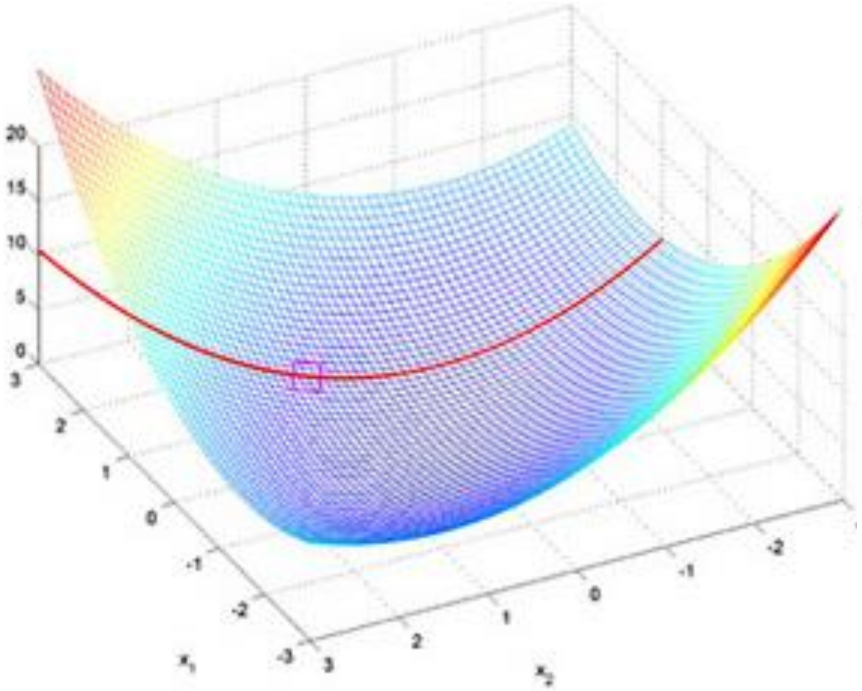
MEGR 7090/8090: Advanced Optimal Control

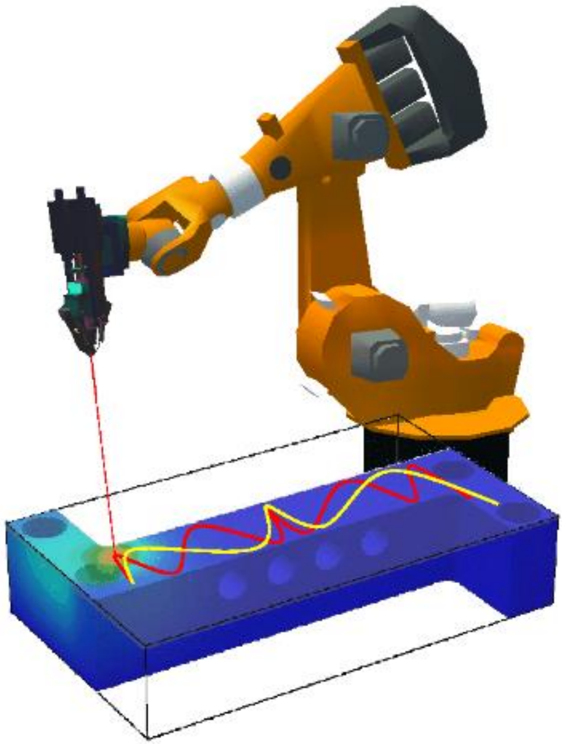
UNC CHARLOTTE



$$\boxed{V_n(\mathbf{x}_n) = \min_{\{\mathbf{u}_n,\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}} \left[\frac{1}{2}\sum_{k=n}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right]}$$

$$\begin{aligned}
V_n(\mathbf{x}_n) &= \min_{\{\mathbf{u}_n,\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}} \left[\frac{1}{2}\sum_{k=n}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right] \\
&= \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + \underbrace{\min_{\{\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}}\left[\frac{1}{2}\sum_{k=n+1}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right]}_{V_{n+1}(\mathbf{x}_{n+1})}\right] \\
&= \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + V_{n+1}(\mathbf{x}_{n+1})\right]
\end{aligned}$$

$$\boxed{V_n(\mathbf{x}_n) = \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + V_{n+1}(\mathbf{x}_{n+1})\right]}$$



Lecture 15

October 12, 2017

Bellman's Principle of Optimality – Reminder

Conditions: Bellman's principle applies to discrete-time systems (also can be used for continuous-time systems) of the form:

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

(Successor state ($\mathbf{x}(k+1)$) depends on the originating state ($\mathbf{x}(k)$) and the control signal at stage k)

Notation: Given an objective function structure $J\big(u, \mathbf{x}(0)\big) = \sum_{i=0}^{N-1} g(\mathbf{x}(i), u(i))$:

- $J^*_{\mathbf{x}_i(j)\rightarrow\mathbf{x}_f(N)}$ = Optimal *cost to go* from intermediate state x_i at step j to final state $\mathbf{x}_f$ at step N
- $J^*_{\mathbf{x}_o(j-M)\rightarrow\mathbf{x}_i(j)}$ = Optimal *cost to arrive* at intermediate state $\mathbf{x}_i$ at step j from originating state $\mathbf{x}_o$ at step j-M (often the optimal cost to arrive is the ***only*** cost to arrive, since there is only one way to get from $\mathbf{x}_o$ to $\mathbf{x}_i$ in one step
- $J^*_{\mathbf{x}_o(j-M)\rightarrow\mathbf{x}_i(j)\rightarrow\mathbf{x}_f(N)}$ = Optimal cost to go from originating state $\mathbf{x}_o$ at step j-1 to final state $\mathbf{x}_f$ at step N, through intermediate state $\mathbf{x}_i$ at step j

