Problem Set #2: Numerical Optimal Control

Exercise 1: Formulation of OCPs

Consider a chemical continuous stirred tank reactor (CSTR) in which the exothermic irreversible reaction

$$A \to B$$

takes place. The dynamics of the CSTR are as follows, see [1] for details.

$$\dot{c}_A = \frac{q}{V}(c_{Af} - c_A) - k_0 e^{\frac{-E}{RT}} c_A \tag{1a}$$

$$\dot{T} = \frac{q}{V}(T_f - T) + \frac{-\triangle H}{\rho C_p} k_0 e^{\frac{-E}{RT}} c_A + \frac{UA}{V\rho C_p}(u_1 - T). \tag{1b}$$

The states c_A and T describe the concentration of subtance A and the reactor temperature in K. The coolant stream temperature u_1 is the considered input variable. The objective is to compute an input signal such that the system reaches the setpoint $c_{As} = 0.159[mol/l], T_s = 375[K]$. The coolant stream temperature u_1 is subject to the input constraint $u_1 \in [270, 330]$. The system parameters are listed in the following table.

Table 1: Parameters for CSTR.

q	100	$[L/min]$	c_{Af}	1	$[mol/L]$
T_f	350	$[K]$	V	100	$[L]$
ρ	1000	$[g/L]$	C_p	0.239	$[J/(g \cdot K)]$
$-\triangle H$	$5 \cdot 10^4$	$[J/mol]$	$\frac{E}{R}$	8750	$[K]$
k_0	$7.2 \cdot 10^{10}$	$[min^{-1}]$	UA	$5 \cdot 10^4$	$[J/(minK)]$

a) We want to calculate optimal open-loop inputs which steer the system (1) to the considered setpoint. Formulate three different optimal control problems, such that the solution of each of these problems yields the considered open-loop inputs.

Solution: In general, a fixed-end-time OCP for this problem could read as follows:

$$\min_{u(\cdot)} \quad \int_0^{t_f} l(c_A(t), T(t), u_1(t))dt + \phi(c_A(t_f), T(t_f)) \tag{2a}$$

subject to

$$\text{the dynamics (1) with } (c_A(0), T(0))^T = (c_{A0}, T_0)^T \tag{2b}$$

$$\text{the constraints } u(t) \in [270, 330] \tag{2c}$$

$$\psi_i(c_A(t_f), T(t_f)) = 0, i = 1, \ldots, n_\psi \tag{2d}$$

Possible choices for l, ϕ and ψ are:

- Get exactly to the optimal steady state with minimal input usage:

$$l(c_A(t), T(t), u_1(t)) = u_1(t)^2, \quad \phi(c_A(t_f), T(t_f)) = 0$$

$$\psi_j(c_A(t_f), T(t_f)) = j(t_f) - j_s, j \in \{c_A, T\}$$

- Minimize distance from desired steady state along the trajectory

$$l(c_A(t),T(t),u_1(t)) = \sum_j \omega_j (j(t) - j_s)^2, \quad \phi(c_A(t_f),T(t_f)) = 0$$

with $\omega_j > 0$ and $j \in \{c_A, T, u_1\}$, while $\psi_j = \phi = 0$. Note that this requires the input $u_{1,s}$ corresponding to the desired steady state.

- Get approximately to the desired steady state at final time while minimizing the input usage:

$$l(c_A(t),T(t),u_1(t)) = u_1(t)^2$$

$$\phi(c_A(t_f),T(t_f)) = \sum_j \omega_j (j(t_f) - j_s)^2$$

with $j \in \{c_A, T\}$ and $\omega_j \gg 0$. Note that for very large ω_j the quadratic Mayer term approximates the terminal constraint ψ from 1).

- Can you come up with further formulations?

Exercise 2: Direct Discretization

In this exercise we will solve an optimal control problem (OCP) via a (naive) direct simultaneous approach. We consider the following OCP:

$$\min_{u(\cdot)} \int_0^1 \mathbf{x}^T(t)\mathbf{x}(t) + 0.005u^2(t)dt \tag{3a}$$

subject to $\forall t \in [0,1]$:

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} 0 & 1 \\ 0 & c \end{pmatrix} \mathbf{x}(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t) \quad c \in \mathbb{R} \tag{3b}$$

$$u(t) \in [-20, 20] \tag{3c}$$

$$g(t,\mathbf{x}) = 8(t-0.5)^2 - 0.5 - x_2 \geq 0 \tag{3d}$$

$$\mathbf{x}(0) = (0,-1)^T. \tag{3e}$$

A simple procedure to integrate an ODE $\dot{\mathbf{x}}(t) = \mathbf{f}(t,\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0$ is given by the Euler forward discretization

