CASE STUDIES IN TRAJECTORY OPTIMIZATION: TRAINS, PLANES, AND OTHER PASTIMES

ROBERT J. VANDERBEI

Operations Research and Financial Engineering
Princeton University
ORFE-00-3

Revised July 27, 2000

ABSTRACT. This is the first in a series of papers presenting case studies in modern large-scale constrained optimization, the purpose of which is to illustrate how recent advances in algorithms and modeling languages have made it easy to solve difficult optimization problems using off-the-shelf software. In this first paper, we consider four trajectory optimization problems: (a) how to operate a train efficiently, (b) how to putt a golf ball on an uneven green so that it arrives at the cup with minimal speed, (c) how to fly a hang glider so as to maximize or minimize the range of the glide, and (d) how to design a slide to make a toboggan go from beginning to end as quickly as possible.

In addition to the tutorial aspects of this paper, we also present evidence suggesting that the widely used trapezoidal discretization method is inferior in several ways to the simpler midpoint discretization method.

1. INTRODUCTION

This is the first in a series of papers presenting case studies in constrained optimization. The purpose of these studies is to illustrate how recent advances in algorithms and modeling languages now make it easy to solve once difficult optimization problems using off-the-shelve software. A secondary goal is to show that it is nonetheless still possible to make subtle errors in a model which will render it (a) more difficult than it needs to be or (b) infeasible or, worse, (c) feasible but giving the wrong answer. In the past, many of the optimization problems we present here were thought to be very difficult to solve and it was

Date: July 27, 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.

unclear whether failures were due to bad algorithms or bad models. Today, one can say that failures are almost always due to bad models.

In this paper we consider trajectory optimization problems. Our first example is about how to drive a train so as to minimize fuel costs. We follow this with two examples from the world of sports: golfing and flying. Subsequent papers in this series will treat applications in electrical engineering (filter and antennae-array design) and in civil engineering (topology optimization of structures).

Throughout the paper we present several optimization models. We express these models in the AMPL modeling language [9]. This language provides a common mechanism for conveying problems to codes to solve them. When solving problems we generally use two different codes: (a) LOQO [13, 14, 15, 2], which implements and interior-point method for general nonlinear optimization and (b) SNOPT [10], which implements an active set strategy for solving these problems.

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:

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

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:

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

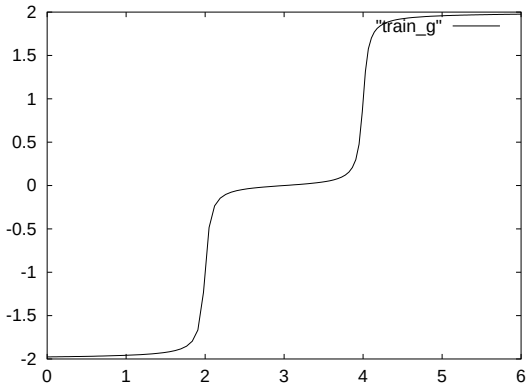

FIGURE 1. The acceleration profile caused by hills. The first two miles are uphill, then there are two miles of flat, and the last two miles are downhill.

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

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

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 ϵ 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

$$z_1 = 2, \qquad z_2 = 4,$$

$$s_1 = 2, \qquad s_2 = 0, \qquad s_3 = -2.$$

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,...,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)     j=0,1,...,N-1,
```

The discrete approximation for acceleration is defined similarly:

```
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} >=0.0, <=uamax, :=0.0;
var ub{1..N-1} >=0.0, <=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;
```

```
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] > train_x;
printf {i in 1..N-1}: "%10f %10f \n",
    i*h, u[i] > train_a;
printf {i in 0..N-1}: "%10f %10f \n",
    i*h, v[i] > train_v;
```

FIGURE 2. The AMPL model `trainh.mod`.

```
a[j] = (vx[j+0.5]-vx[j-0.5])/(T/N)    j=1,...,N-1,
```

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" phenomenom 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

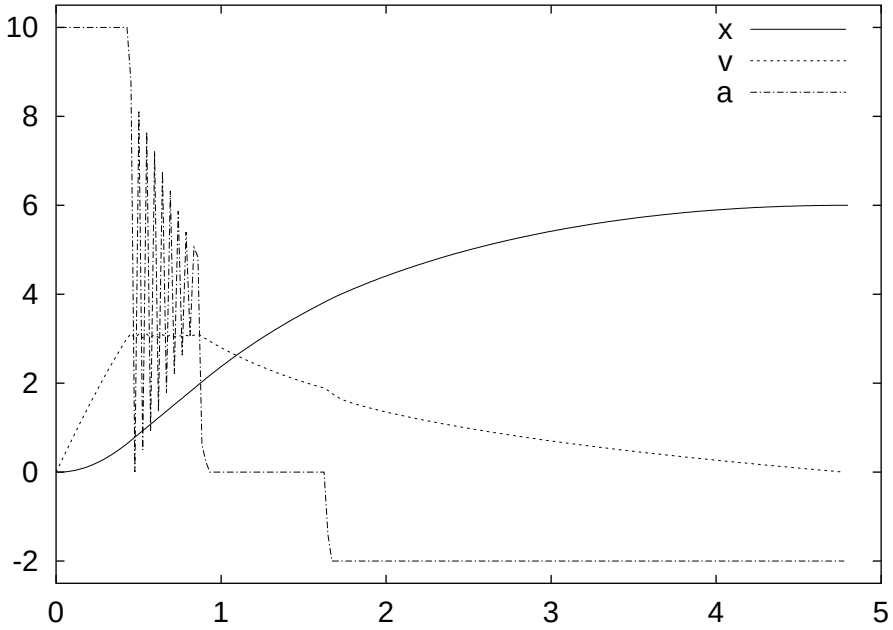

FIGURE 3. Output produced with `N = 201`.

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.

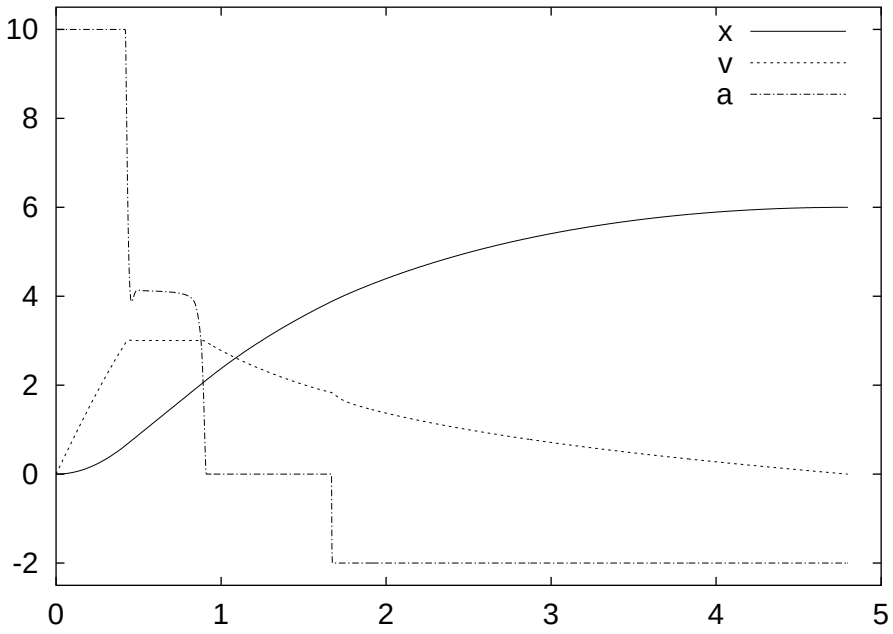

FIGURE 4. LOQO output produced with `N = 2001`.

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 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 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 should provide just the amount of acceleration needed to maintain the initial speed throughout the trip.

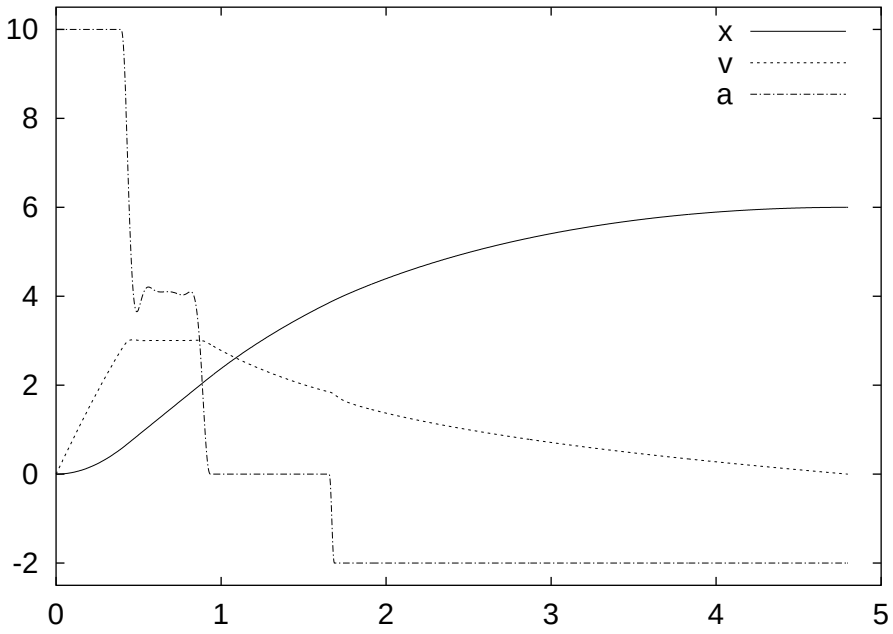

FIGURE 5. LOQO output produced with `N = 2001`.

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,...,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)     j=0,1,...,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)     j=0,1,...,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.

