ENGR-E 511; ENGR-E 399

“Machine Learning for Signal Processing”

Module 01: Lecture 03:

Optimization

Minje Kim

Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje

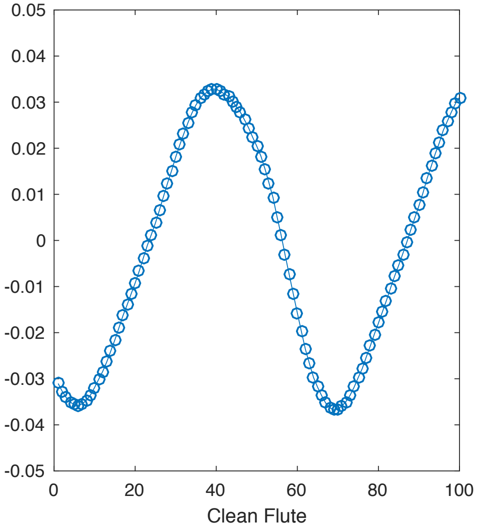
INDIANA UNIVERSITY

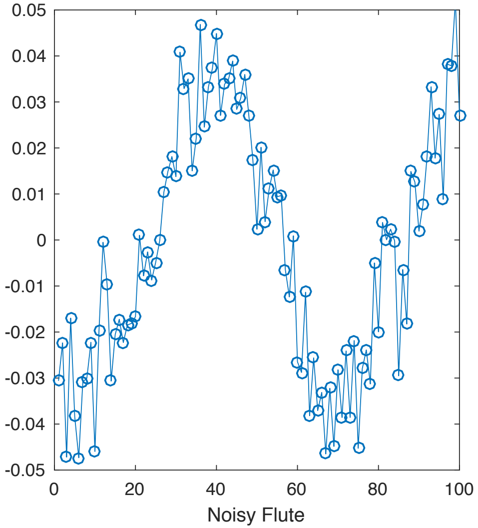
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

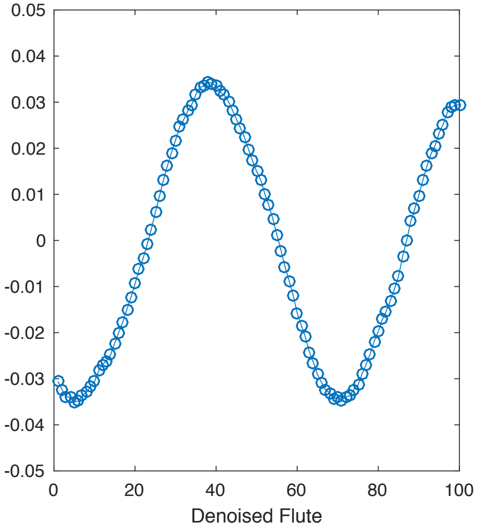
Audio Denoising

- Least mean square

- We want to find out a filter that denoises the noisy input







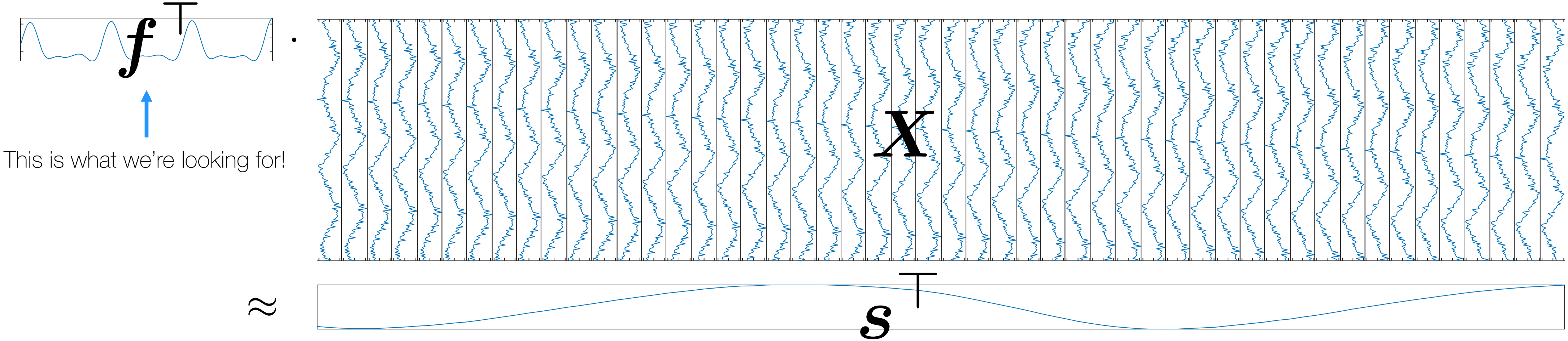
Audio Denoising

- Least mean square

- Let's assume that there is a filter that does this job

$$s[n] \approx f * x[n] = \sum_{\tau=0}^{T-1} f[\tau]x[n-\tau]$$

- In other words



$$\boldsymbol{s}^\top \approx \boldsymbol{f}^\top \cdot \boldsymbol{X}$$

Audio Denoising

- Least mean square

- What we find: the filter that minimizes
  - the error between the source and the reconstruction

$$\arg\min_{\boldsymbol{f}} \mathcal{E}\left(\boldsymbol{s}^\top || \boldsymbol{f}^\top \boldsymbol{X}\right)$$

