

# ENERGY MANAGEMENT FOR TRAINS ON HILLS

ROBERT J. VANDERBEI

Operations Research and Financial Engineering  
Princeton University

Revised October 28, 2000

ABSTRACT. In this case study, we consider how to operate a train efficiently.

## 1. INTRODUCTION

In this case study we consider a trajectory optimization problem: how to drive a train so as to minimize fuel costs.

We express our optimization models in the AMPL modeling language [3]. This language provides a common mechanism for conveying problems to codes to solve them. When solving problems we generally use two different solvers: (a) LOQO [6, 7, 8, 1], which implements an interior-point method for general nonlinear optimization and (b) SNOPT [4], which implements an active set strategy with a quasi-Newton method for the QP subproblem.

## 2. TRAINS

An important problem in transportation is to minimize fuel costs in the operation of a train. To keep things simple, we consider a segment of track that is straight although it may contain hills and valleys. Let  $x$  denote position along the track measured from some fixed reference point. Letting  $v$  denote the derivative of position with respect to time and  $a$  the time-derivative of  $v$ , we arrive at the following equations describing the motion of the train:

$$\begin{aligned} v &= \dot{x} \\ a &= \dot{v} \\ (1) \quad a &= h(x) - (a + b|v| + cv^2) + u_a - u_b. \end{aligned}$$

---

*Date:* October 28, 2000.

*1991 Mathematics Subject Classification.* Primary 65L10 Secondary 34B15.

*Key words and phrases.* trajectory optimization, optimal control, constrained optimization.

Research supported by NSF grant DMS-9870317, ONR grant N00014-98-1-0036.

Here,  $h(x)$  represents the acceleration/deceleration caused by going down/up hills,  $a$ ,  $b$ , and  $c$  are constants so that the three terms  $a + b|v| + cv^2$  represent friction (both from the track and from the surrounding air),  $u_a$  represents the acceleration provided by the engines, and  $u_b$  represents the deceleration from applying the brakes. The control variables are the functions  $u_a$  and  $u_b$ . The objective is to take the train from one place given by initial condition

$$\begin{aligned}x(0) &= x_0 \\v(0) &= v_0\end{aligned}$$

to another given by

$$\begin{aligned}x(T) &= x_f \\v(T) &= v_f\end{aligned}$$

in such a way as to minimize fuel costs, which we take to be proportional to the total amount of work done:

$$\int_0^T u_a(t)v(t)dt.$$

To get a specific instance of this problem, we take the initial position to be zero, the final position to be 6.0 km, the initial and final velocities to be zero, the total trip time to be 4.8 minutes and

$$a = 0.3, \quad b = 0.14, \quad c = 0.16.$$

Finally, the hill function  $h$  is taken to be

$$h(x) = \sum_{j=1}^{m-1} (s_{j+1} - s_j) \frac{1}{\pi} \tan^{-1} \frac{x - z_j}{\epsilon}$$

where  $m$  represents the number of hill sections,  $s_j$  is the slope along the  $j$ -th section,  $z_j$  is the breakpoint between the  $j$ -th and the  $j + 1$ -st section, and  $\epsilon$  gives a spread which is related to the length of the train itself. Our specific choice involves an initial uphill climb followed by a level section and then a final downhill run. Hence, it has  $m = 3$  and

$$\begin{aligned}z_1 &= 2, & z_2 &= 4, \\s_1 &= 2, & s_2 &= 0, & s_3 &= -2.\end{aligned}$$

A plot of  $h$  is shown in Figure 1.

