Second-Order Optimization

Optimization Techniques (ENGG*6140)

School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Winter 2023

Newton's Method

Newton-Raphson root finding method

- We can find the root of a function $f : \boldsymbol{x} \mapsto f(\boldsymbol{x})$ by solving the equation $f(\boldsymbol{x}) \overset{\text{set}}{=} 0$.
- The root of function can be found iteratively where we get closer to the root over iterations.
- One of the iterative root-finding methods is the **Newton-Raphson method** [1]. In every iteration, it finds the next solution as:

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - \frac{f(\boldsymbol{x}^{(k)})}{\nabla f(\boldsymbol{x}^{(k)})}, \tag{1}$$

where $\nabla f(\boldsymbol{x}^{(k)})$ is the derivative of function w.r.t. $\boldsymbol{x}$.

Univariate Newton's method

- Recall Eq. (1): We saw that Newton-Raphson method solves $f(\boldsymbol{x}) \overset{\text{set}}{=} 0$ by:

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - \frac{f(\boldsymbol{x}^{(k)})}{\nabla f(\boldsymbol{x}^{(k)})}.$$

- In unconstrained optimization, we can find the extremum (minimum or maximum) of the function by setting its derivative to zero, i.e.,

$$\nabla f(\boldsymbol{x}) \overset{\text{set}}{=} 0.$$

- Therefore, the root of $\nabla f(\boldsymbol{x}) \overset{\text{set}}{=} 0$ can be found by Newton-Raphson method. We replace $f(\boldsymbol{x})$ with $\nabla f(\boldsymbol{x})$ in Eq. (1):

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - \eta^{(k)} \frac{\nabla f(\boldsymbol{x}^{(k)})}{\nabla^2 f(\boldsymbol{x}^{(k)})}, \tag{2}$$

where $\nabla^2 f(\boldsymbol{x}^{(k)})$ is the second derivative of function w.r.t. $\boldsymbol{x}$ and we have included a step size at iteration k denoted by $\eta^{(k)} > 0$. This step size can be either fixed or adaptive.

Multivariate Newton's method

- Recall Eq. (2):

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - \eta^{(k)} \frac{\nabla f(\boldsymbol{x}^{(k)})}{\nabla^2 f(\boldsymbol{x}^{(k)})}.$$

- If $\boldsymbol{x}$ is multivariate, i.e. $\boldsymbol{x} \in \mathbb{R}^d$, Eq. (2) is written as:

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - \eta^{(k)} \left(\nabla^2 f(\boldsymbol{x}^{(k)})\right)^{-1} \nabla f(\boldsymbol{x}^{(k)}), \tag{3}$$

  where $\nabla f(\boldsymbol{x}^{(k)}) \in \mathbb{R}^d$ is the gradient of function w.r.t. $\boldsymbol{x}$ and $\nabla^2 f(\boldsymbol{x}^{(k)}) \in \mathbb{R}^{d \times d}$ is the Hessian matrix w.r.t. $\boldsymbol{x}$.

- Because of the second derivative or the Hessian, this optimization method is a second-order method. The name of this method is the **Newton's method**.

Newton's Method for Unconstrained Optimization

Newton's Method for Unconstrained Optimization

- Consider the following optimization problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}). \tag{4}$$

  where $f(.)$ is a convex function.
- Iterative optimization can be first-order or second-order. Iterative optimization updates solution iteratively:

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}, \tag{5}$$

  The update continues until $\Delta\boldsymbol{x}$ becomes very small which is the convergence of optimization.
- Recall that in the first-order optimization, the step of updating is $\Delta\boldsymbol{x} := -\nabla f(\boldsymbol{x})$.

Newton's Method for Unconstrained Optimization

- Near the optimal point $\boldsymbol{x}^*$, gradient is very small so the second-order Taylor series expansion of function becomes:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^*) + \underbrace{\nabla f(\boldsymbol{x}^*)^\top}_{\approx 0}(\boldsymbol{x} - \boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^\top \nabla^2 f(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*)$$

$$\approx f(\boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^\top \nabla^2 f(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*). \tag{6}$$

  This shows that the function is almost quadratic near the optimal point.

- Following this intuition, Newton's method uses Hessian $\nabla^2 f(\boldsymbol{x})$ in its updating step:

$$\Delta \boldsymbol{x} := -\nabla^2 f(\boldsymbol{x})^{-1} \nabla f(\boldsymbol{x}). \tag{7}$$

- In the literature, this equation is sometimes restated to:

$$\nabla^2 f(\boldsymbol{x})\, \Delta \boldsymbol{x} := -\nabla f(\boldsymbol{x}). \tag{8}$$

Newton's Method for Equality Constrained Optimization

Newton's method for equality constrained optimization

- The optimization problem may have equality constraints:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{9}$$

- After a step of update by $\boldsymbol{p} = \Delta\boldsymbol{x}$, this optimization becomes:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}+\boldsymbol{p}) \\ \text{subject to} \quad & \boldsymbol{A}(\boldsymbol{x}+\boldsymbol{p}) = \boldsymbol{b}. \end{aligned} \tag{10}$$

