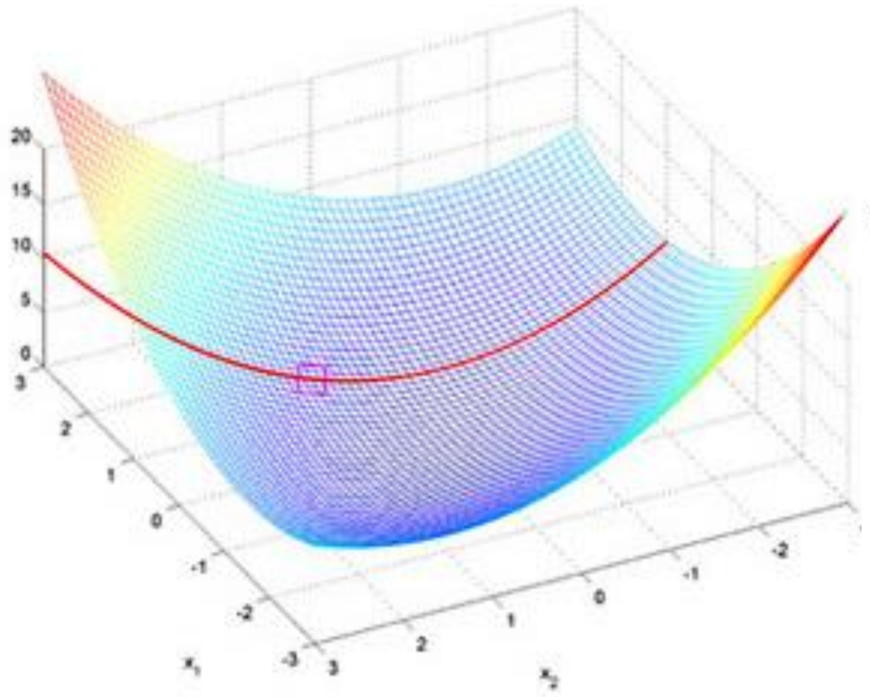
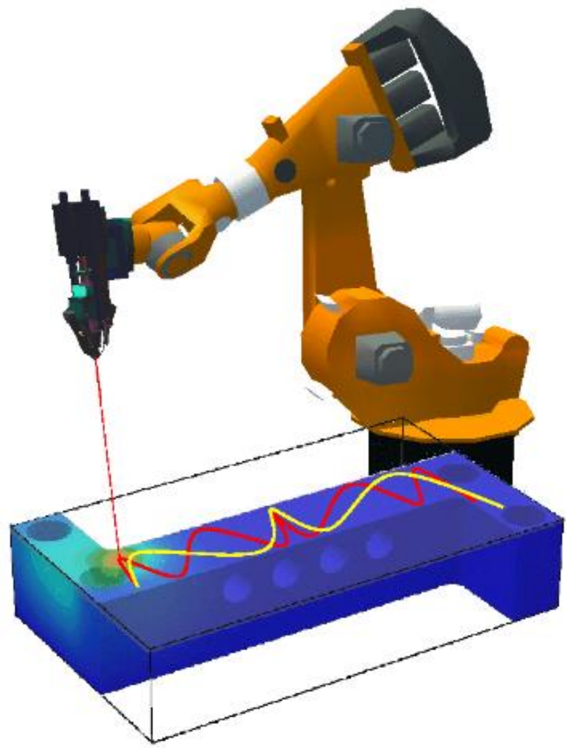
MEGR 3090/7090/8090: Advanced Optimal Control



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

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

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



Lecture 7

September 12, 2017

Summary of Basic (non-adaptive) Gradient Descent and Pure Newton's Method

Basic problem: $\mathbf{u}^* = \arg\min_{\mathbf{u}} J(\mathbf{u})$ subject to: $\mathbf{u} \in \mathbb{R}^{\dim(\mathbf{u})}$

Non-adaptive gradient descent update law:

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

Summary and observations:

- Gradient descent makes a ***linear approximation*** of the objective function and moves in the direction of steepest descent
- Amount of movement in each iteration can be tuned through selection of α
- In some cases, it will be useful to adjust α from one iteration to the next

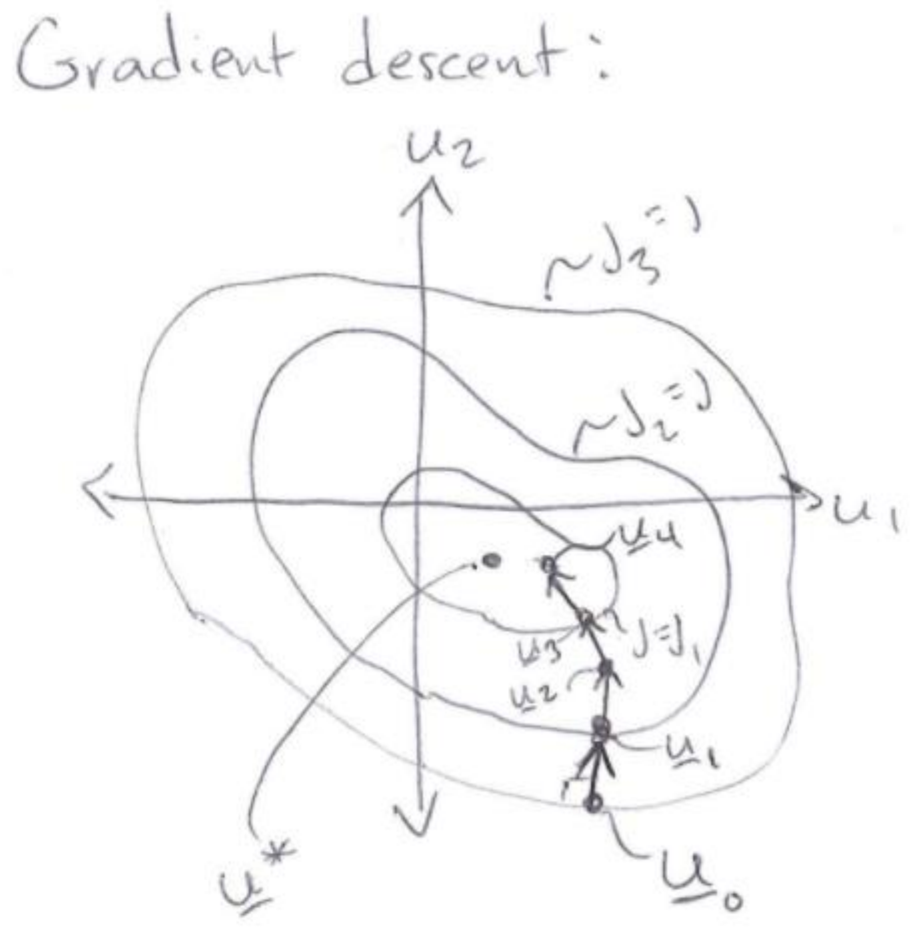
Pure Newton's method update law:

