7 Numerical Algorithms

All Matlab-files that are needed are available to download from the course pages, but some lines of code are missing and the task is to complement the files to be able to solve the given problems. All places where you have to write some code are indicated by three question marks (???).

A basic second-order system will be used in the exercises below and the model and the problem are presented here. Consider a 1D motion model of a particle with position $z(t)$ and speed $v(t)$. Define the state vector $x = (z \;\; v)^T$ and the continuous time model

$$\dot{x}(t) = f(x(t), u(t)) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t). \tag{7.1}$$

The problem is to go from the state $x_i = x(0) = (1 \;\; 1)^T$ to $x(t_f) = (0 \;\; 0)^T$, where $t_f = 2$, but such that the control input energy $\int_0^{t_f} u^2(t)dt$ is minimized. Thus, the optimization problem is

$$\begin{aligned} \min_u \;\; & J = \int_0^{t_f} f_0(x, u)dt \\ s.t. \;\; & \dot{x}(t) = f(x(t), u(t)) \\ & x(0) = x_i \\ & x(t_f) = x_f \end{aligned} \tag{7.2}$$

where $f_0(t) = u^2(t)$ and $f(\,\cdot\,)$ is given in (7.1).

7.1 *Discretization Method 1.* In this approach we will use the discrete time model

$$x[k+1] = \bar{f}(x[k], u[k]) = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix} x[k] + \begin{pmatrix} 0 \\ h \end{pmatrix} u[k] \tag{7.3}$$

to represent the system where $t = kh$, $h = t_f/N$ and $x[k] = x(kh)$. The discretized version of the problem (7.2) is then

$$\begin{aligned} \min_{u[n], n=0,\ldots,N-1} \;\; & h \sum_{k=0}^{N-1} u^2[k] \\ s.t. \;\; & x[k+1] = \bar{f}(x[k], u[k]) \\ & x[0] = x_i \\ & x[N] = x_f. \end{aligned} \tag{7.4}$$

Define the optimization parameter vector as

$$y = \begin{pmatrix} x[0]^T \; u[0] \;\; x[1]^T \; u[1] \;\; \ldots \;\; x[N-1]^T \; u[N-1] \;\; x[N]^T \; u[N] \end{pmatrix}^T. \tag{7.5}$$

Note that $u[N]$ is superfluous, but is included to make the presentation and the code more convenient. Furthermore, define

$$\mathcal{F}(y) = h \sum_{k=0}^{N-1} u^2[k] \tag{7.6}$$

and

$$\mathcal{G}(y) = \begin{pmatrix} g_1(y) \\ g_2(y) \\ g_3(y) \\ g_4(y) \\ \vdots \\ g_{N+2}(y) \end{pmatrix} = \begin{pmatrix} x[0] - x_i \\ x[N] - x_f \\ x[1] - \bar{f}(x[0], u[0]) \\ x[2] - \bar{f}(x[1], u[1]) \\ \vdots \\ x[N] - \bar{f}(x[N-1], u[N-1]) \end{pmatrix} \tag{7.7}$$

Then the optimization problem in (7.4) can be expressed as the constrained problem

$$\min_y \;\; \mathcal{F}(y) \tag{7.8}$$

$$s.t. \;\; \mathcal{G}(y) = 0. \tag{7.9}$$

(a) Show that the discrete time model in (7.3) can be obtained from (7.1) by using the Euler-approximation

$$x[k+1] = x[k] + hf(x[k], u[k]). \tag{7.10}$$

and complement the file `secOrderSysEqDisc.m` with this model.

(b) Complement the file `secOrderSysCostCon.m` with a discrete time version of the cost function.

(c) Note that the constraints are linear for this system model and this is very important to exploit in the optimization routine. In fact the problem is a quadratic programming problem, and since it only contain equality constraints it can be solved as a linear system of equations.

Complete the file `mainDiscLinconSecOrderSys.m` by creating the linear constraint matrix A_{eq} and the vector B_{eq} such that the constraints defined by $\mathcal{G}(y) = 0$ is expressed as $A_{eq}y = B_{eq}$. Then run the Matlab script.

