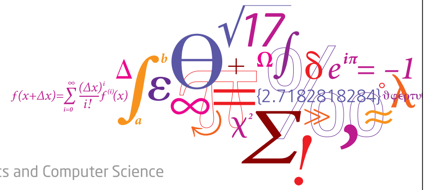
02465: Introduction to reinforcement learning and control

Direct methods and control by optimization

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)



DTU Compute
Department of Applied Mathematics and Computer Science

Lecture Schedule

Dynamical programming

1. The finite-horizon decision problem
   31 January
2. Dynamical Programming
   7 February
3. DP reformulations and introduction to Control
   14 February

Control

4. Discretization and PID control
   21 February
5. **Direct methods and control by optimization**
   28 February
6. Linear-quadratic problems in control
   7 March
7. Linearization and iterative LQR
   14 March

Reinforcement learning

8. Exploration and Bandits
   21 March
9. Policy and value iteration
   4 April
10. Monte-carlo methods and TD learning
    11 April
11. Model-Free Control with tabular and linear methods
    18 April
12. Eligibility traces and value-function approximations
    25 April
13. Q-learning and deep-Q learning
    2 May

Syllabus: `https://02465material.pages.compute.dtu.dk/02465public`
Help improve lecture by giving feedback on DTU learn

Reading material:

- [Her24, Chapter 15]

Learning Objectives

- Direct methods for optimal control
- Trajectory planning for linear-quadratic problems using optimization
- Trajectory planning using trapezoidal collocation

Project part 1

- Great job! Part 2 is online
- Survey on course experience on DTU Learn
- Thanks to the student who caught a problem with problem 1 for this weeks exercises; please point out all potential mistakes!

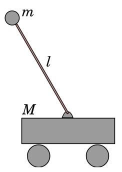
Dynamics

Dynamics of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$$

- $\boldsymbol{x}(t) \in \mathbb{R}^n$ is a complete description of the system at t
- $\boldsymbol{u}(t) \in \mathbb{R}^d$ are the controls applied to the system at t
- The time t belongs to an interval $[t_0, t_F]$ of interest

Example: Cartpole



- Coordinates are $\boldsymbol{x} = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}$ (angle, angular velocity, cart position, cart velocity)
- Action u is one-dimensional; the force applied to cart
- Dynamics are
$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$$
where $\boldsymbol{f}$ is a fairly complicated function

Constraints

$$\text{Equality constraint:} \quad x = c \tag{1}$$
$$\text{Inequality constraint:} \quad a \leq x \leq b \tag{2}$$

Any realistic physical system has constraints

- Simple boundary constraints

$$\boldsymbol{x}_{\text{low}} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{\text{upp}}$$
$$\boldsymbol{u}_{\text{low}} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{\text{upp}}$$

- End-point constraints:

$$\begin{aligned} \boldsymbol{x}_{0,\text{ low}} &\leq \boldsymbol{x}(t_0) \leq \boldsymbol{x}_{0,\text{ upp}} \\ \boldsymbol{x}_{F,\text{ low}} &\leq \boldsymbol{x}(t_F) \leq \boldsymbol{x}_{F,\text{upp}}. \end{aligned} \tag{3}$$

- Time constraints

$$\begin{aligned} t_{0,\text{ low}} &\leq t_0 \leq t_{0,\text{ upp}} \\ t_{F,\text{ low}} &\leq t_F \leq t_{F,\text{upp}}. \end{aligned} \tag{4}$$

Cost and policy

- The cost function is of the form

$$J_{\boldsymbol{u}}(\boldsymbol{x}, t_0, t_F) = \underbrace{c_F\left(t_0, t_F, \boldsymbol{x}(t_0), \boldsymbol{x}(t_F)\right)}_{\text{Mayer Term}} + \underbrace{\int_{t_0}^{t_F} c(\tau, \boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) d\tau}_{\text{Lagrange Term}}$$

