Optimal Control
Lecture 8

Solmaz S. Kia
Mechanical and Aerospace Engineering Dept.
University of California Irvine
solmaz@uci.edu

Note: These slides only cover small part of the lecture. For details and other discussions consult your class notes.
Reading suggestion: Sections 3.1-3.8 of Ref [1] (see the syllabus/class website for the list of the references)".

Outline

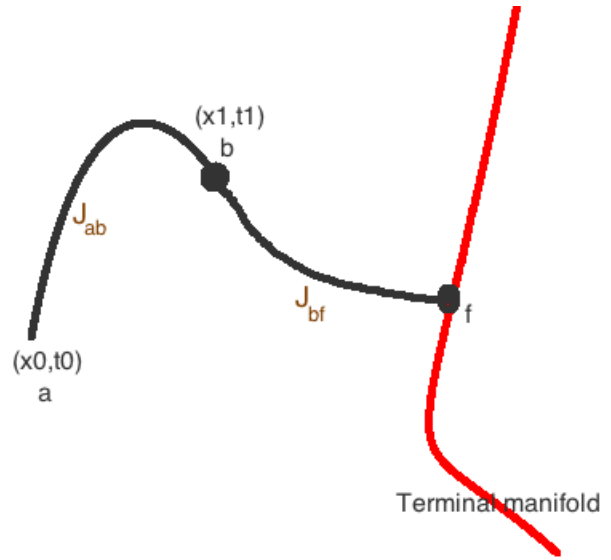
- Principle of optimality
  - Dynamic Programing

Principle of optimality

The optimal path for a multi-stage decision process:

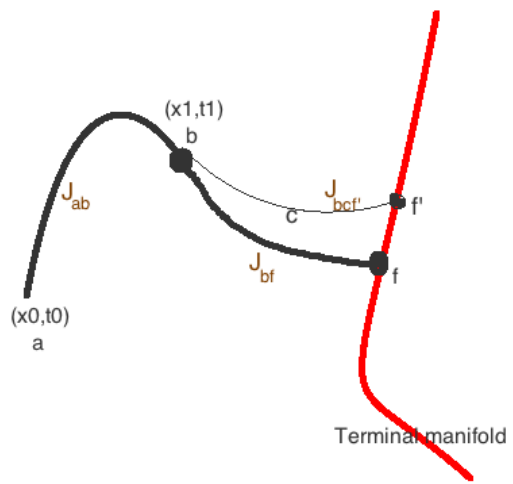
Suppose the first decision made at a (point (x_0, t_0)) results in segment $a - b$ with cost J_{ab} and the remaining decision yields segment $b - f$ (from b, point $((x_1, t_1))$) with cost of J_{bf} to arrive at the terminal manifold. The minimum cost J_{af} from $a - f$ is

$$J^{\star}_{af} = J_{ab} + J_{ab}$$



Assertion If $a - b - f$ is the optimal path from a to f then $b - f$ is the optimal path from b to f.

proof- by contradiction (see page 54 of Kirk)



Principle of optimality

Principle of optimality (due to Bellman)

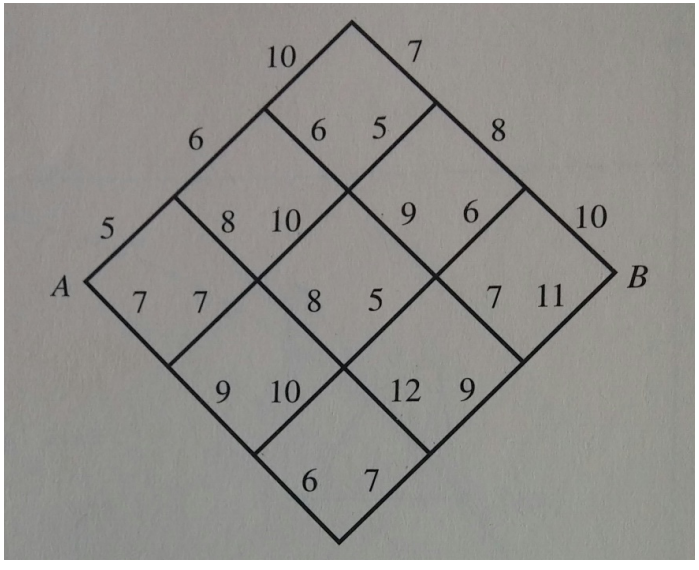
- An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.
- All points on an optimal path are possible initial points for that path
- Suppose the optimal solution for a problem passes through some intermediate point (x_1, t_1) then the optimal solution to the same problem starting at (x_1, t_1) must be the continuation of the same path.