- Let's use the **mean squared error**

$$\mathcal{E}\left(\boldsymbol{s}^\top || \boldsymbol{f}^\top \boldsymbol{X}\right) = \frac{1}{N}(\boldsymbol{s}^\top - \boldsymbol{f}^\top \boldsymbol{X}) \cdot (\boldsymbol{s}^\top - \boldsymbol{f}^\top \boldsymbol{X})^\top = \frac{1}{N}(\boldsymbol{s}^\top - \hat{\boldsymbol{s}}^\top) \cdot (\boldsymbol{s} - \hat{\boldsymbol{s}}) \qquad (\text{where } \hat{\boldsymbol{s}}^\top = \boldsymbol{f}^\top \boldsymbol{X})$$

$$= \frac{1}{N}\boldsymbol{d}^\top \boldsymbol{d} = \frac{1}{N}\sum_n d_n^2 = \frac{1}{N}\sum_n (s_n - \hat{s}_n)^2 \qquad (\text{where } \boldsymbol{d} = \boldsymbol{s} - \hat{\boldsymbol{s}})$$

$$= \frac{1}{N}\sum_n (s_n - \sum_\tau f_\tau X_{\tau n})^2$$

$$= \frac{1}{N}\sum_n (s_n - \sum_{\tau \neq \tau'} f_\tau X_{\tau n} - f_{\tau'} X_{\tau' n})^2 = \frac{1}{N}\sum_n (C_n - f_{\tau'} X_{\tau' n})^2 \qquad (\text{where } C_n = s_n - \sum_{\tau \neq \tau'} f_\tau X_{\tau n})$$

$$= \frac{1}{N}\sum_n X_{\tau' n}^2 f_{\tau'}^2 - 2C_n f_{\tau'} + C_n^2 = \frac{1}{N}\left(\sum_n X_{\tau' n}^2\right) f_{\tau'}^2 - 2\left(\sum_n C_n\right) f_{\tau'} + \left(\sum_n C_n^2\right)$$

- **The mean squared error is a quadratic function (of one of your filter values)!**

Audio Denoising

- Least mean square

- Some linear algebra

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a} \qquad \frac{\partial \sum_i a_i x_i}{\partial x_i} = a_i \qquad \frac{\partial \boldsymbol{x}^\top \boldsymbol{a}}{\partial \boldsymbol{x}} = \boldsymbol{a}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{x} + \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{x}$$

- Partial derivatives

$$\mathcal{E}\big(\boldsymbol{s}^\top || \boldsymbol{f}^\top \boldsymbol{X}\big) = \frac{1}{N}(\boldsymbol{s}^\top - \boldsymbol{f}^\top \boldsymbol{X}) \cdot (\boldsymbol{s}^\top - \boldsymbol{f}^\top \boldsymbol{X})^\top$$

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}} = \frac{\partial\big(\boldsymbol{s}^\top \boldsymbol{s} - \boldsymbol{f}^\top \boldsymbol{X}\boldsymbol{s} - \boldsymbol{s}^\top \boldsymbol{X}^\top \boldsymbol{f} + \boldsymbol{f}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{f}\big)}{\partial \boldsymbol{f}}$$

$$= \frac{\partial\big(-2\boldsymbol{f}^\top \boldsymbol{X}\boldsymbol{s} + \boldsymbol{f}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{f}\big)}{\partial \boldsymbol{f}}$$

$$= -2\boldsymbol{X}\boldsymbol{s} + 2\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{f} \quad = 0$$

- Therefore: $\arg\min_{\boldsymbol{f}} \mathcal{E}\big(\boldsymbol{s}^\top || \boldsymbol{f}^\top \boldsymbol{X}\big) = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\boldsymbol{s}$

Google “matrix cookbook”

Audio Denoising
- Least mean square

- Now what?
  - We learned a filter that can denoise a particular kind of signal
    - i.e. A flute note corrupted by Gaussian noise
  - Testing on another flute note contaminated with Gaussian noise

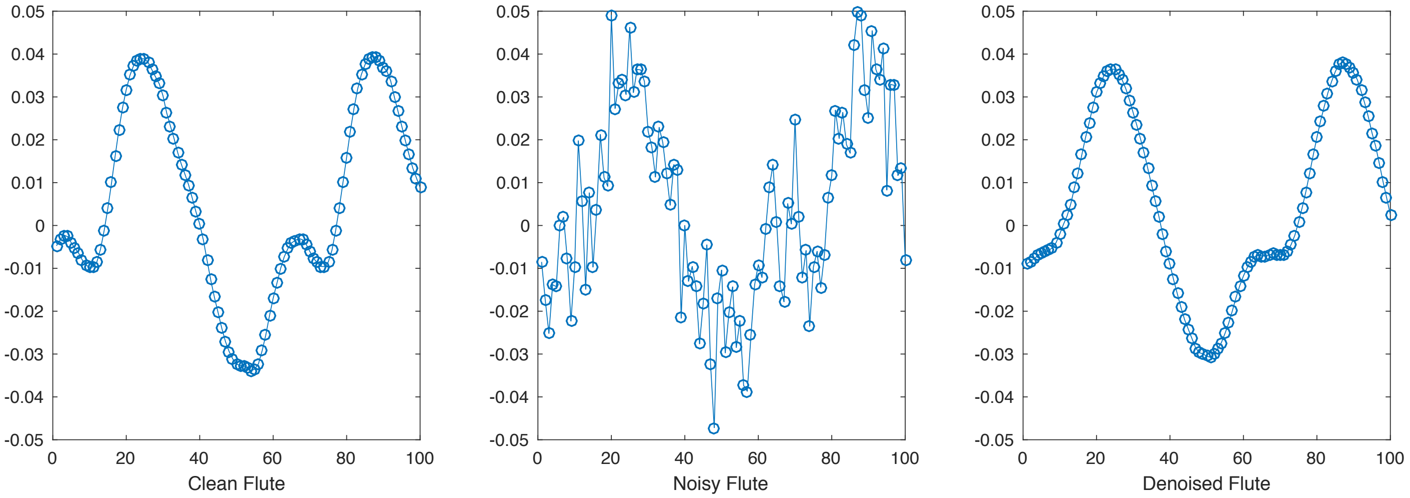


Image Denoising
- Gradient descent

- 2D denoising filter
    - Images are in gray scale
        - Black: 0
        - White: 1
- If you flatten the filter and patches, the procedure boils down to
    - An inner product

