Computational Experience with Exploiting Order Variation in Mesh Refinement for Direct Transcription Methods

J. T. Betts* N. Biehn† S. L. Campbell‡ W. P. Huffman§

Abstract

The numerical theory for Implicit Runge Kutta methods shows that there can be order reduction when these methods are applied to either stiff or differential algebraic equations. A previous paper introduced a way to try and compensate for this order reduction in designing mesh refinement strategies. This paper presents the results from a number of computational studies on the effectiveness of this approach. In addition we present a new test problem which can be used to examine the efficiency of codes developed for a particular class of applications.

Key words: Optimal control, Implicit runge-kutta, Mesh refinement.

AMS subject classifications: 65K10.

1 Introduction

The direct transcription approach to solving optimal control problems parameterizes the dynamic variables using values at mesh points on the interval thus transcribing the problem into a finite dimensional nonlinear programming (NLP) problem. The NLP problem is solved and the discretization refined if necessary. A method for iteratively refining the mesh such that the discrete problem is an adequate approximation to the continuous one was presented in [5]. Like most direct transcription optimization codes, this method assumed the order of the discretization was known and constant. However, during the course of the optimization process the actual order of the discretization may vary with iteration and location because the activation of constraints creates differential algebraic equations (DAEs) on subarcs or there is a local change in stiffness or roundoff error effects are occurring on finer meshes. There is a complex interaction between order and the optimization process. The DAE theory correctly predicts order reduction but does not always correctly predict what that order reduction is for optimization problems [4]. Having the wrong value for the order can seriously impact mesh refinement algorithms [4]. A major modification of the mesh refinement strategy in [5] which attempts to compensate for this order reduction was given in [3]. One computational example was given. Here we present results from the first extensive testing of the approach of [3]. A longer version of this paper with more detail and examples is found on the third author's web site. In addition, a new test problem of some independent interest is presented. Our discussion will be in terms of a particular industrial optimization code, $\mathbb{SOCS}$, developed at Boeing. However, the comments and observations are relevant to any other optimal control code with a similar overall design philosophy.

$\mathbb{SOCS}$ solves optimal control problems by the direct transcription approach. Typically the dynamics of the system are defined for $t_I \le t \le t_F$ by a set of state equations, $\mathbf{y}' = \mathbf{f}(\mathbf{y}(t), \mathbf{u}(t), t)$, boundary conditions, algebraic path

*Mathematics and Engineering Analysis, The Boeing Company, P.O. Box 3707, MS 7L-21, Seattle, WA 98124-2207. john.t.betts@boeing.com.

†North Carolina State University, Operations Research Program, Raleigh, NC 27695-7913. ndbiehn@unity.ncsu.edu.

‡North Carolina State University, Department of Mathematics, Raleigh, NC 27695-8205. slc@math.ncsu.edu. Research supported in part by NSF Grants DMS-9714811 and DMS-9802259.

§Mathematics and Engineering Analysis, The Boeing Company, P.O. Box 3707, MS 7L-21, Seattle, WA 98124-2207. whuffman@redwood.rt.cs.boeing.com.

constraints, simple state bounds, and simple control bounds. The problem is to determine the $\mathbf{u}(t)$ that minimizes the performance index $J = \phi\left(\mathbf{y}(t_I), t_I, \mathbf{y}(t_F), t_F\right)$. $\mathbb{SOCS}$ can handle other formulations and problems with multiple phases.