```
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} >= 0.0, <= uamax, := 0.0;
var ub{0..N} >= 0.0, <= 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;
```

```
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] > train_x;
printf {i in 0..N}: "%10f %10f \n",
    i*h, v[i] > train_v;
printf {i in 0..N}: "%10f %10f \n",
    i*h, u[i] > train_a;
```

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 [8] suite on test problems. The CUTE model was itself adapted from a paper by Kautsky and Nichols [11].

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.

3. PUTTING

The problem of how to putt provides a simple framework to continue our discussion of trajectory optimization. One of the lessons to be learned with this example is how easy it is to make a wrong model. With this in mind, we advise the interested golfer to read the entire section because the first model, right as it may appear, is wrong.

3.1. **The Alessandrini Model.** We begin with a discussion of the problem essentially as it appears in [1].

Given a golf ball sitting at rest on a putting green, the problem is to figure out how to hit the ball so that it will go into the cup. To make sure that it does not just skim over the cup and stop at some point far beyond, we try to have the ball arrive at the cup with the smallest momentum possible.

The Normal Vector. We assume that the elevation of the green is given as $(x, y, z(x, y))$ together and that its shape is given by $(x/a)^2 + (y/b)^2 \leq 1$. Two tangent vectors to the

surface are provided by $(1, 0, \partial z/\partial x)$ and $(0, 1, \partial z/\partial y)$. By taking the cross product of these two vectors, we obtain an upward pointing normal vector to the surface:

$$(-\partial z/\partial x, -\partial z/\partial y, 1).$$

The normal force N exerted by the surface of the green on the golf ball must point in this direction and its magnitude must be such that the total force in this direction vanishes (to keep the ball rolling on the surface).

The Normal Force. Since the only forces that are not tangential to the green are the force of gravity and the normal force itself, we must have the projection of the force of gravity on the normal direction be exactly opposite to the magnitude of the normal force:

$$-mg(e_z \cdot N)/\|N\| = -\|N\|,$$

where m is mass of the ball, g is acceleration due to gravity, e_z is the unit vector pointing in the vertical direction, and of course N is proportional to the normal vector given above. From this relation, we get that

$$N_z = \frac{mg}{(\partial z/\partial x)^2 + (\partial z/\partial y)^2 + 1}$$

and that

$$N_x = -\partial z/\partial x N_z \qquad N_y = -\partial z/\partial y N_z.$$

Friction. There is friction between the ball and the green. It is assumed to be proportional to the normal force and to point in a direction opposite to the velocity:

$$F = -\mu\|N\|\frac{v}{\|v\|}.$$

Equations of Motion. If we denote the trajectory by $u(t) = (x(t), y(t), z(t))$, then the equations of motion are

$$\begin{aligned} v &= \dot{u} \\ a &= \dot{v} \\ ma &= N + F - mge_z. \end{aligned} \tag{2}$$

Boundary Conditions. The initial and final positions are known,

$$u(0) = u_0 \qquad \text{and} \qquad u(T) = u_f,$$

but the time T at which the final position is reached is a variable.

As with the train example, this problem can be cast as a (nonconvex) nonlinear optimization problem using either a midpoint or a trapezoidal discretization rule. Inspired by our lesson from the previous example indicating some advantages to the midpoint rule, we start with this method. We discuss the trapzoidal rule at the end of this section.

Using the midpoint rule, we can let `x[j]`, `y[j]`, and `z[j]` denote the positional coordinates at time jT/N, `j=0,1,...,N`, and then define discrete approximations to the three components of velocity at the midpoint of each time interval as follows:

```
vx[j+0.5] = (x[j+1]-x[j])/(T/N)    j=0,1,...,N-1,
vy[j+0.5] = (y[j+1]-y[j])/(T/N)    j=0,1,...,N-1,
vz[j+0.5] = (z[j+1]-z[j])/(T/N)    j=0,1,...,N-1.
```

Discrete approximations for acceleration are defined similarly:

```
ax[j] = (vx[j+0.5]-vx[j-0.5])/(T/N)    j=1,...,N-1,
ay[j] = (vy[j+0.5]-vy[j-0.5])/(T/N)    j=1,...,N-1,
az[j] = (vz[j+0.5]-vz[j-0.5])/(T/N)    j=1,...,N-1.
```

The equations of motion given by (2) complete the constraints defining the model:

```
ax[j] = (Nx[j] + Fr_x[j])/m,
ay[j] = (Ny[j] + Fr_y[j])/m,
az[j] = (Nz[j] + Fr_z[j])/m - g.
```

Here, `Nx[j]`, `Ny[j]`, and `Nz[j]` are shorthand for

```
Nz[j] = m*g/(dzdx[j]^2 + dzdy[j]^2 + 1),
Nx[j] = -dzdx[j]*Nz[j],
Ny[j] = -dzdy[j]*Nz[j]
```

and `Fr_x[j]`, `Fr_y[j]`, and `Fr_x[j]` are shorthand for the three components of friction along the trajectory. Our first ampl model for this problem is shown in Figure 7. In this particular instance, the shape of the green involves two rather flat sections with a smooth ramp between them. The ball is initially on the lower section and the cup is on the higher section. The function $z(x, y)$ used to define this ramp is

$$z(x, y) = -0.3 \arctan(x + y).$$

Neither LOQO nor SNOPT was able to solve the model shown in Figure 7. When this happens, it is natural to suspect that the problem is infeasible. Why should the model in Figure 7 be infeasible? Alessandrini was able to solve supposedly the same model (using a different elevation function for the green). We tried several different surfaces and they all failed with all codes *except* when the surface is planar (including tilted planar surfaces). Every optimizer we tried is able to solve such planar problems easily. This proved to be a good hint that something is wrong with the model.

After much pondering, it occured to us that z is being specified in two ways—once as an explicit function of x and y and a second time as the solution to a differential equation. Since the differential equation is computed by a somewhat crude discretization, it is entirely possible that the two specifications are enough different from each other to render the model infeasible. So, we tried two things:

(1) Remove from the model the explicit statement of how `z` depends on `x` and `y`. That is, we changed

```
var z {i in 0..n} = -0.3*atan(x[i]+y[i]);
```

to just

```
var z {i in 0..n};
```

```
param g := 9.8; # acc due to gravity
param m := 0.01; # mass of a golf ball
param x0 :=  1; # coords of start pt
param y0 :=  2;
param xn :=  1; # coords of ending pt
param yn := -2;
param n := 50; # num of time points
param mu;

var T >= 0; # total time for the putt
var x{0..n};  # coords of the traj
var y{0..n};

var z   {i in 0..n}
   = -0.3*atan(x[i]+y[i]);
var dzdx{i in 0..n}
   = -0.3/(1+(x[i]+y[i])^2);
var dzdy{i in 0..n}
   = -0.3/(1+(x[i]+y[i])^2);

