Proceedings of the 42nd IEEE
Conference on Decision and Control
Maui, Hawaii USA, December 2003

Initialization of Direct Transcription Optimal Control Software

J. T. Betts
Math. & Engineering Analysis
The Boeing Company
P.O. Box 3707, MS 7L-21
Seattle, Washington 98124-2207.

S. L. Campbell
North Carolina State University
Department of Mathematics
Raleigh, NC 27695-8205

N. N. Kalla
North Carolina State University
Operations Research Program
Raleigh, NC 27695.

Abstract— **Direct transcription optimal control codes have been very successful. One common source of numerical difficulty is getting a feasible solution on the first iteration. Overcoming this often requires a high level of expertise on the user's part. This paper discusses research aimed at the development of a collection of utilities to assist users in solving complex industrial optimal control problems with direct transcription codes.**

I. Introduction

Direct transcription has proved in practice to be effective in solving a wide variety of optimal control problems [2]. One of the most common difficulties with direct transcription codes is getting a feasible solution on the first iteration. There will always be problems that require user expertise and this expertise should be used when available. See for example [4]. However, any software's usefulness is greatly increased the more easily it is usable on complex problems by less experienced users. There is a need for more sophisticated initialization utilities that can be called on for difficult problems. This paper will examine one such utility that is under development. Our discussion is in terms of the direct transcription software SOCS [5] but the comments are relevant to other direct transcription codes. Page limitations prevent us from discussing how this utility would fit within a larger library of utilities. Similarly we have had to delete proofs of the estimates given later and include only a few numerical examples.

A. Direct Transcription

The direct transcription approach to solving optimal control problems parameterizes the dynamic variables using values at grid points on the time interval thus transcribing the problem into a finite dimensional nonlinear programming (NLP) problem. The NLP problem is solved and the discretization refined if necessary. SOCS solves optimal control problems by the direct transcription approach. Typically the dynamics of the system are defined on $t_I \le t_F$ by state equations, $y' = f(y(t), u(t), t)$, boundary conditions, algebraic path constraints, simple state bounds, and simple control bounds. The problem is to determine the $u(t)$ that minimizes a performance index J. SOCS can handle problems with multiple phases. At each grid iteration, the time interval is divided into segments $t_I = t_1 < t_2 < \ldots < t_M = t_F$. The two primary discretization schemes used in SOCS are the second order trapezoidal (TR) and the fourth order Hermite-Simpson (HS). Each scheme produces a distinct set of NLP variables. SOCS currently solves this large, sparse NLP using a sequential quadratic programming (SQP) method. SOCS can adjust the actual grid time values through scaling and this is often done if the final time is not known. Internally the grids are refined only by the addition of extra grid points so that each grid is a subset of the previous grid.

B. Initialization

SOCS needs an initial guess consisting of an initial grid, values of the states and controls on this grid, and values of any parameters. Currently users specify either a uniform grid with a stated number of points or a grid of their choice. They can also either specify the function values directly or let SOCS generate them with a linear interpolation between user specified values at the endpoints of the grid. Experience has shown that *uniform grids are often not acceptable*. Any utility needs to not only determine the size of the grid but also the distribution of the grid points and the values of y, u at those grid points.

This initialization problem has certain aspects in common with the problem of initialization of iterative boundary value problem (BVP) solvers. This is not surprising since it is well known that the necessary conditions for optimal control problems are BVP and the numerical solutions of optimal control problems often act like the numerical solutions of BVP. Also the direct approach used by SOCS is based on global collocation like a BVP solver.

Our approach will take some ideas from the initialization of BVP especially monitor functions [1]. However, there are important differences between our problem and a standard BVP including some differences from the usual numerical analysis where one can just assume the grids are fine enough. Once a feasible solution is found the remaining grid refinements are performed by the sophisticated grid refinement procedure in SOCS. There will be a number of refinement iterations to determine the control and resolve the dynamics. Taking too fine an initial grid will lead to unnecessarily large NLP problems when trying to find the control. Also, the initial guess for the control is often not very good so the grid distribution is likely to not be clustered in those places needed to determine the control sufficiently accurately. Thus in our situation taking too fine an initial grid pays a much larger computational burden than with a plain BVP solver. Also, practical control problems often take place in the presence of a large number of state and control inequality

0-7803-7924-1/03/$17.00 ©2003 IEEE

constraints. Finally we want to make most of the utility as transparent to the user as possible.