**2.1. Midpoint Discretization Method.** This problem can be cast as a (nonconvex) nonlinear optimization problem by discretizing the time interval  $[0, T]$  into  $N$  small time intervals and writing discrete approximations for the derivatives that appear in the model. There are many ways to do this. In this paper, we discuss two popular discretizations: midpoint discretization and trapezoidal discretization. We begin with the midpoint method. Letting  $x[j]$  denote the value of  $x$  at time  $jT/N$ ,  $j=0, 1, \dots, N$ , we define a discrete approximation to the velocity at the midpoint of each time interval as follows:

$$v[j+0.5] = (x[j+1] - x[j]) / (T/N) \quad j=0, 1, \dots, N-1,$$

The discrete approximation for acceleration is defined similarly:

$$a[j] = (vx[j+0.5] - vx[j-0.5]) / (T/N) \quad j=1, \dots, N-1,$$

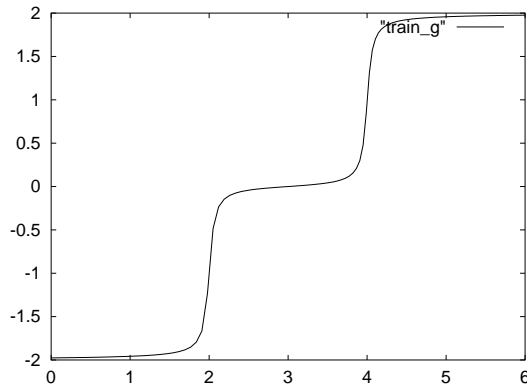


FIGURE 1. The acceleration profile caused by hills. The first two miles are uphill, then there are two miles of flat, and the last two miles are downhill.

This approximation is called *midpoint discretization*. The equations of motion given by (1) can then be written as:

$$a[j] = h(x[j]) - (a+b*v[j]+c*v[j]^2) + u_a[j] - u_b[j] .$$

Of course, velocities are defined at the half-integer points yet we access values here at the whole integer points. Hence, it is necessary to provide reasonable values for velocities at these integer points; we use the average of the two nearest half-integer values. The complete model expressing the midpoint discretization in the AMPL modeling language is shown in Figure 2.

**2.2. Ringing.** A graph showing  $x$ ,  $v$ , and  $a$  as functions of time is shown in Figure 3. Note the “ringing” phenomenon apparent in the acceleration during times of medium acceleration. Such a phenomenon suggests that something is wrong. To test whether it is a bug in the optimization algorithm, we solved the problem with two completely different solvers: LOQO and SNOPT. Both solvers produced similar ringing. We conclude that ringing is intrinsic to the model. Next, we refined the discretization to  $N = 501$  and solved the problem again. This time, LOQO and SNOPT both exhibited ringing but it was much more pronounced in LOQO. The objective functions matched out to the 8 digits of accuracy requested by the two solvers. (For those who like to keep score, LOQO solves the  $N=501$  problem in 6.86 seconds and SNOPT requires 35.71 seconds to solve the problem.)

Reflection sheds some light on what is happening. With a highly refined partition, a control scheme that alternates between two values becomes indistinguishable from one that applies the average of the two all the time. Imagine riding in a car with someone who pumps the gas pedal. The speed remains essentially constant and the rate of consumption of fuel is just the average of the pumped and unpumped rates. Hence, other than making passengers sick (I’ve been there), this control is just as good as a smooth one.

This reasoning suggests that the set of near optimal solutions is large. Sometimes interior-point methods get into trouble in such cases. In fact, trouble is assured if the set of optimal solutions is unbounded. For the train problem, setting  $N=1001$  LOQO finds a solution that is accurate only to 3 digits whereas SNOPT still can get 8 digits (although it takes a long time). Larger values of  $N$  cause even more trouble. Clearly ringing is bad for interior-point methods.