Bellman's principle of optimality: $J^*_{\mathbf{x}_o(j-M)\rightarrow\mathbf{x}_i(j)\rightarrow\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(j-M)\rightarrow\mathbf{x}_i(j)} + J^*_{\mathbf{x}_i(j)\rightarrow\mathbf{x}_f(N)}$

Bellman's Principle of Optimality – Reminder

If $\underline{x}$ is a valid state (given knowledge of $\underline{x}(k)$ & $u(i), i \geq k$, we can predict $\underline{x}(k+i), i \geq 1$ w/o knowing $\underline{x}(k-j), j \geq 1$ or $u(k-j), j \geq 1$)

Corollary: $$J^*_{\underline{x}_0(j-M) \rightarrow \underline{x}_i(j) \rightarrow x_f(j+N)} = J^*_{\underline{x}_0(j-M) \rightarrow x_i(j)} + J^*_{\underline{x}_i(j) \rightarrow \underline{x}_f(j+N)}$$

Dynamic Programming – Backward Recursion – Reminder

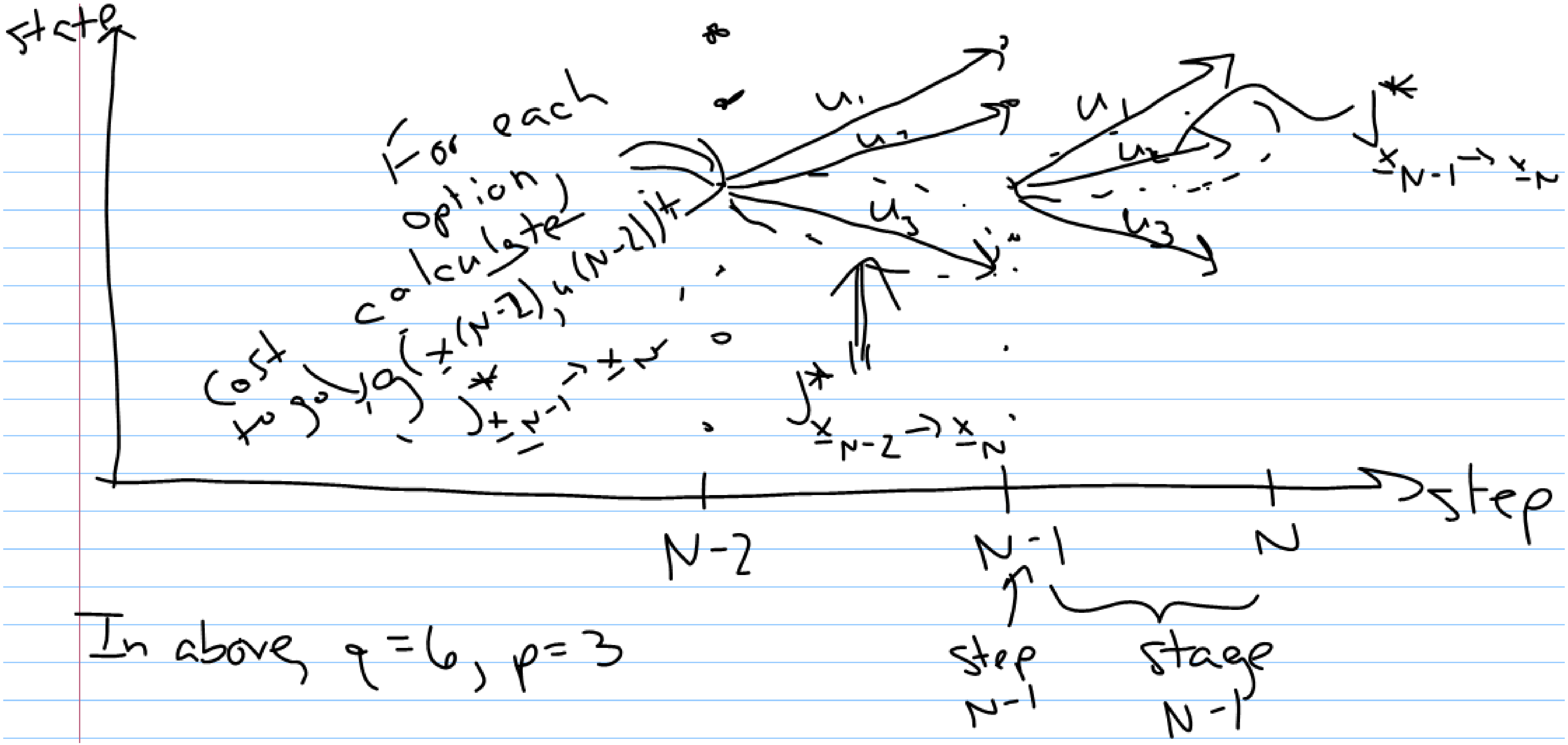
Getting set up:

