Interior Point Methods
in Engineering Applications

Dieter Kraft*

1 Introduction

The pathbreaking contribution of KARMARKAR[24] to the solution of linear programming problems has initiated the so-called interior-point revolution, which in turn also had and has a great influence on methods for solving nonlinear optimization problems of the form

$$\text{(1.1)} \qquad \begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } c_i(x) \geq 0, \quad i = 1, \ldots, m, \end{aligned}$$

where x is a vector of dimension n and $f(x)$ and $c_i(x)$ are assumed to be twice continuously differentiable.

The purpose of this paper is to review some interior-point algorithms and their implementations into software packages. It is well known that interior-point methods are especially efficient for large (and sparse) problems. For better understanding of the interior-point paradigm we start with the linear programming problem, extend this to quadratic programming problems, and finish with nonlinear nonconvex problems of type (1.1). We end with some examples from engineering applications and numerical comparisons of several state-of-the-art software codes. Our derivation follows the lines given by VANDERBEI[19], and repeats some results from FORSGREN et al.[5].

2 Motivation

In engineering applications there often exists the need to solve the following optimal control problem (OCP):

Find control functions $\mathbf{u}(t), \quad \mathbf{u} \in \mathbb{R}^m, \quad t \in [t_0, t_f]$,
to minimize the cost function

$$\text{(2.1)} \qquad J = \phi(\mathbf{y}(t_f), t_f), \quad \mathbf{y} \in \mathbb{R}^n,$$

subject to differential constraints

$$\text{(2.2)} \qquad \dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), \mathbf{u}(t), t),$$

and algebraic (often called state- and/or control-) constraints

$$\text{(2.3)} \qquad \mathbf{g}(\mathbf{y}(t), \mathbf{u}(t), t) \geq \mathbf{0},$$

*Munich University of Applied Sciences, Department of Mechanical Engineering, dieter.kraft@fhm.edu.
This research was supported by the Bavarian Ministry of Science and Technology within the BaCaTec project http://www.bacatec.de.

and (for the sake of simplicity separated) boundary conditions[1]

$$\begin{aligned} \mathbf{h_1}(\mathbf{y}(t_0), \mathbf{u}(t_0), t_0) &= \mathbf{0}, \\ \mathbf{h_2}(\mathbf{y}(t_f), \mathbf{u}(t_f), t_f) &= \mathbf{0}. \end{aligned} \tag{2.4}$$

The infinite-dimensional problem (OCP) can generally not be solved exactly on a computer [26]. Therefore (OCP) is approximated by a discrete version (DOCP) on a mesh $0, \ldots, N$. This can be done in a number of different ways, e.g. finite difference methods, collocation methods, and finite element methods [20].

Define, for $k = 0, \ldots, N$, a mesh

$$\Delta := t_0 < t_1 < \cdots < t_{N-1} < t_f, \tag{2.5}$$

with uniform mesh size

$$h := (t_f - t_0)/N, \tag{2.6}$$

and grid points

$$t_{k+1} = t_k + h, \quad k = 0, \ldots, N-1, \tag{2.7}$$

and let

$$\mathbf{y}(t_k) =: \mathbf{y}_k, \quad \mathbf{u}(t_k) =: \mathbf{u}_k, \quad \mathbf{f}(\mathbf{y}(t_k), \mathbf{u}(t_k), t_k) =: \mathbf{f}_k.$$

We consider two methods to transform (OCP) to (DOCP):

$$\min_{(\mathbf{y}, \mathbf{u}, t_f)} \phi(\mathbf{y}_f, \mathbf{u}_f, t_f), \tag{2.8}$$

namely finite differences and collocation.

2.1 Finite Difference Method

One of the simplest and most easy to understand finite difference method is the trapezoidal rule, where in addition to (2.8) the difference equations

$$\mathbf{h_0} := \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h}{2}(\mathbf{f}_k + \mathbf{f}_{k+1}) = \mathbf{0}, \quad k = 0, \ldots, N-1, \tag{2.9}$$

and state/control constraints

$$\mathbf{g}(\mathbf{y}_k, \mathbf{u}_k, t_k) \geq \mathbf{0}, \quad k = 0, \ldots, N-1, \tag{2.10}$$

and boundary conditions

$$\begin{aligned} \mathbf{h_1}(\mathbf{x}_0, \mathbf{u}_0, t_0) &= \mathbf{0}, \\ \mathbf{h_2}(\mathbf{y}_f, \mathbf{u}_f, t_f) &= \mathbf{0}, \end{aligned} \tag{2.11}$$

have to be satisfied.
Remarks:
1) Eq. (2.9) can also be interpreted as collocation at Lobatto points.
2) The Jacobian of (2.9) is a sparse matrix.
3) Eqs. (2.8–10) form a nonlinear program, like (1.1).

[1] An example with multipoint boundary conditions is given in Section 6.2.4.

2.2 Collocation Method

In addition to mesh (2.5), for a collocation method of order k, collocation points

(2.12) $0 \le \rho_1 < \cdots < \rho_k \le 1$

are defined for each subinterval $[t_i, t_{i+1}]$ together with subgrid points

(2.13) $t_{ij} = t_i + \rho_j h, \quad i = 0, \ldots, N, \quad j = 1, \ldots, k.$

Now, let $\mathbf{y}_\Delta(t)$ be an interpolating polynomial of order $k+1$ defined on $[t_i, t_{i+1}]$ by the interpolation conditions

(2.14) $\mathbf{y}_\Delta(t_i) = \mathbf{y}_i =: \mathbf{y}(t_i),$

and the collocation conditions

(2.15) $\dot{\mathbf{y}}_\Delta(t_{ij}) = \mathbf{f}(\mathbf{y}_\Delta(t_{ij}), \mathbf{u}_\Delta(t_{ij}), t_{ij}), \quad i = 0, \ldots, N, \quad j = 1, \ldots, k.$

We express $\mathbf{y}_\Delta(t)$ on $[t_i, t_{i+1}]$ as its Taylor expansion about t_i

(2.16) $\mathbf{y}_\Delta(t) = \mathbf{y}_i + h \sum_{j=1}^{k} \frac{(t - t_i)^2}{j! h^j} w_{ij},$

where w_{ij} are free parameters to be adjusted. Eq. (2.16) has to be evaluated at points t_{ij} in the collocation conditions (2.15), therefore we have

$$\mathbf{y}_\Delta(t_{ij}) = \mathbf{y}_i + h \sum_{j=1}^{k} \frac{\rho^j}{j!} w_{ij},$$

and

$$\dot{\mathbf{y}}_\Delta(t_{ij}) = \sum_{j=1}^{k} \frac{\rho^{j-1}}{(j-1)!} w_{ij}.$$

We construct from the polynomial pieces a piecewise polynomial **pp** which is continuous on $[t_0, t_f]$, that means we require the following continuity conditions

(2.17) $\mathbf{y}_\Delta(t_i) = \mathbf{y}_\Delta(t_{i+1}), \quad i = 1, \ldots, N-1.$

The free variables in the collocation method are now

(2.18) $\mathbf{v} = (\mathbf{u}_\Delta(t_{ij}), \mathbf{y}_i, w_{ij})',$

together with (in free end time problems) t_f and with $\dim(\mathbf{v}) = nN + kmN + knN + (1)$. We summarize the collocation method:

> Minimize wrt. $\mathbf{v}$ the cost function (2.8)
> subject to
> the collocation conditions (2.15),
> the continuity conditions (2.17),
> the state constraints (2.10),
> and the boundary conditions (2.11).

Remarks:
1) Often, the collocation points ρ_i are chosen as the zeros of the Legendre polynomials of different order. Sometimes, this is called collocation at Gaussian points [8].
2) If $k = 1$ then $\rho_1 = \frac{1}{2}$, and the method is equivalent to the mid-point rule.

It will turn out, see e.g. Section 6, that interior-point optimization algorithms are particularly efficient for solving problems of type (DOCP). We therefore describe these methods and especially their primal-dual aspects in detail starting from their roots in linear programming.

3 Linear Programming

Associated to a linear program in standard form

$$
\begin{aligned}
&\text{maximize } c'x \\
(3.1)\qquad &\text{subject to } Ax \le b, \\
&\qquad\qquad\quad x \ge 0,
\end{aligned}
$$

with x and c vectors of dimension n, b a vector of dimension m, and A a matrix of dimension $m \times n$, and where $'$ denotes the transpose of a vector or matrix, is a dual problem

$$
\begin{aligned}
&\text{minimize } b'y \\
(3.2)\qquad &\text{subject to } A'y \ge c, \\
&\qquad\qquad\quad y \ge 0,
\end{aligned}
$$

with y a vector of dual variables of dimension n.

3.1 The Model Equations

First, both problem formulations (3.1) and (3.2) are converted to equality-constrained problems by adding slack variables w and z, respectively:

$$
\begin{aligned}
&\text{maximize } c'x \\
(3.3)\qquad &\text{subject to } Ax + w = b, \\
&\qquad\qquad\quad x, w \ge 0,
\end{aligned}
$$

and

$$
\begin{aligned}
&\text{minimize } b'y \\
(3.4)\qquad &\text{subject to } A'y - z = c, \\
&\qquad\qquad\quad y, z \ge 0.
\end{aligned}
$$

Now, in the primal problem (3.3) the sign constraints on the variables are incorparated into the cost function via logarithmic barrier functions with barrier parameter μ

$$
\begin{aligned}
(3.5)\qquad &\text{maximize } c'x + \mu \sum_{j=1}^{n} \ln x_j + \mu \sum_{i=1}^{m} \ln w_i \\
&\text{subject to } Ax + w = b.
\end{aligned}
$$

This equality-constrained problem is solved by the method of Lagrange[17]. First, the Lagrange function is defined with an m-vector of Lagrange multipliers y

$$
(3.6)\qquad L(x, w, y) = c'x + \mu \sum_{j=1}^{n} \ln x_j + \mu \sum_{i=1}^{m} \ln w_i + y'(b - Ax - w).
$$

Then, the first-oder necessary conditions for a critical point of problem (3.5) are evaluated

$$
\begin{aligned}
&\frac{\partial L}{\partial x_j} = c_j + \mu \frac{1}{x_j} - \sum_{i=1}^{m} y_i a_{ij} &= 0, \quad j = 1, \ldots, n, \\
(3.7)\qquad &\frac{\partial L}{\partial w_i} = \mu \frac{1}{w_i} - y_i &= 0, \quad i = 1, \ldots, m, \\
&\frac{\partial L}{\partial y_i} = b_i - \sum_{j=1}^{n} a_{ij} x_j - w_i &= 0, \quad i = 1, \ldots, m.
\end{aligned}
$$

The set of equations (3.7) is written in matrix form

$$
\begin{aligned}
A'y - \mu X^{-1}e &= c,\\
y &= \mu W^{-1}e,\\
Ax + w &= b,
\end{aligned}
$$

where $X = \operatorname{diag}(x_j)$, $W = \operatorname{diag}(w_i)$, and e is the vector with all entries equal to one of respective dimension. Defining $z := \mu X^{-1}e$ and re-arranging leads to the following system

$$
\begin{aligned}
&(3.9a) \quad Ax + w = b,\\
&(3.9b) \quad A'y - z = c,\\
&(3.9c) \quad XZe = \mu e,\\
&(3.9d) \quad YWe = \mu e.
\end{aligned}
$$

System (3.9) is a set of $2(m+n)$ equations in $2(m+n)$ unknowns $(x_\mu, y_\mu, w_\mu, z_\mu)$. The set

$$(3.10) \quad \mathcal{S} = \{(x_\mu, y_\mu, w_\mu, z_\mu),\ \mu > 0\}$$

of solutions of (3.9) is called the central path. The first two of these equations are the equality constraints in the primal and dual problem, respectively, while the last two are the complementarity conditions in the duality theory for $\mu = 0$, see e.g. [15]. They are sometimes called μ-complementarity conditions. Also, for $\mu = 0$ conditions (3.9) are the Karush-Kuhn-Tucker(KKT) necessary conditions. Note that these equations are nonlinear, and it is this fact that makes interior-point methods difficult to tackle. Two results on the existence and uniqueness of a solution of system (3.9) are given. For a proof see e.g. [15] or [19].

Theorem 3.1. *A critical point of the set of equations* (3.9) *exists if and only if both the primal feasible set* $\mathcal{P} = \{Ax \le b,\ x \ge 0\}$ *and the dual feasible set* $\mathcal{D} = \{A'y \ge c,\ y \ge 0\}$ *have a non-empty interior.*

Theorem 3.2. *If a critical point exists it is unique, and there the linear program attains its maximum.*

3.2 The Solution Procedure

The Search Direction. The system (3.9) is solved iteratively for a fixed value of μ by a damped Newton method to find the solution u^* of a function $F_\mu(u) = 0$

$$(3.11) \quad J_\mu(u_k)\Delta u_k = -F_\mu(u_k),$$

where $u = (x'_\mu, y'_\mu, w'_\mu, z'_\mu)'$ is the vector of primal and dual variables and primal and dual slack variables, respectively, and the next step is generated by

$$(3.12) \quad u_{k+1} = u_k + \alpha_k \Delta u_k,$$

with an initial starting estimate $u_0 > 0$, Δu_k as the solution of Eq. (3.11), and $\alpha_k \in (0,1]$ a damping factor. In (3.11) $F_\mu : \mathbb{R}^{2(m+n)} \mapsto \mathbb{R}^{2(m+n)}$ is the vector

$$
F_\mu = \begin{pmatrix} Ax_k + w_k - b \\ A'y_k - z_k - c \\ X_k Z_k e - \mu e \\ Y_k W_k e - \mu e \end{pmatrix},
$$

and $J_\mu : \mathbb{R}^{2(m+n)\times 2(m+n)} \mapsto \mathbb{R}^{2(m+n)}$ the JACOBIan matrix

$$J_\mu = \begin{pmatrix} A & & I & \\ & A' & & -I \\ Z_k & & & X_k \\ & W_k & Y_k & \end{pmatrix}.$$

Summarized, we have to solve in each iteration an (in practice large but sparse) linear system

$$\begin{pmatrix} A & & I & \\ & A' & & -I \\ Z_k & & & X_k \\ & W_k & Y_k & \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \\ \Delta w_k \\ \Delta z_k \end{pmatrix} = - \begin{pmatrix} Ax_k + w_k - b \\ A'y_k - z_k - c \\ X_k Z_k e - \mu e \\ Y_k W_k e - \mu e \end{pmatrix}. \tag{3.13}$$

In system (3.13) the last two equations can be used to eliminate

$$\Delta w_k = Y_k^{-1}(\mu e - Y_k W_k e - W_k \Delta y_k) \tag{3.14}$$

and

$$\Delta z_k = X_k^{-1}(\mu e - X_k Z_k e - Z_k \Delta x_k), \tag{3.15}$$

thus reducing (3.13) to

$$\begin{pmatrix} X_k^{-1} Z_k & A' \\ A & -Y_k^{-1} W_k \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} c - A'y_k + \mu X_k^{-1} e \\ b - Ax_k - \mu Y_k^{-1} e \end{pmatrix}. \tag{3.16}$$

This is a symmetric system and the matrix is called the reduced KKT matrix. A numerical solution can be achieved e.g. by the HARWELL-Library subroutine `MA27` or `MA27` [11].

The Step Length. It is required, see Eqs. (3.3) and (3.4), that all variables stay positive. Therefore, possibly the full stepsize from Eq. (3.13) cannot be applied but must be restricted to $u_k + \alpha_k \Delta u_k > 0$, or

$$\frac{1}{\alpha_k} > -\frac{\Delta u_k}{u_k}.$$

Also, we must not overstep the optimal NEWTON step $\alpha_k = 1$. Therefore the actual steplength implementation is

$$\alpha_k = \min\Big(1, \theta\Big(\max_i -\frac{\Delta u_{k_i}}{u_{k_i}}\Big)^{-1}\Big), \tag{3.17}$$

where $0.9 < \theta < 1$ is a safeguard to obtain strict positive variables.

Update of the Barrier Parameter. There are several ways to treat the update of the barrier parameter μ. The so-called short-step method, see e.g. [15], starts with a large value of $\mu_0 \gg 1$ and reduce it by the following relationship

$$\mu_{l+1} = \mu_l \Big(1 - \frac{1}{6\sqrt{n}}\Big),$$

which for large n is near one.

We prefer VANDERBEIS [19] procedure

$$\mu_{l+1} = \delta \frac{x'_{\mu_l} z_{\mu_l} + y'_{\mu_l} w_{\mu_l}}{n+m}, \tag{3.18}$$

with δ a heuristic parameter to guarantee sufficient reduction of μ. In practice $\delta = 0.1$ has turned out to be a reliable value.

