5. Quasi-Newton Method for IK

Game Engineering & XR Technologies
Prof. HyeongYeop Kang
siamiz@khu.ac.kr
IIIXR LAB

Introduction

Quasi-Newton Methods

- Recall that the second-order newton methods often converge much more quickly but it can be very expensive to calculate and store the inverse of the Hessian matrix at each iteration.

- Therefore, most people prefer quasi-newton methods to approximate the Hessian.
  - Quasi-Newton methods approximate the Hessian matrix or its inverse, resulting in faster convergence and reduced computational complexity.

- Quasi-Newton Computation flow:
  - Start with an initial guess for the optimal solution.
  - Compute the gradient at the current point.
  - Approximate the Hessian matrix or its inverse.
  - Update the solution using the approximated Hessian matrix.
  - Repeat steps until a stopping criterion is met (reaching a maximum number of iterations or a desired level of convergence.)

Introduction

Quasi-Newton VS. Newton method.

- Advantages
  - Reduced computational complexity.
  - Faster Convergence (especially for ill-conditioned problems).
  - More robust as Quasi-Newton methods are less sensitive to the initial guess and can handle non-quadratic objective functions.
- Disadvantages
  - Less accurate Hessian matrix.
  - Slower convergence for some problems (when the approximations are not sufficiently accurate or when the problem is well-conditioned, the Newton method can converge more quickly than Quasi-Newton methods).

Recent research topics:

- Improves hessian approximations.
- Parallelization and distributed computing.
- Application to machine learning.
- Understanding the convergence properties and establishing better stopping criteria.

Basics ^^

Ill-conditioned problem

- Ill-conditioned problems are those where the condition number of the Hessian matrix is large, making the optimization process numerically challenging.
  - The condition number of a matrix is the ratio of its largest and smallest eigenvalues.
  - A high condition number implies that the matrix is sensitive to small perturbations in the input data, leading to numerical instability in the optimization process.
- This can arise from various sources:
  - Poor scaling of variables.
  - High-dimensional problems.
    - Training large-scale neural network, sparse signal recovery problem, etc.
  - Intrinsic properties of the objective function.
- This can be mitigated by using:
  - Appropriate variable scaling.
  - Preconditioning.
    - Transform the original problem into an equivalent one with better numerical properties.
  - Use of optimization algorithms that are more robust to ill-conditioning (e.g. trust region method).
    - Trust region methods focus on finding a suitable step size and direction by defining a region around the current point.
    - Adaptive gradient methods such as AdaGrad, RMSProp, and Adam adaptively adjust the learning rate for each parameter in the optimization problem, making them more robust to ill-conditioning.

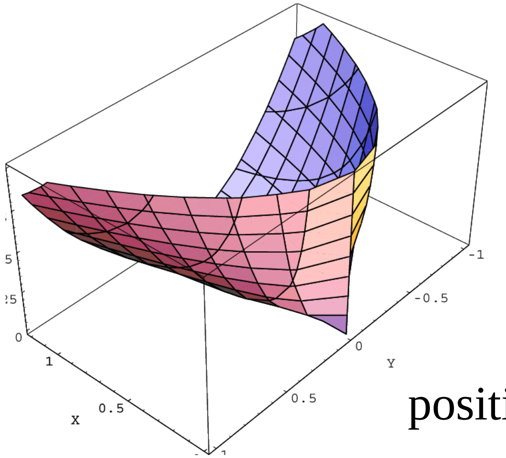
Basics ^^

Definite Matrix[1]

- Let x^T is the transpose of x and 0 denotes the n-dimensional zero-vector.
  - An $n \times n$ symmetric real matrix M is said to be positive-definite if $x^T M x > 0$ for all non-zero x in R^n.
  - An $n \times n$ symmetric real matrix M is said to be positive-semidefinite or non-negative-define if $x^T M x \geq 0$ for all x in R^n.
  - An $n \times n$ symmetric real matrix M is said to be negative-definite if $x^T M x < 0$ for all non-zero x in R^n.
  - An $n \times n$ symmetric real matrix M is said to be negative-semidefinite or non-positive-define if $x^T M x \leq 0$ for all x in R^n.

- Positive-definite and positive-semidefinite real matrices are at the basis of convex optimization, if the Hessian matrix of a function is positive-definite at a point p, then the function is convex near p.

- Conversely, if the function is convex near p, then the Hessian matrix is positive-semidefinite at p.

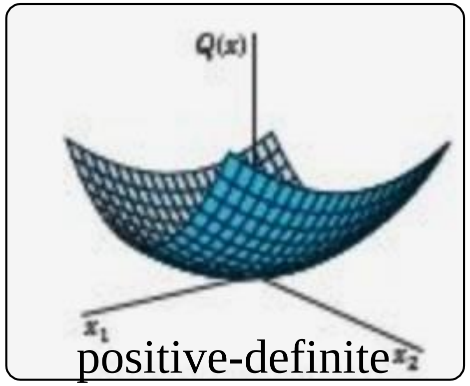


positive semidefinite cone

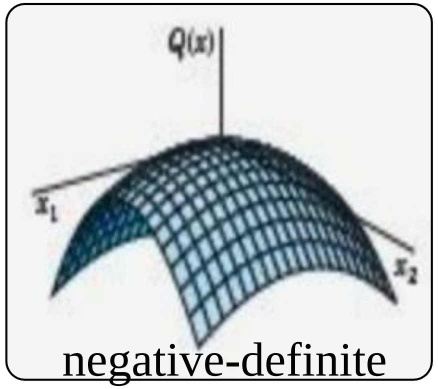
Basics ^^

Definite Matrix[1]

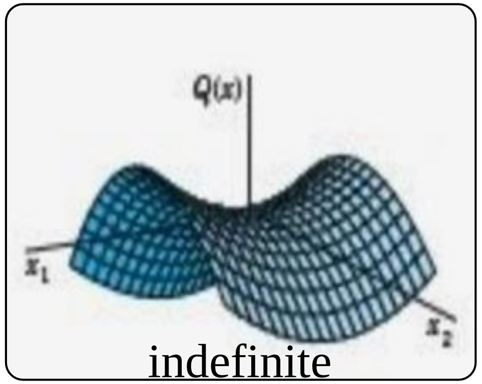
- Definite matrices are important in optimization problems because they provide insights into the curvature of the objective function, which can be crucial for determining the convergence and stability of optimization problems.
  - The properties of positive-definite matrix *M* include 1) all eigenvalues are positive, 2) the matrix is invertible, and 3) the matrix is symmetric.
    - ✓ The positive definite matrices correspond to convex functions, which have a unique global minimum. This property ensures that optimization algorithms converge to a unique solution.
  - The properties of negative-definite matrix *M* include 1) all eigenvalues are negative, 2) the matrix is invertible, and 3) the matrix is symmetric.
    - ✓ The negative definite matrices correspond to concave functions, which have a unique global maximum. This property ensures that optimization algorithms converge to a unique solution.
  - The properties of indefinite matrix *M* include 1) the matrix has both positive and negative eigenvalues, and 2) the matrix is symmetric (if it is a real matrix).
    - ✓ The indefinite matrices correspond to functions that exhibit both convex and concave regions, making optimization more challenging.