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{f}(t,\mathbf{x}(t)), \tag{4}$$

where $h > 0$ is the integration step size.

a) Integrate the system (3b) for the given initial condition (3e) from $t = 0$ to $t = 1$. Consider $\forall t \in [0,1]$: $u(t) = 0, c = 1$ and $h = 0.02$. *Hint: Use* `Yalmip`*, represent the states and inputs as* `sdpvar` *variables and apply* (4)*.*

Solution:

```
t0 = 0;
tf = 1;
h  = 0.02;
c = 1;
x0 = [0; -1];
N = (tf - t0)/h +1;

% create sdpvars
x = sdpvar(2, N+1);
u = sdpvar(1, N+1);
t = sdpvar(1, N+1);

```

```
13
14 t(1)     = t0;
15 x(:,1)   = x0;
16
17
18 % loop over dynamics
19 for i=1:N
20     x(:,i+1) =  x(:, i) + h*([0, 1; 0, c]*x(:,i));
21     t(i+1)   = t(i) + h;
22 end
23
24 % convert sdpvar to double
25 X = double(x);
26 T = double(t);
27
28 % visualize
29 figure
30 plot(T, X(1,:), 'r', 'Linewidth', 2)
31 hold on
32 plot(T, X(2,:), 'b', 'Linewidth', 2)
33 legend('x_1', 'x_2')
34 xlabel('t')
35 ylabel('x_1, x_2')
36 grid on
```

This gives the solution depicted in Figure 1.

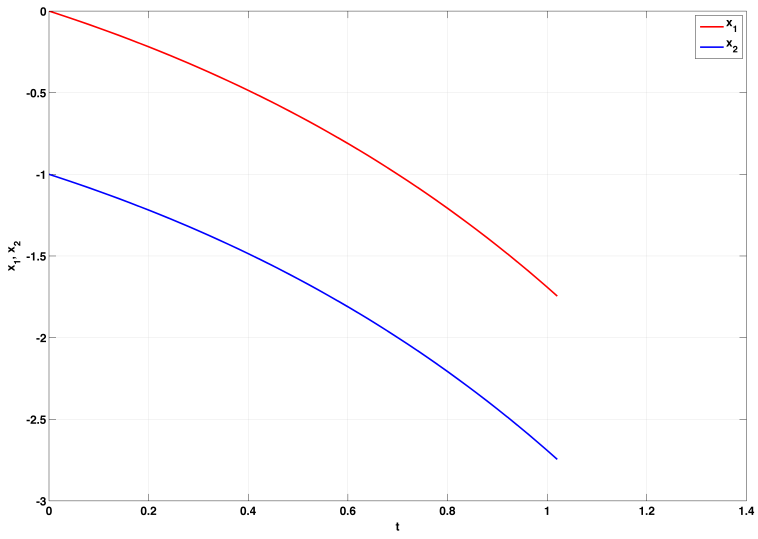

Figure 1: Open-loop trajectories, obtained via Euler forward discretization.

b) Assume that you discretized the dynamics (3b) via (4) and the inputs are parametrized as piecewise constant. Furthermore, suppose that the constraints are evaluated the discretization points $t_{k+1} = t_k + h$ and the objective is approximated as $\sum_{k=0}^{N-1} h(\mathbf{x}(t_k)^T\mathbf{x}(t_k) + 0.005u^2(t_k))$. Of which type is the resulting NLP?

Solution: The dynamics and the constraints lead to linear and affine inequalties. The objective leads to a function that is quadratic in $x(t_k)$ and $u(t_k)$. Thus we obain a quadratic program.

c) Solve the OCP (3) via a direct simultaneous approach. Use a piecewise constant parametrization of the input

$$\text{For } t \in [t_k, t_{k+1}]: \quad u(t) = w_{k+1}, \qquad t_{k+1} = t_k + h, \quad k = 0 : N-1$$

Plot the optimal input profile. Plot $x_1(t)$ versus t and $x_2(t), g(t,\mathbf{x}) + x_2(t)$ versus t. *Hint: Define the constraints and the objective as* `sdpvar` *variables.*