# v[i] denotes the deriv at midpt of
# the interval i(T/n) to (i+1)(T/n).
var vx{i in 0..n-1} = (x[i+1]-x[i])*n/T;
var vy{i in 0..n-1} = (y[i+1]-y[i])*n/T;
var vz{i in 0..n-1} = (z[i+1]-z[i])*n/T;

# a[i] denotes the accel at midpt of
# the interval (i-0.5)(T/n)
# to (i+0.5)(T/n), i.e. at i(T/n).
var ax{i in 1..n-1} = (vx[i]-vx[i-1])*n/T;
var ay{i in 1..n-1} = (vy[i]-vy[i-1])*n/T;
var az{i in 1..n-1} = (vz[i]-vz[i-1])*n/T;

var Nz{i in 1..n-1}
   = m*g/(dzdx[i]^2 + dzdy[i]^2 + 1);
var Nx{i in 1..n-1} = -dzdx[i]*Nz[i];
var Ny{i in 1..n-1} = -dzdy[i]*Nz[i];
var Nmag{i in 1..n-1}
   = m*g/sqrt(dzdx[i]^2 + dzdy[i]^2 + 1);

var vx_avg{i in 1..n-1}
   = (vx[i]+vx[i-1])/2;
var vy_avg{i in 1..n-1}
   = (vy[i]+vy[i-1])/2;
var vz_avg{i in 1..n-1}
   = (vz[i]+vz[i-1])/2;
```

```
var speed{i in 1..n-1}
   = sqrt(vx_avg[i]^2 + vy_avg[i]^2
                      + vz_avg[i]^2);

var Frx{i in 1..n-1}
   = -mu*Nmag[i]*vx_avg[i]/speed[i];
var Fry{i in 1..n-1}
   = -mu*Nmag[i]*vy_avg[i]/speed[i];
var Frz{i in 1..n-1}
   = -mu*Nmag[i]*vz_avg[i]/speed[i];

minimize finalspeed:
   vx[n-1]^2 + vy[n-1]^2;

s.t. newt_x {i in 1..n-1}:
   ax[i] = (Nx[i] + Frx[i])/m;
s.t. newt_y {i in 1..n-1}:
   ay[i] = (Ny[i] + Fry[i])/m;
s.t. newt_z {i in 1..n-1}:
   az[i] = (Nz[i] + Frz[i] - m*g)/m;

s.t. xinit: x[0] = x0;
s.t. yinit: y[0] = y0;
s.t. zinit: z[0]
   = -0.3*atan(x[0]+y[0]);

s.t. xfinal: x[n] = xn;
s.t. yfinal: y[n] = yn;
s.t. zfinal: z[n]
   = -0.3*atan(x[n]+y[n]);

s.t. onthegreen {i in 0..n}:
   x[i]^2 + y[i]^2 <= 16;

let T := 1.5;

let mu := 0.2;
let {i in 0..n}
   y[i] := (i/n)*yn + (1-i/n)*y0;
let {i in 0..n}
   x[i] := y[i]^2/2;

let mu := 0.25;
solve;
```

FIGURE 7. A first AMPL model for the putting problem.

(2) Remove from the model the part of the differential equation that relates to the `z` component of the trajectory. That is, we removed the constraints `newt_z`, `zinit`, and `zfinal`.

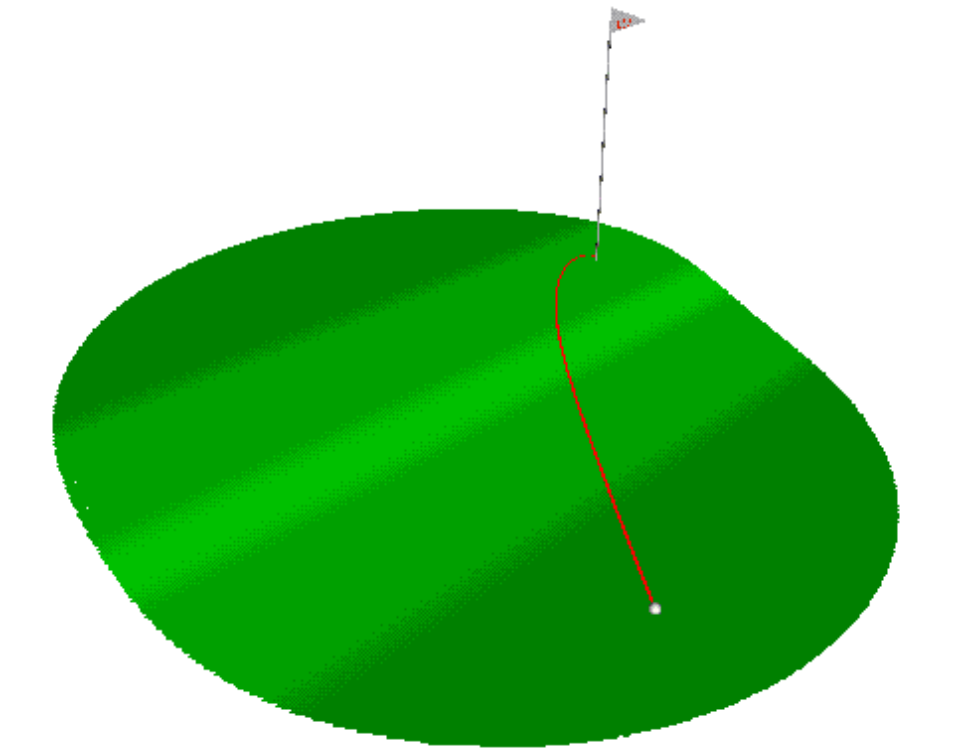

FIGURE 8. The trajectory obtained from the model in Figure 7 with the elevation constraint removed. Click on the figure to start a 3-D animation. In the animation, click on the flag to start the ball rolling.

The first of these changes produces a model that solves easily while the second one appears still to be infeasible. Hence, we seem to be on to something but more errors may be lurking. The trajectory found with the elevation constraint removed is shown in Figure 8. This trajectory looks rather reasonable. But take a look at Figure 9 which shows the same trajectory from a different angle. From this angle, we clearly see that the ball is not staying on the green but instead is flying through air to the cup. This indicates that our differential equation for z is wrong. And, if it is wrong, then the equations for x and y ought to be wrong as well.

But what is wrong? The derivation was straightforward—how could it possibly be wrong?

3.2. **The Correct Putting Model.** The key to understanding what is wrong with our implementation of the Alessandrini model is contained in the observation that the model in Figure 7 is solvable when and only when the surface of the green is planar. This suggests that the derivation is only valid for that case. What is different when the surface is not planar? Well, if you drive a car over the crest of a hill you feel lighter than normal (pun

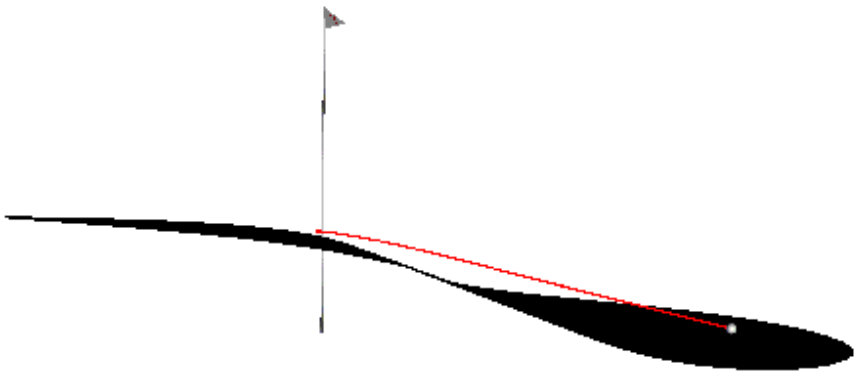

FIGURE 9. Again, the trajectory obtained from the model in Figure 7 with the elevation constraint removed. This time, however, viewed from a different angle.

intended), whereas if you speed through a valley you feel heavier. The weight that one feels is the magnitude of the normal force. Hence, this normal force is not constant when the surface has hills and valleys. As you go through a valley, the normal force must be greater than nominal in order to accelerate you along the arc defining the upward bending curve.

From this discussion, it is easy now to see that the magnitude of the normal force must be such that it compensates both for the pull of gravity and for the out-of-tangent-plane acceleration along the path:

$$\|N\| = mg\frac{e_z \cdot N}{\|N\|} + m\frac{a(t) \cdot N}{\|N\|}.$$

From this relation we can deduce that

$$N_z = m\frac{g - a_x(t)\frac{\partial z}{\partial x} - a_y(t)\frac{\partial z}{\partial y} + a_z(t)}{(\partial z/\partial x)^2 + (\partial z/\partial y)^2 + 1}.$$

Everything else in the previous derivation remains the same.

The complete correct model is shown in Figure 10. Looking down from above, the trajectory looks essentially the same as the one in Figure 8. However, from a perspective close to the green we see in Figure 11 that this trajectory does follow the surface correctly (as it must given the model).

