

# ECE276A: Sensing & Estimation in Robotics

## Lecture 3: Color Vision and Parameter Estimation

Instructor:

Nikolay Atanasov: [natanasov@ucsd.edu](mailto:natanasov@ucsd.edu)

Teaching Assistants:

Qiaojun Feng: [qif007@eng.ucsd.edu](mailto:qif007@eng.ucsd.edu)

Tianyu Wang: [tiw161@eng.ucsd.edu](mailto:tiw161@eng.ucsd.edu)

Ibrahim Akbar: [iakbar@eng.ucsd.edu](mailto:iakbar@eng.ucsd.edu)

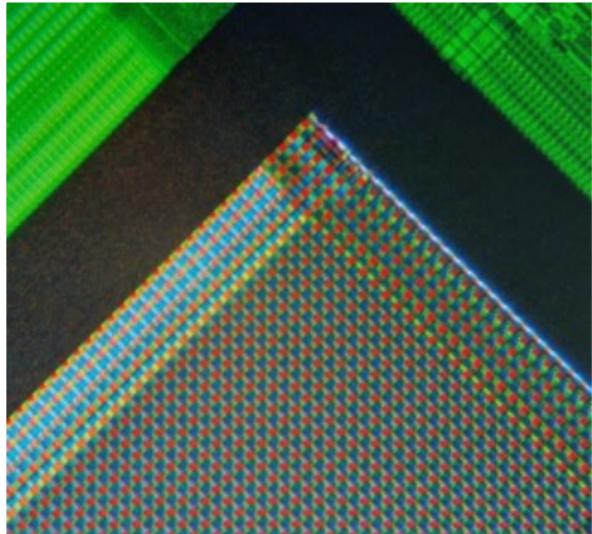
You-Yi Jau: [yjau@eng.ucsd.edu](mailto:yjau@eng.ucsd.edu)

Harshini Rajachander: [hrajacha@eng.ucsd.edu](mailto:hrajacha@eng.ucsd.edu)



# Color Imaging

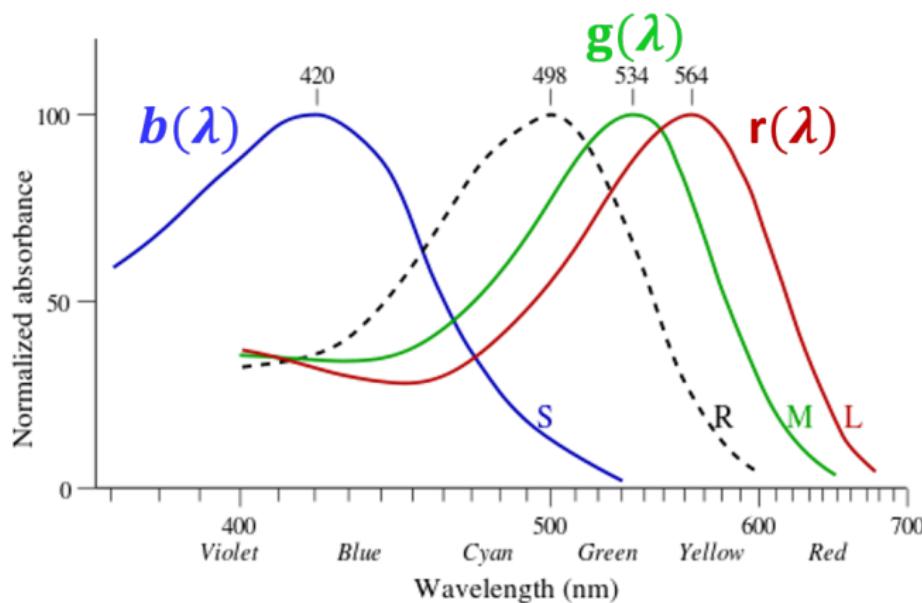
- ▶ Image sensor: converts light into small bursts of current
- ▶ Analog imaging technology uses charge-coupled devices (CCD) or complementary metal-oxide semiconductors (CMOS)
- ▶ CCD/CMOS photosensor array:
  - ▶ A phototransistor converts light into current
  - ▶ Each transistor charges a capacitor to measure:  
**#photons/sampling time**
  - ▶ R,G,B filters are used to modify the absorption profiles of photons
- ▶ Analog-to-digital conversion of R,G,B transistor values to pixel values:



$$\underbrace{R = 127}_{\text{8 bits (0-255)}}, \quad G = 200, \quad B = 103 \quad (\text{24-bit color})$$

