Optimal Control Part II

CMPT 882

Feb. 11

Optimal Control

Final cost

Running cost

$$\underset{u(\cdot)}{\text{minimize}} \; l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)\,dt$$

Cost functional, $J\big(x(\cdot), u(\cdot)\big)$

$$\text{subject to } \dot{x}(t) = f\big(x(t), u(t)\big)$$

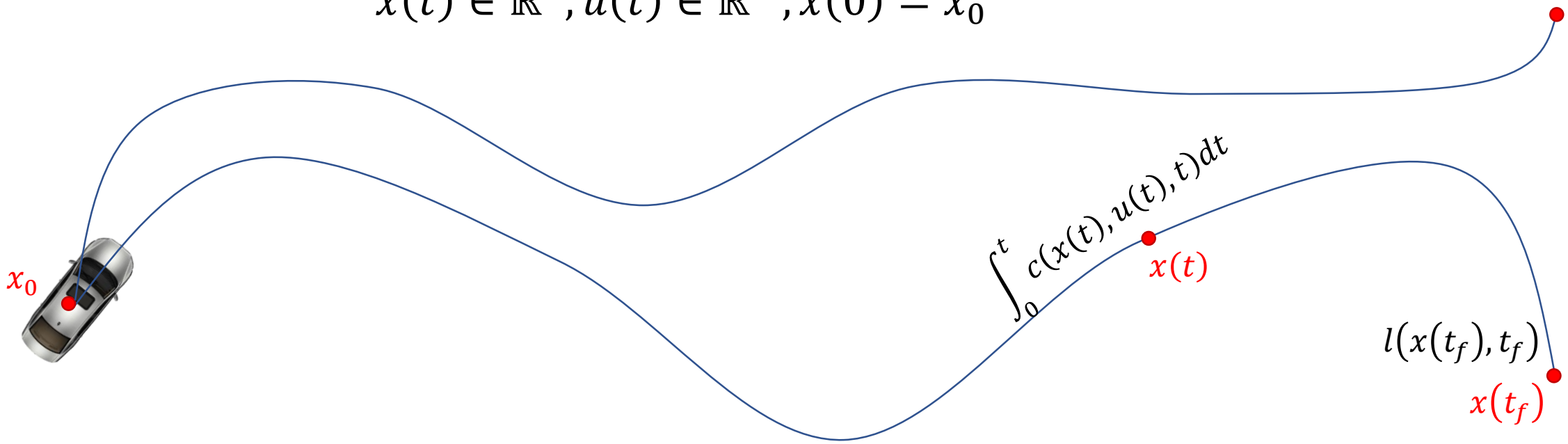
Dynamic model

$$g\big(x(t), u(t)\big) \geq 0$$

Additional constraints

- Eg. actuation limits

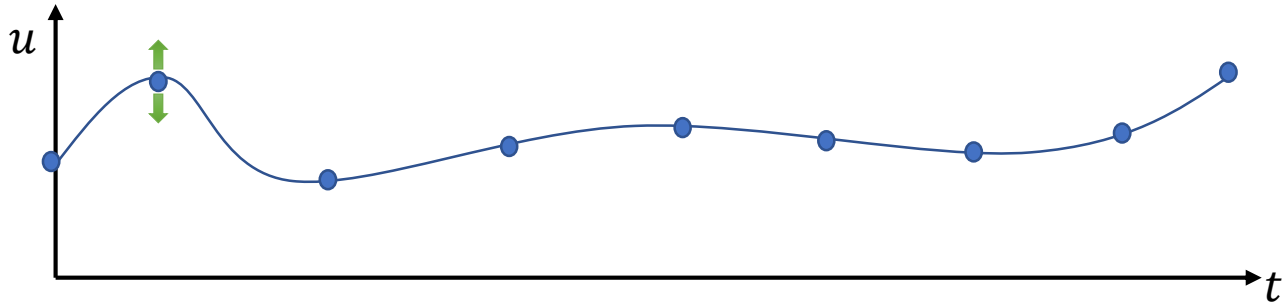
$$x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, x(0) = x_0$$



Optimal Control: Types of Solutions

$$\underset{u(\cdot)}{\text{minimize}}\ l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)dt$$

$$\text{subject to } \dot{x}(t) = f\big(x(t), u(t)\big)$$



- Open-loop control
  - Find $u(t)$ for $t \in [0, t_f]$
  - Scalable, but errors will add up
- Closed-loop control
  - Find $u(t, x)$ for $t \in [0, t_f], x \in \mathbb{R}^n$
  - Not scalable, but robust
  - "Special" techniques needed (eg. Reinforcement learning) for large n
- Receding horizon control:
  - Find $u(t)$ for $t \in [0, T]$, use $u(t)$ for $t \in [0, h]$, then find $u(t)$ for $t \in [h, T + h]$ and repeat
  - Has features of both open- and closed-loop control