Convergence Criteria. The following measures are defined: primal feasibility

$$\rho = b - Ax - w, \tag{3.19}$$

dual feasibility

$$\sigma = c - A'y + z, \tag{3.20}$$

and complementarity

$$\gamma = x'z + y'w. \tag{3.21}$$

With these measures, the sequence (3.12) is stopped at a solution if the following convergence criteria are satisfied:

$$\|\rho\|_1 < \varepsilon \quad \wedge \quad \|\sigma\|_1 < \varepsilon \quad \wedge \quad \gamma < \varepsilon,$$

where ε is a small positive tolerance. Also, it is stopped if the following holds

$$\|x\|_\infty > M \quad \vee \quad \|y\|_\infty > M,$$

where M is a large positive tolerance. In the former case the primal problem is unbounded, in the latter case the dual problem is unbounded which means the primal is infeasible.

3.3 A Prototype Algorithm

Here we present a prototype procedure of the algorithm presented in the previous section. It is implemented in `Matlab` [9].

```
function [u,k,status] = IPLP(A,b,c,u)
%Path-Following algorithm as described in Chapter 17 and 18 of rvdb's LP book
% max c'x s.t. Ax<=b, x>=0
% [m,n]=size(A), n=length(c), m=length(b)
% u are the initial estimates of
% primal variables x, dim(x)=n,
% dual variables    y, dim(y)=m,
% primal slacks      w, dim(w)=m,
% dual slacks         z, dim(z)=n,
% in that order
% and xout are the respective output variables
% k is the number of iterations
% status=0: optimal solution found
% status=2: primal infeasibilty
% status=4: dual infeasibilty (unbounded problem)
% status=1: solution not found in 500 iterations
% version: implemented by the members of the class
% "Modeling, Simulation and Optimization"  summer 2004

% check errors
error(nargchk(3,4,nargin)); % either 3 or 4 input arguments
[m,n]=size(A);
```

```
nm=n+m;
nmm=nm+m;
nmmn=nmm+n;
p=nmmn;
if nargin == 4
   l=length(u);
   if l ~= p
      error('length(u) must be 2*(m+n)');
   end
end
if length(b) ~= m
   error('length(b) must be m');
end
if length(c) ~= n
   error('length(c) must be n');
end
if nargin == 3
   u=ones(p,1); % default initial variables
end
u=u(:); % columnize!
em=ones(m,1);
en=ones(n,1);
DELTA=1/10;
R=9/10;
EPS=1e-8;
HUGE=1e10;
ITERMAX=500;
status=1;

for k=1:ITERMAX,  % iteration loop
x=u(1:n);
y=u(1+n:nm);
w=u(1+nm:nmm);
z=u(1+nmm:nmmn);
X=diag(x);
Xi=diag(1./x);
Y=diag(y);
Yi=diag(1./y);
W=diag(w);
Z=diag(z);
gamma=z'*x+y'*w; % duality gap
mu=DELTA*gamma/nm;
rho=b-A*x-mu*Yi*em; % primal infeasibilty
sigma=c-A'*y+mu*Xi*en; % dual infeasibilty
% check convergence (Paragraph 17.5.3)
if gamma<EPS && norm(b-A*x-w,1)<EPS && norm(c-A'*y+z,1)<EPS,
   status=0;
   break; % optimal
end % if
if norm(x,inf)>HUGE,
   status=2;
```

```
      break; %primal infeasibility
   end % if
   if norm(y,inf)>HUGE,
      status=4;
      break; %dual infeasibility
   end % if
   % reduced KKT-system (18.6-18.7)
   KKT=[Xi*Z,A';A,-Yi*W]; % symmetric KKT-matrix
   du=KKT\[sigma;rho]; % solve it
   dx=du(1:n);
   dy=du(1+n:nm);
   % calculate DELTA slacks
   dz=Xi*(mu*en-X*Z*en-Z*dx);
   dw=Yi*(mu*em-Y*W*em-W*dy);
   du=[du;dw;dz];
   theta=max(-du./u); % ratio test (17.6)
   theta=min(1,R/theta);
   u=u+theta*du;
   end % for
```

4 Quadratic Programming

A quadratic program usually is formulated as follows[2]

$$
\begin{aligned}
&\text{minimize } \frac{1}{2}x'Qx + c'x \\
&\text{subject to } Ax \geq b, \\
&\qquad\qquad\quad x \geq 0,
\end{aligned}
\tag{4.1}
$$

where it can be assumed that the matrix Q is symmetric.[3]

Associated to this program is a dual formulation[4]

$$
\begin{aligned}
&\text{maximize } b'y - \frac{1}{2}x'Qx \\
&\text{subject to } A'y + z - Qx = c, \\
&\qquad\qquad\quad y, z \geq 0.
\end{aligned}
\tag{4.2}
$$

4.1 The Model Equations

As in linear programming, the general inequality constraints are changed to equality constraints by introducing slack variables w

$$
\begin{aligned}
&\text{minimize } \frac{1}{2}x'Qx + c'x \\
&\text{subject to } Ax - w = b, \\
&\qquad\qquad\quad x, w \geq 0.
\end{aligned}
$$

[2] Attend that the cost function is quadratic in the variables and the constraints are linear. Also, the cost function is minimized and the constraints are non-negative.

[3] A non-symmetric $\bar{Q}$ can be made symmetric through $Q = \bar{Q}'\bar{Q}/2$

[4] For a thorough discussion of duality in quadratic programming see [19, pp. 399–402]

Again, the bounds on the variables (x, w) are tied to the cost function via barrier functions[5]

$$\begin{aligned}&\text{maximize } \frac{1}{2}x'Qx + c'x - \mu\sum_{j=1}^{n}\ln x_j - \mu\sum_{i=1}^{m}\ln w_i \\ &\text{subject to } Ax - w = b.\end{aligned} \tag{4.3}$$

After defining the LAGRANGE function

$$L(x, w, y) = \frac{1}{2}x'Qx + c'x - \mu\sum_{j=1}^{n}\ln x_j - \mu\sum_{i=1}^{m}\ln w_i + y'(b - Ax + w), \tag{4.4}$$

and setting its partial derivatives to zero we get a set of first-order necessary conditions for a critical point of problem (4.1)

$$\begin{aligned} Ax - w &= b, \\ A'y + z - Qx &= c, \\ XZe &= \mu e, \\ YWe &= \mu e, \end{aligned} \tag{4.5}$$

where as in Eq. (3.9) we defined $z = \mu X^{-1}e$.

To guarantee a solution of problem (4.1) it is necessary that the matrix Q is positive semidefinite, i.e. the eigenvalues of Q must be non-negative. This means that the quadratic cost function of the primal problem is a convex function over the convex set $\mathcal{P} = \{Ax - w = b,\ x, w \geq 0\}$. If this is valid the following can be stated:

Theorem 4.1. *If a convex quadratic program has a solution $(x^*, w^*) \in \mathcal{P}$ and its dual program has a solution $(x^*, y^*, z^*) \in \mathcal{D} = \{A'y + z - Qx = c,\ x, y, z \geq 0\}$ such that*

$$0 = {y^*}'w^* + {z^*}'x^* \tag{4.6}$$

then the primal solution is optimal for the primal problem and the dual solution is optimal for the dual problem.

A proof of this theorem for which convexity is essential can be found in [19].

4.2 The solution Procedure

The procedure to solve the necessary conditions (4.5) is analog to that in section 3.2 with the appropriate modifications.

The Search Direction. Again, NEWTONs method is used to find the zero of (4.5)

$$J_\mu(u_k)\Delta u_k = -F_\mu(u_k),$$

in connection with the update of the variables

$$u_{k+1} = u_k + \alpha_k \Delta u_k.$$

This time F_μ and J_μ write as

$$F_\mu = \begin{pmatrix} Ax_k - w_k - b \\ A'y_k + z_k - c - Qx_k \\ X_k Z_k e - \mu e \\ Y_k W_k e - \mu e \end{pmatrix},$$

[5] Note the minus signs because we are minimizing!

and

$$J_\mu = \begin{pmatrix} A & & -I & \\ -Q & A' & & I \\ Z_k & & & X_k \\ & W_k & Y_k & \end{pmatrix}.$$

After elimination of Δw_k and Δz_k from the last two equations

$$\Delta w_k = Y_k^{-1}(\mu e - Y_k W_k e - W_k \Delta y_k), \tag{4.7}$$

and

$$\Delta z_k = X_k^{-1}(\mu e - X_k Z_k e - Z_k \Delta x_k), \tag{4.8}$$

we arrive at the following symmetric KKT system

$$\begin{pmatrix} -X_k^{-1} Z_k - Q & A' \\ A & Y_k^{-1} W_k \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} c + Q x_k - A' y_k - \mu X_k^{-1} e \\ b - A x_k + \mu Y_k^{-1} e \end{pmatrix}. \tag{4.9}$$

The technical remainder of this section, namely steplength, barrier-update, and convergence criteria is the same as in section 3.2. Therefore it is omitted here, and we present, again implemented in `Matlab`

4.3 A Prototype Algorithm

```
function [u,k,status] = IPQP(A,b,c,Q,u)
%Path-Following QP-algorithm as described in Chapter 23 of rvdb's LP book
% min c'x+x'Qx/2 s.t. Ax>=b, x>=0
% [m,n]=size(A), n=length(c), m=length(b), [n,n]=size(Q)
% u are the initial estimates of
% primal variables x, dim(x)=n,
% dual variables    y, dim(y)=m,
% primal slacks      w, dim(w)=m,
% dual slacks         z, dim(z)=n,
% in that order
% and xout are the respective output variables
% k is the number of iterations
% status=0: optimal solution found
% status=2: primal infeasibilty
% status=4: dual infeasibilty (unbounded problem)
% status=1: solution not found in 500 iterations
% version: implemented by the members of the class
% "Modeling, Simulation and Optimization"  summer 2004

% check errors
error(nargchk(4,5,nargin)); % either 3 or 4 input arguments
[m,n]=size(A);
[q,q]=size(Q);
Q=(Q+Q')/2;
nm=n+m;
nmm=nm+m;
nmmn=nmm+n;
```

```
if nargin == 5
   l=length(u);
   if l ~= nmmn
      error('length(u) must be 2*(m+n)');
   end
end
if length(b) ~= m
   error('length(b) must be m');
end
if length(c) ~= n
   error('length(c) must be n');
end
if q ~=n
   error('size(Q) must be nxn');
end % if
if nargin == 4
   u=ones(nmmn,1); % default initial variables
end
u=u(:); % columnize!
em=ones(m,1);
en=ones(n,1);
DELTA=1/10;
R=9/10;
EPS=1e-8;
HUGE=1e10;
ITERMAX=500;
status=1;
for k=1:ITERMAX,  % iteration loop
x=u(1:n);
y=u(1+n:nm);
w=u(1+nm:nmm);
z=u(1+nmm:nmmn);
X=diag(x);
XI=diag(1./x);
Y=diag(y);
YI=diag(1./y);
W=diag(w);
Z=diag(z);
gamma=z'*x+y'*w; % duality gap
mu=DELTA*gamma/nm;
rho=b-A*x+w;
rho1=b-A*x+mu*YI*em; % primal infeasibilty
sigma=c-A'*y-z+Q*x;
sigma1=c-A'*y-mu*XI*en+Q*x; % dual infeasibilty
% check convergence (Paragraph 17.5.3)
if gamma<EPS && norm(rho,1)<EPS && norm(sigma,1)<EPS,
   status=0;
   break; % optimal
end % if
if norm(x,inf)>HUGE,
```

```
      status=2;
      break; %primal infeasibility
   end % if
   if norm(y,inf)>HUGE,
      status=4;
      break; %dual infeasibility
   end % if
   % reduced KKT-system (18.6-18.7)
   KKT=[-(XI*Z+Q),A';A,YI*W]; % symmetric KKT-matrix
   du=KKT\[sigma1;rho1]; % solve it
   dx=du(1:n);
   dy=du(1+n:nm);
   % calculate DELTA slacks
   dz=XI*(mu*en-Z*(X*en+dx));
   dw=YI*(mu*em-W*(Y*em+dy));
   du=[dx;dy;dw;dz];
   theta=max(-du./u); % ratio test (17.6)
   theta=min(1,R/theta);
   u=u+theta*du;
   end % for
```

5 Nonlinear Programming

Now, we are in the position to develop interior point methods for the general problem (1.1), which we repeat here for convenience

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } c_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

As above, the inequality constraints are transformed to equality constraints via slack variables w_i

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } c_i(x) - w_i = 0, \quad i = 1, \ldots, m, \\ &\qquad w_i \geq 0, \end{aligned}$$

and then, the bounds on the slacks are tied to the cost function via a logarithmic barrier function, resulting in the equality constrained problem

$$\begin{aligned} &\text{minimize } B_\mu(x, w) = f(x) - \mu \sum_{i=1}^{m} \ln w_i \\ &\text{subject to } c_i(x) - w_i = 0, \quad i = 1, \ldots, m. \end{aligned} \tag{5.1}$$

5.1 The Model Equations

The Lagrange function associated to this problem with parameters y_i is

$$L_\mu(x, w, y) = f(x) - \mu \sum_{i=1}^{m} \ln w_i - \sum_{i=1}^{m} y_i(c_i(x) - w_i).$$

The first-order necessary conditions for a critical point of (5.1) are

$$
\begin{aligned}
&\frac{\partial L}{\partial x_j} = \frac{\partial f}{\partial x_j}(x) - \sum_{i=1}^{m} y_i \frac{\partial c_i}{\partial x_j}(x) && = 0, \quad j = 1, \dots, n, \\
(5.2)\qquad &\frac{\partial L}{\partial w_i} = -\mu \frac{1}{w_i} + y_i && = 0, \quad i = 1, \dots, m, \\
&\frac{\partial L}{\partial y_i} = c_i - w_i && = 0, \quad i = 1, \dots, m,
\end{aligned}
$$

which can be written in compact form as follows

$$
(5.3)\qquad
\begin{aligned}
\nabla f(x) - \nabla c(x)' y &= 0, \\
W Y e - \mu e &= 0, \\
c(x) - w &= 0.
\end{aligned}
$$

5.2 The Solution Procedure

The set of equations (5.3) is again solved by NETWONs method to find the solution u^* of $F_\mu(u) = 0$, with

$$
u = \begin{pmatrix} x \\ w \\ y \end{pmatrix}, \qquad \Delta u = \begin{pmatrix} \Delta x \\ \Delta w \\ \Delta y \end{pmatrix},
$$

and

$$
F_\mu(u) = \begin{pmatrix} \nabla f(x) - \nabla c(x)' y \\ W Y e - \mu e \\ c(x) - w \end{pmatrix} = \begin{pmatrix} \sigma \\ -W\gamma \\ -\rho \end{pmatrix},
$$

where ρ and σ are measures of primal and dual infeasibility, respectively. To point out the analogy to quadratic programming we define the following HESSE matrix

$$
H(x, y) = \nabla^2 f(x) - \sum_{i=1}^{m} y_i \nabla^2 c_i(x)
$$

and JACOBI matrix

$$
A(x) = \nabla c(x)
$$

and write NEWTONs equations $J_\mu(u_k)\Delta u_k = -F_\mu(u_k)$ as follows

$$
\begin{pmatrix} H(x_k, y_k) & 0 & -A(x_k)' \\ 0 & Y_k & W_k \\ A(x_k) & -I & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta w_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} -\sigma \\ W\gamma \\ \rho \end{pmatrix},
$$

which, after being symmetrized, reads as follows

$$
(5.4)\qquad
\begin{pmatrix} -H(x_k, y_k) & 0 & A(x_k)' \\ 0 & -Y_k W_k^{-1} & -I \\ A(x_k) & -I & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta w_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} \sigma \\ -\gamma \\ \rho \end{pmatrix}.
$$

After multiplying the first row by minus one and eliminating Δw_k from the second row

$$
\Delta w_k = W Y^{-1} (\gamma - \Delta y_k)
$$

the following symmetric reduced KKT system is obtained

$$\tag{5.5} \begin{pmatrix} -H(x_k, y_k) & A'(x_k) \\ A(x_k) & W_k Y_k^{-1} \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} \sigma \\ \rho + W_k Y_k^{-1} \gamma \end{pmatrix}.$$

Having solved this, the vector of variables is updated with a damped NEWTON step

$$u_{k+1} = u_k + \alpha_k \Delta u_k.$$

In system (5.4) $H(x_k, y_k)$ is positive semidefinite if $f(x_k)$ is convex, each $c_i(x_k)$ is concave and each $y_{k_i} \geq 0$. Therefore, The JACOBIan is positive semidefinite and the system can be solved by off-the-shelf solvers, e.g. `MA27` of the HARWELL library.

5.3 Modifications for Nonlinearity and Nonconvexity

In contrast to convex quadratic programming, it does not suffice in general convex nonlinear optimization to choose the steplength a_k by the simple ratio test (3.17). VANDERBEI and SHANNO [31] have given a simple example in which the NEWTON method only converges with the introduction of a merit function to find the appropriate stepsize. Other researchers, e.g. GERTZ and GILL [21], have proposed a trust region approach to force convergence of the algorithm. Recently, also filter line-search algorithms have been implemented, e.g. [29], [6].