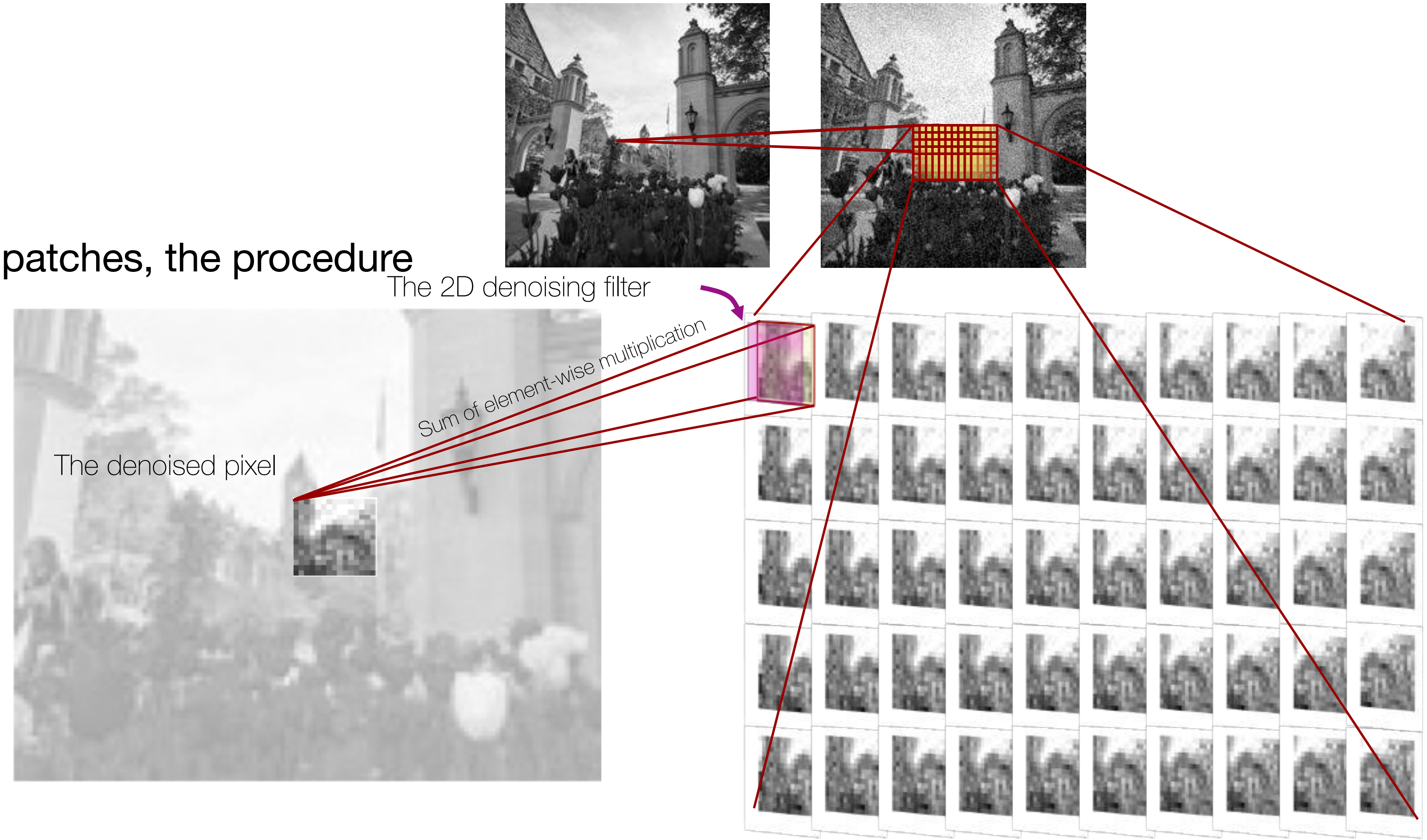


Image Denoising

- Gradient descent

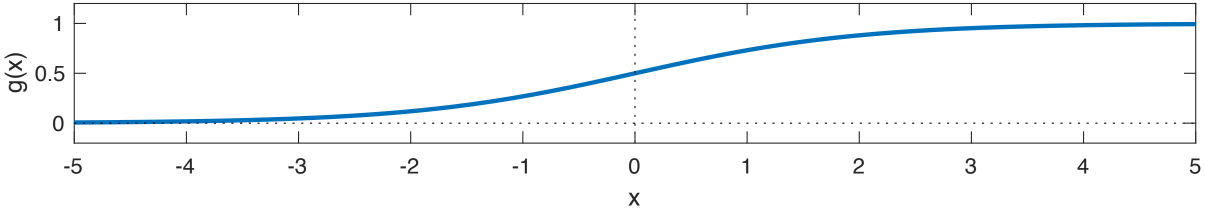
- What I don't like about the error function I used before:

$$\mathcal{E}(\boldsymbol{s}^\top || \boldsymbol{f}^\top \boldsymbol{X}) = \frac{1}{N}(\boldsymbol{s}^\top - \boldsymbol{f}^\top \boldsymbol{X}) \cdot (\boldsymbol{s}^\top - \boldsymbol{f}^\top \boldsymbol{X})^\top$$

  - The reconstructed images pixels should be neither negative nor bigger than 1
  - No way to control this in the original error function

- Logistic function can take care of it

$$g(x) = \frac{1}{1 + e^{-x}}$$



  - On a vector input:

$$g(\boldsymbol{x}) = \begin{bmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_D) \end{bmatrix}$$

- The new error function: $\mathcal{E}(\boldsymbol{s}^\top || g(\boldsymbol{f}^\top \boldsymbol{X})) = \frac{1}{N}(\boldsymbol{s}^\top - g(\boldsymbol{f}^\top \boldsymbol{X})) \cdot (\boldsymbol{s}^\top - g(\boldsymbol{f}^\top \boldsymbol{X}))^\top$

Image Denoising

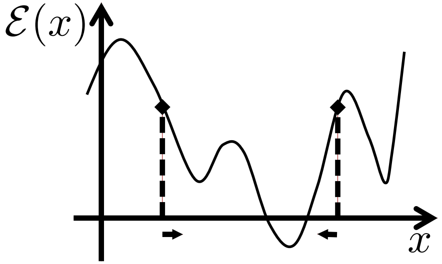
- Gradient descent

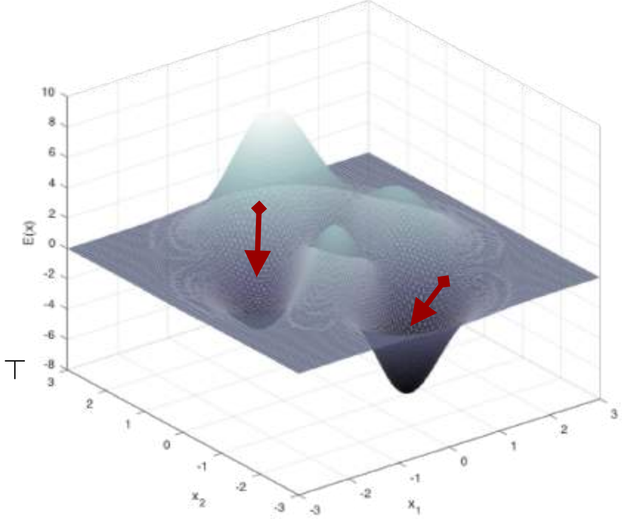
- Is this a quadratic function of $\boldsymbol{f}$?