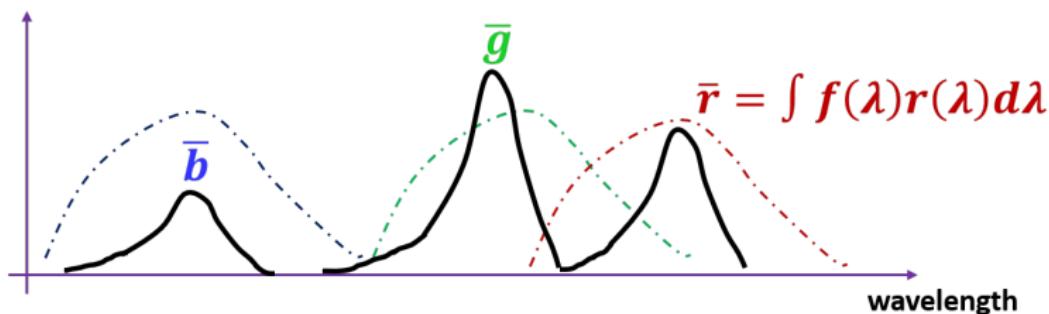
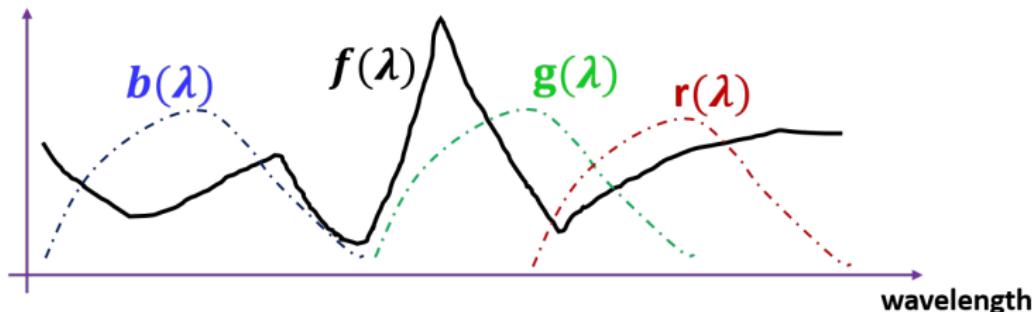
# Why RGB, Why 3?

- ▶ **Retina:** types of photoreceptors: **rod** & **cone** cells (S,M,L)
- ▶ **Rod cells:**
  - ▶ insensitive to wavelength but highly sensitive to intensity
  - ▶ mostly saturated during daylight conditions



## ► Cone cells:

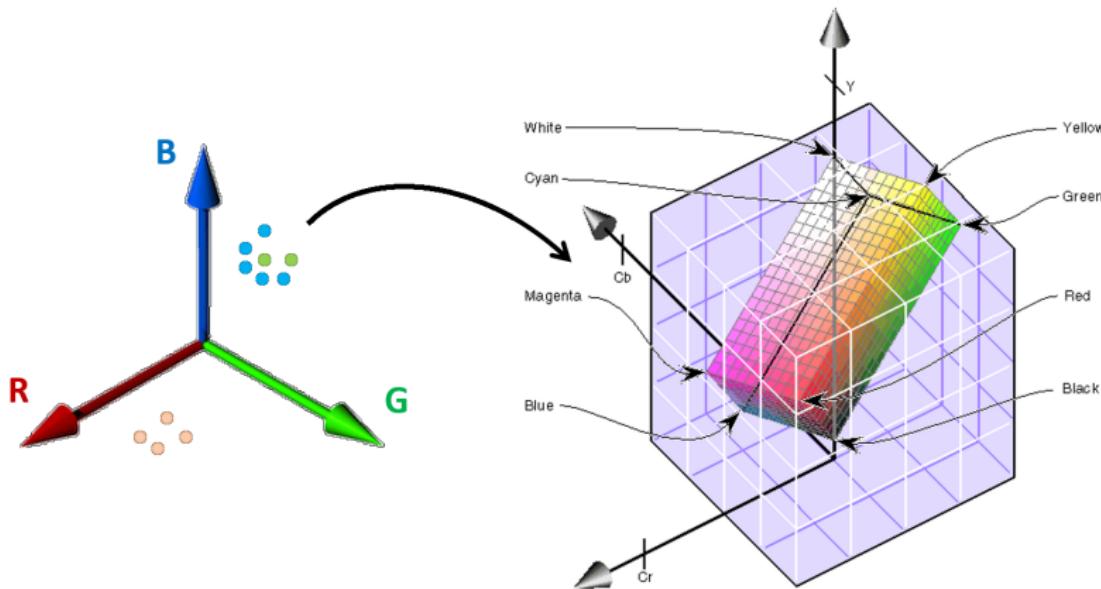
- Given an arbitrary light spectral distribution  $f(\lambda)$ , the cone cells act as filters that provide a **convolution-like signal** to the brain:



- Color blind people are deficient in 1 or more of these cones
- Other animals (e.g., fish) have more than 3 cones

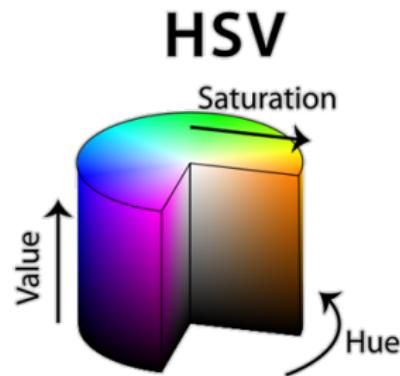
## Luma-Chroma Color Space

- ▶ **YUV (YCbCr)**: a linear transformation of RGB
  - ▶ Luminance/Brightness ( $Y$ )  $\approx (R + G + B)/3$     } **gray-scale image**
  - ▶ Blueness ( $U/Cb$ )  $\approx (B - G)$
  - ▶ Redness ( $V/Cr$ )  $\approx (R - G)$               } **chrominance**
- ▶ Used in analog TV for PAL/SECAM composite color video standards



# HSV and LAB Color Spaces

- ▶ **HSV**: cylindrical coordinates of RGB points
  - ▶ **Hue (H)**: angular dimension (red  $\approx 0^\circ$ , green  $\approx 120^\circ$ , blue  $\approx 240^\circ$ )
  - ▶ **Saturation (S)**: pure red has saturation 1, while tints have saturation  $< 1$
  - ▶ **Value/Brightness (V)**: achromatic/gray colors ranging from black ( $V = 0$ , bottom) to white ( $V = 1$ , top)



- ▶ **LAB**: nonlinear transformation of RGB; device independent
  - ▶ Lightness (L): from black ( $L = 0$ ) to white ( $L = 100$ )
  - ▶ Position between green and red/magenta (A)
  - ▶ Position between blue and yellow (B)

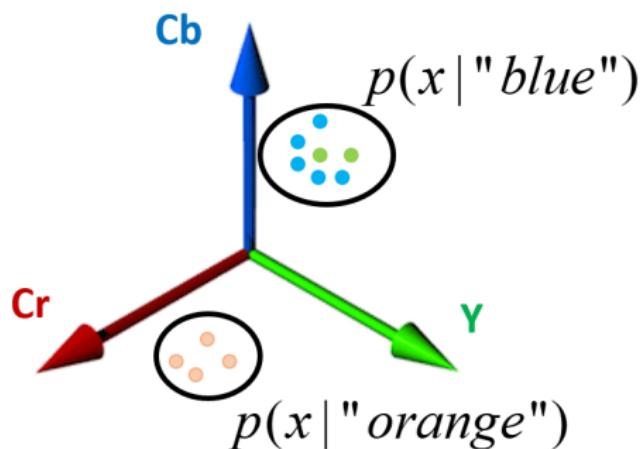
# Image Formation

- ▶ Pixel values depend on:
  - ▶ Scene **geometry**
  - ▶ Scene **photometry** (illumination and reflective properties)
  - ▶ Scene **dynamics** (moving objects)
- ▶ Using camera images to infer a representation of the world is challenging because the shape, material properties, and motion of the observed scene are in general unknown
- ▶ **Color Segmentation:** aims to segment the 3-D color space into a set of discrete volumes:
  - ▶ Each pixel is a **3-D vector**:  $\mathbf{x} = (Y, Cb, Cr)$
  - ▶ Discrete color labels:  $y \in \{1, \dots, N\}$

