Optimal Control 2018

Kaoru Yamamoto

Optimal Control 2018

L1: Functional minimization, Calculus of variations (CV) problem

L2: Constrained CV problems, From CV to optimal control

L3: Maximum principle, Existence of optimal control

L4: Maximum principle (proof)

L5: Dynamic programming, Hamilton-Jacobi-Bellman equation

L6: Linear quadratic regulator

L7: **Numerical methods for optimal control problems**

Exercise sessions (20%):
Solve 50% of problems in advance. Hand-in later.
Mini-project (20%):
Study and present your own optimal control problem. Apr 5 (Thu)?
Written take-home exam (60%).

Optimal control problem

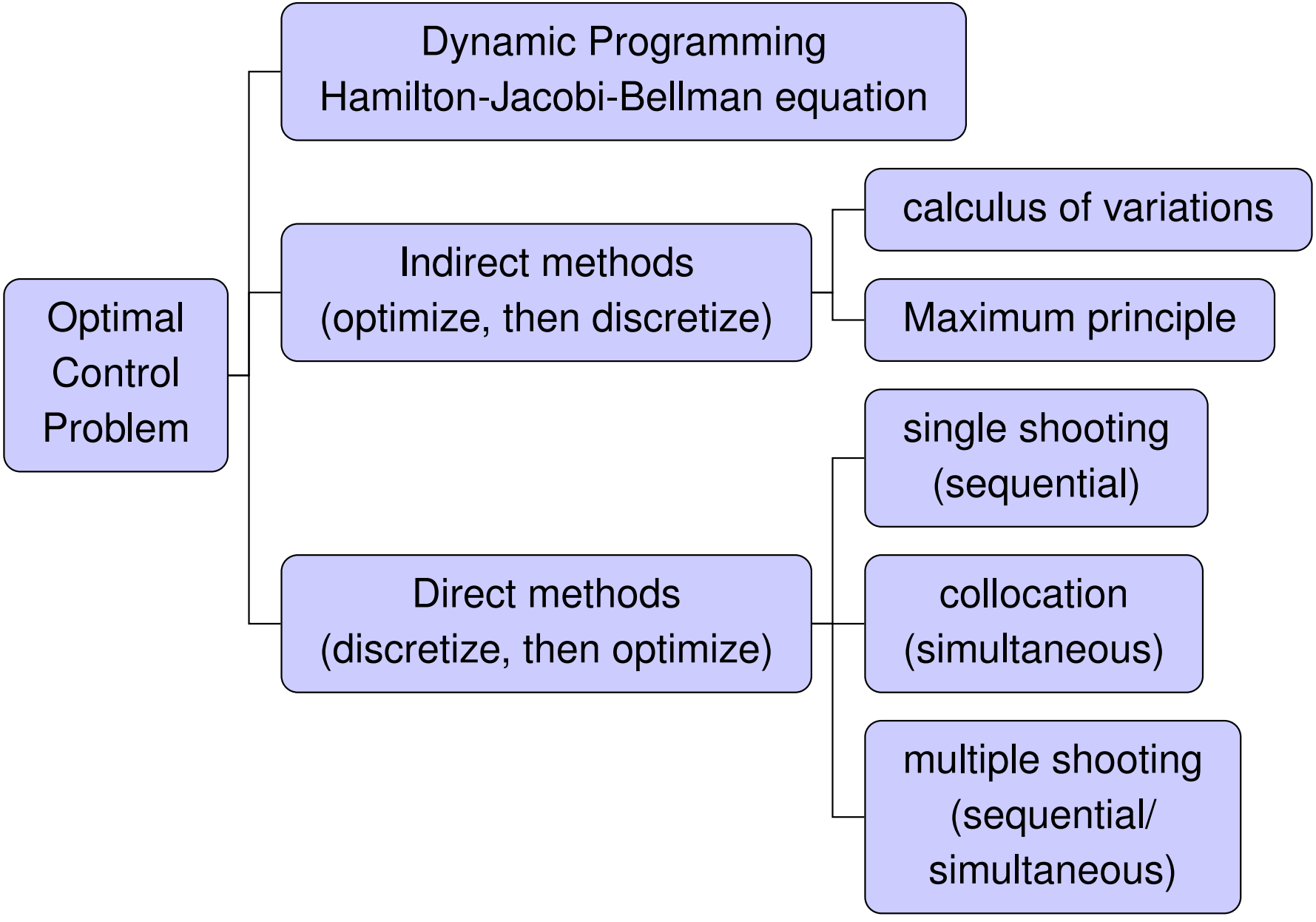
Find a control $u \in U \subset \mathbb{R}^m$ that minimizes the cost

