

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

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

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

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

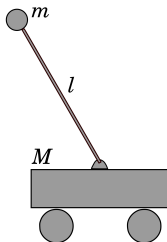
- Great job! Part 2 is online
- Survey on course experience on DTU Learn
- Thanks to the student who caught a problem with problem 1 for this weeks exercises; please point out all potential mistakes!

Dynamics of the form

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

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

Recap from last week
Example: Cartpole



- Coordinates are $\mathbf{x} = [x \quad \dot{x} \quad \theta \quad \dot{\theta}]$ (angle, angular velocity, cart position, cart velocity)
- Action u is one-dimensional; the force applied to cart
- Dynamics are

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

where \mathbf{f} is a fairly complicated function

$$\text{Equality constraint: } x = c \quad (1)$$

$$\text{Inequality constraint: } a \leq x \leq b \quad (2)$$

Any realistic physical system has constraints

- Simple boundary constraints

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

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

- End-point constraints:

$$\mathbf{x}_{0, \text{low}} \leq \mathbf{x}(t_0) \leq \mathbf{x}_{0, \text{upp}} \quad (3)$$

$$\mathbf{x}_{F, \text{low}} \leq \mathbf{x}(t_F) \leq \mathbf{x}_{F, \text{upp}}.$$

- Time constraints

$$t_{0, \text{low}} \leq t_0 \leq t_{0, \text{upp}} \quad (4)$$

$$t_{F, \text{low}} \leq t_F \leq t_{F, \text{upp}}.$$

- The cost function is of the form

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

- Necessary constraint $-u_{\max} < u(t) < u_{\max}$ and $\mathbf{x}_0 = [0 \ 0 \ \pi \ 0]$
- Goal is to bring \mathbf{x} to $\mathbf{x}^g = [1 \ 0 \ 0 \ 0]$
- Up-right cartpole, version 1:

-

$$J_u(t_0, t_F, \mathbf{x}) = \|\mathbf{x}(t_F) - \mathbf{x}^g\|^2 + \lambda \int_{t_0}^{t_F} \mathbf{u}(t)^\top \mathbf{u}(t)$$

- Constraints $t_0 = 0, t_F = 3$ (complete in 3 seconds)
- Up-right cartpole, version 2:

-

$$J_u(t_0, t_F, \mathbf{x}) = t_F - t_0$$

- Constraints $\mathbf{x}_F = \mathbf{x}^g$

Endless combinations; depends on goal + method you are using

The continuous-time control problem

Given system dynamics for a system

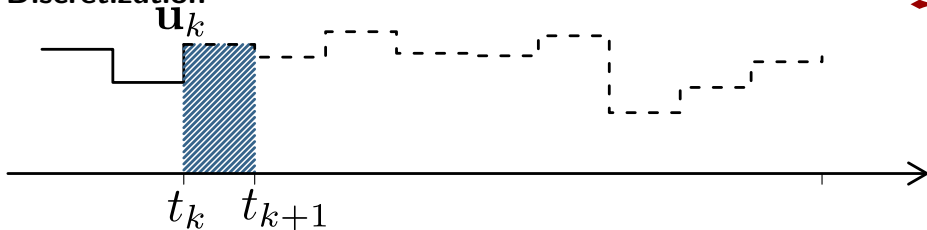
$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t))$$

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

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

(Minimization subject to all constraints)

Discretization



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

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

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

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

Recap from last week

Approaches to control



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

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

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

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

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

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 \mathbf{f} and \mathbf{h} .

A special case of the optimization task:

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

- When \mathbf{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

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

quadratic cost function

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

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

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

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

- General case

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k)$$

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

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

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

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

- + **Easy to understand**
- A big, non-linear program (we cannot avoid that for general dynamics)
- - **Unstable: small changes in \mathbf{u}_0 can mean big changes in \mathbf{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

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \quad (5)$$

Step 1: Must evaluate this ODE somehow

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

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

Step 2: Computing this integral

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

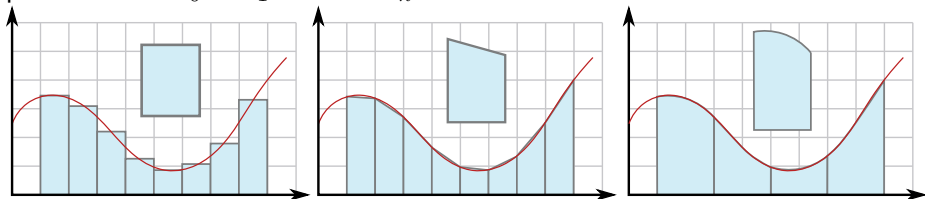
subject to eq. (5) and whatever constraints are imposed on the system.