Outline: Open-Loop Control

- Optimal Control Problems
- Differential flatness
- Direct Methods (Numerical Methods)
  - Shooting methods
  - Collocation
  - CasADi Matlab toolbox

Differential Flatness Definition

A nonlinear system $\dot{x} = f(x, u)$ is differentially flat if there exists a function α such that

$$z = \alpha\left(x, u, \dots, u^{(p)}\right)$$

and we can write the solutions of the nonlinear system as functions of z and a finite number of derivatives

$$x = \beta\left(z, \dot{z}, \dots, z^{(q)}\right)$$
$$u = \gamma\left(z, \dot{z}, \dots, z^{(q)}\right)$$

- z is called the “flat outputs”

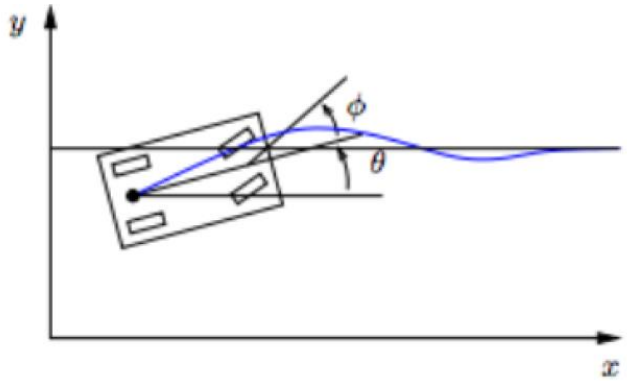
Differential Flatness

- How to check if a system is differentially flat:
  - Mathematically, find α, β, γ such that

$$z = \alpha\left(x, u, \dots, u^{(p)}\right)$$
$$x = \beta\left(z, \dot{z}, \dots, z^{(q)}\right)$$
$$u = \gamma\left(z, \dot{z}, \dots, z^{(q)}\right)$$

  - Practically, do a search to see if someone else found it for your system
    - Simple car models, cars pulling cars (trailers), quadrotor
  - If you cannot find α, β, γ in the literature, another option is to be the first one to find it, and publish a paper
    - D. Mellinger and V. Kumar. *Minimum snap trajectory generation and control for quadrotors*, ICRA 2011.

Differential Flatness Example



$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \frac{v}{l}\tan\phi$$

- The following car model is differentially flat:
- Why?

$$z = \alpha\left(\boldsymbol{x}, u, \dots, u^{(p)}\right) \longrightarrow z = (x, y) \Rightarrow \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

$$\boldsymbol{x} = \beta\left(z, \dot{z}, \dots, z^{(q)}\right) \longrightarrow \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \arctan\left(\frac{\dot{x}}{\dot{y}}\right) \end{bmatrix} \Rightarrow \boldsymbol{x} = \begin{bmatrix} z_1 \\ z_2 \\ \arctan\left(\frac{\dot{z}_1}{\dot{z}_2}\right) \end{bmatrix}$$

$$u = \gamma\left(z, \dot{z}, \dots, z^{(q)}\right) \longrightarrow \begin{bmatrix} v \\ \phi \end{bmatrix} = \begin{bmatrix} \frac{\dot{x}}{\cos\theta} \\ \arctan\left(\frac{l\dot{\theta}}{v}\right) \end{bmatrix} \Rightarrow u = \begin{bmatrix} \frac{\dot{z}_1}{\cos\left(\arctan\left(\frac{\dot{z}_1}{\dot{z}_2}\right)\right)} \\ \arctan\left(\frac{l\dot{z}_1 \frac{d}{dt}\left(\arctan\left(\frac{\dot{z}_1}{\dot{z}_2}\right)\right)}{\cos\left(\arctan\left(\frac{\dot{z}_1}{\dot{z}_2}\right)\right)}\right) \end{bmatrix}$$

Trajectory Generation for Simple Car Model

- Suppose $\boldsymbol{x}_0 = (0,0,0), \boldsymbol{x}_f = (1,0,0), t_f = T.$
  - Goal: Find $\boldsymbol{x}(t)$ and $u(t)$ such that $\boldsymbol{x}(0) = x_0, \boldsymbol{x}(T) = \boldsymbol{x}_f$, and

