4$. Then for $t_n - t_0 \leqslant Const.$ with $h := \max_n h_n$, the global error satisfies*

$$y_n - y(t_n) = \mathrm{O}(h^{\min(p, 2s-5)}), \quad z_n - z(t_n) = \mathrm{O}(h^{s-1}),$$

$$P_z(t_n)(z_n - z(t_n)) = \mathrm{O}(h^{\min(p, 2s-6)}), \quad u_n - u(t_n) = \mathrm{O}(h^{s-3}).$$

If in addition the function k in (11) *is linear in u the assumption $s \geqslant 4$ can be relaxed to $s \geqslant 3$ and we have*

$$y_n - y(t_n) = \mathrm{O}(h^{\min(p, 2s-4)}), \quad P_z(t_n)(z_n - z(t_n)) = \mathrm{O}(h^{\min(p, 2s-4)}).$$

The technique of proof is similar to the ones given in [19,22–24]. The condition $s \geqslant 4$ is needed to ensure convergence for the u-component.

Theorem 3 is simply Theorems A.3 and A.4 applied with $p = 2s - 2$.

References

[1] A. Barclay, P.E. Gill, J. Ben Rosen, SQP methods and their application to numerical optimal control, Report NA 97-3, Department of Mathematics, University of California, San Diego, 1997.

[2] J.T. Betts, P.D. Frank, A sparse nonlinear optimization algorithm, J. Optim. Theory Appl. 82 (1994) 519–541.

[3] J.T. Betts, W.P. Huffman, Mesh refinement in direct transcription methods for optimal control, Optim. Control Appl. Methods 19 (1998) 1–21.

[4] L.T. Biegler, Optimization strategies for complex process models, Adv. Chem. Eng. 18 (1992) 197–256.

[5] H.G. Bock, H.X. Phu, J.P. Schlöder, Extremal solutions of some constrained control problems, Optimization 35 (1995) 345–355.

[6] H.G. Bock, H.X. Phu, J.P. Schlöder, The method of orienting curves and its application to manipulator trajectory planning, Numer. Funct. Anal. Optim. 18 (1997) 213–225.

[7] H.G. Bock, J.P. Schlöder, V.H. Schulz, Differential-algebraic equations and their connection to optimization, Heidelberg, preprint 96-58, 1996. Documentation for a SIAM short course.

[8] K.E. Brenan, Differential-algebraic equations issues in the direct transcription of path constrained optimal control problems, Ann. Numer. Math. 1 (1994) 247–263.

[9] K.E. Brenan, A large-scale generalized reduced gradient algorithm for flight optimization and sizing problems, The Aerospace Corporation, ATR-94(8489)-3, 1994.

[10] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations, SIAM, Philadelphia, PA, 1996.

[11] S.L. Campbell, High index differential algebraic equations, Mech. Struct. Mach. 23 (1995) 199–222.

[12] J.E. Cuthrell, L.T. Biegler, On the optimization of differential-algebraic process systems, A.I.Ch.E. J. 33 (1987) 1257–1270.

[13] A.L. Dontchev, Error estimates for a discrete approximation to constrained control problems, SIAM J. Numer. Anal. 18 (1981) 500–514.

[14] A.L. Dontchev, W.W. Hager, A new approach to Lipschitz continuity in state constrained optimal control, Systems Control Lett. 35 (1998) 137–143.

[15] P.J. Enright, Optimal finite-thrust spacecraft trajectories using direct transcription and nonlinear programming, Ph.D. Thesis, University of Illinois, 1991.

[16] P.J. Enright, B.A. Conway, Discrete approximations to optimal trajectories using direct transcription and nonlinear programming, AIAA J. Guidance Control Dyn. 15 (1992) 994–1002.

[17] W.F. Feehery, P.I. Barton, Dynamic simulation and optimization with inequality path constraints, Comp. Chem. Eng. 20 (1996) S707–S712.

[18] E. Hairer, L.O. Jay, Implicit Runge–Kutta methods for higher index differential-algebraic systems, WSSIAA, Contrib. Numer. Math. 2 (1993) 213–224.

[19] E. Hairer, Ch. Lubich, M. Roche, The Numerical Solution of Differential-Algebraic Systems by Runge–Kutta Methods, Lecture Notes in Mathematics, Vol. 1409, Springer, Berlin, 1989.

[20] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II, Stiff and Differential Algebraic Problems, 2nd Revised Edition, Springer, Berlin, 1991.

[21] L.O. Jay, Convergence of a class of Runge–Kutta methods for differential-algebraic systems of index 2, BIT 33 (1993) 137–150.

[22] L.O. Jay, Collocation methods for differential-algebraic equations of index 3, Numer. Math. 65 (1993) 407–421.

[23] L.O. Jay, Runge–Kutta type methods for index three differential-algebraic equations with applications to Hamiltonian systems, Ph.D. Thesis, Department of Mathematics, University of Geneva, Switzerland, 1994.

[24] L.O. Jay, Convergence of a class of Runge–Kutta methods for differential-algebraic systems of index 3, Appl. Numer. Math. 17 (1995) 97–118.