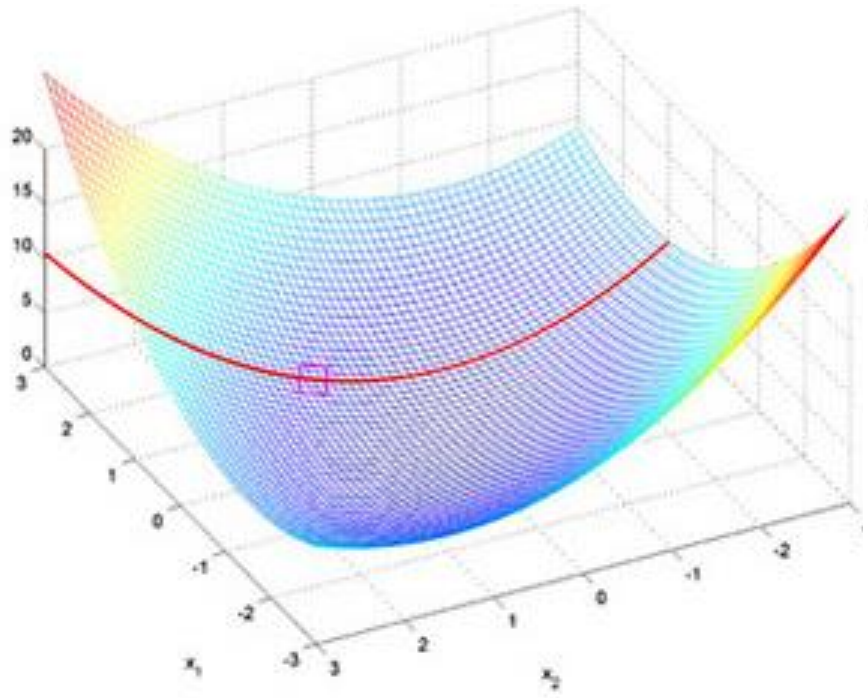
MEGR 3090/7090/8090: Advanced Optimal Control

UNC CHARLOTTE



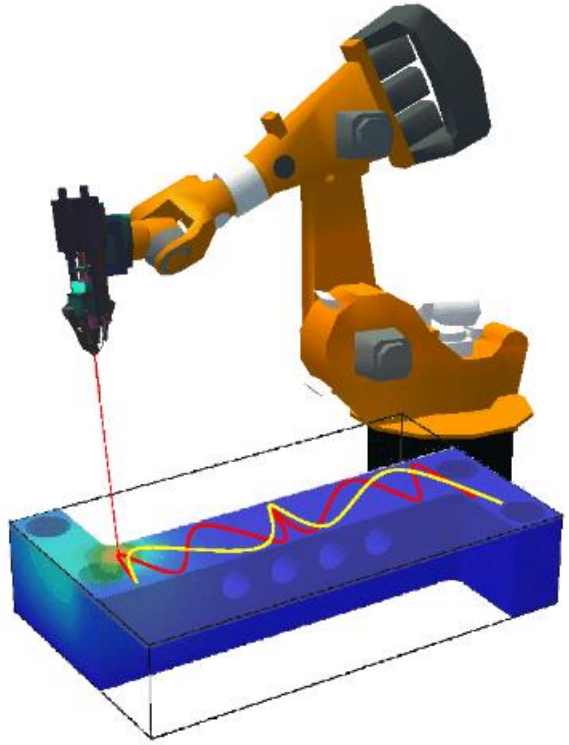
$$V_n(\mathbf{x}_n) = \min_{\{\mathbf{u}_n, \mathbf{u}_{n+1}, \cdots, \mathbf{u}_{N-1}\}} \left[\frac{1}{2} \sum_{k=n}^{N-1} \left(\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right) + \frac{1}{2} \mathbf{x}_N^T \mathbf{Q}_N \mathbf{x}_N \right]$$

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

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

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

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



Lecture 6

September 7, 2017

Summary

Given an **unconstrained** optimization problem (minimize $J(\mathbf{u})$ subject to $\mathbf{u} \in \mathbb{R}^{\dim(\mathbf{u})}$), a **finite optimum** (minimizer) can be determined through the following procedure:

- Compute $\nabla J(\mathbf{u})$, and determine $\mathbf{u}^*$ for which $\nabla J(\mathbf{u}^*) = 0$
- Compute the Hessian to determine whether candidate $\mathbf{u}^*$ values are indeed local or global minimizers (or neither)

Potential complication: Finding $\mathbf{u}^*$ for which $\nabla J(\mathbf{u}^*) = 0$ often leads to a system of nonlinear equations for which an **analytical** solution is hard to obtain. Today, we will learn about efficient **numerical** techniques for converging to $\mathbf{u}^*$

Complications of Analytical Optimization Techniques - Example

Suppose that $J(\mathbf{u}) = u_1 + u_2 + u_3 e^{-u_2} + u_2^2 e^{-u_1}$

- Compute $\nabla J(\mathbf{u})$
- Is the resulting system of equations linear or nonlinear?
- Can the resulting system of equations be solved analytically?