(d) Now suppose that the constraints are nonlinear. This case can be handled by passing a function that computes $\mathcal{G}(y)$ and its Jacobian. Let the initial and terminal state constraints, $g_1(y) = 0$ and $g_2(y) = 0$, be handled as above and complete the function `secOrderSysNonlcon.m` by computing

$$c_{eq} = (g_3(y) \;\; g_4(y) \;\; \dots \;\; g_{N+2}(y)) \tag{7.11}$$

and the Jacobian

$$[\nabla_y c_{eq}]^T = \texttt{ceqJac} = \begin{pmatrix} \frac{\partial g_3}{\partial x[0]} & \frac{\partial g_3}{\partial u[0]} & \frac{\partial g_3}{\partial x[1]} & \frac{\partial g_3}{\partial u[1]} & \dots \\ \frac{\partial g_4}{\partial x[0]} & \frac{\partial g_4}{\partial u[0]} & \frac{\partial g_4}{\partial x[1]} & \frac{\partial g_4}{\partial u[1]} & \dots \\ \vdots & & & & \\ \frac{\partial g_{N+2}}{\partial x[0]} & \frac{\partial g_{N+2}}{\partial u[0]} & \frac{\partial g_{N+2}}{\partial x[1]} & \frac{\partial g_{N+2}}{\partial u[1]} & \dots \end{pmatrix}^T \tag{7.12}$$

Each row in (7.12) is computed in each loop in the for-statement in `secOrderSysNonlcon.m`. Note that the length of the state vector is 2. Also note that Matlab wants the Jacobian to be transposed, but that operation is already added last in the file. Finally, run the Matlab script `mainDiscNonlinconSecOrderSys.m` to solve the problem again.

7.2 *Discretization Method 2.* In this exercise we will also use the discrete time model in (7.3). However, the terminal constraints are removed by including it in the objective function. If the terminal constraints are not fulfilled they will penalize the objective function. Now the optimization problem can be defined as

$$\begin{aligned} \min_{u[n], n=0,\dots,N-1} \;\; & c||x[N] - x_f||^2 + h\sum_{k=0}^{N-1} u^2[k] \\ s.t. \;\; & x[k+1] = \bar{f}(x[k], u[k]) \\ & x[0] = x_i \end{aligned} \tag{7.13}$$

where $\bar{f}(\,\cdot\,)$ is given in (7.3), $N = t_f/h$ and c is a predefined constant. This problem can be rewritten as an unconstrained nonlinear problem

$$\min_{u[n], n=0,\dots,N-1} \;\; J = \mathcal{F}(x_i, x_f, h, u[0], u[1], ..., u[N-1]) \tag{7.14}$$

where $w(\,\cdot\,)$ contains the system model recursion (7.3). The optimization problem will be solved by using the Matlab function `fminunc`.

(a) Write down an explicit algorithm for the computation of $\mathcal{F}(\,\cdot\,)$.

(b) Complement the file `secOrderSysCostUnc.m` with the cost function $\mathcal{F}(\,\cdot\,)$. Solve the problem by running the script `mainDiscUncSecOrderSys.m`.

(c) Compare method 1 and 2. What are the advantages and disadvantages? Which method handles constraints best? Suppose the task was also to implement the optimization algorithm itself, which method would be easiest to implement an algorithm for (assuming that the state dynamics is non-linear)?

(d) (Optional) Implement an unconstrained gradient method that can replace the `fminunc` function in `mainDiscUncSecOrderSys.m`.

7.3 *Gradient Method.* As above the terminal constraint is added as a penalty in the objective function in order to introduce the PMP gradient method here in a more straightforward way. Thus, the optimization problem is

$$\min_u \;\; J = c||x(t_f) - x_f||^2 + \int_0^{t_f} u^2(t)dt \tag{7.15}$$

$$s.t. \;\; \dot{x}(t) = f(x(t), u(t)) \tag{7.16}$$

$$x(0) = x_i. \tag{7.17}$$

(a) Write down the Hamiltonian and show that the Hamiltonian partial derivatives w.r.t. x and u are

$$\begin{aligned} H_x &= (0 \;\; \lambda_1) \\ H_u &= 2u(t) + \lambda_2, \end{aligned} \tag{7.18}$$

respectively.

(b) What is the adjoint equation and its terminal constraint?