Another feature of many industrial grade optimal control problems is that often some of the functions are not given in the form of explicit functions but come out of interpolation of experimental or tabular data. This suggests that, if possible, the utility should rely only on function evaluations of the right hand side of the differential equation.

II. Monitor Functions

The basic idea of a monitor function is as follows. For a given initial control law $u = v(y,t)$, one integrates the differential equation

$$y' = y' = f(y,u,t),\ y(t_0) = y_0 \quad (1a)$$

$$s' = \phi(y,t),\ s(t_0) = 0 \quad (1b)$$

over the interval $[t_0, t_f]$ where $\phi(y,t) \geq \delta > 0$. Then one equidistributes the grid points using s. Thus if there are to be N grid points, they are taken to be

$$t_k = s^{-1}(ks(t_f)/(N-1)), \quad k = 0,\ldots,N-1$$

The arc length monitor function (ALMF) [1] is

$$\phi(y,t) = \sqrt{1 + f(y,u,t)^T f(y,u,t)} \quad (2)$$

which equidistributes the initial y_k values along the graph of the solution.

The examples that follow show that a simple monitor function like (2) is often not best and, in some cases, will fail to provide any assistance. We will also see that even when (2) works, special care is sometimes needed. Let f_i be the ith entry of f. We also consider monitor functions of the form

$$\phi(y,t) = \left(\alpha + \sum_{i=0}^{n} \beta_i f_i^2\right)^{1/2} \quad (3)$$

where $0 < \alpha \leq 1$ and $\beta_i \geq 0$, $\sum_{i=0}^{n} \beta_i > 0$. Notice that a constant multiple of ϕ will produce the same grid as ϕ. The default choice is (2) which is $\alpha = 1, \beta_i = 1$. The user adjustable parameter α in (3) appears in other applications of monitor functions such as with the use of moving grids in solving PDEs [3]. However, we have not seen the β_i parameters before. The β_i are sometimes very useful [4]. Other monitor functions will be discussed later.

A. Estimating N

Once the monitor function has been chosen and the forward integration has determined t_f, it is necessary to come up with N, the number of points in the initial grid. We know M, the number of accepted time steps of the integrator, the order of the integrator, the desired accuracy of our initial guess, and the order of the SOCS discretization used on the initial grid. Extensive computational testing has previously shown that SOCS usually performs better if initialization is done with a lower order integrator like TR and then later SOCS can switch to HS. However, Example 2 shows that generally the initialization integrator has to be at the requested initialization tolerance or tighter.

Naively assuming that things scale, M/N is a ratio of the estimated stepsize for the integrator to the step size needed by the optimizer discretization (OD) to get a certain accuracy. Let P be the order of the integrator used in finding the grid, 10^{-R} = ODE tolerance requested of the integrator, and 10^{-S} = ϵ_{init} = tolerance wanted for the initial guess, Q = the order of OD. Let $L = t_0 - t_f$. Then ignoring the variability of the grid and the fact that these formulas are only true asymptotically we get that $\left(\frac{L}{N}\right)^Q = 10^{-S}$, $\left(\frac{L}{M}\right)^P = 10^{-R}$ which implies that $\frac{M}{N} = 10^{-S/Q+R/P}$. This suggests that a plausible number of initial grid points is

$$N = \min\{\text{MXGD}, \max\{\text{MNGD}, M10^{S/Q-R/P}\}\} \quad (4)$$

where MXGD, MNGD are the largest and smallest initial grid the user will consider.

Algorithm 1: **(Algorithm for t_k, y_k and initial grid.)** *The user specifies the monitor function, an initial guess control law $u = v(y,t)$, an initial condition $y(t_0) = y_0$, and a subroutine to evaluate $f(y,u,t)$. In addition the user can specify a desired tolerance $\epsilon_{init} = 10^{-S}$ whose default is 10^{-1}. The user can set MXGD and MNGD which have defaults of MXGD=300 and MNGD = 15.*

Step 1: *A variable step, fixed order integrator is used to integrate (1) from t_0 to t_f with a requested tolerance of 10^{-R}. The result is s_f, M the number of accepted time steps, and also t_f if t_f was a variable.*

Step 2: *Let N be given by (4) and let*

$$s_k = s_f \frac{k-1}{N-1}, \quad k = 1,\ldots,N \quad (5)$$

Step 3: *Integrate the system (6)*

$$\frac{dy}{ds} = f(y, v(y,t), t)\phi(y,t)^{-1},\ y(0) = y_0 \quad (6a)$$

$$\frac{dt}{ds} = \phi(y,t)^{-1},\ t(0) = t_0 \quad (6b)$$

from 0 to s_f. The solution $y(s), t(s)$ is output at the $s = s_k$. This gives values of y_k, t_k. Note that this integration need not be done by the same integrator used in Step 1.