At each mesh iteration, the time interval is divided into segments $t_I = t_1 < t_2 < \ldots < t_M = t_F$. Let $\mathbf{y}_k$, $\mathbf{u}_k$, indicate the computed values of the state and control variables at a mesh point. The control variable at an interval midpoint is $\overline{\mathbf{u}}_k \equiv \mathbf{u}(\overline{t})$, $\overline{t} = \frac{1}{2}(t_k + t_{k-1})$. The two primary discretization schemes used in $\mathbb{SOCS}$ are the trapezoidal (TR) and Hermite-Simpson (HS). Both are Implicit Runge Kutta (IRK) methods of the Lobatto IIIA type. Each scheme produces a distinct set of NLP variables. For the HS discretization, the NLP variables are $\left\{\{\mathbf{y}_j, \mathbf{u}_j\}_{j=1}^{M}, \{\overline{\mathbf{u}}_j\}_{j=2}^{M-1}, t_I, t_F\right\}$. The state equations are approximately satisfied by solving the *defect* constraints $\boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left[\mathbf{f}_{k+1} + 4\overline{\mathbf{f}}_{k+1} + \mathbf{f}_k\right]$, $h_k \equiv t_{k+1} - t_k$, $\mathbf{f}_k \equiv \mathbf{f}(\mathbf{y}(t_k), \mathbf{u}(t_k), t_k)$, $\overline{\mathbf{f}}_{k+1} = \mathbf{f}(\overline{\mathbf{y}}_{k+1}, \overline{\mathbf{u}}_{k+1}, \overline{t})$, $\overline{\mathbf{y}}_{k+1} = \frac{1}{2}\left[\mathbf{y}_k + \mathbf{y}_{k+1}\right] + \frac{h_k}{8}\left[\mathbf{f}_k - \mathbf{f}_{k+1}\right]$. The various constraints are replaced by the NLP constraints. Path constraints are imposed at the mesh points. For the HS discretization the path constraints and variable bounds are also imposed at the interval midpoints. $\mathbb{SOCS}$ solves this large, sparse NLP using a sequential quadratic programming (SQP) method.

The first step in the mesh refinement process is to construct an approximation to the continuous solution from the NLP solution. We use spline approximations $\widetilde{\mathbf{y}}(t)$, $\widetilde{\mathbf{u}}(t)$ for $\mathbf{y}(t)$, $\mathbf{u}(t)$. We *assume* $\widetilde{\mathbf{u}}(t)$ is correct (and optimal), and estimate the error between $\widetilde{\mathbf{y}}(t)$ and $\mathbf{y}(t)$. Optimality of $\widetilde{\mathbf{u}}(t)$ is not checked when measuring the discretization error. However, $\widetilde{\mathbf{y}}(t)$ will accurately reflect what $\mathbf{y}(t)$ will be if $\widetilde{\mathbf{u}}(t)$ is used. The new mesh strategy differs in two key ways. Order reduction is estimated on each subinterval on every iteration and error equidistribution is emphasized.

We expect the order reduction to vary along the mesh depending on whether constraints are active and whether the problem is stiff. It varies with iterations due to the shortening of boundary layers and the effects of error buildup on finer meshes. Consistent with the philosophy of $\mathbb{SOCS}$, in estimating the local error we assume the computed control is correct. Define the *absolute local error* on a particular step by $\boldsymbol{\eta}_{i,k} = \int_{t_k}^{t_{k+1}} |\boldsymbol{\varepsilon}_i(s)| ds = \int_{t_k}^{t_{k+1}} \left| \dot{\widetilde{\mathbf{y}}}(s) - \mathbf{f}(\widetilde{\mathbf{y}}(s), \widetilde{\mathbf{u}}(s), s) \right| ds$. Using spline approximations, we evaluate the integral using Romberg quadrature.

We estimate the order reduction r by comparing the behavior on two successive mesh refinement iterations. We assume the order reduction is the same for all $I+1$ subdivisions of the old interval. Thus the "resolution" of our order reduction estimates is dictated by the old "coarse" grid.

Suppose we are going to subdivide the current grid. Let I_k be the number of points to add to interval k and let $\epsilon_k = \max_i \frac{\boldsymbol{\eta}_{i,k}}{(w_i+1)} \left(\frac{1}{1+I_k}\right)^{p-r_k+1}$ for integers $I_k \geq 0$. The new mesh is constructed by choosing I_k to solve the nonlinear *integer programming problem*: minimize: $\phi(I_k) = \max_k \epsilon_k$ subject to $\sum_{k=1}^{n_s} I_k \leq M-1$ and $I_k \leq M_1$. This minimizes the maximum error over all of the intervals in the current mesh by adding at most $M-1$ total points. The number of points that are added to a single interval is limited to M_1. Typically we use $M_1 = 5$.