<pre> param N := 201; param time := 4.8; param length := 6.0; param ns := 3; param z{1..ns-1}; param s{1..ns}; param h := time/N; param uamax := 10.0; param ubmax := 2.0; param aa:= 0.3; param bb := 0.14; param cc := 0.16; param eps := 0.05; param pi := 4*atan(1);  var x{0..N}; var v{i in 0..N-1} = (x[i+1]-x[i])/h; var v_avg{i in 1..N-1}     = (v[i]+v[i-1])/2; var a{i in 1..N-1} = (v[i]-v[i-1])/h; var ua{1..N-1} &gt;=0.0, &lt;=uamax, :=0.0; var ub{1..N-1} &gt;=0.0, &lt;=ubmax, :=0.0; var u {i in 1..N-1} = ua[i]-ub[i];  minimize energy:     sum {i in 1..N-1} ua[i]*v_avg[i]*h; </pre>	<pre> s.t. newton {i in 1..N-1}:     h*a[i] =     h*     (         - sum {j in 1..ns-1}             (s[j+1]-s[j])*             atan((x[i]-z[j])/eps)/pi         - aa - bb*v_avg[i] - cc*v_avg[i]^2         + u[i]     );  s.t. x_init: x[0] = 0; s.t. x_finl: x[N] = length; s.t. v_init: v[0] = 0; s.t. v_finl: v[N-1] = 0;  data; param z := 1 2.0 2 4.0; param s := 1 2.0 2 0.0 3 -2.0;  solve;  printf {i in 0..N}: "%10f %10f \n",     i*h, x[i] &gt; train_x; printf {i in 1..N-1}: "%10f %10f \n",     i*h, u[i] &gt; train_a; printf {i in 0..N-1}: "%10f %10f \n",     i*h, v[i] &gt; train_v; </pre>
--	--

FIGURE 2. The AMPL model trainh.mod.

**2.3. Smoothing.** The model can be improved by adding to the objective some measure of the work of “pumping the pedal”. For example, after adding a tiny correction

$$0.00000001 * \sum \{i \text{ in } 1..N-2\} (u[i+1] - u[i])^2/h$$

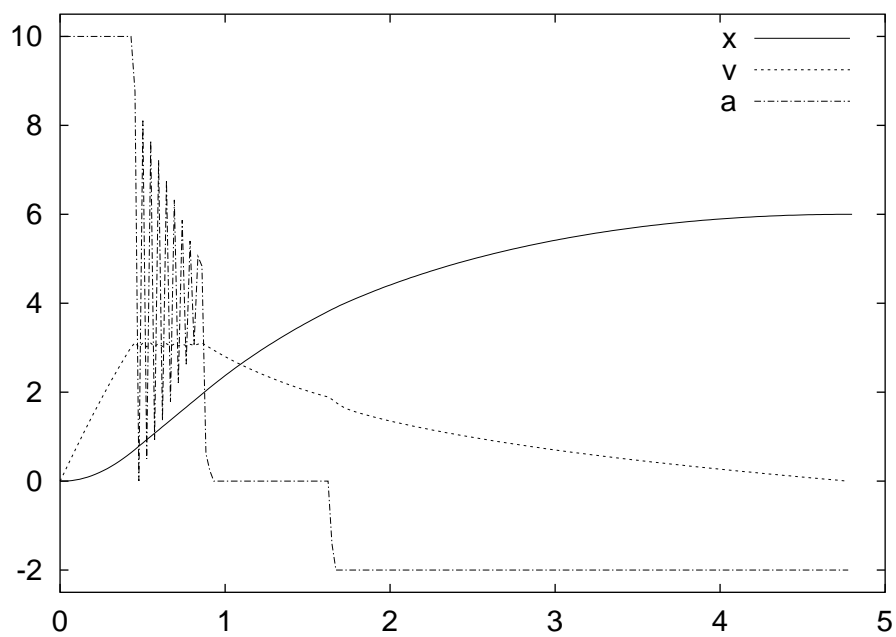
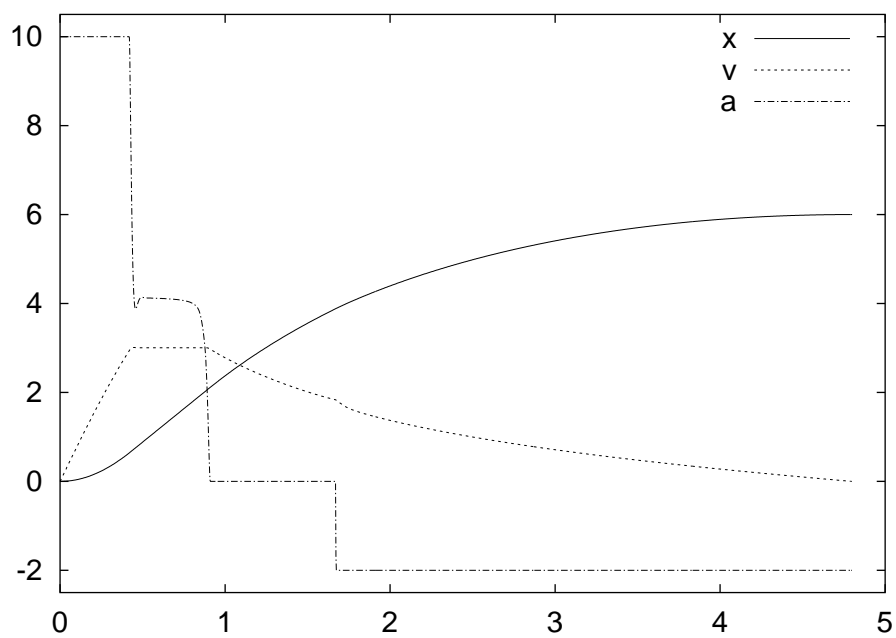
to the objective function in the model shown in Figure 2, LOQO has no trouble solving the  $N=1001$  case and gets an answer that exhibits no ringing whatsoever. The solution to the  $N=2001$  case is shown in Figure 4.

