Shooting Methods

CMPT 419/983

Mo Chen

SFU Computing Science

2/10/2019

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

$$\begin{aligned} \underset{u(\cdot)}{\text{minimize}} \quad & l(x(T), T) + \int_0^T c(x(t), u(t), t)dt \\ \text{subject to} \quad & \dot{x} = f(x, u) \\ & g\big(x(t), u(t)\big) \geq 0, \qquad t \in [0, T] \\ \text{where} \quad & x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, x(0) = x_0 \end{aligned}$$

- Discretize: $0 < t_1 < \cdots < t_N \coloneqq T$

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

Single shooting

$$\begin{aligned} \underset{u(\cdot)}{\text{minimize}} \quad & l(x(T),T) + \int_0^T c(x(t),u(t),t)dt \\ \text{subject to} \quad & \dot{x} = f(x,u) \\ & g\big(x(t),u(t)\big) \geq 0, \qquad t \in [0,T] \\ \text{where} \quad & x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, x(t_0) = x_0 \end{aligned}$$

- 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

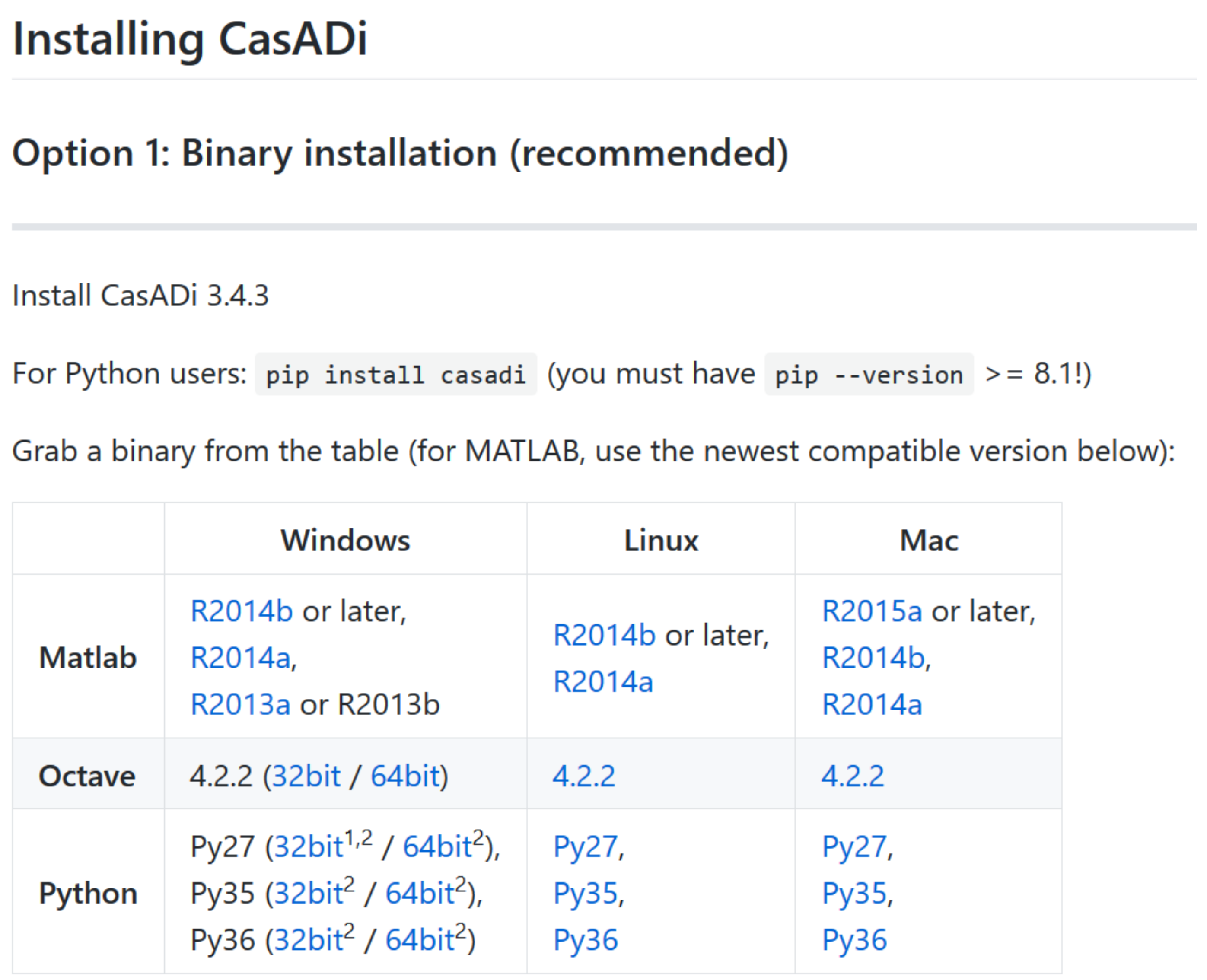
Example

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

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!)