$$\mathcal{E}(\boldsymbol{s}^\top \| g(\boldsymbol{f}^\top \boldsymbol{X})) = \frac{1}{N}(\boldsymbol{s}^\top - g(\boldsymbol{f}^\top \boldsymbol{X})) \cdot (\boldsymbol{s}^\top - g(\boldsymbol{f}^\top \boldsymbol{X}))^\top = \frac{1}{N}\left(\boldsymbol{s}^\top - \frac{1}{1+\exp(-\boldsymbol{f}^\top \boldsymbol{X})}\right) \cdot \left(\boldsymbol{s}^\top - \frac{1}{1+\exp(-\boldsymbol{f}^\top \boldsymbol{X})}\right)^\top$$

  - Maybe not. I have no idea.

- Now we need a different strategy: **gradient descent**





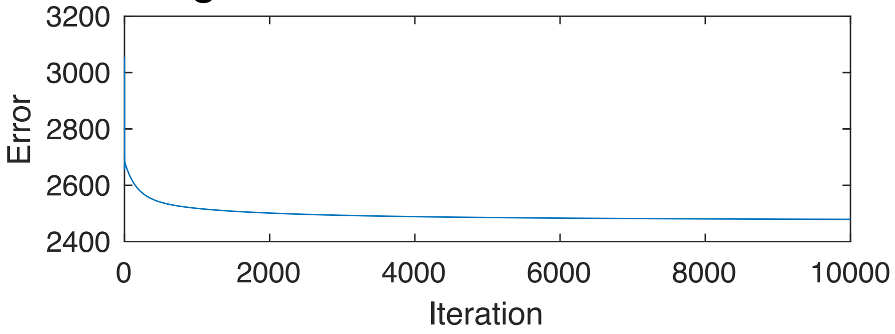
- Partial differentiation $\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}} = -\frac{2}{N}\boldsymbol{X}\left\{(\boldsymbol{s}^\top - g(\boldsymbol{f}^\top \boldsymbol{X})) \odot g'(\boldsymbol{f}^\top \boldsymbol{X})\right\}^\top$
- Minimization
  - Negative gradient: $\nabla \boldsymbol{f}^{(i)} = -\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{(i)}}$
  - Update rule:

$$\boldsymbol{f}^{(i+1)} \leftarrow \boldsymbol{f}^{(i)} + \rho \nabla \boldsymbol{f}^{(i)}$$

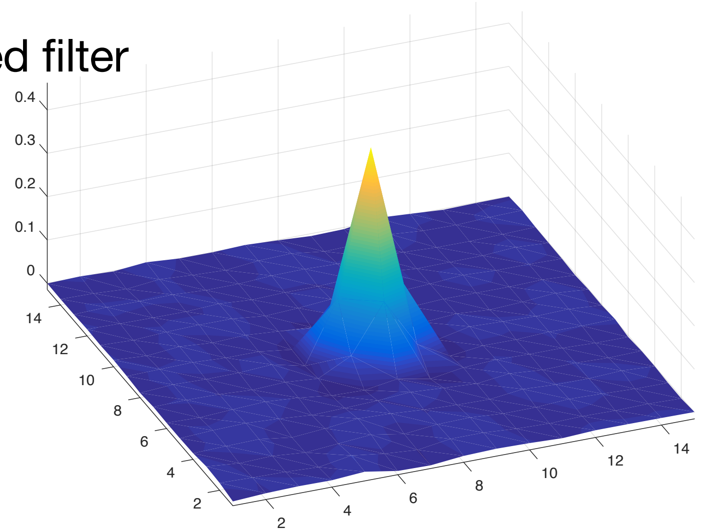
Image Denoising

- Gradient descent

- Convergence



- Learned filter



- Denoising results (training)

Original　　Noisy　　Denoised

Image Denoising

- Gradient descent

○ Denoising results (test)

Original

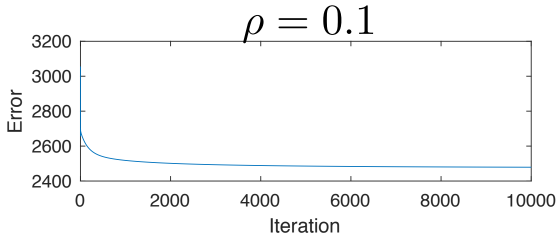
Noisy

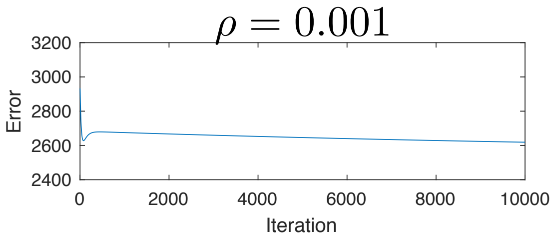
Denoised

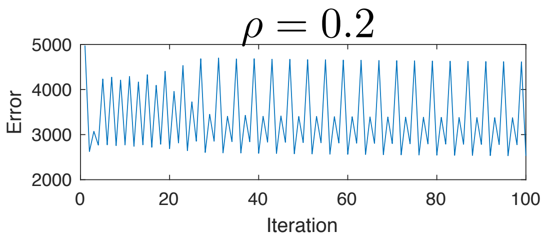
Image Denoising

- Gradient descent

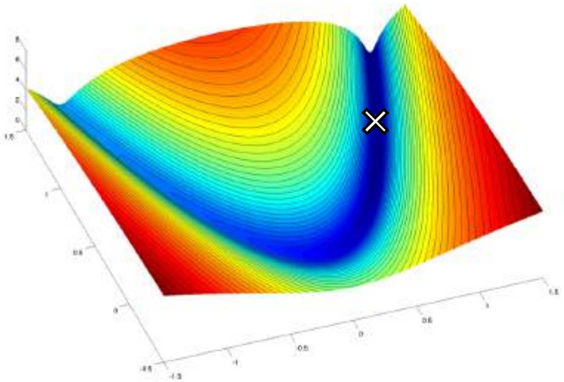
- Limitations of the gradient descent method $\boldsymbol{f}^{(i+1)} \leftarrow \boldsymbol{f}^{(i)} + \rho \nabla \boldsymbol{f}^{(i)}$
  - Step size