- Quantize control variables into p discrete values
- Quantize state variables into q discrete values

Solution algorithm:

- Start at step N-1. For each of the q allowable state variables, calculate the stage cost $(g(\mathbf{x}(N-1), u(N-1))$ for each of the p allowable control variables that lead to constraint satisfaction. Control variables that do not satisfy constraints are termed ***inadmissible***. Record the optimal control signals and corresponding stage costs for each originating state.
- Move to step N-2. For each of the q allowable state variables, calculate the stage cost $(g(\mathbf{x}(N-1), u(N-1))$ and associated intermediate state, $\mathbf{x}_i(N-1)$, for each of the p allowable control variables that lead to constraint satisfaction. To determine which control variable is optimal, compute the total cost to go as $J^*_{\mathbf{x}_o(N-2)\to\mathbf{x}_i(N-1)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(N-2)\to\mathbf{x}_i(N-1)} + J^*_{\mathbf{x}_i(N-1)\to\mathbf{x}_f(N)}$
- Move to step N-3 and repeat the process (total cost to go is now $J^*_{\mathbf{x}_o(N-3)\to\mathbf{x}_i(N-2)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(N-3)\to\mathbf{x}_i(N-2)} + J^*_{\mathbf{x}_i(N-2)\to\mathbf{x}_f(N)}$). Keep stepping backward in time until step 0.

Dynamic Programming – Backward Recursion – Reminder



In above, $q=6$, $p=3$

Dynamic Programming – Backward Recursion – Reminder

At every stage, we create a chart:

$N-j$	u_1	u_2	u_3	
$\underline{x}_1$	$\underline{x}_{N-j+1}$			
$\underline{x}_2$				
$\underline{x}_3$				
$\underline{x}_4$				
$\underline{x}_5$				
$\underline{x}_6$				

$g(\underline{x}(N-j), u(N-j)) + J^*_{\underline{x}_{N-j+1} \to \underline{x}_N}$

Dynamic Programming – Forward Recursion – Reminder

UNC CHARLOTTE

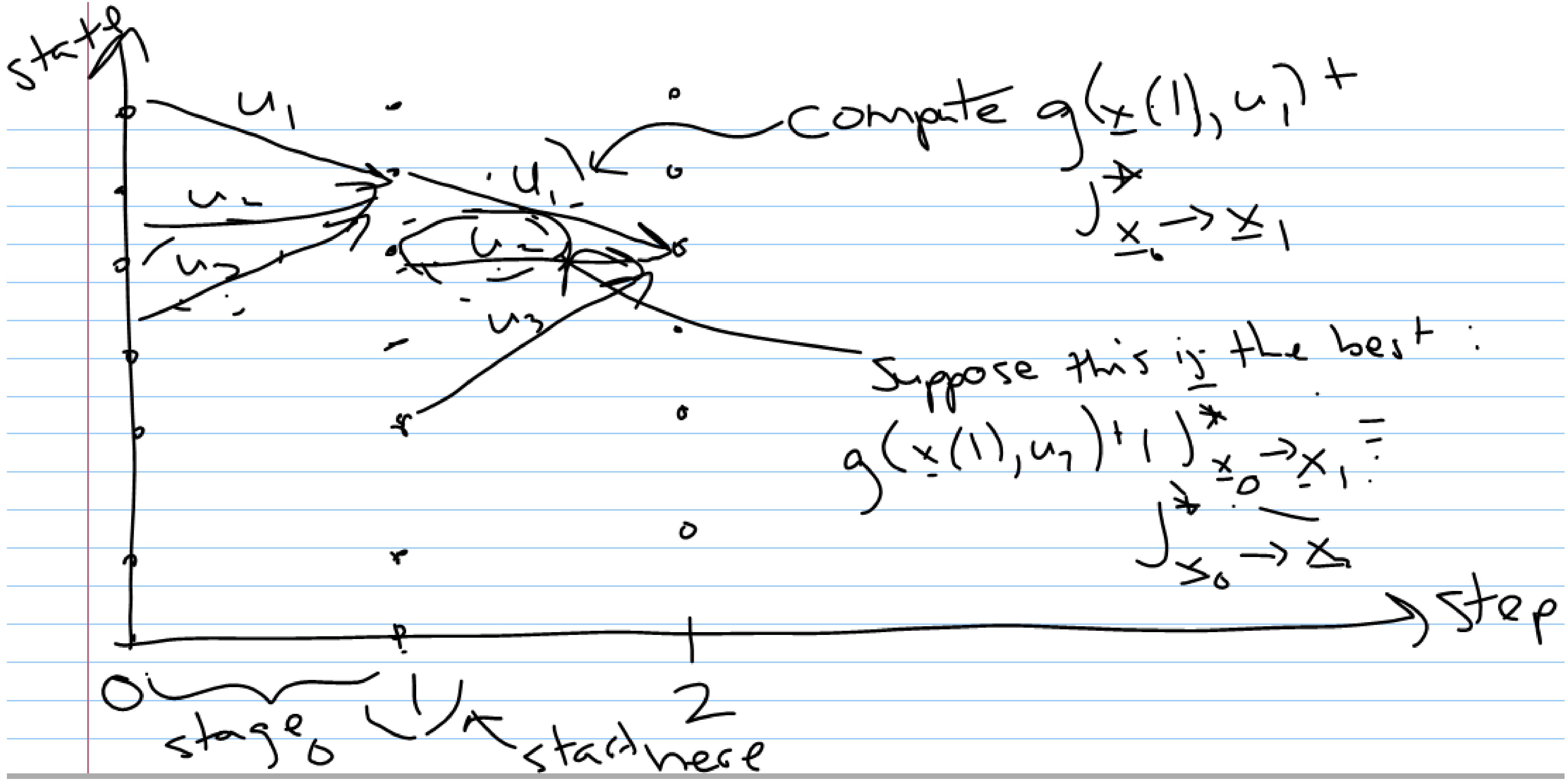
Getting set up:

- Quantize control variables into p discrete values
- Quantize state variables into q discrete values
- Usually the initial state, $\mathbf{x}(0)$, is specified (you don't get to choose your initial state)

Solution algorithm:

- Start at step 1. For each of the q allowable values of $\mathbf{x}(1)$, back-compute the value of $\mathbf{x}(0)$ and stage cost that results from each of the p allowable control values. For each admissible control value (whose associated $\mathbf{x}(0)$ satisfies initial condition requirements), determine the optimal stage cost– this cost, denoted by $J^*_{\mathbf{x}_0(0)\to\mathbf{x}_t(1)}$, is the ***optimal cost to arrive*** at state $\mathbf{x}_t(1)$.
- Move to step 2. For each of the q allowable values of $\mathbf{x}(2)$, back-compute $\mathbf{x}(1)$ and the associated stage cost for each of the p allowable control variables that lead to constraint satisfaction. To determine which control sequence is optimal, compute the total cost to arrive as $J^*_{\mathbf{x}_o(0)\to\mathbf{x}_i(1)\to\mathbf{x}_f(2)} = J^*_{\mathbf{x}_o(0)\to\mathbf{x}_i(1)} + J^*_{\mathbf{x}_i(1)\to\mathbf{x}_f(2)}$
- Move to step 2 and repeat the process. Keep stepping forward in time until step N.

Dynamic Programming – Forward Recursion – Reminder



Problems with Recursion Algorithms

Reminder: State values are quantized as $\mathbf{x}_i \in \{\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_q\}, i = 0 \dots N$, and control values are quantized as $u_i \in \{\bar{u}_1, \dots, \bar{u}_p\}, i = 0 \dots N-1$

Problem with backward recursion: What happens when the application of some control input u_i at some initial state $\mathbf{x}_i$ results in a successor state $\mathbf{x}_{i+1}$ that is not contained in $\{\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_q\}$? How do we compute the cost to go?

Problem with forward recursion: What happens when we back-compute $\mathbf{x}_i$ from $\mathbf{x}_{i+1}$ and u_i, only to find that $\mathbf{x}_i \notin \{\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_q\}$? How do we compute the cost to arrive?

The solution to both of the above problems involves ***interpolation***, which is ***essential to most applications of dynamic programming*** (except for some very simple examples)

Problems with Recursion Algorithms

$x_a, x_b, x_i \in \mathbb{R}$

1

x_a, J_a^*

u_i

x_i, J_i^*?

x_b, J_b^*

i

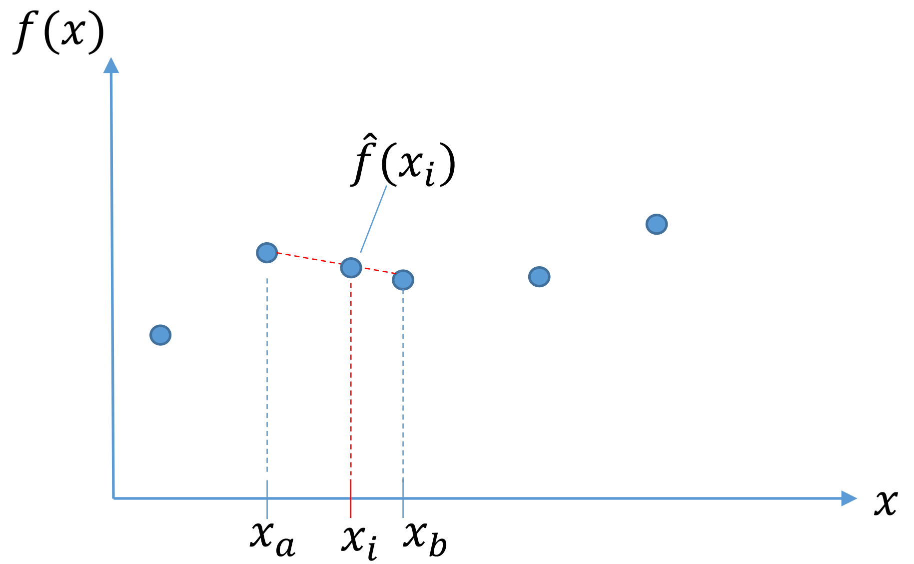
Linear Interpolation – Scalar State

Generic situation: We need to estimate the value of some function $f(x_i)$, but exact values of f are only known at neighboring grid points x_a and x_b, where $x_a < x_i, x_b > x_i$

- In the ***specific*** case of backward recursion, $f(x_i) = J^*_{i\to N}(x_i)$, i.e., the function value to be estimated is the ***cost to go***
- In the ***specific*** case of forward recursion, $f(x_i) = J^*_{0\to i}(x_i)$, i.e., the function value to be estimated is the ***cost to arrive***