## Classification Problem

- ▶ Pixel values are noisy
- ▶ Learn a **probabilistic model**  $p(y | \mathbf{x})$  of the color classes  $y$  given color-space **training data**  $D = \{(\mathbf{x}_i, y_i)\}$
- ▶ Define a color map that transforms a color-space input into a discrete color label:

$$\mathbf{x} \xrightarrow{\text{classifier}} \arg \max_y p(y | \mathbf{x})$$



# Color-based Object Detection

Robot Soccer Example: real-time robot vision system



RGB color image at 30 fps from camera

Color Segmentation



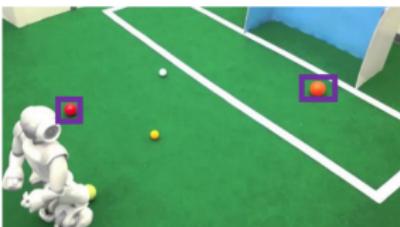
Each pixel is labelled by symbolic colors

Union-find algorithm

Connected components (blobs)

Extract region properties: centroid, bounding box, major/minor axis, etc.

Classify objects based on shape



# Project 1: Color Segmentation

- ▶ Train a probabilistic color model based on a set of training images
- ▶ Use the model to classify the colors on an unseen test image
- ▶ Detect a blue barrel based on the color segmentation (last year was red!)



## Project 1 Tips

- ▶ Define  $K$  color classes, e.g., barrel-blue, not-barrel-blue, brown, green
- ▶ Label examples for each color class to obtain a training dataset  
 $D = \{\mathbf{x}_i, y_i\}$  (use **roipoly**)
- ▶ Train a discriminative  $p(y | \mathbf{x})$  (Logistic Regression) or  $p(y | \mathbf{x})$  generative (Gaussian or Gaussian Mixture) model
- ▶ Given a test image, classify each pixel into one of the  $K$  color classes using your model
- ▶ Find blue regions (use **findContours**)
- ▶ Enumerate blue region combinations and score them based on “barrelness” (use **regionprops**)
- ▶ Experiment with different colorspaces and parameters

## Example: findContours and regionprops

- ▶ Use the openCV function “findContours” to combine individual pixels into blue regions:

```
import numpy as np  
import cv2  
im = cv2.imread('test.jpg')  
binaryIm = myBlueDetector(im)  
contours, hierarchy = cv2.findContours(binaryIm,  
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

- ▶ Enumerate blue region combinations and score them based on “barrelness” using **regionprops**