The only expensive function used is f which is needed by SOCS anyway. Other functions are all simple scalar functions of f. The algorithm assumes the utility provides the integrator so that the order constants are known. We will see later that to get the intervals accurately, we sometimes need to reduce MaxStep. This suggests that perhaps M should be estimated separately from the generation of the grid.

B. Computational Examples

In the computational tests we used ODE45 and ODE23 from MATLAB as the integrators in Steps 1 and 3. These are fixed order variable step Runge-Kutta codes. Unless stated

otherwise the integration tolerance was 10^{-6}. The computed initialization grids are much sparser than the ODE45 grids and do not have the initial clustering of the ODE45 grid due to initial stepsize variation. This shows the *importance of using the monitor function* and not just trying to adapt the ODE45 grid.

1) Example 1: Fast Transients & Long Interval: Example (7) is due to A. V. Rao [8]. It has fast transients very close to either end of a long interval. SOCS solves this problem already using the default utilities. However, this example illustrates several points. TR is the initialization discretization so that $Q = 2$.

The optimization problem, with $t_0 = 0$, $t_f = 10,000$ is

$$\min_u J = \min_u \int_0^{t_f} x^2 + u^2 \, dt \tag{7a}$$

$$x' = -x^3 + u \tag{7b}$$

$$x(0) = 1, \; x(t_f) = 1.5 \tag{7c}$$

Part of any initial guess is the initial choice of the controls. Of course, the true optimal control is usually not known. However, often some rough properties of the control, such as when it will be active, are known. It is expected in problem (7) that the control will be active near the two ends of the interval and not do much for most of the middle portion. While a user cannot be expected to know exactly when this will occur they might well know that it does occur. This suggests that a reasonable choice for an initial u might be

$$u_a(t) = \begin{cases} -1 & \text{if} \quad 0 \le t < a \\ 0 & \text{if} \quad a \le t < 10000 - a \\ 1 & \text{if} \quad 10000 - a \le t \le 10000 \end{cases} \tag{8}$$

We pick $a = 400$ as a guess. The rationale for doing this is that it might provide a better grid just knowing that there is more activity early and late. First we note that if we just tried using (8), and the SOCS defaults for the state variables and the same starting grid of $N = 13$ that worked fine with the defaults, SOCS was unable to get consistency without our having to go to a finer initial grid.

ODE45 had trouble integrating past 9600. Computed values looked reasonable up to this time. Our best guess is that this was due to an underflow or zero divide in some of the heuristics due to an order reduction near 9600 because the solution is only one time continuously differentiable there. Switching to ODE23 we had no difficulty. This illustrates the *desirability of having more than one integrator available for the user to call in the utility.*

Figure 1 shows the grid gotten using ODE23 and our algorithm using arclength, with $\epsilon_{\text{init}} = 10^{-1}$ using $\alpha = \beta = 1$. The lower grid shows the time steps for ODE23. The computed top grid is uniform and essentially the same size as the grid we used with the SOCS defaults.

We can emphasize putting points where something happens by weighing the inactive periods less heavily by reducing α. Figure 2 shows the result for our preferred value

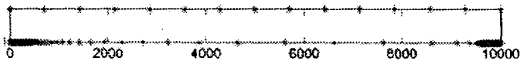

Fig. 1. ODE 23 grid and standard algorithm grid for Example 1.

of $\alpha = 10^{-6}$ while Figure 3 gives the grid gotten using $\alpha = 10^{-10}$. Figure 3 is not desirable either because the discretization requires some points throughout the interval.

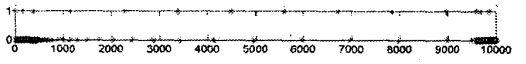

Fig. 2. ODE 23 grid and computed grid for $\alpha = 10^{-6}$ for Example 1.