Note: I accidentally wrote something different here than what is used in example 2 later in the notes. However, in **both** cases, the resulting system of equations is **nolinear** and **can't be solved analytically.**

Complications of Analytical Optimization Techniques - Example

UNC CHARLOTTE

$$J(\underline{u}) = u_1 + u_2 + u_3 e^{-u_2} + u_2^2 e^{-u_1}$$

$$\nabla J = \left[\underbrace{1 - u_2^2 e^{-u_1}}_{\frac{\partial J}{\partial u_1}} \quad \underbrace{1 + 2u_2 e^{-u_1} - u_3 e^{-u_2}}_{\frac{\partial J}{\partial u_2}} \quad \underbrace{e^{-u_2}}_{\frac{\partial J}{\partial u_3}} \right]$$

$$\nabla J(\underline{u}^*) = \underline{0}$$

Complications of Analytical Optimization Techniques - Example

UNC CHARLOTTE

$$1 - u_2^{*2} e^{-u_1^*} = 0$$

$$1 + 2u_2^* e^{-u_1^*} - u_3^* e^{-u_2^*} = 0$$

$$e^{-u_2^*} = 0$$

} nonlinear

Gradient Descent for Numerical Optimization

Gradient descent is an **iterative** algorithm that starts with an initial guess, u_0, and adjust this initial guess at each iteration **in the direction of** $\nabla J(\mathbf{u})$ to converge to the minimizer of $J(\mathbf{u})$ (which we have denoted by $\mathbf{u}^*$).

Algorithm:

Specify $\mathbf{u_0}$, $\Delta u_{convergence}$, and α
Set $k = 0$
Set Δu to something big

while $\Delta u \geq \Delta u_{convergence}$

$$\mathbf{u_{k+1}} = \mathbf{u_k} - \alpha(\nabla J\ (\mathbf{u_k}))^T$$

$$\Delta u = \|\mathbf{u_{k+1}} - \mathbf{u_k}\|$$

$$k = k + 1$$

end

Update law – Move in the direction of greatest descent (slope)

Important notation:

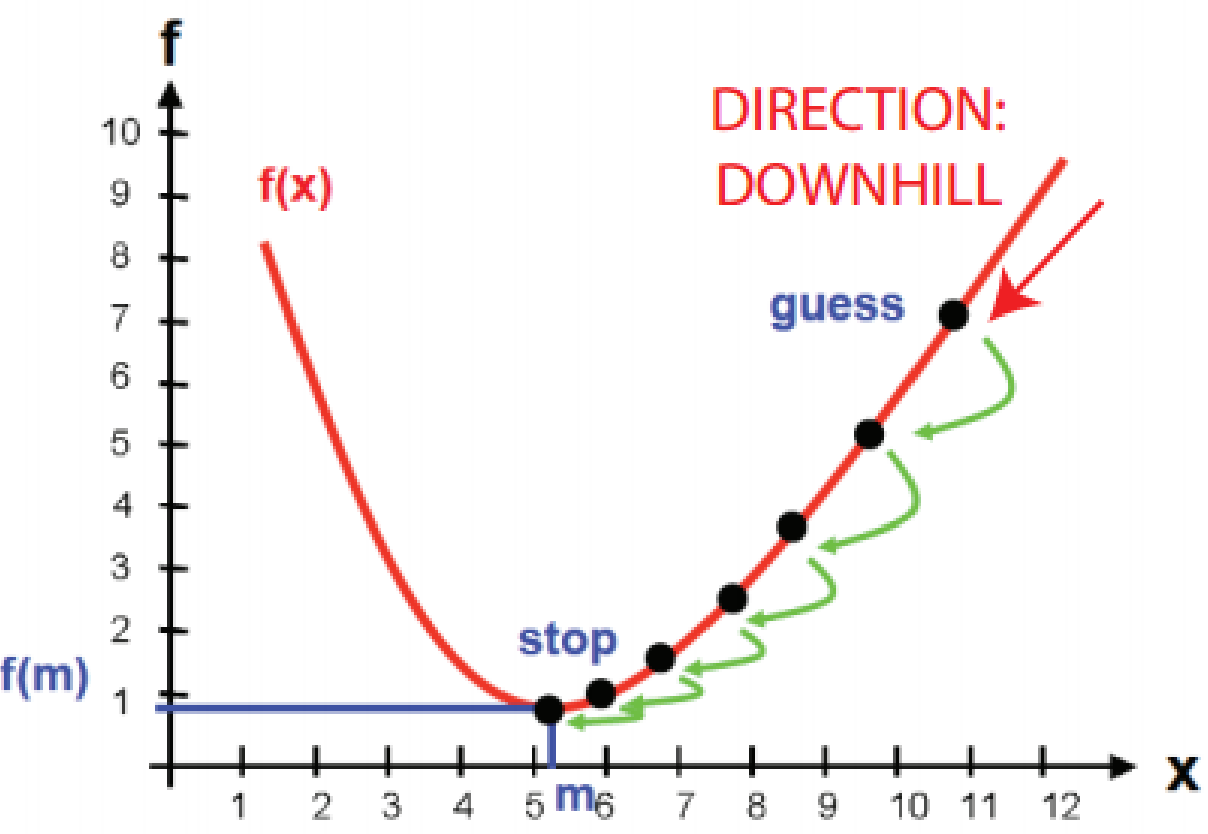
$\Delta u_{convergence}$ = convergence threshold (how close do successive estimates need to lie in order to terminate the search?)