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \frac{v}{l}\tan\phi$$

- Method using differential flatness:
  - $\boldsymbol{x}_0 = \beta\left(z(0), \dot{z}(0), \dots, z^{(q)}(0)\right) \Rightarrow \begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix} z_1(0) \\ z_2(0) \\ \arctan\left(\frac{\dot{z}_1(0)}{\dot{z}_2(0)}\right)\end{bmatrix}$
  - $\boldsymbol{x}_f = \beta\left(z(T), \dot{z}(T), \dots, z^{(q)}(T)\right) \Rightarrow \begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix} z_1(T) \\ z_2(T) \\ \arctan\left(\frac{\dot{z}_1(T)}{\dot{z}_2(T)}\right)\end{bmatrix}$
  - Find $z(t)$ that satisfies the above
  - Once $z(t)$ is found, then we can obtain $\boldsymbol{x}$ and u: $\boldsymbol{x} = \beta\left(z, \dot{z}, \dots, z^{(q)}\right), u = \gamma\left(z, \dot{z}, \dots, z^{(q)}\right)$

In x-space, we need to consider dynamics
In z-space, β and γ replace dynamics

Trajectory Generation for Simple Car Model

- Find $z(t)$ that satisfies
  - Still difficult…

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix} z_1(0) \\ z_2(0) \\ \arctan\left(\frac{\dot{z}_1(0)}{\dot{z}_2(0)}\right)\end{bmatrix}, \qquad \begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix} z_1(T) \\ z_2(T) \\ \arctan\left(\frac{\dot{z}_1(T)}{\dot{z}_2(T)}\right)\end{bmatrix}$$

- New plan:
  - Let $\psi_1(t) = 1, \psi_2(t) = t, \psi_3(t) = t^2, \psi_4(t) = t^3$
  - Let $z_1(t) = b_{10} + b_{11}t + b_{12}t^2 + b_{13}t^3$ $\quad z_2(t) = b_{20} + b_{21}t + b_{22}t^2 + b_{23}t^3$

$$\Rightarrow \dot{z}_1(t) = b_{11} + 2b_{12}t + 3b_{13}t^2 \qquad \Rightarrow \dot{z}_2(t) = b_{21} + 2b_{22}t + 3b_{23}t^2$$

$$\Rightarrow \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 1 & T & T^2 & T^3\\ 0 & 1 & 2T & 3T^2\end{bmatrix}\begin{bmatrix}b_{10}\\ b_{11}\\ b_{12}\\ b_{13}\end{bmatrix} = \begin{bmatrix}z_1(0)\\ \dot{z}_1(0)\\ z_1(T)\\ \dot{z}_1(T)\end{bmatrix} \qquad \Rightarrow \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 1 & T & T^2 & T^3\\ 0 & 1 & 2T & 3T^2\end{bmatrix}\begin{bmatrix}b_{20}\\ b_{21}\\ b_{22}\\ b_{23}\end{bmatrix} = \begin{bmatrix}z_2(0)\\ \dot{z}_2(0)\\ z_2(T)\\ \dot{z}_2(T)\end{bmatrix}$$

What to do with b?

$$(**) \qquad \begin{bmatrix} \psi_1(0) & \psi_2(0) & \cdots & \psi_N(0) \\ \dot{\psi}_1(0) & \dot{\psi}_2(0) & \cdots & \dot{\psi}_N(0) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1^{(q)}(0) & \psi_2(0)^{(q)} & \cdots & \psi_N^{(q)}(0) \\ \psi_1(T) & \psi_2(T) & \cdots & \psi_N(T) \\ \dot{\psi}_1(T) & \dot{\psi}_2(T) & \cdots & \dot{\psi}_N(T) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1^{(q)}(T) & \psi_2^{(q)}(T) & \cdots & \psi_N^{(q)}(T) \end{bmatrix} \begin{bmatrix} b_{i1} \\ b_{i2} \\ \vdots \\ b_{iN} \end{bmatrix} = \begin{bmatrix} z_i(0) \\ \dot{z}_i(0) \\ \vdots \\ z_i^{(q)}(0) \\ z_i(T) \\ \dot{z}_i(T) \\ \vdots \\ z_i^{(q)}(T) \end{bmatrix}$$

$$z(t) = \sum_{i=1}^{N} b_i \psi_i(t), \quad \dot{z}(t) = \sum_{i=1}^{N} b_i \dot{\psi}_i(t), \quad \cdots \quad z^{(q)}(t) = \sum_{i=1}^{N} b_i \phi_i^{(q)}(t)$$

$$\boldsymbol{x} = \boldsymbol{\beta}\left(\boldsymbol{z}, \dot{\boldsymbol{z}}, \ldots, \boldsymbol{z}^{(\boldsymbol{q})}\right)$$

$$\boldsymbol{u} = \boldsymbol{\gamma}\left(\boldsymbol{z}, \dot{\boldsymbol{z}}, \ldots, \boldsymbol{z}^{(\boldsymbol{q})}\right)$$