```
param g := 9.8; # acc due to gravity
param m := 0.01; # mass of a golf ball
param x0 :=  1; # coords of start pt
param y0 :=  2;
param xn :=  1; # coords of ending pt
param yn := -2;
param n := 50; # num of time points
param mu;

var T >= 0; # total time for the putt
var x{0..n};  # coords of the traj
var y{0..n};

var z   {i in 0..n}
   = -0.3*atan(x[i]+y[i]);
var dzdx{i in 0..n} =
   -0.3/(1+(x[i]+y[i])^2);
var dzdy{i in 0..n} =
   -0.3/(1+(x[i]+y[i])^2);

# v[i] denotes the deriv at midpt of
# the interval i(T/n) to (i+1)(T/n).
var vx{i in 0..n-1} = (x[i+1]-x[i])*n/T;
var vy{i in 0..n-1} = (y[i+1]-y[i])*n/T;
var vz{i in 0..n-1} = (z[i+1]-z[i])*n/T;

# a[i] denotes the accel at the midpt of
# the interval (i-0.5)(T/n)
# to (i+0.5)(T/n), i.e. at i(T/n).
var ax{i in 1..n-1} = (vx[i]-vx[i-1])*n/T;
var ay{i in 1..n-1} = (vy[i]-vy[i-1])*n/T;
var az{i in 1..n-1} = (vz[i]-vz[i-1])*n/T;

var Nz{i in 1..n-1}
   = m*
     (g-ax[i]*dzdx[i]-ay[i]*dzdy[i]+az[i])
     /(dzdx[i]^2 + dzdy[i]^2 + 1);
var Nx{i in 1..n-1} = -dzdx[i]*Nz[i];
var Ny{i in 1..n-1} = -dzdy[i]*Nz[i];
var Nmag{i in 1..n-1}
   = m*
     (g-ax[i]*dzdx[i]-ay[i]*dzdy[i]+az[i])
     /sqrt(dzdx[i]^2 + dzdy[i]^2 + 1);

var vx_avg{i in 1..n-1}
   = (vx[i]+vx[i-1])/2;
var vy_avg{i in 1..n-1}
   = (vy[i]+vy[i-1])/2;
var vz_avg{i in 1..n-1}
   = (vz[i]+vz[i-1])/2;
```

```
var speed{i in 1..n-1}
   = sqrt(vx_avg[i]^2 + vy_avg[i]^2
                      + vz_avg[i]^2);

var Frx{i in 1..n-1}
   = -mu*Nmag[i]*vx_avg[i]/speed[i];
var Fry{i in 1..n-1}
   = -mu*Nmag[i]*vy_avg[i]/speed[i];
var Frz{i in 1..n-1}
   = -mu*Nmag[i]*vz_avg[i]/speed[i];

minimize finalspeed:
   vx[n-1]^2 + vy[n-1]^2;

s.t. newt_x {i in 1..n-1}:
   ax[i] = (Nx[i] + Frx[i])/m;
s.t. newt_y {i in 1..n-1}:
   ay[i] = (Ny[i] + Fry[i])/m;

s.t. xinit: x[0] = x0;
s.t. yinit: y[0] = y0;

s.t. xfinal: x[n] = xn;
s.t. yfinal: y[n] = yn;

s.t. onthegreen {i in 0..n}:
   x[i]^2 + y[i]^2 <= 16;

let T := 1.5;

let mu := 0.25;
let {i in 0..n} y[i] := (i/n)*yn + (1-
i/n)*y0;
let {i in 0..n} x[i] := y[i]^2/2;

solve;
```

FIGURE 10. A second, and this time correct, AMPL model for the putting problem.

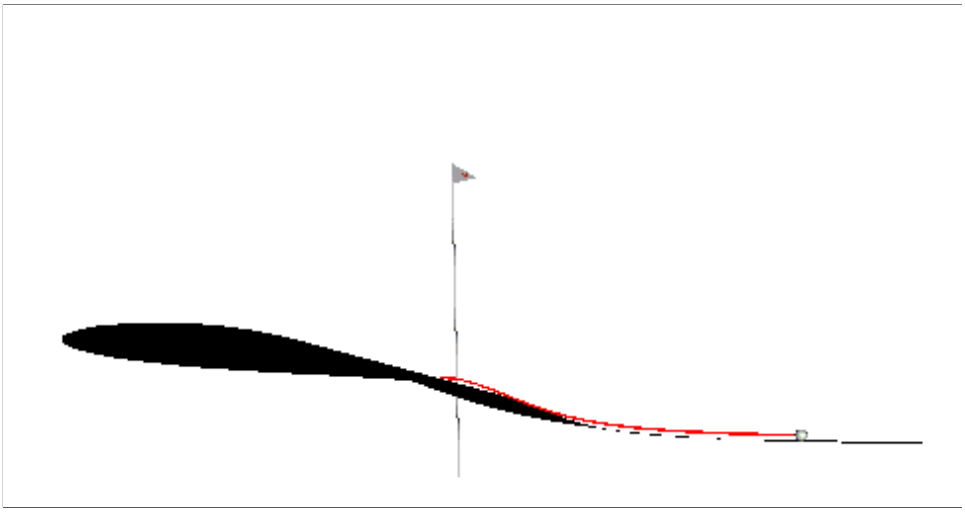

FIGURE 11. A low down view of the trajectory from the correct model shown in Figure 10.

3.3. **Trapezoidal Discretization.** The trapezoidal discretization for the correct formulation of the putting problem is shown in Figure 12. Both SNOPT and LOQO solve this formulation of the problem but each takes about twice as long as when solving the corresponding midpoint discretization formulation. Furthermore, LOQO requires a slight relaxation in the stopping criteria (the infeasibility tolerance needs to be increased from its default of 10^{-6} to 10^{-4}).

The fact that LOQO requires a relaxation in the stopping rule suggests that something might be wrong with the model. John Betts [3] seems to have identified the issue. He points out that the speed of the ball as it arrives at the cup is zero and hence there is a singularity in the differential equation at the final time. Of course, a numerical approximation might never experience the singularity exactly but it still can feel the effect. For the problem at hand, at the optimal solution LOQO has `speed[n] = 1.8e-5` and SNOPT has `speed[n] = 7e-3`. These values are not zero but they are getting close and one could imagine that numerical issues related to the sinugularity of the differential equation are beginning to enter in here. To test this, we changed the optimization objective from minimizing the final speed to minimizing the deviation of the final speed from some small prescribed value. In particular, we tried `(vx[n]^ 2 + vy[n]^ 2 - 0.25)^ 2`. With this objective function, both solvers are able to find a solution in a much more robust fashion (i.e., using fewer iterations and being successful over a wider range of choice of some of the other parameters in the problem). Our local golf expert (aka John Mulvey) indicates that this is the objective function used by real golfers anyway. He says that a real golfer does not want the ball to arrive at the cup with too little speed because then small imperfections in the green can have rather large unpredictable effects in those last few inches near the cup.