α = step length

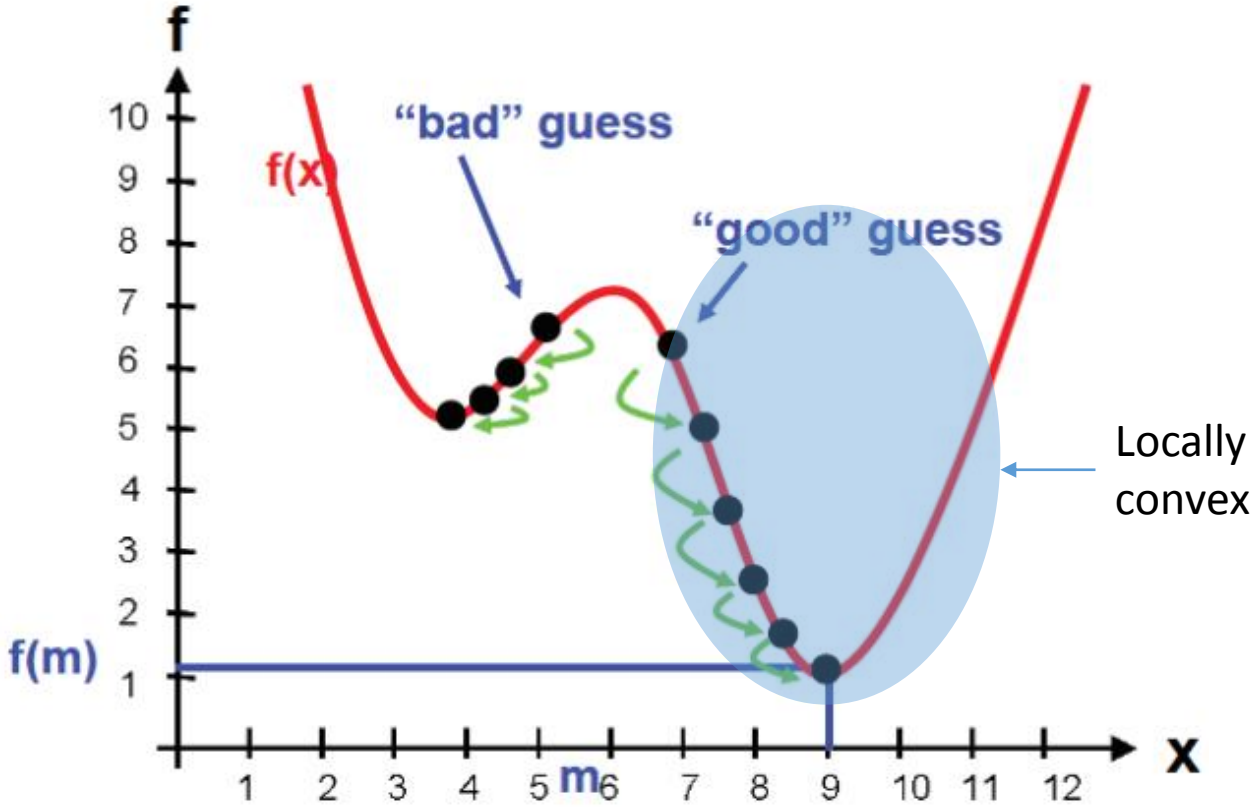
Controls the step size – Can be fixed or iteration-varying

Gradient Descent - Graphical

Image credits: Scott Moura (ecal.berkeley.edu)



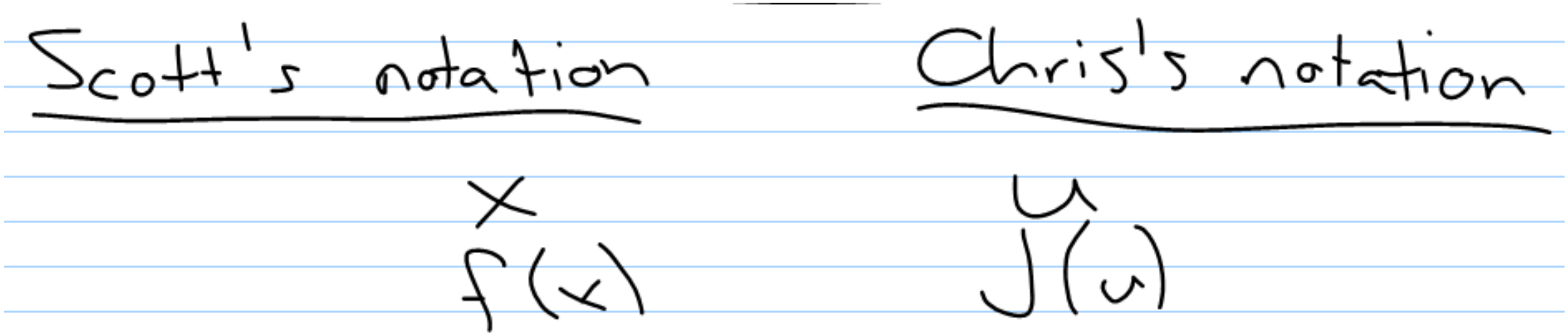
Key point: Iteration-to-iteration adjustment in guess is proportional to slope (gradient).



Key point: Gradient descent is guaranteed to converge for **convex objective functions**, not generally for others. Thus, for objective functions that aren't globally convex, gradient descent is a **local** algorithm.