(c) Complement the files `secOrderSysEq.m` with the system model, the file `secOrderSysAdjointEq.m` with the adjoint equations, the file `secOrderSysFinalLambda.m` with the terminal values of λ, and the file `secOrderSysGradient.m` with the control signal gradient. Finally, complement the script `mainGradientSecOrderSys.m` and solve the problem.

(d) Try some different values of the penalty constant c. What happens if c is "small"? What happens if c is "large"?

7.4 *Shooting Method.* Shooting methods are based on successive improvements of the unspecified terminal conditions of the two point boundary value problem. In this case we can use the original problem formulation in (7.2).

Complement the files `secOrderSysEqAndAdjointEq.m` with the combined system and adjoint equation, and the file `theta.m` with the final constraints. The main script is `mainShootingSecOrderSys.m`.

7.5 *Reflections.*

(a) What are the advantages and disadvantages of discretization methods?

(b) Discuss the advantages and disadvantages of the problem formulation in 7.1 compared to the formulation in 7.2.

(c) What are the advantages and disadvantages of gradient methods?

(d) Compare the methods in 7.1, 7.2 and 7.3 in terms of accuracy and complexity/speed. Also compare the results for different algorithms used by `fmincon` in Exercises 7.1. Can all algorithms handle the optimization problem?

(e) Assume that there are constraints on the control signal and you can use either the discretization method in 7.1 or the gradient method in 7.3. Which methods would you use?

(f) What are the advantages and disadvantages of shooting methods?

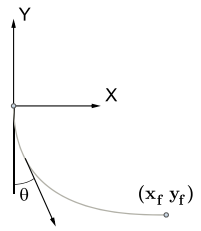

Figure 8.3a. The brachistochrone problem.

involves several of the greatest scientists ever, like Galileo, Pascal, Fermat, Newton, Lagrange, and Euler.

The Brachistochrone problem; Example 1. Let the motion of the particle, under the influence of gravity g, be defined by a time continuous state space model $\dot{X} = f(X, \theta)$ where the state vector is defined as $X = (x \quad y \quad v)^T$ and $(x \quad y)^T$ is the Cartesian position of the particle in a vertical plane and v is the speed, i.e.,

$$\begin{aligned} \dot{x} &= v\sin(\theta) \\ \dot{y} &= -v\cos(\theta) \end{aligned} \tag{8.2}$$

The motion of the particle is constrained by a path that is defined by the angle $\theta(t)$, see Figure 8.3a.

(a) Give an explicit expression of $f(X, \theta)$ (only the expression for $\dot{v}$ is missing).

(b) Define the Brachistochrone problem as an optimal control problem based on this state space model. Assume that the initial position of the particle is in the origin and the initial speed is zero. The final position of the particle is $(x(t_f) \quad y(t_f))^T = (x_f \quad y_f)^T = (10 \quad -3)^T$.

(c) Modify the script `minCurveLengthHeadingCtrl.m` of the minimum curve length example 8.2 above and solve this optimal control problem with PROPT.

8.4 *The Brachistochrone problem; Example 2.* The time for a particle to travel on a curve between the points $p_0 = (0, 0)$ and $p_f = (x_f, y_f)$ is

$$t_f = \int_{p_0}^{p_f} \frac{1}{v} ds \tag{8.3}$$

where ds is an element of arc length and v is the speed.

(a) Show that the travel time t_f is

$$t_f = \int_0^{x_f} \frac{\sqrt{1 + y'(x)^2}}{\sqrt{-2gy(x)}} dx \tag{8.4}$$

in the Brachistochrone problem where the speed of the particle is due to gravity and its initial speed is zero.

(b) Define the Brachistochrone problem as an optimal control problem based on the expression in (8.4). Note that the time variable is eliminated and that y is a function x now.

(c) Use the PMP to show (or derive) that the optimal trajectory is a cycloid

$$y'(x) = \sqrt{\frac{C - y(x)}{y(x)}}. \tag{8.5}$$

with the solution

$$\begin{aligned} x &= \frac{C}{2}(\phi - \sin(\phi)) \\ y &= \frac{C}{2}(1 - \cos(\phi)). \end{aligned} \tag{8.6}$$

(d) Try to solve the this optimal control problem with PROPT/Matlab, by modifying the script `minCurveLength.m` from the minimum curve length example. Compare the result with the solution in Examples 1 above. Try to explain why we have problems here.