ψ_i: basis functions

- Your choice!
- You can choose N too!
- E.g. $\psi_i = x^{i-1}$ -- monomial basis

q is from dynamics

- Determines number of rows
  - i.e. number of equations
- Can't choose this

N is chosen

- Determines number of columns
  - i.e. number of variables in b

- N too small: no solutions
- N very large: many solutions

Optimal Control Problem

Final cost

Running cost

$$\underset{u(\cdot)}{\text{minimize}} \; l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)\,dt$$

$$\text{subject to } \dot{x}(t) = f\big(x(t), u(t)\big)$$

$$g\big(x(t), u(t)\big) \geq 0$$

$$x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m,$$

$$x(0) = x_0, x\big(t_f\big) = x_f$$

Cost functional, $J\big(x(\cdot), u(\cdot)\big)$

Dynamic model

Additional constraints

- Eg. actuation limits

Optimal Control Problem

Final cost

Running cost

$$\underset{\boldsymbol{u}(\cdot)}{\text{minimize}} \; l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)dt$$

$$\text{subject to } \boldsymbol{\dot{x}(t) = f\big(x(t), u(t)\big)}$$

$$g\big(x(t), u(t)\big) \geq 0$$

$$\boldsymbol{x(t) \in \mathbb{R}^n}, u(t) \in \mathbb{R}^m,$$

$$\boldsymbol{x(0) = x_0, x\big(t_f\big) = x_f}$$

Cost functional, $J\big(x(\cdot), u(\cdot)\big)$

Dynamic model

Additional constraints

- Eg. actuation limits

$$\underset{\boldsymbol{b}}{\text{minimize}} \; l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)dt$$

$$\text{subject to } (**)$$

$$g\big(x(t), u(t)\big) \geq 0$$

$$u(t) \in \mathbb{R}^m, x(0) = x_0$$

Optimal Control Problem

Final cost

Running cost

$$\underset{\boldsymbol{u}(\cdot)}{\text{minimize}} \; l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), \boldsymbol{u}(\boldsymbol{t}), t)dt$$

Cost functional, $J\big(x(\cdot), u(\cdot)\big)$

$$\text{subject to } \dot{\boldsymbol{x}}(\boldsymbol{t}) = \boldsymbol{f}\big(\boldsymbol{x}(\boldsymbol{t}), \boldsymbol{u}(\boldsymbol{t})\big)$$

Dynamic model

$$g\big(x(t), u(t)\big) \geq 0$$

Additional constraints

- Eg. actuation limits

$$\boldsymbol{x}(\boldsymbol{t}) \in \mathbb{R}^{\boldsymbol{n}}, u(t) \in \mathbb{R}^m,$$

$$\boldsymbol{x}(\boldsymbol{0}) = \boldsymbol{x_0}, \boldsymbol{x}\big(\boldsymbol{t_f}\big) = \boldsymbol{x_f}$$

$$\underset{\boldsymbol{b}}{\text{minimize}} \; l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)dt$$

$$\text{subject to } (**)$$

$$g\big(x(t), u(t)\big) \geq 0$$

$$u(t) \in \mathbb{R}^m, x(0) = x_0$$

$$\boldsymbol{z} = \boldsymbol{\alpha}\big(\boldsymbol{x}, \boldsymbol{u}, \dots, \boldsymbol{u}^{(\boldsymbol{p})}\big)$$

$$\boldsymbol{x} = \boldsymbol{\beta}\big(\boldsymbol{z}, \dot{\boldsymbol{z}}, \dots, \boldsymbol{z}^{(\boldsymbol{q})}\big)$$

$$\boldsymbol{u} = \boldsymbol{\gamma}\big(\boldsymbol{z}, \dot{\boldsymbol{z}}, \dots, \boldsymbol{z}^{(\boldsymbol{q})}\big)$$

Differential Flatness: Key Points

$$\underset{\boldsymbol{b}}{\text{minimize}}\ l\big(x(t_f), t_f\big) + \int_0^{t_f} c(x(t), u(t), t)dt$$

subject to $(**)$

$$g\big(x(t), u(t)\big) \geq 0$$

$$u(t) \in \mathbb{R}^m, x(0) = x_0$$

$$\boldsymbol{z = \alpha\big(x, u, \dots, u^{(p)}\big)}$$

$$\boldsymbol{x = \beta\big(z, \dot{z}, \dots, z^{(q)}\big)}$$

$$\boldsymbol{u = \gamma\big(z, \dot{z}, \dots, z^{(q)}\big)}$$

- Trajectory generation via solving algebraic equations $(**)$
- Other constraints can be transformed into z space
- Cost/performance index also transformed into z space
- After obtaining b, we can obtain x and u

Direct methods

- Differential flatness
  - Algebraic method for special system dynamics

- **Direct shooting**
  - **Parametrize control**
  - **Numerical example with CasADi**