Merit Functions. There are many merit functions reported in the literature, e.g. the FORSGREN-GILL augmented penalty-barrier merit function [4], an exact penalty function [31] which augments the cost function with the weighted one-norm of the primal infeasibility vector $\rho(x, w)$, or a FIACCO-MCCORMICK-type merit function [31] which we consider in this paper

$$\tag{5.6} \Phi_{\mu,\nu}(x, w) = f(x) - \mu \sum_{i=1}^{m} \ln(w_i) + \frac{1}{2}\nu \|\rho(x, w)\|_2^2.$$

With this merit function the following result of the search directions can be proved [31]
Theorem 5.1. *If the dual normal matrix*

$$\tag{5.7} N(x, y, w) := H(x, y) + A(x)' W^{-1} Y A(x)$$

is positive definit, then, the search directions have the following properties:
(1) If $\rho(x, w) = 0$, then

$$\begin{pmatrix} \nabla_x B_\mu(x, w) \\ \nabla_w B_\mu(x, w) \end{pmatrix}' \begin{pmatrix} \Delta x \\ \Delta w \end{pmatrix} \leq 0,$$

(2) There exists $\nu_{min} \geq 0$ such that, for every $\nu > \nu_{min}$

$$\begin{pmatrix} \nabla_x \Phi_{\mu,\nu}(x, w) \\ \nabla_w \Phi_{\mu,\nu}(x, w) \end{pmatrix}' \begin{pmatrix} \Delta x \\ \Delta w \end{pmatrix} \leq 0.$$

In both cases, equality holds if and only if (x, w) satisfies (5.3) *for some y.*

Remark (1): Property (1) of the Theorem shows that the search direction $(\Delta x, \Delta w)$ is a descent direction for the barrier function $B_\mu(x, w)$ if the current solution is primal feasible.
Remark (2): Property (2) of the Theorem shows that the search direction $(\Delta x, \Delta w)$ is a descent direction for the merit function $\Phi_{\mu,\nu}(x, w)$ if ν is sufficiently large.

Filter Methods.

To be expanded ...

Trust Region Methods.

To be expanded ...

Convexification. If $H(x, y)$ fails to be positive semidefinite then the dual normal matrix $N(x, y, w)$ might fail to be positive definite, and we lose the descent properties of Theorem 5.1. To retain those properties, we perturb the HESSEan

$$\bar{H}(x, y) = H(x, y) + \lambda I, \quad \lambda \geq 0,$$

and compute the search directions using $\bar{H}$ instead of H. Now let $\bar{N}$ the dual normal matrix associated with $\bar{H}$ then the following result guarantees reduction of primal and dual infeasibility and complementarity

Theorem 5.2. *Let*

$$\begin{aligned}
\bar{x} &= x_k + \theta \Delta x_k, \\
\bar{y} &= y_k + \theta \Delta y_k, \\
\bar{w} &= w_k + \theta \Delta w_k, \\
\bar{\rho} &= \rho(\bar{x}, \bar{w}), \\
\bar{\sigma} &= \sigma(\bar{x}, \bar{y}),
\end{aligned}$$

then

$$\begin{aligned}
\bar{\rho} &= (1 - \theta)\rho(x_k, w_k) + o(\theta), \\
\bar{\sigma} &= (1 - \theta)\sigma(x_k, y_k) - \lambda\theta\Delta x_k + o(\theta), \\
\bar{w}'\bar{y} &= (1 - \theta(1 - m\frac{\mu}{w_k' y_k}))w_k' y_k + o(\theta).
\end{aligned}$$

Proof: See [31].

5.4 A Prototype Algorithm

To be tested; not yet stable enough

6 Implementations and Results

6.1 Optimization Codes

There area great many of proposed interior point algorithms (an excellent review is presented in [5]) but only a small number of state-of-the-art implementations into efficient software. The most well-known of these are `IPOPT` [6], `KNITRO` [28], and `LOQO` [31]. While the latter two programs are commercial codes, the first one is published as open source code [1] under the Common Public License (CPL) [2]. `IPOPT` solves nonlinear optimization problems of the following form

$$\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & f(x), \\
\text{s.t.} \quad & c(x) = 0, \\
& x_L \leq x \leq x_U,
\end{aligned} \tag{6.1}$$

where inequality constraints, $c_i(x) \geq 0$, can be reformulated in the above format by introducing slack variables. For comparison, we include into our tests `SNOPT` [14], an algorithm of the active-set SQP type. We perform our tests in the `AMPL` environment [30], a modeling program for mathematical programming.

Before detailed results of some engineering applications will be presented the efficiency of the codes tested in the `COPS` [3] benchmarking set [10] is evaluated in the following two tables.

No	Name	`FILTER`	`KNITRO`	`LOQO`	`MINOS`	`SNOPT`
1	polygon	5	1	4	3	2
2	elec	5	1	4	3	2
3	camshape	2	5	3	1	4
4	chain	3	1	2	3	3
5	pinene	5	1	2	3	4
6	marine	4	1	2	5	3
7	channel	5	1	2	3	4
8	robot	4	1	5	3	2
9	steering	3	1	5	4	2
10	rocket	4	3	1	5	2
11	glider	2	1	3	5	4
12	gasoil	5	2	1	3	4
13	methanol	5	2	1	3	4
14	catalyst	5	2	1	3	4
15	torsion	4	5	1	3	2
16	bearing	5	4	1	3	2
17	minsurf	2	4	1	3	5
18	triangle	2	1	4	5	3
19	tetra	3	1	2	5	4
20	lane_emden	5	1	2	3	4
21	dirichlet	3	2	1	4	5
22	henon	3	1	2	4	5
	$\sum/22$	3.82	1.91	2.27	3.50	3.36

Table 1: Code Ranking in the COPS Report

In Table 1 the average time ranking for all three parameter sets of each example as taken from the respective Table Ex.2 is listed for each solver row-wise. The column-sum of these rankings is divided by the number of examples. The overall ranking shows the advantage of the interior point algorithms.

In Table 2 the test of this report on the `COPS` set are summarized. I only could include `IPOPT` 2.2.1c, `LOQO` 6.06 and `SNOPT` 6.2-1 in this comparison as I only have full unrestricted licenses for these solvers. Therefore, also in the `analyze` phase of the calculations `MINOS` was replaced by `SNOPT`. Shown are the the values of `ampl`'s `_total_solve_time` in seconds for each example, parameter set and solver, respectively. The result indicated by **** means that the solver timed out (`_total_solve_time`>1800 seconds) or could not solve that problem. Detailled results can be obtained by the author. The ranking in the last line is performed as in Table 1. All computations were performed on an Intel Pentium 4 XEON 2GHz CPU with 1024GB RAM and a 512KB cache, running Linux Fedora Core 2. The default values of the solver parameters have been chosen. Again, the superiority of interior point solvers is obvious, as also has been pointed out by GOULD, recently [22].

	Code	IPOPT			LOQO			SNOPT		
No	Name	par1	par2	par3	par1	par2	par3	par1	par2	par3
1	polygon	1.05	6.96	25.57	2.05	5.63	8.40	3.53	30.17	586.17
2	elec	1.11	20.89	114.13	1.00	13.35	200.28	0.79	4.82	50.57
3	camshape	1.19	1.49	2.02	77.26	102.99	3.32	3.78	6.25	8.76
4	chain	0.13	0.23	12.10	0.35	0.68	1.45	4.50	20.57	65.54
5	pinene	0.90	1.91	3.59	0.90	1.99	4.45	6.68	19.87	122.52
6	marine	0.50	1.05	2.24	0.71	1.64	4.49	3.89	10.19	33.48
7	channel	0.37	0.72	1.43	1.03	2.50	6.97	3.90	13.46	44.84
8	robot	0.75	1.81	4.50	****	****	****	3.25	15.42	137.77
9	steering	1.41	17.75	1.11	****	****	****	5.71	27.99	166.38
10	rocket	1.18	2.37	5.28	0.77	1.62	3.65	11.76	21.33	51.76
11	glider	4.77	25.25	102.48	****	5.49	****	8.18	****	****
12	gasoil	0.70	1.27	2.14	0.36	0.80	2.51	1.84	6.17	22.04
13	methanol	0.66	1.64	3.70	0.70	1.65	4.68	4.06	13.02	53.14
14	catalyst	0.36	0.70	1.46	0.43	1.03	2.97	10.93	36.50	88.52
15	torsion	0.87	1.45	1.88	0.74	1.11	1.52	7.64	23.73	52.94
16	bearing	0.69	1.03	1.58	0.59	0.91	1.23	6.87	19.58	42.96
17	minsurf	2.74	6.00	8.57	2.33	3.42	****	918.21	****	****
18	triangle	0.96	0.64	2.72	****	****	52.32	494.08	111.18	****
19	tetra	****	3.07	6.01	4.63	2.50	6.46	77.72	44.42	163.15
20	lane_emden	56.37	220.79	577.41	83.98	634.70	1572.53	735.66	****	****
21	dirichlet	95.15	241.69	1232.27	266.95	715.85	1350.49	****	****	****
22	henon	104.86	118.49	****	135.37	175.19	577.19	****	****	****
	Rank		1.36			1.91			2.73	

Table 2: My Cops Tests

6.2 Examples

Most of our test examples, all of which are dynamic optimization problems, are included in the `COPS` benchmarking set.[6]

6.2.1 Sounding Rocket

The problem of maximizing the altitude of a sounding rocket was first posed by R. H. Goddard in 1919 [12]. With state variables $(v(t),\ h(t),\ m(t))$, velocity, altitude, and mass, respectively, and control variable $T(t)$, the rocket thrust, the problem can be formulated as follows

$$
\begin{aligned}
&\max_{T(t)} \quad h(t_f), && \\
&\text{s.t.} \quad \dot{v}(t) = \frac{T(t) - D(v(t), h(t))}{m(t)} - g(h(t)), && v(t_0) = 0, \\
&\qquad \dot{h}(t) = v(t), && h(t_0) = h_0, \\
&\qquad \dot{m}(t) = -\frac{T(t)}{c}, && m(t_0) = m_0, \quad m(t_f) = c_m m(t_0), \\
&\qquad 0 \leq T(t) \leq T_{max},
\end{aligned}
\tag{6.2}
$$

where h_0 is the radius of the earth, $0 < c_m < 1$ is a fraction, and c is the (constant) specific impulse of the rocket. Note, that in Eq. (6.2) the last equation represents a control constraint. The drag D depends on the velocity and altitude

$$
D = c_D v^2 \exp\Big(-c_h \frac{h(t) - h_0)}{h_0}\Big),
$$

where c_D and c_h are constants, and g the gravitational force per unit mass varies with altitude

$$
g = g_0 \Big(\frac{h_0}{h(t)}\Big)^2,
$$

with g_0 the gravitational force per unit mass on the earth's surface. We apply the following data: $T_{max} = \frac{7}{2} m_0 g_0$, $c_D = \frac{1}{2} c_v m_0 / g_0$, $c = \frac{1}{2}\sqrt{g_0 h_0}$, $c_h = 500$, $c_m = \frac{3}{5}$, $c_v = 620$, and we normalize without loss of generality $g_0 = h_0 = m_0 = 1$.

To solve the optimal control problem (6.2) we apply the $O(h^2)$ trapezoidal discretization, i.e. for $\dot{z}(t) = f(z(t), u(t))$ the state $z(t)$ is approximated by the difference equation

$$
z[i+1] = z[i] + \frac{1}{2} h(f(z[i], u[i]) + f(z[i+1], u[i+1])),
\tag{6.3}
$$

on a uniform mesh $i = 0 \ldots N$ with steplength $h = (t_f - t_0)/N$.

All three solvers solve the problem. The timing results are shown in line 10 of Table 2. Both the interior-point codes are more efficient than the active-set code by a factor of almost ten. The states and optimal control are presented in Fig. 1 and Fig. 2. The bang-singular-bang character of the thrust can be realized.

[6] The model and data files of those examples not included in the `COPS` benchmarking set can be downloaded from http://www-neos.mcs.anl.gov/neos/solvers/NCO:IPOPT/solver-sample.html

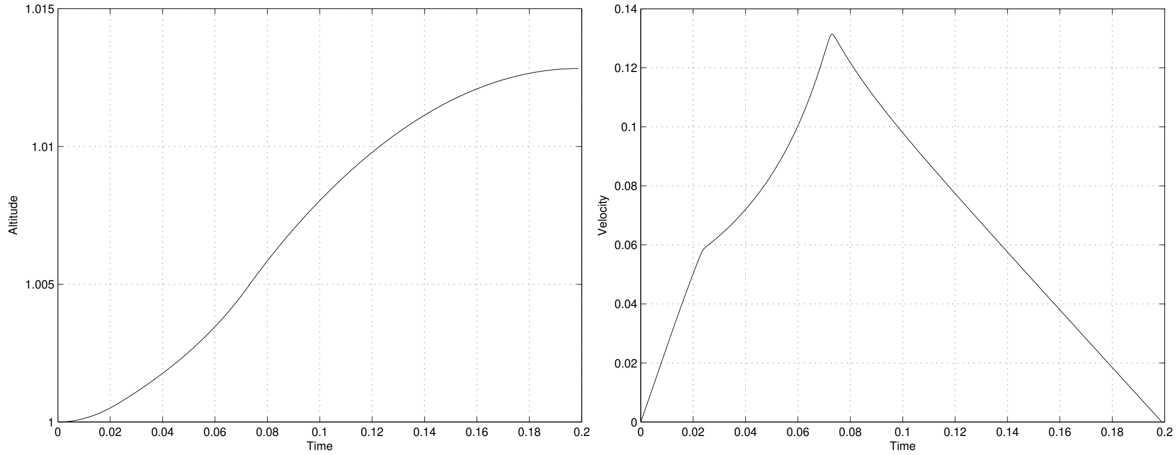

Figure 1: Altitude and Velocity of the Goddard Rocket Problem

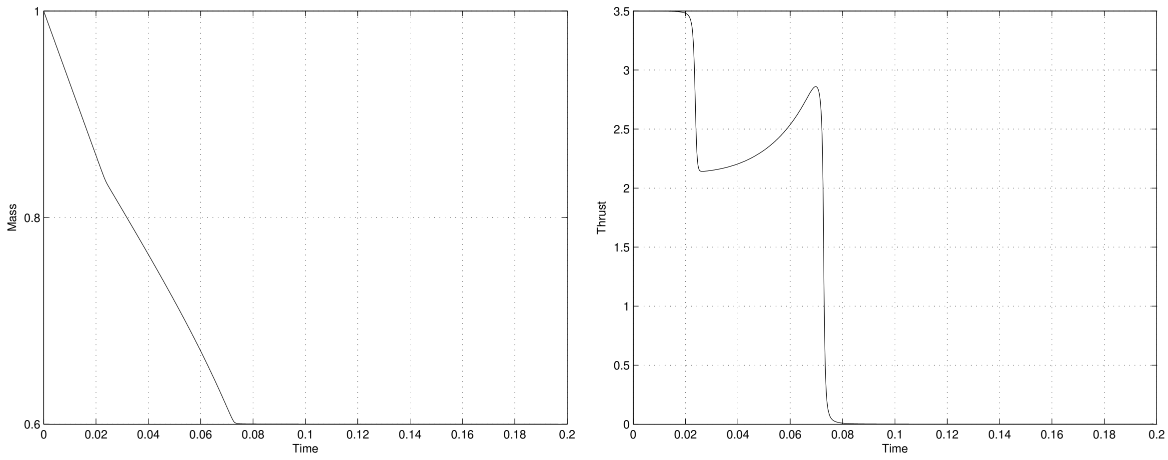

Figure 2: Mass and Thrust of the Goddard Rocket Problem

6.2.2 A Toy Car

A simplified description of the motion of a car in a horizontal plane is given by the following equations [7, pages 97–98]

$$
\begin{aligned}
&\dot{x} = v\cos\theta, && x_{min} \le x(t) \le x_{max}, && t_0 < t < t_f, && x(t_0) = x_0, && x(t_f) = x_f,\\
&\dot{y} = v\sin\theta, && y_{min} \le y(t) \le y_{max}, && t_0 < t < t_f, && y(t_0) = y_0, && y(t_f) = y_f,\\
&\dot{\theta} = \frac{v}{L}\tan\psi, && \theta_{min} \le \theta(t) \le \theta_{max}, && t_0 < t < t_f, && \theta(t_0) = \theta_0, && \theta(t_f) = \theta_f,\\
&\dot{v} = a - \mu v, && v_{min} \le v(t) \le v_{max}, && t_0 < t < t_f, && v(t_0) = v_0, && v(t_f) = v_f,
\end{aligned}
\tag{6.4}
$$

where x and y are the coordinates of the car, θ is its heading angle measured positive counter-clockwise from the x-axis, v is the velocity of the car, $L = 11.5$ the distance between the fore and rear axle. The car is controlled by the it's accelearation a, and the steering angle ψ which the front wheels make with the car's body. The friction of the car is described by the damping factor $\mu = 1/6$.