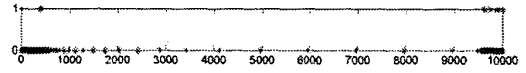

Fig. 3. ODE 23 grid and initialization grid with $\alpha = 10^{-10}$ for Example 1.

While SOCS had no difficulty in solving this problem using the current default initialization it is interesting to compare what happens if we take the grids coming from our initialization utility to what happens with using grids from the default initialization option of SOCS. The solution of the optimal control problem using the defaults and the utility grid took 11 and 12 iterations and had final grids of 216 and 205 respectively. However, the CPU time was reduced from 26.67 to 14.6. CPU results must always be viewed cautiously. Examination of the data suggests that this reduction is coming from having to do less work on each iteration. Even though the choices of α and the value of 400 were based on nothing more than the idea something happens infrequently and at the ends, and no analysis was done to optimize these choices, the result was reduced CPU time because of quicker iterations. Also the ODE error, ERRODE, was reduced much more quickly at first with the new guess.

2) Example 2: Singularities and highly nonlinear: Our second test problem is known as the Pleiades problem [6]. It consists of seven stars moving in a coordinate plane. They each have mass m_i and location x_i, y_i. The only force acting on them is gravitational attraction which obeys the inverse square law. The stars are considered to be point masses. Let $r_{i,j} = (x_i - x_j)^2 + (y_i - y_j)^2$ be the square of the distance between stars i and j. The system is

$$x' = v \tag{9a}$$

$$y' = w \tag{9b}$$

$$v' = f(x, y) \tag{9c}$$

$$w' = g(x, y) \tag{9d}$$

$$f_i = \sum_{j \neq i} m_j \frac{x_j - x_i}{r_{i,j}^{3/2}} \qquad g_i = \sum_{j \neq i} m_j \frac{y_j - y_i}{r_{i,j}^{3/2}} \tag{9e}$$

As in [6] we take $m_i = i$, the initial conditions given in [6], and the time interval [0, 2].

There are 28 state variables and the equations are highly nonlinear. Numerical integration of the solution shows that trajectories for these initial conditions exist on $[0, 2]$, but there are several near collisions around 1.22, 1.44, 1.62, 1.67, where the distance squared gets as low as 10^{-3}. At these near collisions the acceleration gets high, so that like many space missions, there are long periods of relatively slow motion and periods of rapid motion. Because the problem is planar the singularities can pose extra difficulties. If an approximation starts on one side of a star, and the true solution is on the other side of the star, then singularities may be encountered during grid refinement. Figure 4 shows the trajectories. The numbers indicates which star and where the trajectories start.

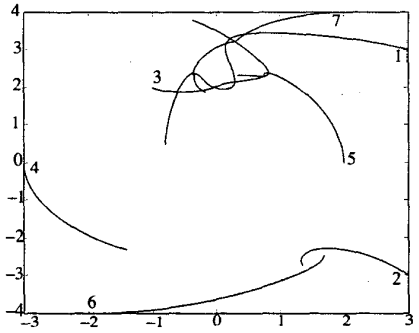

Fig. 4. All trajectories for Example 3 on [0, 2].

There are no control variables. Thus when we pass the problem to SOCS with an initial condition there are no degrees of freedom. This is an important case that can occur when inequality constraints are active. It is also a problem that a user of SOCS might want to solve to examine their formulation. This is also a good test problem since the usual initialization routine in SOCS failed to get a feasible solution when started on moderate sized or small grids. The case $N = 10$ was an outlier where SOCS found a feasible solution but then could not get feasibility on the second iteration. The monitor function enabled us to get a consistent initialization but only when we used the higher order HS rather than the usually preferable TR.

Since we knew from simulation that there were near collisions we generated test grids using ODE45 with a requested ODE tolerance of 10^{-6}. The integrator took 240 time steps. We then generated initial grids for different tolerances, or equivalently, for different requested values of N. At a requested initialization tolerance of 10^{-4} and using a fourth order initialization discretization, we got the time grid in Figure 5 which turned out to be a good grid provided HS was used. With $N = 66$ and higher SOCS was able to find a feasible solution and iterate to convergence provided that we used HS as the initialization discretization.

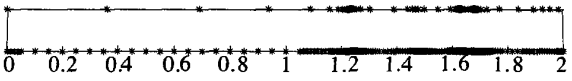

Fig. 5. Successful grid using $N = 77$ for Example 3.