- Collocation
  - Parametrize both state and control

Single shooting

$$\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & l\big(x(t_f), t_f\big) + \int_{t_0}^{t_f} c(x(t), u(t), t)dt \\
\text{subject to} \quad & \dot{x} = f(x, u) \\
& g\big(x(t), u(t)\big) \geq 0, \qquad t \in [t_0, t_f] \\
\text{where} \quad & x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, x(t_0) = x_0
\end{aligned}$$

- Discretize: $t_0 < t_1 < \cdots < t_N \coloneqq t_f$

  $u(t) = q_i$ for $t \in [t_i, t_{i+1}]$

Single shooting

$$\begin{aligned} \underset{u(\cdot)}{\text{minimize}} \quad & l\big(x(t_f), t_f\big) + \int_{t_0}^{t_f} c(x(t), u(t), t)dt \\ \text{subject to} \quad & \dot{x} = f(x, u) \\ & g\big(x(t), u(t)\big) \geq 0, \qquad t \in [t_0, t_f] \\ \text{where} \quad & x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, x(t_0) = x_0 \end{aligned}$$

- Discretize: $t_0 < t_1 < \cdots < t_N \coloneqq t_f$

  $u(t) = q_i$ for $t \in [t_i, t_{i+1}]$

- Numerically integrate dynamics and cost:
  - Simple example:

Dynamics (Forward Euler): $x(t_{i+1}) \approx x(t_i) + f(x(t_i), q_i)(t_{i+1} - t_i)$

Cost: $\int_{t_0}^{t_f} c(x(t), u(t), t)dt \approx \sum_{i=0}^{N-1} c(x(t_i), q_i, t_i)(t_{i+1} - t_i)$

Single shooting

$$\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & l\big(x(t_f), t_f\big) + \int_{t_0}^{t_f} c(x(t), u(t), t)dt \\
\text{subject to} \quad & \dot{x} = f(x, u) \\
& g\big(x(t), u(t)\big) \geq 0, \qquad t \in [t_0, t_f] \\
\text{where} \quad & x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, x(t_0) = x_0
\end{aligned}$$

- Discretized problem:

$$\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & l(x(t_N), t_N) + \sum_{i=0}^{N-1} c(x(t_i), q_i, t_i)(t_{i+1} - t_i) \\
\text{subject to} \quad & \forall i \in \{0,1, \dots, N-1\}, \\
& \quad x(t_{i+1}) = x(t_i) + f(x(t_i), q_i)(t_{i+1} - t_i) \\
& \quad g(x(t_i), q_i) \geq 0
\end{aligned}$$

Introduction to CasADi

- Numerical optimization software
  - Tailored towards transcription of optimal control problems into NLPs
  - Eg. single shooting, multiple shooting, collocation
- Interfaces
  - Matlab, Python, C++, etc.
- Tools:
  - NLP solvers (eg. IPOPT, SNOPT, etc.)
  - Convex solvers (eg. Gurobi, Cplex, etc.)
  - Symbolic integrators

Introduction to CasADi

- Home page
  - https://github.com/casadi/casadi/wiki
- Installation
  - https://github.com/casadi/casadi/wiki/InstallationInstructions
- User guide
  - http://casadi.sourceforge.net/v3.4.0/users_guide/casadi-users_guide.pdf

Example

$$
\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & \int_0^{10} (x_1^2 + x_2^2 + u^2)dt \\
\text{subject to} \quad & \dot{x}_1 = (1 - x_2^2)x_1 - x_2 + u \\
& \dot{x}_2 = x_1 \\
& x_1 \geq -0.25 \\
& -1 \leq u \leq 1
\end{aligned}
$$

- Reformulation as nonlinear program (NLP)
  - $N = 100$ with uniform time intervals
  - Forward Euler integration for dynamics
  - First-order integration

Adapted from example in casADi user manual

Example

$$
\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & \int_0^{10} (x_1^2 + x_2^2 + u^2)dt \\
\text{subject to} \quad & \dot{x}_1 = (1 - x_2^2)x_1 - x_2 + u \\
& \dot{x}_2 = x_1 \\
& x_1 \geq -0.25 \\
& -1 \leq u \leq 1
\end{aligned}
$$

1. Installation
2. Preliminary setup
3. Integrating dynamics
4. NLP formulation
5. Solve and plot

Adapted from example in casADi user manual

https://github.com/casadi/casadi/wiki/InstallationInstructions

Installing CasADi

Option 1: Binary installation (recommended)

Install CasADi 3.4.3

For Python users: `pip install casadi` (you must have `pip --version` >= 8.1!)