```
from skimage.measure import label, regionprops  
props = skimage.measure.regionprops(contour_mask)
```

# Orange Ball Recognition

- Center of mass:

$$(c_X, c_Y) = \frac{1}{N_p} \sum_p (x_p, y_p)$$

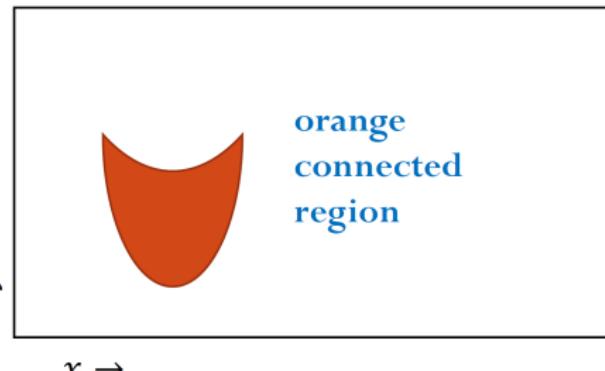
- Fit an ellipse:

$$V_{XX} = \frac{1}{N_p} \sum_p (x_p - c_X)^2$$

$$V_{YY} = \frac{1}{N_p} \sum_p (y_p - c_Y)^2$$

$$V_{XY} = \frac{1}{N_p} \sum_p (x_p - c_X)(y_p - c_Y)$$

Color image:



- Recognize a spherical ball based on thresholds  $\epsilon_0, \epsilon_1$  on the eigenvalues

$$\lambda_0, \lambda_1 \text{ of } \begin{bmatrix} V_{XX} & V_{XY} \\ V_{XY} & V_{YY} \end{bmatrix}$$

- size:  $\min \lambda_1, \lambda_2 \geq \epsilon_0$

- eccentricity:  $1 - \epsilon_1 \leq \frac{\lambda_1}{\lambda_2} \leq 1 + \epsilon_1$

## Supervised Learning

- ▶ Given **iid** training data  $D := \{\mathbf{x}_i, y_i\}_{i=1}^n$  of examples  $\mathbf{x}_i \in \mathbb{R}^d$  with associated labels  $y_i \in \mathbb{R}$  (often also written as  $D = (X, \mathbf{y})$ ), generated from an unknown joint pdf
- ▶ **Goal:** learn a function:  $h : \mathbb{R}^d \rightarrow \mathbb{R}$  that can assign a label  $y$  to a given data point  $\mathbf{x}$ , either from the training dataset  $D$  or from an unseen test set generated from the same unknown pdf
- ▶ The function  $h$  should perform “well”:

- ▶ **Classification** (discrete  $\mathbf{y} \in \{-1, 1\}^n$ ):

$$\min_h Loss_{0-1}(h) := \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{h(\mathbf{x}_i) \neq y_i}$$

- ▶ **Regression** (continuous  $\mathbf{y} \in \mathbb{R}^n$ ):

$$\min_h RMSE(h) := \sqrt{\frac{1}{n} \sum_{i=1}^n (h(\mathbf{x}_i) - y_i)^2}$$

# Generative vs Discriminative Models

## ► Generative model

- ▶  $h(\mathbf{x}) := \arg \max_y p(y, \mathbf{x})$
- ▶ Choose  $p(y, \mathbf{x})$  so that it approximates the unknown data-generating pdf
- ▶ Can generate new examples  $\mathbf{x}$  with associated labels  $y$  by sampling from  $p(y, \mathbf{x})$
- ▶ **Examples:** Naive Bayes, Mixture Models, Hidden Markov Models, Restricted Boltzmann Machines, Latent Dirichlet Allocation, etc.

## ► Discriminative model

- ▶  $h(\mathbf{x}) := \arg \max_y p(y|\mathbf{x})$
- ▶ Choose  $p(y|\mathbf{x})$  so that it approximates the unknown label-generating pdf
- ▶ Because it models  $p(y|\mathbf{x})$  directly, a discriminative model cannot generate new examples  $\mathbf{x}$  but given  $\mathbf{x}$  it can predict (discriminate)  $y$ .
- ▶ **Examples:** Linear Regression, Logistic Regression, Support Vector Machines, Neural Networks, Random Forests, Conditional Random Fields, etc.

## Parameteric Learning

- ▶ Represent the pdfs  $p(y|\mathbf{x}; \omega)$  (discriminative) or  $p(y, \mathbf{x}; \omega)$  (generative) using parameters  $\omega$
- ▶ Estimate/optimize/learn  $\omega$  based on the training set  $D$  in a way that  $\omega^*$  produces good results on a test set
- ▶ Parameter estimation strategies:
  - ▶ **Maximum Likelihood Estimation (MLE):**
    - ▶ Training:  $\omega_{MLE} := \arg \max_{\omega} p(D | \omega)$
    - ▶ Testing:  $p(y^* | \mathbf{x}^*, D) \approx p(y^* | \mathbf{x}^*, \omega_{MLE})$
  - ▶ **Maximum A Posteriori (MAP):**
    - ▶ Training:  $\omega_{MAP} := \arg \max_{\omega} p(\omega | D) = \max_{\omega} p(D | \omega)p(\omega)$