The problem is to satisfy Eqs. (6.4) and to

$$
\begin{aligned}
&\min \quad (t_f - t_0),\\
\text{s.t.}\quad & a_{min} \le a(t) \le a_{max},\\
& \psi_{min} \le \psi(t) \le \psi_{max}.
\end{aligned}
\tag{6.5}
$$

Again, we use trapezoidal discretization (6.3) to approximate the differential equations (6.6) on a grid with $n = 1000$ intervals with the following initial guess for $i = 0, \dots, n$:

$$
\begin{aligned}
t_{f,0} &= 1.0,\\
x_{i,0} &= x_f i/n,\\
y_{i,0} &= \begin{cases} 20i/n & \text{if } i < n/2 \\ 20(1 - i/n) & \text{if } i \ge n/2 \end{cases},\\
\theta_{i,0} &= (1 - 2i/n)\frac{\pi}{2},\\
v_{i,0} &= \begin{cases} 270i/n & \text{if } i < n/2 \\ 270(1 - i/n) & \text{if } i \ge n/2 \end{cases},\\
a_{i,0} &= \begin{cases} a_{max} & \text{if } i < n/2 \\ a_{min} & \text{if } i \ge n/2 \end{cases},\\
\psi_{i,0} &= \frac{1}{2}\psi_{min},
\end{aligned}
$$

and the data summarized in Table 3.

	min	max	0	f
x	0	55	0	50
y	0	7	0	0
θ	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$
v	0	150	0	0
a	-300	300	—	—
ψ	$-\frac{\pi}{3}$	$\frac{\pi}{3}$	—	—

Table 3: Data for the Toy Car

The results are shown in Table 4. Both the interior point methods are faster than the active-set methods by a factor of more than ten. Graphs of the states and controls are given in Figs.

Algorithm	Cost	Iterations	CPU
`IPOPT`	0.8766	56	7.11
`LOQO`	0.8766	78	10.51
`MINOS`	0.8767	6710	125.73
`SNOPT`	0.8799	21966	214.47

Table 4: Results for the Toy Car

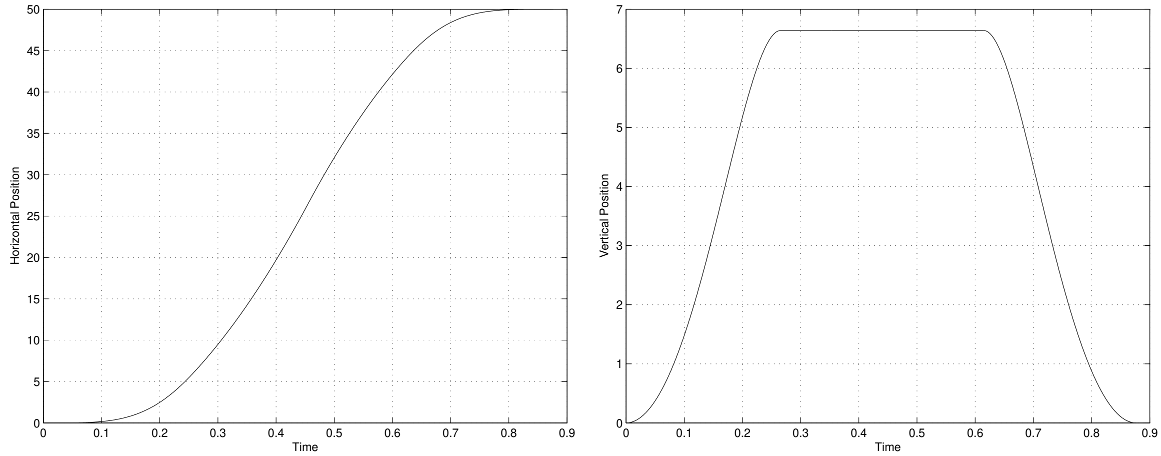

Figure 3: Horizontal and Vertical Position of the Toy Car

18 to 20. Both acceleration and steering angle are of bang-bang type. Note that none of the state constraints are active at the solution. They have been introduced to let `LOQO`, `MINOS`, and `SNOPT` find a solution, all of which outrun a time theshold of 1800 seconds without the state constraints.

6.2.3 Truck and Trailer

A single-axle trailer with length l is pivoted about the center point of the rear axle of a truck. Let ψ be the angle between the centerline of the truck and a reference axis x, α the angle between trailer and truck centerlines, and (x, y) the coordinates of the center point of the

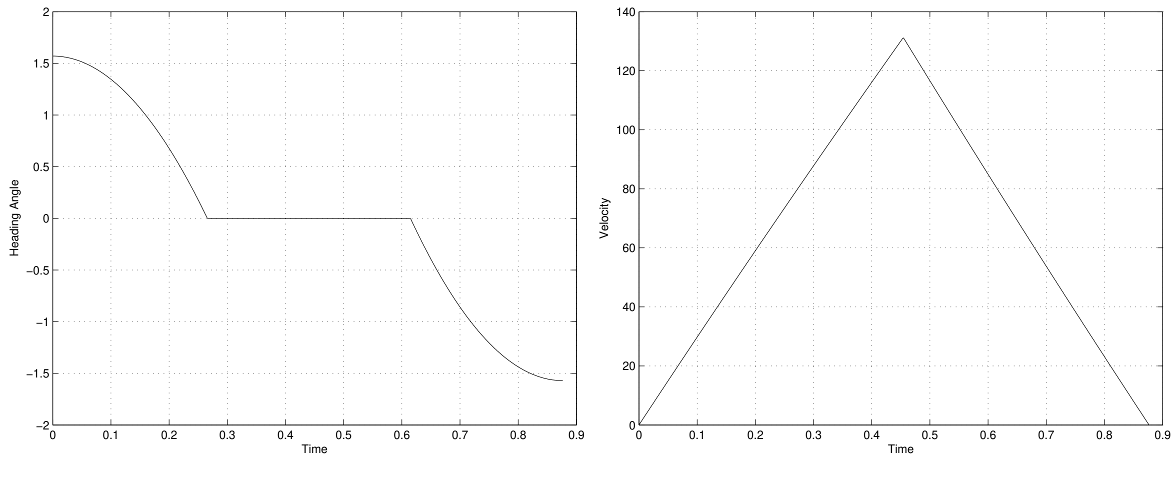

Figure 4: Heading Angle and Velocity of the Toy Car

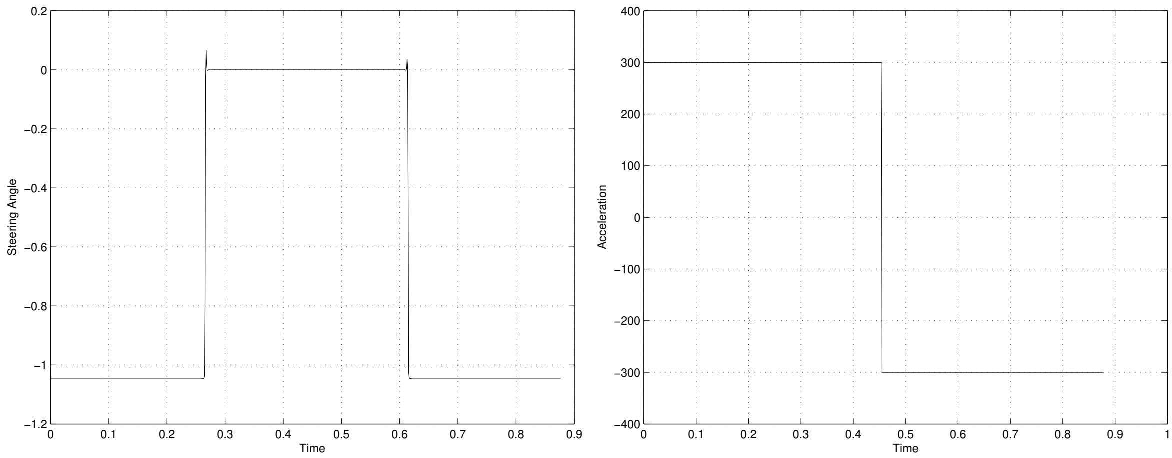

Figure 5: Steering Angle and Acceleration of the Toy Car

rear axle of the truck. The control variable is $u = (b/l)\tan(\delta)$, where δ is the steering angle of the truck front wheels and b the distance of the truck axles. The independent variable is d the distance traveled by the center point of the truck rear axle, the variable (d, x, y) are normalized to the length l.

The equations of motion are described in [12, pages 397–400]:

$$
\begin{aligned}
\psi' &= u, & \psi(0) &= 0, & \psi(f) &= -3\pi/4,\\
\alpha' &= u - \sin(\alpha), & \alpha(0) &= 0, & \alpha(f) &= 0,\\
x' &= \cos(\psi), & x(0) &= 0, & x(f) &= 1.3,\\
y' &= \sin(\psi), & y(0) &= 0, & y(f) &= -0.3.
\end{aligned}
\tag{6.6}
$$

The problem is to satisfy Eqs. (6.6) and to

$$
\begin{aligned}
&\max_{u(t)} \quad d_f,\\
&\text{s.t.} \quad u_{min} \le u(t) \le u_{max}.
\end{aligned}
\tag{6.7}
$$

Again, we use trapezoidal discretization (6.3) to approximate the differential equations (6.6) on a grid with $n = 1000$ intervals with the following initial guess for $i = 0, \dots, n$:

$$
\begin{aligned}
d_0 &= 1.0,\\
\psi_{i,0} &= 0.0,\\
\alpha_{i,0} &= 0.0,\\
x_{i,0} &= i/n,\\
y_{i,0} &= i/n,\\
u_{i,0} &= (1 - 2i/n)u_{max}.
\end{aligned}
$$

Only `IPOPT` is able to solve this problem within 196 iterations and _total_solve_time = 17.22 with final value of objective function $d_f = 5.899480$. Fig. 6 shows the control and the position of the truck.

Both `LOQO` and `SNOPT` stop "far from the solution" with iteration limit 10000 exceeded and with superbasics limit too small at 5050 iterations, respectively.

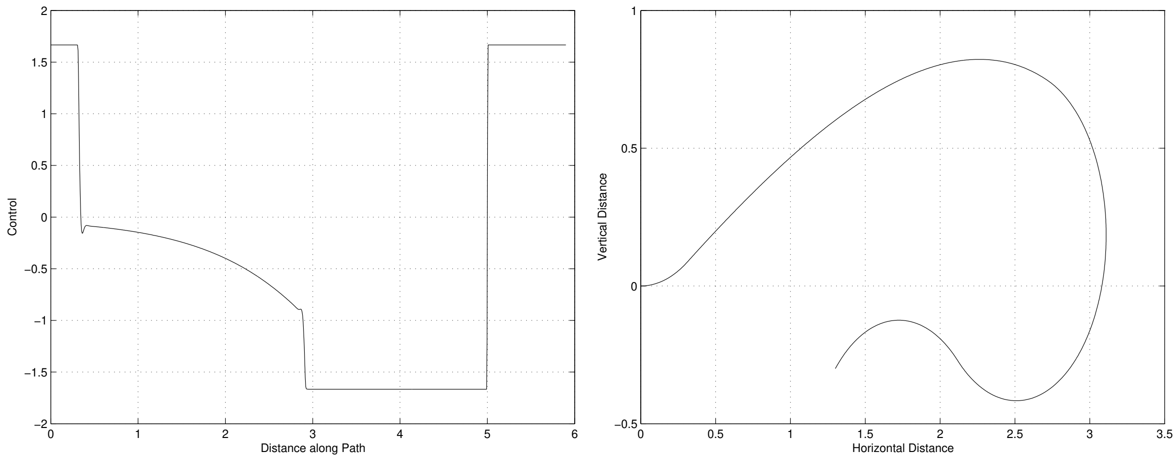

Figure 6: Control and Position of the Truck and Trailer Problem

6.2.4 Ship Cruise

The famous Zermelo problem of a ship cruising with velocity v in a position dependent flow field [13, page 67 ff] has been modified by Erb [25]. The dynamic model of the ship with coordinates x and y is given by

$$(6.8)\qquad \begin{aligned} \dot{x} &= v\cos\theta + w, \\ \dot{y} &= v\sin\theta, \end{aligned}$$

where θ is the heading angle of the ship and $w = a\frac{y}{h}$ the velocity component of the current. We define a rudder angle ζ and assume that it controls the velocity by the following law

$$(6.9)\qquad v = v_0\left(1 - \left(\frac{2\zeta}{\pi}\right)^2\right),$$

whereas it changes the heading angle by the following rule

$$(6.10)\qquad \dot{\theta} = \sin(b\zeta).$$

Also we assume that there is a linear dependency between actuating the rudder and the rudder position

$$(6.11)\qquad \dot{\zeta} = u.$$

Equations (6.8), (6.10), and (6.11) define the model of the vessel which starts at $x(t_0) = x_0,\ y(t_0) = y_0$ and shall arrive at $x(t_f) = x_f,\ y(t_f) = y_f$ in minimum time

$$(6.12)\qquad \min(t_f - t_0).$$

Now, consider that the vessel should have a rendezvous with a support ship which also starts at time t_0 at position $x_{s0},\ y_{s0}$ and moves along a line

$$(6.13)\qquad \begin{aligned} x_s &= x_{s0} - ct, \\ y_s &= y_{s0} - dt. \end{aligned}$$

The encounter of the two vehicles at time t_r defines a two-phase optimal control problem:[7]

[7] Multiple phase problems can be solved in `ampl` using the named problems construct [30, Section 14.5, Pages 311 ff.]. To show this technique we enclose the source code of `zermelo.mod` in Appendix 1.

- Phase 1: $t \in [t_0, t_r]$,
- Phase 2: $t \in [t_r, t_t]$.

Of course, the final time of phase 1 equals the initial time of phase 2, and the continuity of the trajectory is enforced by the equality constraints

$$
\begin{aligned}
x(tr-) &= x(tr+), \\
y(tr-) &= y(tr+), \\
\theta(tr-) &= \theta(tr-), \\
\zeta(tr-) &= \zeta(tr-).
\end{aligned} \tag{6.14}
$$

Also two state constraints[8] are defined: In phase 1 a circular coral reef centered at $x_c,\ y_c$ with radius r_c should not be entered

$$(x(t) - x_c)^2 + (y(t) - y_c)^2 \geq r_c^2, \quad \forall t \in]t_0, t_r[, \tag{6.15}$$

and in phase 2 a circular algae field centered at $x_a,\ y_a$ with radius r_a should not be entered

$$(x(t) - x_a)^2 + (y(t) - y_a)^2 \geq r_a^2, \quad \forall t \in]t_r, t_f[. \tag{6.16}$$

The data of the problem are $a = 0.7$, $b = 2.0$, $c = 0.0825$, $d = 0.55$, $h = 1.0$, $x_0 = 3.66$, $y_0 = -1.86$, $x_f = -3.0$, $y_f = 1.8$, $x_a = -1.2$, $y_a = 0.9$, $r_a = 1.0$, $x_c = 4.0$, $y_c = 1.1$, $r_c = 1.28$, $x_{s0} = 2.0$, $y_{s0} = 3.0$. The minimum time solution is composed of $t_f = t_1 + t_2 = 4.19201 + 3.25396 = 7.44597$. The timing results using a trapezoidal discretization (6.3) on a uniform mesh over $n = 100$ intervals are shown in Table 5, where also the needed iterations for solving phase 1 and 2 respectively are presented.[9]

solver	`filterSQP`	`ipopt`	`knitro`	`lancelot`	`loqo`	`minos`	`npopt`	`npsol`	`snopt`	`donlp2`	`cfsqp`
time	1.79	0.49	0.44	28.70	0.59	1.41	855.13	576.79	2.65	3997.35	39894
iter1	10	16	17	336	32	386	214	214	901	399	1000
iter2	11	22	20	184	22	504	33	37	834	66	1000

Table 5: Total Solving Time

In this test, also the midpoint rule, i.e. collocation at the Gaussian point $\rho_i = 0.5$ has been applied. The results are shown in Table 6.

solver	`filterSQP`	`ipopt`	`knitro`	`lancelot`	`loqo`	`minos`	`npopt`	`npsol`	`snopt`	`donlp2`	`cfsqp`
time	1.79	0.49	0.44	28.70	0.59	1.41	855.13	576.79	2.65	3997.35	39894
iter1	10	16	17	336	32	386	214	214	901	399	1000
iter2	11	22	20	184	22	504	33	37	834	66	1000

Table 6: Total Solving Time

The cruise is demonstrated graphically in the $x-y-$plane in Fig. 7, while the control function u for the two phases is given in Fig. 8.

[8] These physical constraints are augmented by box constraints on all states to alleviate the work of the minimizers; some of them cannot solve the problem because of making too large steps. These constraints can be taken from the source code which is given in Appendix 1.

[9] From the timing results, this problem looks like an easy one. But try to change the initial estimates in the data file, and you can make the test arbitrarily difficult. For the default (zero) values of the variables, no solver is able to create a solution.