Grab a binary from the table (for MATLAB, use the newest compatible version below):

	Windows	Linux	Mac
Matlab	R2014b or later, R2014a, R2013a or R2013b	R2014b or later, R2014a	R2015a or later, R2014b, R2014a
Octave	4.2.2 (32bit / 64bit)	4.2.2	4.2.2
Python	Py27 (32bit[1,2] / 64bit[2]), Py35 (32bit[2] / 64bit[2]), Py36 (32bit[2] / 64bit[2])	Py27, Py35, Py36	Py27, Py35, Py36

Import and preliminary setup

- sym command: symbolic variables
- Symbolic differentiation and integration inside casadi
- Note that `xdot` and `Jt` are also symbolic

```
import casadi.*
```

```
%% Preliminaries
T = 10; % Time horizon
N = 100; % number of control intervals

% Declare model variables
x1 = SX.sym('x1');
x2 = SX.sym('x2');
x = [x1; x2];
u = SX.sym('u');

% Model equations
xdot = [(1-x2^2)*x1 - x2 + u; x1];

% Objective term
Jt = x1^2 + x2^2 + u^2;
```

Integrating Dynamics

- f takes as input x and u, and outputs xdot and Jt, as defined previously
- One can replace Forward Euler with for example RK4
  - Code just shows Forward Euler for one time step
- F takes as input an initial condition X0, and a vector of controls U, and outputs the final state X and total cost Q

```
%% Integrate dynamics
% Forward Euler
dt = T/N;

f = Function('f', {x, u}, {xdot, Jt}); % f has two inputs and two outputs

X0 = MX.sym('X0', 2);
U = MX.sym('U');

[Xdot, Jk] = f(X0, U);
X = X0 + dt*Xdot;
Q = Jk*dt;

% F has two inputs and two outputs, with the corresponding names
F = Function('F', {X0, U}, {X, Q}, {'x0','p'}, {'xf', 'qf'});

% Evaluate at a test point
Fk = F('x0',[0.2; 0.3],'p',0.4);
disp(Fk.xf)
disp(Fk.qf)
```

NLP Formulation

- Start with empty NLP
  - Controls `w`, `w0`, with bounds `lbw`, `ubw`
  - `J` is the total cost
    - Will use loop to add up (integrate) the cost
  - `g` is constraint function on x
    - `lbg`, `ubg` give bounds

$$
\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & \int_0^{10} (x_1^2 + x_2^2 + u^2)dt \\
\text{subject to} \quad & \dot{x}_1 = (1 - x_2^2)x_1 - x_2 + u \\
& \dot{x}_2 = x_1 \\
& x_1 \geq -0.25 \\
& -1 \leq u \leq 1
\end{aligned}
$$

```
%% Formulate the NLP
% Start with empty NLP
w={};
w0 = [];
lbw = [];
ubw = [];
J = 0;
g={};
lbg = [];
ubg = [];

% Iterate through time
Xk = [0; 1];
for k=0:N-1
```

NLP Formulation

$$
\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & \int_0^{10} (x_1^2 + x_2^2 + u^2)dt \\
\text{subject to} \quad & \dot{x}_1 = (1 - x_2^2)x_1 - x_2 + u \\
& \dot{x}_2 = x_1 \\
& x_1 \geq -0.25 \\
& -1 \leq u \leq 1
\end{aligned}
$$

- Control bounded between -1 and 1
  - Need a constraint at every time step
- Add up cost at every time step
- State constraints at every time step

```
% Iterate through time to obtain constraints at every time step
Xk = [0; 1];
for k=0:N-1
  % New NLP variable for the control
  Uk = MX.sym(['U_' num2str(k)]);
  w = {w{:}, Uk};
  lbw = [lbw, -1]; % Control is bounded between -1 and 1
  ubw = [ubw,  1];
  w0 = [w0,  0];   % Initial guess for the control is all zeros

  % Integrate till the end of the interval
  Fk = F('x0',Xk,'p', Uk);
  Xk = Fk.xf;
  J = J+Fk.qf;

  % Add inequality constraint: x1 is bounded between -0.25 and infinity
  g = {g{:}, Xk(1)};
  lbg = [lbg; -.25];
  ubg = [ubg;  inf];
end
```