Using principle of optimality we can construct a numerical procedure called Dynamic Programming to obtain optimal control for multi-stage decision making problems.

Dynamic Programing: example

Problem(Bryson): find the path from A to B traveling only to the right, such that the sum of the numbers on the segments along this path is a minimum.

- minimum time path from A to B: if you think of numbers as time to travel
- control decision is: up-right or down-right (only two possible value at each node
- state is between 1 to 4
- there are 20 possible paths from A to B (traveling only to the right)

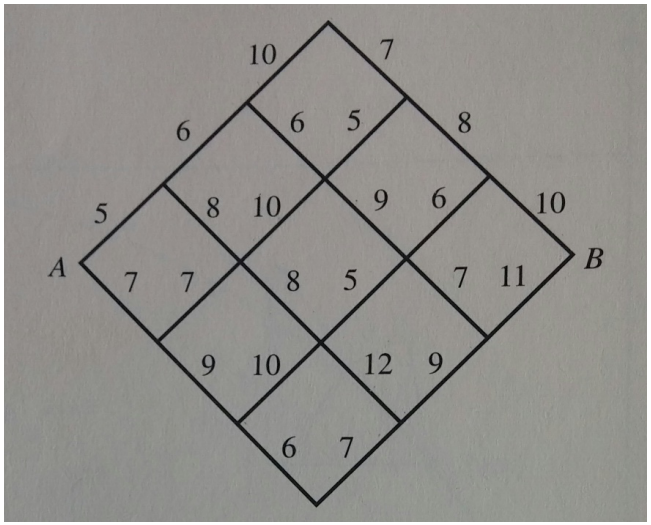


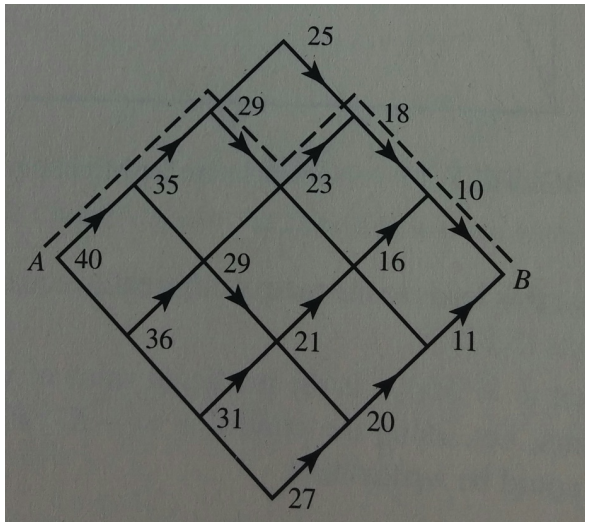
Solution approaches

1. There are 20 possible paths: evaluate each and compute the travel time (pretty tedious approach)
2. Start at B and work backwards, invoking the principle of optimality along the way.

Dynamic Programing: example

Problem(Bryson): find the path from A to B traveling only to the right, such that the sum of the numbers on the segments along this path is a minimum.