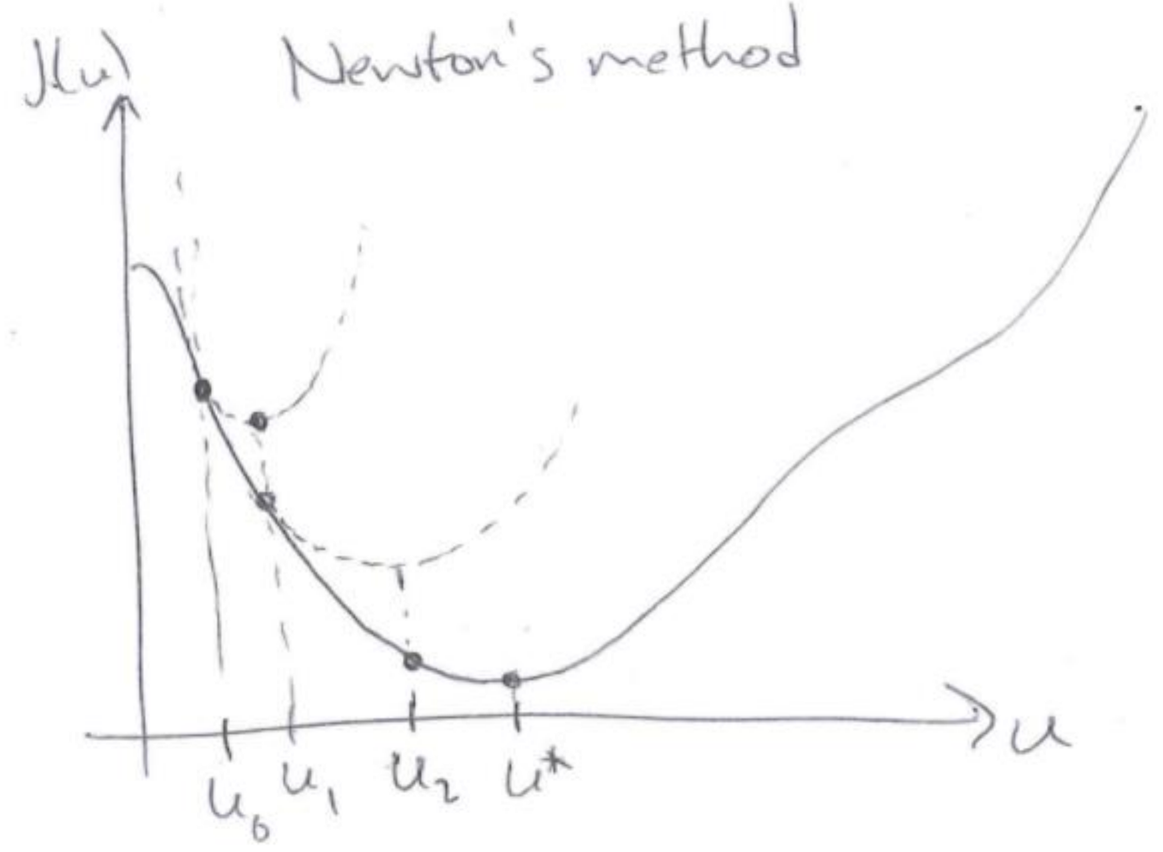
$$\mathbf{u_{k+1}} = \mathbf{u_k} - \left(H(\mathbf{u_k})\right)^{-1}(\nabla J(\mathbf{u_k}))^T$$

Summary and observations:

- Newton's method makes a ***quadratic approximation*** of the objective function and moves ***all the way*** to the minimum of this approximation at each iteration
- Pure Newton's method ***does not provide any tuning parameters!***
- In some cases, it will be useful to just move ***in the direction*** of the approximated minimum

Summary of Basic (non-adaptive) Gradient Descent and Pure Newton's Method





Complications of Gradient Descent Optimization - Example

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:

- Apply gradient descent with $\alpha = 0.1, \alpha = 0.15, \alpha = 0.2, \alpha = 0.25$, and $\alpha = 0.3$
- For each case, determine (i) whether gradient descent converges and (ii) how quickly it converges…do you see a tradeoff here?