An estimate of $f(x_i)$ is obtained through 1D linear interpolation:

$$\hat{f}(x_i) = \frac{(x_b - x_i)f(x_a) + (x_i - x_a)f(x_b)}{x_b - x_a}$$



$\hat{f}(x_i)$

Vector of x values for which $f(x)$ is tabulated

Vector of corresponding $f(x)$ values

x_i

MATLAB syntax: `fi = interp1(x_vec,f_vec,xi);`

Linear Interpolation – Scalar State

$$J_i^* = w J_a^* + (1-w) J_b^*$$

$$= \frac{(x_i - x_b)}{x_a - x_b} J_a^* + \frac{(x_a - x_i)}{x_a - x_b} J_b^*$$

DP-specific syntax:

$J^* = \text{interp1}(\text{admissible states, corresponding } J^*, x_i)$

Backward: successor

Fwd: predecessor

2D "Bilinear" Interpolation – Two State Variables

Situation: We need to estimate the value of some function $f(\mathbf{x}_i)$, but exact values of f are only known at neighboring 2D grid points $[x_{1a} \quad x_{2a}]^T$, $[x_{1b} \quad x_{2a}]^T$, $[x_{1a} \quad x_{2b}]^T$, and $[x_{1b} \quad x_{2b}]^T$, where $x_{1a} < x_{1i}, x_{1b} > x_i, x_{2a} < x_{1i}, x_{2b} > x_i$

An estimate of $f(\mathbf{x}_i)$ is obtained through 2D linear interpolation:

Perform two 1D interpolations in the x_1 direction

$$f_{prelim}([x_{1i} \quad x_{2a}]^T) = \frac{(x_{1b} - x_{1i})(f([x_{1a} \quad x_{2a}]^T)) + (x_{1i} - x_{1a})(f([x_{1b} \quad x_{2a}]^T))}{x_{1b} - x_{1a}}$$

$$f_{prelim}([x_{1i} \quad x_{2b}]^T) = \frac{(x_{1b} - x_{1i})(f([x_{1a} \quad x_{2b}]^T)) + (x_{1i} - x_{1a})(f([x_{1b} \quad x_{2b}]^T))}{x_{1b} - x_{1a}}$$

Perform a 1D interpolation in the x_1 direction:

$$\hat{f}([x_{1i} \quad x_{2i}]^T) = \frac{(x_{2b} - x_{2i})(f_{prelim}([x_{1i} \quad x_{2a}]^T)) + (x_{1i} - x_{1a})(f([x_{1i} \quad x_{2b}]^T))}{x_{2b} - x_{2a}}$$

2D "Bilinear" Interpolation – Two State Variables

Examples of vector state variables:

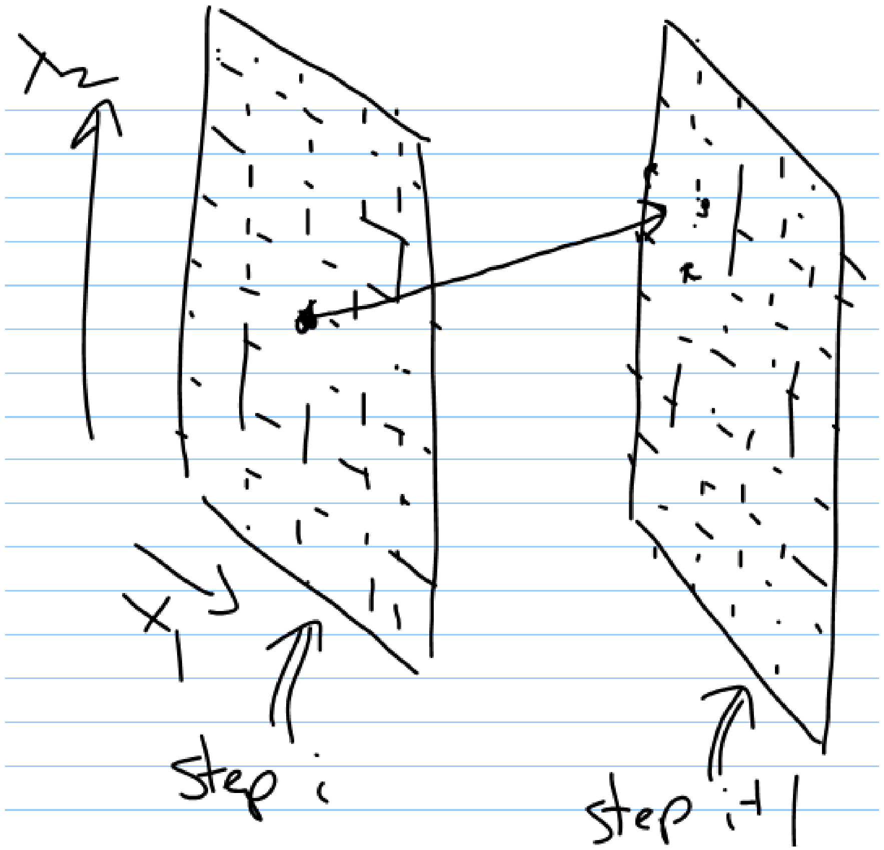
HW 3, P2: Mass-spring-damper

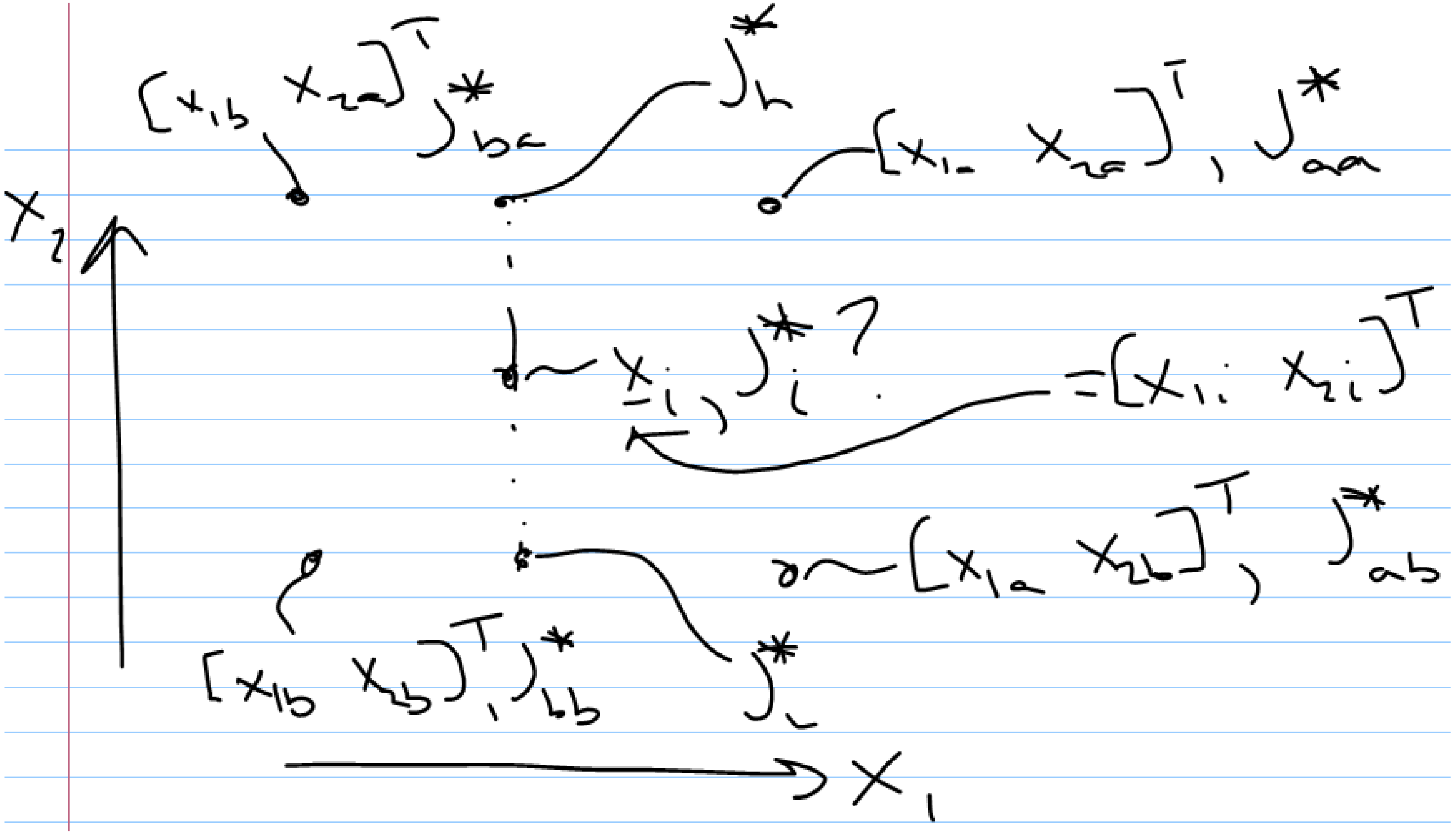
$$\underline{x} = [y \quad \dot{y}]^T \quad \ldots \quad \underline{x} \in \mathbb{R}^2$$

HW 3, P3: Vehicle

$$\underline{x} = [p \quad v]^T \quad \ldots \quad \underline{x} \in \mathbb{R}^2$$

2D "Bilinear" Interpolation – Two State Variables





2D "Bilinear" Interpolation – Two State Variables

$$J^*_h = w_h J^*_{ba} + (1 - w_h) J^*_{aa} = \frac{x_{1a} - x_{1i}}{x_{1a} - x_{1b}} J^*_{ba} + \frac{x_{1b} - x_{1i}}{x_{1a} - x_{1b}} J^*_{aa}$$

$$J^*_L = \frac{x_{1a} - x_{1i}}{x_{1a} - x_{1b}} J^*_{bb} + \frac{x_{1b} - x_{1i}}{x_{1a} - x_{1b}} J^*_{ab}$$

$$J^*_i = \frac{x_{2i} - x_{2b}}{x_{2a} - x_{2b}} J^*_h + \frac{x_{2a} - x_{2i}}{x_{2a} - x_{2b}} J^*_L$$