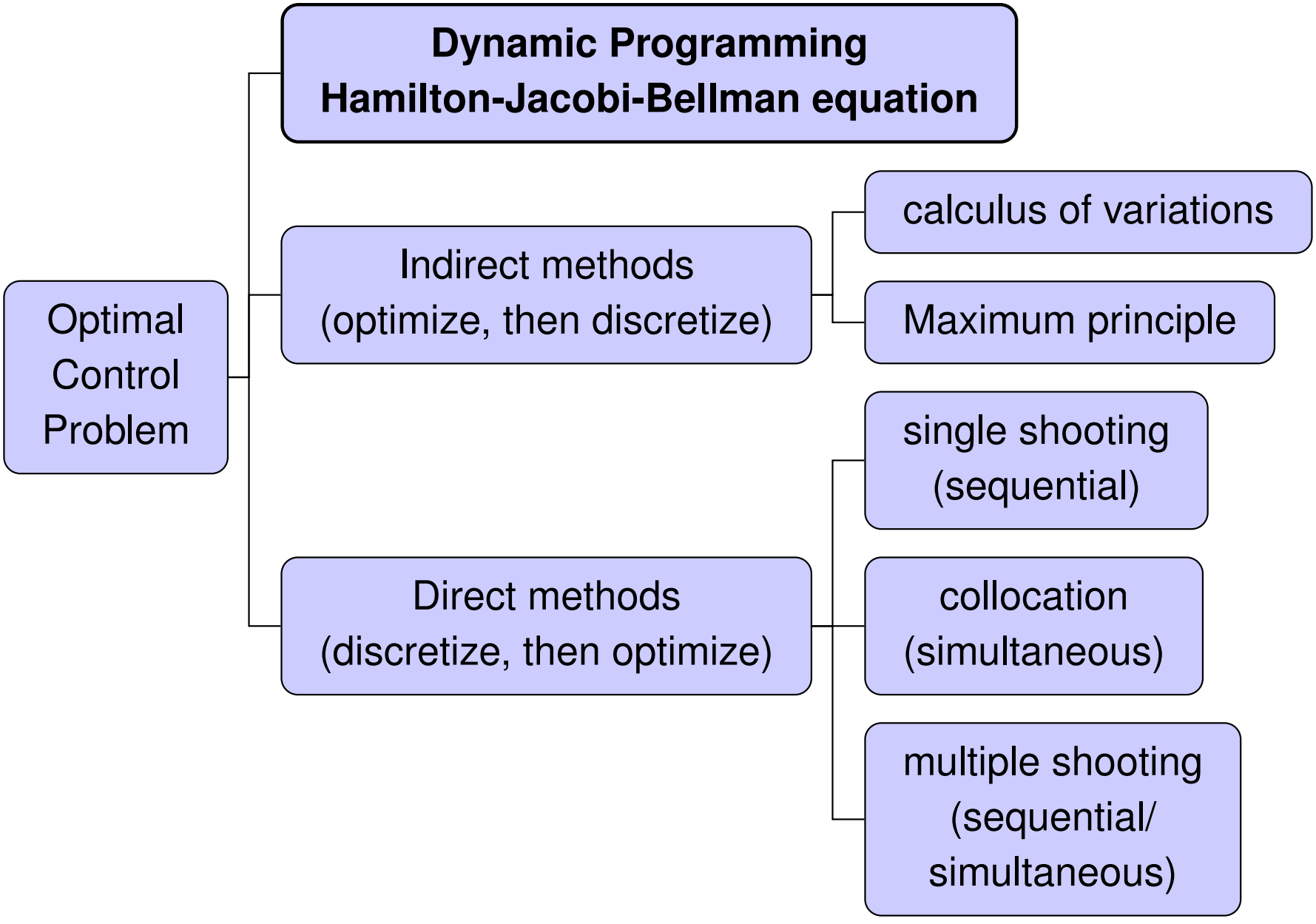
$$J(u) := \int_{t_0}^{t_f} \underbrace{L(t, x(t), u(t))}_{\text{running cost}}\, dt + \underbrace{K(t_f, x_f)}_{\text{terminal cost}}$$

subject to

$$\dot{x} = f(t, x, u),\ x(t_0) = x_0,\ x \in \mathbb{R}^n.$$

- Infinite-dimensional optimization problem. Discretization needed.
- When and how?


Optimal Control Problem
Dynamic Programming Hamilton-Jacobi-Bellman equation
Indirect methods (optimize, then discretize)
calculus of variations
Maximum principle
Direct methods (discretize, then optimize)
single shooting (sequential)
collocation (simultaneous)
multiple shooting (sequential/ simultaneous)


Optimal Control Problem
Dynamic Programming
Hamilton-Jacobi-Bellman equation
Indirect methods
(optimize, then discretize)
calculus of variations
Maximum principle
Direct methods
(discretize, then optimize)
single shooting
(sequential)
collocation
(simultaneous)
multiple shooting
(sequential/
simultaneous)

Hamilton-Jacobi-Bellman (HJB) equation

- Value function (optimal cost-to-go): $V(t,x) = \inf_{u_{[t,t_1]}} J(t,x,u)$.
- HJB equation:

$$-V_t(t,x) = \inf_{u \in U} \{L(t,x,u) + \langle V_x(t,x), f(t,x,u) \rangle\}.$$