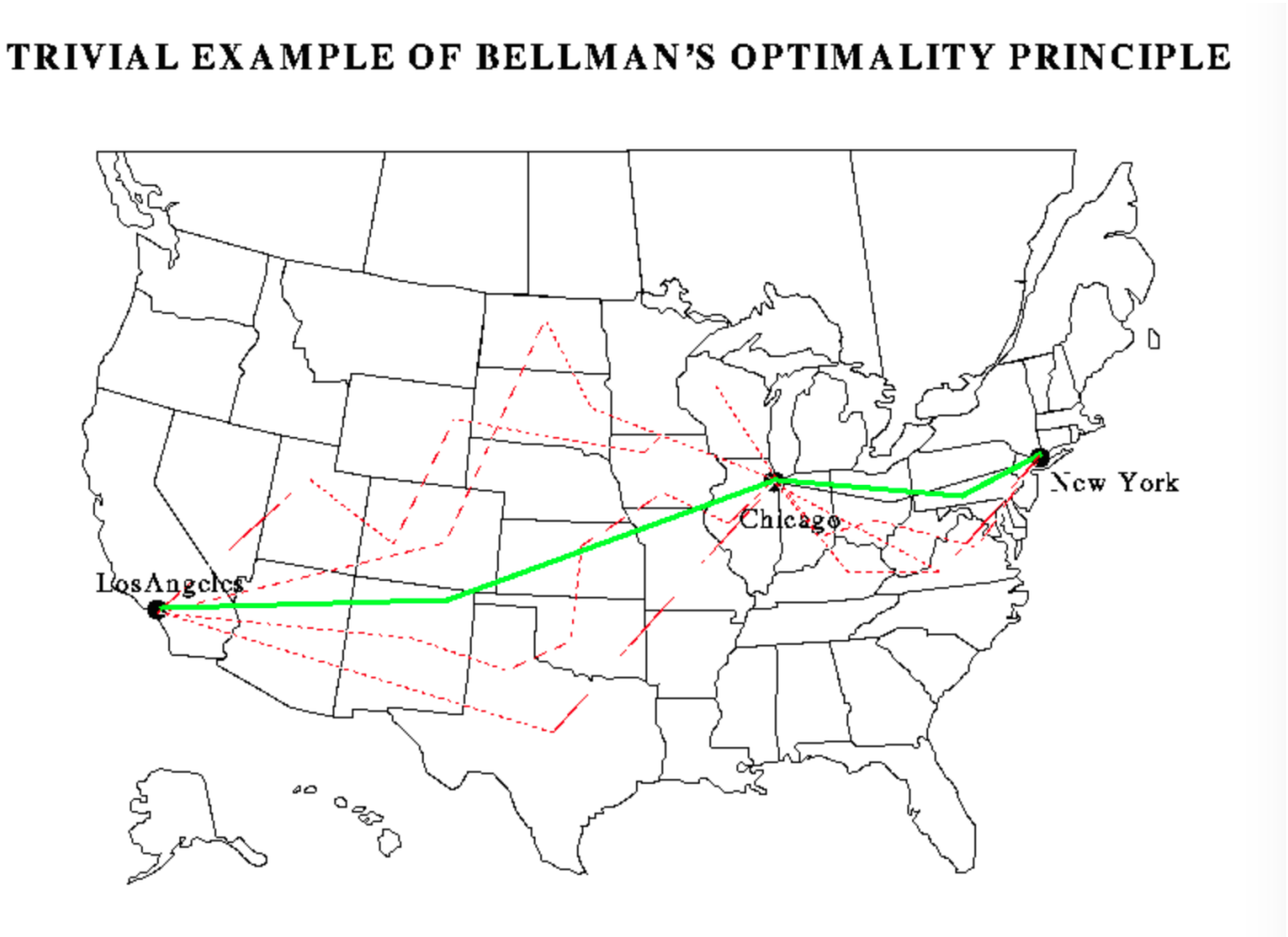




- For dynamic programing (DP) we need to find 15 numbers to solve this problem rather than evaluate the travel time for 20 paths
- Modest difference here, but scales up for larger problems.
- Let $n =$ number of segments on side (3 here) then:
  - Number of routes scales as $\sim (2n)!/(n!)2$
  - Number DP computations scales as $\sim (n+1)^2 1$

Segments on a side	3	4	5	6	7	n
Number of the routes	20	70	252	724	2632	$(2n)!/(n!)^2$
DP computations	15	24	35	48	63	$(n+1)^2-1$