Cartpole

- Necessary constraint $-u_{\max} < u(t) < u_{\max}$ and $\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & \pi & 0 \end{bmatrix}$
- Goal is to bring $\boldsymbol{x}$ to $\boldsymbol{x}^g = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$
- Up-right cartpole, version 1:
  - $$J_u(t_0, t_F, \boldsymbol{x}) = \|\boldsymbol{x}(t_F) - \boldsymbol{x}^g\|^2 + \lambda \int_{t_0}^{t_F} \boldsymbol{u}(t)^\top \boldsymbol{u}(t)$$
  - Constraints $t_0 = 0, t_F = 3$ (complete in 3 seconds)
- Up-right cartpole, version 2:
  - $$J_u(t_0, t_F, \boldsymbol{x}) = t_F - t_0$$
  - Constraints $\boldsymbol{x}_F = \boldsymbol{x}^g$

Endless combinations; depends on goal + method you are using

The continuous-time control problem

Given system dynamics for a system

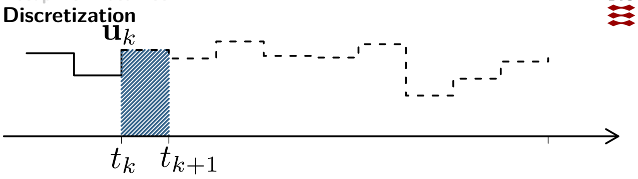
$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$$

Obtain $\boldsymbol{u} : [t_0; t_F] \to \mathbb{R}^m$ as solution to

$$\boldsymbol{u}^*, \boldsymbol{x}^*, t_0^*, t_F^* = \arg\min_{\boldsymbol{x}, \boldsymbol{u}, t_0, t_F} J_{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{u}, t_0, t_F).$$

(Minimization subject to all constraints)

Discretization



- Simplest choice: Eulers method
- Choose grid size N: $t_0, t_1, \ldots, t_N = t_F$, $t_{k+1} - t_k = \Delta$
- $\boldsymbol{x}_k = \boldsymbol{x}(t_k), \boldsymbol{u}_k = \boldsymbol{u}(t_k)$

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ &= \boldsymbol{x}_k + \Delta \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k) \end{aligned}$$

$$J_{\boldsymbol{u}=(\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1})}(\boldsymbol{x}_0) = c_f(t_0, \boldsymbol{x}_0, t_F, \boldsymbol{x}_F) + \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \Delta c(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k)$$

- Simple but not very exact

Approaches to control

- Last week: Rule-based methods (build $\boldsymbol{u}(t) = \pi(\boldsymbol{x}, t)$ directly)
- **Today: Optimization-based methods:**

$$\boldsymbol{u}^* = \arg\min_{\boldsymbol{u}} J_{\boldsymbol{u}}(\boldsymbol{x}_0)$$

  - Direct optimization of a discretized version of the problem
- Next week: DP-inspired planning methods

Infrastructure: Nonlinear program

A non-linear program is an optimization task of the form

$$\min_{z\in\mathbb{R}^n} E(\boldsymbol{z}) \quad \text{subject to}$$
$$\boldsymbol{h}(\boldsymbol{z}) = 0$$
$$\boldsymbol{g}(\boldsymbol{z}) \leq 0$$
$$\boldsymbol{z}_{\text{low}} \leq \boldsymbol{z} \leq \boldsymbol{z}_{\text{upp}}$$

i.e. the objective is to find the $\mathbf{z}$ that minimizes E under the constraints.

- If problem is not too complex, can use methods such as **sequential convex programming** to find $\mathbf{z}^*$.
- Requires luck and engineering
  - Needs a good initial guess
  - Improves when given gradient of J and Jacobian of $\boldsymbol{f}$ and $\boldsymbol{h}$.