TABLE I
SUCCESSFUL INITIALIZATION OF EXAMPLE 3 USING HS.

N	NPT	Iterations	Initial CPU	Total CPU
66	509	4	440	2359.39
77	464	4	180	189.55
96	381	3	95	924.39
117	465	3	74	593.28
135	484	3	70	525.79

The code was not able to find a feasible solution if the default initialization was used or if the TR was used even if initialization iterations were allowed to run for a long time. Table I provides a summary of the successful runs. All used the monitor function grid. NPT is the final grid found by SOCS which gave ODETOL of 10^{-7}.

C. Alternative monitor functions

It has been observed in the moving grid literature that it might be desirable to choose the grid to equidistribute the error in the approximation rather than to equidistribute along the solution graph. One choice is to try and pick a grid to minimize the error of linear interpolation [3]. The more difficult case where there are time and spatial grids is discussed in [7]. Minimizing the error of linear interpolation leads to a different family of monitor functions of the form

$$\tilde{\phi}(y,t) = \gamma + \|f\|^{\frac{1}{m}} = \gamma + \left(\sum_{i=1}^{n} |f_i|^2\right)^{\frac{1}{2m}} \tag{10}$$

One choice of γ is

$$\gamma = \frac{1}{t_f - t_0}\int_{t_0}^{t_f} \|f\|^{\frac{1}{m}} dt \tag{11}$$

We shall refer to (10), (11) as monitor2. Suppose that we have the exact solution y of the differential equation and a time grid $\{t_i\}$ that comes from a monitor function. Suppose that we have a piecewise linear interpolation $\overline{y}$ to y on this grid. The grid has N subintervals. The question is whether the grid from (10) leads to a better uniform approximation and what the consequences, if any, of this are. The argument in [3] was for a specific test problem. By modifying that argument we can prove a similar result in an asymptotic sense in Lemmas 1–3. [The argument in [3] was not asymptotic.]

Lemma 1: Suppose that y is a smooth function defined on $[0, T]$. Let δ be a nonnegative continuous function defined on the range of y. Let $\Gamma = \int_0^T \delta(y(t))dt$. Define the monitor function ϕ by

$$\phi(y) = \frac{\rho\Gamma}{T} + \delta(y) \tag{12}$$

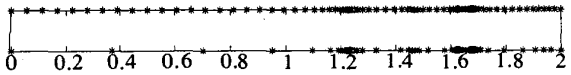

Fig. 6. Grids on iteration 1 with $N = 77$.

where $\rho > 0$ is a parameter. Let $\Delta_k = t_{k+1} - t_k$. If (12) is used to equidistribute the grid, then $\Delta_k \leq \frac{(1+\rho)}{\rho}\frac{T}{N}$. Earlier we had only indirect control of the largest subinterval in the grid. The parameter ρ in (12) gives us direct control on the grid spacing since it sets an upper bound on the length of the longest grid interval.

Lemma 2: Let $L = \int_0^T \sqrt{1 + \|y'\|^2}dt$ and $\rho > 0$. Suppose that the modified arc length monitor function $\phi = \frac{\rho L}{T} + \sqrt{1 + \|y'\|^2}$ is used to generate a grid on $[0, T]$. Then the L^∞ difference between a smooth function y and its linear interpolation $\bar{y}$ on that grid is $\|y - \bar{y}\|_\infty \leq \frac{2L}{N}(1 + \rho)$,

Lemma 3: Suppose that $\Gamma = \int_0^T \|y'\|^{1/m}dt$. Consider $\phi = \frac{\rho\Gamma}{T} + \|y'\|^{1/m}$ with $m = 2$. Then the L^∞ difference between a smooth function y and its linear interpolation $\bar{y}$ on a grid from the monitor function is asymptotically $\|y - \bar{y}\|_\infty \leq \frac{C}{N^2}$.

1) Computational Comparison: We examined the behavior of the new monitor function in comparison to that of the arclength monitor function on the Pleiades problem. Since we already knew that the arclength monitor function worked well at NPT =77, we began with NPT=77 and $\rho = 1$. The new monitor2 function did not work nor did it work at a somewhat higher value of NPT. This was not too surprising, since with $\rho = 1$, we had $\Delta_k \leq 4/77$ so that a quarter of the points were forced in the first half of the interval and fewer grid points were being used for the difficult second half.