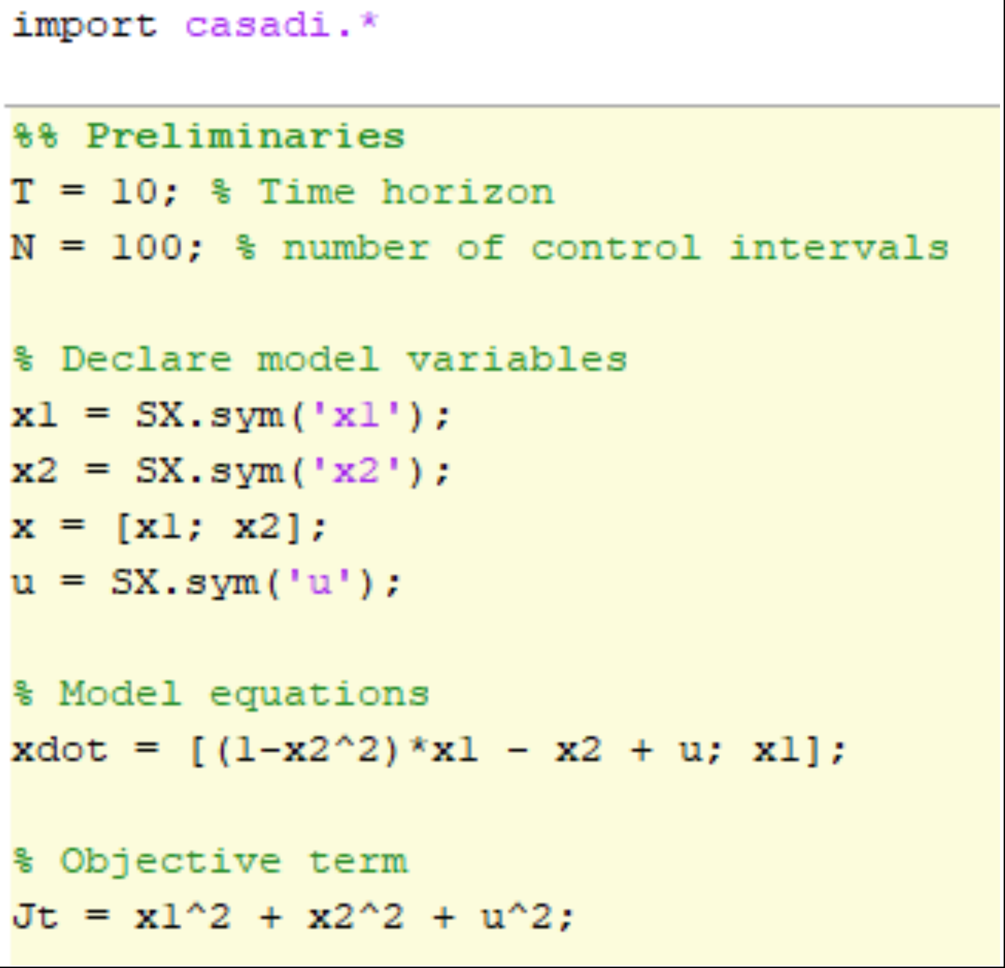
Grab a binary from the table (for MATLAB, use the newest compatible version below):