positive-definite



negative-definite

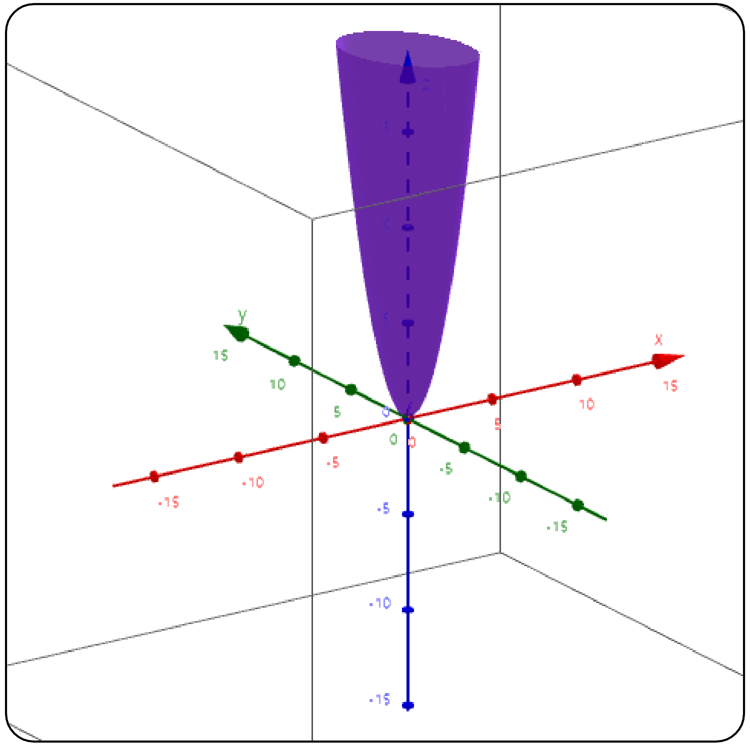


indefinite

Basics ^^

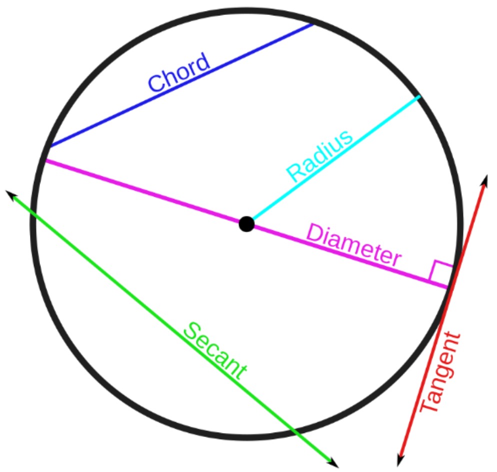
Definite Matrix Example

- Consider a simple quadratic optimization problem: minimize $f(x, y) = 3x^2 + 2xy + 2y^2$.
  - $H_f(x, y) = \begin{bmatrix} 6 & 2 \\ 2 & 4 \end{bmatrix}$
  - For the Hessian matrix to be positive-definite, it must satisfy the following conditions:
    - ✓ All its eigenvalues must be positive.
    - ✓ It must be symmetric.
  - Two eigenvalues are $-\sqrt{5}+5$ and $\sqrt{5}+5$.
    - ✓ Therefore, the problem has a unique solution.



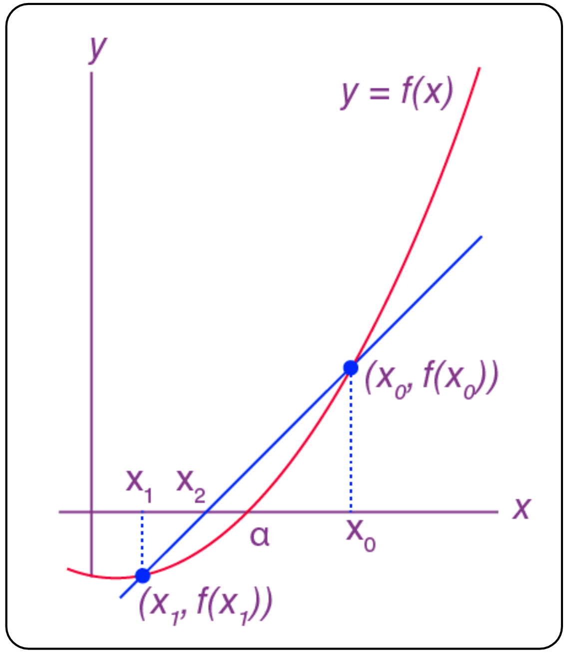
Basics ^^

Secant Method[2]



- The secant method is a root-finding algorithm that is used to find the zeros of a continuous function.
  - This is an iterative method approximates the root of a function by using a linear approximation based on the function's values at two previous points.
  - The term "secant" refers to a straight line that intersects a curve at two distinct points.
  - In the context of the secant method, the secant line connects two points on the graph of the function you are trying to find the root for.

- Given a continuous function $f(x)$, the goal is to find a root of the function, i.e., a point x_d such that $f(x_d) = 0$.
  - The algorithm starts with two initial guess, x_0 and x_1.
  - The slope of the secant line m passing through the points (x_0, $f(x_0)$) and (x_1, $f(x_1)$):
    - ✓ $m = \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}$
  - Extrapolate the secant line to the point where it intersects the x-axis:
    - ✓ $x_2 = x_1 - \dfrac{f(x_1)}{m}$
  - Update the points and repeat the process.