Infrastructure: Linear Quadratic program

A special case of the optimization task:

$$\min \frac{1}{2}\boldsymbol{x}^T Q\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to}$$
$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$
$$F\boldsymbol{x} = \boldsymbol{g}$$

- When Q is positive definite and the problem is not very large the solution can always be found

Optimizing the Discrete Problem: Shooting

Consider the simplest form of a discrete control problem

$$\boldsymbol{x}_{k+1} = A_k\boldsymbol{x}_k + B_k\boldsymbol{u}_k + \boldsymbol{d}_k$$

quadratic cost function

$$\boldsymbol{J}_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}}(\boldsymbol{x}_0) = \boldsymbol{x}_N^T Q_N\boldsymbol{x}_N + \sum_{k=0}^{N-1}(\boldsymbol{x}_k^T Q_k\boldsymbol{x}_k + \boldsymbol{u}_k^T R_k\boldsymbol{u}_k)$$

- Given $\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}$, all the $\boldsymbol{x}_k$'s can be found form the system dynamics:

$$\boldsymbol{x}_2 = A_1\boldsymbol{x}_1 + B_1\boldsymbol{u}_1 + d_1 = A_1(A_0\boldsymbol{x}_0 + B_0\boldsymbol{u}_0 + \boldsymbol{d}_0) + B_1\boldsymbol{u}_1 + d_1$$

- Problem equivalent to optimizing $J_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}}(\boldsymbol{x}_0)$ (which is quadratic) wrt. $\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}$
- This method is called **shooting**
- **+ A single linear-quadratic optimization problem**
- **+ Easy to understand**

Optimizing the Discrete Problem: Shooting

- General case

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$$
$$J_{\boldsymbol{u}=(\boldsymbol{u}_0,\boldsymbol{u}_1,\dots,\boldsymbol{u}_{N-1})}(\boldsymbol{x}_0) = c_f(t_0, \boldsymbol{x}_0, t_F, \boldsymbol{x}_F) + \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

- Get rid of all the $\boldsymbol{x}_k$'s except $\boldsymbol{x}_0$:

$$\boldsymbol{x}_2 = \boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{u}_1) = \boldsymbol{f}(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0), \boldsymbol{u}_1)$$

So just optimize $J_{\boldsymbol{u}=(\boldsymbol{u}_0,\boldsymbol{u}_1,\dots,\boldsymbol{u}_{N-1})}(\boldsymbol{x}_0)$ wrt. $\boldsymbol{u}$

- **+ Easy to understand**
- A big, non-linear program (we cannot avoid that for general dynamics)
- **- Unstable: small changes in u_0 can mean big changes in x_N**
- **- Eulers method is imprecise**
- **- No bueno**. To overcome these issues, we have to take a step back

The continuous-time control problem

Given system dynamics for a system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{5}$$

Step 1: Must evaluate this ODE somehow

Subject to a number of dynamical and constant path and end-point constraints, obtain $\boldsymbol{u} : [t_0; t_F] \to \mathbb{R}^m$ as solution to

Step 2: Computing this integral

$$\min_{t_0, t_F, \boldsymbol{x}(t), \boldsymbol{u}(t)} \underbrace{c_F(t_0, t_F, \boldsymbol{x}(t_0), \boldsymbol{x}(t_F))}_{\text{Mayer Term}} + \underbrace{\int_{t_0}^{t_F} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau)d\tau}_{\text{Lagrange Term}}$$

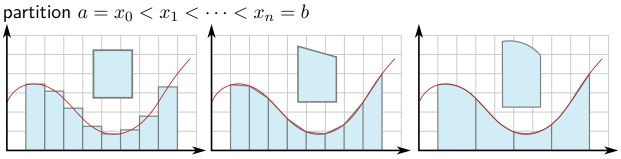
Step 3:
Minimize over all functions?
What about constraints?

subject to eq. (5) and whatever constraints are imposed on the system.