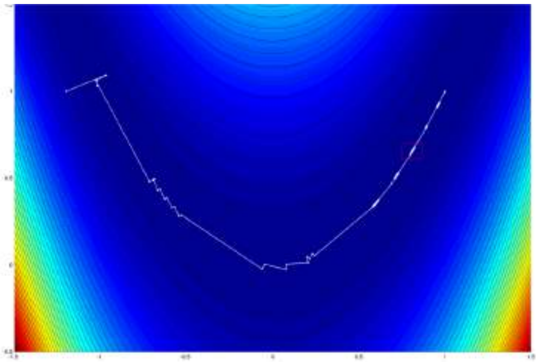




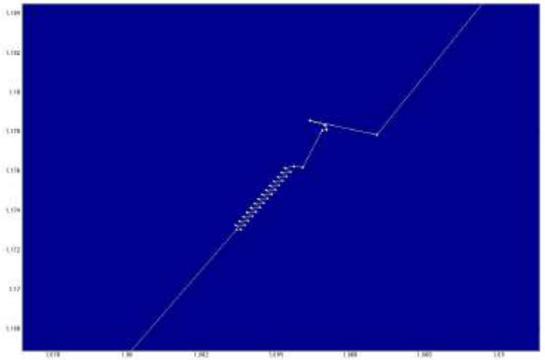
  - Zigzag movement



Rosenbrock function



Trajectory of x



Zigzag pattern

Image Denoising

- Newton's method

- Gradient descent is based on an imperfect approximation
  - First-order Taylor series expansion

    $$f(\theta) \approx f(\theta^*) + f'(\theta^*)(\theta - \theta^*)$$

  - Not accurate enough for non-linear objective functions
  - Why?
    - The gradient is changing its slope
  - Second-order expansion

    $$f(\theta) \approx f(\theta^*) + f'(\theta^*)(\theta - \theta^*) + \frac{1}{2} f''(\theta^*)(\theta - \theta^*)^2$$

    $$f(\theta) \approx f(\theta^*) + \left(f'(\theta^*) + \frac{1}{2} f''(\theta^*)(\theta - \theta^*) \right)(\theta - \theta^*)$$

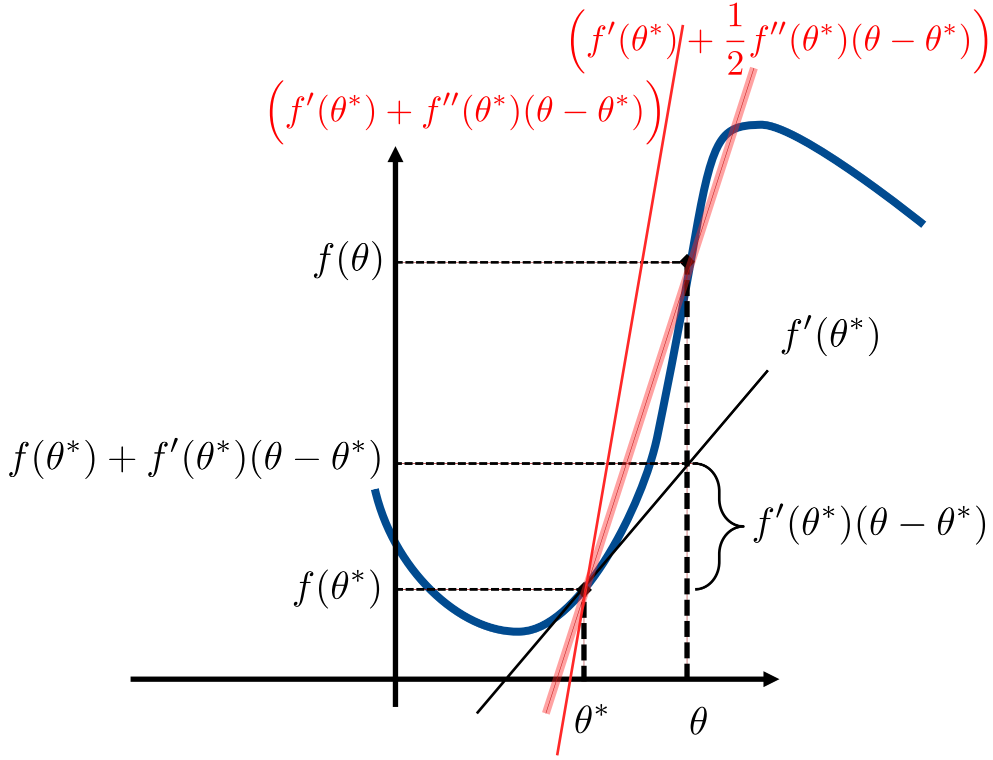


Image Denoising

- Newton's method

- The new update rule with Newton's method
  - Let's derive the update rule from the Taylor's series

$$f(\theta^{(i+1)}) \approx \tilde{f}(\theta^{(i+1)}) = f(\theta^{(i)}) + f'(\theta^{(i)})\Delta\theta + \frac{1}{2}f''(\theta^{(i)})\Delta\theta^2$$

  - We know that $\tilde{f}(\theta^{(i+1)})$ is a quadratic function of $\Delta\theta$
    - So, the stationary point is the local minimum

$$\frac{\partial \tilde{f}(\theta^{(i+1)})}{\partial \theta} = f'(\theta^{(i)}) + f''(\theta^{(i)})\Delta\theta = 0 \qquad \theta^{(i+1)} \leftarrow \theta^{(i)} + \Delta\theta = \theta^{(i)} - \frac{f'(\theta^{(i)})}{f''(\theta^{(i)})}$$