Rather than the first-order smoothness condition given above, one could use a second-order smoothness condition:

$$0.0000000001 * \sum \{i \text{ in } 2..N-2\} (u[i+1]+u[i-1]-2*u[i])^2/h^3$$

LOQO is also able to solve this variant. The solution is shown in Figure 5. It is very similar to the previous smoothed solution, which gives us confidence that we are finding the correct answer to the problem.

We end here our discussion of ringing by remarking that this phenomenon is common; it has nothing to do with the shape of the hills/valleys or with the initial and final conditions. In fact, one can see ringing even in the case where the entire track is level (i.e.,  $h \equiv 0$ ) and the initial and final velocities are both equal to the total distance divided by the total time. In this case, the optimal control should be static. That is, one

FIGURE 3. Output produced with  $N = 201$ .FIGURE 4. LOQO output produced with  $N = 2001$ .

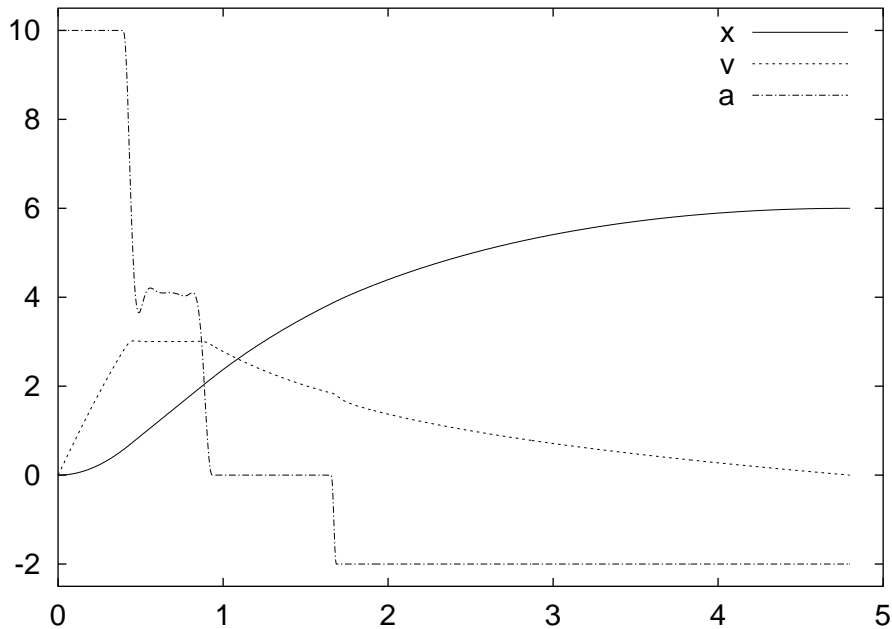


FIGURE 5. LOQO output produced with  $N = 2001$ .