Gradient Descent - Graphical

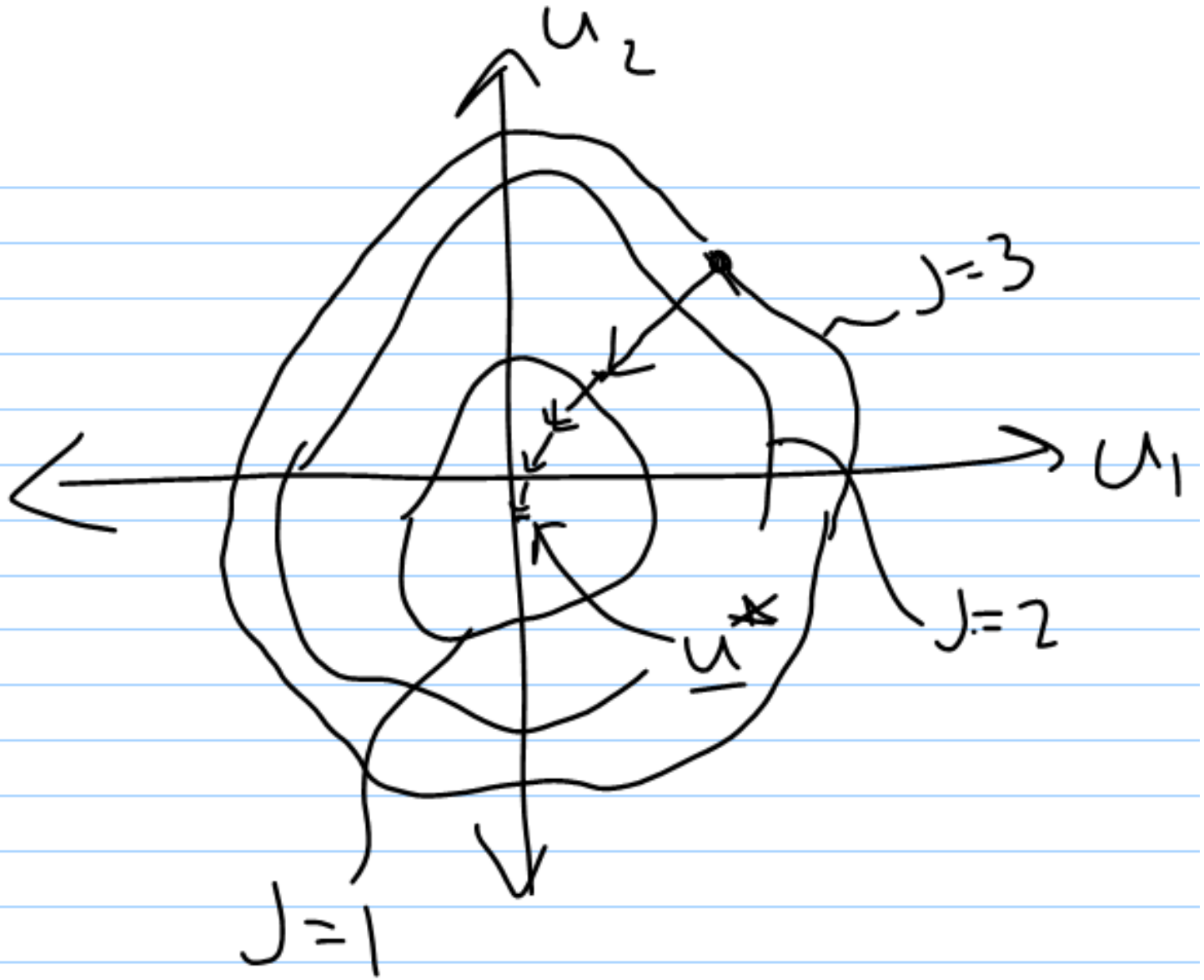
UNC CHARLOTTE



Note: Scott's notation is also the notation used in Papalambros. The reason that I use different notation is not to confuse you! It's because the letter 'u' is reserved in control for the control signal, and the letter 'x' is reserved for the state variable.

Gradient Descent – Graphical – 2D

UNC CHARLOTTE



Gradient Descent – Design Optimization Example 1

UNC CHARLOTTE

Objective function: $J(\mathbf{u}) = 4u_1^2 + 3u_1u_2 + u_2^2$

Tasks:

- Compute $\nabla J(\mathbf{u})$
- Determine the minimizer analytically
- Prove (based on the Hessian) that the minimizer is global
- Compute $\mathbf{u}^*$ numerically using a gradient descent approach

Sample code provided on Canvas

Gradient Descent – Design Optimization Example 1

UNC CHARLOTTE

Ex. 1: $J(\underline{u}) = 4u_1^2 + 3u_1u_2 + u_2^2$

$$\nabla J = \begin{bmatrix} 8u_1 + 3u_2 & 2u_2 + 3u_1 \end{bmatrix}$$

$$8u_1^* + 3u_2^* = 0$$

$$2u_2^* + 3u_1^* = 0$$

$$\Rightarrow \underbrace{\begin{bmatrix} 8 & 3 \\ 3 & 2 \end{bmatrix}}_{A} \begin{bmatrix} u_1^* \\ u_2^* \end{bmatrix} = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{b}$$

Gradient Descent – Design Optimization Example 1

UNC CHARLOTTE

$$\underline{u}^* = A^{-1} b = \underline{0}$$

$$H = \begin{bmatrix} \frac{\partial^2 J}{\partial u_1^2} & \frac{\partial^2 J}{\partial u_1 \partial u_2} \\ \frac{\partial^2 J}{\partial u_2 \partial u_1} & \frac{\partial^2 J}{\partial u_2^2} \end{bmatrix} = \begin{bmatrix} 8 & 3 \\ 3 & 2 \end{bmatrix}$$

$$\det(H) = 16 - 9 = 7 > 0$$

$\Rightarrow H$ is positive def. $\forall \underline{u}$

$\Rightarrow \underline{u}^*$ is a unique global mimizer

Gradient Descent – Design Optimization Example 2

UNC CHARLOTTE

Objective function: $J(\mathbf{u}) = u_1 + u_2 + u_1 e^{-u_2} + u_2^2 e^{-u_1}$

Task: Compute $\mathbf{u}^*$ numerically using a gradient descent approach

Sample code provided on Canvas

Gradient Descent – Optimal Control Example

UNC CHARLOTTE

Consider the following discrete-time system model: $x(k+1) = x(k) + u(k)$

Objective: Minimize $J(\mathbf{u}; x(0)) = \sum_{i=0}^{2} [x(i)^2 + u(i)^2]$

Given: $x(0) = 10$

Tasks:

- Compute (recall, based on last lecture) the control trajectory for which $\nabla J(u) = 0$.
- Prove that this optimal control trajectory.
- Use gradient descent to compute the optimal control trajectory from an initial guess.