```
param g := 9.8; # acc due to gravity
param m := 0.01; # mass of a golf ball
param x0 :=  1; # coords of start pt
param y0 :=  2;
param xn :=  1; # coords of ending pt
param yn := -2;
param n := 50; # num of time points
param mu;

var T >= 0; # total time for the putt
var x{0..n};  # coords of the traj
var y{0..n};

var z   {i in 0..n}
   = -0.3*atan(x[i]+y[i]);
var dzdx{i in 0..n}
   = -0.3/(1+(x[i]+y[i])^2);
var dzdy{i in 0..n}
   = -0.3/(1+(x[i]+y[i])^2);

var vx{i in 0..n};
var vy{i in 0..n};
var vz{i in 0..n};

var ax{i in 0..n};
var ay{i in 0..n};
var az{i in 0..n};

var Nz{i in 0..n}
   = m*
     (g-ax[i]*dzdx[i]-ay[i]*dzdy[i]+az[i])
     /(dzdx[i]^2 + dzdy[i]^2 + 1);
var Nx{i in 0..n} = -dzdx[i]*Nz[i];
var Ny{i in 0..n} = -dzdy[i]*Nz[i];
var Nmag{i in 0..n}
   = m*
     (g-ax[i]*dzdx[i]-ay[i]*dzdy[i]+az[i])
     /sqrt(dzdx[i]^2 + dzdy[i]^2 + 1);

var speed{i in 0..n}
   = sqrt(vx[i]^2 + vy[i]^2 + vz[i]^2);
```

```
var Frx{i in 0..n}
   = -mu*Nmag[i]*vx[i]/speed[i];
var Fry{i in 0..n}
   = -mu*Nmag[i]*vy[i]/speed[i];
var Frz{i in 0..n}
   = -mu*Nmag[i]*vz[i]/speed[i];

minimize finalspeed: vx[n]^2 + vy[n]^2;

s.t. vx_def {i in 1..n}:
   (vx[i]+vx[i-1])/2=(x[i]-x[i-1])/(T/n);
s.t. vy_def {i in 1..n}:
   (vy[i]+vy[i-1])/2=(y[i]-y[i-1])/(T/n);
s.t. vz_def {i in 1..n}:
   (vz[i]+vz[i-1])/2=(z[i]-z[i-1])/(T/n);

s.t. ax_def {i in 1..n}:
   (ax[i]+ax[i-1])/2=(vx[i]-vx[i-1])/(T/n);
s.t. ay_def {i in 1..n}:
   (ay[i]+ay[i-1])/2=(vy[i]-vy[i-1])/(T/n);
s.t. az_def {i in 1..n}:
   (az[i]+az[i-1])/2=(vz[i]-vz[i-1])/(T/n);

s.t. newt_x {i in 0..n}:
   ax[i] = (Nx[i] + Frx[i])/m;
s.t. newt_y {i in 0..n}:
   ay[i] = (Ny[i] + Fry[i])/m;

s.t. xinit: x[0] = x0;
s.t. yinit: y[0] = y0;
s.t. xfinal: x[n] = xn;
s.t. yfinal: y[n] = yn;

s.t. onthegreen {i in 0..n}:
   x[i]^2 + y[i]^2 <= 16;

let T := 1.5;

let {i in 0..n}
   x[i] := (i/n)*xn + (1-i/n)*x0;
let {i in 0..n}
   y[i] := (i/n)*yn + (1-i/n)*y0;
let {i in 0..n} vx[i] := (xn-x0)/T;
let {i in 0..n} vy[i] := (yn-y0)/T;

let mu := 0.25;

solve;
```

FIGURE 12. The correct putting model with a trapezoidal discretization.

It is interesting to note that the midpoint rule is “less” bothered by the singularity issue. The reason is that the final speed in that model is the average final speed over the last time interval. This number is small but not as small as the final speed in the trapezoidal rule. For

example, LOQO gets a final speed of `8e-3` with this discretization, which is a few orders of magnitude larger than it got with the trapezoidal rule.

3.4. **Lessons.**

(1) It is deceptively easy to formulate a problem incorrectly.
(2) Incorrect formulations are surprisingly likely to be infeasible.
(3) Infeasibility is especially hard for nonlinear solvers to detect reliably.
(4) In the early days of optimization, a nonconvex problem with 10 or more variables was considered exceedingly hard to solve. In its most compact form, the problem here only really has 2 decision variables: the x and y components of the initial velocity vector that the putter imparts to the golf ball. After giving the ball its initial kick, the rest is determined by physics. One could formulate the problem this way. There would be just two decision variables and there would be a fairly complicated integrator function that would determine if the trajectory actually arrives at the hole and, if it does, the speed at which it arrives there. Using this integrator function as a "black box", one could make an optimization problem with just two variables. However, with modern optimization technology it is easy to incorporate the physics into the optimization model as we have done here and get a much larger model but one that is not any more difficult to solve. In fact, by expressing both the optimization part of the model and the physics in the same place and using the same "language" provides a level of model control that was totally lacking before. For example, if the physics is wrong, as it was in our first attempt, then the optimization problem is likely to be infeasible. If the physics and the optimization are separated from each other it is especially hard to identify who/what is at fault. By having them together, it is easy to print out variables, trajectories, dual variables, etc. and all of this information can be useful in figuring out what is wrong with a model.
(5) It wasn't mentioned in the discussion above, but one of the lessons in this example is how important it is to give an initial solution that is close to the optimal solution. For example, the optimal value of T is close to 2 in the examples above. We initialized T to be 1.5. Both LOQO and SNOPT find the right solution for any value of T between 1 and 3 but outside this range the solvers start to get into trouble. For example, neither of the solvers was able to solve the problem when initialized with $T = 5$.

4. HANG GLIDING

The problem we now consider is to compute the flight inputs to a hang glider so as to provide a maximum range flight. The specific problem we shall analyze is taken from [7].

One should note that the model presented here also applies to the flight of an airplane with its engines off (only the data are different). In this case, maximizing the range could be a life-saving endeavor.

The hang glider (with pilot) is pulled down by the force of gravity associated with its mass m, has a lifting force L acting perpendicular to its velocity relative to the air, and

a drag force D acting in a direction opposite to the relative velocity. Denote by x the horizontal position of the glider, by v_x the horizontal component of the absolute velocity, by y the vertical position, and by v_y the vertical component of absolute velocity.

Recall that one of the lessons of the previous section is the importance of scrutinizing every model carefully looking for errors. With that in mind and with our apologies for not practicing what we preach, we ask the reader to trust us as we assert that the following description of the equations of motion for a hang glider is correct.

4.1. **Stable Airmass.** The equations of motion for a glider in a stable airmass are as follows:

$$\begin{aligned} v_x &= \dot{x}, & a_x &= \dot{v}_x, & a_x &= \tfrac{1}{m}(-L\tfrac{v_y}{v_r} - D\tfrac{v_x}{v_r}), \\ v_y &= \dot{y}, & a_y &= \dot{v}_y, & a_y &= \tfrac{1}{m}(L\tfrac{v_x}{v_r} - D\tfrac{v_y}{v_r}) - g \end{aligned}$$

with

$$v_r = \sqrt{v_x^2 + v_y^2}, \quad L = \frac{1}{2}c_L\rho S v_r^2, \quad \text{and} \quad D = \frac{1}{2}c_D(c_L)\rho S v_r^2.$$

In [7], it is assumed that there was an updraft 250 meters into the flight. To keep the situation simple, we start by assuming that the air is still. In the next subsection, we shall consider updrafts and more complicated situations.

The glider is controlled by the lift coefficient c_L (the pilot pushes or pulls on the control bar to change c_L). The drag coefficient c_D is assumed to depend on the lift coefficient as

$$c_D(c_L) = c_0 + kc_L^2$$

where c_0 and k are fixed parameters, $c_0 = 0.034$ and $k = 0.069662$ being realistic values (and the ones used in [7]). In addition, there are limits on the lift coefficient:

$$0 \le c_L \le c_{L\max} := 1.4$$

(corresponding to the control bar being pulled in all the way and pushed out all the way, respectively). The other constants in the problem have the following specific values:

m	$= 100$	mass of glider and pilot
S	$= 14$	wing area
ρ	$= 1.13$	air density
g	$= 9.81$	acc due to gravity.

The boundary conditions are:

$$\begin{aligned} x(0) &= 0, & & \\ y(0) &= 1000, & y(T) &= 900, \\ v_x(0) &= 13.23, & v_x(T) &= 13.23, \\ v_y(0) &= -1.288, & v_y(T) &= -1.288. \end{aligned}$$

The total time T for the flight is, of course, a variable. The objective is to maximize $x(T)$.

With a stable airmass, one expects that, for appropriate choice of boundary conditions, the optimal control will be static. The optimal static control is found by minimizing the ratio of drag to lift (or, equivalently, maximizing L/D):

$$D/L = \frac{c_0}{c_L} + kc_L.$$

The minimum occurs at $c_L = \sqrt{c_0/k} = 0.69862$. Then using the fact that accelerations in a static solution vanish, we deduce that

$$\begin{aligned} -\frac{v_x}{v_y} &= \frac{L}{D} = 10.274 \\ v_r &= \sqrt{\frac{2mg}{\rho S\sqrt{c_D^2 + c_L^2}}} = 13.2901. \end{aligned}$$

From this we quickly compute that

$$v_x = 13.23, \qquad v_y = -1.288.$$

That is, the initial and final velocities given above in the boundary conditions for the dynamic version of the problem match the optimal values for the static version of the problem. Hence, we expect the optimal solution of the dynamic problem to be in fact static. Let's see if this is what we get.

The AMPL model for the midpoint discretization is shown in Figure 13. With this discretization, we have the following variables