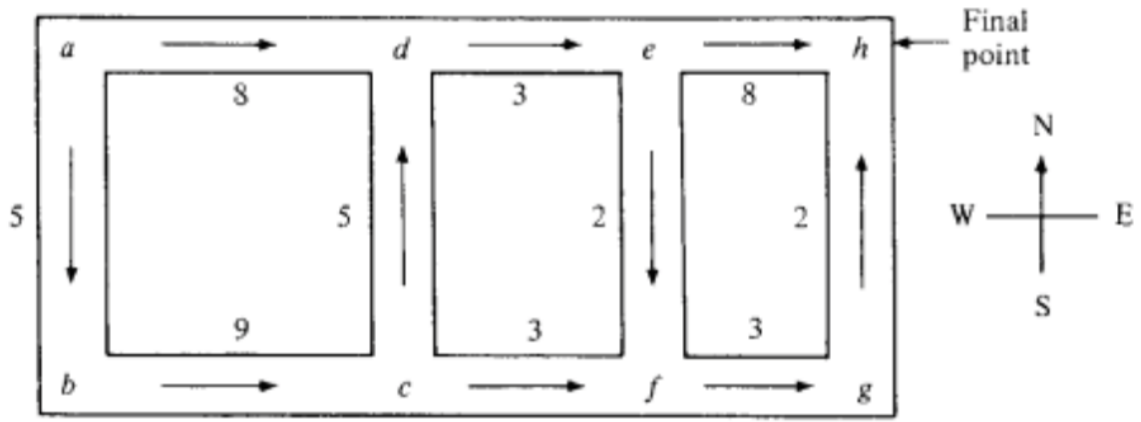
Dynamic Programing: an example



http://sipi.usc.edu/~ortega/RD_Examples/boxDP.html

Dynamic Programing: example

Problem: Minimize cost to travel from c to h moving only along the direction of arrows.



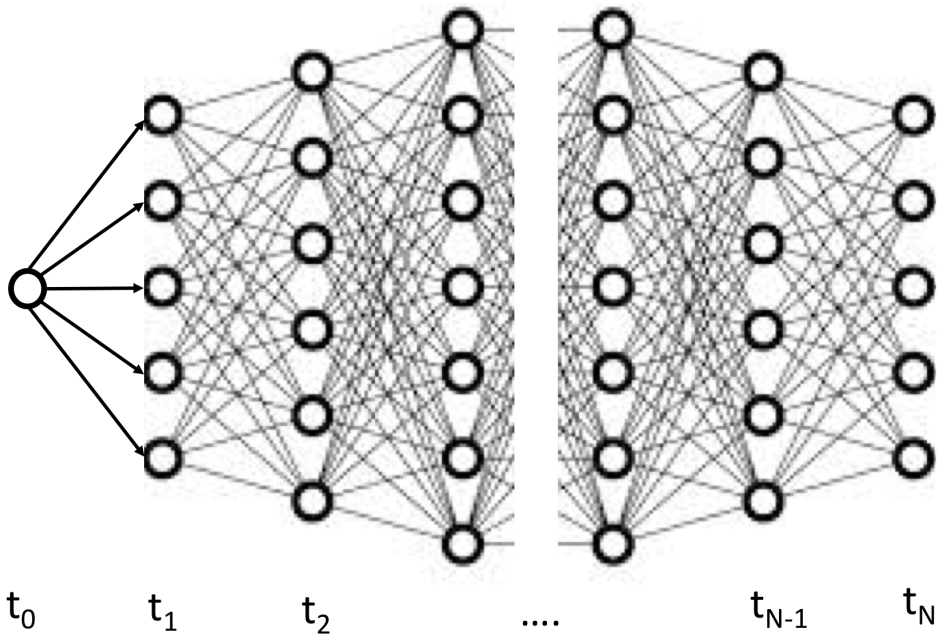
- g to h: goes directly to h, i.e., $J^{\star}_{gh} = 2$
- e to h: a possible path goes through f, we need to compute the cost of going from f to h first.
- f to h: $J^{\star}_{fh} = J_{fg} + J^{\star}_{gh} = 3 + 2 = 5$
- e to h: $J^{\star}_{eh} = \min\{J_{eh}, J_{efh}\} = \min\{J_{eh}, [J_{ef} + J^{\star}_{fh}]\} = \min\{8, 2 + 5\} = 7$,
  $e \to f \to g \to h$
- d to h: $J^{\star}_{dh} = J_{de} + J^{\star}_{eh} = 3 + 7 = 10$
- c to h:
  $J^{\star}_{ch} = \min\{J_{cdh}, J_{cfh}\} = \min\{[J_{cd} + J^{\star}_{dh}], [J_{cf} + J^{\star}_{fh}]\} = \min\{[5 + 10], [3 + 5]\} = 8$

Optimal path: $c \to f \to g \to h$

Dynamic Programing: optimal control

Roadmap to use DP in optimal control

- Grid the time/state and find the necessary control
- Grid the time/state and quantize control inputs
- Discrete-time problem: discrete time LQR



A discrete time/quantized space grid with the linkages showing the possible transition in state/time grid through the control commands. It is hard to evaluate all options moving forward through the grid, but we can work backwards and use the principle of optimality to reduce this load.