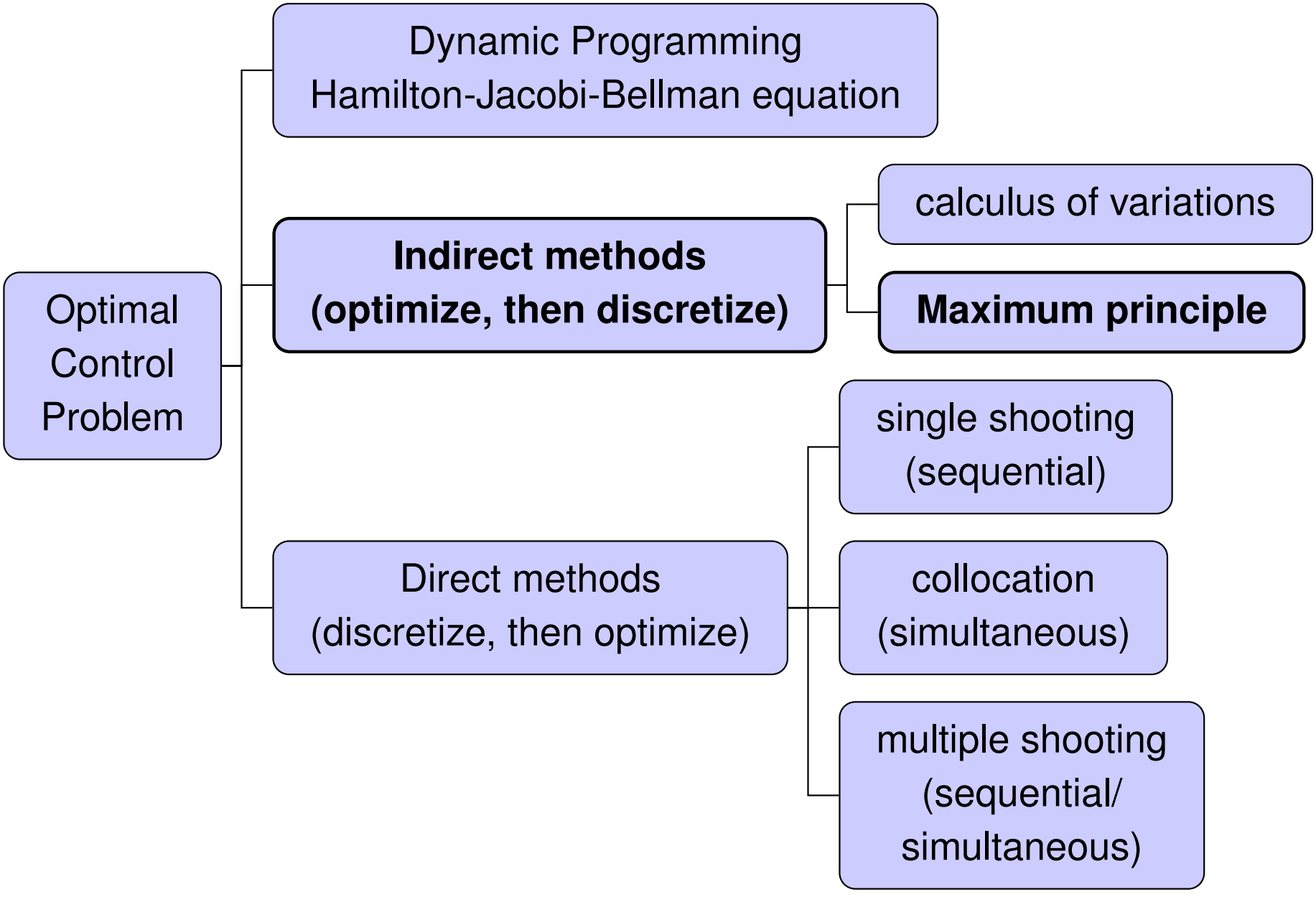
- Solve this partial differential equation (PDE) backwards for $t \in [t_0, t_1]$, starting from

$$V(t_1,x) = K(x).$$

(Often high dimensional PDE, intractable!)

- Optimal controls

$$u^*(x,t) = \operatorname{argmin}_{u \in U} \{L(t,x,u) + \langle V_x(t,x), f(t,x,u) \rangle\}.$$


Optimal Control Problem
Dynamic Programming Hamilton-Jacobi-Bellman equation
Indirect methods (optimize, then discretize)
calculus of variations
Maximum principle
Direct methods (discretize, then optimize)
single shooting (sequential)
collocation (simultaneous)
multiple shooting (sequential/ simultaneous)

Necessary conditions for optimality

Recall the maximum principle:

The Hamiltonian: $H(x, u, p, p_0) = \langle p, f(x, u) \rangle + p_0 L(x, u)$.

Assume that the basic problem has a solution $(u^*(t), x^*(t))$, $u^*(t) \in U$, $x^*(t) \in \mathbb{R}^n$. Then $\exists p^* : [t_0, t_f] \to \mathbb{R}^n$, $\exists p_0^* \leq 0$ satisfying $(p_0^*, p^*(t)) \neq (0, 0)$ $\forall t \in [t_0, t_f]$ and

1) $\dot{x}^* = H_p(t, x^*, u^*, p^*)$, $\dot{p}^* = -H_x(t, x^*, u^*, p^*)$,
$x^*(t_0) = x_0, x^*(t_f) \in S_1$.

2) $H(x^*(t), u^*(t), p^*(t), p_0^*) \geq H(x^*(t), u(t), p^*(t), p_0^*)$
$\forall t \in [t_0, t_f]$, $\forall u \in U$.

3) $H(x^*(t), u^*(t), p^*(t), p_0^*) = 0 \quad \forall t \in [t_0, t_f]$

4) $\langle p^*(t_f), d \rangle = 0 \quad \forall d \in T_{x^*(t_f)} S_1 \quad T_{x^*(t_f)} S_1$: tangent space to S_1.