Sample code provided on Canvas

Complications of Gradient Descent Optimization - Example

α	# of iterations
0.1	24
0.15	17
0.2	12
0.25	43
0.3	Not converged

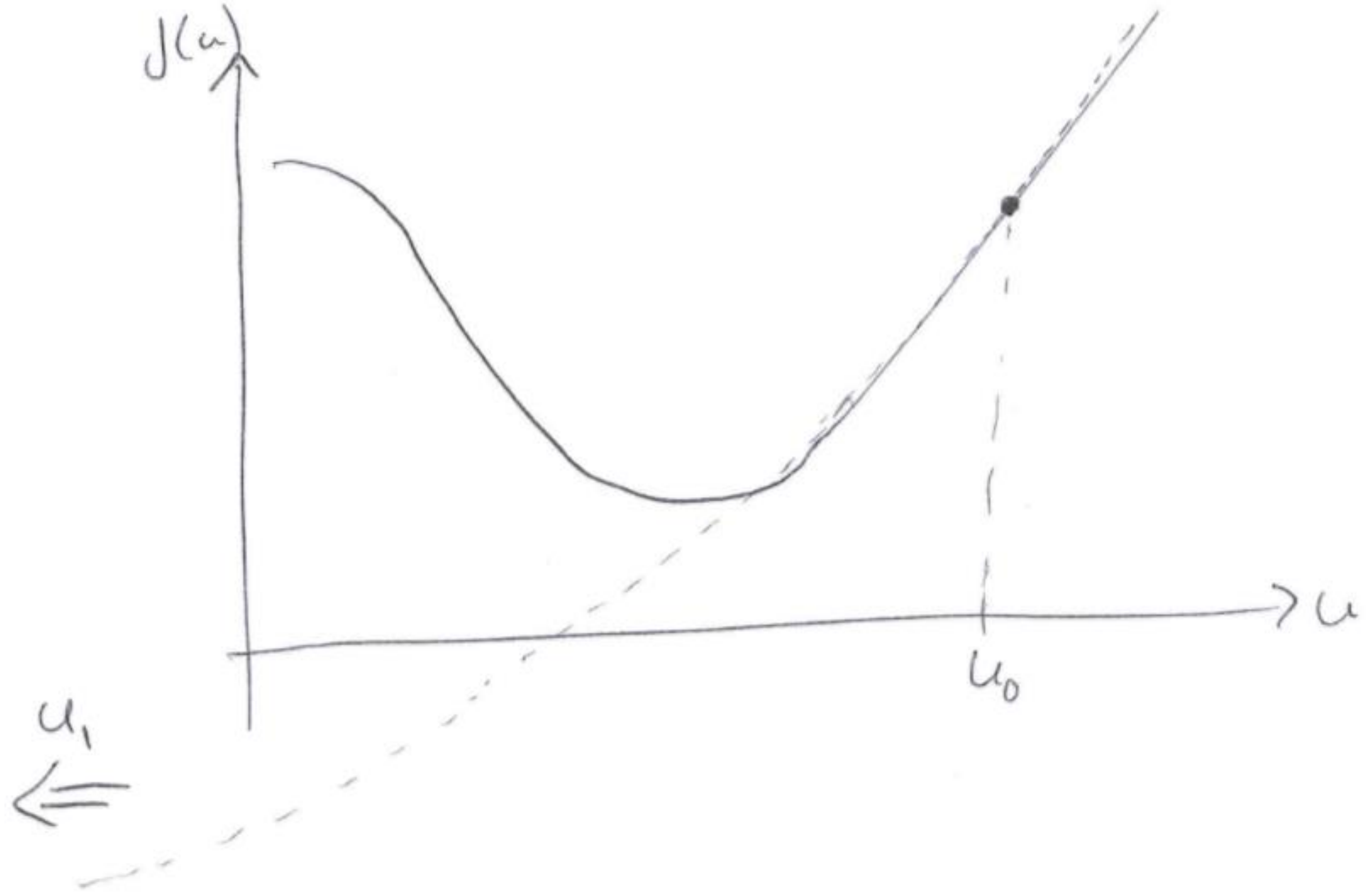
Complications of Newton's Method – Example (Reminder)

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? Examining a plot of $\mathbf{u}$ vs. $J(\mathbf{u})$, can you reason why problems would arise with Newton's method?

Sample code provided on Canvas

Complications of Newton’s Method – Example (Reminder)



Newton’s method can be problematic when the objective function is locally flat (or close to flat), as in the case in the previous example over parts of the design space.

Modifications to Gradient Descent and Newton's Method - Overview

Option 1: Line search

- Use a local linear approximation (as in gradient descent) or quadratic approximation (as in Newton's method) to determine a ***search direction***
- Perform a 1-dimensional search along this direction to determine the size of step to use

Option 2: Trust region (Newton's method)

- Place a hard limit on the amount of change in the decision variable from one iteration to the next (i.e., require $\|\mathbf{u_{k+1}} - \mathbf{u_k}\| \leq \Delta_k$)
- A formal method for adjusting Δ_k at each iteration will be discussed

Modified Gradient Descent and Newton's Method with a Line Search - Fundamentals

Reminder of non-adaptive gradient descent and pure Newton's method update laws (reminder):

- Non-adaptive gradient descent: $\mathbf{u_{k+1}} = \mathbf{u_k} - \alpha(\nabla J\ (\mathbf{u_k}))^T$
- Pure Newton's method: $\mathbf{u_{k+1}} = \mathbf{u_k} - \left(H(\mathbf{u_k})\right)^{-1}(\nabla J(\mathbf{u_k}))^T$

Line search approach – Step 1: Determine a search direction:

- In both the gradient and Newton approach, the vector from $\mathbf{u_k}$ to $\mathbf{u_{k+1}}$ lies in a particular **direction**, which we will denote as $\mathbf{s_k}$
- For gradient descent, $\mathbf{s_k} = -(\nabla J\ (\mathbf{u_k}))^T$
- For Newton's method, $\mathbf{s_k} = -\left(H(\mathbf{u_k})\right)^{-1}(\nabla J(\mathbf{u_k}))^T$

Modified Gradient Descent and Newton's Method with a Line Search - Fundamentals

Gradient descent: $\underline{u}_{k+1} = \underline{u}_k - \alpha \nabla J(\underline{u}_k)^T$

$\uparrow$ Non-adaptive

$$= \underline{u}_k + \alpha \underline{s}_k^g$$

where $\boxed{\underline{s}_k^g = -\nabla J(\underline{u}_k)^T}$

$\Uparrow$ gradient search direction.

Pure Newton's method: $\underline{u}_{k+1} = \underline{u}_k - [H(\underline{u}_k)]^{-1} \nabla J(\underline{u}_k)^T$

$$= \underline{u}_k + \underline{s}_k^n$$

where $\boxed{\underline{s}_k^n = -[H(\underline{u}_k)]^{-1} \nabla J(\underline{u}_k)^T}$

$\Uparrow$ Newton search direction.