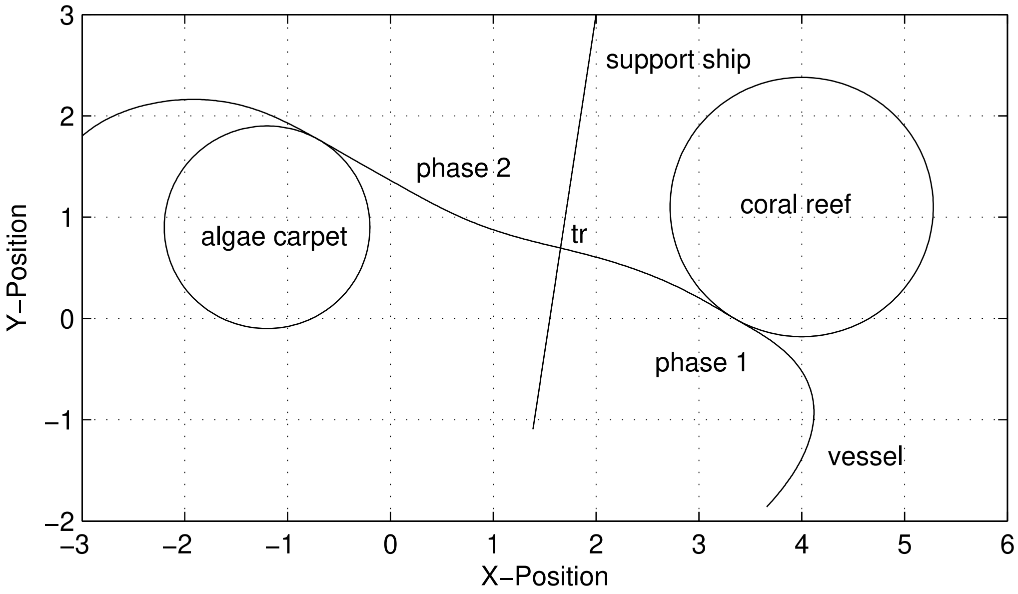

Figure 7: Two-phase Ship Cruise

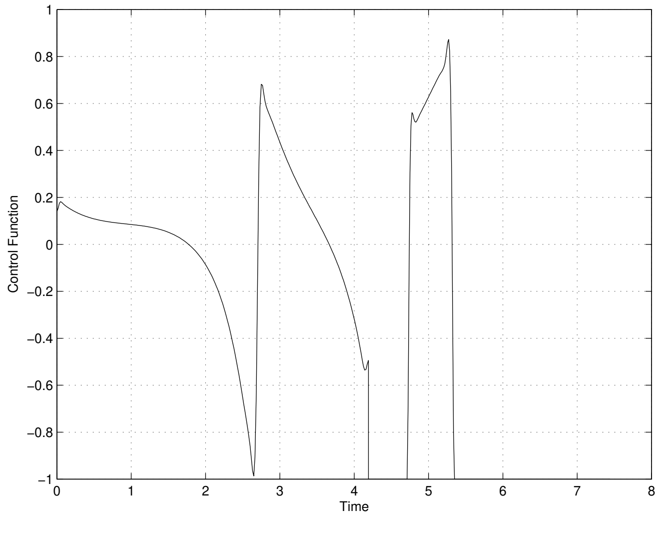

Figure 8: Ship Cruise: Control vs. Time

6.2.5 Robot Arm

A very much simplified model of a two-link robot arm with a spherical and a translational joint in spherical coordinates is [23]:

$$
\begin{aligned}
L\rho'' &= u_\rho, & \rho(t_0) &= 4.5, & \rho(t_f) &= 4.5, & \rho'(t_0) &= \rho'(_{tf}) = 0,\\
J_\theta\theta'' &= u_\theta, & \theta(t_0) &= 0, & \theta(t_f) &= \frac{2}{3}\pi, & \theta'(t_0) &= \theta'(_{tf}) = 0,\\
J_\phi\phi'' &= u_\phi, & \phi(t_0) &= \frac{\pi}{4}, & \phi(t_f) &= \frac{\pi}{4}, & \phi'(t_0) &= \phi'(_{tf}) = 0,
\end{aligned}
\tag{6.17}
$$

where L is the length of the base arm and

$$J_\theta = \frac{(L-\rho)^2+\rho^3}{3}\sin(\phi)^2,$$

and

$$J_\phi = \frac{(L-\rho)^2+\rho^3}{3},$$

are the mass moments of inertia about the θ- and ϕ-axes, respectively. The model is both state- and control-constrained:

$$\rho(t) \in [0, L], \quad |\theta(t)| \le \pi, \quad 0 \le \phi(t) \le \pi, \tag{6.18}$$

and

$$|u_\rho(t)| \le 1, \quad |u_\theta(t)| \le 1, \quad |u_\phi(t)| \le 1. \tag{6.19}$$

To enhance the operational effectiveness of the robot, a minimum-time solution of Eqs. (6.17), under the constraints (6.18) and (6.19) is searched. The second-order differential equations are transformed into a system of first-order differential equations representing positions and velocities, and these are again approximated by a trapezoidal discretization (6.3) on a uniform mesh over n intervals.

While `LOQO` is not able to solve this problem (iteration limit 10000 exceeded), `IPOPT` and `SNOPT` generate the solution $t_f = 9.140947$ with _total_solve_time shown in line 8 of Tab. 2. It ist interesting to note that the number of iterations (10654 for `SNOPT` and 26 for `IPOPT`) gives time/iteration ratio of 0.0129 and 0.1731 for the active-set and interior-point code, respectively. I.e. one iteration in the interior-point code takes more then ten times as much time then in the active-set code. This has been observed already in [22].

In Fig. 9 the state ρ and the (bang-bang-)control u_ρ of the prismatic joint is shown, representatively.

6.2.6 Hang Glider

The problem of maximizing the range of a hang glider in the presence of a thermal upwind is described in [27]:

$$
\begin{aligned}
\dot{x} &= v_x, & x(t_0) &= 0, & x(t_f) &\overset{!}{=} \max,\\
\dot{v}_x &= \frac{1}{m}(-L\sin(\eta) - D\cos(\eta)), & v_x(t_0) &= 13.23, & v_x(t_f) &= 13.23,\\
\dot{y} &= v_y, & y(t_0) &= 1000, & x(t_f) &= 900,\\
\dot{v}_y &= \frac{1}{m}(L\cos(\eta) - D\sin(\eta) - W), & v_y(t_0) &= -1.288, & v_x(t_f) &= 1.288,
\end{aligned}
\tag{6.20}
$$

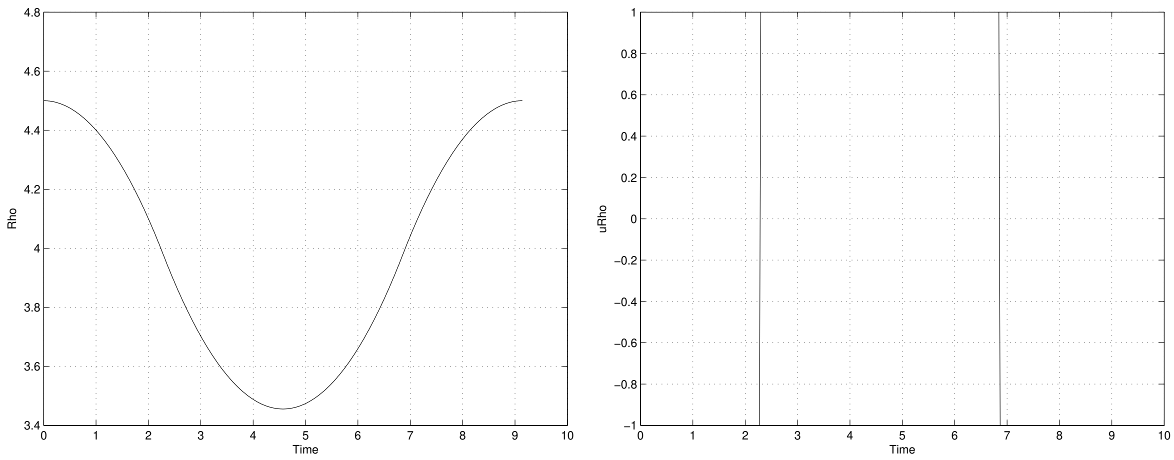

Figure 9: State ρ and Control u_ρ of the Robot Problem

with lift L, drag D, and weight W, respectively,

$$L = \frac{1}{2} c_L \rho S v_R^2, \quad D = \frac{1}{2} c_D \rho S v_R^2, \quad W = mg,$$

and flight path angle η and resultant velocity v_R

$$\eta = \arctan\big(\frac{v_y - u_A}{v_x}\big), \quad v_R = \sqrt{v_x^2 + (v_y - u_A)^2},$$

where $u_A(x)$ is the upwind profile

$$u_A(x) = u_{A_{max}} \exp\Big(-\big(\frac{x}{R} - 2.5\big)^2\Big)\Big(1 - \big(\frac{x}{R} - 2.5\big)^2\Big).$$

The glider is controlled by the constrained lift coefficient $c_L \leq c_{L_{max}}$ which influences the

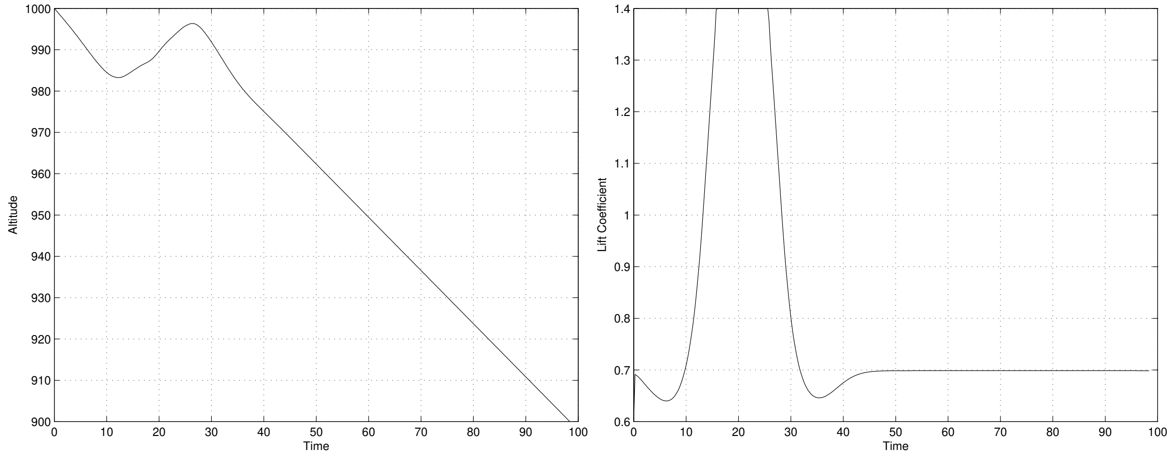

Figure 10: Altitude and Control c_L of the Hang Glider Problem

drag by a quadratic polar

$$c_D = c_{D_0} + k c_L^2.$$

Data of the problem are collected as

$$\begin{array}{rrr} R = 100[\mathrm{m}], & u_{A_{max}} = 2.5[\mathrm{ms}^{-1}], & c_{L_{max}} = 1.4, \\ c_{D_0} = 0.034, & k = 0.069662, & m = 100[\mathrm{kg}], \\ S = 14[\mathrm{m}^2], & g = 9.81[\mathrm{ms}^{-2}], & \rho = 1.13[\mathrm{kgm}^{-3}]. \end{array}$$

The results of Table 11.2 in the `COPS` set and those documented here are contradictory:

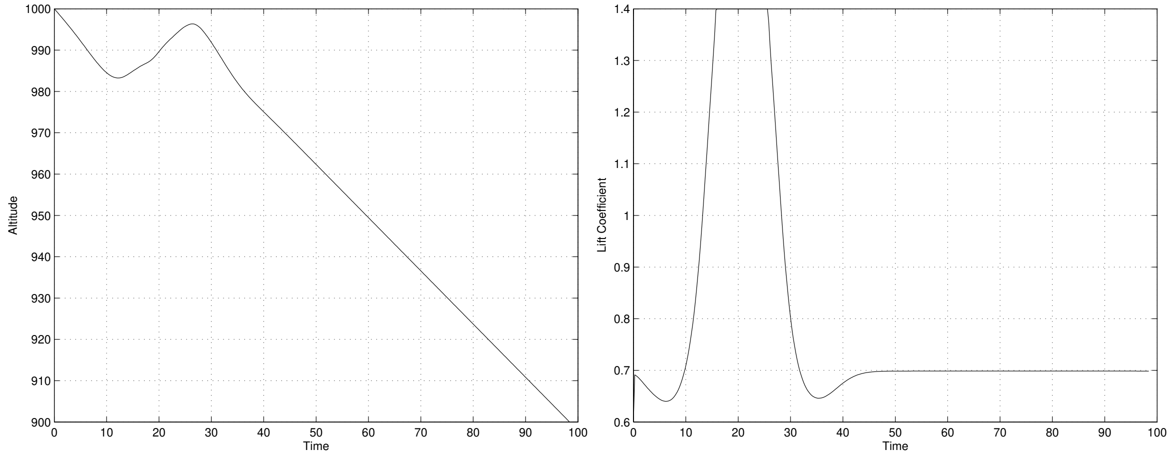

Figure 11: Horizontal and Vertical Velocity of the Hang Glider Problem

`SNOPT` there solves with parameter set $n_h = 100$ and fails for both $n_h = 200$ and 400; here it fails for $n_h = 100$ and 400 while it solves for $n_h = 200$. `LOQO` there solves with parameter set $n_h = 200$ and fails for both $n_h = 100$ and 400; here it fails for $n_h = 200$ and 400 while it solves for $n_h = 100$. `IPOPT` solves all three cases as can be seen from line 11 in Table 2. Also, the free end-time is $t_f = 98.41$ instead of 105 as reported in [10]. Graphs of the states and the lift coefficient c_L are shown in Fig. 10 and Fig. 11.

6.2.7 Shuttle Reentry

The shuttle reentry problem is one of the classical problems in optimal control calculations [18, pages 191–197]; here we follow the model description given in [32, pages 133–138]:

$$
\begin{aligned}
&\max \theta(t_f), \\
&\text{subject to} \\
&\dot{h} = v\sin(\gamma), && h(t_0) = 260000, && h(t_f) = 80000, \\
&\dot{\theta} = \frac{v}{r}\cos(\gamma)\cos(\psi), && \theta(t_0) = 0, && \theta(t_f) = \text{free}, \\
&\dot{v} = -\frac{D}{m} - g\sin(\gamma), && v(t_0) = 25600, && v(t_f) = 2500, \\
&\dot{\gamma} = \frac{L}{mv}\cos(\beta) + \left(\frac{v}{r} - \frac{g}{v}\right)\cos(\gamma), && \gamma(t_0) = -1, && \gamma(t_f) = -5, \\
&\dot{\psi} = \frac{L}{mv\cos(\gamma)}\sin\beta + \frac{v}{r}\cos(\gamma)\sin(\psi)\tan(\theta), && \psi(t_0) = 0, && \psi(t_f) = \text{free}, \\
&0 \le h, \\
&1 \le v, \\
&\theta_{min} \le \theta \le \theta_{max}, \\
&\gamma_{min} \le \gamma \le \gamma_{max}, \\
&\alpha_{min} \le \alpha \le \alpha_{max}, \\
&\beta_{min} \le \beta \le \beta_{max}, \\
&q = q_a q_r \le q_{max}..
\end{aligned} \tag{6.21}
$$

The following data (all in English units, and α in degree) characterize the problem:

$$
\begin{aligned}
&a_0 = -0.20704, && a_1 = 0.029244, && b_0 = 0.07854, \\
&b_1 = -0.61592 \times 10^{-2}, && b_2 = 0.621408 \times 10^{-3}, && c_0 = 1.0672181, \\
&c_1 = -1.9213774 \times 10^{-2}, && c_2 = 2.1286289 \times 10^{-4}, && c_3 = -1.0117249 \times 10^{-6}, \\
&h_r = 23800, && \mu = 1.4076539 \times 10^{16}, && \rho_0 = 2.378 \times 10^{-3}, \\
&m = 6309.44, && R_e = 2.09029 \times 10^{7}, && S = 2690, \\
&D = \frac{1}{2}c_D S\rho v^2, && L = \frac{1}{2}c_L S\rho v^2, && g = \frac{\mu}{r^2}, \quad q_{max} = 70, \\
&r = R_e + h, && \rho = \rho_0\exp\left(-\frac{h}{h_r}\right), && c_L = a_0 + a_1\alpha, \\
&c_D = b_0 + b_1\alpha + b_2\alpha^2, && q_r = 17700\sqrt{\rho}\left(\frac{v}{10000}\right)^{3.07}, && q_a = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3, \\
&\theta_{max} = -\theta_{min} = 89, && \gamma_{max} = -\gamma_{min} = 89, && \alpha_{max} = -\alpha_{min} = 90, \\
&\beta_{min} = -89, && \beta_{max} = 1.
\end{aligned}
$$

Again, we use trapezoidal discretization (6.3) to approximate the differential equations (6.6) with the following initial guess for $i = 0, \ldots, n$, where we vary the mesh size n to simulate an

