technische universität dortmund

EPFL

Optimal Control (EE-736)

Part II.2: Comments on Indirect Methods

Timm Faulwasser & Yuning Jiang

ie3, TU Dortmund

timm.faulwasser@ieee.org
yuning.jiang@epfl.ch

Block course @ EPFL

Version EE736.2024.I

Overview

Overview

How to Solve Optimal Control Problems?

Core challenge:

How to compute the infinite dimensional object $u(\cdot)$?

How to Solve Optimal Control Problems?

Core challenge:

How to compute the infinite dimensional object $u(\cdot)$?

Two main options:

How to Solve Optimal Control Problems?

Core challenge:

How to compute the infinite dimensional object $u(\cdot)$?

Two main options:

1. Discretize problem.
2. Optimize discretized problem.

$\rightarrow$ **Direct solution methods.**

How to Solve Optimal Control Problems?

Core challenge:

How to compute the infinite dimensional object $u(\cdot)$?

Two main options:

1. Discretize problem.	1. Optimize problem ($=$ get NCOs).
2. Optimize discretized problem.	2. Discretize to solve NCOs.
$\rightarrow$ **Direct solution methods.**	$\rightarrow$ **Indirect solution methods.**

Overview of Solution Approaches

Continuous Time Optimal Control

Overview of Solution Approaches

Continuous Time Optimal Control

Hamilton-Jacobi-Bellman-Equation

- Tabulation in state space
- Closed-loop optimal control

Indirect Methods

- Based on PMP (NCO of OCP)

Direct Solution Methods

- Convert OCP into NLP

Overview of Solution Approaches

Continuous Time Optimal Control

Hamilton-Jacobi-Bellman-Equation
- Tabulation in state space
- Closed-loop optimal control

Indirect Methods
- Based on PMP (NCO of OCP)

Direct Solution Methods
- Convert OCP into NLP

Direct Single Shooting
- Discretized states and controls in NLP
- Simultaneous approach

Direct Collocation
- Discretized states and controls in NLP (collocation)
- Simultaneous approach

Direct Multiple Shooting
- Discretized controls and node start values in NLP
- Simultaneous approach

Overview of Solution Approaches

Continuous Time Optimal Control

Hamilton-Jacobi-Bellman Equation

Not covered in course!

- Tabulation in state space
- Closed-loop optimal control

Indirect Methods

- Based on PMP (NCO of OCP)

Direct Solution Methods

- Convert OCP into NLP

Direct Single Shooting

- Discretized states and controls in NLP
- Simultaneous approach

Direct Collocation

- Discretized states and controls in NLP (collocation)
- Simultaneous approach

Direct Multiple Shooting

- Discretized controls and node start values in NLP
- Simultaneous approach

Overview

Introduction

Indirect Solution Approaches

Basic Idea of Indirect Shooting

$$\min_{u(\cdot)} \quad \int_{t_0}^{t_1} \ell(t, x(t), u(t))\mathrm{d}t$$

subject to: (P)

$$\begin{aligned} \dot{x} &= f(t, x, u), \quad x(t_0) = x_0 \\ u(\cdot) &\in \mathcal{C}[t_0, t_1]^{n_u} \\ \Psi(x(t_1)) &= 0 \end{aligned}$$

Basic Idea of Indirect Shooting

$$\min_{u(\cdot)} \quad \int_{t_0}^{t_1} \ell(t, x(t), u(t))\mathrm{d}t$$

subject to: (P)

$$\begin{aligned} \dot{x} &= f(t, x, u), \quad x(t_0) = x_0 \\ u(\cdot) &\in \mathcal{C}[t_0, t_1]^{n_u} \\ \Psi(x(t_1)) &= 0 \end{aligned}$$

NCOs:

$$\begin{aligned} \dot{x}^\star &= \quad H_\lambda(t, x^\star, u^\star, \lambda^\star), \quad x^\star(t_0) = x_0 \\ \dot{\lambda}^\star &= -H_x(t, x^\star, u^\star, \lambda^\star), \\ \lambda^\star(t_1) &= \quad \phi_x(x^\star(t_1)) + (\nu^\star)^\top \Psi_x(x^\star(t_1)), \\ 0 &= \quad H_u(t, x^\star, u^\star, \lambda^\star), \\ \Psi(x^\star(t_1)) &= 0 \end{aligned}$$

Basic Idea of Indirect Shooting

Formulate NCOs, then solve NCOs:

First optimize, then discretize.

Split NCOs into two parts:

- NCOs enforced at each iteration.
- NCOs modified at each iteration; i.e. enforced upon convergence.

Basic Idea of Indirect Shooting

Formulate NCOs, then solve NCOs:

First optimize, then discretize.

Split NCOs into two parts:

- NCOs enforced at each iteration.
- NCOs modified at each iteration; i.e. enforced upon convergence.