should provide just the amount of acceleration needed to maintain the initial speed throughout the trip. But, without the smoothing terms mentioned above, both LOQO and SNOPT find nonstatic ringing-type solutions.

**2.4. Trapezoidal Discretization.** We end this section with a description of the second common method for discretizing first-order differential equations. This method is called the *trapezoidal discretization*. With this discretization, values for  $v$  and  $a$  are defined at the same discrete times as for  $x$ ; that is, at  $jT/N$ ,  $j=0, 1, \dots, N$ . Instead of giving a formula defining each velocity in terms of a difference of positions, we give constraints that say that the average value of the values of  $v$  at two adjacent times is equal to the appropriate difference in the positional values:

$$(v[j+1]+v[j])/2 = (x[j+1]-x[j])/(T/N) \quad j=0, 1, \dots, N-1,$$

Constraints that must be satisfied by the accelerations are similar:

$$(a[j+1]+a[j])/2 = (v[j+1]-v[j])/(T/N) \quad j=0, 1, \dots, N-1,$$

The model in its entirety is shown in Figure 6. Generally speaking trapezoidal discretizations are more popular than their midpoint counterparts, but there are drawbacks.

First of all, for the train models that we are considering the midpoint method is less affected by the ringing phenomenon. This is seen from the fact that the  $1.0e-8$  factor used in the midpoint method has to be increased to  $1.0e-6$  in the trapezoidal method before LOQO can solve the model. Furthermore, even with this larger smoothing factor, the midpoint model solves in 83 iterations whereas the trapezoidal model requires 187.

<pre> param N := 2001; param time := 4.8; param length := 6.0; param ns := 3; param z{1..ns-1} ; param s{1..ns} ; param h := time/N; param uamax := 10.0; param ubmax := 2.0; param aa:= 0.3; param bb := 0.14; param cc := 0.16; param eps := 0.05; param pi := 4*atan(1);  var x{0..N}; var v{0..N}; var a{0..N}; var ua{0..N} &gt;= 0.0, &lt;= uamax, := 0.0; var ub{0..N} &gt;= 0.0, &lt;= ubmax, := 0.0; var u {i in 0..N} = ua[i] - ub[i];  minimize energy:   sum {i in 0..N} ua[i]*v[i]*h   + 0.000001*sum {i in 0..N-1}       (u[i+1] - u[i])^2/h; </pre>	<pre> s.t. v_def {i in 0..N-1}:   (v[i+1]+v[i])/2 = (x[i+1]-x[i])/h;  s.t. a_def {i in 0..N-1}:   (a[i+1]+a[i])/2 = (v[i+1]-v[i])/h;  s.t. newton {i in 0..N}:   h*a[i] =   h*   (   - sum {j in 1..ns-1}     (s[j+1]-s[j])*atan((x[i]- z[j])/eps)/pi   - aa - bb*v[i] - cc*v[i]^2   + u[i]   );  s.t. x_init: x[0] = 0; s.t. x_finl: x[N] = length; s.t. v_init: v[0] = 0; s.t. v_finl: v[N] = 0;  data; param z := 1 2.0 2 4.0; param s := 1 2.0 2 0.0 3 -2.0;  solve;  printf {i in 0..N}: "%10f %10f \n",   i*h, x[i] &gt; train_x; printf {i in 0..N}: "%10f %10f \n",   i*h, v[i] &gt; train_v; printf {i in 0..N}: "%10f %10f \n",   i*h, u[i] &gt; train_a; </pre>
--	--

FIGURE 6. The AMPL model `trainh_trap.mod` illustrating the trapezoidal discretization.