8.5 *The Brachistochrone problem; Example 3.* To avoid the problem when solving the previous problem in PROPT/Matlab we can use the results in the exercise 8.4c). The cycloid equation (8.5) can be rewritten as

$$y(x)(y'(x)^2 + 1) = C. \tag{8.7}$$

7 Numerical Algorithms

7.1 (a)

$$\begin{aligned} x[k+1] &= x[k] + hf(x[k], u[k]) \\ &= x[k] + h\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x[k] + h\begin{pmatrix} 0 \\ 1 \end{pmatrix} u[k] \\ &= \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix} x[k] + \begin{pmatrix} 0 \\ h \end{pmatrix} u[k] \end{aligned} \tag{7.1}$$

secOrderSysEqDisc.m

```
function xk1 = secOrderSysEqDisc(xk, uk, h)
% secOrderSysEqDisc - Discrete-time dynamic equation
%
% xk  - State at time index k
% uk  - Input at time index k
% h - Sampeling time
% xk1 - State at time index k+1

xk1 = zeros(2,1);
%#x(1) = ???;
%#x(2) = ???;
xk1(1) = xk(1) + h*xk(2);
xk1(2) = xk(2) + h*uk;
```

(b)

secOrderSysCostCon.m

```
function J = secOrderSysCostCon(y, h)
% secOrderSysCostCon - Cost function J = F(y)
% y - Optimization vector: y = [x[0]^T u[0],...,x[N]^T,u[N]]
% h - Sampeling time
% J = Value of cost function

%#F = ???;
J = h*sum(y(3:3:end).^2);
```

(c)

$$A_{eq} = \begin{pmatrix} E & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & E \\ -F & E & 0 & 0 & \dots & 0 & 0 \\ 0 & -F & E & 0 & \dots & 0 & 0 \\ 0 & 0 & -F & E & \dots & 0 & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & \dots & -F & E \end{pmatrix} \tag{7.2}$$

$$B_{eq} = (x_i \;\; x_f \;\; 0 \;\; 0 \;\; \dots \;\; 0)^T. \tag{7.3}$$

where

$$F = \begin{pmatrix} 1 & h & 0 \\ 0 & 1 & h \end{pmatrix} \tag{7.4}$$

and

$$E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{7.5}$$

mainDiscLinconSecOrderSys.m

```
% Second order optimal control problem solved by using fmincon on
% a discretizised model.
% The constraints due to the system model are handled as nonlinear
% constraints to illustrate the handling of a general system model.
N = 50; % number of sample points
tf = 2; % final time
h = tf/N; % sample time
%%% Constraints
xi = [1,1]'; % initial constraint
xf = [0,0]'; % final constraint
n = 3*(N+1);
%#E = [???];
%#F = [???];
E = [1, 0, 0; 0, 1, 0];
F = [1, h, 0; 0, 1, h];
Aeq = zeros(4 + 2*N, n); % linear constrints
%#Aeq(1:2,1:2) = ???;
%#Aeq(3:4,end-2:end-1) = ???;
Aeq(1:2,1:2) = eye(2);
Aeq(3:4,end-2:end-1) = eye(2);
```

```
for i = 1:N
  ii = 4 + 2*(i-1) + (1:2);
  jj = 3*(i-1) + (1:6);
  Aeq(ii,jj) = [-F, E];
end
%#Beq = [???];
Beq = [xi; xf; zeros(2*N,1)];
%%% Start guess
y0 = [xi; 0]*ones(1,N+1);
y0 = y0(:);
%%% Optimization options
options = optimset('fmincon');
options = optimset(options, 'Algorithm','interior-point');
options = optimset(options, 'GradConstr','on');
%%% Solve the problem
%#[X,fval,exitflag,output] = fmincon(@secOrderSysCostCon, y0, [], [], ???, ???, ...
%#  [], [], [], options, h);
[y,fval,exitflag,output] = fmincon(@secOrderSysCostCon, y0, [], [], Aeq, Beq, ...
  [], [], [], options, h);
%%% Show the result
fprintf('Final value of the objective function: %0.6f \n', fval)
tt = h*(0:N);
z = y(1:3:end)';
v = y(2:3:end)';
u = y(3:3:end)';
figure
subplot(2,1,1), plot( tt, z, 'b.-', tt, v, 'g+-'), xlabel('t'), legend('z','v')
title('Second Order System state variables');
subplot(2,1,2), plot( tt(1:end-1), u(1:end-1), 'b+-'), xlabel('t'), ylabel('u')
title('Second Order System control');
```