  - No step size (not always)
  - Finds the global minimum if the objective function is quadratic

- Multidimensional case
  - Gradient descent: $\boldsymbol{f}^{(i+1)} \leftarrow \boldsymbol{f}^{(i)} + \rho\nabla\boldsymbol{f}^{(i)}$ $\qquad \nabla\boldsymbol{f}^{(i)} = -\frac{\partial\mathcal{E}}{\partial\boldsymbol{f}^{(i)}}$
  - Newton's method: $\boldsymbol{f}^{(i+1)} \leftarrow \boldsymbol{f}^{(i)} - (\boldsymbol{H}^{(i)})^{-1}\nabla\boldsymbol{f}^{(i)}$ $\qquad H_{mn}^{(i)} = \frac{\partial^2\mathcal{E}}{\partial f_m^{(i)}\partial f_n^{(i)}}$
    - Problematic if there are too many parameters

Image Denoising

- Newton’s method

- Newton’s method for optimizing the Rosenbrock function

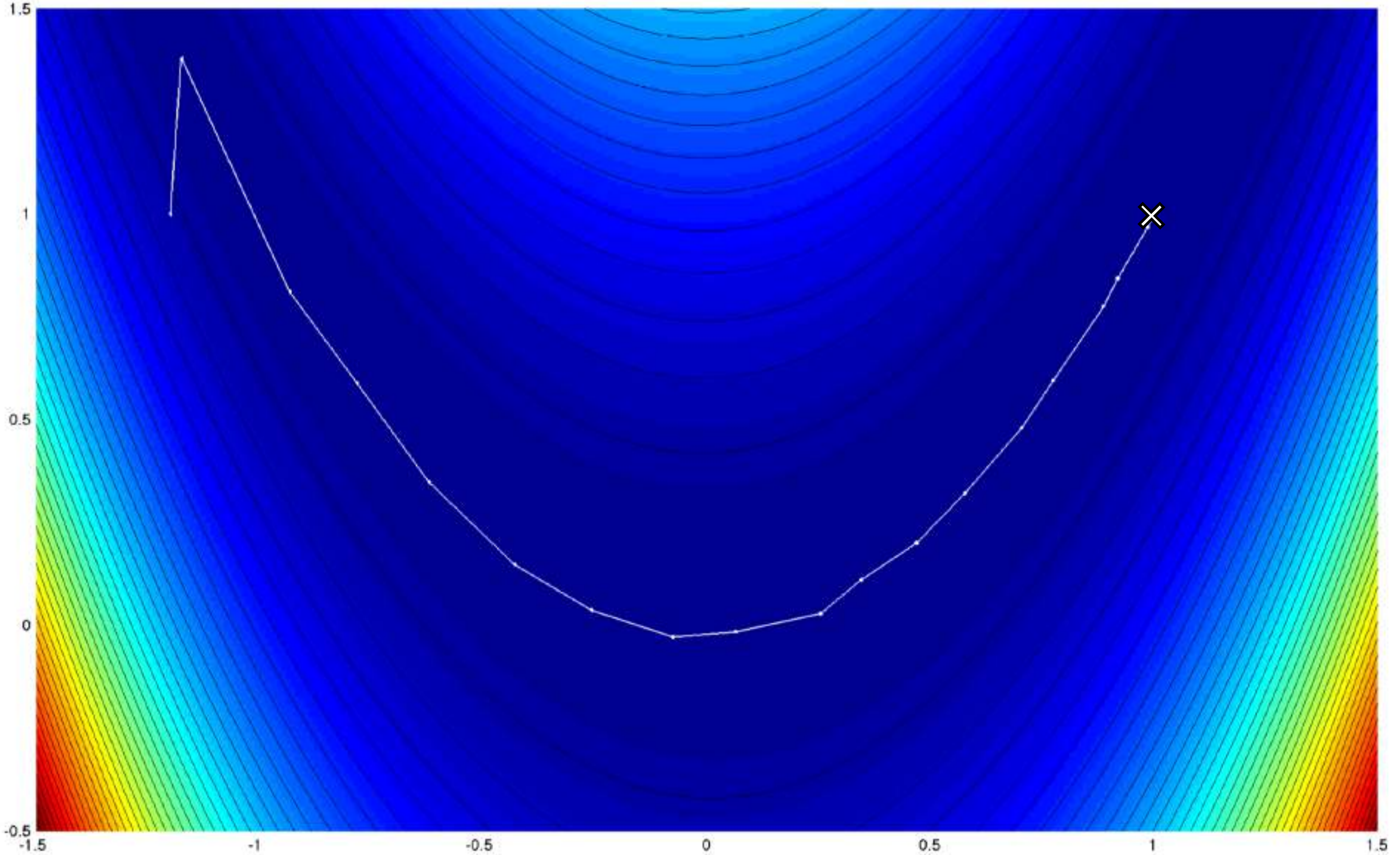


Image Denoising

- Newton's method

- You don't want to calculate the inverse-Hessian matrix at every iteration
  - Especially if you have too many parameters
  - Inversion is expensive
  - There are a lot of approximation techniques
- Quasi Newton's methods
  - BFGS calculates an approximation $\boldsymbol{G}^{(i)} \approx \left(\boldsymbol{H}^{(i)}\right)^{-1}$

$$\boldsymbol{G}^{(i+1)} \leftarrow \boldsymbol{G}^{(i)} + \frac{\boldsymbol{p}\boldsymbol{p}^\top}{\boldsymbol{p}^\top\boldsymbol{v}} - \frac{(\boldsymbol{G}^{(i)}\boldsymbol{v})\boldsymbol{v}^\top\boldsymbol{G}^{(i)}}{\boldsymbol{v}^\top\boldsymbol{G}^{(i)}\boldsymbol{v}} + (\boldsymbol{v}^\top\boldsymbol{G}^{(i)}\boldsymbol{v})\boldsymbol{u}\boldsymbol{u}^\top$$