Modified Gradient Descent and Newton's Method with a Line Search - Fundamentals

Reminder of non-adaptive gradient descent and pure Newton's method update laws (reminder):

- Non-adaptive gradient descent: $\mathbf{u_{k+1}} = \mathbf{u_k} - \alpha(\nabla J\,(\mathbf{u_k}))^T$
- Pure Newton's method: $\mathbf{u_{k+1}} = \mathbf{u_k} - \left(H(\mathbf{u_k})\right)^{-1}(\nabla J(\mathbf{u_k}))^T$

Line search approach – Step 2: Given the search direction, modify the update law:

This is the actual line search

- New update law: $\mathbf{u_{k+1}} = \mathbf{u_k} + \alpha_k \mathbf{s_k}$ where $\alpha_k = \arg \min_{0 \le \alpha_k \le \alpha_{max}} J(\mathbf{u_{k+1}})$
- Key points:
  - The objective function value ***cannot increase*** from one step to the next (since $\alpha_k = 0$ is allowable)
  - There are many available methods for performing the 1-D search from 0 to α_{max}
  - For Newton's method, taking $\alpha_{max} = 1$ is common, since this implies that the optimization will not go ***beyond*** the estimated optimal value at the next iteration

Modified Gradient Descent and Newton's Method with a Line Search - Fundamentals

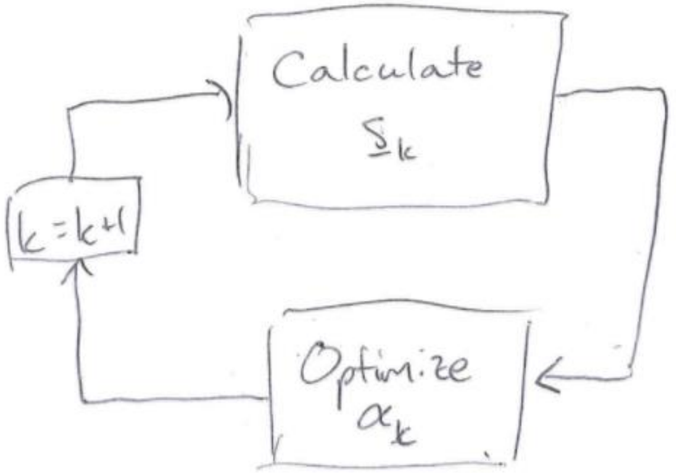
Modified gradient descent & Newton's method:

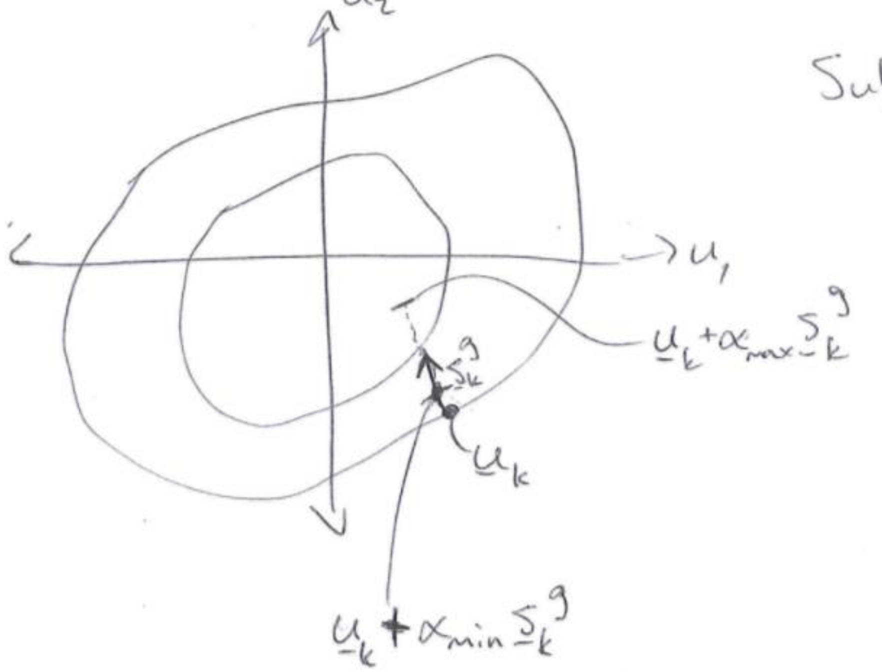
Adaptive gradient descent: $\underline{u}_{k+1} = \underline{u}_k + \alpha_k \underline{s}_k^g$

Modified Newton's method: $\underline{u}_{k+1} = \underline{u}_k + \alpha_k \underline{s}_k^n$

$$\alpha_k = \arg \min_{\alpha} \left[J(\underbrace{\underline{u}_k + \alpha \underline{s}_k}_{\underline{u}_{k+1}}) \right]$$

Subject to: $\alpha_{min} \le \alpha \le \alpha_{max}$




u_2
u_1
$\underline{u}_k + \alpha_{max} \underline{s}_k^g$
$\underline{s}_k^g$
$\underline{u}_k$
$\underline{u}_k + \alpha_{min} \underline{s}_k^g$

Line Search Techniques

Option 1: Gridded Search

- Simply step through the domain of available α
- Guaranteed to result in the optimal α, up to grid resolution

Option 2: 1-D Newton search along the gradient direction

- For the gradient descent approach, once $\mathbf{s_k}$ has been determined, approximate $J(\mathbf{u_{k+1}})$ as quadratic:

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

- Setting the derivative with respect to α_k equal to 0 gives:

$$\alpha_k = \frac{\mathbf{s_k^T} \mathbf{s_k}}{\mathbf{s_k^T} H_k \mathbf{s_k}}$$

Line Search Techniques

Derivation of the “option 2” result:

$$J(\underline{u}_{k+1}) = J(\underline{u}_k) + \underbrace{\nabla J(\underline{u}_k)}_{-\underline{s}_k^T}\underbrace{(\underline{u}_{k+1}-\underline{u}_k)}_{\alpha \underline{s}_k} + \frac{1}{2}\underbrace{(\underline{u}_{k+1}-\underline{u}_k)}_{\alpha \underline{s}_k}^T H(\underline{u}_k)\underbrace{(\underline{u}_{k+1}-\underline{u}_k)}_{\alpha \underline{s}_k}$$

$$= J(\underline{u}_k) - \alpha \underline{s}_k^T \underline{s}_k + \frac{1}{2}\alpha^2 \underline{s}_k^T \, H(\underline{u}_k)\underline{s}_k$$

$$\frac{\partial J(\underline{u}_{k+1})}{\partial \alpha} = -\underline{s}_k^T \underline{s}_k + \alpha \overset{\in \mathbb{R}}{\boxed{\underline{s}_k^T H(\underline{u}_k)\underline{s}_k}} = 0 \ @ \ \alpha^*$$

$$\Rightarrow \alpha^* = \frac{\underline{s}_k^T \underline{s}_k}{\underline{s}_k^T H(\underline{u}_k)\underline{s}_k}$$