The termination criteria for the mesh refinement in $\mathbb{SOCS}$ is a desired error tolerance δ. We want the *predicted* errors to be "safely" below, say at $\hat{\delta} = \kappa\delta$ where $0 < \kappa < 1$. Typically we set $\kappa = 1/10$. One of our long range interests is in the optimal control of systems where high precision is needed, for example, in the milling of parts to high tolerances. Accordingly we take $\delta = 10^{-7}$ in the tests that follow.

2 Computational Tests

In evaluating the new mesh strategy we want to make sure that we do not have worse performance on standard problems and to see if there is improved performance on some important class of applications. In making comparisons we face an immediate technical problem in that the requested discretization error tolerance ERRODE is computed differently in the two versions of $\mathbb{SOCS}$. We expect the new $\mathbb{SOCS}$ estimate to be more accurate. For the Alp Rider and Venus flyby problems described below this is not important because it is easy to see improvements with the new version. However, for a more general comparison it is important to understand the impact these differences in ERRODE have. This is examined for the CSTR problem. For the remainder of this paper $\mathbb{SOCS}$ refers to the original version (Version 3.0), while NSOCS uses the new mesh strategy (Version 4.1). Except for Sections 2.1 and 2.3 HS was used on all iterates.

2.1 The Boeing Test Set

The Standard $\mathbb{SOCS}$ Test Problems (SSTP) [1] is a suite of problems that are used to test changes in $\mathbb{SOCS}$ and to compare $\mathbb{SOCS}$ to other codes on. SSTP consists of 53 optimal control and/or boundary value problems, with and without, path constraints. Many of the problems are highly nonlinear. The test set includes some classical academic control test problems and a number of optimal control problems from applications. The later are primarily aircraft

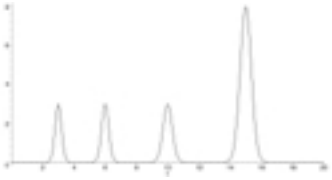

Figure 1a: Constraint surface diameter for (1).

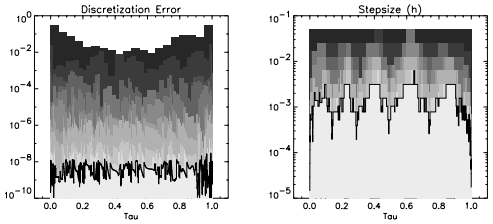

Figure 1b: Mesh refinement history for NSOCS on (1).

flight control problems and are of modest size (5-20 algebraic-differential equations). Table 1 compares NSOCS to $\mathbb{SOCS}$ on SSTP. From this table, and a finer examination of the results which space prohibits presenting, we see that there is little change for most "standard" problems. There is a substantial improvement for some problems. On "easy" low dimensional problems NSOCS sometimes uses slightly more CPU time. On more difficult problems, the NLP solve CPU time dominates and the smaller M value and better control of iterations by NSOCS more than compensates for the extra mesh refinement computation. The remainder of this paper considers problems which are not in SSTP.

Total Time Decrease (16601.63 vs. 12956.20)	-28 %
Average Percent Change (all problems)	-0.23 %
Maximum Percent Increase (all problems)	109.31 %
Minimum Percent Decrease (all problems)	-62.20 %

Table 1: $\mathbb{SOCS}$ mesh refinement strategy vs NSOCS strategy on SSTP.

2.2 Alp Rider

There are a number of applications where one wishes to optimize the performance of a responsive system which is trying to follow a complex surface which may change abruptly. Examples include terrain following aircraft and the rapid machining of complex mechanical parts. In some cases, the answer must be known to high percision. With tool paths, high precision is necessary in order to get a high quality part. These types of problems are not reflected in SSTP. Accordingly we have constructed the test problem (1) which we call Alp Rider. Here $p(t,a,b) = e^{-b(t-a)^2}$.