Solution:

```
function P2EX2

clc
t0 = 0;
tf = 1;
h  = 0.02;
c = 1;
x0 = [0; -1];
N = (tf - t0)/h +1;

x = sdpvar(2, N+1);
u = sdpvar(1, N+1);
t = sdpvar(1, N+1);
g = sdpvar(1, N+1);
F = sdpvar(1, N+1);


t(1)    = t0;
x(:,1)  = x0;
F(1)    = 0;

%% express discretized dynamics as equality constraints
for i=1:N
    x(:,i+1) =  x(:, i) + h*([0, 1; 0, c]*x(:,i) + [0; 1]*u(i));
    t(i+1)   = t(i) + h;
    g(i)     = 8*(t(i)-0.5)^2 -0.5 - x(2,i);
    F(i+1)   = F(i) + h*(x(1,i)^2 + x(2,i)^2 + 5E-3*u(i)^2);
end
g(N+1) =  8*(t(N+1)-0.5)^2 -0.5 - x(2, N+1);

%% add contraints
input_constraint = [];
state_constraint = [];

% input
for i  =  1:N
    input_constraint = input_constraint + set(-20<= u <= 20);
    state_constraint = state_constraint + set(0<= g);
end
input_constraint = input_constraint +  set(u(N) == u(N+1));

%% express objective
objective =  sum(F(i+1));

% solve QP via Yalmip
yalmip_options = sdpsettings('showprogress',1, 'verbose',0, ...
'solver', 'quadprog');%, 'TolX', 1E-7, 'TolFun', 1E-7);
yalmip_options.quadprog.TolX = 1E-14;
yalmip_options.quadprog.TolFun = 1E-14;

constraints = input_constraint + state_constraint;
solvesdp(constraints, objective,yalmip_options) %solve problem

%% plot solutions
x = double(x);
u = double(u);
g = double(g);
double(objective)
t_plot = [t0:1e-4:tf];
g_plot = 8.*(t_plot-0.5).^2 -0.5;
```

```
61
62  figure
63  subplot(3,1,1)
64  plot(t, x(1,:))
65  xlabel('t')
66  ylabel('x_1')
67
68  subplot(3,1,2)
69  plot(t, x(2,:), 'b')
70  hold on
71  plot(t_plot, g_plot, 'r')
72  xlabel('t')
73  ylabel('x_2, g')
74  axis([0, 1, -1, 0.6]);
75
76  subplot(3,1,3)
77  stairs(t, u(:))
78  xlabel('t')
79  ylabel('u')
```

This gives the solution depicted in Figure 2.

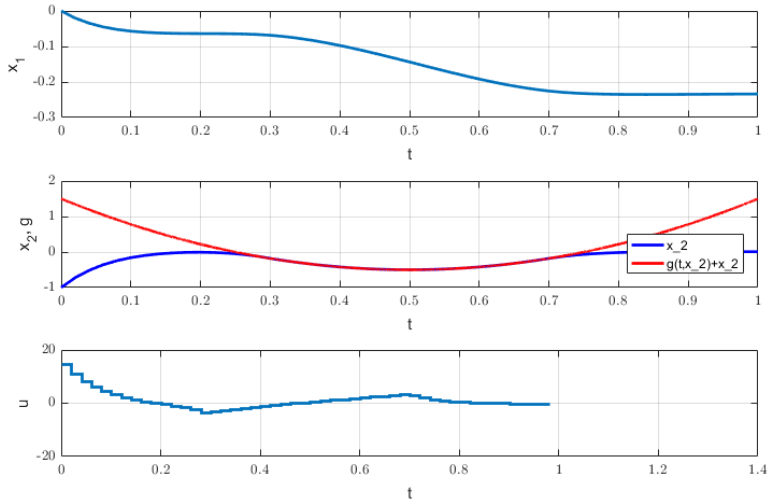

Figure 2: Optimal solution trajectories, obtained via Euler forward discretization.

d) In order to verify your solution integrate (3b) subject to your optimal solution $u^\star(t)$ with a variable stepsize integrator from MATLAB. Add the obtained trajectories to the plots from part c). *Hint: Use the solver* ode45 *and set the relative and absolute error tolerances to* 10^{-12}. *Integrate over the intervals* $[t_k, t_{k+1}], \quad k = 0 : N-1$ *separately by keeping* u *constant.*

Solution:

```
%% verify solution via accurate integration
options = odeset('RelTol', 1E-12, 'AbsTol', 1E-12);
t_ode = [];
z_ode = [];

z0i = [x0; 0];
for i = 1:N
    t_span = [(i-1)*h, i*h]+t0;
    [t_i, z_i] = ode45(@dynamics, t_span, z0i, options, u(i), c);
    %[t_i, z_i] = ode45(@dynamics, t_span, z0i, options, (u(i)+u(i+1))/2, c);
    t_ode = [t_ode; t_i];
    z_ode = [z_ode; z_i];
    z0i   = z_ode(end, :);
end

z_ode(end, 3)
figure(gcf)
subplot(3,1,1)
hold on
plot(t_ode, z_ode(:,1), 'g')

subplot(3,1,2)
hold on
plot(t_ode, z_ode(:,2), 'g')

function dx = dynamics(t,x,u,c)
dx(1,1) =   x(2);
dx(2,1) = c*x(2) + u;
dx(3,1) =   x(1)^2 + x(2)^2 + 0.005*u^2;
end
```

This gives the solution depicted in Figure 3. One can clearly see that the simple Euler forward discretization leads to some error in the trajectories.