Solve and Plot

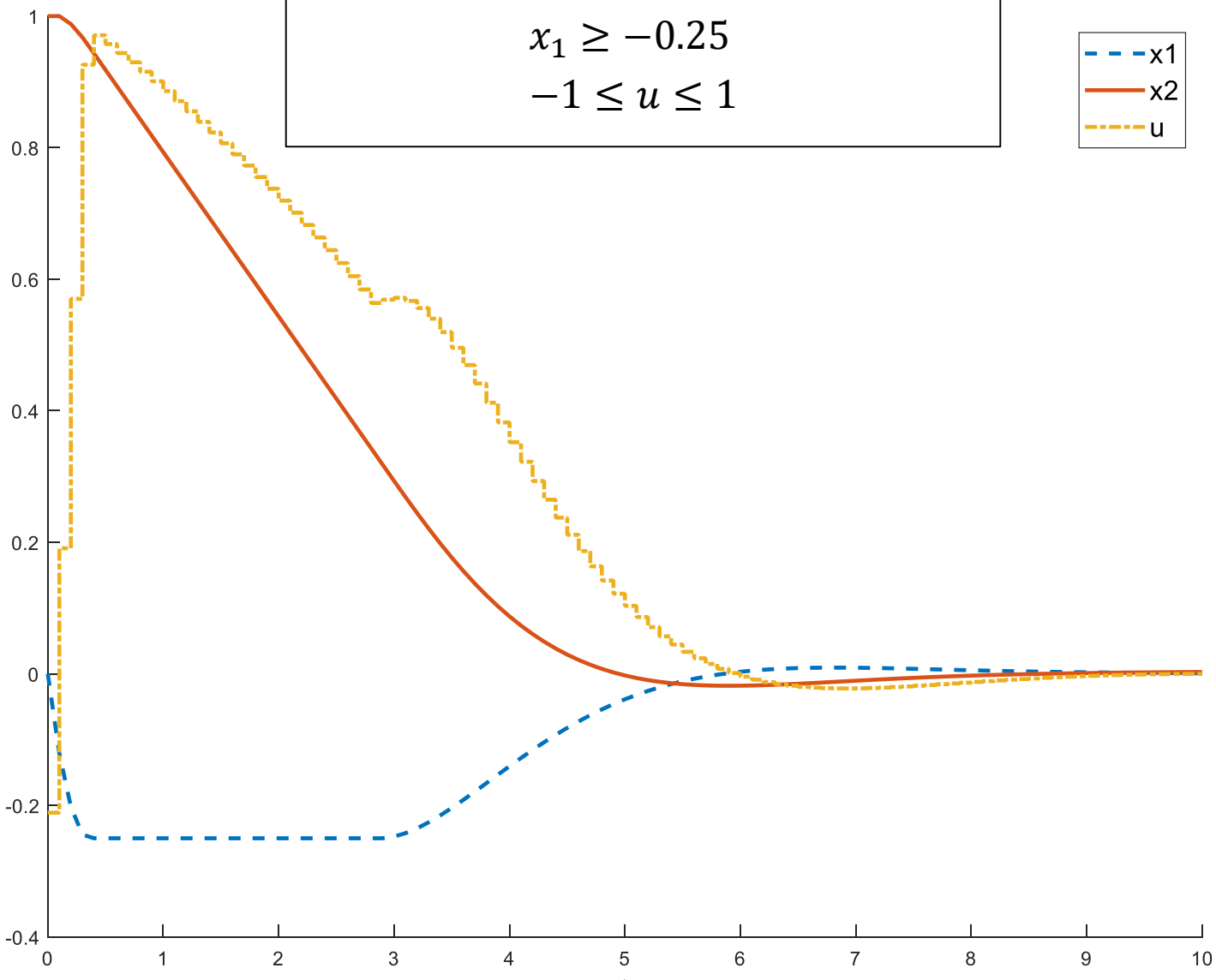
- See comments and documentation
  - Code will be uploaded to CourSys
- IPOPT is a built-in NLP solver
- Online documentation has python and Matlab examples

```
%% Solve the NLP
% Create an NLP solver
prob = struct( ...          % Problem struct
  'f', J, ...               % Objective function
  'x', vertcat(w{:}), ... % Decision variables (control)
  'g', vertcat(g{:}));    % Constraints

% use IPOPT solver (interior point optimizer)
solver = nlpsol('solver', 'ipopt', prob);

% Call the solver
sol = solver( ...
  'x0', w0, ...     % Initial guess
  'lbx', lbw, ...   % Bounds for the decision variables (control)
  'ubx', ubw, ...
  'lbg', lbg, ...   % Bounds for the constraint
  'ubg', ubg);

w_opt = full(sol.x);
```

Solve and Plot

- See comments and documentation
  - Code will be uploaded to CourSys
- IPOPT is a built-in NLP solver
- Online documentation has python and Matlab examples

$$\begin{aligned} \underset{u(\cdot)}{\text{minimize}} \quad & \int_0^{10} (x_1^2 + x_2^2 + u^2)dt \\ \text{subject to} \quad & \dot{x}_1 = (1 - x_2^2)x_1 - x_2 + u \\ & \dot{x}_2 = x_1 \\ & x_1 \geq -0.25 \\ & -1 \leq u \leq 1 \end{aligned}$$



Welcome to CasADi's documentation!

1. Introduction

CasADi is an open-source software tool for numerical optimization in general and optimal control (i.e. optimization involving differential equations) in particular. The project was started by Joel Andersson and Joris Gillis while PhD students at the Optimization in Engineering Center (OPTEC) of the KU Leuven under supervision of Moritz Diehl.