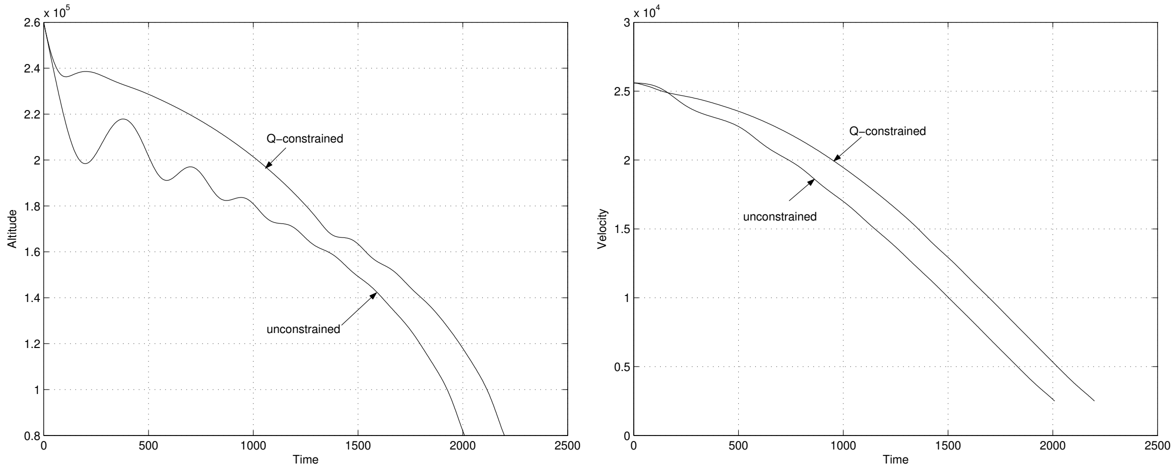

Figure 12: Altitude and Velocity of the Shuttle Problem

ad-hoc mesh-refinement procedure.

$$
\begin{aligned}
t_{f_0} &= 2000.0,\\
h_{i,0} &= h(t_0) - i/n(h(t_0) - h(t_f)),\\
v_{i,0} &= v(t_0) - i/n(v(t_0) - v(t_f)),\\
\psi_{i,0} &= \psi(t_0) - i/n(\psi(t_0) - \psi(t_f)),\\
\theta_{i,0} &= \theta(t_0) + i/n(\theta(t_f) - \theta(t_0)),\\
\gamma_{i,0} &= \gamma(t_0) - i/n(\gamma(t_0) - \gamma(t_f))^4,\\
\alpha_{i,0} &= 17.4\pi/180,\\
\beta_{i,0} &= (i/n - 1)75\pi/180.
\end{aligned}
$$

Only `IPOPT` can solve this problem, both the competitors stop with "iteration limit exceeded". We summarize the results in Table 7. To give an idea of the "size" of the problem, the number of variables (vars), constraints (constr), lower and upper bounds (lbounds, ubounds) is appended.

n	t_f	$\theta(t_f)$	iter	time	vars	constr	lbounds	ubounds
250	2008.5932	34.14094	84	11.01	1750	1250	1499	1000
500	2008.5811	34.14112	88	23.60	3500	2500	2999	2000
750	2008.5749	34.14115	119	53.26	5250	3750	4499	3000
1000	2008.5696	34.14117	115	71.45	7000	5000	5999	4000
1500	2008.5769	34.14118	145	147.14	10500	7500	8999	6000
2000	2008.5499	34.14118	137	158.62	14000	10000	11999	8000

Table 7: Mesh-Refinement for the Shuttle Problem

6.2.8 Low-Thrust Orbit Transfer

The following problem is generally considered a hard one [32, page 155]. The motion of a space vehicle in Cartesian coordinates is governed by the following equation

$$(6.22)\quad \ddot{\mathbf{r}} + \mu\frac{\mathbf{r}}{r^3} = \mathbf{a}_d,$$

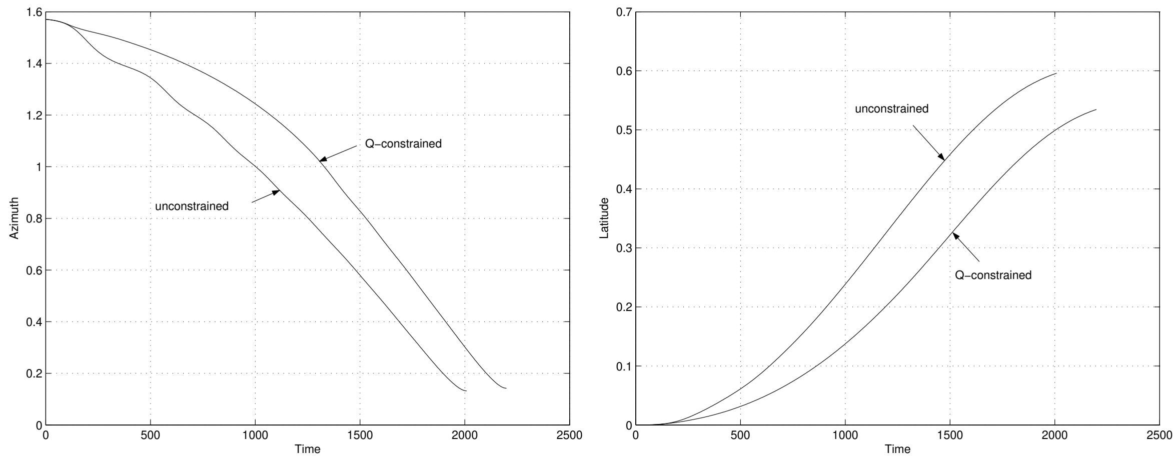

Figure 13: Azimuth and Latitude of the Shuttle Problem

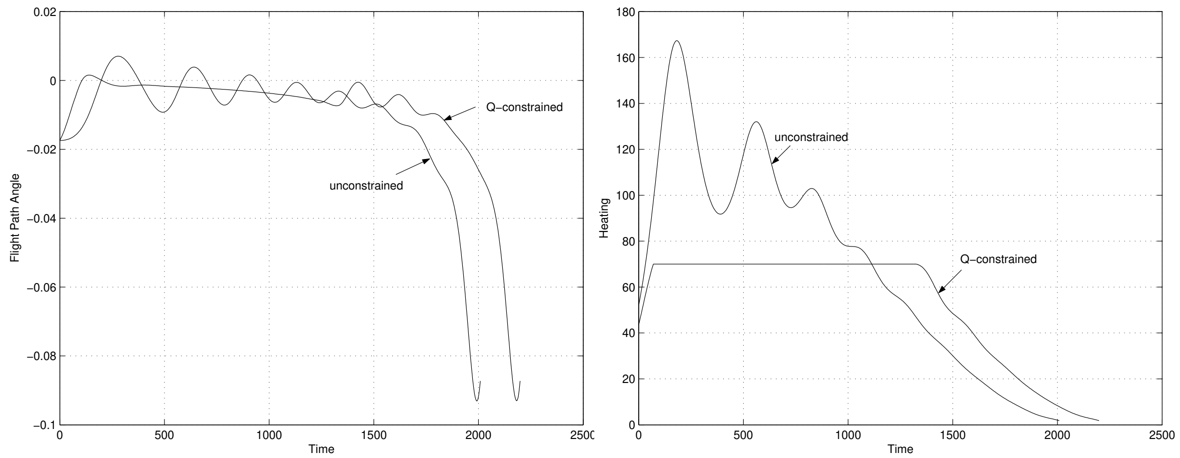

Figure 14: Flight Path and Heating of the Shuttle Problem

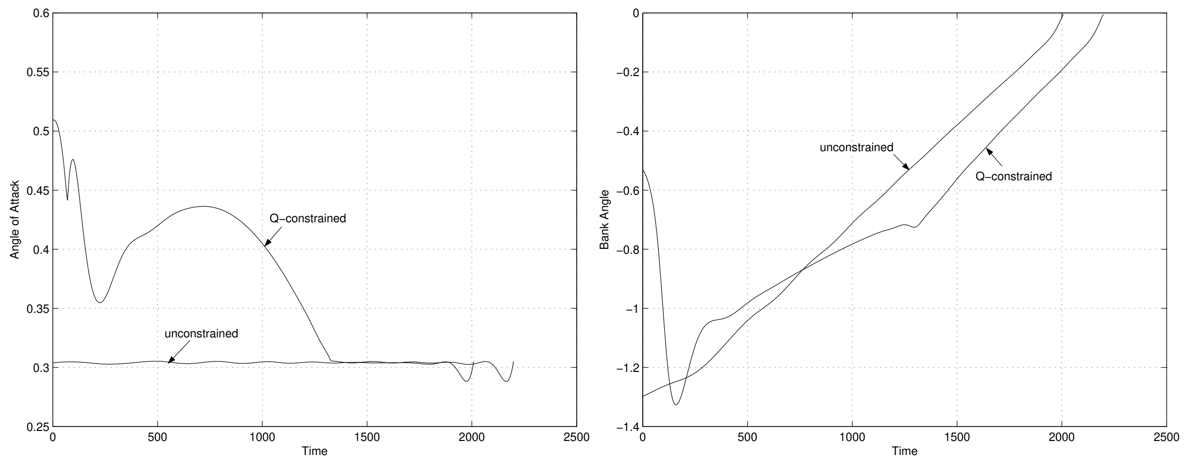

Figure 15: Angle of Attack and Bank Angle of the Shuttle Problem

where $\mathbf{r}$ is the inertial position vector and r its magnitude, $\mathbf{a}_d$ is the is the disturbing acceleration, and μ the gravitational constant. Here, the minimum-fuel transfer of a spacecraft from a low circular earth orbit to a mission orbit is described in modified equinoctial elements $\mathbf{y} = (p, f, g, h, k, L)'$ as follows [16]:

(6.23) $\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta} + \mathbf{b}$,

where the dynamics matrix is

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{2p}{q}\sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}}\sin L & \frac{1}{q}\sqrt{\frac{p}{\mu}}((1+q)\cos L + f) & -\frac{g}{q}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L) \\ -\sqrt{\frac{p}{\mu}}\cos L & \frac{1}{q}\sqrt{\frac{p}{\mu}}((1+q)\sin L + g) & \frac{f}{q}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L) \\ 0 & 0 & \frac{s^2}{2q}\sqrt{\frac{p}{\mu}}\cos L \\ 0 & 0 & \frac{s^2}{2q}\sqrt{\frac{p}{\mu}}\sin L \\ 0 & 0 & \frac{1}{q}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L) \end{pmatrix}$$

and the vector

$$\mathbf{b} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \sqrt{\mu p}\left(\frac{p}{q}\right)^2 \end{pmatrix}.$$

The equinoctial elements $\mathbf{y}$ are related to Cartesian coordinates $(\mathbf{r}, \mathbf{v})$ as follows

$$\mathbf{r} = \begin{pmatrix} \frac{r}{s^2}((1+\alpha^2)\cos L + 2hk\sin L) \\ \frac{r}{s^2}((1-\alpha^2)\sin L + 2hk\cos L) \\ \frac{2r}{s^2}(h\sin L - k\cos L) \end{pmatrix},$$

$$\mathbf{v} = \begin{pmatrix} -\frac{1}{s^2}\sqrt{\frac{p}{\mu}}((1+\alpha^2)\sin L - 2hk\cos L + (1+\alpha^2)g - 2fhk) \\ -\frac{1}{s^2}\sqrt{\frac{p}{\mu}}((-1+\alpha^2)\cos L + 2hk\sin L + (-1+\alpha^2)f - 2ghk) \\ \frac{2}{s^2}\sqrt{\frac{p}{\mu}}(h\cos L + k\sin L + fhgk) \end{pmatrix}.$$

The following variables are used

$$\begin{aligned} q &= 1 + f\cos L + g\sin L, \\ r &= \frac{p}{q}, \\ \alpha^2 &= h^2 - k^2, \\ s^2 &= 1 + h^2 + k^2. \end{aligned}$$

The disturbing acceleration vector $\mathbf{a}_d$ in Eq. (6.22) is replaced by Δ in Eq. (6.23) which is composed of two parts

$$\boldsymbol{\Delta} = \boldsymbol{\Delta}_g + \boldsymbol{\Delta}_T,$$

the first representing the disturbance by the oblate earth and the second that of the thrust.

We obtain $\boldsymbol{\Delta}_g$ as

$$\boldsymbol{\Delta}_g = \mathbf{Q}'\boldsymbol{\delta}\mathbf{g} = \delta g_n \mathbf{i}_n - \delta g_r \mathbf{i}_r,$$

where

$$\delta g_n = -\frac{\mu\cos\phi}{r^2}\sum_{k=2}^{4}\left(\frac{R_e}{r}\right)^k P_k' J_k,$$

and

$$\delta g_r = -\frac{\mu}{r^2}\sum_{k=2}^{4}(1+k)\Big(\frac{R_e}{r}\Big)^k P_k J_k,$$

and ϕ is the geocentric latitude

$$\sin\phi = \frac{2}{s^2}(h\sin L - k\cos L),$$

$P_k(\sin\phi)$ is the k^{th}-order Legendre polynomial, P_k' it's derivative and J_k are the zonal harmonic coefficients. The matrix $\mathbf{Q}$ is built up of the unit vectors

$$\mathbf{Q} = \begin{pmatrix}\mathbf{i}_r & \mathbf{i}_\theta & \mathbf{i}_h\end{pmatrix} = \begin{pmatrix}\frac{\mathbf{r}}{\|\mathbf{r}\|} & \frac{(\mathbf{r}\times\mathbf{v})\times\mathbf{r}}{\|\mathbf{r}\times\mathbf{v}\|\|\mathbf{r}\|} & \frac{\mathbf{r}\times\mathbf{v}}{\|\mathbf{r}\times\mathbf{v}\|}\end{pmatrix},$$

and the unit vector defining the local north direction is, with $\mathbf{e}_n = (0,0,1)'$,

$$\mathbf{i}_n = \frac{\mathbf{e}_n - (\mathbf{e}_n'\mathbf{i}_r)\mathbf{i}_r}{\|\mathbf{e}_n - (\mathbf{e}_n'\mathbf{i}_r)\mathbf{i}_r\|}.$$

The thrust component of $\mathbf{\Delta}$ is

$$\mathbf{\Delta_T} = \frac{(1+\tau/100)Tg_0}{w}\mathbf{u},$$

where $\mathbf{u} = (u_r, u_\theta, u_h)'$ is the vector of control variables, T is the magnitude of the (low) thrust, τ is a throttle factor, and w is the weight of the vehicle.

The development of the equinoctial elements in Eq. (6.23) is augmented by the weight equation

$$\dot{w} = -\frac{(1+\tau/100)T}{I_{sp}}, \tag{6.24}$$

in which I_{sp} is the specific impulse of the engine.

Now, we are in a position to complement the problem formulation:

$$\max_{\mathbf{u}} w(t_f) \tag{6.25}$$

subject to Eq. (6.23) and Eq. (6.24) and to the control constraints

$$\begin{aligned}\|\mathbf{u}\| &= 1,\\ \tau_L \le \tau &\le 0,\end{aligned} \tag{6.26}$$

with initial conditions

$$\begin{aligned}p(t_0) &= 21837080.052835,\\ f(t_0) &= 0.0,\\ g(t_0) &= 0.0,\\ h(t_0) &= -0.25396764647494,\\ k(t_0) &= 0.0,\\ L(t_0) &= \pi,\\ w(t_0) &= 1.0,\end{aligned}$$

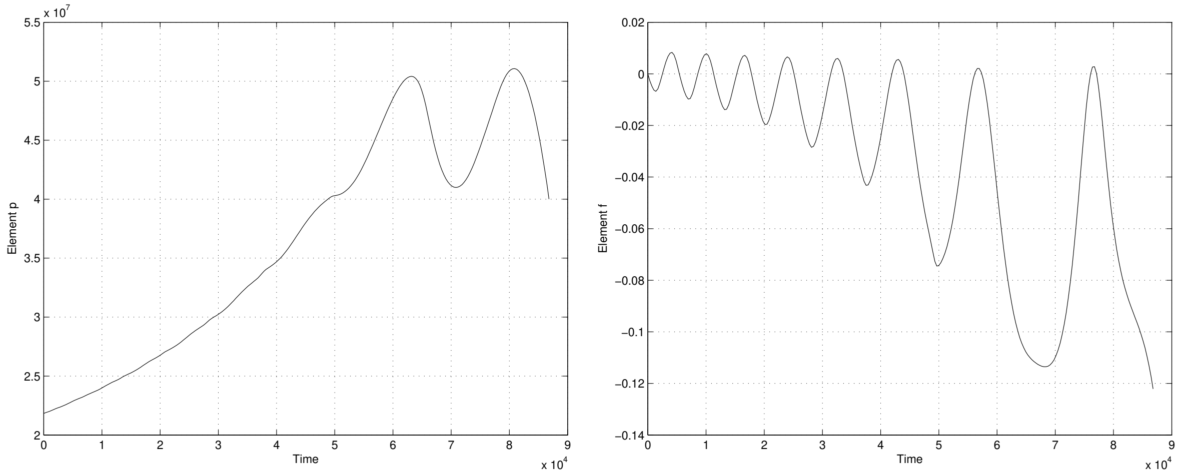

Figure 16: Equinoctial Elements of the Orbiter

and final conditions

$$\begin{aligned}
p(t_f) &= 40007346.015232,\\
\sqrt{f^2+g^2} &= 0.73550320568829,\\
\sqrt{h^2+k^2} &= 0.61761258786099,\\
fh+gk &= 0.0,\\
gh-fk &\le 0.0.
\end{aligned}$$

The following constants complete the problem

$$\begin{array}{lll}
\mu = 1.407645794 \times 10^{16}, & g_0 = 32.174, & I_{sp} = 450,\\
T = 4.446618 \times 10^{-3}, & R_e = 20925662.73, & \tau_L = -50.0,\\
J_2 = 1082.639 \times 10^{-6}, & J_3 = -2.565 \times 10^{-6}, & J_4 = -1.608 \times 10^{-6}.
\end{array}$$

To transcribe the optimal control problem into a finite-dimensional nonlinear programming problem we again use trapezoidal discretization (6.3) to approximate the differential equations. The initial guess can be taken from the Appendix in which we give the file orbit.mod for the convenience of the reader.