Basics ^^

Secant Method

- The secant method is defined by the recurrence relation:

$$x_n = x_{n\text{-}1} - f(x_{n\text{-}1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}$$

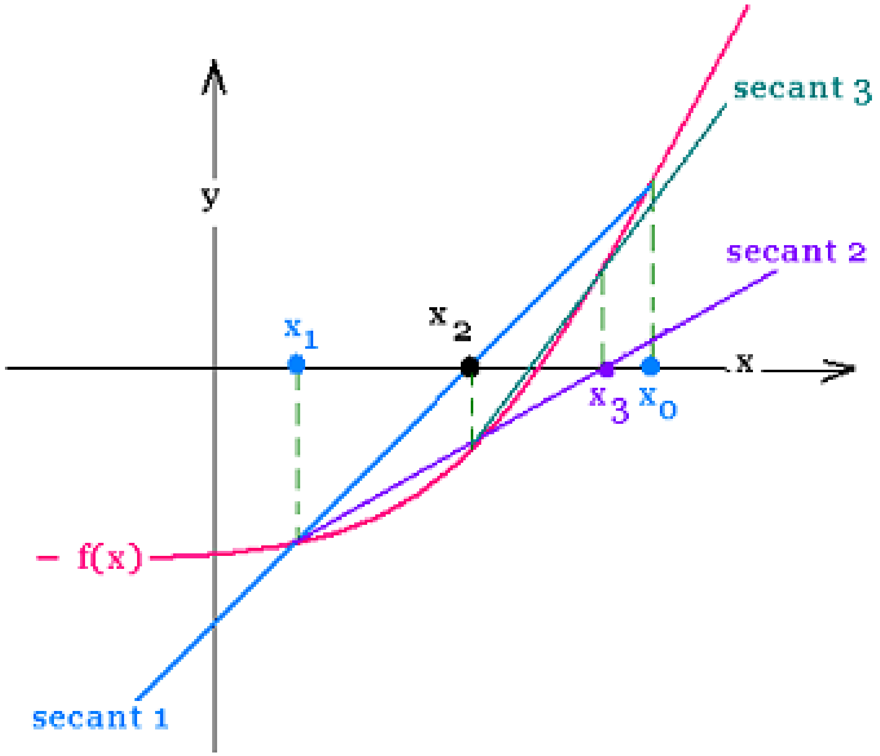
- The above process is iteratively processed until certain criterion is satisfied.

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)},$$

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)},$$

$$\vdots$$

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}.$$



First three iterations of the secant method

Practice

Q. Assume that we are finding an root of an equation $f(x) = x^3 - x - 1$ using Secant method. Let $x_0 = 1$ and $x_1 = 2$.

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)},$$

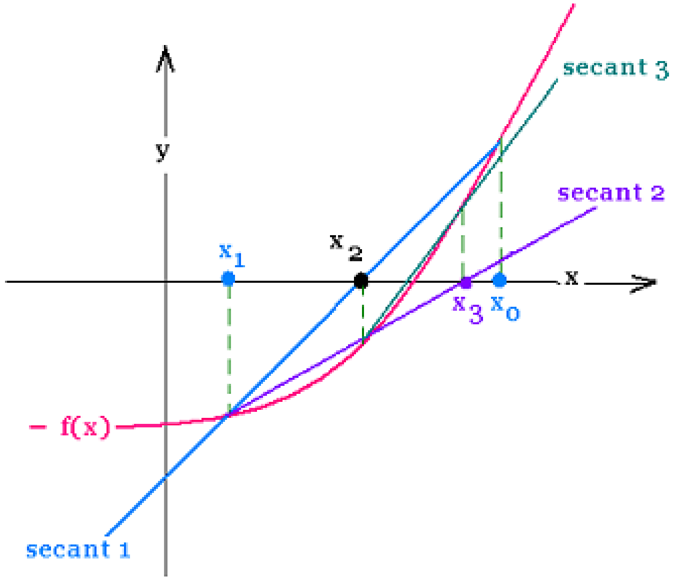
$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)},$$

$$\vdots$$

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}.$$

Q1. Calculate x_2.

Q2. Find a root (threshold = 0.001).



First three iterations of the secant method

Solution

Q. Assume that we are finding an root of an equation $f(x) = x^3 - x - 1$ using Secant method. Let $x_0 = 1$ and $x_1 = 2$.

Q1. Calculate x_2.

Q2. Find a root.

$x_0 = 1$ and $x_1 = 2$

$f(x_0) = f(1) = -1$ and $f(x_1) = f(2) = 5$

$$\therefore x_2 = x_0 - f(x_0) \cdot \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

$$x_2 = 1 - (-1) \times \frac{2 - 1}{5 - (-1)}$$

$x_2 = 1.16667$

$\therefore f(x_2) = f(1.16667) = -0.5787$

$x_1 = 2$ and $x_2 = 1.16667$

$f(x_1) = f(2) = 5$ and $f(x_2) = f(1.16667) = -0.5787$

$$\therefore x_3 = x_1 - f(x_1) \cdot \frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

$$x_3 = 2 - 5 \times \frac{1.16667 - 2}{-0.5787 - 5}$$

$x_3 = 1.25311$

$\therefore f(x_3) = f(1.25311) = -0.28536$

$x_2 = 1.16667$ and $x_3 = 1.25311$

$f(x_2) = f(1.16667) = -0.5787$ and $f(x_3) = f(1.25311) = -0.28536$

$$\therefore x_4 = x_2 - f(x_2) \cdot \frac{x_3 - x_2}{f(x_3) - f(x_2)}$$

$$x_4 = 1.16667 - (-0.5787) \times \frac{1.25311 - 1.16667}{-0.28536 - (-0.5787)}$$

$x_4 = 1.33721$

$\therefore f(x_4) = f(1.33721) = 0.05388$

$x_3 = 1.25311$ and $x_4 = 1.33721$

$f(x_3) = f(1.25311) = -0.28536$ and $f(x_4) = f(1.33721) = 0.05388$

$$\therefore x_5 = x_3 - f(x_3) \cdot \frac{x_4 - x_3}{f(x_4) - f(x_3)}$$

$$x_5 = 1.25311 - (-0.28536) \times \frac{1.33721 - 1.25311}{0.05388 - (-0.28536)}$$

$x_5 = 1.32385$

$\therefore f(x_5) = f(1.32385) = -0.0037$

$x_4 = 1.33721$ and $x_5 = 1.32385$

$f(x_4) = f(1.33721) = 0.05388$ and $f(x_5) = f(1.32385) = -0.0037$

$$\therefore x_6 = x_4 - f(x_4) \cdot \frac{x_5 - x_4}{f(x_5) - f(x_4)}$$

$$x_6 = 1.33721 - 0.05388 \times \frac{1.32385 - 1.33721}{-0.0037 - 0.05388}$$

$x_6 = 1.32471$

$\therefore f(x_6) = f(1.32471) = -0.00004$

Basics ^^

Frobenius norm

- The Frobenius norm, also known as the Euclidean norm, is a matrix norm that is used to measure the size of a matrix.
  - It is defined for an $m \times n$ matrix A with elements a_{ij} as follows:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |aij|^2}$$