    - ▶ Testing:  $p(y^* | \mathbf{x}^*, D) \approx p(y^* | \mathbf{x}^*, \omega_{MAP})$
  - ▶ **Bayesian Inference:**
    - ▶ Training: estimate  $p(\omega | D) \propto p(D | \omega)p(\omega)$
    - ▶ Testing:  $p(y^* | \mathbf{x}^*, D) = \int p(y^* | \mathbf{x}^*, \omega)p(\omega | D)d\omega$

## Unconstrained Optimization

- The MLE, MAP, and, often, Bayesian Inference approaches lead to an optimization problem of the form:

$$\min_{\omega} J(\omega)$$

### Descent Direction Theorem

Suppose  $J$  is differentiable at  $\bar{\omega}$ . If  $\exists \delta\omega$  such that  $\nabla J(\bar{\omega})^T \delta\omega < 0$ , then  $\exists \epsilon > 0$  such that  $J(\bar{\omega} + \alpha\delta\omega) < J(\bar{\omega})$  for all  $\alpha \in (0, \epsilon)$ .

- The vector  $\delta\omega$  is called a **descent direction**
- The theorem states that if a descent direction exists at  $\bar{\omega}$ , then it is possible to move to a new point that has a lower  $J$  value.
- **Steepest descent direction:**  $\delta\omega := -\frac{\nabla J(\bar{\omega})}{\|\nabla J(\bar{\omega})\|}$
- Based on this theorem, we can derive conditions for determining the optimality of  $\bar{\omega}$

## Optimality Conditions

### First-order Necessary Condition

Suppose  $J$  is differentiable at  $\bar{\omega}$ . If  $\bar{\omega}$  is a local minimizer, then  $\nabla f(\bar{\omega}) = 0$ .

### Second-order Necessary Condition

Suppose  $J$  is twice-differentiable at  $\bar{\omega}$ . If  $\bar{\omega}$  is a local minimizer, then  $\nabla f(\bar{\omega}) = 0$  and  $\nabla^2 f(\bar{\omega}) \succeq 0$ .

### Second-order Sufficient Condition

Suppose  $J$  is twice-differentiable at  $\bar{\omega}$ . If  $\nabla f(\bar{\omega}) = 0$  and  $\nabla^2 f(\bar{\omega}) \succ 0$ , then  $\bar{\omega}$  is a local minimizer.

### Necessary and Sufficient Condition

Suppose  $J$  is differentiable at  $\bar{\omega}$ . If  $J$  is **convex**, then  $\bar{\omega}$  is a global minimizer  
**if and only if**  $\nabla J(\bar{\omega}) = 0$ .

## Descent Optimization Methods

- ▶ Convex unconstrained optimization: just need to solve the equation  $\nabla J(\omega) = 0$  to determine the optimal parameters  $\omega^*$
- ▶ Even if  $J$  is not convex, we can obtain a critical point by solving  $\nabla J(\omega) = 0$
- ▶ However,  $\nabla J(\omega) = 0$  might not be easy to solve explicitly
- ▶ **Descent methods:** iterative methods for unconstrained optimization. Given an initial guess  $\omega^{(k)}$ , take a step of size  $\alpha^{(k)} > 0$  along a certain direction  $\delta\omega^{(k)}$ :

$$\omega^{(k+1)} = \omega^{(k)} + \alpha^{(k)} \delta\omega^{(k)}$$

- ▶ Different methods differ in the way  $\delta\omega^{(k)}$  and  $\alpha^{(k)}$  are chosen but
  - ▶  $\delta\omega^{(k)}$  should be a descent direction:  $\nabla J(\omega^{(k)})^T \delta\omega^{(k)} < 0$  for all  $\omega^{(k)} \neq \omega^*$
  - ▶  $\alpha^{(k)}$  needs to ensure sufficient decrease in  $J$  to guarantee convergence:

$$\alpha^{(k),*} \in \arg \min_{\alpha>0} J(\omega^{(k)} + \alpha \delta\omega^{(k)})$$

Usually  $\alpha^{(k)}$  is obtained via **inexact line search methods**