This is a nasty constrained minimization problem

Numerical integration

Suppose we wish to approximate a function $f(x)$. Divide interval into a partition $a = x_0 < x_1 < \dots < x_n = b$

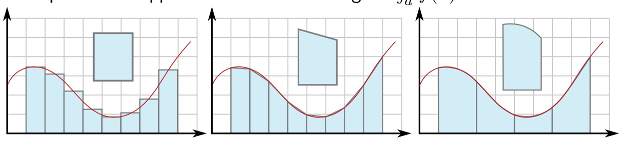


Choices corresponds to

- Piecewise constant
- Piecewise linear
- Piecewise 2nd order polynomial (use midpoint to fit the three parameters)

Approximation and integration

Each provide an approximation for the integral: $\int_a^b f(x)dx$



- Midpoint rule: $\approx \sum_{i=0}^{n-1} f\left(\frac{x_{i+1}+x_i}{2}\right)\Delta_i$
- Trapezoid rule: $\approx \frac{\Delta x}{2}\left(f\left(x_0\right)+2f\left(x_1\right)+2f\left(x_2\right)+\cdots+2f\left(x_{n-1}\right)+f\left(x_n\right)\right)$
- Simpson's rule: $\approx$ $\frac{\Delta x}{3}\left(f\left(x_0\right)+4f\left(x_1\right)+2f\left(x_2\right)+4f\left(x_3\right)+2f\left(x_4\right)+\cdots+4f\left(x_{n-1}\right)+f\left(x_n\right)\right)$

General Collocation: Time discretization

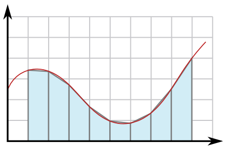
- Given t_0 and t_F and N
- We discretize the time into N intervals:

$$t_0 < t_1 < t_2 < \cdots < t_{N-1} = t_F$$

- Specifically $t_k = t_0 + \frac{k}{N-1}(t_F - t_0)$
- For later use we define:

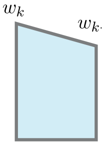
$$\begin{aligned}
h_k &= t_{k+1} - t_k, \quad k = 0, \ldots, N-2 \\
\boldsymbol{x}_k &= \boldsymbol{x}\left(t_k\right), \quad k = 0, \ldots, N-1 \\
\boldsymbol{u}_k &= \boldsymbol{u}\left(t_k\right) \\
c_k &= c\left(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k\right) \\
\boldsymbol{f}_k &= \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k\right)
\end{aligned}$$

Trapezoid collocation



Trapezoid collocation assumes

$$\int_{t_0}^{t_F} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau)d\tau \quad \approx \sum_{k=0}^{N-2} \frac{1}{2} h_k\left(c_k + c_{k+1}\right)$$



We can at this point evaluate the cost if we know $\boldsymbol{x}$ and $\boldsymbol{u}$!

$$c_F\left(t_0, t_F, \boldsymbol{x}_0, \boldsymbol{x}_N\right) + \frac{1}{2}\sum_{k=0}^{N-2} h_k\left(c_k + c_{k+1}\right)$$

Collocating system dynamics

Recall

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t)$$

Integrating both sides

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}}(t)dt = \int_{t_k}^{t_{k+1}} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt$$

Using **trapezoid collocation** we on the right-hand side and integrating the left

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k \approx \frac{1}{2}h_k\left(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k\right)$$

Trapezoid collocation: System dynamics

- Constraints are translated to simply apply to their knot points:

$$\begin{aligned}
x < 0 \quad &\rightarrow \quad x_k < 0 \\
u < 0 \quad &\rightarrow \quad u_k < 0 \\
\boldsymbol{h}(t, \boldsymbol{x}, \boldsymbol{u}) < \boldsymbol{0} \quad &\rightarrow \quad \boldsymbol{h}\left(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k\right) < \boldsymbol{0}
\end{aligned}$$

- Boundary constraints still just apply at boundary:

$$\boldsymbol{g}\left(t_0, \boldsymbol{x}\left(t_0\right), \boldsymbol{u}\left(t_0\right)\right) < \boldsymbol{0} \quad \rightarrow \quad \boldsymbol{g}\left(t_0, \boldsymbol{x}_0, \boldsymbol{u}_0\right) < \boldsymbol{0}$$

