ECE276A: Sensing & Estimation in Robotics
Lecture 3: Color Vision and Parameter Estimation

Instructor:

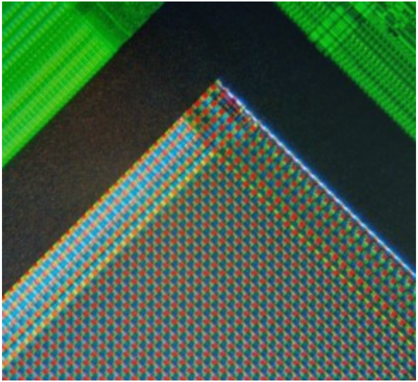
Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Qiaojun Feng: qif007@eng.ucsd.edu
Tianyu Wang: tiw161@eng.ucsd.edu
Ibrahim Akbar: iakbar@eng.ucsd.edu
You-Yi Jau: yjau@eng.ucsd.edu
Harshini Rajachander: hrajacha@eng.ucsd.edu

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Color Imaging



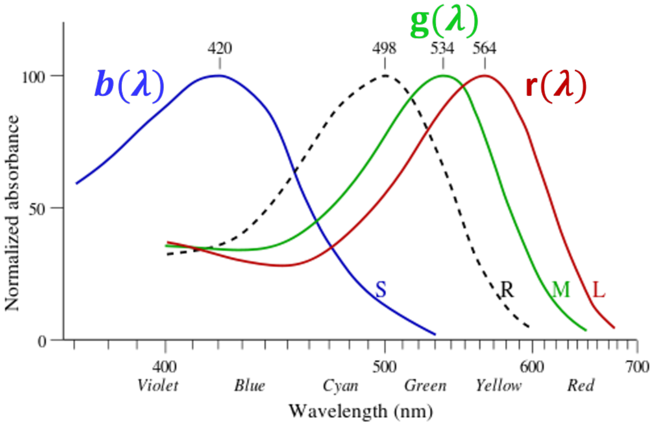
- Image sensor: converts light into small bursts of current
- Analog imaging technology uses charge-coupled devices (CCD) or complementary metal-oxide semiconductors (CMOS)
- CCD/CMOS photosensor array:
  - A phototransistor converts light into current
  - Each transistor charges a capacitor to measure:

    #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:

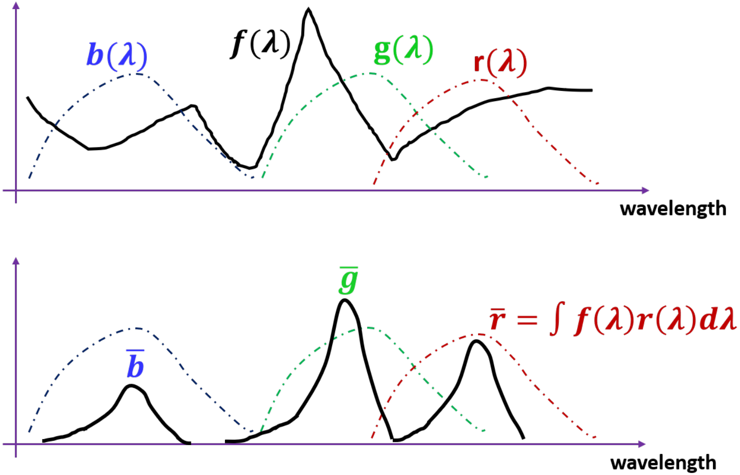
$$\underbrace{R = 127}_{\text{8 bits (0-255)}}, \quad G = 200, \quad B = 103 \quad (\textbf{24-bit color})$$

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



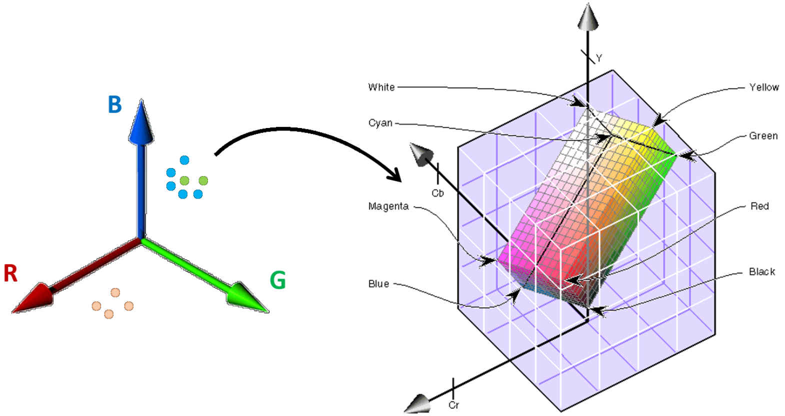
- **Cone cells**:
  - Given an arbitrary light spectral distribution $f(\lambda)$, the cone cells act as filters that provide a **convolution-like signal** to the brain:



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

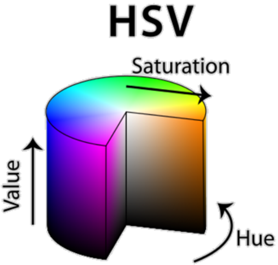
Luma-Chroma Color Space

- **YUV (YCbCr)**: a linear transformation of RGB
  - Luminance/Brightness (Y) $\approx (R + G + B)/3$ } **gray-scale image**
  - Blueness (U/Cb) $\approx (B - G)$ } **chrominance**
  - 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 $\approx 0°$, green $\approx 120°$, blue $\approx 240°$)
  - **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)

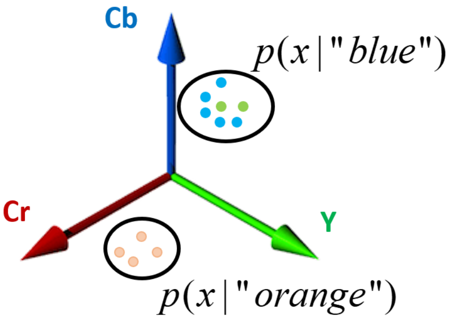
Image Formation

- 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: $y \in \{1, \ldots, N\}$

Classification Problem

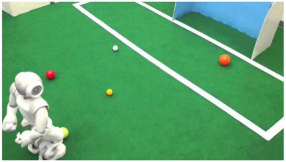
- Pixel values are noisy
- Learn a **probabilistic model** $p(y \mid \mathbf{x})$ of the color classes y given color-space **training data** $D = \{(\mathbf{x}_i, y_i)\}$
- Define a color map that transforms a color-space input into a discrete color label:

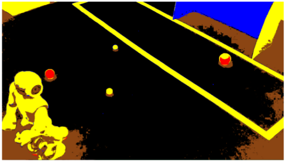
$$\mathbf{x} \xrightarrow{\text{classifier}} \arg\max_{y} p(y \mid \mathbf{x})$$