Only `IPOPT` is possible to solve this (hard) problem. We present The results in Table

n	t_f	$w(t_f)$	τ	niter	cputime
150	86629	0.2202604590366861	-8.91038	661	403
200	86796	0.2202252223455872	-9.08187	839	686
250	86802	0.2202070599215732	-9.08633	889	899
400	80943	0.2001646797364390	5.5219e-9	827	1376

Table 8: Results of the Orbiter Problem

State and control variables are shown in Figs. 16–19.

6.2.9 Cam Shape

Although the optimization of the shape of a cam is not an optimal control problem it plays an important role in the design and simulation of dynamic systems. Therefore it is included in this report.

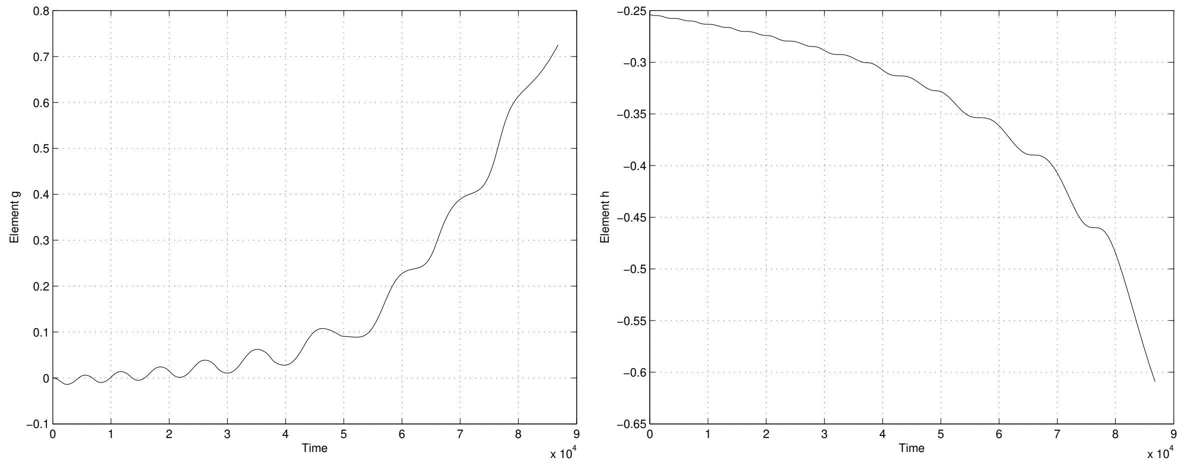

Figure 17: Equinoctial Elements of the Orbiter, contd.

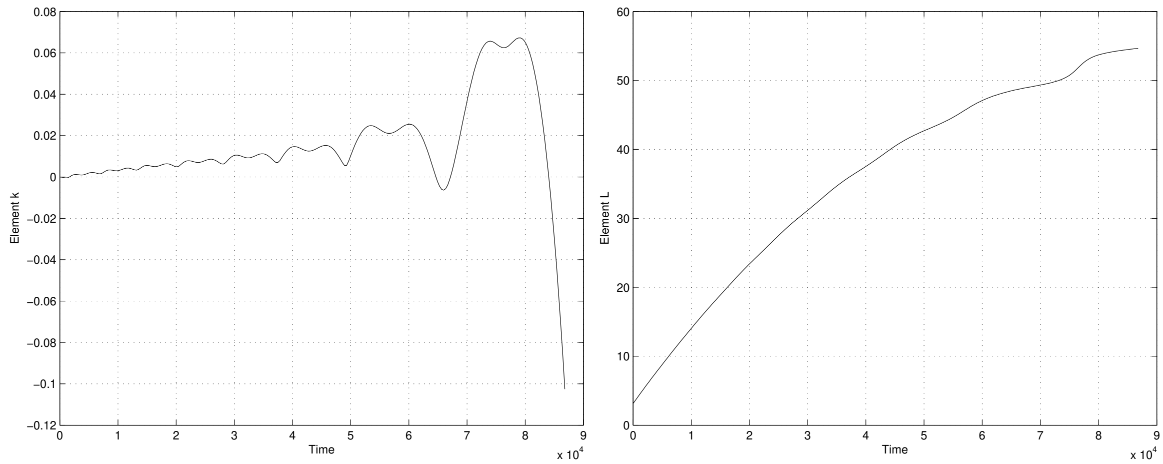

Figure 18: Equinoctial Elements of the Orbiter, contd.

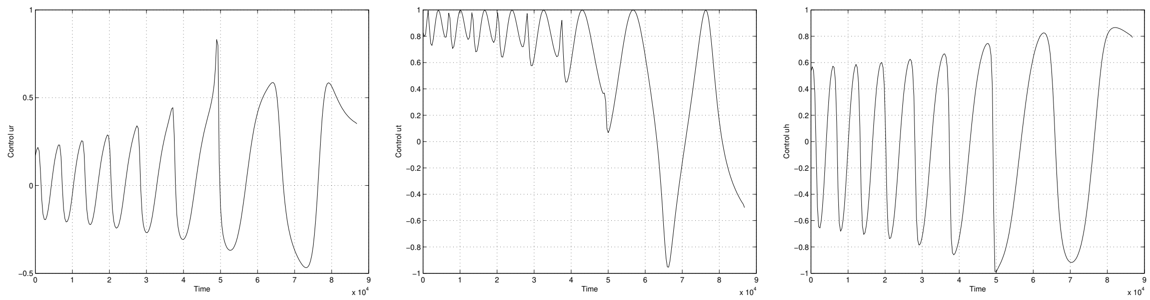

Figure 19: Control Variables of the Orbiter Problem

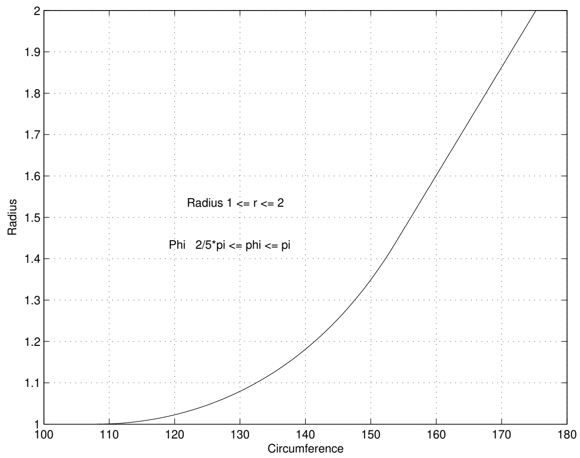

Figure 20: Radius of the Cam-Shape Problem

The design goal is to maximize the area of the valve opening

$$(6.27) \quad f(r) = \frac{\pi r_v^2}{n} \sum_{i=1}^{n} r_i$$

for one rotation of a convex cam with constraints on the curvature and on the radii of the cam. The design variables, $r_{min} \leq r_i \leq r_{max}$, $i = 1, \dots, n$, represent the radii of the cam at equally spaced angles distributed over an angle of $\frac{4}{5}\pi$. The remaining part $(\frac{6}{5}\pi)$ of the circumference is circular with radius r_{min}. The convexity constraints can be formulated as

$$A(r_{i-1}, r_{i+1}) \leq A(r_{i-1}, r_i) + A(r_i, r_{i+1}),$$

where $A(r_i, r_j)$ is the area of the triangle defined by the origin and the points r_i and r_j on the surface. This inequality is equivalent to

$$(6.28) \quad 2r_{i-1}r_{i+1}\cos(\theta) \leq r_i(r_{i-1} + r_{i+1}), \quad i = 0, \dots, n+1,$$

where $r_{-1} = r_0 = r_{min}$, $r_{n+1} = r_{max}$, $r_{n+2} = r_n$, and $\theta = \frac{2}{5(n+1)}\pi$ is the angle between discretization points. The curvature constraint is expressed by

$$(6.29) \quad -\alpha \leq \frac{r_{i+1} - r_i}{\theta} \leq \alpha.$$

The following data are used: $r_{min} = 1$, $r_{max} = 2$, $r_v = 1$, and $\alpha = 1.5$.

All three solvers solve the problem as can be seen from line 3 of Table 2 with a strange behavior of `LOQO` for the lower number of parameters. Also, in this "easy" problem the interior-point solver `IPOPT` shows an advantegeous behavior compared to the active-set solver `SNOPT`. The radius is constant $r = r_{\min}$ along the angle $0 \leq \phi \leq 2\pi/5$ until it reaches the value $r = r_{\max}$ at about 175 as is shown in Fig. 14 for the half-circumference of the cam. The modeling of this problem is certainly improvable as the shape of the cam is not continuously differentiable at $\phi \approx 180$.

Acknowledgements. I am grateful to Philip Gill, Michael Saunders and Bob Vanderbei for providing sources and/or permanent full licenses of their codes `SNOPT`, `MINOS` and `LOQO`, respectively. Also, I would like to thank Richart Waltz for granting me a test license of `KNITRO`. I thank Andreas Wï¿½hter for extensive discussion of his code `IPOPT` and I admire his decision to publish it as open source code. May this be an encouraging example for other code developers to follow. I appreciate the suggestions Klaus Well and Sven Erb made for some applications.

References

[1] http://www-124.ibm.com/developerworks/oss/coin/Ipopt/.

[2] http://www.opensource.org/licenses/cpl.php.

[3] http://www-unix.mcs.anl.gov/∼more/cops/.

[4] Forsgren A. and P. E. Gill. Primal-dual interior methods for nonconvex nonlinear programming. *SIAM J. Optimization*, 8(4):1132–1152, 1998.

[5] Forsgren A., P. E. Gill, and M. H. Wright. Interior methods for nonlinear optimization. *SIAM Review*, 44(4):525–597, 2002.

[6] Wächter A. and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Technical report, IBM T.J. Watson Research Center, March 2004.

[7] U. M. Asher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998.

[8] De Boor C. and B. Swartz. Collocation at gauusian points. *SIAM J. Numerical Analysis*, 31(11):916–921, 1973.

[9] Moler C. *Numerical Computing with MATLAB*. SIAM, Philadelphia, 2004.

[10] Dolan E. D., J. J. More, and T. S. Munson. Benchmarking optimization with cops 3.0. Technical report, Argonne National Laboratory, 2004.

[11] I. S. Duff. Ma57–a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw.*, 30(2):118–144, 2004.

[12] Bryson A. E. *Dynamic Optimization*. Addison-Wesley, Menlo Park, 1999.

[13] Bryson A. E. and Y. C. Ho. *Applied Optimal Control*. Blaisdell, Waltham, MA, 1969.

[14] Gill P. E., W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM J. Optimization*, 12(4):979–1006, 2002.

[15] Jarre F. and J. Stoer. *Optimierung*. Springer, Berlin, 2004.

[16] Walker M. J. H., B. Ireland, and J. Owens. A set of modified equinoctial orbit elements. *Celestial Mechanics*, (36):809–419, 1985.

[17] Nocedal J. and S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[18] Stoer J. and R. Bulirsch. *Einfuehrung in die numerische Mathematik*, volume II. Springer–Verlag, Berlin, 1978.

[19] Vanderbei R. J. *Linear Programming, Foundations and Extensions*. Kluwer Academic Publishers, Boston, 1997.

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

[21] Gertz E. M. and P. E. Gill. A primal-dual trust region algorithm for nonlinear pragramming. Technical Report Numerical Analysis Group NA 02-1, University of California, San Diego, 2002.

[22] Gould N. I. M. Some reflections on the current state of active-set and interior-point methods for constrained optimization. *SIAG/OPT Views-and-News*, 14(1):2–7, April 2003.

[23] Mössner-Beigel M. Optimale steuerung für industrieroboter unter berücksichtigung der getriebebedingten elastizität. Master's thesis, IWR, Universität Heidelberg, 1999. unpublished.

[24] Karmarkar N. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[25] Erb S. O. Mit dem schiff von a nach b. Technical report, Universität Stuttgart, 2003. unpublished report.

[26] Bulirsch R. and D. Kraft, editors. *Computational Optimal Control*. Birkhäuser, Basel, 1994.

[27] Bulirsch R., E. Nerz, H. J. Pesch, and O. von Stryk. Combining direct and indirect methods in optimal control: Range maximization of a hang glider. In Bulirsch R., A. Miele, J. Stoer, and K. H. Well, editors, *Optimal Control*, pages 273–288, Boston, 1993.

[28] Byrd R., M. E. Hribar, and J. Nocedal. An interior point method for large scale nonlinear programming. *SIAM J. Optimization*, 9(4):877–900, 1999.

[29] Fletcher R. and S. Leyffer. Nonlinear programming without a penalty function. *Math. Program.*, 91:239–269, 2002.

[30] Fourer R., D. M. Gay, and B. Kernighan. *AMPL – A Modeling Language for Mathematical Programming*. Brooks/Cole-Thomson, Pacific Grove, second edition, 2003.

[31] Vanderbei R. and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. Technical Report SOR-97-21, Statistics and Operations Research, Princeton University, 1997.