Dynamic Programing: optimal control

$$\text{minimize } J = h(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t), t)dt, \quad \text{s.t.}$$

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

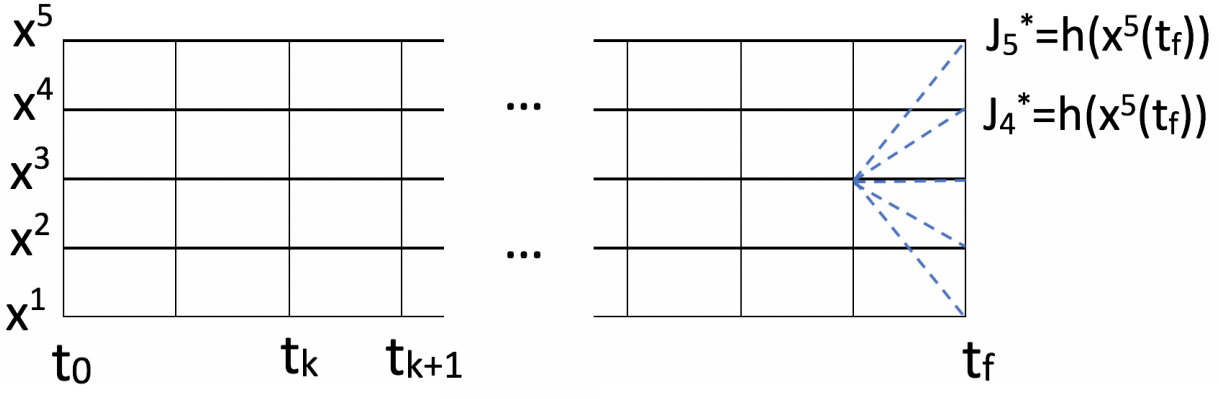
$$x(t_0) = x_0 = \text{fixed},$$

$$t_f = \text{fixed}$$

We will discuss including constraints on $x(t)$ and $u(t)$

DP solution

1. develop a grid over space/time
2. evaluate the final cost at possible final states $x_i(t_f)$: $J_i^\star = h(x_i(t_f)) \ \forall i$



Dynamic Programing: optimal control (cont'd)

3. back up 1 step in time and consider all possible ways of completing the problem
To obtain the cost of a control action, we approximate the integral in the cost.
   - let $u^{ij}(t_k)$ be the control action that takes the system from $x^i(t_k)$ to $x^j(t_{k+1})$ at time $t_k + \Delta t$. Then the approximate cost of going from $x^i(t_k)$ to $x^j(t_{k+1})$:

$$\int_{t_k}^{t_{k+1}} g(x(t), u(t), t)dt \approx g(x^i(t_k), u^{ij}(t_k), t_k)\Delta t.$$

   - $u^{ij}(t_k)$ is computed from the system dynamics:

$$\dot{x} = a(x, u, t) \Rightarrow \frac{x(t_{k+1}) - x(t_k)}{\Delta t} = a(x(t_k), u(t_k), t_k) \Rightarrow$$
$$x^j(t_{k+1}) = x^i(t_k) + a(x^i(t_k), u^{ij}(t_k), t_k)\Delta t \Rightarrow u^{ij}(t_k)$$

     If the system is control affine $\dot{x} = f(x, t) + g(x, t)u$, the control $u^{ij}(t_k)$ can be computed from $u^{ij}(t_k) = g(x_k^i, t_k)^{-1}(\frac{x^j(t_{k+1}) - x^i(t_k)}{\Delta t} - f(x_k^i, t_k))$

   - So far for any combination of x_k^i and x_{k+1}^j on the state/time grid we can evaluate the incremental cost $\Delta J(x_k^i, x_{k+1}^j)$ of making the state transition.

   - Assuming you know already the optimal path from each new terminal point x_{k+1}^j, the optimal path from x_k^i is established from

$$J^\star(x_k^i, t_k) = \min_{x_{k+1}^j} \left[\Delta J(x_k^i, x_{k+1}^j) + J^\star(x_{k+1}^j\right]$$

Dynamic Programing: optimal control (cont'd)