	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

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

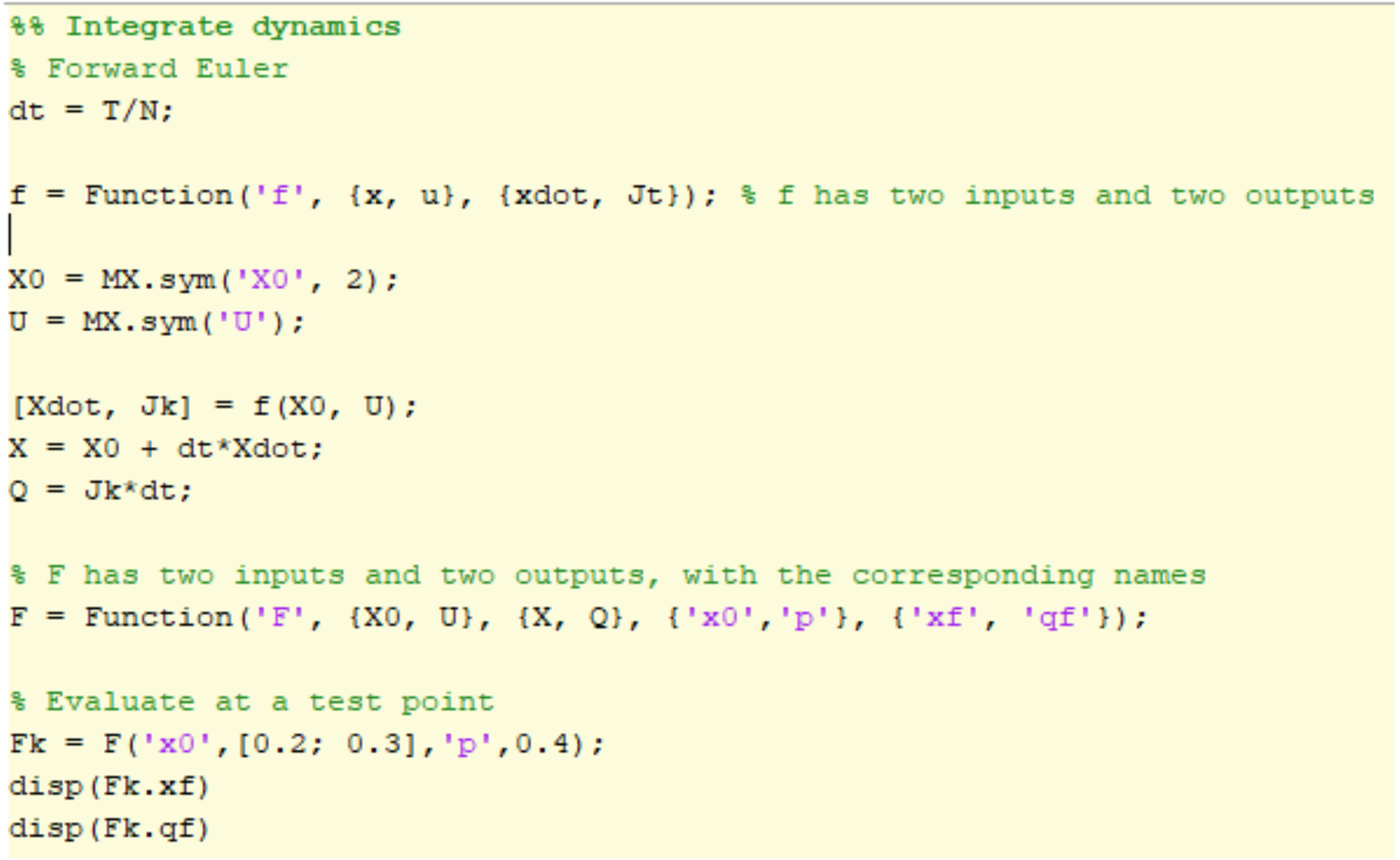
- sym command: symbolic variables
- Symbolic differentiation and integration inside casadi
- Note that `xdot` and `Jt` are also symbolic



```
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

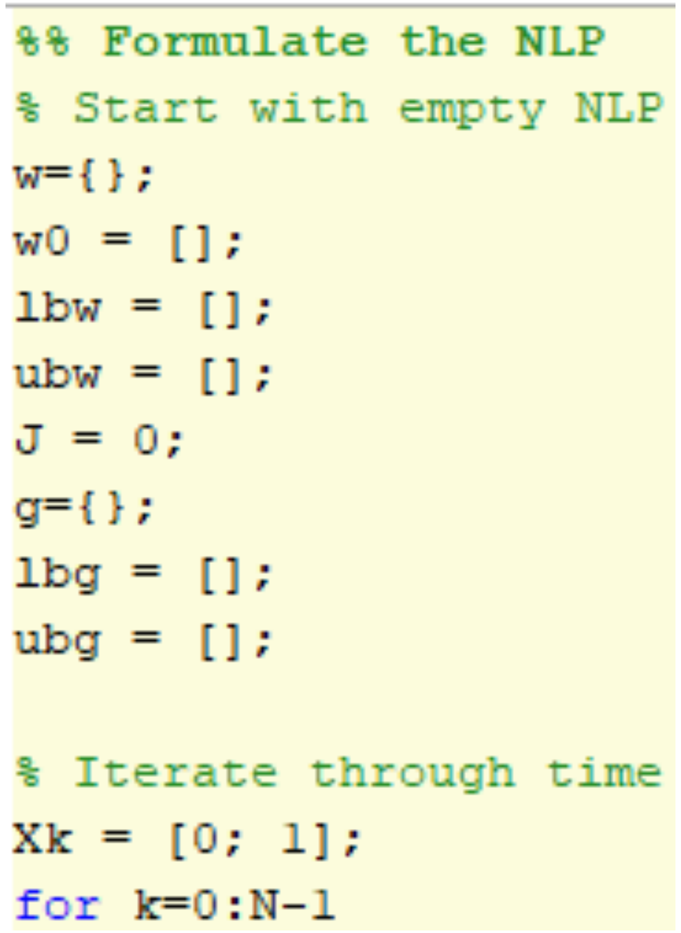


```
%% Integrate dynamics
% Forward Euler
dt = T/N;