(d)

mainDiscNonlinconSecOrderSys.m

```
% Second order optimal control problem solved by using fmincon on
% a discretizised model.
% The constraints due to the system model are handled as nonlinear
% constriants to illustrate the handling of a general system model.
N = 50; % number of sample points
tf = 2; % final time
h = tf/N; % sample time
%%% Constraints
xi = [1,1]'; % initial constraint
xf = [0,0]'; % final constraint
n = 3*(N+1);
Aeq = zeros(4, n);
```

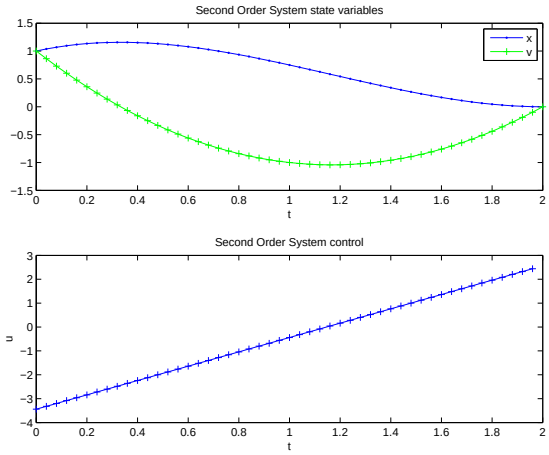

Figure 7.1a. Result of exercise 7.1 (c); discrete time method with linear constraints.

```
Aeq(1:2,1:2) = eye(2);
Aeq(3:4,end-2:end-1) = eye(2);
Beq = [xi; xf];
%%% Start guess
y0 = [xi; 0]*ones(1,N+1);
y0 = y0(:);
%%% Optimization options
options = optimset('fmincon');
options = optimset(options, 'Algorithm','interior-point');
options = optimset(options, 'GradConstr','on');
%%% Solve the problem
[y,fval,exitflag,output] = fmincon(@secOrderSysCostCon, y0, [], [], Aeq, Beq, ...
  [], [], @secOrderSysNonlcon, options, h);
%%% Show the result
fprintf('Final value of the objective function: %0.6f \n', fval)
tt = h*(0:N);
z = y(1:3:end)';
v = y(2:3:end)';
u = y(3:3:end)';
```

```
figure
subplot(2,1,1), plot( tt, z, 'b.-', tt, v, 'g+-'), xlabel('t'), legend('z','v')
title('Second Order System state variables');
subplot(2,1,2), plot( tt(1:end-1), u(1:end-1), 'b+-'), xlabel('t'), ylabel('u')
title('Second Order System control');
```