$$\min_u J(x,u) = \min_u \int_0^{20} 10^2(x_1^2+x_2^2+x_3^2+x_4^2) + 10^{-2}(u_1^2+u_2^2)\, dt \tag{1a}$$

$$x_1' = -10x_1 + u_1 + u_2, \qquad x_2' = -2x_2 + u_1 + 2u_2 \tag{1b}$$

$$x_3' = -3x_3 + 5x_4 + u_1 - u_2, \quad x_4' = 5x_3 - 3x_4 + u_1 + 3u_2 \tag{1c}$$

$$x_1^2+x_2^2+x_3^2+x_4^2 \geq 3p(t,3,12) + 3p(t,6,10) + 3p(t,10,6) + 8p(t,15,4) + .01 \tag{1d}$$

$$x^T(0) = [2,1,2,1], \quad x^T(20) = [2,3,1,-2] \tag{1e}$$

Eigenvalues of the linear system are $\{-10, -2, -3 \pm 5i\}$ so that it has both rapidly decaying and oscillating modes. The function on the right of (1d) is given in Figure 1a. NSOCS on (1) is given in Table 2.

Mesh	M	ERRODE	CPU	Mesh	M	ERRODE	CPU
1	21	0.32E+00	0.22E+02	7	253	0.11E-04	0.14E+03
2	41	0.29E-01	0.66E+02	8	304	0.37E-05	0.16E+03
3	76	0.74E-02	0.63E+02	9	607	0.35E-06	0.54E+03
4	84	0.89E-03	0.35E+02	10	785	0.16E-06	0.92E+03
5	119	0.18E-03	0.65E+02	11	992	0.29E-07	0.14E+04
6	194	0.31E-04	0.95E+02	CPU Total =3543.81			

Table 2: New mesh strategy on Alp Rider.

In comparison, $\mathbb{SOCS}$ used 27370.44 (sec), 2494 mesh points, and 10 mesh refinement iterations. The CPU time has been reduced from 7.6 hours to 0.98 hours. The numerical solution appears in the Figure 2. The first graph in Figure 1b gives the discretization error on each mesh iteration. Notice how the curves get flatter as the error is equi-

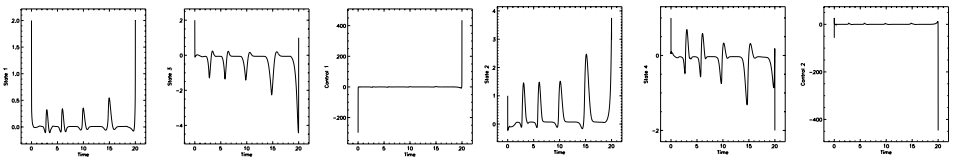

Figure 2: Solution of Alp Rider.

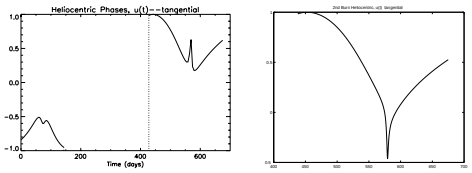

Figure 3a: Portions of old and new controls for Venus flyby.

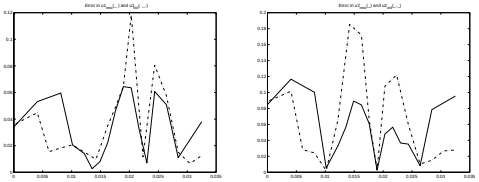

Figure 3b: Control error for CSRT, $M = 17$.

distributed. The second graph plots mesh size against its location for each iteration. As expected the finest meshes occur in the vicinity of each peak and near the boundaries where there are rapid dives and climbs. Table 2 illustrates another important difference. In $\mathbb{SOCS}$ the mesh typically doubles in size for each iteration until the last few iterates. Sometimes it was necessary to continue iterations when the problem was at its largest in order to resolve sensitive regions. In NSOCS the emphasis is on equidistributing the error. If after adding less than $M-1$ mesh points, the most error is in a subinterval where we have already added 5 points, then that mesh refinement is ended and we do another NLP. Thus sensitive areas are resolved earlier before the mesh gets too fine. Note that in Table 2 in iterations 3-8, mesh doubling does not occur. Only 8 points are added at iteration 4. At iteration 8 there is a mesh doubling.