Indirect methods

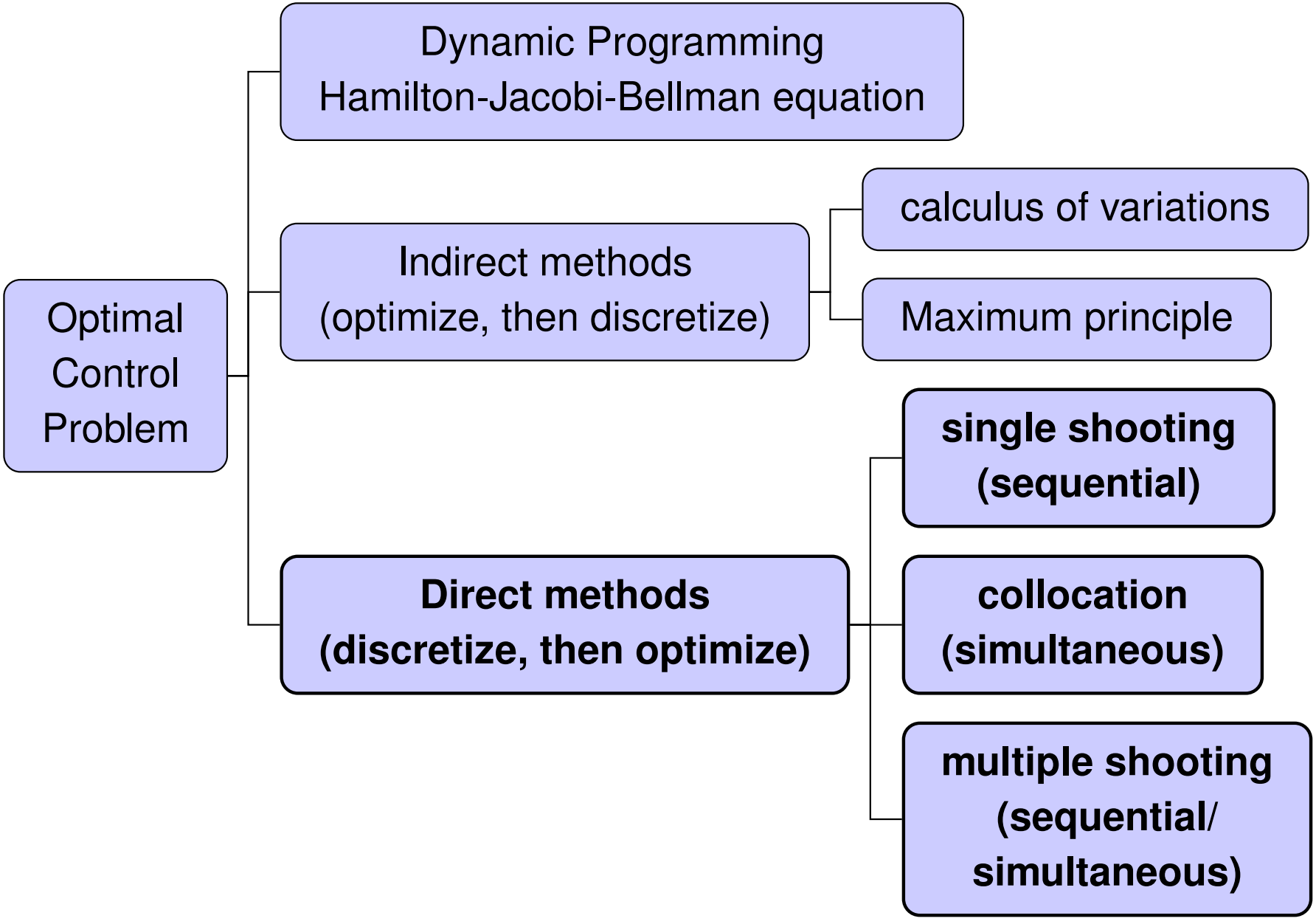
1) $\dot{x}^* = H_p(t, x^*, u^*, p^*)$, $\dot{p}^* = -H_x(t, x^*, u^*, p^*)$,
$x^*(t_0) = x_0$, $x^*(t_f) \in S_1$.

4) $\langle p^*(t_f), d \rangle = 0 \quad \forall d \in T_{x^*(t_f)} S_1 \quad T_{x^*(t_f)} S_1$: tangent space to S_1.

- n boundary conditions imposed on (x^*, p^*) at $t = t_0$ and n more at $t = t_f$. $\Rightarrow$ two point boundary value problem with $2n$ ordinary differential equations (ODEs).
- Can solve with gradient methods, shooting methods, or collocation (e.g., MATLAB `bvp4c`).

Sounds good, but...

- Good initial guess needed. How to guess $p^*(t_0)$?
- Path constraints (discontinuity in state/adjoint equations or control functions) difficult to handle.


Optimal Control Problem
Dynamic Programming Hamilton-Jacobi-Bellman equation
Indirect methods (optimize, then discretize)
calculus of variations
Maximum principle
Direct methods (discretize, then optimize)
single shooting (sequential)
collocation (simultaneous)
multiple shooting (sequential/ simultaneous)

Direct methods

- First discretize, then optimize.
- Approximate the original infinite-dimensional problem by a finite-dimensional, nonlinear programming problem (NLP).