This is a nasty constrained minimization problem

Recap from last week

Numerical integration

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



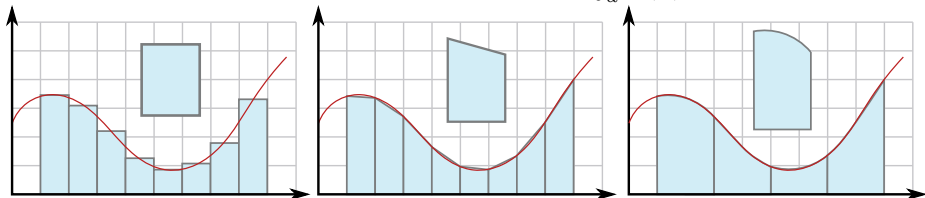
Choices corresponds to

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

Recap from last week

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: $\approx \frac{\Delta x}{2} (f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n))$
- Simpson's rule: $\approx \frac{\Delta x}{3} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n))$

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

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

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

$$h_k = t_{k+1} - t_k, \quad k = 0, \dots, N-2$$

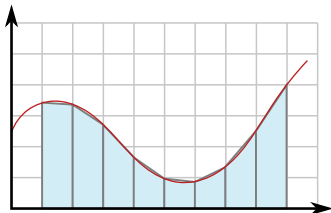
$$\mathbf{x}_k = \mathbf{x}(t_k), \quad k = 0, \dots, N-1$$

$$\mathbf{u}_k = \mathbf{u}(t_k)$$

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

$$\mathbf{f}_k = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, t_k)$$

Recap from last week
Trapezoid collocation

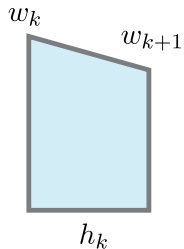


Trapezoid collocation assumes

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

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

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



Recap from last week

Collocating system dynamics



Recall

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

Integrating both sides

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

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

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

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

$$x < 0 \quad \rightarrow \quad x_k < 0$$

$$u < 0 \quad \rightarrow \quad u_k < 0$$

$$\mathbf{h}(t, \mathbf{x}, \mathbf{u}) < \mathbf{0} \quad \rightarrow \quad \mathbf{h}(t_k, \mathbf{x}_k, \mathbf{u}_k) < \mathbf{0}$$

- Boundary constraints still just apply at boundary:

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

Trapezoid collocation: First attempt

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

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

Such that

$$\mathbf{h}(t_k, \mathbf{x}_k, \mathbf{u}_k) < \mathbf{0}$$

$$\mathbf{g}(t_0, t_F, \mathbf{x}_0, \mathbf{x}_F) \leq \mathbf{0}$$

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

$$k = 0, \dots, N - 2 : \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \frac{1}{2} h_k (\mathbf{f}_{k+1} + \mathbf{f}_k)$$

Wait, did we just solve it?

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

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

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

Trapezoid collocation method

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

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

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

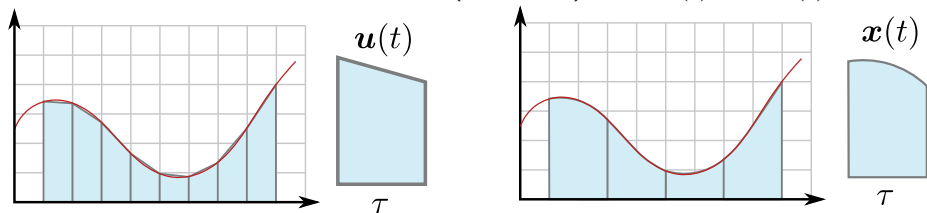
$$\mathbf{h}(t_k, \mathbf{x}_k, \mathbf{u}_k) \leq \mathbf{0} \quad (8)$$