NCOs:

$$
\begin{aligned}
\dot{x}^\star &= \quad H_\lambda(t, x^\star, u^\star, \lambda^\star), \qquad x^\star(t_0) = x_0 \\
\dot{\lambda}^\star &= -H_x(t, x^\star, u^\star, \lambda^\star), \\
\lambda^\star(t_1) &= \quad \phi_x(x^\star(t_1)) + (\nu^\star)^\top \Psi_x(x^\star(t_1)), \\
0 &= \quad H_u(t, x^\star, u^\star, \lambda^\star), \\
\Psi(x^\star(t_1)) &= 0
\end{aligned}
$$

A Basic Indirect Shooting Algorithm

1. Choose $\varepsilon > 0$. Guess λ_0^0, ν^0. Set $k = 0$.
2. Integrate from t_0 to t_1

$$\begin{aligned} \dot{x}^k &= H_\lambda(t, x^k, u^k, \lambda^k), \quad & x(t_0)^k &= x_0 \\ \dot{\lambda}^k &= -H_x(t, x^k, u^k, \lambda^k), \quad & \lambda(t_0)^k &= \lambda_0^k \\ 0 &= H_u(t, x^k, u^k, \lambda^k). \end{aligned}$$

A Basic Indirect Shooting Algorithm

1. Choose $\varepsilon > 0$. Guess λ_0^0, ν^0. Set $k = 0$.
2. Integrate from t_0 to t_1

$$\begin{aligned}
\dot{x}^k &= \quad H_\lambda(t, x^k, u^k, \lambda^k), \quad x(t_0)^k = x_0 \\
\dot{\lambda}^k &= -H_x(t, x^k, u^k, \lambda^k), \quad \lambda(t_0)^k = \lambda_0^k \\
0 &= \quad H_u(t, x^k, u^k, \lambda^k).
\end{aligned}$$

3. Compute defect of transversality and terminal conditions:

$$\mathcal{F}(\lambda_0^k, \nu^k) \doteq \begin{bmatrix} \lambda(t_1)^k - \quad \phi_x(x(t_1)^k) - (\nu^k)^\top \Psi_x(x(t_1)^k) \\ \Psi(x(t_1)^k) \end{bmatrix}.$$

4. If $\|\mathcal{F}(\lambda_0^k, \nu^k)\| \leq \varepsilon \quad \rightarrow \quad$ STOP.

A Basic Indirect Shooting Algorithm

1. Choose $\varepsilon > 0$. Guess λ_0^0, ν^0. Set $k = 0$.
2. Integrate from t_0 to t_1

$$\begin{aligned} \dot{x}^k &= \quad H_\lambda(t, x^k, u^k, \lambda^k), \quad & x(t_0)^k = x_0 \\ \dot{\lambda}^k &= -H_x(t, x^k, u^k, \lambda^k), \quad & \lambda(t_0)^k = \lambda_0^k \\ 0 &= \quad H_u(t, x^k, u^k, \lambda^k). \end{aligned}$$

3. Compute defect of transversality and terminal conditions:

$$\mathcal{F}(\lambda_0^k, \nu^k) \doteq \begin{bmatrix} \lambda(t_1)^k - \quad \phi_x(x(t_1)^k) - (\nu^k)^\top \Psi_x(x(t_1)^k) \\ \Psi(x(t_1)^k) \end{bmatrix}.$$

4. If $\|\mathcal{F}(\lambda_0^k, \nu^k)\| \leq \varepsilon \quad \rightarrow \quad$ STOP.
5. Update λ_0^k, ν^k to enforce $\mathcal{F}(\lambda_0^k, \nu^k) \rightarrow 0$. $k \leftarrow k + 1$. GOTO 1.

How to do Step 5?

- Defect condition
$$\lim_{k\to\infty} \mathcal{F}(\lambda_0^k, \nu^k) = 0$$
is a root finding problem, i.e., we can apply Newton's method.

How to do Step 5?

- Defect condition

$$\lim_{k\to\infty} \mathcal{F}(\lambda_0^k, \nu^k) = 0$$

  is a root finding problem, i.e., we can apply Newton's method.

- Compute defect gradients $\nabla_{\lambda_0^k}\mathcal{F}$ and $\nabla_{\nu^k}\mathcal{F}$ and solve

$$\begin{bmatrix} \nabla_{\lambda_0^k}\mathcal{F} & \nabla_{\nu^k}\mathcal{F} \end{bmatrix} \begin{bmatrix} \delta\lambda^k \\ \delta\nu^k \end{bmatrix} = -\mathcal{F}(\lambda_0^k, \nu^k).$$

How to do Step 5?

- Defect condition
$$\lim_{k\to\infty} \mathcal{F}(\lambda_0^k, \nu^k) = 0$$
is a root finding problem, i.e., we can apply Newton's method.