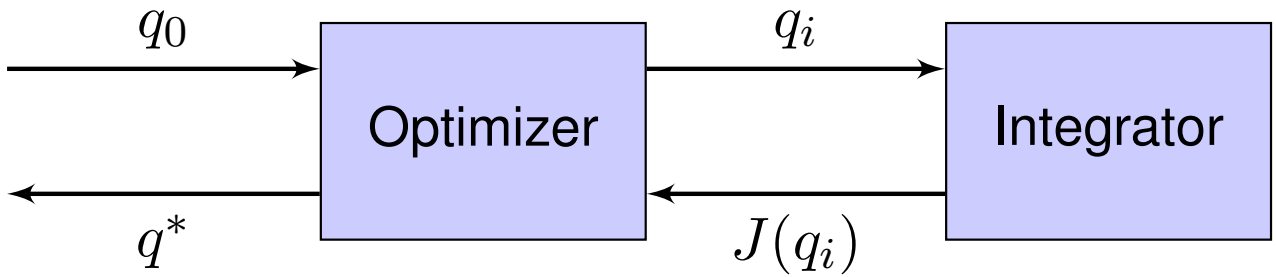
Two main approaches:

- **Sequential methods:** discretize controls, simulate dynamic system using embedded numerical integrator, and update control iteratively based on sensitivities.
  - direct single shooting.
- **Simultaneous methods:** discretize all system variables.
  - direct collocation, (direct multiple shooting)

Direct single shooting

- **Control parametrization:** Parametrize control $u(t)$ using a finite number of parameters by, e.g., piecewise polynomials q, i.e., q determines u (approximately), and determines x through ODEs.
- **Parameter optimization problem:**

$$\min_u J(u) \text{ s.t. } \dot{x} = f(x,u), x(t_0) = x_0$$
$$\Downarrow$$
$$\min_q J(q) \text{ s.t. } \dot{x} = f(x,q), x(t_0) = x_0$$



+ very simple to implement, manageable size of NLP
− poor convergence without gradients, cannot handle unstable systems, state constraints difficult
- sometimes the only feasible way to go

Direct multiple shooting

- **Control parametrization:** Divide the time horizon into N intervals $t_0 = t_0 < t_1 < \cdots < t_N = t_f$. Parametrize control by piecewise constants; $\tilde{u}(t) = q_i,\ t \in [t_i, t_{i+1}),\ i = 0, \ldots, N-1$.
- **State discretization:** Introduce $N+1$ vectors $s_0, s_1, \ldots, s_N$ and solve the decoupled state equations

$$\dot{x}_i(t) = f(x_i(t), q_i),\ t \in [t_i, t_{i+1}]$$

with $x_i(t_i) = s_i$ (initial conditions) and
$s_{i+1} = x_i(t_{i+1})$ (continuity constraint).

- **Parameter optimization problem:**

$$\min_{\bar{s},\bar{q}} J(\bar{q})$$

$$\text{s.t. } C_{\text{ineq}}(s_i, q_i) \leq 0,\ C_f(s_N) = 0.$$

+ path constraints approximated by point constraints, memory efficient

± large NLP, but sparse

Direct collocation

- Discretize the state and control at a set of suitably chosen points in the time horizon $[t_0, t_f] \Rightarrow$ NLP
- The resulting NLP is then solved using well-known solvers, e.g., IPOPT, SNOPT, KNITRO.
- How to approximate the state, given $\dot{x} = f(x, u), x(t_0) = x_0$?
  - h methods
  - p methods
  - hp methods
- In the end, we need an efficient way to approximate
  1. the integration in the cost function $\int_{t_0}^{t_f} L dt$,
  2. the differential equation $\dot{x} = f(x, u), x(t_0) = x_0$, and
  3. constraints on the state and control.

  – pseudospectral methods

State approximation – h method

- Divide the time horizon $[t_0, t_f]$ into K mesh intervals.
- The state is approximated using the same fixed-degree (often low-order) polynomial in each mesh interval.
- Finer mesh $\rightarrow$ more accurate approximation
- e.g., Euler methods, Runge-Kutta methods

Example:

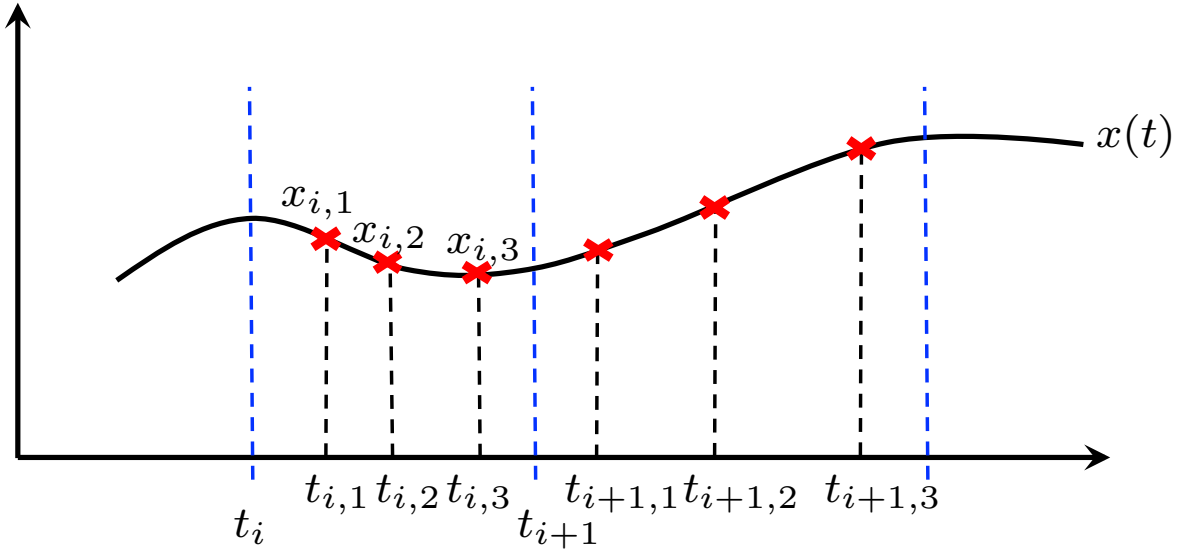
$$\dot{x} \approx \frac{x_{k+1} - x_k}{h}, \quad h := (t_f - t_0)/K$$
$$\Rightarrow x_{k+1} = x_k + hf(x_k, u_k)$$
$$\Rightarrow C(\bar{x}, \bar{u}) = 0$$