The first value of ρ where we got a feasible solution with $N = 77$ was at $\rho = 1/7$ so that $\Delta_k \leq \frac{1+\frac{1}{7}}{\frac{1}{2}}\frac{2}{77} = \frac{16}{77}$. The grids which found feasibility are given in Figure 6. The top grid is with the new monitor function and $\rho = 1/7$. The bottom grid is from the arc length monitor function. The arc length grid is much more focused on the second interval. SOCS was allowed to run to convergence for both of these grids. Table III gives the results using monitor2. Both of these initial grids led to convergence in only four iterations. Both initial grids gave comparable ERRODE. However, the new monitor function produced a final grid that was 31%.

smaller and a CPU time that was half as much. Whether this is an artifact of this problem or a more general phenomena is unclear but it is clearly worth investigating more carefully. It is possible that since the new monitor function is more closely linked to the error estimate that it does a better job of equidistributing on the early grids.

2) Modification of monitor2: The computational results suggest that maybe the new monitor function is not really distributing the grid according to the ERRODE as well as expected since it had to go back and put more points near the closest near collisions. To understand why this might be the case we return to the proof of Lemma 3 and note

TABLE II

77 NPT INITIAL ARC LENGTH GRID AND RESULTING ITERATION ON PLEIADES.

GRID	NPT	NFE	NRHS	ERRODE	CPU
1.0	77	1250	191250	0.30E-03	180
2.0	153	4799	1463695	0.36E-05	1400
3.0	305	410	249690	0.15E-06	260
4.0	466	116	107996	0.18E-07	190
	466	6575	2012631		1,986

TABLE III

77 NPT INITIAL MONITOR2 GRID. $k = 7$, AND RESULTING ITERATION ON PLEIADES.

GRID	NPT	NFE	NRHS	ERRODE	CPU
1.0	77	2390	365670	0.35E-03	340
2.0	153	889	271145	0.61E-05	270
3.0	166	116	38396	0.12E-05	50
4.0	331	291	192351	0.67E-07	220
	331	3686	8657562		878

that there is the constant M which is a bound for f_y. For purposes of discussion we can think of the Pleiades problem as being modeled by a scalar differential equation $y' = f(y) = \frac{1}{(y+\epsilon)^2}$. Then $|f_y| = 2\frac{1}{|y+\epsilon|^3} = 2|y'|^{3/2}$. Thus if we are using the new monitor function we see that on subintervals which include times close to the closest collision that the error estimate and M become very large. Can we modify the monitor function so that we get it second order without the large bound?

Suppose we know that our problem has the additional bound that $\|f_y\| \leq \kappa\|y'\|^\alpha$. Then modifying the proof of Lemma 3 gives $\|y(t) - \bar{y}(t)\| \leq \frac{C_2}{N^2}$ if we take $\bar{\Gamma} = \int_0^T \|y'\|^{\frac{1+\alpha}{2}}dt$ and use the monitor function $\phi = \|y'\|^{\frac{1+\alpha}{2}} + \frac{\rho\bar{\Gamma}}{T}$. Then the previous discussion suggests that for the Pleiades problem we should choose $\alpha = \frac{3}{2}$ so that

$$\bar{\Gamma} = \int_0^T \|y'\|^{\frac{5}{4}}dt, \ \phi = \|y'\|^{\frac{5}{4}} + \frac{\rho\bar{\Gamma}}{T} \tag{13}$$

To test these ideas we again consider the Pleiades problem but using the monitor function given by (13) which we will refer to as monitor3. We considered $\rho = 1/k$ with $k = 1, 3, 7, 9$. It appears to be giving a feasible solution of comparable order to that of monitor2 and the size of the final grids is comparable. However, it takes more CPU time due to a very expensive second iteration. The cause of this is not clear.