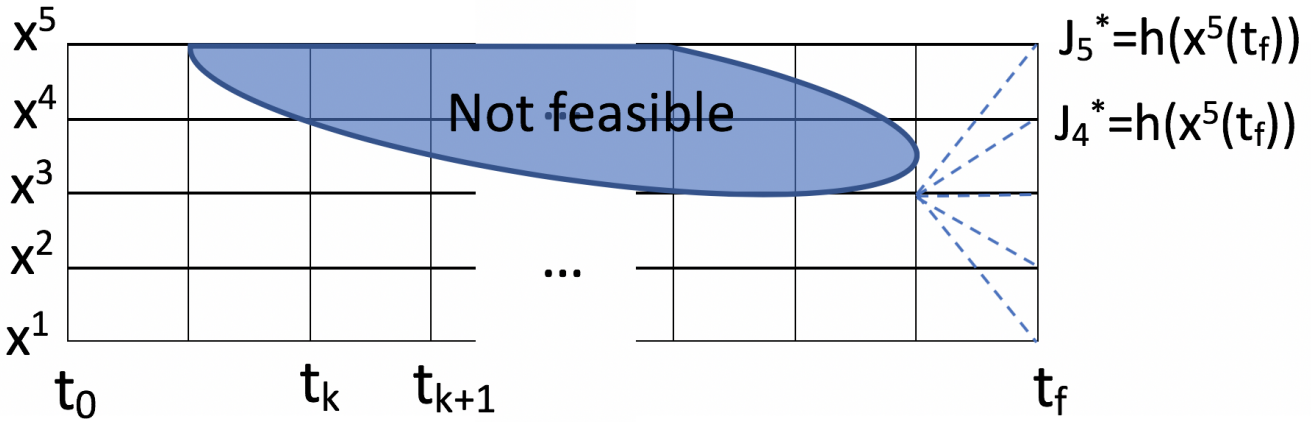
-Then for each x_k^i the output is
  * Best x_{k+1}^i to pick that gives the lowest cost
  * Control input required to achieve this best cost

4. then work backwards on time until you reach x_0, when only one value of x is allowed because of the given initial condition

---

Couple of points about the process that is explained above

- with constraints on the state, certain values of $x(t)$ might not be allowed at certain time t.



- with bounds on the control, certain state transitions might not be allowed from one time step to another
- the process extends to higher dimensions. Just have to define a grid of points in x and t. See Kirk's book for more details.

Dynamic Programing: optimal control (cont'd)

Extension of the method discussed earlier to the case of free end time with some additional constraint on the final state $m(x(t_f), t_f) = 0$, i.e.,

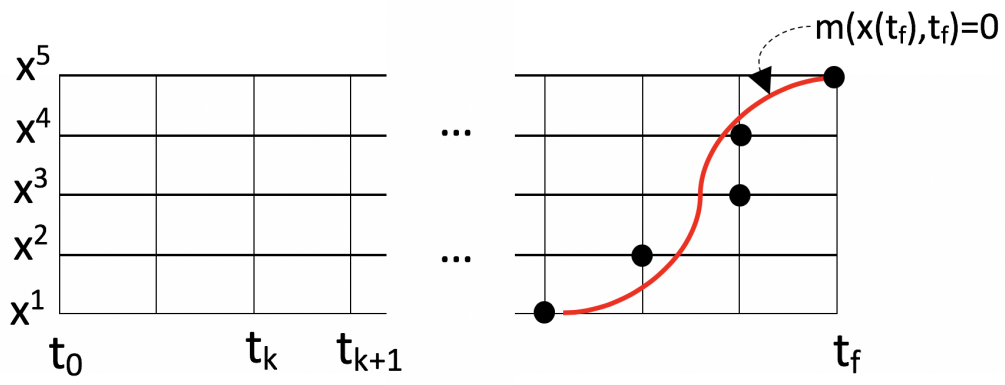
$$\text{minimize } J = h(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t), t)dt, \quad \text{s.t.}$$

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

$$x(t_0) = x_0 = \text{fixed},$$

$$m(x(t_f), t_f) = 0 \quad t_f = \text{free}$$

- find a group of points on the state/time grid that (approximately) satisfy the terminal constrain



- evaluate cost for each point and work backward from there

Dynamic Programing: optimal control (cont'd)