```
T, x{0,..,N}, y{0,..,N}, cL{1,..,N-1}
```

and the following equality constraints

```
newt_x{i in 1..N-1}, newt_y{i in 1..N-1},
x_ic,  y_ic, vx_ic, vy_ic,
y_fc, vx_fc, vy_fc.
```

Hence, with this formulation the problem involves `3N+2` variables and `2N+5` equality constraints leaving `N-3` degrees of freedom over which we optimize. Using `N=150`, LOQO solves this problem in 45 interior-point iterations (4.57 seconds on a 366 MHz PC). At optimality we have `x[N]=1027.383` and `T=77.6699`. The control input as a function of time turns out to be constant as we hoped.

Now, let's consider a trapezoidal discretization. The AMPL model is shown in Figure 14. In this case, we have the following variables:

```
T, x{0,..,N}, y{0,..,N}, vx{0,..,N}, vy{0,..,N}, cL{0,..,N}
```

and the following equality constraints for the discretized problem:

```
param N;

param  x_0; param  y_0;
param vx_0; param vy_0;
param  x_n; param  y_n;
param vx_n; param vy_n;
param cL_0;
param cL_n;

param cL_min; param cL_max;
param c0; param k;
param S;  param rho;

param m; param g; param W := m*g;

var T >= 0;

# State Variables
var x {i in 0..N};
var y {i in 0..N};

# Control variables
var cL {i in 1..N-1} >= cL_min, <= cL_max;

# Abbreviations
var vx {i in 0..N-1} = N*(x[i+1]-x[i])/T;
var vy {i in 0..N-1} = N*(y[i+1]-y[i])/T;
var ax {i in 1..N-1} = N*(vx[i]-vx[i-1])/T;
var ay {i in 1..N-1} = N*(vy[i]-vy[i-1])/T;
var cD {i in 1..N-1} = c0+k*cL[i]^2;
var Vx {i in 1..N-1} = (vx[i]+vx[i-1])/2;
var Vy {i in 1..N-1} = (vy[i]+vy[i-1])/2;
var vr {i in 1..N-1}
      = sqrt(((vx[i]+vx[i-1])/2)^2
             + Vy[i]^2);
var D {i in 1..N-1}
      = .5*cD[i]*rho*S*vr[i]^2;
var L {i in 1..N-1}
      = .5*cL[i]*rho*S*vr[i]^2;
var sin_eta {i in 1..N-1} = Vy[i]/vr[i];
var cos_eta {i in 1..N-1} = Vx[i]/vr[i];
```

```
maximize final_x: x[N];

s.t. newt_x{i in 1..N-1}:
    ax[i] = (-L[i]*sin_eta[i]
            - D[i]*cos_eta[i])/m;
s.t. newt_y{i in 1..N-1}:
    ay[i] = (L[i]*cos_eta[i]
            - D[i]*sin_eta[i] - W)/m;

s.t. novomit_x {i in 1..N-1}:
    -3 <= ax[i] <= 3;
s.t. novomit_y {i in 1..N-1}:
    -3 <= ay[i] <= 3;

# Boundary Conditions
s.t. x_ic : x[0] = x_0;
s.t. y_ic : y[0] = y_0;
s.t. vx_ic: vx[0] = vx_0;
s.t. vy_ic: vy[0] = vy_0;

s.t. y_fc : y[N] = y_n;
s.t. vx_fc: vx[N-1] = vx_n;
s.t. vy_fc: vy[N-1] = vy_n;

# Data which needs to be reinitialized

data;

param N := 150;
param  x_0 :=  0;    param  y_0 := 1000;
param vx_0 := 13.23; param vy_0 :=   -1.29;

param y_n := 900;
param vx_n := 13.23; param vy_n :=   -1.29;

param cL_min := 0; param cL_max := 1.4;
param c0 := 0.034; param k := 0.069662;
param S := 14;  param rho := 1.13;

param m := 100; param g := 9.80665;

# initial guess
let {j in 0..N} x[j] := 5000*j/N;
let {j in 0..N} y[j] := -100*j/N+1000;
let {j in 1..N-1} cL[j] := .7;
let T := 30;

solve;
```

FIGURE 13. The hang-glider range-maximization model using a midpoint discretization.

```
x_eqn {1..N}, y_eqn {1..N}, vx_eqn {1..N}, vy_eqn {1..N},
x_ic, y_ic, vx_ic, vy_ic, y_fc, vx_fc1, vy_fc1.
```

```
param N;

param  x_0; param  y_0;
param vx_0; param vy_0;
param  x_n; param  y_n;
param vx_n; param vy_n;
param cL_0; param cL_n;

param cL_min; param cL_max;
param c0; param k;
param S;  param rho;

param m; param g; param W := m*g;

var T >= 10, <= 200;

# State Variables
var x {i in 0..N};
var y {i in 0..N};
var vx {i in 0..N} <= 30, >= -30;
var vy {i in 0..N} <= 30, >= -30;

# Control variables
var cL {i in 0..N} >= cL_min, <= cL_max;

# Abbreviations
var cD {i in 0..N} = c0+k*cL[i]^2;
var vr {i in 0..N} = sqrt(vx[i]^2 + vy[i]^2);
var D {i in 0..N} = .5*cD[i]*rho*S*vr[i]^2;
var L {i in 0..N} = .5*cL[i]*rho*S*vr[i]^2;
var sin_eta {i in 0..N} = vy[i]/vr[i];
var cos_eta {i in 0..N} = vx[i]/vr[i];
var ax {i in 0..N} = (-L[i]*sin_eta[i]
                      - D[i]*cos_eta[i])/m;
var ay {i in 0..N} = (L[i]*cos_eta[i]
                    - D[i]*sin_eta[i]
                    - W)/m;

maximize final_x: x[N];

# Trapezoidal Discretization
s.t. x_eqn {j in 1..N}:
   (x[j]-x[j-1])/(T/N) = (vx[j]+vx[j-1])/2;

s.t. y_eqn {j in 1..N}:
   (y[j]-y[j-1])/(T/N) = (vy[j]+vy[j-1])/2;

s.t. vx_eqn {j in 1..N}:
   (vx[j]-vx[j-1])/(T/N)
   = (ax[j]+ax[j-1])/2;

s.t. vy_eqn {j in 1..N}:
   (vy[j]-vy[j-1])/(T/N)
   = (ay[j]+ay[j-1])/2;