D. Computational Comments

For sensitive problems the *integration in the various monitor functions must be done carefully*. In fact, at first we thought we had a programming error in that when we went to generate the t grid the value of t_f was sometimes off by 15% on intervals like $[0, 3]$ or $[0, 4]$. This is turn can lead to the clustering of grid points near a sensitive spot being misplaced. A look at the s values shows why. s was computed

using ODE45 and an Abstol of 1e-8 on [0 3]. The corresponding graph of $t(s)$ on the second integration is shown in Figure 7. It is giving a value of $t_f \approx 3.3$ instead of 3. Reflecting the graph of Figure 7 gives the graph in Figure 8b. This should be the same as Figure 8a but it is not. The difficulty occurs in two places. The rapid rise in $s(t)$ introduces some error. But there are also problems in the second integration. The long flat middle part encourages the integrator to be overly aggressive in maintaining a large stepsize. These difficulties were overcome by replacing AbsTol by RelTol and more importantly putting in a MaxStep bound of 0.1. This forced the integrators to use smaller stepsizes.

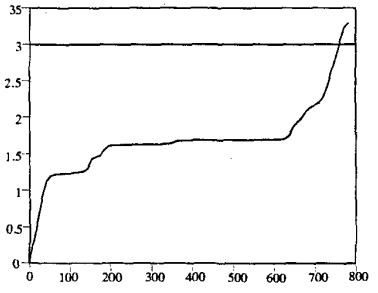

Fig. 7. Graph of $t(s)$ for monitor 3 with $k = 9$.

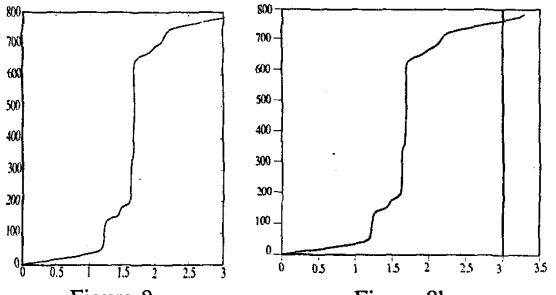

Fig. 8. Reflected Graphs

In Figure 9 we set $\Gamma = \overline{\Gamma} = 0$ and scaled the magnitude so that all three monitor functions gave the same value of s_f. The dashed line is $\|y'\|^{1/2}$ from monitor2, the solid line is the arclength and the line with the small squares on it is $\|y'\|^{5/4}$ from monitor3.

III. Conclusion

Initialization of direct transcription codes is a major difficulty in many complex optimization problem. This paper has examined the use of monitor functions with the goal of developing initialization utilities. While monitor functions can be very helpful, and are sometimes essential, we have seen that one monitor function does not suffice. A good initialization utility needs to have a family of monitor functions,

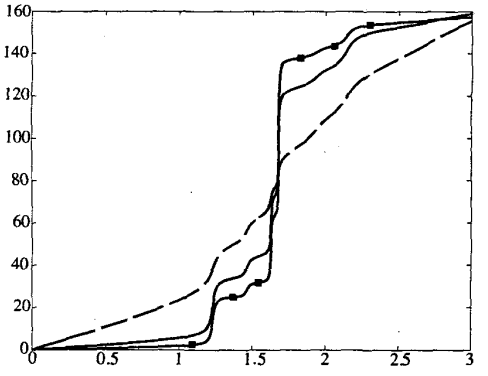

Fig. 9. The three measures of y'.

the capability of the user to specify an initialization tolerance, a family of integrators to generate the grids, and a choice of different order initialization discretizations.

IV. Acknowledgements

Research supported in part by NSF Grants DMS-0101802 and ECS-0114095.

V. References

[1] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, SIAM, Philadelphia, 1995.

[2] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, 2001.

[3] G. Beckett, J. A. Mackenzie, A. Ramage, and D. M. Sloan, *On the numerical solution of one-dimensional PDEs using adaptive methods based on equidistribution*, J. Comp. Physics, 167 (2001), 372-392.

[4] J. T. Betts and S. O. Erb, *Optimal low thrust trajectories to the moon*, SIAM J. Appl. Dynamical Systems, 2 (2003), 144-170.

[5] J. T. Betts and W. P. Huffman, *Sparse Optimal Control Software SOCS*, Mathematics and Engineering Analysis Technical Document MEA-LR-085, Boeing Information and Support Services, July, 1997.

[6] E. Hairer, S. P. Norsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer-Verlag, second revised edition, 1993.

[7] W. Huang, *Variational mesh adaptation: isotropy and equidistribution*, J. Comp. Physics 174, (2001), 903-924.

[8] A. V. Rao and K. D. Mease, *Eigenvector approximate dichotomic basis method for solving hyper-sensitive optimal control problems*, Optimal Cont. Appl. Methods, 20 (1999), 59-77.