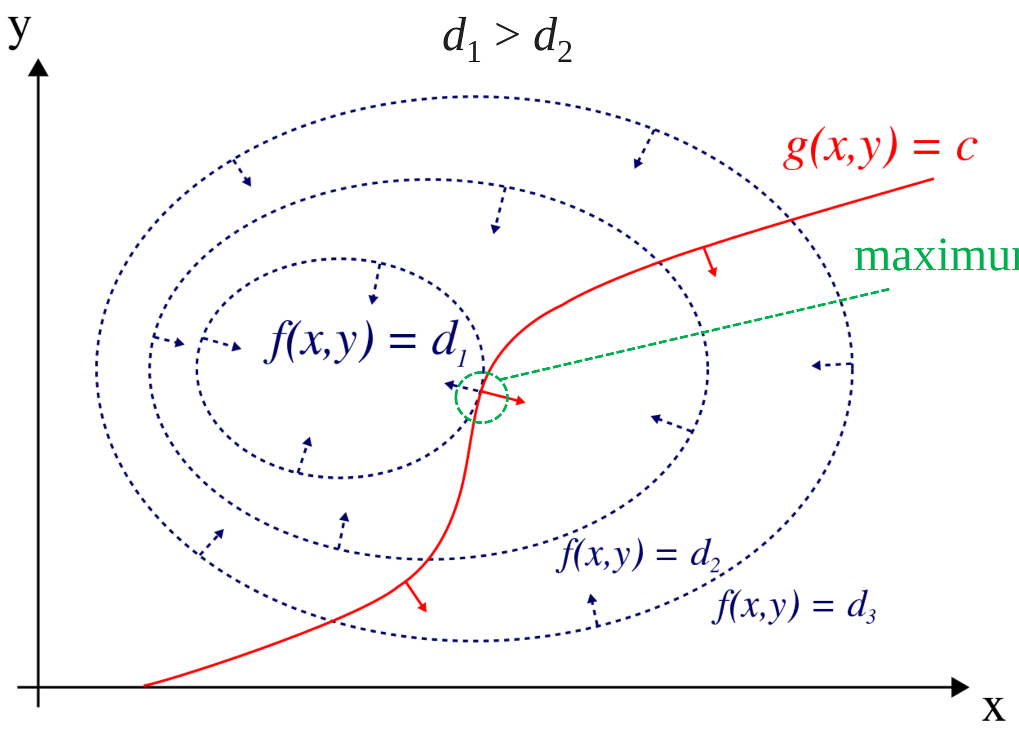
- The Frobenius norm has a relationship with trace properties.
  - The trace of a square matrix A, denoted tr(A), is defined to be the sum of elements on the main diagonal of A.
  - The trace of an $n \times n$ matrix A is defined as: $\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn}$
  - Then, the squared Frobenius norm equals the square root of the matrix trace of AA^H, where A^H is the conjugate transpose.

$$\|A\|_F = \sqrt{\mathrm{Tr}(AA^H)}$$

Basics ^^

Lagrange Multiplier[LM]

- The Lagrange multiplier method is used in optimization problem to find the maximum or minimum of a function subject to certain constraints.
  - In other words, this is useful when we want to optimize and objective function while satisfying a set of equality constraints.
- The method works by incorporating the constraints into the objective function using additional variables called Lagrange multipliers (this multipliers helps to penalize violations of the constraints).



The red curve shows the constraint $g(x, y) = c$.
The blue curves are contours of $f(x, y)$.
The point where the red constraint tangentially touches a blue contour is the maximum of $f(x, y)$ along the constraint.

Basics ^^

Lagrange Multiplier

- Method overview:
  - Suppose you have an objective function $f(x, y)$ that you want to optimize, subject to the constraint $g(x, y) = 0$.
    - Maximize/minimize f(x, y)
    - subject to $g(x, y) = 0$
  - Define the Lagrange function $L(x, y, \lambda)$ as follows:
    - $L(x, y, \lambda) = f(x, y) + \lambda g(x, y)$, where λ is the Lagrange multiplier.
  - Compute the partial derivatives of $L(x, y, \lambda)$ with respect to each variable x , y, and λ.
  - Set these partial derivatives equal to zero to find the critical points of the Lagrangian.
  - Solve the resulting system of equations to find the optimal values for x and λ.
  - Plug the optimal x values back into the original objective function $f(x, y)$ to find the maximum or minimum value.

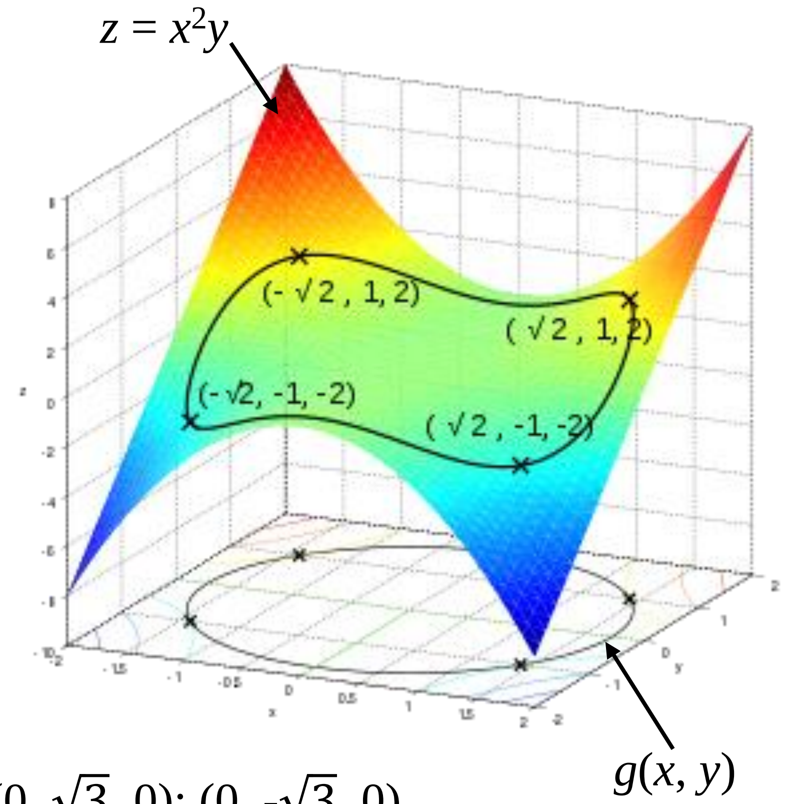
Basics ^^

Lagrange Multiplier Example

- Suppose one wants to find the maximum values of $f(x, y) = x^2y$ with the condition that the x- and y- coordinates lie on the circle around the origin with radius $\sqrt{3}$.
  - maximize $f(x, y) = x^2y$ with respect to $g(x, y) = x^2 + y^2 - 3 = 0$.