The second disadvantage to using trapezoidal discretizations involves the minimal number of variables/constraints needed to express the model. The variables representing derivatives ( $v$  and  $a$  in the model in question) must be explicitly represented in the model and are determined only indirectly via the constraints they must satisfy. With the midpoint discretization there is more flexibility. These variables can be treated in the same explicit way, i.e., represented explicitly and then defined via constraints. But, they can also be given simply as “abbreviations” for their explicit formulas in terms of the undifferentiated variables (i.e., position) and then never be seen by the optimization algorithm. The model shown in Figure 2 uses this latter approach. It results in many fewer variables and constraints. While reducing the number of variables and/or constraints in a large-scale sparse optimization problem does not always mean faster solution times, it often does and that is the case here. For  $N=2001$ , the midpoint model has 5998 variables, 2000 constraints, and solves in 86 seconds (using 83 iterations of the basic algorithm). On the other

hand, the trapezoidal model has 10006 variables, 6004 constraints, and solves in 355 seconds (using 187 iterations). More iterations are needed because, as stated earlier, this method suffers more from ringing but even on a per iteration basis the midpoint model solves twice as fast.

The trapezoidal discretization of the train model that we've studied here derives from the model `trainh` in the CUTE [2] suite on test problems. The CUTE model was itself adapted from a paper by Kautsky and Nichols [5].

**2.5. Lessons.** After studying hundreds of nonlinear optimization problems, we have learned many lessons about how to formulate models appropriately and what type of algorithm will solve these problems efficiently and robustly. While a single example is not sufficient for deducing these lessons, it can be used to illustrate them. The lessons illustrated by the train problem can be summarized as follows:

- (1) Discrete approximations to continuous problems can exhibit unexpected pathological behaviour such as the ringing we saw here.
- (2) Optimization problems with large sets of optimal (or nearly optimal) solutions can present numerical difficulties for interior-point methods.
- (3) Interior-point methods are often more efficient than active-set methods on large problems.
- (4) Midpoint discretizations have fewer degrees of freedom than trapezoidal discretizations and therefore are less likely to exhibit ringing.
- (5) With midpoint discretization one can eliminate the higher-order derivatives from the optimization model producing a reduced model that may solve more efficiently than the expanded version.

## REFERENCES

- [1] H.Y. Benson, D.F. Shanno, and R.J. Vanderbei. Interior-Point Methods for Nonconvex Nonlinear Programming: Jamming and Comparative Numerical Testing. Technical Report ORFE-00-2, Dept. of Operations Research and Financial Engineering, Princeton University, Princeton NJ, 2000. Submitted to *Math. Prog.* 1
- [2] I. Bongartz, A.R. Conn, N. Gould, and Ph.L. Toint. Constrained and unconstrained testing environment. <http://www.cse.clrc.ac.uk/Activity/CUTE+74>. 8
- [3] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press, 1993. 1
- [4] P.E. Gill, W. Murray, and M.A. Saunders. User's guide for SNOPT 5.3: A Fortran package for large-scale nonlinear programming. Technical report, Systems Optimization Laboratory, Stanford University, Stanford, CA, 1997. 1
- [5] J. Kautsky and N.K. Nichols. OTEP-2: Optimal train energy programme, mark 2. Technical Report Numerical Analysis Report NA/4/83, Dept. of Mathematics, University of Reading, 1983. 8
- [6] R.J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 12:451–484, 1999. 1
- [7] R.J. Vanderbei. LOQO user's manual—version 3.10. *Optimization Methods and Software*, 12:485–514, 1999. 1
- [8] R.J. Vanderbei and D.F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999. 1

ROBERT J. VANDERBEI, PRINCETON UNIVERSITY, PRINCETON, NJ