Line Search Techniques - Continued

Option 3: Bracketing

- Step 1: Pick two points between $\alpha = \alpha_{min} = 0$ and $\alpha = \alpha_{max}$…denote these by α_1 and α_2
- Step 2: Evaluate $J(\mathbf{u}_k + \alpha_1 \mathbf{s}_k)$ and $J(\mathbf{u}_k + \alpha_2\ \mathbf{s}_k)$
- Step 3:
    - If $J(\mathbf{u}_k + \alpha_1 \mathbf{s}_k) < J(\mathbf{u}_k + \alpha_2\ \mathbf{s}_k)$, set $\alpha_{max} = \alpha_2$ and repeat step 1
    - If $J(\mathbf{u}_k + \alpha_2 \mathbf{s}_k) < J(\mathbf{u}_k + \alpha_1\ \mathbf{s}_k)$, set $\alpha_{min} = \alpha_1$ and repeat step 1

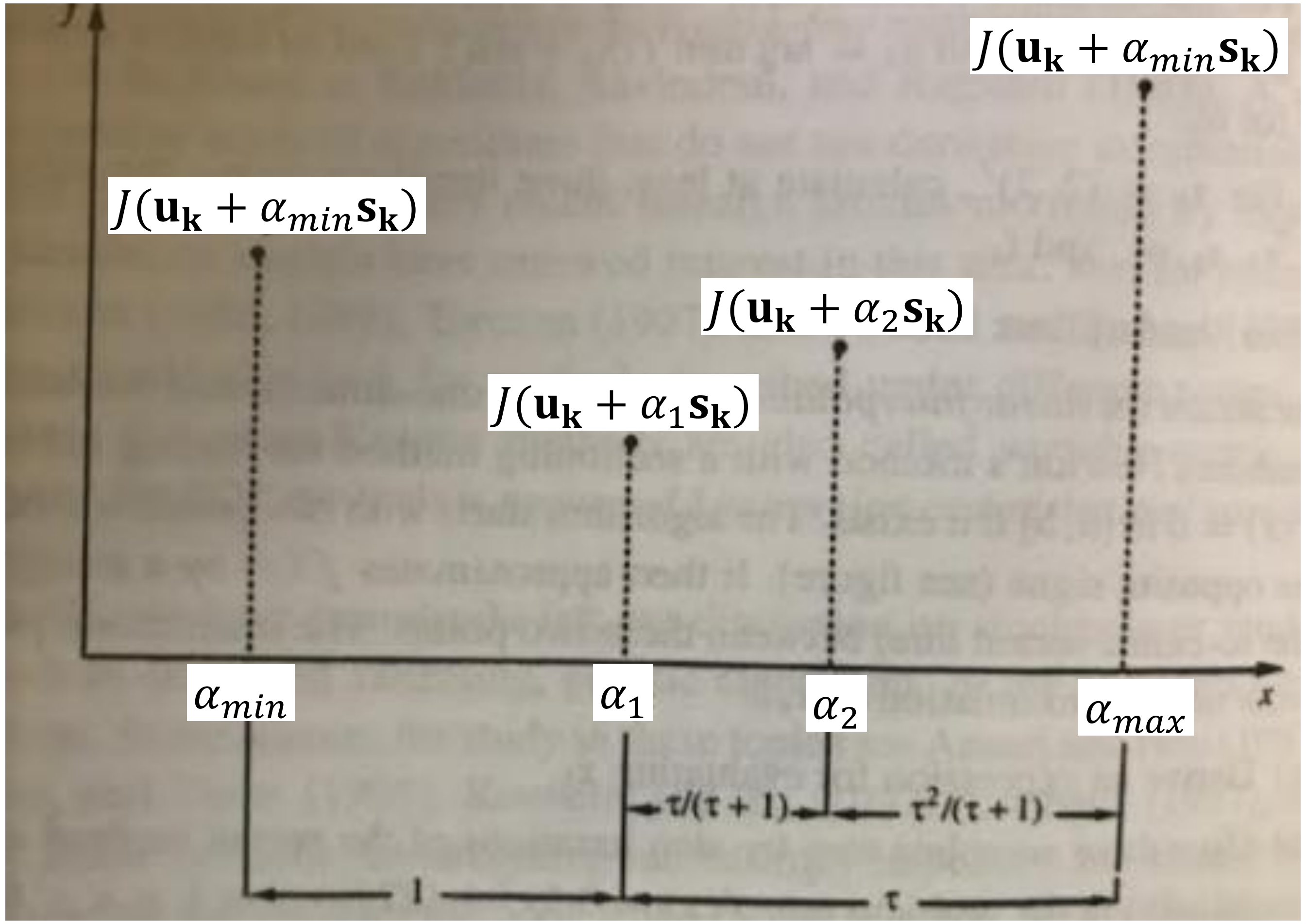
Key challenge – How to select α_1 and α_2 in a non-arbitrary way…One option, shown on the next slide, is a ***golden section search***

Note: Bracketing and 1D Newton searches are **only** guaranteed to converge to the optimum **when a single local minimizer exists** between α_{min} and α_{max}