- As there is just a single constraint, there is a single multiplier, say λ.
  - $L(x, y, \lambda) = f(x, y) + \lambda g(x, y) = x^2y + \lambda\,(x^2 + y^2 - 3)$.

- The gradient can be calculated as:
  - $\nabla_{x,y,\lambda} L(x, y, \lambda) = (\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}, \frac{\partial L}{\partial \lambda}) = (2xy + 2\lambda x, x^2 + 2\lambda y, x^2 + y^2 - 3)$



- Set these partial derivatives as zeros, $\nabla_{x,y,\lambda} L(x, y, \lambda) = 0$.
  - $2xy + 2\lambda x = 0 \rightarrow x(y + \lambda) = 0$
  - $x^2 + 2\lambda y = 0 \rightarrow x^2 = -2\lambda y$
  - $x^2 + y^2 - 3 \rightarrow x^2 + y^2 = 3$
  - Six critical points of L: $(\sqrt{2}, 1, -1)$; $(-\sqrt{2}, 1, -1)$; $(\sqrt{2}, -1, 1)$; $(-\sqrt{2}, -1, 1)$; $(0, \sqrt{3}, 0)$; $(0, -\sqrt{3}, 0)$
  - Evaluating the objectives at these points: $f(\pm\sqrt{2}, 1) = 2$; : $f(\pm\sqrt{2}, -1) = -2$; : $f(0, (\pm\sqrt{3}) = 0$.

- Therefore, the objective function attains the maximum subject to the constraints at $(\pm\sqrt{2}, 1)$.

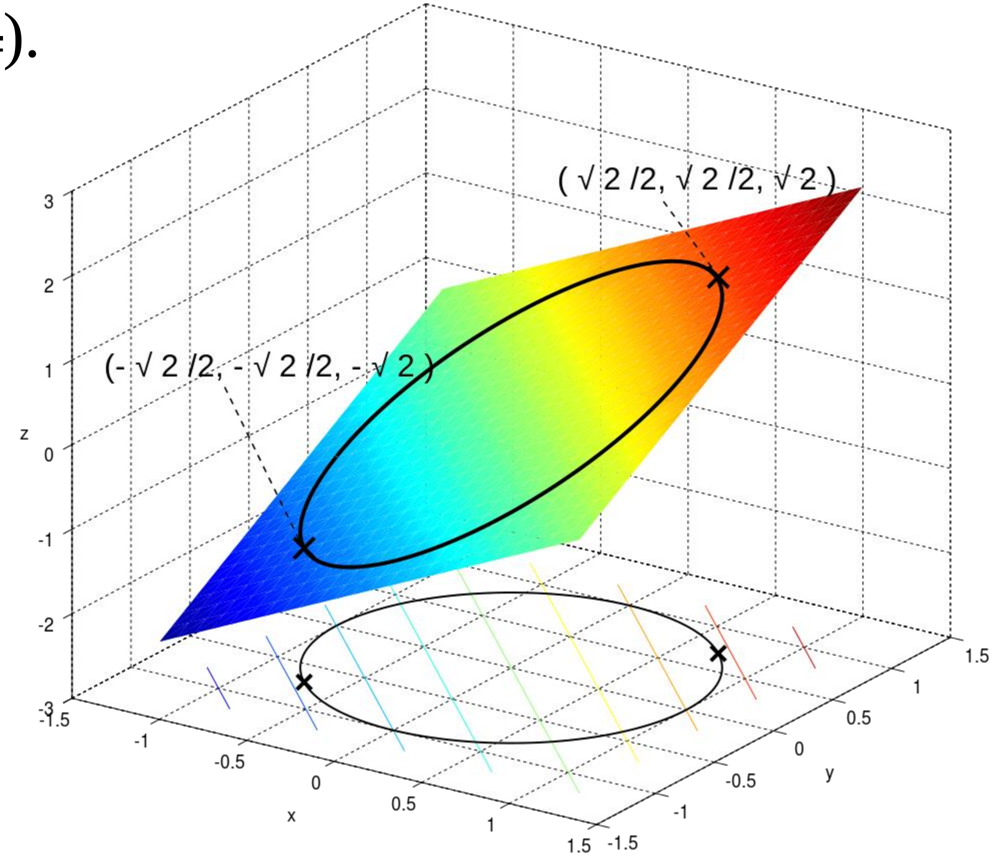
Practice

Q. Suppose we wish to maximize $f(x, y) = x + y$ subject to the constraint $x^2 + y^2 = 1$. Finds the constrained maximum.

Solution

Q. Suppose we wish to maximize $f(x, y) = x + y$ subject to the constraint $x^2 + y^2 = 1$. Finds the constrained maximum.

- $L(x, y, \lambda) = f(x, y) + \lambda g(x, y) = x + y + \lambda\,(x^2 + y^2 - 1)$,
- $\nabla_{x,y,\lambda} L(x, y, \lambda) = (\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}, \frac{\partial L}{\partial \lambda}) = (1 + 2\lambda x, 1 + 2\lambda y, x^2 + y^2 - 1)$
- Two critical points of L: $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -\frac{1}{\sqrt{2}})$; $(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, \frac{1}{\sqrt{2}})$.
- Thus the constrained maximum is $\sqrt{2}$.



Quasi-Newton Method

Search for extrema (optimization problem)[4]

- Search for a minimum or maximum of a scalar-valued function = Search for the zeros of the gradient of that function
  - In other words, if $g = \nabla f$, then searching for the zeroes of the vector-valued function g corresponds to the search for the extrema of the scalar-valued function f.
  - Then, the $J(\mathrm{g}) = H(f)$ (Recall that $J[\nabla f(x)] = H[f(x)]$).