2D "Bilinear" Interpolation – Two State Variables

$$\underline{u}^*_h = w_h \underline{u}^*_{ba} + (1-w_h)\underline{u}^*_{aa}$$

$$\underline{u}^*_\ell = w_h \underline{u}^*_{bb} + (1-w_h)\underline{u}^*_{ab}$$

$$\Rightarrow \underline{u}^*_i = \frac{x_{2i}-x_{2b}}{x_{2a}-x_{2b}}\underline{u}^*_h + \frac{x_{2a}-x_{2i}}{x_{2a}-x_{2b}}\underline{u}^*_\ell$$

Key point: The interpolation weights on the control sequences are the same as the interpolation weights on the optimal costs to go.

n-D Interpolation – 2+ State Variables

UNC CHARLOTTE

MATLAB syntax for 2D interpolation:

Vector of x_1 values for which $f(\mathbf{x})$ is tabulated

Matrix of corresponding $f(\mathbf{x})$ values

$\hat{f}(\mathbf{x}_i)$

```
fi = interp2(x1_vec,x2_vec,f_mat,x1i,x2i);
```

Vector of x_2 values for which $f(\mathbf{x})$ is tabulated

x_{1i}

x_{2i}

Fortunately, MATLAB syntax exists for nD linear interpolation

$\hat{f}(\mathbf{x}_i)$

n-dimensional matrix of corresponding $f(\mathbf{x})$ values

Syntax: fi = interpn(x1_vec,…,xn_vec,f_mat,x1i,…xni)

Vectors of $x_1 \ldots x_n$ values for which $f(\mathbf{x})$ is tabulated (note: you need to fill in the "…" in MATLAB

$x_{1i} \ldots x_{ni}$ (note: you need to fill in the "…" in MATLAB

Dynamic Programming – Backward Recursion Example with Interpolation

System dynamics: $x(k+1) = 1.5x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{N-1} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \dots N$

where $x(0) = 1$ and $u_i \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \dots N-1$

- Perform dynamic programming, using interpolation, with state variables quantized as $x_i \in \{-1, -0.5, 0, 0.5, 1\}, i = 1 \dots N$ … take $N = 5$ …sample code on Canvas
- Note: This was the same example that gave us problems earlier

Dynamic Programming – Forward Recursion Example

System dynamics: $x(k+1) = 1.5x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{N-1} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \dots N$

where $x(0) = 1$ and $u_i \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \dots N-1$

- Determine the optimal control sequence, $\mathbf{u}^*$, using dynamic programming, with state variables quantized as $x_i \in \{-1, -0.5, 0, 0.5, 1\}, i = 1 \dots N$, for $N = 5$ and $N = 10$ …Sample code available on Canvas

- How many cost function evaluations are required? How does this compare with that of an exhaustive search, and how does the number depend on N?

Side Note on Forward Recursion

UNC CHARLOTTE

Instead of back-computing the predecessor state (then needing to interpolate when that state does not belong to the quantized set), why not compute the required control signal for a specified predecessor state?

$$\underbrace{\underline{x}(k+1)}_{\checkmark} = A\underset{?}{\underline{x}(k)} + \underbrace{Bu(k)}_{\checkmark}$$

$$\underbrace{\underline{x}(k)}_{\checkmark} = A\underset{?}{\underline{x}(k-1)} + \underbrace{Bu(k-1)}_{\checkmark}$$

$$A\underline{x}(k-1) = Bu(k-1) - \underline{x}(k)$$

$$\Rightarrow \underline{x}(k-1) = A^{-1}\left(Bu(k-1) - \underline{x}(k)\right)$$

Key point: Predecessor state can be computed reasonably easily.

Side Note on Forward Recursion

Instead of back-computing the predecessor state (then needing to interpolate when that state does not belong to the quantized set), why not compute the required control signal for a specified predecessor state?

$$\underbrace{\underline{x}(k)}_{\checkmark} = \underbrace{A\underline{x}(k-1)}_{\checkmark} + \underbrace{Bu(k-1)}_{?}$$

$$\Rightarrow Bu(k-1) = \underline{x}(k) - A\underline{x}(k-1)$$

$$\Rightarrow u(k-1) = B^{-1}(\underline{x}(k) - A\underline{x}(k-1))$$

not square.

Key point: Required control input cannot be computed in general, since B is non-square (and is generally narrow, so a solution will typically not exist)

Dynamic Programming – Insights

Key question: When our recursive algorithm is complete and the dust has settled, what have we learned?

With ***backward recursion***, we have learned the optimal control sequence for all candidate (gridded) ***initial*** states (values of $\mathbf{x}(0)$)

- …And we can specify a constraint set on terminal states (or any intermediate state)

With ***forward recursion***, we have learned the optimal control sequence for all candidate (gridded) ***terminal*** states (values of $\mathbf{x}(N)$)

- …And we can specify a constraint set of initial states (or any intermediate state)

Dynamic Programming – Insights

Key question: Just how ***global*** is dynamic programming? After all, we know that the faster convex optimization techniques (LP, QP, SQP) are ***just local***, and an exhaustive search is ***global but slow***

Answer: The dynamic programming solution is ***globally optimal up to the grid resolution*** (sometimes referred to as the “mesh”)

Interpretation: There may be control sequences, $\mathbf{u}$, for which $J(\mathbf{u}) < J(\mathbf{u}_{DP}^*)$, but these sequences include elements that are not in the quantized set of control signals (i.e., $\exists u_i$ such that $u_i \in \{\bar{u}_1, \ldots, \bar{u}_p\}$)

Dynamic Programming – Computational Complexity

Key question: How many objective function evaluations are required by dynamic programming?