# Boundary Conditions
s.t. x_ic : x[0] = x_0;
s.t. y_ic : y[0] = y_0;
s.t. vx_ic : vx[0] = vx_0;
s.t. vy_ic : vy[0] = vy_0;

s.t. y_fc : y[N] = y_n;
s.t. vx_fc1: vx[N] = vx_n;
s.t. vy_fc1: vy[N] = vy_n;

s.t. monotone_x {i in 1..N}:
      x[i] >= x[i-1];
s.t. novomit_x {i in 0..N}:
      -3 <= ax[i] <= 3;
s.t. novomit_y {i in 0..N}:
      -3 <= ay[i] <= 3;

# Data which needs to be reinitialized

data;

param N := 150;
param  x_0 :=  0;    param  y_0 := 1000;
param vx_0 := 13.23; param vy_0 :=   -1.29;

param y_n := 900;
param vx_n := 13.23; param vy_n :=   -1.29;

param cL_min := 0; param cL_max := 1.4;
param c0 := 0.034; param k := 0.069662;
param S := 14; param rho := 1.13;

param m := 100; param g := 9.80665;

# initial guess
let {j in 0..N} x[j] := 5000*j/N;
let {j in 0..N} y[j] := -100*j/N+1000;
let {j in 0..N} vx[j] := 13.23;
let {j in 0..N} vy[j] := -1.29;
let {j in 0..N} cL[j] := 0.7;
let T := 30;

solve;
```

FIGURE 14. The hang-glider range-maximization model using a trapezoidal discretization.

With this formulation, the problem involves `5N+6` variables and `4N+7` equality constraints leaving `N-1` degrees of freedom over which we optimize. This is 2 more than with the previous model. Using `N=150`, LOQO solves this problem in 141 interior-point iterations (24.2 seconds on a 366 MHz PC). At optimality we have `x[N] = 1027.488` and `T = 77.7241`. After flying more than a kilometer, this optimal solution is better than the previous one by 10 centimeters. Perhaps this just reflects a difference in the discretization or maybe it is really a different answer. To see which it is, let's look at the control input—Figure 16. From the control input we see that this solution is definitely not static. But it is also not implementable as it is discontinuous at $t = 0$ (followed by nontrivial control inputs for approximately the first 9 seconds of the flight).

The discontinuity of the control input suggests that the model formulation, i.e. the discretization, has too many degrees of freedom. To check this hypothesis, we tried introducing *continuity* constraints. First we added just one such constraint:

```
cL[0] = cL[1]
```

With this constraint, we got a solution that was closer to the static solution but was still not itself static. So, we added a second continuity constraint:

```
cL[1] = cL[2]
```

With these two constraints, the model solves in 222 interior-point iterations (41.1 seconds on a 366 MHz PC). At optimality we get `x[N] = 1027.383` and `T = 77.6699` in exact agreement with the static solution. Furthermore, the optimal control is again static.

It is noteworthy that the "correct" number of degrees of freedom for the adjusted trapezoidal rule matches the number of degrees of freedom from the midpoint rule. Clearly, something fundamental is going on here.

4.2. **Unstable Airmass.** The original problem studied in [7] involved an updraft 250 meters into the flight. The vertical velocity profile for this updraft is given by

$$u_a(x) = u_m e^{-\left(\frac{x}{R}-2.5\right)^2}\left(1-\left(\frac{x}{R}-2.5\right)^2\right)$$

and is shown in Figure 15. To change the model to account for this unstable airmass profile, we simply replace every occurance of v_y in the model with $v_y - u_a(x)$. Both the midpoint discretization and the trapezoidal discretization (with the two c_L continuity constraints) are easy to solve. See Figures 17–22.

The trapezoidal discretization without the two extra continuity constraints exhibits the same anomolous behaviour as was illustrated by the stable air example. This version of the model appears in the COPS suite of problems [4]. The fact that is causes trouble for some solvers is documented in [5].

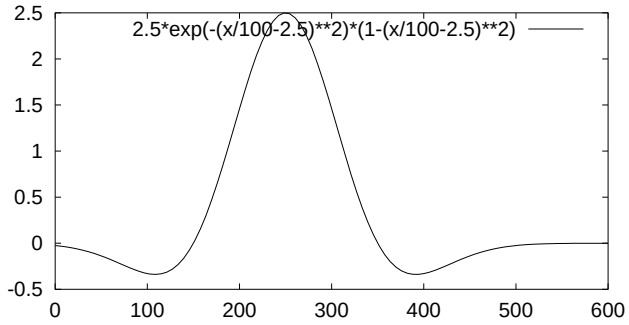

FIGURE 15. Updraft profile

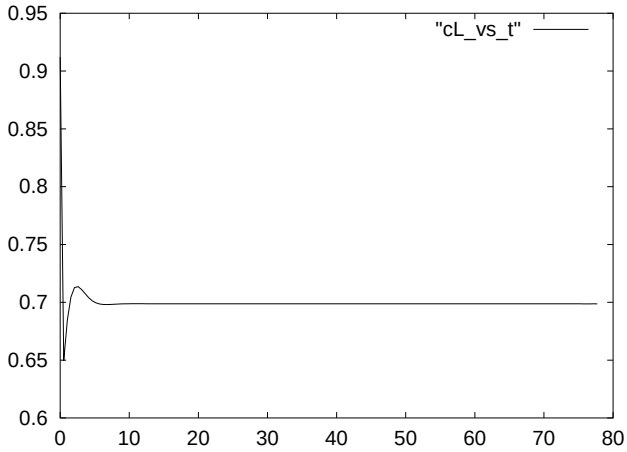

FIGURE 16. Control input as a function of time. Trapezoidal discretization method.

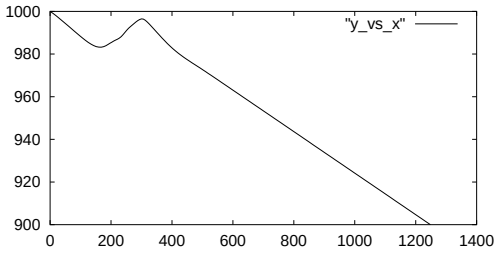

FIGURE 17. y vs x

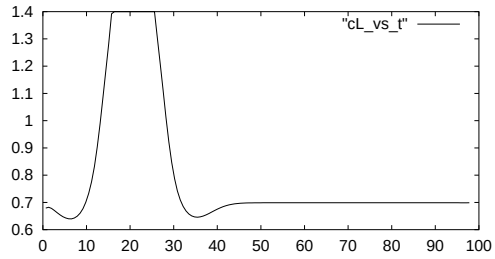

FIGURE 20. cL vs t

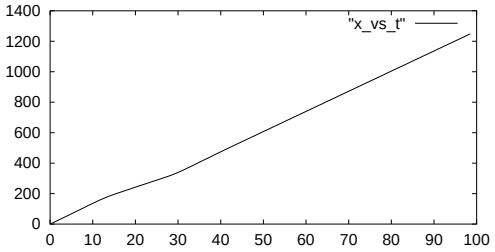

FIGURE 18. x vs t

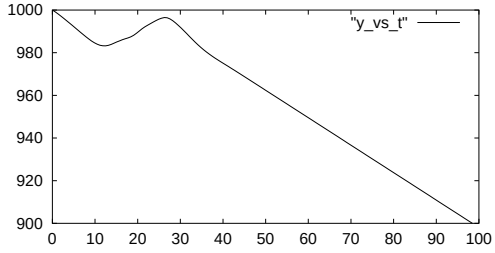

FIGURE 21. y vs t

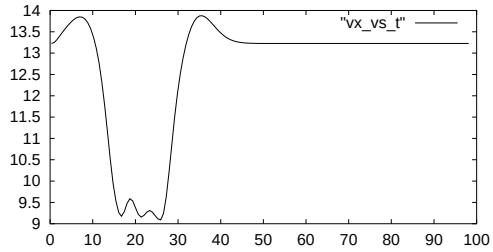

FIGURE 19. v_x vs t

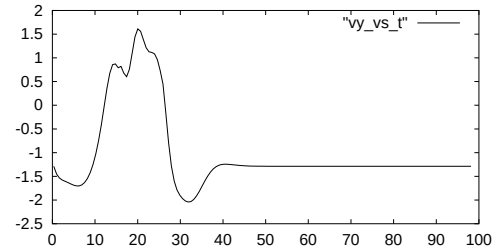

FIGURE 22. v_y vs t

For more on optimal control of flight paths, we refer the reader to Stengel's classic text [12] and to the recent book by Bryson [6].

4.3. **Lessons.** In this section there are two lessons:

(1) Before solving a problem of interest, always test a model by first solving a problem whose solution is mathematically tractible.
(2) The two discretization methods that we've discussed throughout this paper are commonly used for simple numerical integration of ODEs. In this case, the problem is well-posed if the number of equations matches the number of variables; i.e., there are no degrees of freedom. One can show that a problem is well-posed with respect to midpoint discretization if and only if it is well-posed with respect to trapezoidal discretization. However, as we saw in this example, for control problems, i.e. problems where there are degrees of freedom, the midpoint discretization will sometimes have fewer degrees of freedom than the trapezoidal discretization. It seems that the midpoint discretization has the "correct" number and that the trapezoidal discretization has too many.

5. TOBOGGANING

It wouldn't be right to end our discussion of trajectory optimization without discussing the famous Brachistochrone problem. We do it here in this last section in the context of designing a slide which will get the rider from the beginning to the end in the shortest amount of time.

We assume in this study that the slide will be constructed out of frictionless material. We let x denote, as usual, horizontal displacement and we let y denote vertical displacement. In contrast with earlier conventions, we assume that y increases as one moves downward. The slide will connect two points having coordinates $(x_0, y_0) = (0, 0)$ and (x_f, y_f). The toboggan starts from rest at the top of the slide. Hence, initially the toboggan has zero kinetic energy and zero potential energy (due to gravity). Since the slide is frictionless, there is no loss of energy as heat. By conservation of energy, the total energy must always remain zero. So, when the toboggan has dropped to level y, we have

$$\frac{1}{2}mv^2 - mgy = 0.$$

Here, v denotes the speed of the toboggan. From this relation we see that

$$v = \sqrt{2gy}.$$

Now, the time to do the run can be computed as an integral of differential chunks of time

$$T = \int_0^T dt = \int_0^T \frac{ds(t)}{v(t)}.$$