- Newton's method assumes that the function can be locally approximated as a quadratic in the region around the optimum, and uses the gradient and Hessian matrix to find the stationary point.
  - Solving optimization problem using multivariate Newton Method: $\theta_{\mathrm{n}+1} = \theta_{\mathrm{n}} - \dfrac{\nabla F(\theta_n)}{H_F(\theta_n)}$.
  - To reduce the computation cost of the Hessian matrix, Quasi-Newton method updates Hessian by using successive gradient vectors instead.

Quasi-Newton Method

Search for extrema (optimization problem)

- How to approximate the Hessian matrix?
  - This is achieved by using a positive-definite matrix *B*, which is updated from iteration to iteration using information from the gradient of the objective function.

- Here, secant equation plays a crucial role in computing Hessian approximation based on the gradient information.
  - Given a scalar function $F(\theta)$, its Taylor series expansion around point θ_n is given by:
    - $F(\theta_{n+1}) = F(\theta_n) + \nabla F(\theta_n)(\triangle\theta) + O(\triangle\theta^2)$
    - (after differentiate) $\nabla F(\theta_{n+1}) = \nabla F(\theta_n) + H_F(\theta_n)(\triangle\theta)$
    - $H_F(\theta_n)(\triangle\theta) = \nabla F(\theta_{n+1}) - \nabla F(\theta_n)$

  - Assuming that the Hessian matrix is almost constant between θ_n and θ_{n+1}, the above equation can be rewritten as:
    - $B_F(\theta_{n+1})(\triangle\theta) \approx \nabla F(\theta_{n+1}) - \nabla F(\theta_n)$, where *B* is the Hessian approximation.

  - This approximation states that the updated Hessian approximation should satisfy the relationship between the difference in gradients ($\nabla F(\theta_{n+1}) - \nabla F(\theta_n)$) and the difference in iterates ($\triangle\theta$).
    - In the secant method, the slope term $\frac{\theta_n - \theta_{n-1}}{f(\theta_n) - f(\theta_{n-1})}$ is an approximation of the first derivative $\frac{1}{f'(\theta_n)}$.
    - In Quasi-Newton method, the term $\frac{\nabla F(\theta_{n+1}) - \nabla F(\theta_n)}{\theta_{n+1} - \theta_n}$ is an approximation of the second derivative $B_F(\theta_{n+1})$.
    - The analogy between the two methods is that both use information from two points to approximate a derivative. So, we call them both "secant".

Quasi-Newton Method

Quasi-Newton steps[6]

- The general template of the quasi-newton steps is as follows:
  - Solve $\triangle\theta_n = B_F^{-1}(\theta_{n+1})(\nabla F(\theta_{n+1}) - \nabla F(\theta_n))$
  - Determine step size (or learning rate) t_n.
  - Update $\theta_{n+1} = \theta_n + t_n \triangle\theta_n$ (where t denotes learning rate)

- However, the computation of the Hessian approximation is also very expensive, especially when dealing with a large number of variables or when the function being optimized is complex.
  - To address this issue, Quasi-Newton method updates the $B_F(\theta_{n+1})$ by using $B_F(\theta_n)$ information from the previous iteration.

- There are several algorithms for updating B.

Method	$B_{k+1} =$	$H_{k+1} = B_{k+1}^{-1} =$
BFGS	$B_k + \frac{y_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^{\mathrm{T}}}{\Delta x_k^{\mathrm{T}} B_k \Delta x_k}$	$\left(I - \frac{\Delta x_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}\right) H_k \left(I - \frac{y_k \Delta x_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}\right) + \frac{\Delta x_k \Delta x_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}$
Broyden	$B_k + \frac{y_k - B_k \Delta x_k}{\Delta x_k^{\mathrm{T}} \Delta x_k} \Delta x_k^{\mathrm{T}}$	$H_k + \frac{(\Delta x_k - H_k y_k) \Delta x_k^{\mathrm{T}} H_k}{\Delta x_k^{\mathrm{T}} H_k y_k}$
Broyden family	$(1 - \varphi_k) B_{k+1}^{\mathrm{BFGS}} + \varphi_k B_{k+1}^{\mathrm{DFP}}, \quad \varphi \in [0, 1]$	
DFP	$\left(I - \frac{y_k \Delta x_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}\right) B_k \left(I - \frac{\Delta x_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}\right) + \frac{y_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}$	$H_k + \frac{\Delta x_k \Delta x_k^{\mathrm{T}}}{\Delta x_k^{\mathrm{T}} y_k} - \frac{H_k y_k y_k^{\mathrm{T}} H_k}{y_k^{\mathrm{T}} H_k y_k}$
SR1	$B_k + \frac{(y_k - B_k \Delta x_k)(y_k - B_k \Delta x_k)^{\mathrm{T}}}{(y_k - B_k \Delta x_k)^{\mathrm{T}} \Delta x_k}$	$H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^{\mathrm{T}}}{(\Delta x_k - H_k y_k)^{\mathrm{T}} y_k}$

Broyden-Fletcher-Goldfarb-Shanno Method

Broyden-Fletcher-Goldfarb-Shanno (BFGS) Method

- Generally, BFGS method is preferred over the other methods.
  - Positive definiteness: BFGS updates guarantee that the updated Hessian approximation B_{n+1} remains positive definite if the initial matrix B_0 is positive definite and if the curvature condition ($y_n^{\mathrm{T}} \triangle\theta > 0$)is satisfied.
    - For instance, SR1 updates do not guarantee that the updated matrix will be positive definite.

  - Convergence rate: BFGS usually exhibits better convergence properties compared to the other methods.

  - Numerical stability: The BFGS update rule is numerically stable.
    - For example, SR1 updates can sometimes lead to ill-conditioned Hessian approximations, which can cause numerical difficulties.

- However, in this class, we will derive the Broyden method.
  - It usually converges slowly compared to BFGS method.
  - But, its derivation is easier to learn.
  - Based on this, we can learn the underlying ideas of how B_{n+1} is updated by B_n.

Broyden Method Derivation

Broyden Method

- Our goal is to find the matrix B_{n+1} that is close as possible to $B_{n,}$ while satisfying the secant equation.