This document aims at giving a condensed introduction to CasADi. After reading it, you should be able to formulate and manipulate expressions in CasADi's symbolic framework, generate derivative information efficiently using **algorithmic differentiation**, to set up, solve and perform forward and adjoint sensitivity analysis for systems of ordinary differential equations (ODE) or differential-algebraic equations (DAE) as well as to formulate and solve nonlinear programs (NLP) problems and optimal control problems (OCP).

CasADi is available for C++, Python and MATLAB/Octave with little or no difference in performance. In general, the Python API is the best documented and is slightly more stable than the MATLAB API. The C++ API is stable, but is not ideal for getting started with CasADi since there is limited documentation and since it lacks the interactivity of interpreted languages like MATLAB and Python. The MATLAB module has been tested successfully for Octave (version 4.0.2 or later).

1.1. What CasADi is and what it is not

CasADi started out as a tool for algorithmic differentiation (AD) using a syntax borrowed from computer algebra systems (CAS), which explains its name. While AD still forms one of the core functionalities of the tool, the scope of the tool has since been considerably broadened, with the addition of support for ODE/DAE integration and sensitivity analysis, nonlinear programming and interfaces to other numerical tools. In its current form, it is a general-purpose tool for gradient-based numerical optimization – with a strong focus on optimal control – and CasADi is just a name without any particular meaning.

It is important to point out that CasADi is **not** a conventional AD tool, that can be used to calculate derivative information from existing user code with little to no modification. If you have an existing model written in C++, Python or MATLAB/Octave, you need to be prepared to reimplement the model using CasADi syntax.

Secondly, CasADi is **not** a computer algebra system. While the symbolic core does include an increasing set of tools for manipulate symbolic expressions, these capabilities are very limited compared to a proper CAS tool.

Finally, CasADi is not an "optimal control problem solver", that allows the user to enter an OCP and then gives the solution back. Instead, it tries to provide the user with a set of "building blocks" that can be used to implement general-purpose or specific-purpose OCP solvers efficiently with a modest programming effort.

1.2. Help and support

If you find simple bugs or lack some feature that you think would be relatively easy for us to add, the simplest thing is simply to write to the forum, located at http://forum.casadi.org/. We check the forum regularly and try to respond as quickly as possible. The only thing we expect for this kind of support is that you cite us, cf. Section 1.3, whenever you use CasADi in scientific work.

If you want more help, we are always open for academic or industrial cooperation. An academic cooperation usually take the form of a co-authorship of a peer reviewed paper, and an industrial cooperation involves a negotiated consulting contract. Please contact us directly if you are interested in this.

1.3. Citing CasADi

If you use CasADi in published scientific work, please cite the following:

```
@Article{Andersson2018,
```

Single shooting

$$
\begin{aligned}
\underset{q}{\text{minimize}} \quad & l(x(t_N), t_N) + \sum_{i=0}^{N-1} c(x(t_i), q_i, t_i)(t_{i+1} - t_i) \\
\text{subject to} \quad & \forall i \in \{0,1, \dots, N-1\}, \\
& \quad x(t_{i+1}) = x(t_i) + f(x(t_i), q_i)(t_{i+1} - t_i) \\
& \quad g(x(t_i), q_i) \geq 0
\end{aligned}
$$

- Main disadvantage: integration error

```
>> Xk

Xk =

F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(F(
>> |
```

Adapted from example in casADi user manual

Multiple shooting

$$\underset{q}{\text{minimize}} \quad l(\boldsymbol{x(t_N)}, t_N) + \sum_{i=0}^{N-1} c(\boldsymbol{x(t_i)}, q_i, t_i)(t_{i+1} - t_i)$$

subject to $\forall i \in \{0,1,\dots,N-1\}$,

$$\boldsymbol{x(t_{i+1})} = \boldsymbol{x(t_i)} + f(\boldsymbol{x(t_i)}, q_i)(t_{i+1} - t_i)$$

$$g(\boldsymbol{x(t_i)}, q_i) \geq 0$$

$$\underset{\boldsymbol{s},q}{\text{minimize}} \quad h(\boldsymbol{s_N}, t_N) + \sum_{i=0}^{N-1} c(\boldsymbol{s_i}, q_i, t_i)(t_{i+1} - t_i)$$

subject to $\forall i \in \{0,1,\dots,N-1\}$,

$$\boldsymbol{s_{i+1}} = \boldsymbol{s_i} + f(\boldsymbol{s_i}, q_i)(t_{i+1} - t_i)$$

$$g(\boldsymbol{s_i}, q_i) \geq 0$$

Shooting Method Disadvantages

- Numerical integration
  - Potentially slow
  - Numerical errors