where

$$C(\bar{x}, \bar{u}) = \begin{pmatrix} x_1 - (x_0 + hf(x_0, u_0)) \\ x_2 - (x_1 + hf(x_1, u_1)) \\ \vdots \\ x_K - (x_{K-1} + hf(x_{K-1}, u_{K-1})) \end{pmatrix}$$

State approximation – p method

- Divide the time horizon $[t_0, t_f]$ into K (fixed) mesh intervals.
- Approximate the state in each interval by an Nth order polynomial
- Higher-order polynomial $\rightarrow$ more accurate approximation



$$x(t) = \sum_{j=1}^{N+1} x_{i,j} P_j(t), \; t \in [t_i, t_{i+1}], \; P_j \text{ : interpolation polynomials.}$$

State approximation – hp method

- Hybrid between an h method and a p method.
- Both the number of mesh K and the degree of the approximating polynomial N_K within each mesh interval can change.

Pseudospectral methods

- An efficient way to approximate
  1. the integration in the cost function $\int_{t_0}^{t_f} L dt$,
  2. the differential equation $\dot{x} = f(x, u), x(t_0) = x_0$, and
  3. constraints on the state and control.
- The state and control parametrized by a polynomial approximation, e.g., Lagrange polynomial approximation.
- The cost function approximated by numerical quadrature, e.g., Gaussian quadrature.

Software packages:

- $\mathbb{GPOPS} - \mathbb{II}$ (MATLAB)
- PROPT (MATLAB)
- $\mathcal{PSOPT}$ (C++)
- **NLOptControl.jl (Julia)**

NLP solvers

- IPOPT (Interior Point OPTimizer) iteratively solves NLPs using primal-dual interior-point method with filter-based line search
- A decent initial guess of the solution is important to find a decent local optimum
- Usually better to guess an optimal input, and then simulate the system using this input
- Or solve a simpler but related optimization problem and use that as a guess

NLOptControl.jl

- hp-pseudospectral method written in julia
- https://juliampc.github.io/MPCDocs.jl/v0.1.3/ for the installation guide and some tutorials
- **Ipopt.jl** or **KNITRO.jl** for the NLP solver
- **JuMP.jl** (Julia for Mathematical Optimization) and **DifferentialEquations.jl** as a part of the software.