Sample code provided on Canvas

Gradient Descent – Optimal Control Example

UNC CHARLOTTE

Ex. 3

$$x(k+1) = x(k) + u(k)$$

$$J(\underline{u}; x(0)) = \sum_{i=0}^{2} \left[x(i)^2 + u(i)^2 \right]$$

$$x(0) = 10$$

$$\Rightarrow J(\underline{u}; x(0)) = 10^2 + u(0)^2 + [10 + u(0)]^2 + u(1)^2 + [10 + u(0) + u(1)]^2 + u(2)^2$$

Gradient Descent – Optimal Control Example

UNC CHARLOTTE

$$\nabla J = \begin{bmatrix} 2u(0) + 2[10 + u(0)] + 2[10 + u(0) + u(1)] \\ 2u(1) + 2[10 + u(0) + u(1)] \\ 2u(2) \end{bmatrix}$$

$$\Rightarrow u^*(2) = 0$$

$$6u(0) + 2u(1) = -40$$

$$2u(0) + 4u(1) = -20$$

Gradient Descent – Optimal Control Example

UNC CHARLOTTE

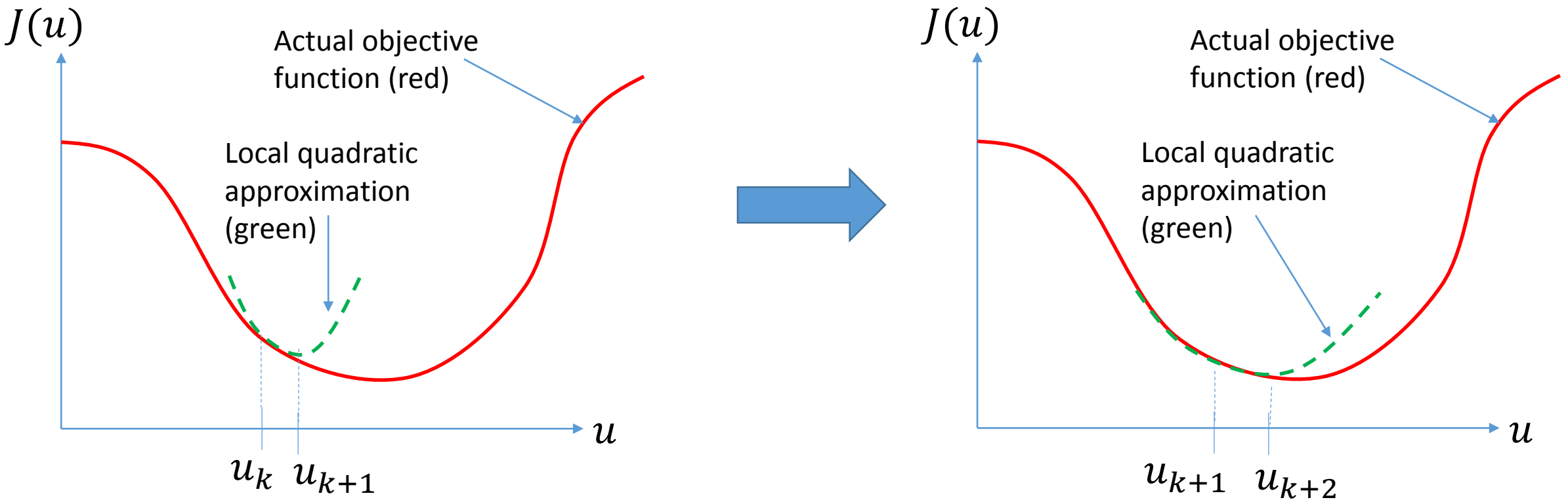
$$\Rightarrow \begin{bmatrix} 6 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \end{bmatrix} = \begin{bmatrix} -40 \\ -20 \end{bmatrix}$$

$$\Rightarrow \underline{u}^* = \begin{bmatrix} -6 & -2 & 0 \end{bmatrix}^T$$

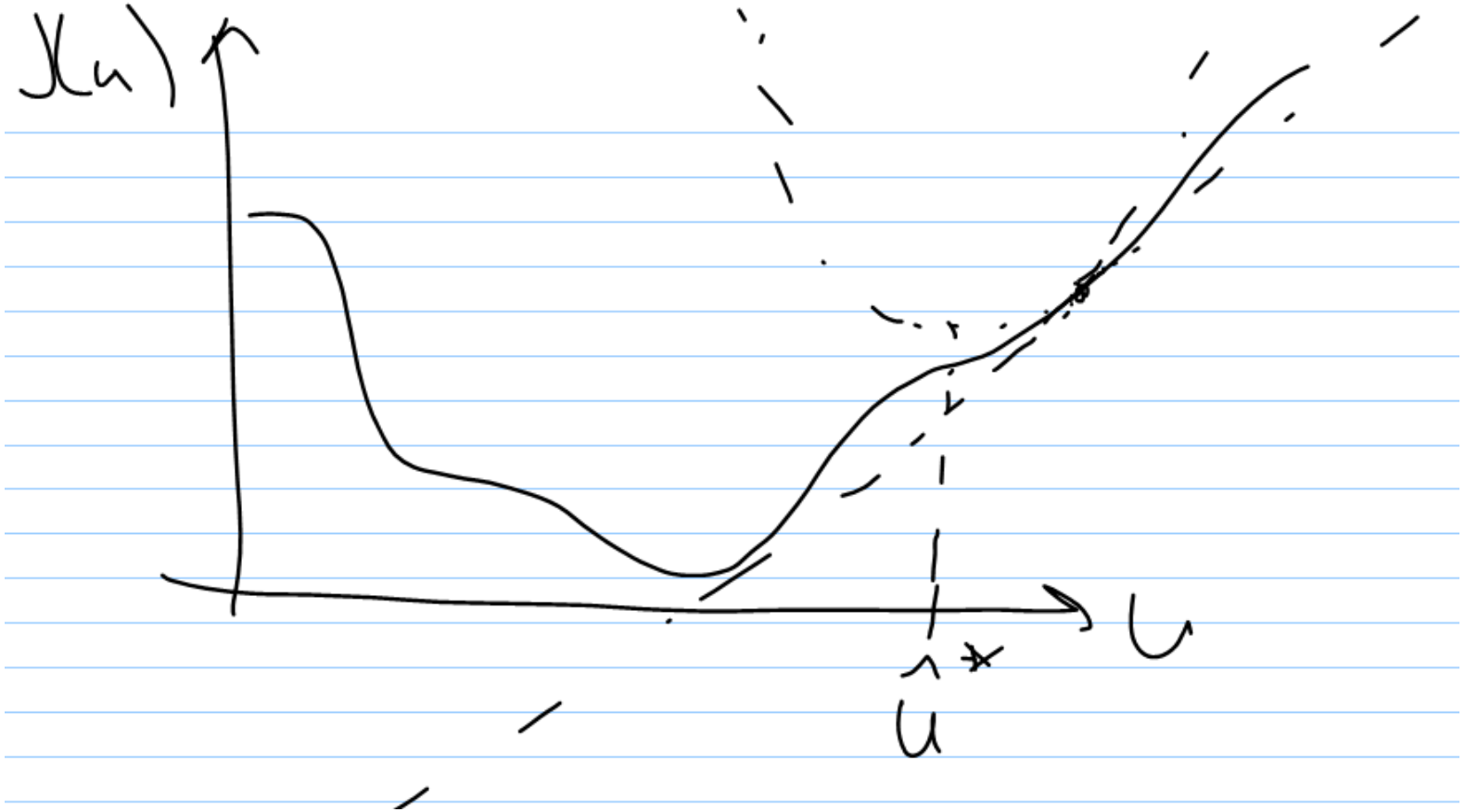
Newton's Method (also known as unconstrained sequential quadratic programming (SQP))