2.3 Venus flyby

NSOCS places extra emphasis on getting things right where there are rapid transients. In complex optimal control problems there can be regions where there are rapid dynamics for which high precision is necessary. We give one example from [2]. The problem is for a rocket to travel from the earth around Venus and then on to Mars. Gravitational effects of Mercury, Jupiter, and the sun are included. There is to be an initial burn, a long coast and a final burn. There are lower bounds on the nearness of the approach to Venus. Realistic parameter values are used. A fixed thrust engine is assumed. Control variables are thrust direction (rocket orientation) and whether the motor is on or off. The rocket is given an initial mass (400,000 kg) and a date (675 days later) to be at Mars. Final mass is to be optimized. The larger the final mass, the less fuel consumed, and the more payload that can be delivered. This problem was solved using $\mathbb{SOCS}$ [2]. About 6 days were spent in the region of Venus. During the long periods between planetary approaches Δt_i was 2-4 days. At the closest to Venus, it dropped to 6 minutes. The final answer is sensitive to what happens near Venus. This problem was resolved with NSOCS. There is still a small, but quick maneauver at around 570 days but it is changed. $\mathbb{SOCS}$ gave 294748.8480 kg while NSOCS gives 297496.0376 which is an increase of 2747 kg. The better job in refining the rapid dynamics and reducing error has lead to an optimum that, for practical purposes, in significantly larger. The left half of Figure 3a shows parts of the control history from [2]. The right half shows an expanded version of the new control in the region of the upward spike in the left half. NSOCS used 20% more mesh points and 30% more CPU time. Its better error estimator correctly determined the need for more mesh points.

2.4 Two-Stage CSTR

This problem is from [6]. It is a standard problem in the chemical engineering literature. Note the large constants. We have varied the problem slightly by removing the bounds on the controls u_i and adding a penalty term to the end. Both approaches work well. It is of interest to see how they are different. For this problem we do a more detailed analysis to examine the effect of the different ways that ERRODE is computed.

$$\text{(2a)} \qquad J(x,u) = \int_0^{0.325} x_1^2 + x_2^2 + x_3^2 + x_4^2 + 0.1(u_1^2 + u_2^2)dt$$

$$\text{(2b)} \qquad x_1' = -3x_1 + g_1(x), \qquad x_2' = -11.1558x_2 + g_1(x) - 8.1558(x_2 + 0.1592)u_1$$

$$\text{(2c)} \qquad x_3' = 1.5(0.5x_1 - x_3) + g_2(x), \quad x_4' = 0.75x_2 - 4.9385x_4 + g_2(x) - 3.4385(x_4 + 0.122)u_2$$

$$\text{(2d)} \qquad g_1(x) = \alpha_1(0.5251 - x_1)\exp(\tfrac{-10}{x_2+\beta_1}) - \alpha_2(0.4748 + x_1)\exp(\tfrac{-15}{x_2+\beta_1}) - 1.4280$$

$$g_2(x) = \alpha_1(0.4236 - x_2)\exp(\tfrac{-10}{x_4+\beta_2}) - \alpha_2(0.5764 + x_3)\exp(\tfrac{-15}{x_4+\beta_2}) - 0.5086 \tag{2e}$$

$$x(0)^T = [0.1962, -0.0372, 0.0946, 0, 0], \quad x(0.325) = 0 \tag{2f}$$

with $\alpha_1 = 1.5 \times 10^7, \alpha_2 = 1.5 \times 10^{10}, \beta_1 = 0.6932, \beta_2 = 0.6560$. Iteration statistics are given in Tables 3a and 3b.

Mesh	M	ERRODE	CPU
1.0	3	0.54E-02	0.24E+01
2.0	5	0.82E-03	0.45E+00
3.0	9	0.40E-04	0.71E+00
4.0	17	0.50E-05	0.74E+00
5.0	33	0.31E-06	0.13E+01
6.0	65	0.15E-07	0.21E+01
	65		7.74

Table 3a: NSOCS on CSTR

Mesh	M	ERRODE	CPU
1.0	3	0.36E+00	0.22E+01
2.0	5	0.30E-01	0.36E+00
3.0	9	0.23E-02	0.49E+00
4.0	17	0.16E-03	0.46E+00
5.0	33	0.69E-05	0.14E+01
6.0	65	0.43E-06	0.11E+01
7.0	111	0.98E-07	0.18E+01
	111		7.91