Example: time optimal control of double integrator

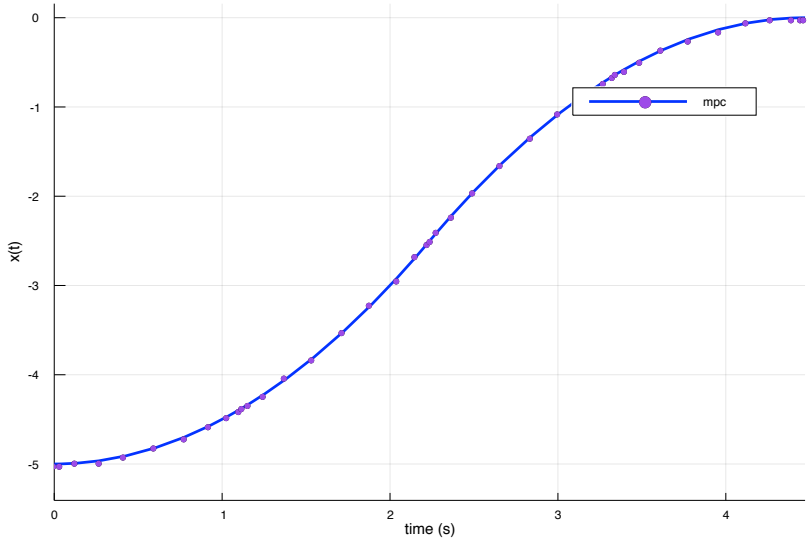
Problem: minimize t_f

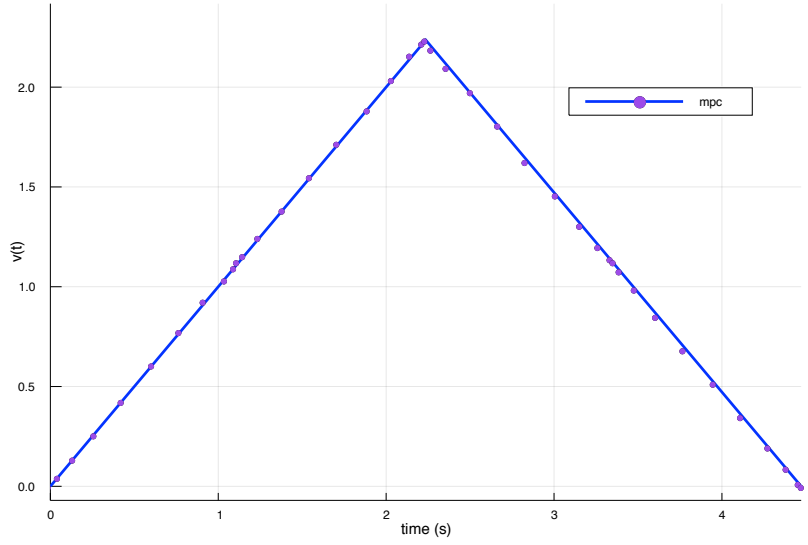
subject to $\dot{x}_1 = x_2$, $\dot{x}_2 = u$, $u \in [-1, 1]$,

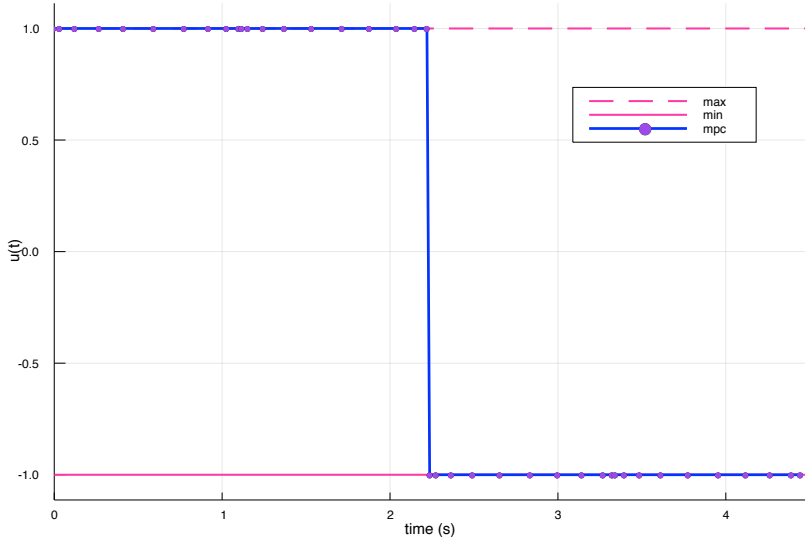
$x_1(0) = -5$, $x_2(0) = 0$, $x_1(t_f) = 0$, $x_2(t_f) = 0$

```
1  using NLOptControl, PrettyPlots
2
3  n=define(numStates=2,numControls=1,X0=[-5,0],XF=[0,0],CL=[-1],CU=[1]);
4
5  states!(n,[:x,:v];descriptions=["x(t)","v(t)"]);
6  controls!(n,[:u];descriptions=["u(t)"]);
7
8  dx=[:(v[j]),:(u[j])]
9  dynamics!(n,dx)
10
11 configure!(n;(:finalTimeDV=>true));
12
13 @NLobjective(n.mdl, Min, n.tf);
14
15 optimize!(n);
16
17 allPlots(n)
```

Result


0
-1
-2
-3
-4
-5
x(t)
0
1
2
3
4
time (s)
mpc


2.0
1.5
1.0
0.5
0.0
v(t)
0
1
2
3
4
time (s)
mpc


1.0
0.5
0.0
-0.5
-1.0
u(t)
0
1
2
3
4
time (s)
max
min
mpc

Example: minimum time-fuel control of double integrator

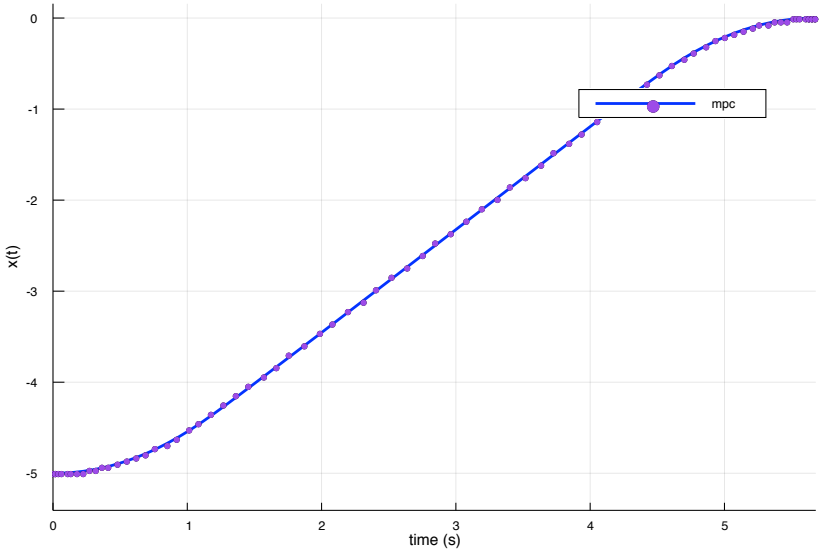
Problem: minimize $\int_0^{t_f} (1 + |u(t)|)dt$