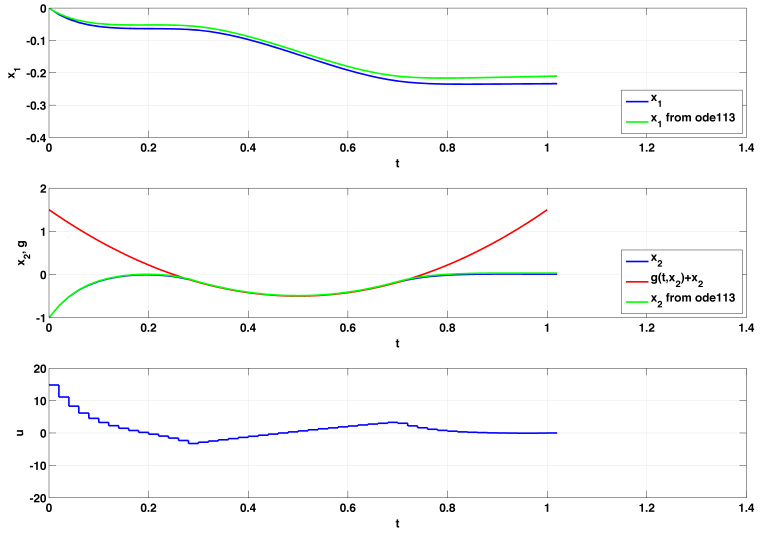

Figure 3: Optimal solution trajectories ($c = 1$) via Euler forward and solutions via ode45.

e) Now, set the model parameter set $c = 3$. Repeat c) and d) for this value of c. What do you observe? Try to explain your results. Can you suggest remedies?

Solution: This gives the solution depicted in Figure 4. One can clearly see that the error is increased compared to part d). This can be explained by the fact that for $c = 3$ we have shifted poles of system (3b) further to the right half plane.

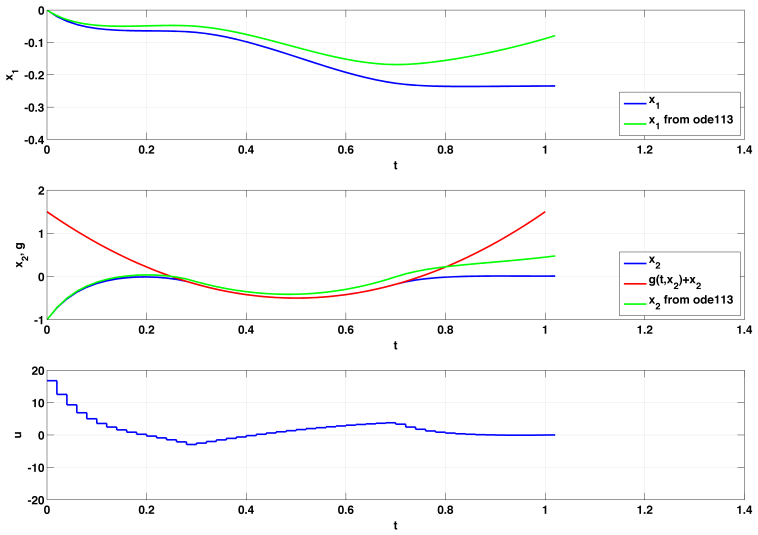

Figure 4: Optimal solution trajectories ($c = 3$) via Euler forward and solutions via `ode45`.

To avoid this error we have to improve the precision of the intergation. For general systems (linear or nonlinear) this can be achieved either via smaller stepsizes h or by shifting to a higher order integration scheme.[1]

f) Now, replace the Euler forward discretization by a second-order Runge-Kutta integration scheme. Use Heun's method[2] which reads

$$\tilde{\mathbf{x}}(t+h) = \mathbf{x}(t) + h\mathbf{f}(t, \mathbf{x}(t)) \tag{5}$$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \frac{h}{2}\Big(\mathbf{f}(t, \mathbf{x}(t)) + \mathbf{f}(t+h, \tilde{\mathbf{x}}(t+h))\Big). \tag{6}$$

Repeat c) and d) for this value of c. What do you observe? Compare your results to ones obtained in f).