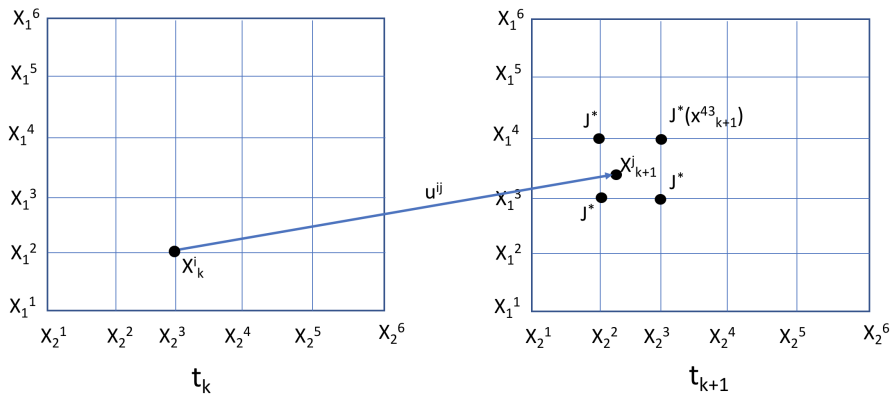
The previous formulation picked x's and used the sate equation to determine the control needed to transition between the quantized states across time.

- For more general case problems, it might be better to pick the u's and use those to determine the propagated x's

$$J^\star(x_k^i, t_k) = \min_{u_k^{ij}} \left[\Delta J(x_k^i, u_k^{ij}) + J^\star(x_{k+1}^j, t_{k+1}) \right] =$$

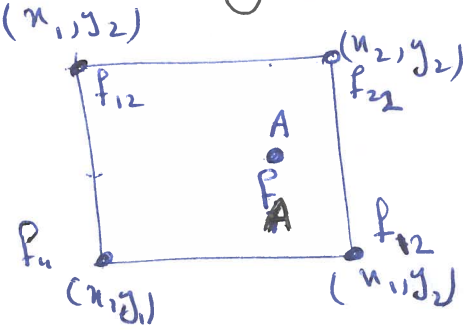
$$\min_{u_k^{ij}} \left[g(x_k^i, u_k^{ij}, t_k)\Delta t + J^\star(x_{k+1}^j, t_{k+1}) \right]$$

- To this end, the control inputs should be quantized as well.
- Then, it is likely that terminal points from one time step to the next will not lie on the state discrete points: must interpolate the cost to go between between them



Interpolation (these are not the only possibilities)

Consider $f(x,y)$. Suppose you know



$f_{11} = f(x_1,y_1)$, $f_{12} = f(x_1,y_2)$, $f_{21} = f(x_2,y_1)$

$f_{22} = f(x_2,y_2)$ and you want to use bilinear interpolation to find the value of function f at point A where $x_1 \le x_A \le x_2$, $y_1 \le y_A \le y_2$

We do first linear interpolation in x direction

$$f(x,y_1) \approx \frac{x_2-x}{x_2-x_1} f_{11} + \frac{x-x_1}{x_2-x_1} f_{21}$$

$$f(x,y_2) \approx \frac{x_2-x}{x_2-x_1} f_{12} + \frac{x-x_1}{x_2-x_1} f_{22}$$

We proceed by interpolating in the y-direction to obtain the desired estimate

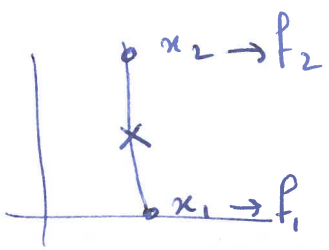
$$f(x,y) \approx \frac{y_2-y}{y_2-y_1} f(x,y_1) + \frac{y-y_1}{y_2-y_1} f(x,y_2)$$

$$\vdots$$

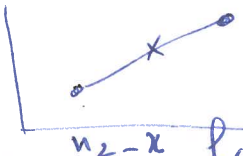
$$= \frac{1}{(x_2-x_1)(y_2-y_1)} \begin{bmatrix} x_2-x & x-x_1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} y_2-y \\ y-y_1 \end{bmatrix}$$

Linear interpolation

$$y = y_1 + (y_2-y_1)\frac{x-x_1}{x_2-x_1}$$



$$f(x) = f(x_1) + \left(\frac{f(x_2)-f(x_1)}{x_2-x_1}\right)(x-x_1)$$



$$= \frac{x_2-x}{x_2-x_1} f(x_1) + \frac{x-x_1}{x_2-x_1} f(x_2)$$

Dynamic programing: curse of dimensionality

- Main concern with dynamic programming is how badly it scales
- Given $\mathfrak{m}$ quantized states with dimension $\mathfrak{n}$ and N points in time, the number of calculations for dynamic programing is $\mathrm{N}\mathfrak{m}^{\mathfrak{n}}$

"Curse of Dimensionality"

see Dynamic Programing by R. Bellman (1957),