- This can be achieved by minimizing the Frobenius norm of the difference between B_{n+1} and $B_{n,}$ subject to the secant equation constraint:
  - minimize $||B_{n+1} - B_n||_F^2 = ||A||_F^2 = tr(AA^T)$ $*\, trace\ property$
  - subject to $B_F(\theta_{n+1})\triangle\theta_n = \triangledown F(\theta_{n+1})$ - $\triangledown F(\theta_n)$

    $\rightarrow (B_F(\theta_n) + A)\ \triangle\theta_n = \triangledown F(\theta_{n+1})$ - $\triangledown F(\theta_n)$ $*\, B_F(\theta_{n+1}) - B_F(\theta_n) = A$

    $\rightarrow \triangledown F(\theta_{n+1})$ - $\triangledown F(\theta_n)$ - $B_F(\theta_n)\triangle\theta_n$ - $A\triangle\theta_n = 0$

- Here, Lagrange multiplier λ is used to form the Lagrangian:
  - $L(A, \lambda) = tr(AA^T) + \lambda^T(\triangledown F(\theta_{n+1}) - \triangledown F(\theta_n) - B_F(\theta_n)\triangle\theta_n - \mathrm{A}\triangle\theta_n)$
  - To find the optimal matrix A, we take the gradient of the Lagrangian with trace trick:
    - $\nabla L(A, \lambda) = \mathrm{tr}(\lambda^T(\triangledown F(\theta_{n+1}) - \triangledown F(\theta_n) - B_F(\theta_n)\triangle\theta_n - \mathrm{A}\triangle\theta_n) + AA^T)$
    - $\triangledown_A L(A, \lambda) = \mathrm{tr}(-\lambda^T \mathrm{d}A\triangle\theta_n + \mathrm{d}AA^T + A^T\mathrm{d}A)$

      $= \mathrm{tr}((-\triangle\theta_n\lambda^T + 2A^T)\mathrm{dA})$ $*\, dA\ is\ factored\ out$
    - $\frac{df}{dA^T} = (-\triangle\theta_n\lambda^T + 2A^T)$ * using the trace trick, $df = \mathrm{tr}(\frac{df}{dA^T}\mathrm{dA}) \leftrightarrow \triangledown_A L(A, \lambda) = \mathrm{tr}((-\triangle\theta_n\lambda^T + 2A^T)\mathrm{dA})$
    - As our goal is to find critical point, $-\triangle\theta_n\lambda^T + 2A^T = 0 \rightarrow -\lambda\triangle\theta_n{}^T + 2A = 0$
    - Therefore, $A = \frac{1}{2}\lambda\triangle\theta_n{}^T$

Broyden Method Derivation

Broyden Method

- Here, Lagrange multiplier λ is used to form the Lagrangian:
  - Continued:
    - We have $\bigtriangledown F(\theta_{n+1})$ - $\bigtriangledown F(\theta_n)$ - $B_F(\theta_n)\triangle\theta_n$ - $A\triangle\theta_n = 0$ and $A = \frac{1}{2}\lambda\triangle{\theta_n}^T$
    - Then, $\bigtriangledown F(\theta_{n+1}) - \bigtriangledown F(\theta_n) - B_n\triangle\theta_n = \frac{1}{2}\lambda\triangle{\theta_n}^T\triangle\theta_n$
    - $\lambda = \dfrac{2\bigtriangledown F(\theta_{n+1}) - 2\bigtriangledown F(\theta_n) - 2B_n\triangle\theta_n}{\triangle{\theta_n}^T\triangle\theta_n}$
    - $A = (\bigtriangledown F(\theta_{n+1}) - \bigtriangledown F(\theta_n) - B_n\triangle\theta_n)\dfrac{\triangle{\theta_n}^T}{\triangle{\theta_n}^T\triangle\theta_n}$
    - $B_{n+1} = B_n + \dfrac{(\bigtriangledown F(\theta_{n+1}) - \bigtriangledown F(\theta_n) - B_n\triangle\theta_n)}{\triangle{\theta_n}^T\triangle\theta_n}\triangle{\theta_n}^T$

Broyden	$B_k + \dfrac{y_k - B_k\Delta x_k}{\Delta x_k^{\mathrm{T}}\,\Delta x_k}\Delta x_k^{\mathrm{T}}$

Broyden Method Code

Samples (by 황주영)

$$f(x,y) = \begin{bmatrix} y\exp(x) - 2y \\ xy - y^3 \end{bmatrix}$$

```
def f(z):
    x,y = z
    fval = [y*np.exp(x)-2*y, x*y - y**3]
    return np.array(fval)

def dfdx(z):
    x,y = z
    fval = [y*np.exp(x), y]
    return np.array(fval)

def dfdy(z):
    x,y = z
    fval = [y*np.exp(x)-2, x-3*y**2]
    return np.array(fval)
```

```
def QuasiNewton(x):
    x0 = x
    #f(x0)
    B = np.array([dfdx(x0),dfdy(x0)]) #B0
    S = 0
    Y = 0

    i = 0
    while i<=18:
        x_prev = x
        x = x_prev - np.dot(np.linalg.inv(B),f(x_prev))
        S = x-x_prev
        Y = B * S

        prev_B = B
        B = prev_B + (Y - prev_B*S) * (S.transpose() / np.dot(S.transpose(),S))
        i+=1
        print(" iterator : " , i, " ,f(x) :  " ,f(x))


    return 0

QuasiNewton(np.array([1.0, 0.5]))
```

- $x_{k+1} = x_k - \boxed{B_k^{-1}} f(x_k)$
- $S_{k-1} = x_k - x_{k-1}$
- $y_{k-1} = B_k S_{k-1}$
- $B_{k+1} = B_k + (y_k - B_k S_k)\dfrac{S_k^T}{S_k^T S_k}$