Newton's method is an **iterative** algorithm that starts with an initial guess, $\mathbf{u_0}$, and adjust this initial guess at each iteration **based on a locally quadratic approximating of $J(\mathbf{u})$**, to converge to the minimizer of $J(\mathbf{u})$ (which we have denoted by $\mathbf{u}^*$).

Graphical idea (2 iterations – left = iteration k, right = iteration k+1):



Newton's Method (also known as unconstrained sequential quadratic programming (SQP))



With gradient descent, you make a local linear approximation of the objective function – when doing so, the minimizer of that approximation is at infinity, so you clearly should only move **in the direction** of that minimizer. With Newton's method, you make a quadratic approximation, so the minimizer is finite. So it becomes reasonable in some instances to go **all the way** to the minimizer of the approximated objective function. When you go all the way, it's called "pure" Newton's method.

Newton's Method – Deriving the Update Law

UNC CHARLOTTE

Newton's method is predicated on deriving a **local quadratic approximation** of $J(\mathbf{u})$ in the vicinity of $\mathbf{u_k}$ (where $\mathbf{u_k}$ is the estimate of $\mathbf{u}^*$ at iteration k).

To derive the quadratic approximation, we do a second-order Taylor series expansion of $J(\mathbf{u})$ in the vicinity of $\mathbf{u_k}$:

$$J(\mathbf{u_{k+1}}) = J(\mathbf{u_k}) + \nabla J(\mathbf{u_k})(\mathbf{u_{k+1}} - \mathbf{u_k}) + \frac{1}{2}(\mathbf{u_{k+1}} - \mathbf{u_k})^T H(\mathbf{u_k})(\mathbf{u_{k+1}} - \mathbf{u_k})$$

Differentiating $J(\mathbf{u_{k+1}})$ with respect to $\mathbf{u_{k+1}}$ and setting the derivative (gradient) equal to 0 yields the condition to the minimizer:

$$\nabla J(\mathbf{u_k})^T + H(\mathbf{u_k})(\mathbf{u_{k+1}} - \mathbf{u_k}) = 0$$

The resulting update law is:

$$\mathbf{u_{k+1}} = \mathbf{u_k} - (H(\mathbf{u_k}))^{-1} \nabla J(\mathbf{u_k})^T$$

Newton's Method – Deriving the Update Law

UNC CHARLOTTE

$$\nabla J(\underline{u}_{k+1})^T + H(\underline{u}_k)(\underline{u}_{k+1} - \underline{u}_k) = 0$$

$$H(\underline{u}_k)(\underline{u}_{k+1} - \underline{u}_k) = -\nabla J(\underline{u}_k)^T$$

$$\Rightarrow \underline{u}_{k+1} - \underline{u}_k = -H^{-1}(\underline{u}_k)\nabla J(\underline{u}_k)^T$$

$$\Rightarrow \underline{u}_{k+1} = \underline{u}_k - H^{-1}(\underline{u}_k)\nabla J(\underline{u}_k)^T$$

(Pure) Newton step

Newton's Method – Deriving the Update Law

UNC CHARLOTTE

Newton's method is an **iterative** algorithm that starts with an initial guess, $\mathbf{u}_0$, and adjust this initial guess at each iteration **based on a locally quadratic approximating of** $J(\mathbf{u})$, to converge to the minimizer of $J(\mathbf{u})$ (which we have denoted by $\mathbf{u}^*$).

Algorithm:

Specify $\mathbf{u_0}$, $\Delta u_{convergence}$, and α
Set $k = 0$
Set Δu to something big

while $\Delta u \geq u_{convergence}$

$$\mathbf{u}_{k+1} = \mathbf{u}_k - (H(\mathbf{u_k}))^{-1} \nabla J(\mathbf{u_{k+1}})^T$$

$$\Delta u = \|\mathbf{u_{k+1}} - \mathbf{u_k}\|$$

$$k = k + 1$$

end

Update law – Move to the value of u that minimizes the local quadratic approximation

Important notation:

$\Delta u_{convergence}$ = convergence threshold (how close do successive estimates need to lie in order to terminate the search?)