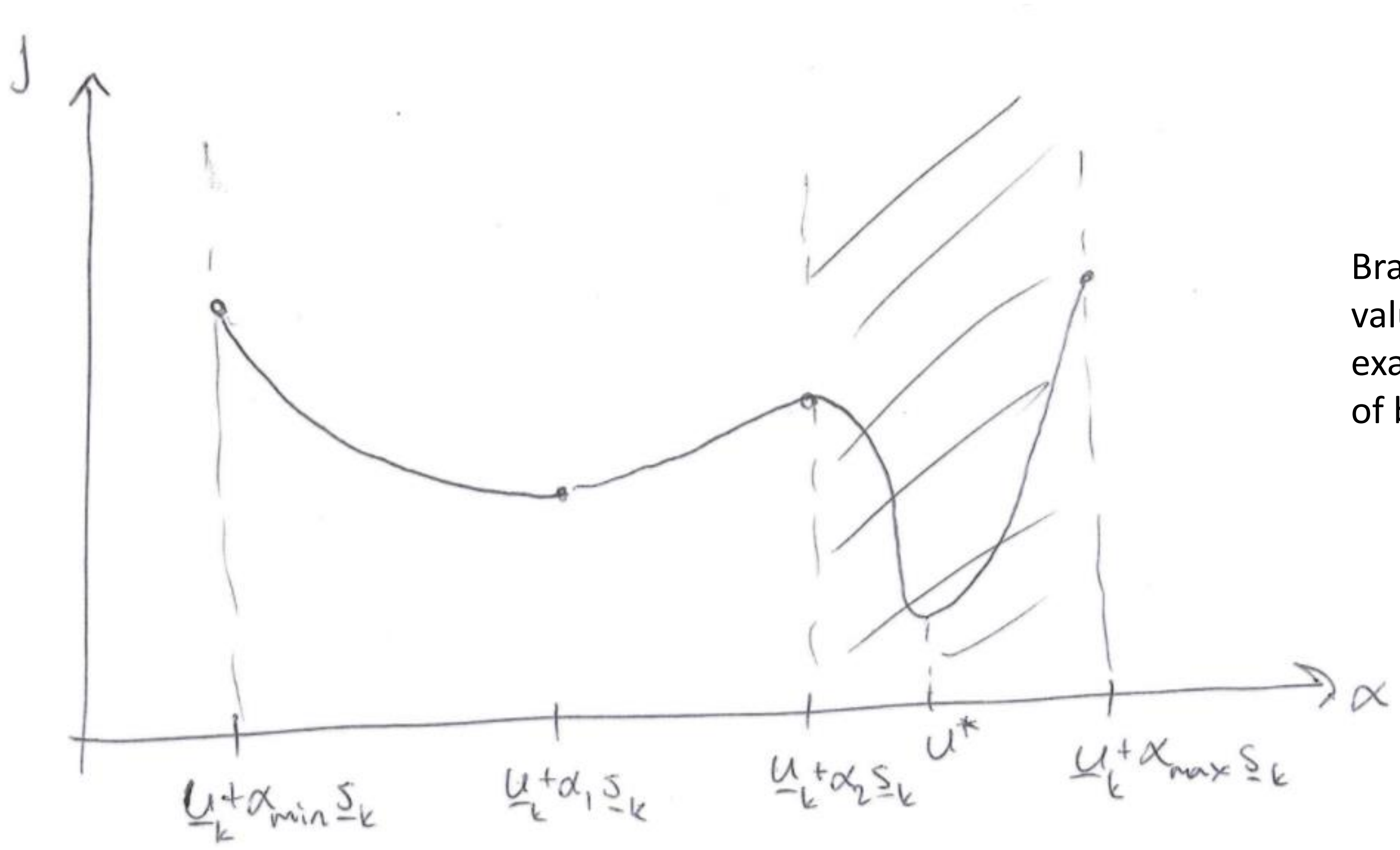
Line Search Techniques - Continued

Key challenge with bracketing – How to select α_1 and α_2 in a non-arbitrary way…One option, shown on here, is a ***golden section search:***

Source: Papalambros textbook, page 332 (modified)


$J(\mathbf{u_k} + \alpha_{min}\mathbf{s_k})$
$J(\mathbf{u_k} + \alpha_{min}\mathbf{s_k})$
$J(\mathbf{u_k} + \alpha_2\mathbf{s_k})$
$J(\mathbf{u_k} + \alpha_1\mathbf{s_k})$
α_{min}
α_1
α_2
α_{max}
x
$\tau/(\tau+1)$
$\tau^2/(\tau+1)$
1
τ

Line Search Techniques - Continued



J
α
$\underline{u}_k + \alpha_{min} \underline{s}_k$
$\underline{u}_k + \alpha_1 \underline{s}_k$
$\underline{u}_k + \alpha_2 \underline{s}_k$
u^*
$\underline{u}_k + \alpha_{max} \underline{s}_k$

Bracketing can discard globally optimal values of α, as is the case in the example to the left. This is a disadvantage of bracketing approaches.

Gradient Descent with Line Search – Example

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:

- Apply gradient descent with a line search (i.e., $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{s}_k$) where $\alpha_k = \arg \min_{0 \le \alpha_k \le \alpha_{max}} J(\mathbf{u_{k+1}})$
- Compare the number of iterations required for convergence to the required number with fixed α

Sample code provided on Canvas

Newton's Method with Line Search – Example

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

Tasks:

- Apply Newton's method with a line search (i.e., $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{s}_k$) where $\alpha_k = \arg \min_{0 \le \alpha_k \le \alpha_{max}} J(\mathbf{u_{k+1}})$
- Compare this result with the pure Newton's method result we obtained earlier

Sample code provided on Canvas

Modified Newton's Method with a Trust Region - Fundamentals

UNC CHARLOTTE

Main idea: Ensure that $\|\mathbf{u_{k+1}} - \mathbf{u_k}\| \leq \Delta$ at every step. ***Note:*** This transforms the **unconstrained** optimization into one in which a **constraint** is applied at each iteration

Mathematical formulation:

- As with pure Newton's method, the objective function is approximated locally as:

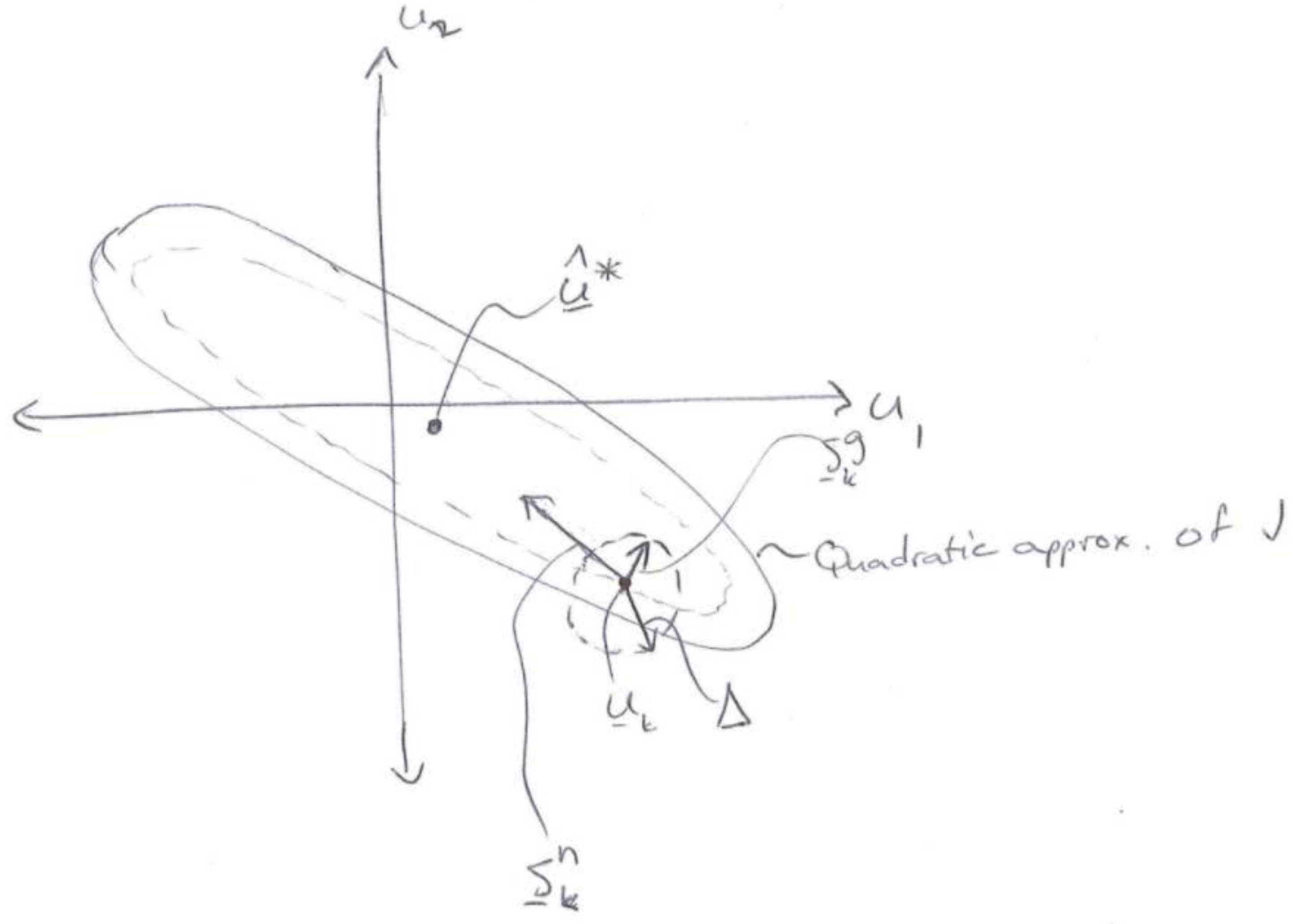
$$J(\mathbf{u_{k+1}}) = J(\mathbf{u_k}) + \nabla J(\mathbf{u_k})(\mathbf{u_{k+1}} - \mathbf{u_k}) + \frac{1}{2}(\mathbf{u_{k+1}} - \mathbf{u_k})^T H(\mathbf{u_k})(\mathbf{u_{k+1}} - \mathbf{u_k})$$

- Taking $\delta\mathbf{u} \triangleq \mathbf{u_{k+1}} - \mathbf{u_k}$, the modified optimization problem at each iteration is:

$$\delta\mathbf{u} = \arg\min \nabla J(\mathbf{u_k})\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T H(\mathbf{u_k})\delta\mathbf{u}$$

Subject to: $\|\delta\mathbf{u}\| \leq \Delta$

Modified Newton's Method with a Trust Region - Fundamentals



Modified Newton's Method with a Trust Region - Fundamentals

Optimization problem – reminder: $\delta\mathbf{u} = \arg\min \nabla J(\mathbf{u_k})\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T H(\mathbf{u_k})\delta\mathbf{u}$

Subject to: $\|\delta\mathbf{u}\| \leq \Delta$

Two possibilities exist:

- ***Possibility 1:*** The minimum occurs when $\|\delta\mathbf{u}\| < \Delta$**.** In this case:

$$\delta\mathbf{u} = -\left(H(\mathbf{u_k})\right)^{-1}\nabla J(\mathbf{u_k})^T$$

- ***Possibility 2:*** The minimum occurs on the boundary, when $\|\delta\mathbf{u}\| < \Delta$**.** In this case:

$$\delta\mathbf{u} = -\left(H(\mathbf{u_k}) + \mu I\right)^{-1}\nabla J(\mathbf{u_k})^T$$

…where μ is a Lagrange multiplier. We will address the problem of solving for μ when we discuss constrained optimization.

Adaptive Trust Region

Optimization problem with adaptive trust region:

$$\delta\mathbf{u} = \arg\min \nabla J(\mathbf{u_k})\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T H(\mathbf{u_k})\delta\mathbf{u}$$

Subject to: $\|\delta\mathbf{u}\| \leq \Delta_k$

Main idea of the adaptive trust region:

- Evaluate the accuracy of the local quadratic approximation by comparing the actual performance improvement to that which is predicted by the approximation
- Adjust the trust region based on the ratio between actual and estimated improvement:

$$r_k = \frac{AI}{EI}$$

where:

r_k = performance improvement ratio
AI = actual performance improvement
EI = estimated performance improvement