$$\boldsymbol{p} = \boldsymbol{f}^{(i+1)} - \boldsymbol{f}^{(i)}$$

$$\boldsymbol{v} = \nabla\boldsymbol{f}^{(i+1)} - \nabla\boldsymbol{f}^{(i)}$$

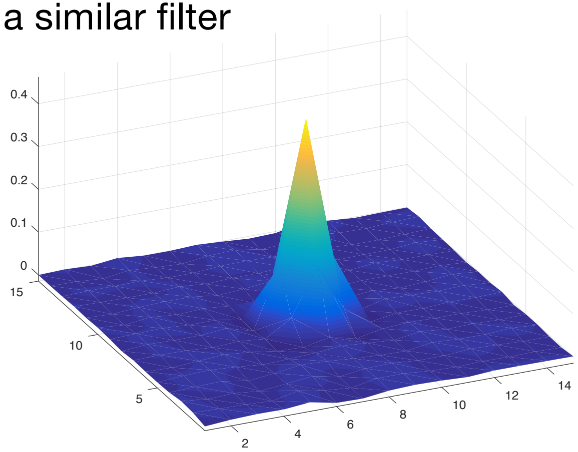
$$\boldsymbol{u} = \frac{\boldsymbol{p}}{\boldsymbol{p}^\top\boldsymbol{v}} - \frac{\boldsymbol{G}^{(i)}\boldsymbol{v}}{\boldsymbol{v}^\top\boldsymbol{G}^{(i)}\boldsymbol{v}}$$

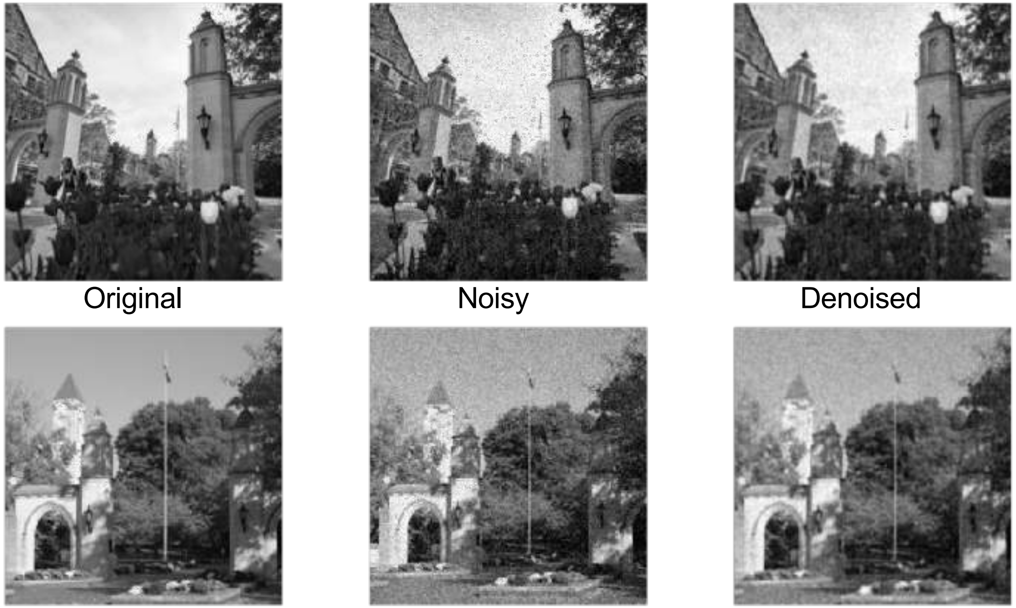
  - We use gradients and parameters to approximate Hessian

Image Denoising

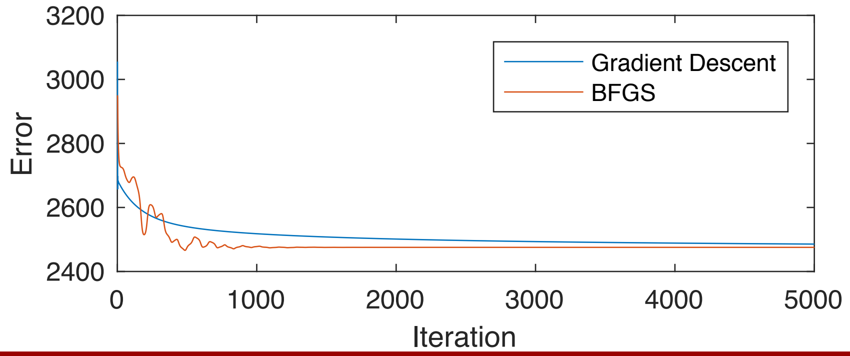
- Newton's method

- BFGS results
  - Similar denoising performance
  - Learns a similar filter





- Convergence



Constrained Optimization

- Maximum likelihood estimation with Lagrange multiplier

- You're analyzing Prof. K's writing to build a model out of it
- Somebody claims that a newly discovered document looks like written by Prof. K
  - You'll use the model to determine whether the spurious article is a forgery or not
- You collect many articles from Prof. K's journal as a training set and count the number of three keywords:
  - "Research": 19 times, "Burger": 15 times, "Spinach": 7 times
- Now you find the multinomial distribution that best fits the data
  - Multinomial distribution

$$P(X_1 = x_1, X_2 = x_2, \cdots, X_K = x_K; N, p_1, p_2, \cdots, p_K) = \frac{N!}{x_1! x_2! \cdots x_K!} p_1^{x_1} p_2^{x_2} \cdots p_K^{x_K}$$

$$0 \leq p_k \leq 1 \qquad \sum_{k=1}^{K} p_k = 1 \qquad N = \sum_{k=1}^{K} x_k$$