```
param n := 512;

param x {j in 0..n} := j/n;

# Variables
var y {j in 0..n} >= 0;

# Abbreviations
var dydx {j in 1..n}
   = (y[j]-y[j-1])/(x[j]-x[j-1]);
var f {j in 1..n}
   = sqrt( (1+dydx[j]^2)/y[j-1] );
```

```
minimize time:
   sum {j in 1..n}
      f[j]*(x[j]-x[j-1]) ;

subject to y0: y[0] = 1.0e-12;
subject to yn: y[n] = 1;

subject to monotone {j in 1..n}:
   dydx[j] >= 0;

let {j in 0..n} y[j] := x[j];

solve;
```

FIGURE 23. A working AMPL model for the Brachistochrone problem.

This last integral can be reparametrized using any monotone function of time. The natural candidate is horizontal displacement x. With this choice, we get

$$T = \int_0^{x_f} \frac{\sqrt{1 + y'(x)^2}}{\sqrt{2gy(x)}} dx. \tag{3}$$

The problem then is to find a function $y(x)$ that minimizes this integral and satisfies the constraints $y(0) = 0$ and $y(x_f) = y_f$. Using calculus of variations, one can show that the general form of the solution to (3) is a cycloid:

$$\begin{aligned} x &= k^2(\theta - \sin\theta) \\ y &= k^2(1 - \cos\theta). \end{aligned}$$

However, to find the values of k and θ_f to satisfy the original terminal conditions involves solving a transcendental equation—not such an easy task.

The AMPL model expressing the minimization of T as given in (3) is shown in Figure 23. It took some tinkering before we were able to get to the working model shown in the Figure. The main issue is that $y(0) = 0$ appears in the denominator of the integrand when $x = 0$. Hence, the integral is a singular integral. To address this, we first changed the boundary condition to $y(0) = 10^{-12}$. Also, since `dydx[j]` appearing in the definition of `f[j]` represents the value of the derivative at the midpoint of the interval $[j-1, j]$, one would expect to use the best estimate for y at this same place, i.e., `(y[j]+y[j-1])/2`. However, with this choice for denominator in the integrand, the optimal solution exhibits a very large jump discontinuity at $x = 0$. Presumably this is caused by the singularity but the details elude us. Using `y[j-1]`, as shown, works but changing it to `y[j]` renders the problem unsolvable to both LOQO and SNOPT. Again, the reason remains a mystery. It is easy to think of lots of other things to try (and we did) but we stop here in favor of a different line of attack. The model shown in the Figure is solved by LOQO in 26 iterations. It takes 0.85 seconds on a 366 MHz PC.

```
param n := 512;

param y {j in 0..n} := (j/n);

# Variables
var x {j in 0..n};

# Abbreviations
var dxdy {j in 1..n}
   = (x[j] - x[j-1])/(y[j] - y[j-1]);
var f {j in 1..n}
   = sqrt( (dxdy[j]^2+1)/((y[j]+y[j-
1])/2) );
```

```
minimize time:
   sum {j in 1..n} f[j]*(y[j]-y[j-1]) ;

subject to x0: x[0] = 0;
subject to xn: x[n] = 1;

let {j in 0..n} x[j] := y[j];

solve;
```

FIGURE 24. A second working AMPL model for the Brachistochrone problem.

Instead of using horizontal displacement as the parameterization variable in (3), we could equally well have chosen to use the vertical displacement variable. With this choice, we get

$$T = \int_0^{y_f} \frac{\sqrt{1 + x'(y)^2}}{\sqrt{2gy}} dy. \tag{4}$$

The AMPL model for this formulation of the problem is shown in Figure 24. LOQO solves this model in just 10 iterations. It takes only 0.26 seconds on a 366 MHz PC. Clearly, from a numerical perspective, this formulation is much better than the previous one.

5.1. **Lesson.** The lesson to take away from this case study is that one should consider a variety of ways to formulate a given problem. Some might be much easier to solve than others.

REFERENCES

[1] S.M. Alessandrini. A motivational example for the numerical solution of two-point boundary-value problems. *SIAM Review*, 37(3):423–427, 1995. 9

[2] 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. 2

[3] J.T. Betts. *Practical Methods for Optimal Control using Nonlinear Programming*. SIAM, Philadelphia, PA, 2000. 16

[4] A.S. Bondarenko, D.M. Bortz, and J.J. Moré. COPS: Constrained optimization problems. http://www-unix.mcs.anl.gov/ more/cops/. 23

[5] A.S. Bondarenko, D.M. Bortz, and J.J Moré. COPS: Large-scale nonlinearly constrained optimization problems. Technical report, Technical Report ANL/MCS-TM-237, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL, 1998. Revised Oct 1999. 23

[6] A.E. Bryson. *Dynamic Optimization*. Addison Wesley Longman, Inc., Menlo Park, CA, 1999. 26

[7] R. Bulirsch, E. Nerz, H.J. Pesch, and O. von Stryk. Combining direct and indirect methods in optimal control: Range maximization of a hang glider. In R. Bulirsch, A. Miele, J. Stoer, and K.H. Well, editors, *"Optimal Control: Calculus of Variations, Optimal Control Theory and Numerical Methods*, pages 273–288. Birkhauser Verlag, Basel, Boston, Berlin, 1993. 18, 19, 23

[8] A.R. Conn, N. Gould, and Ph.L. Toint. Constrained and unconstrained testing environment. http://www.dci.clrc.ac.uk/Activity.asp?CUTE. 9

[9] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press, 1993. 2

[10] 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. 2

[11] 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. 9

[12] R.F. Stengel. *Optimal Control and Estimation*. Dover, Mineola, NY, 1994. 26

[13] R.J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 12:451–484, 1999. 2

[14] R.J. Vanderbei. LOQO user's manual—version 3.10. *Optimization Methods and Software*, 12:485–514, 1999. 2

[15] R.J. Vanderbei and D.F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999. 2

ROBERT J. VANDERBEI, PRINCETON UNIVERSITY, PRINCETON, NJ