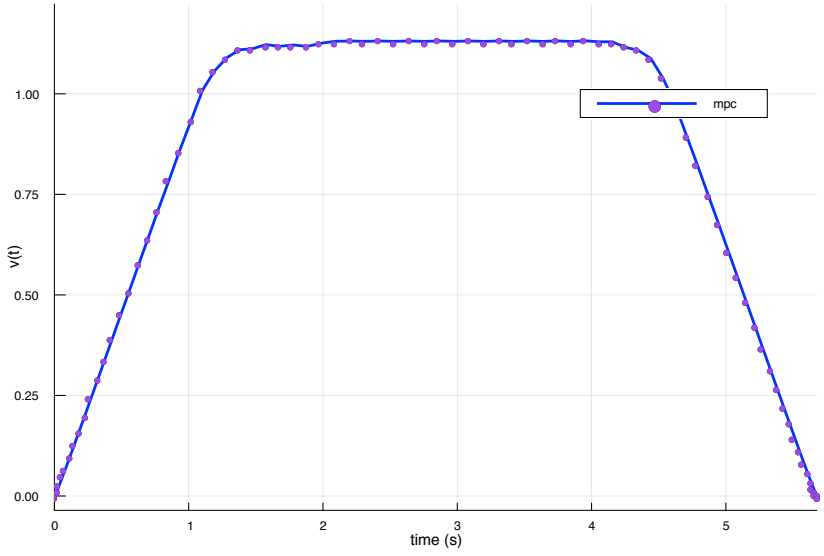
subject to $\dot{x}_1 = x_2$, $\dot{x}_2 = u$, $u \in [-1, 1]$,

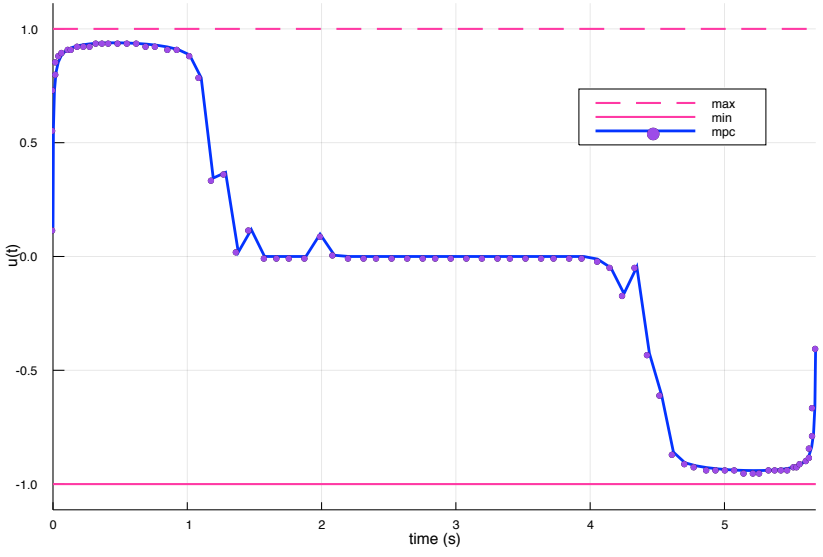
$x_1(0) = -5$, $x_2(0) = 0$, $x_1(t_f) = 0$, $x_2(t_f) = 0$

```
1  using NLOptControl, PrettyPlots
2
3  n=define(numStates=2,numControls=1,X0=[-5,0],XF=[0,0],CL=[-1],CU=[1]);
4
5  states!(n,[:x,:v];descriptions=["x(t)","v(t)"]);
6  controls!(n,[:u];descriptions=["u(t)"]);
7
8  dx=[:(v[j]),:(u[j])]
9  dynamics!(n,dx)
10
11 configure!(n;(:Nck=>[80]),(:finalTimeDV=>true));
12
13 obj=integrate!(n,:(1+abs(u[j])));
14
15 @NLobjective(n.mdl, Min, obj);
16
17 optimize!(n);
18
19 allPlots(n)
```

Result







Possible project (suggested by Bo)

Swing-up of two pendulum on a moving cart. See the handout.

- The simplified model (6)–(8) can be used.
- Suggestion for the parameters: $l_1 = 0.5\,\text{m}, l_2 = 0.1\,\text{m}$ and $g = 10\,\text{m/s}^2$.
- Restriction on u: $|u| \leq u_{\max} = 20\,\text{m/s}^2$.
- Restriction on x: $|x| \leq x_{\max} \in [0.1, 1]\,\text{m}$.
- One can study the cost function with free final time

$$J(u) = \int_0^{t_f} (1 + \alpha u^2) dt$$

  Notice that $\alpha = 0$ gives the minimum time criterion, and $\alpha > 0$ should give a smoother swing-up.
- How does the optimal swing-up strategies change when $x_{\max}$ changes?

References

1. Johan Åkesson. “Numerical Methods for Dynamic Optimization” (Lecture course at LTH, Fall 2009) http://www.control.lth.se/user/jakesson/DynamicOptimization2009/
2. Fredrik Magnusson. “Using the Numerical Methods in JModelica.org for Optimal Control Problems” (Lecture 7 in PhD course “Optimal Control” at LTH, 2014) http://www.control.lth.se/Education/DoctorateProgram/optimal-control-2018/optimal-control-2014.html
3. Michael A. Patterson and Anil V. Rao. 2014. “GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming.” ACM Trans. Math. Softw. 41, 1, Article 1.