Solution: We exchange the Euler forward integration by Heun's method in the `Yalmip` code:

```
x_temp   =  x(:, i) + h*([0, 1; 0, c]*x(:,i) + [0; 1]*u(i));
x(:,i+1) =  x(:, i) + h/2*([0, 1; 0, c]*x(:,i) + [0; 1]*u(i) + ...
[0, 1; 0, c]*x_temp + [0; 1]*u(i+1));
```

Additionally, we have to adapt the input applied for the integration via `ode45`:

```
[t_i, z_i] = ode45(@dynamics, t_span, z0i, options, (u(i)+u(i+1))/2, c);
```

This yields the results as depicted in Figure 5. We see that the errors are reduced by the shift to Heun's method. As a bottomline it is important to keep in mind that the accuracy of the integration plays an important role in the numerical solution of OCPs.

[1] In the case of linear systems, one could also use an extact discretization via the command `c2d`.

[2] Heun's method is also known as *trapezoidal rule*.

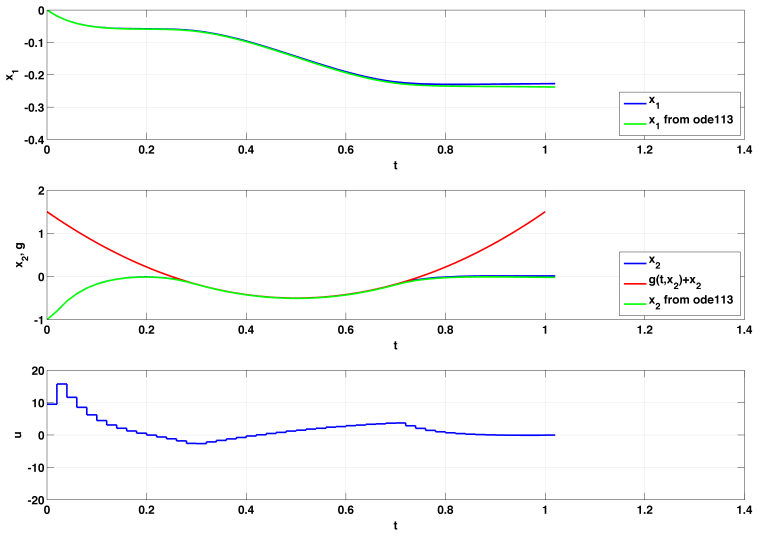

Figure 5: Optimal solution trajectories ($c = 3$) via Heun's method and solutions via `ode45`.

Exercise 3: Direct Collocation

Next, we will try to solve a simple OCP via a simple collocation method. Consider

$$\min_{u(\cdot)} \int_0^1 \frac{1}{2} u^2(t) dt \tag{7a}$$

subject to $\forall t \in [0,1]$:

$$\dot{x}(t) = -x(t) + u(t), \quad x(0) = 1,\ x(1) = 0. \tag{7b}$$

For simplicity we consider a piecewise constant parametrization of the input

$$\text{for } t \in [t_k, t_{k+1}]: \quad u(t) = w_{k+1}, \qquad t_{k+1} = t_k + h, \quad k = 0 : N-1,$$

where h is the length of each collocation stage. On each stage the state trajectory is to be approximated by Lagrange polynomials of degree 3:

$$x(t) = \sum_{i=0}^{3} \xi_i^k \phi_i^{(3)} \left(\frac{t - t_k}{t_{k+1} - t_k} \right), \quad t \in [t_k, t_{k+1}], k = 0, \ldots, N-1 \tag{8}$$

with

$$\phi_i^{(3)} = \prod_{\substack{q=0 \\ q \neq i}}^{3} \frac{\tau - \tau_q}{\tau_i - \tau_q}$$

and the following collocation points $\{\tau_0, \tau_1, \ldots, \tau_3\} = \{0, 0.25, 0.5, 0.75\}$.

a) Reformulate the OCP in Mayer form. Discretise the reformulated problem via collocation. State the corresponding expressions for states and constraints of the resulting NLP.