Trapezoid collocation: First attempt

Optimize over $\boldsymbol{z} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}, t_0, t_f)$

$$\min_{\boldsymbol{z}}\left[c_F\left(t_0, t_F, \boldsymbol{x}_0, \boldsymbol{x}_N\right) + \frac{1}{2}\sum_{k=0}^{N-2} h_k\left(c_k + c_{k+1}\right)\right]$$

Such that

$$\begin{aligned}
&\boldsymbol{h}\left(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k\right) < \boldsymbol{0} \\
&\boldsymbol{g}\left(t_0, t_F, \boldsymbol{x}_0, \boldsymbol{x}_F\right) \leq \boldsymbol{0}
\end{aligned}$$

with convention we iteratively compute $\boldsymbol{x}_{k+1}$ from $\boldsymbol{x}_k$ starting at $k = 0$

$$k = 0, \ldots, N-2: \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{1}{2}h_k\left(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k\right)$$

Wait, did we just solve it?

Almost! The final idea:

- Suppose we let $\boldsymbol{x}_k, \boldsymbol{u}_k$ vary freely (ensure everything can be evaluated)
- But we add the $N-1$ constraints:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{1}{2} h_k \left(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k \right)$$

- The key observation is local changes in $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ have local effects

Trapezoid collocation method

Optimize over $\boldsymbol{z} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \dots, \boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}, t_0, t_F)$

$$\min_{\boldsymbol{z}} \left[c_F(t_0, t_F, \boldsymbol{x}_0, \boldsymbol{x}_N) + \frac{1}{2} \sum_{k=0}^{N-2} h_k (c_k + c_{k+1}) \right] \quad (6)$$

$$\text{Such that } \boldsymbol{z}_{\text{lb}} \leq \boldsymbol{z} \leq \boldsymbol{z}_{\text{ub}} \quad (7)$$

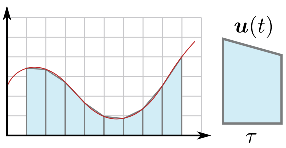
$$\boldsymbol{h}(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k) \leq \boldsymbol{0} \quad (8)$$

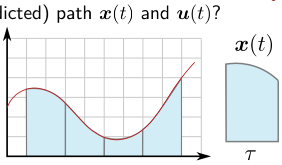
$$\boldsymbol{x}_k - \boldsymbol{x}_{k+1} + \frac{1}{2} h_k \left(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k \right) = 0 \quad (9)$$

- Optimizer also need initial point $\boldsymbol{z}_0$
- Recall $\boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k)$ so last constraint is non-linear

Reconstruction

Given $\boldsymbol{z}$, how do we reconstruct the (predicted) path $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$?





- $\boldsymbol{u}(t)$ was assumed to be linear, using $\tau = t - t_k$:

$$\boldsymbol{u}(t) \approx \boldsymbol{u}_k + \frac{\tau}{h_k} (\boldsymbol{u}_{k+1} - \boldsymbol{u}_k)$$

- For $\boldsymbol{x}(t)$ we assumed

$$\dot{\boldsymbol{x}}(t) \approx \boldsymbol{f}_k + \frac{\tau}{h_k} (\boldsymbol{f}_{k+1} - \boldsymbol{f}_k)$$

- Integrating both sides and using $\boldsymbol{x}(\boldsymbol{t}_k) = \boldsymbol{x}_k$

$$\boldsymbol{x}(t) = \boldsymbol{x}_k + \boldsymbol{f}_k \tau + \frac{\tau^2}{2h_k} (\boldsymbol{f}_{k+1} - \boldsymbol{f}_k)$$