Table 3b: SOCS on CSTR

The two values of the objective function were 1.2343071 and 1.2343072. The two final grids were combined into one mesh which had 127 points. Thus NSOCS chose 16 different places to add points during successive iterations. The 127 mesh points were then used to evaluate the B-spline obtained after both $\mathbb{SOCS}$ and NSOCS had terminated. Table 4 gives the supnorm of the difference between the two evaluated B-splines of $\mathbb{SOCS}$ and NSOCS. The closeness of the optimal values of the two controls shows that the two solutions are equivalent in a cost sense. The larger error in the controls reflects both the lower accuracy to which controls are found and also the sensitivity of the cost to the control. Each solution was put into the other version as a starting value and the change noted. It was of the same order as in Table 4. Thus on the CSTR we may consider the two solutions as practically equivalent.

x_1	1.5928e-07	x_2	1.3191e-06	x_3	9.3717e-08	x_4	1.8637e-06	u_1	7.9273e-03	u_2	2.2433e-02

Table 4: Supnorm of differences of $\mathbb{SOCS}$ and NSOCS solutions

In practice we have seen a number of complex applications where one was forced to take a solution from an intermediate iterate because growth of CPU time meant solving the problem to the requested ERRODE was not practical. Does one of the two approaches do a better job on the intermediate steps? We look at the difference between the approximation on the ith step and the final solution. For $M = 3, 5, 33$, the two methods were essentially identical. Table 5 gives the results for NSOCS and $\mathbb{SOCS}$ at $M = 9, 17$. On the intermediate grids, NSOCS has about half as much error in the control variables. This is consistent with Tables 3a and 3b which show a lower ERRODE for NSOCS. Thus the observed differences in ERRODE between the two methods reflects a real difference and is not just a consequence of ERRODE being computed differently in the two methods. Figure 3b shows the error in the controls for both strategies on the 4th iterate. Note NSOCS (solid line) is more accurate and its error is more equidistributed.

	M	$x_1^6 - x_1^1$	$x_2^6 - x_2^1$	$x_3^6 - x_3^1$	$x_4^6 - x_4^1$	$u_1^6 - u_1^1$	$u_2^6 - u_1^1$
NSOCS	9	2.2600e-04	3.6710e-04	2.0680e-04	4.3410e-04	2.8570e-01	5.9040e-01
	17	2.0300e-05	3.9500e-05	1.1100e-05	2.7300e-05	6.4490e-02	1.1670e-01
$\mathbb{SOCS}$	9	3.8680e-04	2.8000e-04	2.6020e-04	9.4000e-05	4.8361e-01	9.1280e-01
	17	1.6900e-05	4.0000e-05	3.4000e-06	2.4000e-05	1.1830e-01	1.8520e-01

Table 5: Max diff of final solution and ith solution.

2.5 Index Three Constraints

HS and TR do not converge for index three DAEs yet $\mathbb{SOCS}$ is able to solve some of these problems [4]. Problem (3) is more complex than the example given in [4]. The control is not completely determined by the constraint.

$$\min J(u) = \min \int_0^{10} x_1(t)^2 + 10^{-1}x_2(t)^2 + 10^{-3}x_3(t)^2 + 10^{-3}x_4(t)^2 + 10^{-2}u_1(t)^2 + 10^{-2}u_2(t)^2 \, dt \tag{3a}$$

$$x_1' = x_2, \qquad x_2' = u_1 + u_2, \quad x_3' = x_4 + u_2, \quad x_4' = x_3 + u_1 + u_2, \quad 0 \le x_1 - (7 - 0.004(t-10)^4) \tag{3b}$$

with $x_1(0) = 10$, $x_2(0) = 1$, $x_3(0) = -1$, and $x_4(0) = 1$. (3) is linear and index 3 along the constraint which is active from approximately $t = 4.1$ to $t = 10$. Plots of the optimal solution are given in Figure 4a. Tables 6a and 6b contain the iteration statistics. NSOCS is outperforming $\mathbb{SOCS}$. Note $\mathbb{SOCS}$ adds 49 points between the 5th and 6th iterations, yet does not reduce the error in the approximation. The error reduction estimate in NSOCS shows error reduction occuring in three regions, the beginning of the interval where the controls and states are 'fast', the event where unconstrained solution becomes constrained and various spots along the constraint. See Figure 4b. Once convergence is reached on a given subinterval, the order reduction usually becomes zero on that subinterval.