Broyden Method Code

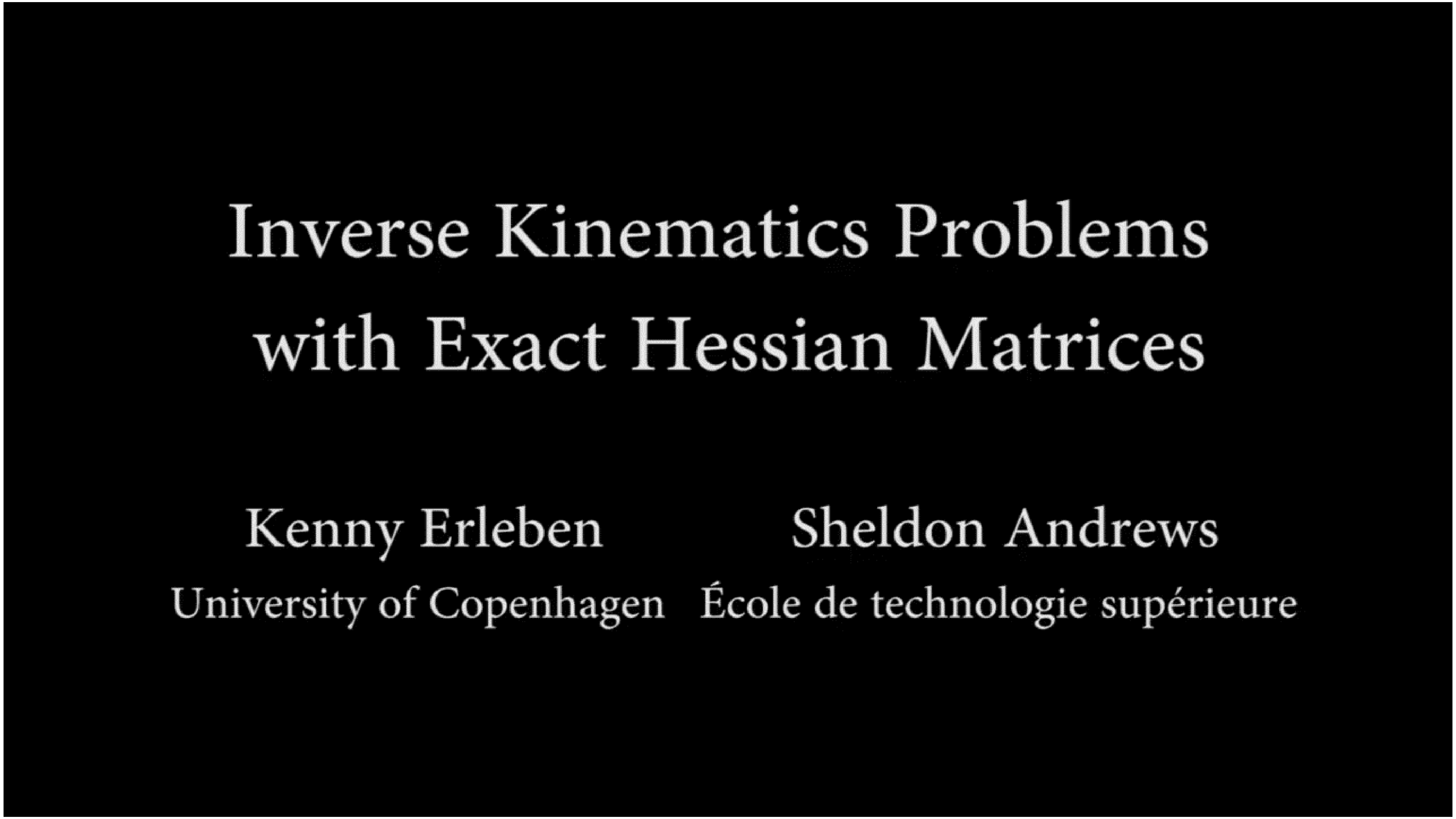
Samples (by 황주영)

```
 iterator :  1
x :  1.1480045216636174  ,  -0.6205998302270295 ,
f(x) :  [-0.71486714 -0.47343102]

 iterator :  2
x :  1.0601562531662045  ,  1.0479307959881072 ,
f(x) :  [ 0.92932811 -0.0398242 ]

 iterator :  3
x :  0.678092507450389  ,  0.22783150354792037 ,
f(x) :  [-0.00680848  0.14266474]

 iterator :  4
x :  0.788714828728349  ,  -0.059254183576068875 ,
f(x) :  [-0.0118844  -0.04652661]

 iterator :  5
x :  0.757979068252288  ,  0.04806308442447753 ,
f(x) :  [0.0064385  0.03631978]

 iterator :  6
x :  0.7830470740175223  ,  -0.032955811894472845 ,
f(x) :  [-0.00619996 -0.02577016]

 iterator :  7
x :  0.7658782830058444  ,  0.02611372224005245 ,
f(x) :  [0.00394011 0.01998213]

 iterator :  8
x :  0.7795193528214697  ,  -0.018846764088756067 ,
f(x) :  [-0.00340041 -0.01468472]

 iterator :  9
x :  0.7696857961862305  ,  0.014684431310338163 ,
f(x) :  [0.00233611 0.01129923]

 iterator :  10
x :  0.7773584321275426  ,  -0.010844181925140514 ,
f(x) :  [-0.00190551 -0.00842854]

 iterator :  11
x :  0.7716968006234874  ,  0.008356750210472438 ,
f(x) :  [0.00136578 0.00644829]

 iterator :  12
x :  0.7760631108438103  ,  -0.006243666705941585 ,
f(x) :  [-0.00107954 -0.00484524]

 iterator :  13
x :  0.7728024536375401  ,  0.004778790139937586 ,
f(x) :  [0.00079245 0.00369295]

 iterator :  14
x :  0.7752991597195883  ,  -0.003592869203825429 ,
f(x) :  [-0.00061525 -0.0027855 ]

 iterator :  15
x :  0.7734225902254784  ,  0.0027386727362738415 ,
f(x) :  [0.00045783 0.00211813]

 iterator :  16
x :  0.7748533501019602  ,  -0.002066188680225507 ,
f(x) :  [-0.00035182 -0.00160098]

 iterator :  17
x :  0.7737740986764748  ,  0.0015711565922652809 ,
f(x) :  [0.00026385 0.00121572]

 iterator :  18
x :  0.7745948861393955  ,  -0.0011876725857687597 ,
f(x) :  [-0.00020156 -0.00091996]
```

Exact Hessian Paper

K_{max} = maximum iteration

Inverse Kinematics Problems with Exact Hessian Matrices (2017)[IKEHM]



Reference

[1] https://en.wikipedia.org/wiki/Definite_matrix
[2] https://en.wikipedia.org/wiki/Secant_method
[3] https://eevibes.com/mathematics/numerical-analysis/what-is-the-secant-method-derivation-of-secant-method/
[4] https://en.wikipedia.org/wiki/Quasi-Newton_method
[5] https://towardsdatascience.com/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504
[6] https://www.stat.cmu.edu/~ryantibs/convexopt-F18/lectures/quasi-newton.pdf
[7] Practical Methods of Optimization, Fletcher, Roger, 1987
[8] http://www.princeton.edu/~aaa/Public/Teaching/ORF363_COS323/F14/ORF363_COS323_F14_Lec7.pdf
[LM] https://en.wikipedia.org/wiki/Lagrange_multiplier
[IKEHM] Erleben, K., & Andrews, S. (2017, November). Inverse kinematics problems with exact hessian matrices. In Proceedings of the 10th International Conference on Motion in Games (pp. 1-6).