Implementation

Algorithm 1 Direct solver

```
function DIRECT-SOLVE(N, GUESS=(t_0^g, t_F^g, x^g, u^g) )
    Define z ← (x_0, u_0, ..., x_{N-1}, u_{N-1}, t_0, t_F) as all optimization variables
    Define grid time points t_k = k/(N-1) (t_F - t_0) + t_0,   k = 0, ..., N-1   ▷ eq. (15.11)
    Define h_k, f_k = f(x_k, u_k, t_k) and c_k = c(x_k, u_k, t_k).
    Define I_eq and I_ineq as empty lists of inequality/equality constraints
    for k = 0, ..., N-2 do
        Append constraint x_{k+1} - x_k = (h_k/2)(f_{k+1} + f_k) to I_eq   ▷ eq. (15.20)
        Add all other path-constraints eq. (15.21) to I_ineq and I_eq
    end for
    Add possible end-point constraints on x_0, x_F and t_0, t_F to I_eq and I_ineq
    Build optimization target E(z) = c_f(t_0, t_F, x_0, x_{N-1}) + Σ_{k=0}^{N-2} (h_k/2)(c_{k+1} + c_k)
    Construct guess time-grid: t_k^g ← k/(N-1) (t_F^g - t_0^g) + t_0^g
    Construct guess states z^g ← (x^g(t_0^g), u^g(t_0^g), ..., x^g(t_{N-1}^g), u^g(t_{N-1}^g), t_0^g, t_F^g)
    Let z* be minimum of E optimized over z subject to I_i and I_eq using guess z^g
    Re-construct u*(t), x*(t) from z* using eq. (15.22) and eq. (15.26)
    Return u*, x* and t_0^*, t_F^*
end function
```


Making it work well

- For small N, method is imprecise, but less sensitive to $\boldsymbol{z}_0$
- For moderate N, method is **very** sensitive to $\boldsymbol{z}_0$
- Initially we do linear interpolation to get $\boldsymbol{z}_0$
- An idea is to use an optimizer for low value of N, obtain solution $\boldsymbol{z}'$
- From this $\boldsymbol{z}'$, we can construct $\boldsymbol{x}'(t)$ and $\boldsymbol{u}'(t)$
- We run optimizer with higher N and an initial guess as $\boldsymbol{x}_k = \boldsymbol{x}'(t_k)$

Implementation

Algorithm 2 Iterative direct solver

Require: An initial guess $\boldsymbol{z}_0^g = (\boldsymbol{x}^g, \boldsymbol{u}^g, t_0^g, t_F^g)$ found using simple linear interpolation

Require: A sequence of grid sizes $10 \approx N_0 < N_1 < \cdots < N_T$

```
for t = 0, T do
    x*, u*, t_0^*, t_F^* ← DIRECT-SOLVE(N_t, z_t^g)
    z_{t+1} ← x*, u*, t_0^*, t_F^*
end for
Return u*, x* and t_0^*, t_F^*
```

Implementation:

```python
# sample.py
ineq_cons = {'type': 'ineq',
             'fun': lambda x: np.array([1 - x[0] - 2 * x[1],
                                        1 - x[0] ** 2 - x[1],
                                        1 - x[0] ** 2 + x[1]]),
             'jac': lambda x: np.array([[-1.0, -2.0],
                                        [-2 * x[0], -1.0],
                                        [-2 * x[0], 1.0]])}
eq_cons = {'type': 'eq',
           'fun': lambda x: np.array([2 * x[0] + x[1] - 1]),
           'jac': lambda x: np.array([2.0, 1.0])}
from scipy.optimize import Bounds
z_lb, z_ub = [0, -0.5], [1.0, 2.0]
bounds = Bounds(z_lb, z_ub)  # Bounds(z_low, z_up)
z0 = np.array([0.5, 0])
res = minimize(J_fun, z0, method='SLSQP', jac=J_jac,
               constraints=[eq_cons, ineq_cons], bounds=bounds)
```

We use sympy because of the gradient/Jacobians

Example: Pendulum

Example: Cartpole, the Kelly task

Task is taken from the excellent [Kel17]