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

A Ampl model file zermelo.mod

```
# Modified Zermelo problem
# Two-phase-problem
# as of S.O. Erb: Mit dem Schiff von A nach B
# unpublished report
# Institute of Flight Mechanics and Flight Control
# University of Stuttgart
# implemented by
# Dieter Kraft
# Munich University of Applied Sciences
# September 2004
# Collocation version

#I do not find an entry to compound commands in the 2nd edition
#Oops!  That's an oversight.  They're the if-then-else, looping,
#and {command; command;...} constructs described in section
#A.20.1 (pp. 493-494).

#First declare things, then issue commands.
```

```
param pi := 4*atan(1);
param n1 > 0, integer; # number of subintervals in phase I
param n2 > 0, integer; # number of subintervals in phase II
param nc > 0, integer; # number of collocation points
param ne > 0, integer; # number of differential equations
param rho {1..nc};     # roots of the k-th degree Legendre polynomial
param fact {j in 0..nc} := if j = 0 then 1 else (prod{i in 1..j} i);
param kappa;           # smoothing factor
param p{1..8};
param ul;
param uu;

var t1 >= 0, <= 4.5, := 4.20;
var t2 >= 0, <= 4.5, := 3.50;

var h1 = t1/n1;
var h2 = t2/n2;

var u1 {1..n1,1..nc};
var v1 {1..n1,1..ne};
var w1 {1..n1,1..nc,1..ne};

var u2 {1..n2,1..nc};
var v2 {1..n2,1..ne};
var w2 {1..n2,1..nc,1..ne};



# Evaluation of the piecewise polynomial and derivative at collocation points.

var pp1 {i in 1..n1, j in 1..nc, k in 1..ne} =
  v1[i,k] + h1*sum {l in 1..nc} w1[i,l,k]*(rho[j]^l/fact[l]);

var Dpp1 {i in 1..n1, j in 1..nc, k in 1..ne} =
  sum {l in 1..nc} w1[i,l,k]*(rho[j]^(l-1)/fact[l-1]);

var ppf1 {i in 1..ne} =
  v1[n1,i] + h1*sum {j in 1..nc} w1[n1,j,i]/fact[j];

var pp2 {i in 1..n2, j in 1..nc, k in 1..ne} =
  v2[i,k] + h2*sum {l in 1..nc} w2[i,l,k]*(rho[j]^l/fact[l]);

var Dpp2 {i in 1..n2, j in 1..nc, k in 1..ne} =
  sum {l in 1..nc} w2[i,l,k]*(rho[j]^(l-1)/fact[l-1]);

var ppf2 {i in 1..ne} =
  v2[n2,i] + h2*sum {j in 1..nc} w2[n2,j,i]/fact[j];

var switch >= 0, integer;

option donlp2_options   'outlev=0';
option filter_options   'outlev=0 maxiter=1000';
option fsqp_options     'outlev=0 maxiter=1000';
option ipopt_options    'iprint=-1';
option knitro_options   'outlev=0';
option lancelot_options 'outlev=0';
```

```
option loqo_options     'outlev=0';
option minlp_options    'outlev=1';
option minos_options    'outlev=1';
option npopt_options    'outlev=1';
option npsol_options    'outlev=1';
option snopt_options    'outlev=1';

let switch := 2;


if switch ==  1 then option solver filter;
if switch ==  2 then option solver ipopt;
if switch ==  3 then option solver knitro;
if switch ==  4 then option solver lancelot;
if switch ==  5 then option solver loqo;
if switch ==  6 then option solver minlp;
if switch ==  7 then option solver minos;
if switch ==  8 then option solver npopt;
if switch ==  9 then option solver npsol;
if switch == 10 then option solver snopt;
if switch == 11 then option solver donlp2;
if switch == 12 then option solver fsqp;



problem phase_1: t1, u1, v1, w1;
#################################


minimize tf1: t1;
#+ kappa/h1*sum{i in 1..n1-1, j in 1..nc} (u1[i+1,j] - u1[i,j])^2;

# Differential constraints

de1 {i in 1..n1, j in 1..nc}:
  Dpp1[i,j,1] = p[1]*((1-(2/pi*pp1[i,j,4])^2)*cos(pp1[i,j,3])-0.7*pp1[i,j,2]);

de2 {i in 1..n1, j in 1..nc}:
  Dpp1[i,j,2] = p[1]*(1-(2/pi*pp1[i,j,4])^2)*sin(pp1[i,j,3]);

de3 {i in 1..n1, j in 1..nc}:
  Dpp1[i,j,3] = sin(2*pp1[i,j,4]);

de4 {i in 1..n1, j in 1..nc}:
  Dpp1[i,j,4] = u1[i,j];

continuity1 {i in 1..n1-1, j in 1..ne}:
  v1[i,j] + h1*sum {k in 1..nc} w1[i,k,j]/fact[k] = v1[i+1,j];

u_bounds1 {i in 1..n1, j in 1..nc}: ul <= u1[i,j] <= uu;

# Algebraic constraints

ae1{i in 1..n1-1}: (v1[i,1]-4.0)^2 + (v1[i,2]-1.1)^2  >= p[6]^2;

# Control constraint
```

```
ce1{i in 1..n1, j in 1..nc}: ul <= u1[i,j] <= uu;

# Initial conditions

ic1: v1[1,1] = p[2];
ic2: v1[1,2] = p[3];

# Final conditions
# Encounter with support ship

fc1: 2.0-0.15*p[1]*p[8]*t1 = ppf1[1];
fc2: 3.0-p[1]*p[8]*t1      = ppf1[2];



problem phase_2: t2, u2, v2, w2;
#################################


minimize tf2: t2;
#+ kappa/h2*sum{i in 1..n2-1, j in 1..nc} (u2[i+1,j] - u2[i,j])^2;

# Differential constraints

de5 {i in 1..n2, j in 1..nc}:
  Dpp2[i,j,1] = p[1]*((1-(2/pi*pp2[i,j,4])^2)*cos(pp2[i,j,3])-0.7*pp2[i,j,2]);

de6 {i in 1..n2, j in 1..nc}:
  Dpp2[i,j,2] = p[1]*(1-(2/pi*pp2[i,j,4])^2)*sin(pp2[i,j,3]);

de7 {i in 1..n2, j in 1..nc}:
  Dpp2[i,j,3] = sin(2*pp2[i,j,4]);

de8 {i in 1..n2, j in 1..nc}:
  Dpp2[i,j,4] = u2[i,j];

continuity2 {i in 1..n2-1, j in 1..ne}:
  v2[i,j] + h2*sum {k in 1..nc} w2[i,k,j]/fact[k] = v2[i+1,j];

u_bounds2 {i in 1..n2, j in 1..nc}: ul <= u2[i,j] <= uu;

# Algebraic state constraints

ae2{i in 1..n2-1}: (v2[i,1]+1.2)^2 + (v2[i,2]-p[7])^2 >= 1;

# Initial conditions
# Phase connecting conditions

ph1: v2[1,1] = ppf1[1];
ph2: v2[1,2] = ppf1[2];
ph3: v2[1,3] = ppf1[3];
ph4: v2[1,4] = ppf1[4];

# Final conditions

fc3: ppf2[1] = p[4];
fc4: ppf2[2] = p[5];
```

```
commands z2.dat;

solve phase_1;

solve phase_2;

display t1, t2, t1+t2, _total_solve_time, solve_result_num, solve_result;


printf {i in 0..n1-1}:
"%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f \n",
i/n1*t1, {j in 1..4} v1[i+1,j], 4.0+p[6]*cos(i/n1*2*pi), 1.1+p[6]*sin(i/n1*2*pi),
2-0.15*p[1]*p[8]*i/n1*t1, 3-p[1]*p[8]*i/n1*t1 > z;
printf {i in 0..n2-1}:
"%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f \n",
t1+i/n2*t2, {j in 1..4} v2[i+1,j], -1.2+cos(i/n2*2*pi), p[7]+1.0*sin(i/n2*2*pi),
v1[n1,1]-0.15*p[1]*p[8]*i/n2*t2, v1[n1,2]-p[1]*p[8]*i/n2*t2 >> z;
```

B Ampl model file orbit.mod

```
# Low Thrust Orbit Maneuver; Betts Chap. 5.3 p. 147

param pi := 4*atan(1);
param Isp := 450;
param mu := 1.407645794e16;
param G0 := 32.174;
param J2 := 1082.639e-6;
param J3 := -2.565e-6;
param J4 := -1.608e-6;
param T := 4.446618e-3;
param Re := 209.2566273e5;
param tauL := -50.0;
param p0 := 218.37080052835e5;
param f0 := 0.0;
param g0 := 0.0;
param h0 := -0.25396764647494;
param k0 := 0.0;
param L0 := pi;
param w0 := 1.0;
param wf := 0.2; # estimated
param pf := 400.07346015232e5;
param fgf := 0.73550320568829;
param hkf := 0.61761258786099;

param n; # discretization intervals

var tf >= 0;
var step = tf/n;

# state variables

var p{0..n};
var f{0..n};
var g{0..n};
var h{0..n};
```

```
var k{0..n};
var L{0..n};
var w{0..n};

# control variables

var ur{0..n};
var ut{0..n};
var uh{0..n};
var tau; # throttle parameter

# dependent variables

var q{i in 0..n} = 1.0 + f[i]*cos(L[i]) + g[i]*sin(L[i]);
var r{i in 0..n} = p[i]/q[i];
var a2{i in 0..n} = h[i]^2 - k[i]^2;
var s2{i in 0..n} = 1.0 + h[i]^2 + k[i]^2;
var r1{i in 0..n} = r[i]/s2[i]*((1.0+a2[i])*cos(L[i])+2*h[i]*k[i]*sin(L[i]));
var r2{i in 0..n} = r[i]/s2[i]*((1.0-a2[i])*sin(L[i])+2*h[i]*k[i]*cos(L[i]));
var r3{i in 0..n} = 2*r[i]/s2[i]*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var v1{i in 0..n} = -1.0/s2[i]*sqrt(mu/p[i])
   *((1.0+a2[i])*sin(L[i])-2*h[i]*k[i]*cos(L[i])+(1.0+a2[i])*g[i]-2*f[i]*h[i]*k[i]);
var v2{i in 0..n} = -1.0/s2[i]*sqrt(mu/p[i])
   *((-1.0+a2[i])*cos(L[i])+2*h[i]*k[i]*sin(L[i])+(-1.0+a2[i])*f[i]+2*g[i]*h[i]*k[i]);
var v3{i in 0..n} = 2.0/s2[i]*sqrt(mu/p[i])
                  *(h[i]*cos(L[i])+k[i]*sin(L[i])+f[i]*h[i]+g[i]*k[i]);
var rnorm{i in 0..n} = sqrt(r1[i]^2+r2[i]^2+r3[i]^2);
var rcrossv1{i in 0..n} = r2[i]*v3[i]-v2[i]*r3[i];
var rcrossv2{i in 0..n} = r3[i]*v1[i]-v3[i]*r1[i];
var rcrossv3{i in 0..n} = r1[i]*v2[i]-v1[i]*r2[i];
var rcrossvnorm{i in 0..n} = sqrt(rcrossv1[i]^2+rcrossv2[i]^2+rcrossv3[i]^2);
var rcrossvcrossr1{i in 0..n} = rcrossv2[i]*r3[i]-rcrossv3[i]*r2[i];
var rcrossvcrossr2{i in 0..n} = rcrossv3[i]*r1[i]-rcrossv1[i]*r3[i];
var rcrossvcrossr3{i in 0..n} = rcrossv1[i]*r2[i]-rcrossv2[i]*r1[i];
var a12{i in 0..n} = 2*p[i]/q[i]*sqrt(p[i]/mu);
var a21{i in 0..n} = sqrt(p[i]/mu)*sin(L[i]);
var a22{i in 0..n} = 1/q[i]*sqrt(p[i]/mu)*((1+q[i])*cos(L[i])+f[i]);
var a23{i in 0..n} = -g[i]/q[i]*sqrt(p[i]/mu)*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var a31{i in 0..n} = -sqrt(p[i]/mu)*cos(L[i]);
var a32{i in 0..n} = 1/q[i]*sqrt(p[i]/mu)*((1+q[i])*sin(L[i])+g[i]);
var a33{i in 0..n} = f[i]/q[i]*sqrt(p[i]/mu)*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var a43{i in 0..n} = sqrt(p[i]/mu)*s2[i]/q[i]/2*cos(L[i]);
var a53{i in 0..n} = sqrt(p[i]/mu)*s2[i]/q[i]/2*sin(L[i]);
var a63{i in 0..n} = sqrt(p[i]/mu)/q[i]*(h[i]*sin(L[i])-k[i]*cos(L[i]));
var sphi{i in 0..n} = 2*(h[i]*sin(L[i])-k[i]*cos(L[i]))/s2[i];
var P2{i in 0..n} = (3*sphi[i]^2-1)/2;
var P3{i in 0..n} = (5*sphi[i]^3-3*sphi[i])/2;
var P4{i in 0..n} = (35*sphi[i]^4-30*sphi[i]^2+3)/8;
var dP2{i in 0..n} = 3*sphi[i];
var dP3{i in 0..n} = (15*sphi[i]^2-3)/2;
var dP4{i in 0..n} = (70*sphi[i]^3-30*sphi[i])/4;
var dgn{i in 0..n} = -mu*sqrt(1-sphi[i]^2)/r[i]^2*
                   ((Re/r[i])^2*dP2[i]*J2+(Re/r[i])^3*dP3[i]*J3+(Re/r[i])^4*dP4[i]*J4);
var dgr{i in 0..n} = -mu/r[i]^2*
                   (3*(Re/r[i])^2*P2[i]*J2+4*(Re/r[i])^3*P3[i]*J3+5*(Re/r[i])^4*P4[i]*J4);
var ir1{i in 0..n} = r1[i]/rnorm[i];
var ir2{i in 0..n} = r2[i]/rnorm[i];
```

```
var ir3{i in 0..n} = r3[i]/rnorm[i];
var it1{i in 0..n} = rcrossvcrossr1[i]/(rcrossvnorm[i]*rnorm[i]);
var it2{i in 0..n} = rcrossvcrossr2[i]/(rcrossvnorm[i]*rnorm[i]);
var it3{i in 0..n} = rcrossvcrossr3[i]/(rcrossvnorm[i]*rnorm[i]);
var ih1{i in 0..n} = rcrossv1[i]/rcrossvnorm[i];
var ih2{i in 0..n} = rcrossv2[i]/rcrossvnorm[i];
var ih3{i in 0..n} = rcrossv3[i]/rcrossvnorm[i];
var hin1{i in 0..n} = -ir3[i]*ir1[i];
var hin2{i in 0..n} = -ir3[i]*ir2[i];
var hin3{i in 0..n} = 1-ir3[i]*ir3[i];
var innorm{i in 0..n} = sqrt(hin1[i]^2+hin2[i]^2+hin3[i]^2);
var in1{i in 0..n} = hin1[i]/innorm[i];
var in2{i in 0..n} = hin2[i]/innorm[i];
var in3{i in 0..n} = hin3[i]/innorm[i];
var dg1{i in 0..n} = dgn[i]*in1[i]-dgr[i]*ir1[i];
var dg2{i in 0..n} = dgn[i]*in2[i]-dgr[i]*ir2[i];
var dg3{i in 0..n} = dgn[i]*in3[i]-dgr[i]*ir3[i];
var g1{i in 0..n} = ir1[i]*dg1[i]+ir2[i]*dg2[i]+ir3[i]*dg3[i];
var g2{i in 0..n} = it1[i]*dg1[i]+it2[i]*dg2[i]+it3[i]*dg3[i];
var g3{i in 0..n} = ih1[i]*dg1[i]+ih2[i]*dg2[i]+ih3[i]*dg3[i];
var T1{i in 0..n} = G0*T*(1+tau/100)/w[i]*ur[i];
var T2{i in 0..n} = G0*T*(1+tau/100)/w[i]*ut[i];
var T3{i in 0..n} = G0*T*(1+tau/100)/w[i]*uh[i];
var Delta1{i in 0..n} = g1[i]+T1[i];
var Delta2{i in 0..n} = g2[i]+T2[i];
var Delta3{i in 0..n} = g3[i]+T3[i];



let n := 100;



maximize weight: 10000*w[n];

# differential equations (trapezoidal rule of discretization)

de1 {i in 0..n-1}: p[i+1] = p[i]+step*(a12[i]*Delta2[i]+a12[i+1]*Delta2[i+1])/2;
de2 {i in 0..n-1}: f[i+1] = f[i]+step*(a21[i]*Delta1[i]+a22[i]*Delta2[i]+a23[i]*Delta3[i]
                   +a21[i+1]*Delta1[i+1]+a22[i+1]*Delta2[i+1]+a23[i+1]*Delta3[i+1])/2;
de3 {i in 0..n-1}: g[i+1] = g[i]+step*(a31[i]*Delta1[i]+a32[i]*Delta2[i]+a33[i]*Delta3[i]
                   +a31[i+1]*Delta1[i+1]+a32[i+1]*Delta2[i+1]+a33[i+1]*Delta3[i+1])/2;
de4 {i in 0..n-1}: h[i+1] = h[i]+step*(a43[i]*Delta3[i]+a43[i+1]*Delta3[i+1])/2;
de5 {i in 0..n-1}: k[i+1] = k[i]+step*(a53[i]*Delta3[i]+a53[i+1]*Delta3[i+1])/2;
de6 {i in 0..n-1}: L[i+1] = L[i]+step*(a63[i]*Delta3[i]+a63[i+1]*Delta3[i+1]
                          + sqrt(mu*p[i])*(q[i]/p[i])^2
                    + sqrt(mu*p[i+1])*(q[i+1]/p[i+1])^2)/2;
de7 {i in 0..n-1}: w[i+1] = w[i]-step*T/Isp*(1+tau/100);

# control constraints

cc {i in 0..n}: sqrt(ur[i]^2+ut[i]^2+uh[i]^2) = 1;
tc: tauL <= tau <= 0.0;

u1 {i in 0..n}: -1 <= ur[i] <= 1;
u2 {i in 0..n}: -1 <= ut[i] <= 1;
u3 {i in 0..n}: -1 <= uh[i] <= 1;
```

```
# state constraints


# boundary values

ic1: p[0] = p0;
ic2: f[0] = f0;
ic3: g[0] = g0;
ic4: h[0] = h0;
ic5: k[0] = k0;
ic6: L[0] = L0;
ic7: w[0] = w0;
if1: p[n] = pf;
if2: sqrt(f[n]^2+g[n]^2) = fgf;
if3: sqrt(h[n]^2+k[n]^2) = hkf;
if4: f[n]*h[n]+g[n]*k[n] = 0.0;
if5: g[n]*h[n]-f[n]*k[n] <= 0.0;

# initial estimates

let {i in 0..n} p[i] := p0+(pf-p0)*i/n;
let {i in 0..n} f[i] := -0.12*i/n;
let {i in 0..n} g[i] := 0.7*(i/n)^4;
let {i in 0..n} h[i] := -0.6*(i/n)^3;
let {i in 0..n} k[i] := 0.02*i/n;
let {i in 0..n} L[i] := (16*pi)*i/n;
let {i in 0..n} w[i] := w0+(wf-w0)*i/n;
let {i in 0..n} ur[i] := 0.1;
let {i in 0..n} ut[i] := 0.7-i/n;
let {i in 0..n} uh[i] := -0.1;
let tau := -9.0;
let tf := 90000.0;


option solver ipopt;

solve;

display _total_solve_time, tf, tau,
if2, if2.slack, if3, if3.slack, if4, if4.slack, if5, if5.slack;

printf {i in 0..n}:
"%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f \n",
tf*i/n, p[i], f[i], g[i], h[i], k[i], L[i], w[i], ur[i], ut[i], uh[i]
> orbit_100_ipopt.out;
```