## Gradient Descent (First Order Method)

- ▶ **Idea:**  $-\nabla_{\omega} J(\omega^{(k)})$  points in the direction of steepest local descent
- ▶ **Gradient descent:** let  $\delta\omega^{(k)} := -\nabla_{\omega} J(\omega^{(k)})$  and iterate:

$$\omega^{(k+1)} = \omega^{(k)} - \alpha^{(k)} \nabla_{\omega} J(\omega^{(k)})$$

- ▶ A good choice for  $\alpha^{(k)}$  is  $\frac{1}{L}$ , where  $L$  is the Lipschitz constant of  $\nabla J$

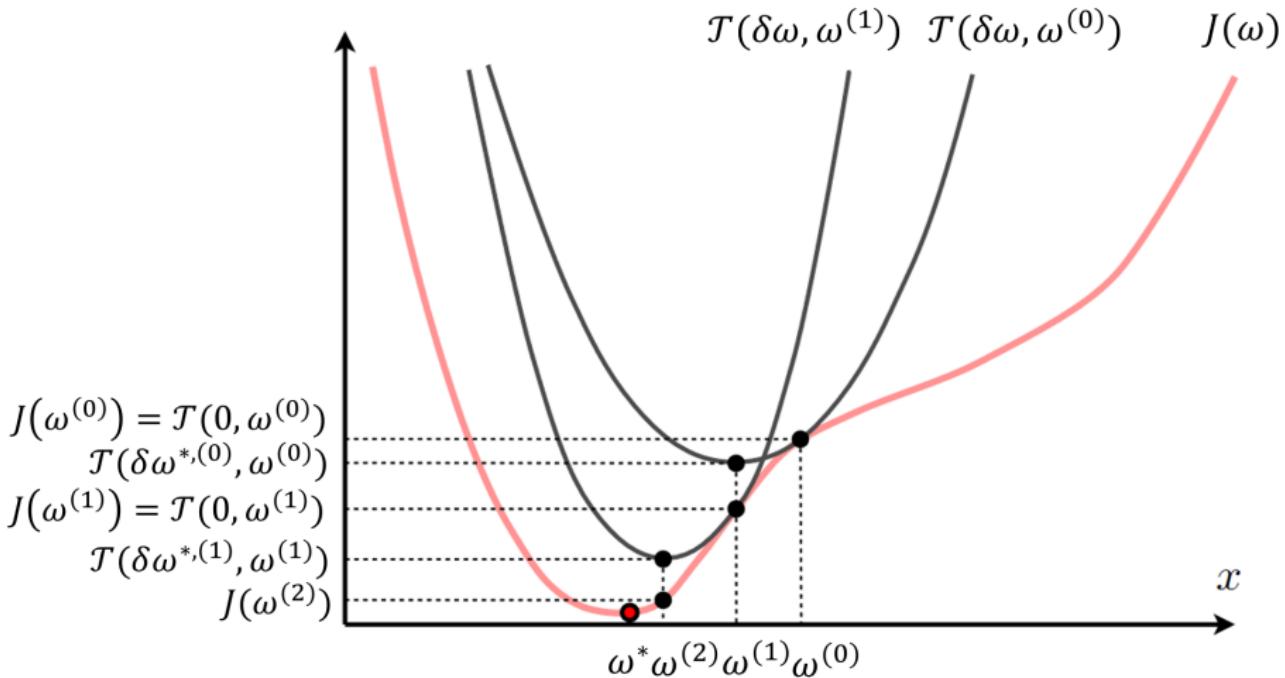
## Newton's Method (Second Order Method)

- ▶ **Newton's method:** iteratively approximates  $J$  by a quadratic function
- ▶ Since  $\delta\omega$  is a 'small' change to the initial guess  $\omega^{(k)}$ , we can approximate  $J$  using a Taylor-series expansion:

$$J(\omega^{(k)} + \delta\omega) \approx J(\omega^{(k)}) + \underbrace{\left( \frac{\partial J(\omega)}{\partial \omega} \Bigg|_{\omega=\omega^{(k)}} \right) \delta\omega}_{\text{Gradient Transpose}} + \frac{1}{2} \delta\omega^T \underbrace{\left( \frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^T} \Bigg|_{\omega=\omega^{(k)}} \right) \delta\omega}_{\text{Hessian}}$$

- ▶ The symmetric Hessian matrix  $\nabla^2 J(\omega^{(k)})$  needs to be positive-definite for this method to work.

## Newton's Method (Second Order Method)



## Newton's Method (Second Order Method)

- ▶ Find  $\delta\omega$  that minimizes the quadratic approximation  $J(\omega^{(k)} + \delta\omega)$
- ▶ Since this is an unconstrained optimization problem,  $\delta\omega^*$  can be determined by setting the derivative with respect to  $\delta\omega$  to zero:

$$\begin{aligned}\frac{\partial J(\omega^{(k)} + \delta\omega)}{\partial \delta\omega} &= \left( \frac{\partial J(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) + \delta\omega^T \left( \frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^T} \Big|_{\omega=\omega^{(k)}} \right) \\ \Rightarrow \quad \left( \frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^T} \Big|_{\omega=\omega^{(k)}} \right) \delta\omega &= - \left( \frac{\partial J(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T\end{aligned}$$

- ▶ The above is a linear system of equations and can be solved when the Hessian is invertible, i.e.,  $\nabla^2 J(\omega^{(k)}) \succ 0$ :

$$\delta\omega^* = - \left[ \nabla^2 J(\omega^{(k)}) \right]^{-1} \nabla J(\omega^{(k)})$$

- ▶ **Newton's method:**

$$\omega^{(k+1)} = \omega^{(k)} - \alpha^{(k)} \left[ \nabla^2 J(\omega^{(k)}) \right]^{-1} \nabla J(\omega^{(k)})$$

## Newton's Method (Comments)

- ▶ Newton's method, like any other descent method, converges only to a **local** minimum
- ▶ **Damped Newton phase:** when the iterates are “far away” from the optimal point, the function value is decreased sublinearly, i.e., the step sizes  $\alpha^{(k)}$  are small
- ▶ **Quadratic convergence phase:** when the iterates are “sufficiently close” to the optimum, full Newton steps are taken, i.e.  $\alpha^{(k)} = 1$ , and the function value converges quadratically to the optimum
- ▶ A **disadvantage** of Newton's method is the need to form the Hessian, which can be numerically ill-conditioned or very computationally expensive in high dimensional problems