A Brief Diversion – Analogies Between Newton's Method for Root Finding and Newton's Method for Minimizing $J(u)$ when u is a Scalar

UNC CHARLOTTE

Newton's method for minimization:

Approximation: $$J(u_{k+1}) = J(u_k) + \left.\frac{dJ}{du}\right|_{u_k}(u_{k+1} - u_k) + \left.\frac{d^2J}{du^2}\right|_{u_k}(u_{k+1} - u_k)^2$$

Derivative approximation:

$$\frac{dJ}{du_{k+1}} = \left.\frac{dJ}{du}\right|_{u_k} + \left.\frac{d^2J}{du^2}\right|_{u_k}(u_{k+1} - u_k)$$

Update law:

$$u_{k+1} = u_k - \frac{\left.\frac{dJ}{du}\right|_{u_k}}{\left.\frac{d^2J}{du^2}\right|_{u_k}}$$

Newton's method for root finding:

Approximation: Given $f(x)$, the linear approximation is:

$$f(x_{k+1}) = f(x_k) + \left.\frac{df}{dx}\right|_{x_k}(x_{k+1} - x_k)$$

Update law:

$$x_{k+1} = x_k - \frac{f(x_k)}{\left.\frac{dJ}{dx}\right|_{x_k}}$$

Note the similarities! (different by one derivative)

Key points:

- Finding the minimizer of $J(u)$ is equivalent to finding the root of $\frac{dJ}{du}$
- When we make a **quadratic** approximation for (non-quadratic) $J(u)$, that is ***equivalent*** to making a **linear** approximation for $\frac{dJ}{du}$.

Newton's Method – Design Optimization Example 1

UNC CHARLOTTE

Objective function: $J(\mathbf{u}) = 4u_1^2 + 3u_1u_2 + u_2^2$

Tasks:

- Compute $\nabla J(\mathbf{u})$ and $H(\mathbf{u})$
- Compute $\mathbf{u}^*$ numerically using Newton's method
- How many iterations did it take to converge to $\mathbf{u}^*$? Why is this not surprising when looking at the structure of $J(\mathbf{u})$?

Sample code provided on Canvas

Newton's Method – Design Optimization Example 2

UNC CHARLOTTE

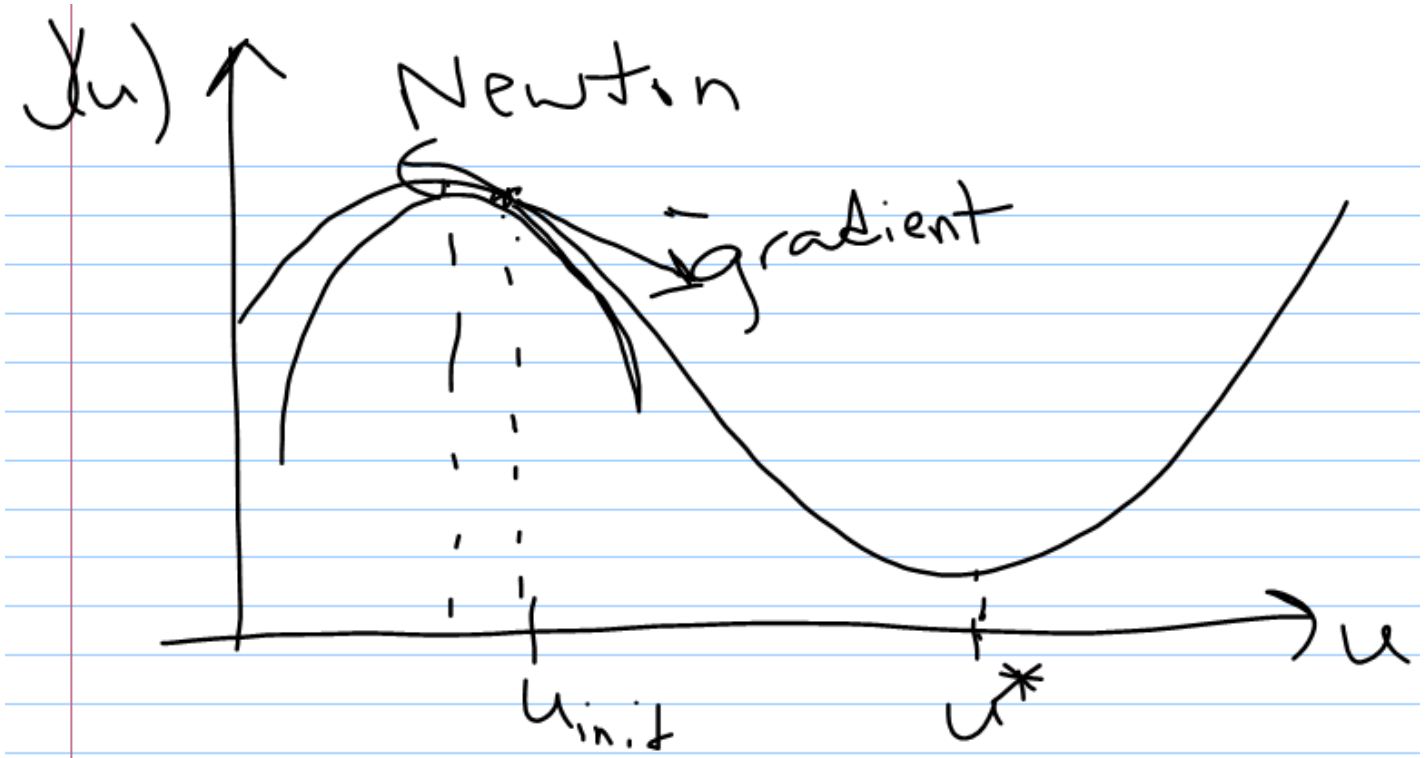
Objective function: $J(\mathbf{u}) = u_1 + u_2 + u_1 e^{-u_2} + u_2^2 e^{-u_1}$

Task: ***Try*** to compute $\mathbf{u}^*$ numerically using Newton's method…do any issues arise?