- Compute defect gradients $\nabla_{\lambda_0^k}\mathcal{F}$ and $\nabla_{\nu^k}\mathcal{F}$ and solve
$$\begin{bmatrix} \nabla_{\lambda_0^k}\mathcal{F} & \nabla_{\nu^k}\mathcal{F} \end{bmatrix} \begin{bmatrix} \delta\lambda^k \\ \delta\nu^k \end{bmatrix} = -\mathcal{F}(\lambda_0^k, \nu^k).$$

- Update
$$\begin{bmatrix} \lambda_0^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \lambda_0^k \\ \nu^k \end{bmatrix} + \begin{bmatrix} \delta\lambda^k \\ \delta\nu^k \end{bmatrix}.$$

Example

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

subject to

$$\dot{x}(t) = u(t)(1 - x(t)), \quad x(0) = -1,\ x(1) = 0$$

- Formulate the NCOs.
- Formulate the defect.
- $\rightarrow$ MATLAB example IndirectSingleShooting.m

Indirect Gradient Methods

Recall the NCO of the OCP (P), whereby we drop the terminal constraint:

$$\begin{aligned} \dot{x}^\star &= \quad H_\lambda(t, x^\star, u^\star, \lambda^\star), \quad x^\star(t_0) = x_0 \\ \dot{\lambda}^\star &= -H_x(t, x^\star, u^\star, \lambda^\star), \\ \lambda^\star(t_1) &= \quad \phi_x(x^\star(t_1)), \\ 0 &= \quad H_u(t, x^\star, u^\star, \lambda^\star) \end{aligned}$$

Observe that

- The dynamics of x run forward in time from x_0 to $x(t_1)$.
- The adjoint dynamics run backward in time from $\lambda(t_1)$ to $\lambda(t_0)$.
- Main idea of indirect gradient methods: forward-backward sweep exploiting the structure of the NCOs with gradient updates.

Indirect Gradient Methods

1. Choose $\varepsilon > 0$. Guess u^0 (appropriately parametrized). Set $k = 0$.
2. Integrate forward from t_0 to t_1

$$\dot{x}^k = \quad H_\lambda(t, x^k, u^k, \lambda^k) = f(t, x^k, u^k), \quad x^\star(t_0) = x_0$$

3. Integrate backward from t_1 to t_0

$$\dot{\lambda}^k = -H_x(t, x^k, u^k, \lambda^k), \quad \lambda^k(t_1) = \quad \phi_x(x^k(t_1))$$

4. Update the controls

$$u^{k+1} = u^k + \alpha\delta u^k, \quad \alpha \in (0, 1]$$

   whereby

$$\delta u^k \doteq H_u(\lambda^k, x^k, u^k)$$

5. IF $\|\delta u^k\| < \varepsilon \quad \rightarrow \quad$ STOP.
6. $k \leftarrow k + 1$. GOTO 1.

Why is this gradient method?

Recall that the computation of the first variation (the Gateaux derivative) of objective functional

$$\delta J(u^\star)$$

lead to the NCO (Euler-Lagrange equations), cf. Slide II.48.
As the forward-backward sweep satisfies the dynamics, the adjoint dynamics and the transversality condition, we have that

$$\delta u^k \doteq H_u(\lambda^k, x^k, u^k) = \delta J(u^k),$$

i.e., $u^k + \alpha\delta u^k$ is a gradient step.

How to make it work?

Constraints?

- Input trajectory needs to be parametrized, e.g. piece-wise constant.
- Input constraints via projection onto the feasible set after gradient step.
- State constraints $\rightarrow$ can be included via penalty or barrier functions
- Initial guess of the input required, backward integration of the adjoints often unstable
- One toolbox which uses this (in more elaborated form):
  T. Englert, A. Völz, F. Mesmer, S. Rhein, K. Graichen. “A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)”. **in** *Optimization and Engineering*: 20.3 (2019), **pages** 769–809

How to make it work?

Constraints?

- Input trajectory needs to be parametrized, e.g. piece-wise constant.
- Input constraints via projection onto the feasible set after gradient step.
- State constraints $\rightarrow$ can be included via penalty or barrier functions
- Initial guess of the input required, backward integration of the adjoints often unstable
- One toolbox which uses this (in more elaborated form):
  T. Englert, A. Völz, F. Mesmer, S. Rhein, K. Graichen. "A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)". **in** *Optimization and Engineering*: 20.3 (2019), **pages** 769–809

Good to know

The backward step, can be understood as a *back propagation* to compute the objective gradient.
This is widely used for deep learning as one formalize the training of deep neural networks as an optimal control problem.

Remarks on Indirect Methods

- Require formulation of the NCOs.
- Input constraints can be considered (not covered here).
- In case of active state-path constraints $\rightarrow$ much more complicated NCOs $\rightarrow$ indirect methods become quite tedious.
- Instable dynamics $\dot{x} = f(x, u)$ lead to numerical issues (good guesses required).
- Indirect methods can be very precise (decisive element: accuracy of integration of dynamics).
- Indirect methods can be extremely memory efficient (no need to store large matrices).