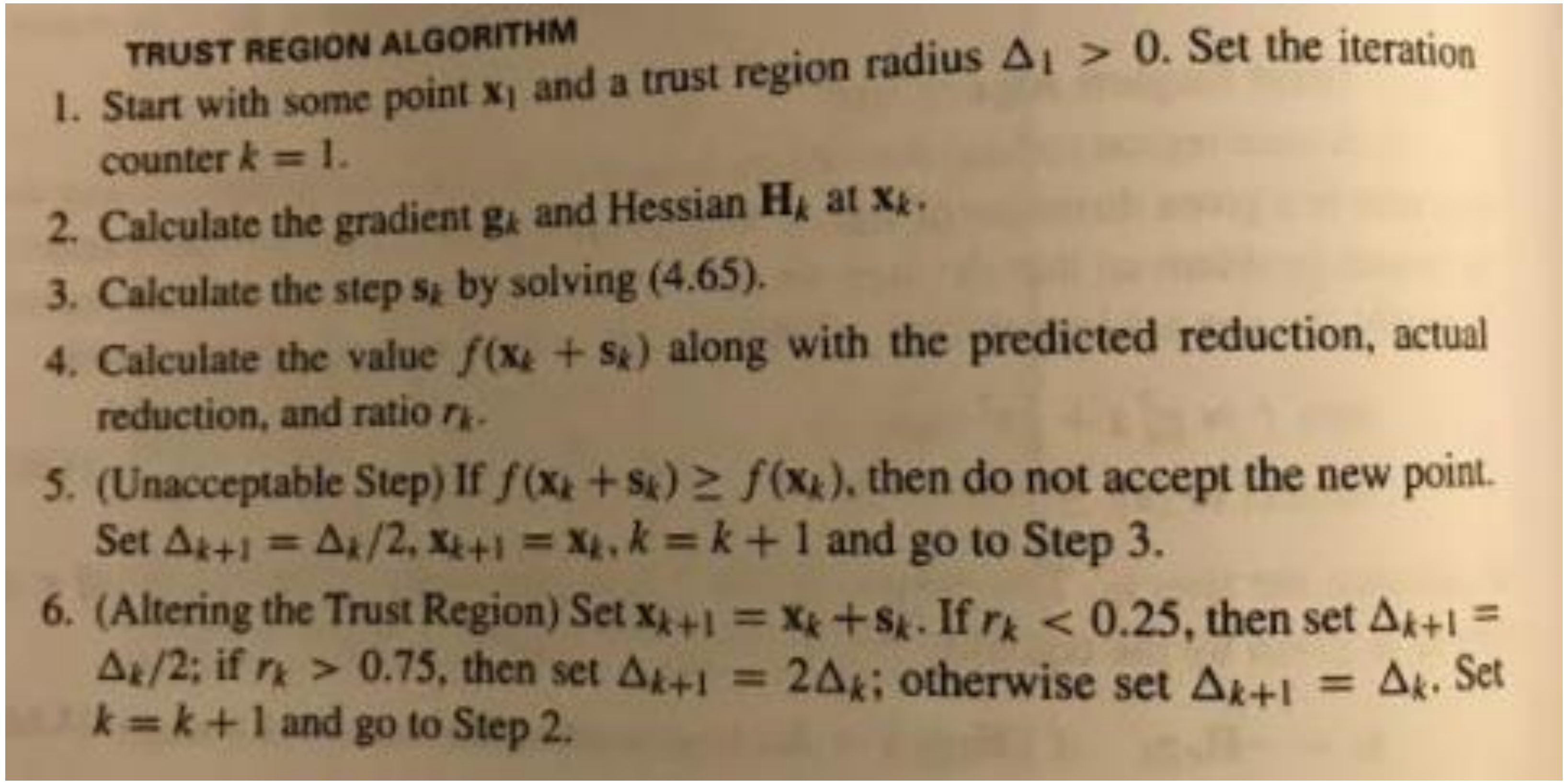
$$AI = J(\mathbf{u_k}) - J(\mathbf{u_k} + \boldsymbol{\delta}\mathbf{u})$$

$$EI = -(\nabla J(\mathbf{u_k})\delta\mathbf{u} + \tfrac{1}{2}\delta\mathbf{u}^T H(\mathbf{u_k})\delta\mathbf{u})$$

Adaptive Trust Region - Continued

A specific algorithm for adjusting the trust region is detailed in Papalambros, with the following variable analogies:

- Papalambros' $\mathbf{x}$ = our $\mathbf{u}$
- Papalambros' $\mathbf{s_k}$ = our $\delta\mathbf{u}$
- Papalambros' $f(\mathbf{x})$ = our $J(\mathbf{u})$



TRUST REGION ALGORITHM

1. Start with some point $\mathbf{x}_1$ and a trust region radius $\Delta_1 > 0$. Set the iteration counter $k = 1$.
2. Calculate the gradient $\mathbf{g}_k$ and Hessian $\mathbf{H}_k$ at $\mathbf{x}_k$.
3. Calculate the step $\mathbf{s}_k$ by solving (4.65).
4. Calculate the value $f(\mathbf{x}_k + \mathbf{s}_k)$ along with the predicted reduction, actual reduction, and ratio r_k.
5. (Unacceptable Step) If $f(\mathbf{x}_k + \mathbf{s}_k) \geq f(\mathbf{x}_k)$, then do not accept the new point. Set $\Delta_{k+1} = \Delta_k/2$, $\mathbf{x}_{k+1} = \mathbf{x}_k$, $k = k + 1$ and go to Step 3.
6. (Altering the Trust Region) Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$. If $r_k < 0.25$, then set $\Delta_{k+1} = \Delta_k/2$; if $r_k > 0.75$, then set $\Delta_{k+1} = 2\Delta_k$; otherwise set $\Delta_{k+1} = \Delta_k$. Set $k = k + 1$ and go to Step 2.

Source: Papalambros textbook, page 162

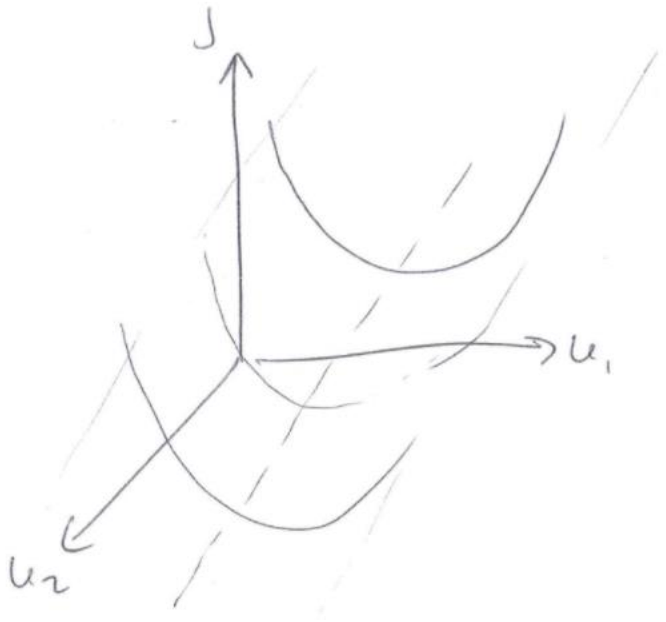
Getting Back to Mitchell's Question About the Interpretation of a Singular Hessian

We all know that a singular Hessian is **bad**, but what does it imply **geometrically**?

Answer: It implies that the quadratic approximation of the response surface **will not have a finite, unique minimizer**

Example situations:

- The quadratic approximation is a ***constant*** (resulting in all points bearing the same objective function value)
- The quadratic approximation is ***linear*** (resulting in an infinite minimizer)
- The quadratic approximation is a ***valley*** (resulting in an infinite minimizer if sloped or a non-unique minimizer if flat)



"The Valley" by Vermillion

Similar quality artwork by Miro – Free, courtesy of Google Images

Preview of next lecture (and beyond)

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