secOrderSysNonlcon.m

```
function [c,ceq,cJac, ceqJac] = secOrderSysNonlcon(y, h)
% [c,ceq,cJac, ceqJac] = secOrderSysCostCon(y, h) - Cost function
% y - Optimization vector: y = [x[0]^T u[0],...,x[N]^T,u[N]]
% h - Sampeling time
% c - Nonlinear inequality constrint c(y) <= 0
% ceq - Nonlinear equality constrint ceq(y) <= 0
% cJac - Jacobian of nonlinear inequality constrint
% ceqJac - Jacobian of nonlinear equality constrint

N = length(y)/3-1; % Compute N
c = []; cJac =[]; % No inequality constrints
ceq = zeros(N*2,1); % (N+1)*2 equality constrints
ceqJac = zeros(N*2, 3*N);
for k = 0:(N-1)
  xk = y((1:2) + k*3); %Extract x[k]
  uk = y(3 + k*3); %Extract u[k]
  xk1 = y((1:2) + (k+1)*3); %Extract x[k+1]
  xk1b = secOrderSysEqDisc(xk, uk, h); %Compute f(x[k],u[k])
  %#ceq((1:2) +k*2) = ???; %Compute ceq_k = x[k+1] - f(x[k],u[k])
  ceq((1:2) + k*2) = xk1 - xk1b; %Compute ceq_k = x[k+1] - f(x[k],u[k])

  % Compute [d(ceq_k)/dx_[k], d(ceq_k)/du_[k], d(ceq_k)/dx_[k+1],
  % d(ceq_k)/du_[k+1]]
  %#ceqJac(1+k*2, 1+k*3) = ???;
  %#ceqJac(1+k*2, 2+k*3) = ???;
  %#ceqJac(1+k*2, 3+k*3) = ???;
  %#ceqJac(1+k*2, 4+k*3) = ???;
  %#ceqJac(1+k*2, 5+k*3) = ???;
  %#ceqJac(1+k*2, 6+k*3) = ???;
  ceqJac(1+k*2, 1+k*3) = -1;
  ceqJac(1+k*2, 2+k*3) = -h;
  ceqJac(1+k*2, 3+k*3) = 0;
  ceqJac(1+k*2, 4+k*3) = 1;
  ceqJac(1+k*2, 5+k*3) = 0;
  ceqJac(1+k*2, 6+k*3) = 0;

  %#ceqJac(2+k*2, 1+k*3) = ???;
  %#ceqJac(2+k*2, 2+k*3) = ???;
  %#ceqJac(2+k*2, 3+k*3) = ???;
  %#ceqJac(2+k*2, 4+k*3) = ???;
  %#ceqJac(2+k*2, 5+k*3) = ???;
  %#ceqJac(2+k*2, 6+k*3) = ???;
  ceqJac(2+k*2, 1+k*3) = 0;
  ceqJac(2+k*2, 2+k*3) = -1;
  ceqJac(2+k*2, 3+k*3) = -h;
  ceqJac(2+k*2, 4+k*3) = 0;
  ceqJac(2+k*2, 5+k*3) = 1;
  ceqJac(2+k*2, 6+k*3) = 0;
end
ceq = ceq';
ceqJac = ceqJac';
```

7.2 (a) (i) $x := x_i$

(ii) for $k = 0$ to $N - 1$ do $x := \bar{f}(x, u[k])$

(iii) $w := c||x - x_f||^2 + h\sum_{k=0}^{N-1} u^2[k]$

(b)

mainDiscUncSecOrderSys.m

```
% Second order optimal control problem solved by using fmincon on
% a discretizised model.
% The final state constraint is added to the cost function as a penalty.
N = 50; % number of sample points
tf = 2; % final time
h = tf/N; % sample time
c = 1e3; % cost on the terminal constraint
%%% Constraints
xi = [1,1]'; % initial constraint
xf = [0,0]'; % final constraint
n = 3*(N+1);
%%% Start guess
u0 = 0*ones(1,N);
%%% Solve the problem
options = optimset('fminunc');
options = optimset(options, 'LargeScale','off');
[u,fval,exitflag,output] = fminunc(@secOrderSysCostUnc,u0,options,h,c,xi,xf);
%%% Show the result
fprintf('Final value of the objective function: %0.6f \n', fval)
X = zeros(2,N+1); X(:,1) = xi;
```

```
for k = 1:N, X(:,k+1) = secOrderSysEqDisc(X(:,k), u(k), h); end
tt = h*(0:N);
z = X(1,:);
v = X(2,:);
figure
subplot(2,1,1), plot( tt, z, 'b.-', tt, v, 'g+-'), xlabel('t'), legend('z','v')
title('Second Order System state variables');
subplot(2,1,2), plot( tt(1:end-1), u, 'b+-'), xlabel('t'), ylabel('u')
title('Second Order System control');
```

secOrderSysCostUnc.m

```
function J = secOrderSysCostUnc(u, h, c, xi, xf)
% secOrderSysCostUnc - Cost function J = F(u)  - unconstrined version
%
% u - Optimization vector with input values: u = [u[0],...,u[N-1]]
% h - Sampeling time
% c - Penalty constant
% xi - Initial constrint
% xf - Final constrint
% J = Value of cost function
N = length(u);
x = xi;
for k = 1:N % Compute x[k] recursively
  x = secOrderSysEqDisc(x, u(k), h);
end
%#J = ???;
J = h*sum(u.^2) + c * norm(x-xf)^2;
```

(c) Method 1 can handle path constraints better. Unconstrained optimization algorithms are in general much easier to implement than algorithms for constrained problems.