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

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

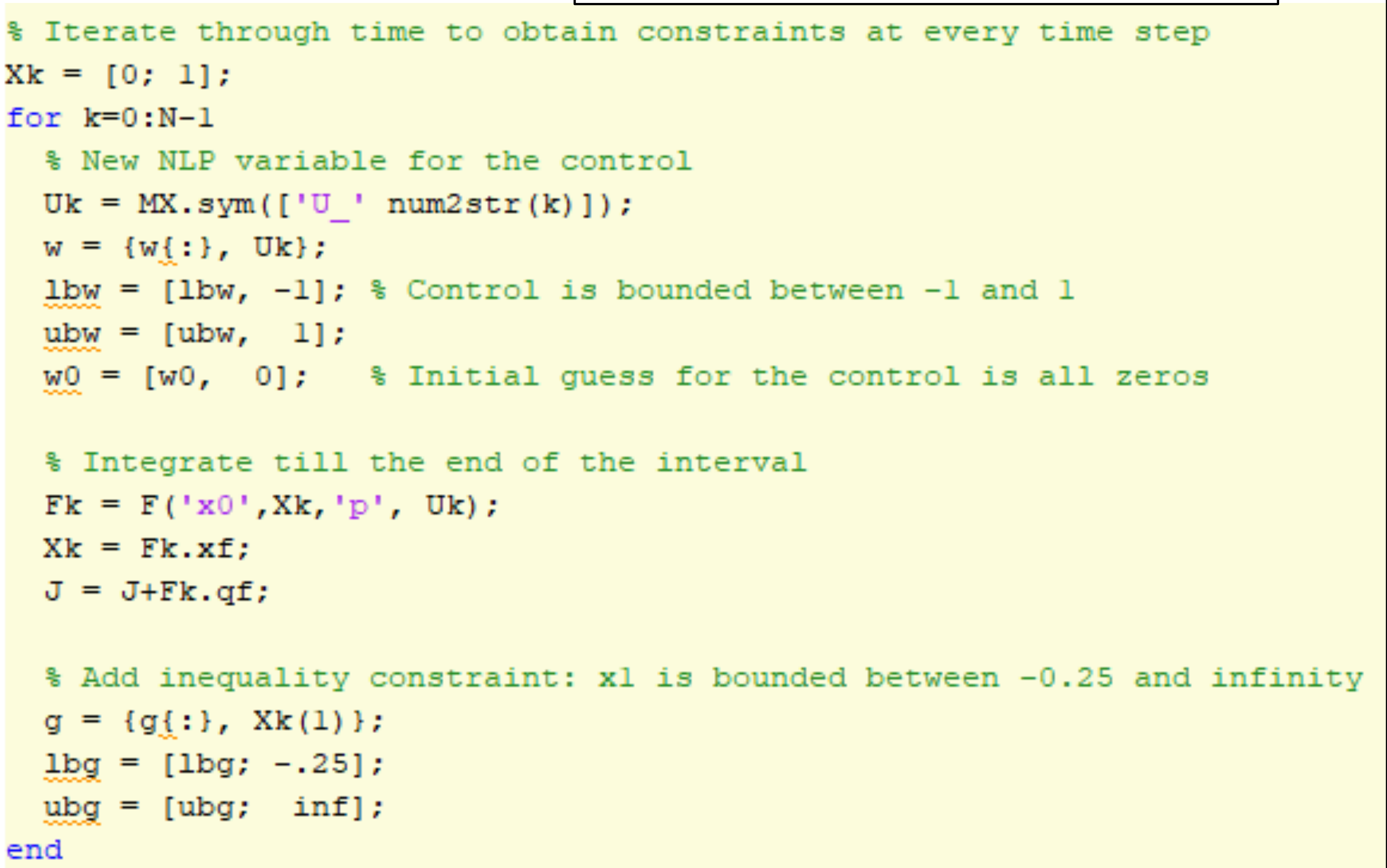


```
%% Formulate the NLP
% Start with empty NLP
w={};
w0 = [];
lbw = [];
ubw = [];
J = 0;
g={};
lbg = [];
ubg = [];

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