- The Lagrangian of this optimization problem is:

$$\mathcal{L} = f(\boldsymbol{x}+\boldsymbol{p}) + \boldsymbol{\nu}^\top(\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{p}) - \boldsymbol{b}),$$

  where $\boldsymbol{\nu}$ is the dual variable.
- The second-order Taylor series expansion of function $f(\boldsymbol{x}+\boldsymbol{p})$ is:

$$f(\boldsymbol{x}+\boldsymbol{p}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^\top \nabla^2 f(\boldsymbol{x})\,\boldsymbol{p}. \tag{11}$$

- Substituting this into the Lagrangian gives:

$$\mathcal{L} = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^\top \nabla^2 f(\boldsymbol{x})\,\boldsymbol{p} + \boldsymbol{\nu}^\top(\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{p}) - \boldsymbol{b}).$$

Newton's method for equality constrained optimization

- We found:

$$\mathcal{L} = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^\top \nabla^2 f(\boldsymbol{x})\, \boldsymbol{p} + \boldsymbol{\nu}^\top (\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{p}) - \boldsymbol{b}).$$

- According to KKT conditions, the primal and dual residuals must be zero:

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \nabla f(\boldsymbol{x}) + \nabla^2 f(\boldsymbol{x})^\top \boldsymbol{p} + \boldsymbol{p}^\top \underbrace{\nabla^3 f(\boldsymbol{x})}_{\approx \boldsymbol{0}} \boldsymbol{p} + \boldsymbol{A}^\top \boldsymbol{\nu} \overset{\text{set}}{=} \boldsymbol{0}$$

$$\implies \nabla^2 f(\boldsymbol{x})^\top \boldsymbol{p} + \boldsymbol{A}^\top \boldsymbol{\nu} = -\nabla f(\boldsymbol{x}), \tag{12}$$

$$\nabla_{\boldsymbol{\nu}} \mathcal{L} = \boldsymbol{A}(\boldsymbol{x}+\boldsymbol{p}) - \boldsymbol{b} \overset{(a)}{=} \boldsymbol{A}\boldsymbol{p} \overset{\text{set}}{=} \boldsymbol{0}, \tag{13}$$

where we have $\nabla^3 f(\boldsymbol{x}) \approx 0$ because the third-order gradient is usually very small compared to the first and second gradients and (a) is because of the constraint $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ in problem (9).

- Eqs. (12) and (13) can be written as a system of equations:

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{x})^\top & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -\nabla f(\boldsymbol{x}) \\ \boldsymbol{0} \end{bmatrix}. \tag{14}$$

Solving this system of equations gives the desired step $\boldsymbol{p}$ (i.e., $\Delta\boldsymbol{x}$) for updating the solution at the iteration.

Starting with non-feasible initial point

- Newton's method can even start with a non-feasible point which does not satisfy all the constraints.
- If the initial point for optimization is not a feasible point, i.e., $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \neq \boldsymbol{0}$, Eq. (13) becomes:

$$\nabla_{\boldsymbol{\nu}} \mathcal{L} = \boldsymbol{A}(\boldsymbol{x} + \boldsymbol{p}) - \boldsymbol{b} \stackrel{\text{set}}{=} \boldsymbol{0} \implies \boldsymbol{A}\boldsymbol{p} = -(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}). \tag{15}$$

- Therefore, for the first iteration, we solve the following system rather than Eq. (14):

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{x})^\top & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\nu} \end{bmatrix} = - \begin{bmatrix} \nabla f(\boldsymbol{x}) \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \end{bmatrix}, \tag{16}$$

and we use Eq. (16) for the rest of iterations because the next points will be in the feasibility set (because we force the solutions to satisfy $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$).

Interior-Point and Barrier Methods: Newton's Method for Inequality Constrained Optimization

Interior-Point Method

- The optimization problem may have inequality constraints:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & y_i(\boldsymbol{x}) \leq 0, \quad i \in \{1, \ldots, m_1\}, \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{17}$$

- We can solve constrained optimization problems using **Barrier methods**, also known as **interior-point methods** [2, 3, 4, 5].
- Interior-point methods were first proposed in 1967 [6].
- The interior-point method is also referred to as the **Unconstrained Minimization Technique (UMT)** or **Sequential UMT (SUMT)** [7] because it converts the problem to an unconstrained problem and solves it iteratively.