7.3 (a) Hamiltonian:

$$H = f_0(x,u) + \lambda^T f(x,u) = u(t)^2 + \lambda_1 v(t) + \lambda_2 u(t) \tag{7.6}$$

$$\begin{aligned} H_x &= (0 \quad \lambda_1) \\ H_u &= 2u(t) + \lambda_2 \end{aligned} \tag{7.7}$$

(b) Adjoint equation:

$$\dot{\lambda}(t) = -H_x(x(t), u(t), \lambda(t)) \tag{7.8}$$

Thus

$$\begin{aligned} \dot{\lambda}_1(t) &= 0 \\ \dot{\lambda}_2(t) &= -\lambda_1 \end{aligned} \tag{7.9}$$

where

$$\lambda(t_f) = \phi_x(x(t_f)) = 2c(x(t_f) - x_{t_f}) \tag{7.10}$$

(c)

secOrderSysEq.m

```
function dx = secOrderSysEq(t, x, tu, u)
% secOrderSysEq - Continuous-time dynamic equation
%
% t  - Time
% x  - State at time t
% tu - Discrete-time time vector
% u  - Discrete-time input vecor
% dx - Time derivative of state at time t

% Compute the input at time t by interpolating the discrete-time inout
% vector
u = interp1(tu, u, t);

%#dx = [???; ???];
dx = [x(2); u];
```

secOrderSysAdjointEq.m

```
function dlambda = secOrderSysAdjointEq(t, lambda, tu, u, tx, x)
% secOrderSysAdjointEq - Continuous-time adjoint equation
%
% t         - Time
% lambda    - Adjoint variable at time t
% tu        - Discrete-time time vector for input vector
% u         - Discrete-time input vecor
% tx        - Discrete-time time vector for state vector
% x         - Discrete-time state vecor
% dx - Time derivative of state at time t
```

```
% Compute the input and state at time t by interpolating the discrete-time
% input and state vectors
u = interp1(tu,u,t);
x = interp1(tx,x,t);

%#dlambda = [???; ???];
dlambda = [0; -lambda(1)];
```

secOrderSysFinalLambda.m

```
function Laf = secOrderSysFinalLambda(x, xf, c)
% secOrderSysFinalLambda - Final constraint on lambda
% x - State
% xf - Final constrint on x(tf)
% Laf - Final constrint on lambda(tf)

%#Laf = ???
Laf = 2*c*(x(end,:)'-xf);
```

secOrderSysGradient.m

```
function Hu = secOrderSysGradient(tLa, La, tu, u, tx, x)
% secOrderSysGradient - Computes the gradient of the Hamiltonian with
% respect to u
%
% tLa - Discrete-time time vector for input vector
% La  - Discrete-time input vecor
% tu  - Discrete-time time vector for input vector
% u   - Discrete-time input vecor
% tx  - Discrete-time time vector for state vector
% x   - Discrete-time state vecor
% Hu  - Derivative of Hamiltonian with respect to u

% Compute the input and state at time t by interpolating the discrete-time
% input and state vectors

% States at time tu
x1 = interp1(tx, x(:,1), tu);
x2 = interp1(tx, x(:,2), tu);
% Adjoint variables at time tu
la1 = interp1(tLa, La(:,1), tu);
la2 = interp1(tLa, La(:,2), tu);

%#Hu = ???
Hu = 2*u + la2;
```