- Constraints: $t_0 = 0, t_F = 2$, end-point constraints $\boldsymbol{x}_0$ and $\boldsymbol{x}_F = \boldsymbol{x}^g$ and $-20 < u(t) < 20$
- $c(\boldsymbol{x}, \boldsymbol{u}, t) = u(t)^2$
- Grid refinement: $N = 10$ then $N = 60$

`lecture_05_cartpole_kelly`

Example: Cartpole, the minimum-time task

From the (also great!) `https://github.com/MatthewPeterKelly/OptimTraj/blob/master/demo/cartPole/MAIN_minTime.m`

- Constraints: $t_0 = 0, t_F > 0$, end-point constraints $\boldsymbol{x}_0$ and $\boldsymbol{x}_F = \boldsymbol{x}^g$ and $-50 < u(t) < 50$
- $c(\boldsymbol{x}, \boldsymbol{u}, t) = t_F - t_0$
- $N = 8, 16, 32, 70$

`lecture_05_cartpole_time`

Optimizing the Discrete Problem - Collocation

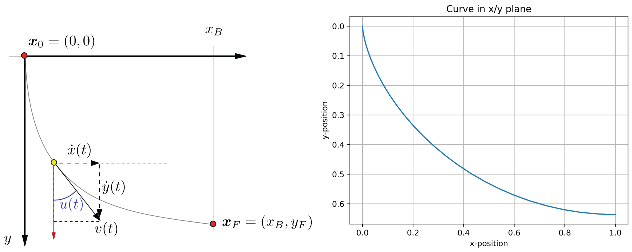
- We can also optimize over both action/state values

The optimisation problem is then defined as

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{x}_N^T Q_N \boldsymbol{x}_N + \sum_{k=0}^{N-1} (\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k) \\ \text{subject to} \quad & F' \boldsymbol{x} \leq \boldsymbol{h}' \\ & F'' \boldsymbol{x} \leq \boldsymbol{h}'' \\ & A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k - \boldsymbol{x}_{k+1} = 0 \end{aligned}$$

Example: Brachistochrone

What is the fastest path for a bead to travel x_B distance in the x-direction?



- Cost: $\min t_F$
- Actions is the angle $u(t)$. Dynamics:

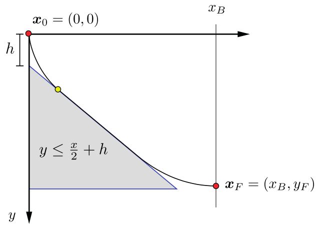
$$\dot{x} = v \sin u, \quad \dot{y} = v \cos u, \quad \dot{v} = g \cos u \tag{10}$$

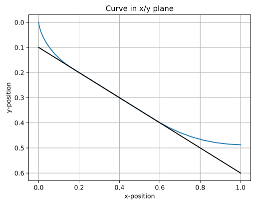
- End-point constraints

$$x(0) = y(0) = v(0) = 0, \quad x(t_F) = x_B$$

Example: Brachistochrone with dynamical constraints

Same as before but bead cannot pass through solid object





- Dynamical constraint

$$h(\boldsymbol{x}) = y - \frac{x}{2} - h \leq 0 \tag{11}$$

Extra: Hermite-Simpson

Hermite-Simpson collocation refers to replacing the Trapezoid rule

$$\int_{t_0}^{t_F} c(\tau) d\tau \approx \sum_{k=0}^{N-1} \frac{h_k}{6} \left(c_k + 4c_{k+\frac{1}{2}} + c_{k+1} \right)$$

For dynamics

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = \frac{1}{6} h_k \left(\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} + \boldsymbol{f}_{k+1} \right)$$

- **Generally better for small N**
- **Scales worse in N**

Tue Herlau.
Sequential decision making.
(Freely available online), 2024.

Matthew Kelly.
An introduction to trajectory optimization: How to do your own direct collocation.
SIAM Review, 59(4):849–904, 2017.
(See **kelly2017.pdf**).