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

NLP Formulation

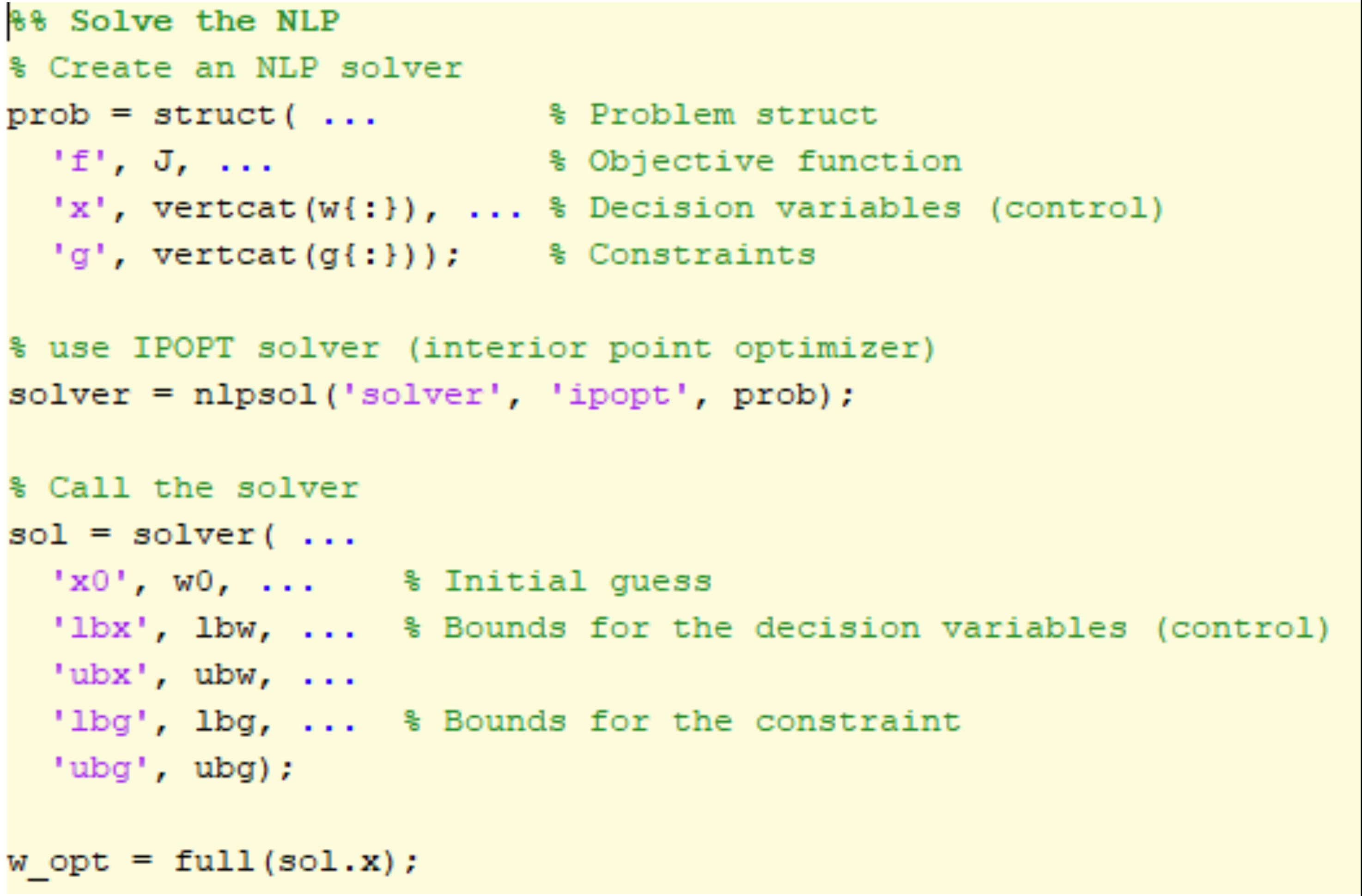
- Control bounded between −1 and 1
  - Need a constraint at every time step
- Compute next state and add up cost at every time step
- State constraints at every time step