Sample code provided on Canvas

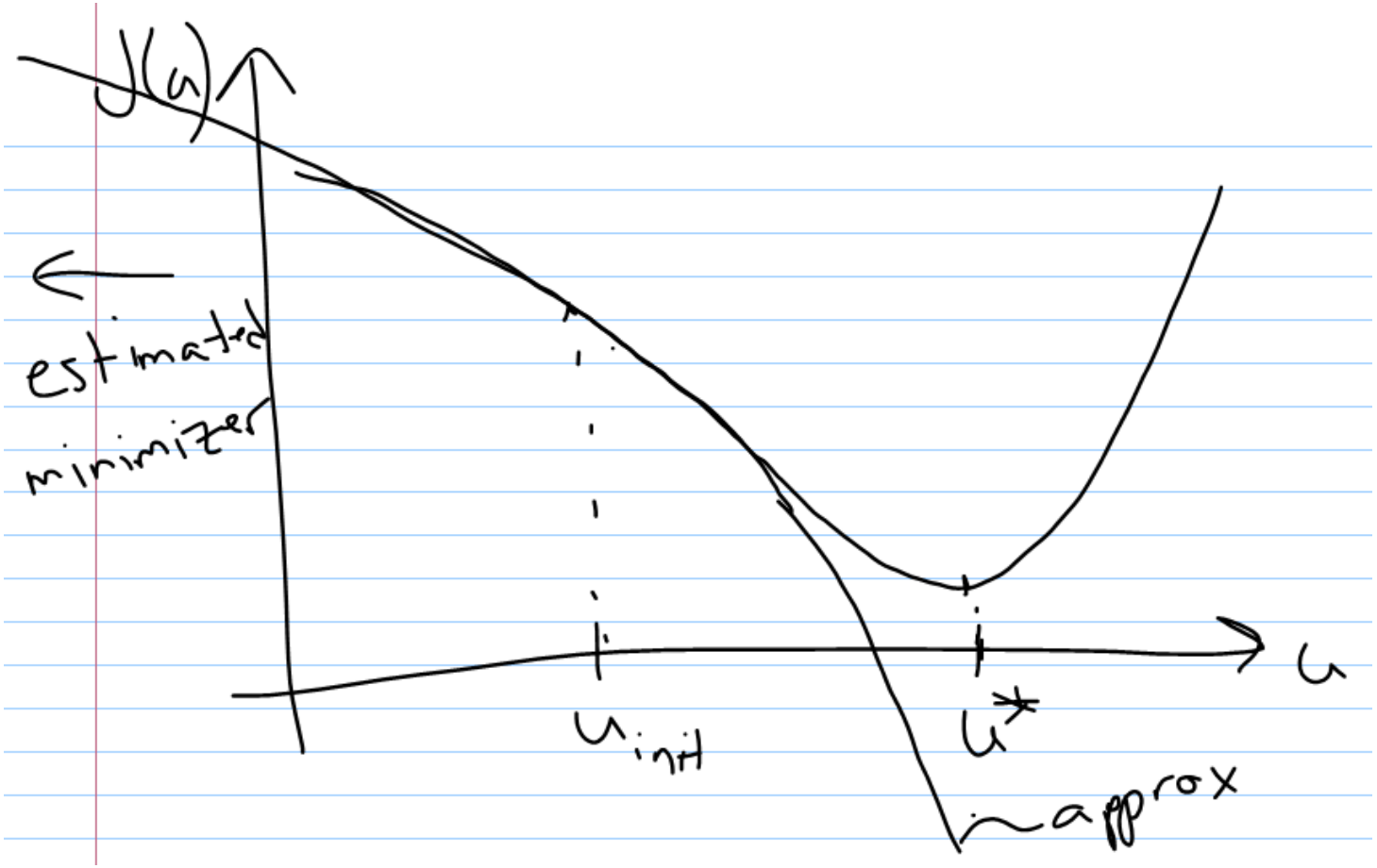
Newton's Method – Design Optimization Example 2

Efficacy of gradient descent & Newton's method depend on initial guess (unless in special case)



Newton's method takes you in the wrong direction in the example to the left.

Newton's Method – Design Optimization Example 2



Newton's method becomes really problematic when the objective function is locally nearly linear. This is the case over much of the domain for example 2.

Gradient Descent – Optimal Control Example

UNC CHARLOTTE

Consider the following discrete-time system model: $x(k+1) = x(k) + u(k)$

Objective: Minimize $J\big(\mathbf{u}; x(0)\big) = \sum_{i=0}^{2} [x(i)^2 + u(i)^2]$

Given: $x(0) = 10$

Tasks:

- Determine how many iterations Newton's method should require to converge to $\mathbf{u}^*$. How can you be sure of your answer?
- Use Newton's method to compute the optimal control trajectory from an initial guess.

Sample code provided on Canvas

Preview of next lecture (and beyond)

UNC CHARLOTTE

Topics for lecture 7:

- Generalized search directions for unconstrained optimizations
- Line searches and modified Newton's method
- Trust region

Beyond lecture 7, we will examine several different techniques for *constrained* convex optimization

- Linear and quadratic programming
- Sequential quadratic programming (SQP)
- Dynamic programming for global optimization