Solution: Rewriting OCP (7) in Mayer form yields:

$$\min_{u(\cdot)} z(1) dt$$

subject to $\forall t \in [0,1]$:

$$\dot{x}(t) = -x(t) + u(t), \quad x(0) = 1,\ x(1) = 0$$
$$\dot{z}(t) = \frac{1}{2} u^2(t), \quad z(0) = 0.$$

The NLP of the collocation approach reads:

$$\min_{\zeta,\xi,\omega} \sum_{i=0}^{3} \zeta_i^N \phi_i^{(3)}(1)$$

subject to

$$\sum_{i=0}^{3} \xi_i^k \left(\frac{1}{t_{k+1}-t_k} \dot{\phi}_i^{(3)}(\tau_q) + \phi_i^{(3)}(\tau_q) \right) - \omega^k = 0, \quad q = 1,2,3, k = 0,\ldots,N-1$$

$$\sum_{i=0}^{3} \xi_i^1 \phi_i^{(3)}(0) = 1, \quad \sum_{i=0}^{3} \xi_i^N \phi_i^{(3)}(1) = 0$$

$$\sum_{i=0}^{3} \xi_i^k \phi_i^{(3)}(1) = \sum_{i=0}^{3} \xi_i^{k+1} \phi_i^{(3)}(0), \quad k = 1,\ldots,N-1$$

$$\sum_{i=0}^{3} \zeta_i^k \left(\frac{1}{t_{k+1}-t_k} \dot{\phi}_i^{(3)}(\tau_q) \right) - \frac{1}{2}(\omega^k)^2 = 0, \quad q = 1,2,3, k = 0,\ldots,N-1$$

$$\sum_{i=0}^{3} \zeta_i^1 \phi_i^{(3)}(0) = 0$$

$$\sum_{i=0}^{3} \zeta_i^k \phi_i^{(3)}(1) = \sum_{i=0}^{3} \zeta_i^{k+1} \phi_i^{(3)}(0), \quad k = 1,\ldots,N-1.$$

b) Write a `MATLAB` script to solve the integration of the reformulated dynamics via `Yalmip`. Consider the given initial condition $x_0 = 1$, $t = 0$ to $t = 1$, the number of stages $N = 10$ and $\forall t \in [0,1]: \quad u(t) = 0$. *Hint: Define the collocation parameters ξ_i^k as* `sdpvar` *variables. Use the files* `lagrange.m`, `dlagrange.m` *to evaluate the Lagrange polynomials.*

Solution:

```
function P2EX3b
close all

%% definition of variables
t0 = 0;
t1 = 1;
N = 10; %# number of stages
tspan = [0:(t1-t0)/N:t1];
x0 = 1
x1 = 0
z0 = 0

tau = [0:0.25:0.75]; % collocation points on each stage
n_co = length(tau); %order of the collocation method +1
u = zeros(1,N); sdpvar(1,N);
xi  = sdpvar(1, n_co*(N)); % state variable x
zi  = sdpvar(1, n_co*(N)); % augmented state for Mayer term

%% collocation constraints
dyn_eq_x = [];
dyn_eq_z = [];
for i = 1:N
    for q = 2:n_co
        %index = (i-1)*n_co+q
        index_span = IndexSpan(i, n_co); %[(i-1)*n_co+1:1:i*n_co]

        h      = tspan(i+1)-tspan(i);
        phi_x     = lagrange([xi(1,index_span)], tau, tau(q));
        dphi_x    = dlagrange([xi(1,index_span)], tau, tau(q));
        dyn_eq_x  = dyn_eq_x + set(1/h*dphi_x + phi_x - u(i) == 0);

```

```
        phi_z     = lagrange([zi(1,index_span)], tau, tau(q));
        dphi_z    = dlagrange([zi(1,index_span)], tau, tau(q));
        dyn_eq_z  = dyn_eq_z + set(1/h*dphi_z - 0.5*u(i)^2 == 0);
    end

   if i == 1 % initial condition
        phi_x     = lagrange(xi(index_span), tau, 0);
        dyn_eq_x  = dyn_eq_x + set(phi_x - x0 ==0);

        phi_z     = lagrange(zi(index_span), tau, 0);
        dyn_eq_z  = dyn_eq_z + set(phi_z - z0 ==0);
    else
    % continuity between stages
        phi_x_left  = lagrange([xi(1,IndexSpan(i-1, n_co))], tau, 1.);
        phi_x_right = lagrange([xi(1,IndexSpan(i, n_co))], tau, 0.);
        dyn_eq_x    = dyn_eq_x + set(phi_x_left == phi_x_right);

        phi_z_left  = lagrange([zi(1,IndexSpan(i-1, n_co))], tau, 1.);
        phi_z_right = lagrange([zi(1,IndexSpan(i, n_co))], tau, 0.);
        dyn_eq_z    = dyn_eq_z + set(phi_z_left == phi_z_right);
   end
end

yalmip_options = sdpsettings('showprogress',1, 'verbose',0);
constraints = dyn_eq_x + dyn_eq_z;
objective = 1;

%% integrate via solving the collocation constraints
result = solvesdp(constraints, objective, yalmip_options)

%% plot solution
u = double(u);
xi = double(xi);
zi = double(zi);

xplot = [];
tplot = [];
Nplot = 10;

for i = 1:N
    dt = [tspan(i):(tspan(i+1)-tspan(i))/Nplot:tspan(i+1)]';
    index_span = IndexSpan(i, n_co);
    tplot = [tplot; dt];
    xplot = [xplot; lagrange([xi(1,index_span)], tau, ...
   (dt - tspan(i))/(tspan(i+1)-tspan(i)))];
end

figure
title('State')
plot(tplot, xplot, 'Linewidth', 2)
xlabel('t')
ylabel('x')
grid on
end
%% subfunction
function span = IndexSpan(i, n_co)
    span = [(i-1)*n_co+1:1:i*n_co];
end
```