```
% 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

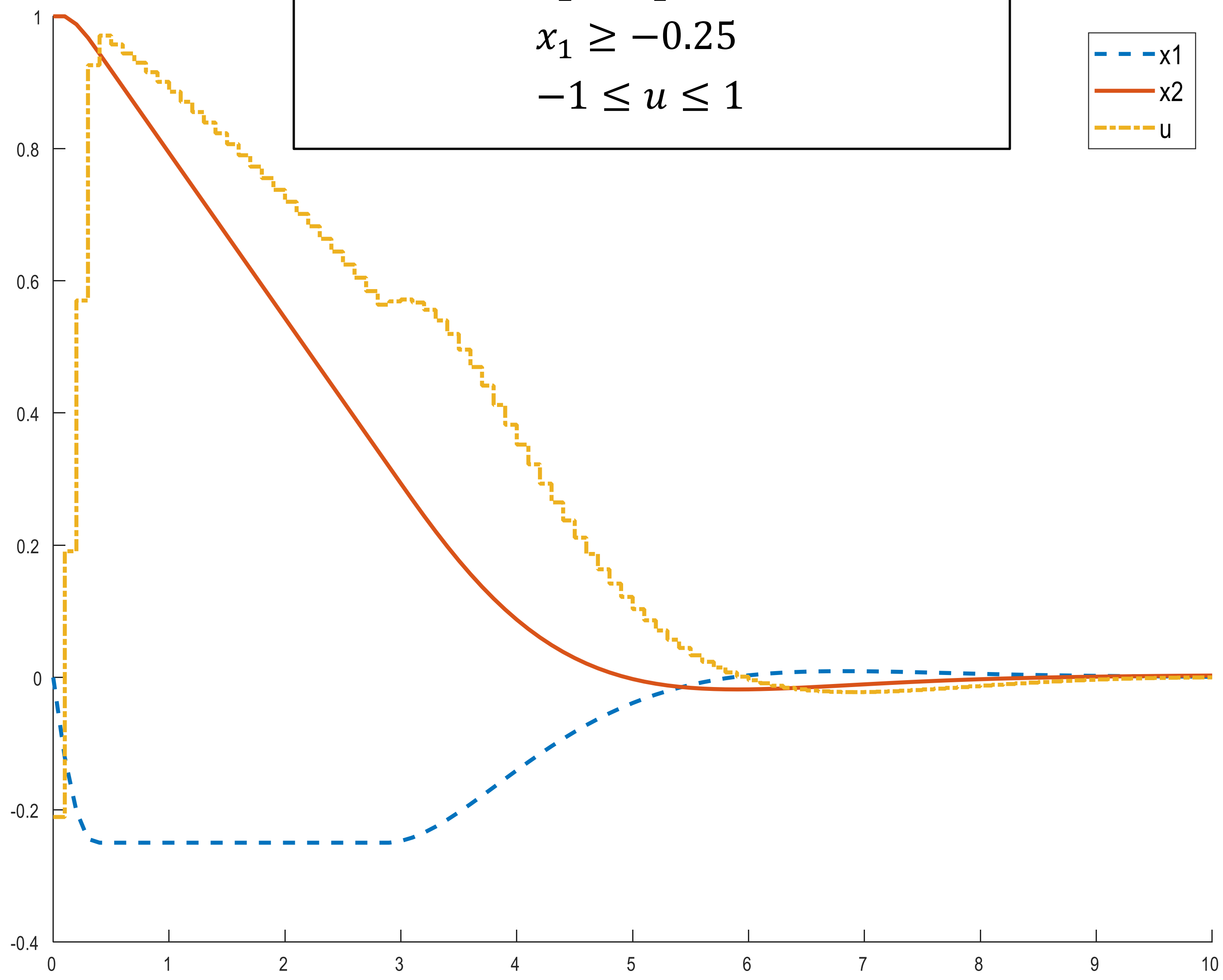


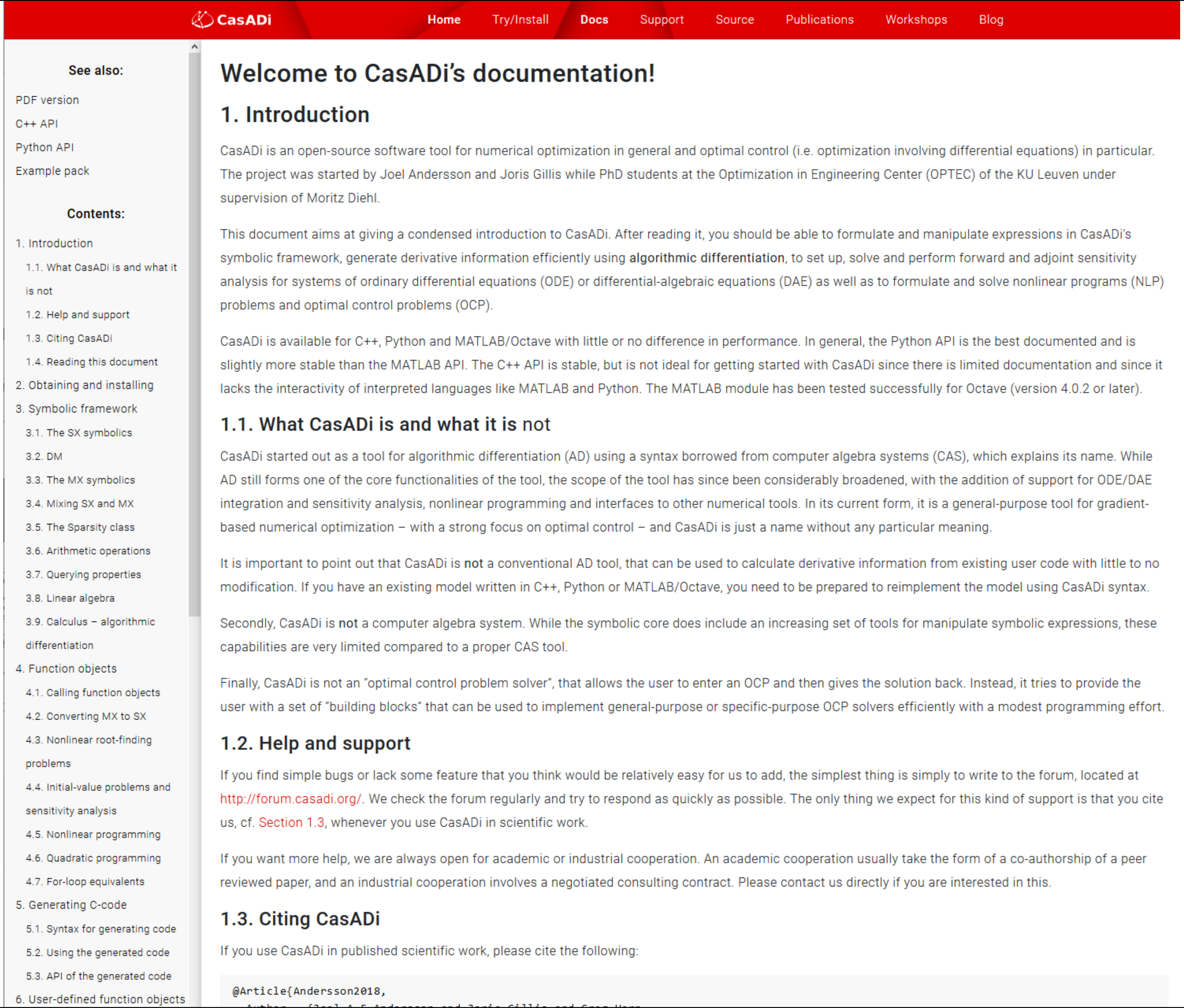
```
%% Solve the NLP
% Create an NLP solver
prob = struct( ...              % Problem struct
  'f', J, ...                   % Objective function
  'x', vertcat(w{:}), ...  % Decision variables (control)
  'g', vertcat(g{:}));     % Constraints

% 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

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





See also:

PDF version
C++ API
Python API
Example pack

Contents:

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

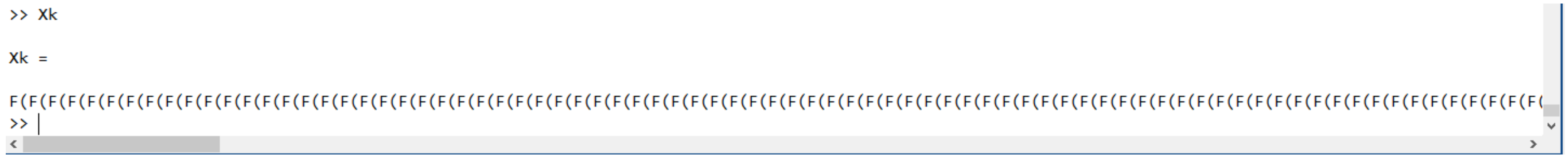
If you use CasADi in published scientific work, please cite the following:

```
@Article{Andersson2018,
```

Single shooting

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

- 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

Single Shooting

- Discretized problem:

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

- Variations: Different numerical schemes
  - For ODE constraint
  - For cost function

- Main disadvantage
  - Integration error
  - Errors in “earlier” controls can greatly affect final state
  - Initial guess matters a lot

Multiple shooting

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

$$\Downarrow$$

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

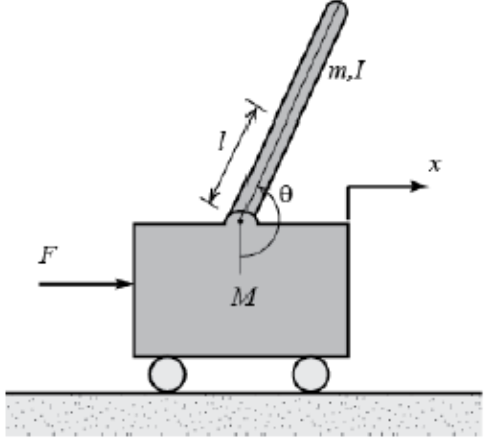
Multiple Shooting

- Discretized problem:

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

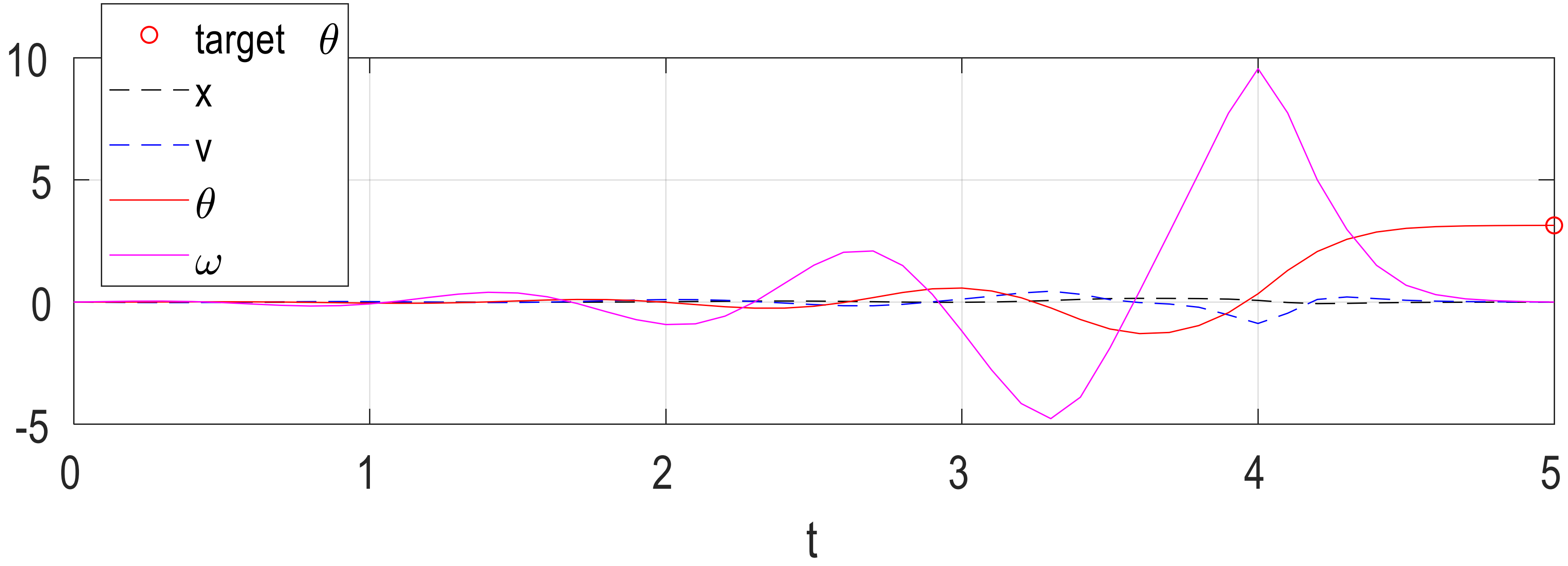
- Same variations as single shooting available (numerical schemes)
- State is now a decision variable
  - State constraints do not necessarily need to be satisfied throughout optimization process
  - Improves numerical stability
  - Reduces integration error

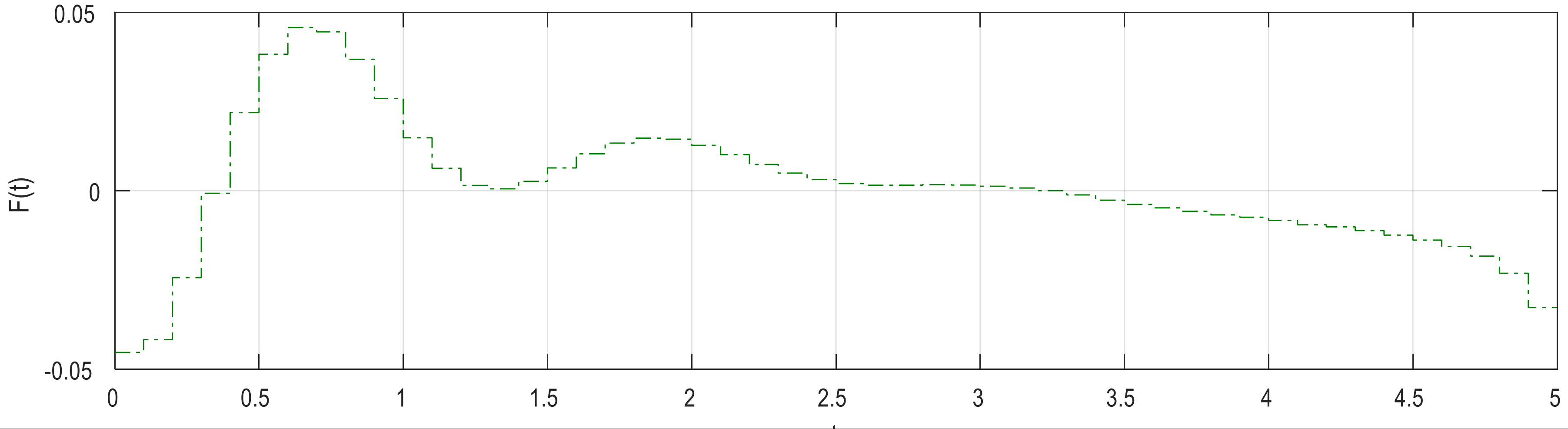
Inverted Pole on Cart



- State: (x, v, θ, ω)
  - Position, speed of cart, angle of pole, angular speed of pole
- Equations of motion:
$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F$$
$$(I+ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta$$
  - Parameters: M, m, l, I, b, g – mass of cart and pole, length and moment of inertial of pole, friction coefficient, acceleration due to gravity
  - Control: F – force of pushing
- Constraints:
  - Start from initial state $(0,0,0,0)$, reach final state $(0,0,\pi,0)$ at time T
  - Maximum force limit
- Cost: Control effort: $\int_0^T F^2(t)dt$

Inverted Pole on Cart





Shooting Method Disadvantages

- Numerical integration
  - Potentially slow
  - Numerical errors

- Open-loop solution