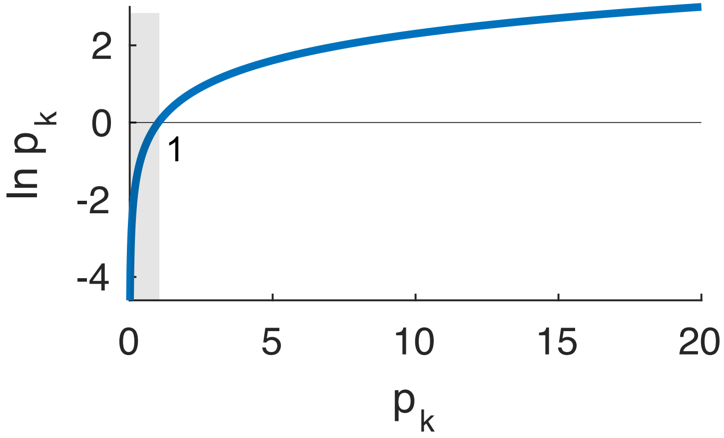
Constrained Optimization

- Maximum likelihood estimation with Lagrange multiplier

- The objective function: $$\arg\max_{p_1,p_2,\cdots,p_K} \frac{N!}{x_1!x_2!\cdots x_K!} p_1^{x_1} p_2^{x_2} \cdots p_K^{x_K}$$
  - Take logarithm for convenience

$$\arg\max_{p_1,p_2,\cdots,p_K} \mathcal{LL}(p_1,p_2,\cdots,p_k) = \arg\max_{p_1,p_2,\cdots,p_K} \ln\left\{\frac{N!}{x_1!x_2!\cdots x_K!} p_1^{x_1} p_2^{x_2} \cdots p_K^{x_K}\right\} = \arg\max_{p_1,p_2,\cdots,p_K} \sum_{k=1}^{K} x_k \ln p_k + Constan$$

- Along a particular dimension p_k we know that the log function does not have a stationary point
- But, we do know that $0 \leq p_k \leq 1$
  - Then, is this the solution? $p_1 = p_2 = \cdots = p_K = 1$
  - No, because we forgot another constraint: $\sum_{k=1}^{K} p_k = 1$



Constrained Optimization

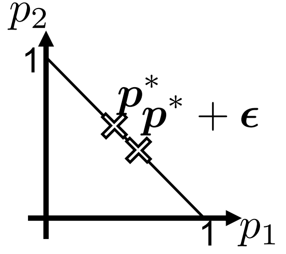
- Maximum likelihood estimation with Lagrange multiplier

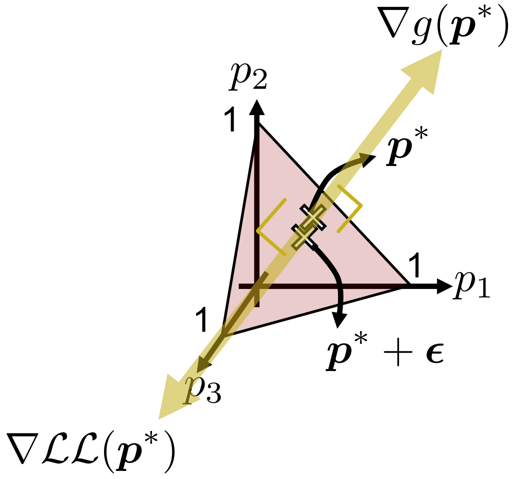
- We saw that MLE is not enough to solve the problem
  - Let's deal with this **equality constraint**
- Let $\boldsymbol{p}^* = [p_1^*, p_2^*, \cdots, p_K^*]^\top$ be our solution to the optimization problem
- And, the equality constraint $g(\boldsymbol{p}) = \sum_k p_k - 1$
  - Since $\boldsymbol{p}^*$ is the solution,

$$g(\boldsymbol{p}^*) = 0$$

- Let's add a vector with very small values
  - that still meet the constraint $g(\boldsymbol{p}^* + \boldsymbol{\epsilon}) = 0$
- Also, because of the Taylor series expansion,

$$g(\boldsymbol{p}^* + \boldsymbol{\epsilon}) = 0 \approx g(\boldsymbol{p}^*) + \boldsymbol{\epsilon}^\top \nabla g(\boldsymbol{p}^*)$$





- Therefore, $\nabla g(\boldsymbol{p}^*)$ is orthogonal to $\boldsymbol{\epsilon}$, and consequently to the constrained surface, too
- Meanwhile, since $\boldsymbol{p}^*$ is the solution, $\mathcal{LL}(\boldsymbol{p}^*)$ is the maximum
  - If we add $\boldsymbol{\epsilon}$, then $\boldsymbol{p}^* + \boldsymbol{\epsilon}$ will decrease the objective function value
  - Hence, $\nabla \mathcal{LL}(\boldsymbol{p}^*)$ is orthogonal to the constrained surface, too

Constrained Optimization

- Maximum likelihood estimation with Lagrange multiplier

- What we just found is that $\nabla\mathcal{LL}(\boldsymbol{p}^*)$ and $\nabla g(\boldsymbol{p}^*)$ are parallel

$$\nabla\mathcal{LL}(\boldsymbol{p}^*) = \lambda\nabla g(\boldsymbol{p}^*) \Leftrightarrow \nabla\mathcal{LL}(\boldsymbol{p}^*) + \lambda\nabla g(\boldsymbol{p}^*) = 0$$

- Now, let's define the final objective function with Lagrange multiplier

$$\nabla\mathcal{LLL}(\boldsymbol{p}, \lambda) = \mathcal{LL}(\boldsymbol{p}) + \lambda g(\boldsymbol{p})$$

  - Therefore, $\boldsymbol{p}$ at the stationary point meets the parallel condition
  - On top of that, we still have the equality constraint to make use of

$$g(\boldsymbol{p}^*) = 0 \Leftrightarrow \sum_k p_k^* = 1$$

- Let's get back to the MLE problem:

$$\arg\max_{\lambda,\boldsymbol{p}} \mathcal{LLL}(\lambda, \boldsymbol{p}) = \arg\max_{\lambda,\boldsymbol{p}} \sum_{k=1}^{K} x_k \ln p_k + \lambda\Big(\sum_k p_k - 1\Big)$$

  - We can derive a series of equations:

$$\frac{\partial\mathcal{LLL}(\lambda, \boldsymbol{p})}{p_k} = \frac{x_k}{p_k} + \lambda = 0 \Leftrightarrow x_k = -\lambda p_k \quad \Leftrightarrow \sum_k x_k = -\lambda\sum_k p_k \quad \Leftrightarrow N = -\lambda$$

  - Finally: $p_k^* = \dfrac{x_k}{N}$

What I didn’t cover

- Line search
- Momentum
- Inequality constraint
- Conjugate gradient
- Stochastic gradient descent
- You name it

Reading material

- Textbook appendix C
- Jorge Nocedal and Stephen Wright, “Numerical Optimization” http://iucat.iu.edu/catalog/11876818

Thank You!