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

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

Reconstruction

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



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

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

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

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

- Integrating both sides and using $\mathbf{x}(t_k) = \mathbf{x}_k$

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

Algorithm 1 Direct solver

- 1: **function** DIRECT-SOLVE(N , GUESS= $(t_0^g, t_F^g, \mathbf{x}^g, \mathbf{u}^g)$)
 - 2: Define $z \leftarrow (\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, t_0, t_F)$ as all optimization variables
 - 3: Define grid time points $t_k = \frac{k}{N-1}(t_F - t_0) + t_0$, $k = 0, \dots, N - 1$ \triangleright eq. (15.11)
 - 4: Define h_k , $\mathbf{f}_k = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, t_k)$ and $c_k = c(\mathbf{x}_k, \mathbf{u}_k, t_k)$.
 - 5: Define I_{eq} and I_{ineq} as empty lists of inequality/equality constraints
 - 6: **for** $k = 0, \dots, N - 2$ **do**
 - 7: Append constraint $\mathbf{x}_{k+1} - \mathbf{x}_k = \frac{h_k}{2}(\mathbf{f}_{k+1} + \mathbf{f}_k)$ to I_{eq} \triangleright eq. (15.20)
 - 8: Add all other path-constraints eq. (15.21) to I_{ineq} and I_{eq}
 - 9: **end for**
 - 10: Add possible end-point constraints on $\mathbf{x}_0, \mathbf{x}_F$ and t_0, t_F to I_{eq} and I_{ineq}
 - 11: Build optimization target $E(z) = c_f(t_0, t_F, \mathbf{x}_0, \mathbf{x}_{N-1}) + \sum_{k=0}^{N-2} \frac{h_k}{2} (c_{k+1} + c_k)$
 - 12: Construct guess time-grid: $t_k^g \leftarrow \frac{k}{N-1}(t_F^g - t_0^g) + t_0^g$
 - 13: Construct guess states $\mathbf{z}^g \leftarrow (\mathbf{x}^g(t_0^g), \mathbf{u}^g(t_0^g), \dots, \mathbf{x}^g(t_{N-1}^g), \mathbf{u}^g(t_{N-1}^g), t_0^g, t_F^g)$
 - 14: Let \mathbf{z}^* be minimum of E optimized over \mathbf{z} subject to I_i and I_{eq} using guess \mathbf{z}^g
 - 15: Re-construct $\mathbf{u}^*(t), \mathbf{x}^*(t)$ from \mathbf{z}^* using eq. (15.22) and eq. (15.26)
 - 16: Return $\mathbf{u}^*, \mathbf{x}^*$ and t_0^*, t_F^*
 - 17: **end function**
-

- For small N , method is imprecise, but less sensitive to z_0
- For moderate N , method is **very** sensitive to z_0
- Initially we do linear interpolation to get z_0
- An idea is to use an optimizer for low value of N , obtain solution z'
- From this z' , we can construct $x'(t)$ and $u'(t)$
- We run optimizer with higher N and an initial guess as $x_k = x'(t_k)$

Algorithm 2 Iterative direct solver

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

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

- 1: **for** $t = 0, T$ **do**
 - 2: $\mathbf{x}^*, \mathbf{u}^*, t_0^*, t_F^* \leftarrow \text{DIRECT-SOLVE}(N_t, \mathbf{z}_t^g)$
 - 3: $\mathbf{z}_{t+1} \leftarrow \mathbf{x}^*, \mathbf{u}^*, t_0^*, t_F^*$
 - 4: **end for**
 - 5: Return $\mathbf{u}^*, \mathbf{x}^*$ and t_0^*, t_F^*
-