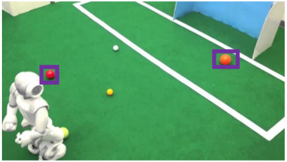


Color-based Object Detection

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 $p(y \mid \mathbf{x})$ (Logistic Regression) or $p(y \mid \mathbf{x})$ 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)
```

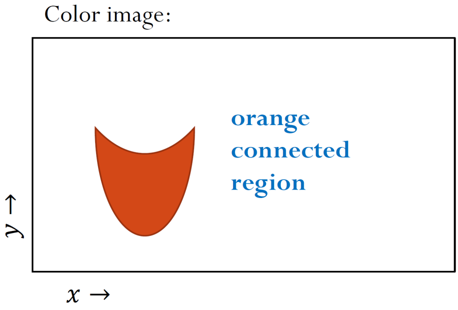
Orange Ball Recognition

- Center of mass:
  $(c_X, c_Y) = \frac{1}{N_p} \sum_p (x_p, y_p)$
- Fit an ellipse:

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

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

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



- Recognize a spherical ball based on thresholds ϵ_0, ϵ_1 on the eigenvalues λ_0, λ_1 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: $h : \mathbb{R}^d \to \mathbb{R}$ 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 ω
- Estimate/optimize/learn ω based on the training set $D = (X, \mathbf{y})$ in a way that ω^* produces good results on a test set
- Parameter estimation strategies:
  - **Maximum Likelihood Estimation (MLE)**: maximize the likelihood of the data D given the parameters ω
  - **Maximum A Posteriori (MAP)**: maximize the likelihood of the parameters ω given the data D
  - **Bayesian Inference**: estimate the whole distribution of the parameters ω given the data D

Parameteric Learning

- **Maximum Likelihood Estimation (MLE)**:

MLE	Discriminative Model	Generative Model
Training	$\omega_{MLE} := \arg\max_{\omega} p(\mathbf{y} \mid X, \omega)$	$\omega_{MLE} := \arg\max_{\omega} p(\mathbf{y}, X \mid \omega)$
Testing	$\arg\max_{y^*} p(y^* \mid \mathbf{x}^*, \omega_{MLE})$	$\arg\max_{y^*} p(y^*, \mathbf{x}^* \mid \omega_{MLE})$

- **Maximum A Posteriori (MAP)**:

MAP	Discriminative Model	Generative Model
Training	$\omega_{MAP} = \arg\max_{\omega} p(\omega \mid \mathbf{y}, X) = \arg\max_{\omega} p(\mathbf{y} \mid X, \omega) p(\omega \mid X)$	$\omega_{MAP} = \arg\max_{\omega} p(\omega \mid \mathbf{y}, X) = \arg\max_{\omega} p(\mathbf{y}, X \mid \omega) p(\omega)$
Testing	$\arg\max_{y^*} p(y^* \mid \mathbf{x}^*, \omega_{MAP})$	$\arg\max_{y^*} p(y^*, \mathbf{x}^* \mid \omega_{MAP})$

- **Bayesian Inference**:

BI	Discriminative Model	Generative Model
Training	$p(\omega \mid \mathbf{y}, X) \propto p(\mathbf{y} \mid X, \omega) p(\omega \mid X)$	$p(\omega \mid \mathbf{y}, X) \propto p(\mathbf{y}, X \mid \omega) p(\omega)$
Testing	$p(y^* \mid \mathbf{x}^*, \mathbf{y}, X) = \int p(y^* \mid \mathbf{x}^*, \omega) p(\omega \mid \mathbf{y}, X) d\omega$	$p(y^*, \mathbf{x}^* \mid \mathbf{y}, X) = \int p(y^*, \mathbf{x}^* \mid \omega) p(\omega \mid \mathbf{y}, X) d\omega$

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 ω^*
- 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 ∇J

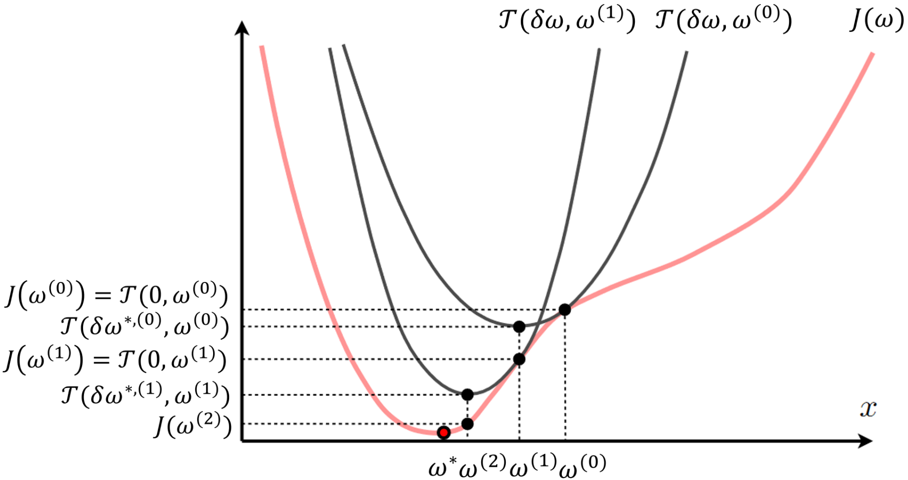
Newton's Method (Second Order Method)

- **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:

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

- 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:

$$\frac{\partial J(\omega^{(k)} + \delta\omega)}{\partial \delta\omega} = \left(\left.\frac{\partial J(\omega)}{\partial \omega}\right|_{\omega=\omega^{(k)}} \right) + \delta\omega^T \left(\left.\frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^T}\right|_{\omega=\omega^{(k)}} \right)$$

$$\Rightarrow \quad \left(\left.\frac{\partial^2 J(\omega)}{\partial \omega \partial \omega^T}\right|_{\omega=\omega^{(k)}} \right) \delta\omega = -\left(\left.\frac{\partial J(\omega)}{\partial \omega}\right|_{\omega=\omega^{(k)}} \right)^T$$

- 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:

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

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

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:

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

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

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

- **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 $J(\omega)$:

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

- Substituting into J leads to:

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

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

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

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:

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

- 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.