The results are depicted in Figure 6.

c) Extend your code in order to solve OCP (7).

Solution:

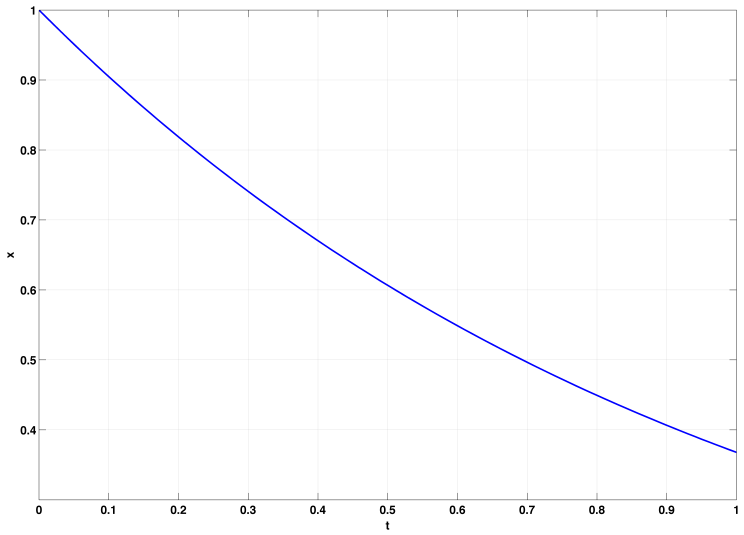

Figure 6: Open-loop trajectories obtained via collocation.

```
function P2EX3c
close all

%% definition of variables
t0 = 0;
t1 = 1;
N = 10; %# number of stages
tspan = [0:(t1-t0)/N:t1];
x0 = 1;
x1 = 0;
z0 = 0;

tau = [0:0.25:0.75]; % collocation points on each stage
n_co = length(tau); %order of the collocation method +1
u = sdpvar(1,N);
xi  = sdpvar(1, n_co*(N)); % state variable x
zi  = sdpvar(1, n_co*(N)); % augmented state for Mayer term

%% collocation constraints
dyn_eq_x = [];
dyn_eq_z = [];
for i = 1:N

    for q = 2:n_co
        %index = (i-1)*n_co+q
        index_span = IndexSpan(i, n_co); %[(i-1)*n_co+1:1:i*n_co]
        h      = tspan(i+1)-tspan(i);
        phi_x     = lagrange([xi(1,index_span)], tau, tau(q));
        dphi_x    = dlagrange([xi(1,index_span)], tau, tau(q));
        dyn_eq_x  = dyn_eq_x + set(1/h*dphi_x + phi_x - u(i) == 0);

        phi_z     = lagrange([zi(1,index_span)], tau, tau(q));
        dphi_z    = dlagrange([zi(1,index_span)], tau, tau(q));
        dyn_eq_z  = dyn_eq_z + set(1/h*dphi_z - 0.5*u(i)^2 == 0);
    end

    if i == 1 % initial condition
        phi_x     = lagrange(xi(index_span), tau, 0);
        dyn_eq_x  = dyn_eq_x + set(phi_x - x0 ==0);