Recap from last week

Implementation:



```
1 # sample.py
2 ineq_cons = {'type': 'ineq',
3             'fun': lambda x: np.array([1 - x[0] - 2 * x[1],
4                                       1 - x[0] ** 2 - x[1],
5                                       1 - x[0] ** 2 + x[1]]),
6             'jac': lambda x: np.array([[ -1.0, -2.0],
7                                       [-2 * x[0], -1.0],
8                                       [-2 * x[0], 1.0]])}
9
10 eq_cons = {'type': 'eq',
11           'fun': lambda x: np.array([2 * x[0] + x[1] - 1]),
12           'jac': lambda x: np.array([2.0, 1.0])}
13
14 from scipy.optimize import Bounds
15 z_lb, z_ub = [0, -0.5], [1.0, 2.0]
16 bounds = Bounds(z_lb, z_ub) # Bounds(z_low, z_up)
17 z0 = np.array([0.5, 0])
18 res = minimize(J_fun, z0, method='SLSQP', jac=J_jac,
19              constraints=[eq_cons, ineq_cons], bounds=bounds)
```

We use sympy because of the gradient/Jacobians


Recap from last week
Example: Pendulum



Example: Cartpole, the Kelly task

Task is taken from the excellent [Kel17]


- Constraints: $t_0 = 0, t_F = 2$, end-point constraints \mathbf{x}_0 and $\mathbf{x}_F = \mathbf{x}^g$ and $-20 < u(t) < 20$
- $c(\mathbf{x}, \mathbf{u}, t) = u(t)^2$
- 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 \mathbf{x}_0 and $\mathbf{x}_F = \mathbf{x}^g$ and $-50 < u(t) < 50$
- $c(\mathbf{x}, \mathbf{u}, t) = t_F - t_0$
- $N = 8, 16, 32, 70$

 lecture_05_cartpole_time

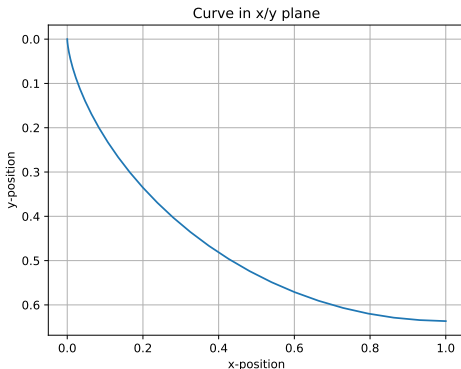
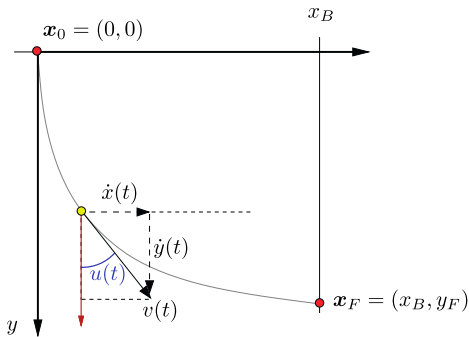
- We can also optimize over both action/state values

The optimisation problem is then defined as

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

Example: Brachistochrone

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



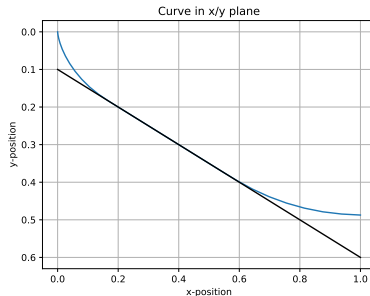
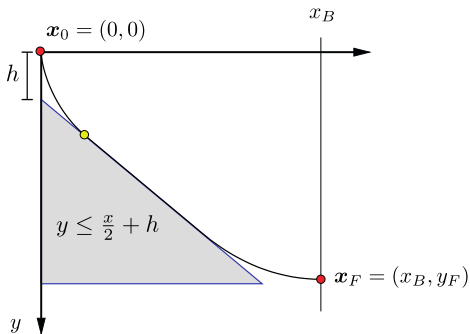
- Cost: $\min t_F$
- Actions is the angle $u(t)$. Dynamics:

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

- End-point constraints

Example: Brachistochrone with dynamical constraints

Same as before but bead cannot pass through solid object



- Dynamical constraint

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



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

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

- **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**).