mainGradientSecOrderSys.m

```
% Second order optimal control problem solved with a gradient method.
% The final state constraint is added to the cost function as a penalty.
N = 2000; % number of sample points
tf = 2; % final time
h = tf/N; % sample time
c = 200; % cost on the terminal constraint
xi = [1,1]'; % initial constraint
xf = [0,0]'; % final constraint
%%% Optimization options
maxIter = 200; % maximum number of iterations
alpha = 0.001; % Step size in the gradient update step
tols = 1e-3;
%%% Start guess
tu = 0:h:tf;
u = linspace(-1, 1, length(tu));
%%% Solve the problem
iter = 1; dua = 1;
while dua > tols
  %%% Simulate system forward
  %#[tx,x] = ode23(@secOrderSysEq,???,xi,[],tu,u);
  [tx,x] = ode23(@secOrderSysEq,[0,tf],xi,[],tu,u);
  %%% Compute final constraint for lambda
  Laf = secOrderSysFinalLambda(x, xf, c);
  %%% Simulate adjoint system backwards
  %#[tLa,La] = ode23(@secOrderSysAdjointEq,???,Laf,[],tx,x,tu,u);
  [tLa,La] = ode23(@secOrderSysAdjointEq,[tf,0],Laf,[],tx,x,tu,u);
  %%% Compute gradient and update the control signal
  Hu = secOrderSysGradient(tLa, La, tu, u, tx, x);
  du = alpha*Hu;
  %#u = ???; % update the control signal
  u = u - du;
  %%% Abort the optimization?
  if iter > maxIter, break; end
  iter = iter + 1;
  dua = norm(du)/sqrt(length(du));
end
%%% Show the result
z = x(:,1)';
v = x(:,2)';
```

```
fval = c*norm(x(end,:)'-xf)^2 + h*sum(u.^2);
fprintf('Final value of the objective function: %0.6f \n', fval)
figure
subplot(2,1,1), plot( tx, z, 'b+-', tx, v, 'g+-'), xlabel('t')
title('Second Order System state variables');
subplot(2,1,2), plot( tu, u, 'b-'), xlabel('t'), ylabel('u')
title('Second Order System control');
```

(d) Small c will cause that the final state is far from the desired final state. For large c the solution will not converge.

7.4

mainShootingSecOrderSys.m

```
% Second order optimal control problem solved with a shooting method.
% The final state constraint is added to the cost function as a penalty.

tf = 2; % final time

xi = [1,1]'; % initial constraint
xf = [0,0]'; % final constraint

%%% Optimization options
optim = optimset('fsolve');
optim.Display = 'off';
optim.Algorithm = 'levenberg-marquardt';

%%% Start guess
lambda0 = [0;0];

%%% Solve the problem
lambda0 = fsolve(@theta, lambda0, optim, xi, xf, tf); % solve mu(lambda0)=0

%%% Check the terminal constraints are satisfied
maxConditionError=1e-3;
if norm( theta(lambda0, xi, xf, tf) ) > maxConditionError
  disp('Warning! Terminal constraints not satisfied!');
end

%%% Show the result
[ts,s] = ode23(@secOrderSysEqAndAdjointEq, [0 tf], [xi;lambda0]);
x1 = s(:,1)';
x2 = s(:,2)';
lambda1 = s(:,3)';
lambda2 = s(:,4)';
u = -lambda2/2;
fval = 0.5*sum(diff(ts)'.*u(1:end-1).^2) + ...
  0.5*sum(diff(ts)'.*u(2:end).^2); % sloppy approximation!!!
fprintf('Final value of the objective function: %0.6f \n', fval)
figure
subplot(2,1,1), plot( ts, x1, 'b+-', ts, x2, 'g+-'), xlabel('t')
title('Second Order System state variables');
subplot(2,1,2), plot( ts, u, 'b-'), xlabel('t'), ylabel('u')
title('Second Order System control');
```

secOrderSysEqAndAdjointEq.m

```
function ds = secOrderSysEqAndAdjointEq(t, s)
% secOrderSysEqAndAdjointEq - Continuous-time joint system
%
% t  - Time
% s  - State and adjoint varible at time t

x = s(1:2);
lambda = s(3:4);
%#u = ???; % Choose u such as Hu = 2*u + la2 = 0
u = -lambda(2)/2;
ds = zeros(4,1);
%#ds(1) = ???;
ds(1) = x(2);
%#ds(2) = ???;
ds(2) = u;
%#ds(3) = ???;
ds(3) = 0;
%#ds(4) = ???;
ds(4) = -lambda(1);
```

theta.m

```
function mu = theta(lambda0, x0, xf, tf)
% theta - Computes deviation from the final constrints for the joint system, (10.4) in
% lambda0 - Adjoint variable at time t=0
% x0      - State at time t=0
% xf      - State at time t=tf
% tf      - Final time

% Integrate the system and adjoint equations forward in time
```