Answer: At every stage (there are N of these), for every candidate state variable (there are q of these), each candidate control variable must be evaluated (there are p of these). So the number of objective function evaluations (E) is:

$$E = Npq$$

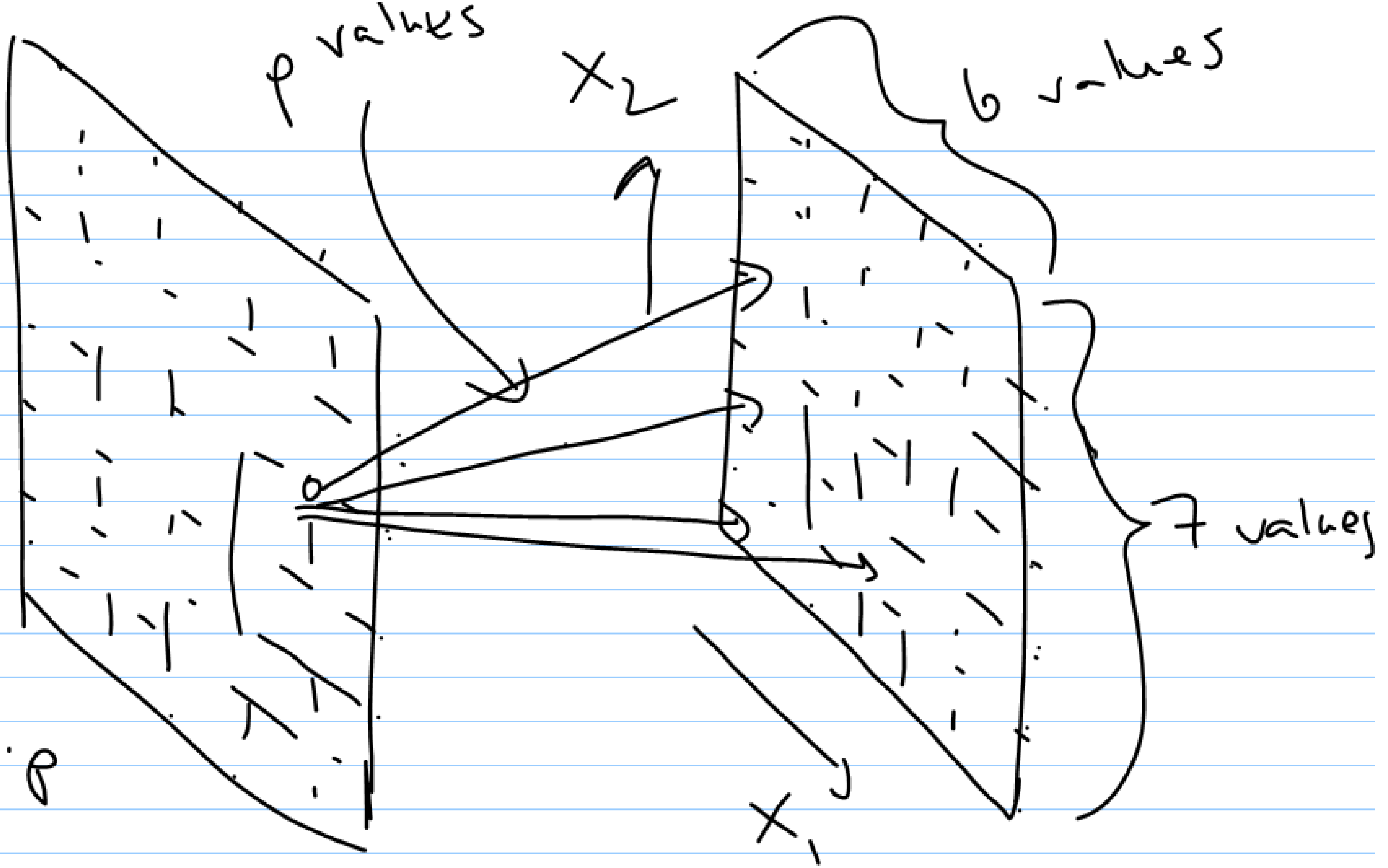
Interpretation:

- Computational complexity is **linear** in the horizon length (N), number of candidate state variables (q), and number of candidate control variables (p) – Nice!
- …but suppose there are n states, each of which is quantized into $\bar{q}$ values. Then $q = \bar{q}^n$, and $E = Np\bar{q}^n$ …this is known as the ***curse of dimensionality!***

Dynamic Programming – Computational Complexity

Total cost fn. evals. per stage = $6 \cdot 7 \cdot p$

$\Rightarrow$ Total cost fn evals. = $6 \cdot 7 \cdot N \cdot p$



Dynamic Programming – Computational Complexity

In general, total cost fun. evals: $N_p \cdot \prod_{i=1}^{\dim} q_i$,

"curse of dimensionality"

In our ex.,
$q_1 = 6,\ q_2 = 7$

Preview of Upcoming Lectures

More dynamic programming:

- More sophisticated examples
- Using Bellman's principle of optimality to derive the control law for the discrete-time linear quadratic regulator