Mesh	M	ERRODE	CPU	Total CPU
1.0	19	0.51E-02	0.20E+01	
2.0	37	0.33E-03	0.27E+01	
3.0	55	0.17E-04	0.46E+01	
4.0	109	0.16E-05	0.19E+02	
5.0	217	0.11E-06	0.59E+02	
6.0	222	0.66E-07	0.57E+02	145.04

Table 6a: NSOCS on (3).

Mesh	M	ERRODE	CPU	Total CPU
1.0	19	0.22E-01	0.15E+01	
2.0	37	0.62E-03	0.18E+01	
3.0	73	0.29E-04	0.55E+01	
4.0	145	0.13E-05	0.19E+02	
5.0	245	0.11E-06	0.52E+02	
6.0	294	0.10E-06	0.81E+02	
7.0	352	0.22E-07	0.96E+02	256.95

Table 6b: $\mathbb{SOCS}$ on (3)

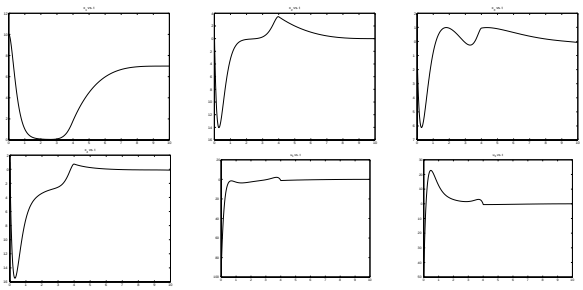

Figure 4a: Solutions of (3).

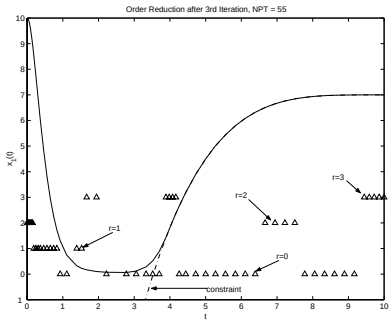

Figure 4b: Order reduction and a solution of (3).

3 Summary and Conclusions

Numerical tests were carried out on a variety of constrained optimal control problems. On straightforward problems the new approach is comparable to the old. On problems where there are rapid transients or state constraints are active, the new approach produces meshes with fewer points. Often CPU times are lower, sometimes dramatically so. Intermediate iterates often provide better approximations. On a space trajectory problem the new emphasis on locating the difficult areas first and equidistributing the error produced a higher quality answer.

References

[1] J. T. Betts, *Sparse nonlinear programming test problems (Release 1)*, BCSTECH-93-047, Boeing Computer Services, 1993.

[2] J. T. Betts, *Optimal Interplanetary Orbit Transfers by Direct Transcription,* J. of the Astronautical Sciences, 42, No. 3, July-Sept 1994, 247-268.

[3] J. T. Betts, N. Biehn, S. L. Campbell, and W. P. Huffman, *Exploiting order variation in mesh refinement for direct transcription methods*, J. Comp. Appl. Math., to appear.

[4] N. Biehn, S. L. Campbell, L. Jay, and T. Westbrook, *Some comments on DAE theory for IRK methods and trajectory optimization*, J. Comp. Appl. Math, to appear.

[5] J. T. Betts and W. P. Huffman, *Mesh refinement in direct transcription methods for optimal control*, Optimal Control Applications & Methods, 19 (1998), pp. 1-21.

[6] R. Luus, W. Mekarapiruk, and C. Storey, *Evaluation of penalty functions for optimal control*, Proc. IOTA'98, Edited by L. Caccetta, K. L. Teo, P. F. Siew, Y. H. Leung, L. S. Jennings, and V. Rehbock, 1998.