## Gauss-Newton's Method

- ▶ **Gauss-Newton** is an approximation to the Newton's method that avoids computing the Hessian. It is applicable when the objective function has the following quadratic form:

$$J(\omega) = \frac{1}{2} \mathbf{u}(\omega)^T \mathbf{u}(\omega)$$

- ▶ The Jacobian and Hessian matrices are:

Jacobian:  $\left. \frac{\partial J(\omega)}{\partial \omega} \right|_{\omega=\omega^{(k)}} = \mathbf{u}(\omega^{(k)})^T \left( \left. \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \right|_{\omega=\omega^{(k)}} \right)$

Hessian: 
$$\begin{aligned} \left. \frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^2} \right|_{\omega=\omega^{(k)}} &= \left( \left. \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \right|_{\omega=\omega^{(k)}} \right)^T \left( \left. \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \right|_{\omega=\omega^{(k)}} \right) \\ &\quad + \sum_{i=1}^M \mathbf{u}_i(\omega^{(k)}) \left( \left. \frac{\partial^2 \mathbf{u}_i(\omega)}{\partial \omega \partial \omega^2} \right|_{\omega=\omega^{(k)}} \right) \end{aligned}$$

## Gauss-Newton's Method

- Near the minimum of  $J$ , the second term in the Hessian is small relative to the first and the Hessian can be approximated according to:

$$\frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^2} \Big|_{\omega=\omega^{(k)}} \approx \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)$$

- The above does not involve any second derivatives and leads to the system:

$$\left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) \delta \omega = - \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \mathbf{u}(\omega^{(k)})$$

- **Gauss-Newton's method:**

$$\omega^{(k+1)} = \omega^{(k)} - \alpha^{(k)} \delta \omega^*$$

## Gauss-Newton's Method (Alternative Derivation)

- ▶ Another way to think about the Gauss-Newton method is to start with a Taylor expansion of  $\mathbf{u}(\omega)$  instead of  $J(\omega)$ :

$$\mathbf{u}(\omega^{(k)} + \delta\omega) \approx \mathbf{u}(\omega^{(k)}) + \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) \delta\omega$$

- ▶ Substituting into  $J$  leads to:

$$J(\omega^{(k)} + \delta\omega) \approx \frac{1}{2} \left( \mathbf{u}(\omega^{(k)}) + \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) \delta\omega \right)^T \left( \mathbf{u}(\omega^{(k)}) + \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) \delta\omega \right)$$

- ▶ Minimizing this with respect to  $\delta\omega$  leads to the same system as before:

$$\left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) \delta\omega = - \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \mathbf{u}(\omega^{(k)})$$

## Levenberg-Marquardt's Method

- ▶ The **Levenberg-Marquardt** modification to the Gauss-Newton method uses a positive diagonal matrix  $\mathbf{D}$  to condition the Hessian matrix:

$$\left( \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right) + \lambda \mathbf{D} \right) \delta \omega = - \left( \frac{\partial \mathbf{u}(\omega)}{\partial \omega} \Big|_{\omega=\omega^{(k)}} \right)^T \mathbf{u}(\omega^{(k)})$$

- ▶ When  $\lambda \geq 0$  is large, the descent vector  $\delta \omega$  corresponds to a very small step in the direction of steepest descent. This helps when the Hessian approximation is poor or poorly conditioned by providing a meaningful direction.