```

```
        phi_z     = lagrange(zi(index_span), tau, 0);
        dyn_eq_z  = dyn_eq_z + set(phi_z - z0 ==0);
    else
    % continuity between stages
        phi_x_left  = lagrange([xi(1,IndexSpan(i-1, n_co))], tau, 1.);
        phi_x_right = lagrange([xi(1,IndexSpan(i, n_co))], tau, 0.);
        dyn_eq_x    = dyn_eq_x + set(phi_x_left == phi_x_right);

        phi_z_left  = lagrange([zi(1,IndexSpan(i-1, n_co))], tau, 1.);
        phi_z_right = lagrange([zi(1,IndexSpan(i, n_co))], tau, 0.);
        dyn_eq_z    = dyn_eq_z + set(phi_z_left == phi_z_right);
    end

    if i == N % terminal constraint
        phi_x_right = lagrange([xi(1,IndexSpan(i, n_co))], tau, 1);
        dyn_eq_x  = dyn_eq_x + set(phi_x_right == x1);
    end
  %  keyboard
end

yalmip_options = sdpsettings('showprogress',1, 'verbose',0);

constraints = dyn_eq_x + dyn_eq_z;
objective = lagrange([zi(1,IndexSpan(N, n_co))], tau, 1.);

%% solving the OCP
result = solvesdp(constraints, objective,yalmip_options)


%% plot solution
u = double(u);
u = [u, u(end)];
xi = double(xi);
zi = double(zi);

xplot = [];
tplot = [];
Nplot = 10;

for i = 1:N
    dt = [tspan(i):(tspan(i+1)-tspan(i))/Nplot:tspan(i+1)]';
    index_span = IndexSpan(i, n_co);
    tplot = [tplot; dt];
    xplot = [xplot; lagrange([xi(1,index_span)], tau, ...
    (dt - tspan(i))/(tspan(i+1)-tspan(i)))];
end

nu = 1/sinh(1);
u_opt = -nu*exp(tplot-1);
x_opt = exp(-tplot)- nu/exp(1)*sinh(tplot);

figure
subplot(2,1,1)
stairs(tspan, u, 'r', 'Linewidth', 2)
hold on
plot(tplot, u_opt, 'b','Linewidth', 2)
grid on
xlabel('t')
ylabel('u')
title('input')
legend('numerical solution', 'analytical solution')

subplot(2,1,2)
plot(tplot, xplot, 'r','Linewidth', 2)
hold on
plot(tplot, x_opt, 'b')
```

```
grid on
xlabel('t')
ylabel('x')
title('state')
legend('numerical solution', 'analytical solution')

%% compute value of objective function
objective = lagrange([zi(1,IndexSpan(i, n_co))], tau, 1.)
end
```

d) Compare your results with the analytical solution which can be found in the class textbook, Example 4.23 (p. 133-134). Consider different numbers of collocation stages $N \in \{5, 10, 20\}$.

Solution: From the textbook we know that the true optimal solution is

$$u^\star(t) = -\nu^\star e^{t-1}, \quad \nu^\star = \frac{1}{\sinh(1)}$$
$$x^\star(t) = \quad e^{-t} - \frac{\nu^\star}{e}\sinh(t).$$

The results for $N = 10$ are depicted in Figure 7. One can see that already in this case we obtain a very good approximation of the true optimal state trajectory.

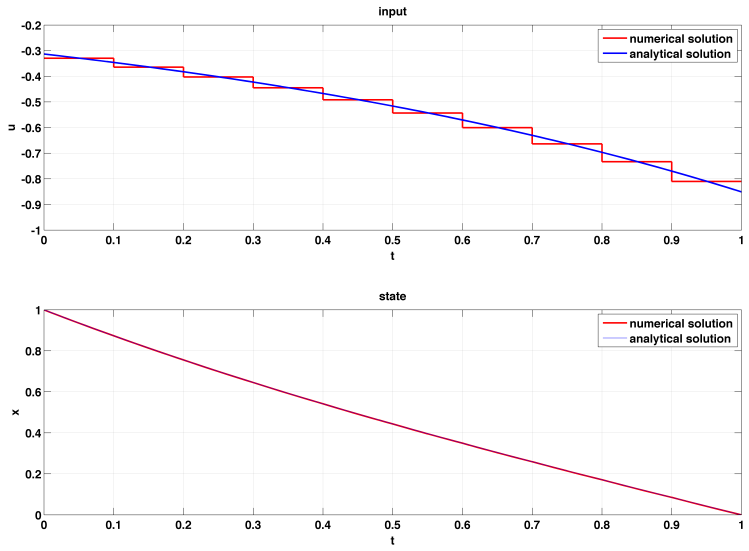

Figure 7: Open-loop trajecotries obtained via collocation ($N = 10$) and true optimal solution.

References

[1] M. Henson and D. Seborg. *Nonlinear Process Control*. Prentice Hall, Upper Saddle River, NJ, 1997.