Interior-Point Method

- The barrier methods or the interior-point methods, convert inequality constrained problems to equality constrained or unconstrained problems. Ideally, we can do this conversion using the indicator function $\mathbb{I}(.)$ which is zero if its input condition is satisfied and is infinity otherwise:

$$\mathbb{I}(\boldsymbol{x} \in \mathcal{S}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{S} \\ \infty & \text{if } \boldsymbol{x} \notin \mathcal{S}. \end{cases} \tag{18}$$

- The problem

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & y_i(\boldsymbol{x}) \leq 0, \quad i \in \{1, \ldots, m_1\}, \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

  is converted to:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) + \sum_{i=1}^{m_1} \mathbb{I}(y_i(\boldsymbol{x}) \leq 0) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{19}$$

Interior-Point Method

- The indicator function is not differentiable because it is not smooth:

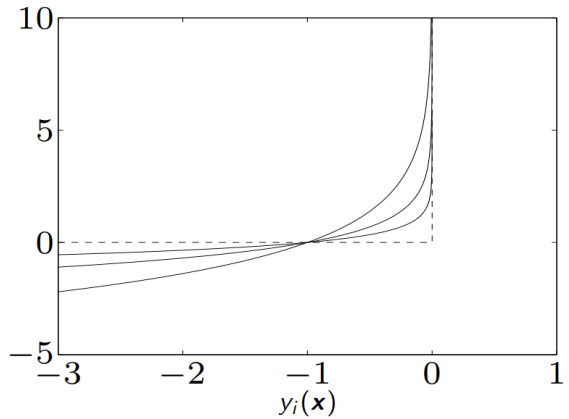
$$\mathbb{I}(y_i(\boldsymbol{x}) \leq 0) := \begin{cases} 0 & \text{if } y_i(\boldsymbol{x}) \leq 0 \\ \infty & \text{if } y_i(\boldsymbol{x}) > 0. \end{cases} \tag{20}$$

  Hence, we can approximate it with differentiable functions called the **barrier functions** [4, 8].

- One of the barrier functions is logarithm, named the **logarithmic barrier** or **log barrier** in short. It approximates the indicator function by:

$$\mathbb{I}(y_i(\boldsymbol{x}) \leq 0) \approx -\frac{1}{t} \log(-y_i(\boldsymbol{x})), \tag{21}$$

  where $t > 0$ (usually a large number such as $t = 10^6$) and the approximation becomes more accurate by $t \to \infty$.



Interior-Point Method

- The problem had become:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) + \sum_{i=1}^{m_1} \mathbb{I}(y_i(\boldsymbol{x}) \leq 0) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned}$$

- The log barrier:

$$\mathbb{I}(y_i(\boldsymbol{x}) \leq 0) \approx -\frac{1}{t} \log(-y_i(\boldsymbol{x})).$$

- It changes the problem to:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) - \frac{1}{t} \sum_{i=1}^{m_1} \log(-y_i(\boldsymbol{x})) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{22}$$

  This optimization problem is an equality constrained optimization problem which we already explained how to solve.

- Note that there exist many approximations for the barrier. One of mostly used methods is the logarithmic barrier.

Interior-Point Method

- If the problem is convex, the iterative solutions of the interior-point method satisfy:

$$\begin{aligned} &\{\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots\} \to \boldsymbol{x}^*, \\ &\{\boldsymbol{\nu}^{(0)}, \boldsymbol{\nu}^{(1)}, \boldsymbol{\nu}^{(2)}, \dots\} \to \boldsymbol{\nu}^*, \\ &f(\boldsymbol{x}^{(0)}) \geq f(\boldsymbol{x}^{(1)}) \geq f(\boldsymbol{x}^{(2)}) \geq \cdots \geq f(\boldsymbol{x}^*), \\ &g(\boldsymbol{\nu}^{(0)}) \leq g(\boldsymbol{\nu}^{(1)}) \leq \cdots \leq g(\boldsymbol{\nu}^*). \end{aligned} \tag{23}$$

- If the optimization problem is a convex problem, the solution of interior-point method is the global solution; otherwise, the solution is local.
- The interior-point and barrier methods are used in many optimization toolboxes such as CVX [9].

Accuracy of the log barrier method

Theorem (On the sub-optimality of log-barrier method)

Let the optimum of problems (17) and (22) be denoted by f^* and f_r^*, respectively. We have:

$$f^* - \frac{m_1}{t} \leq f_r^* \leq f^*, \tag{24}$$

meaning that the optimum of problem (22) is no more than m_1/t from the optimum of problem (17).

Proof.

See our tutorial [10] for proof. □

- The above theorem indicates that by $t \to \infty$, the log-barrier method is more accurate; i.e., the solution of problem (22) is more accurately close to the solution of problem (17).
- This is expected since the approximation in Eq. (21) gets more accurate by increasing t.
- Note that by increasing t, optimization gets more accurate but harder to solve and slower to converge.

Line-search in Newton's Method

Wolfe Conditions and Line-Search in Newton's Method

- Line-search for second-order optimization checks two conditions. These conditions are called the **Wolfe conditions** [11] for finding the suitable step size $\eta^{(k)}$, at iteration k of optimization.
- The step at iteration k is $\boldsymbol{p}^{(k)} = -\nabla^2 f(\boldsymbol{x}^{(k)})^{-1} \nabla f(\boldsymbol{x}^{(k)})$ according to Eq. (7). The Wolfe conditions are:

$$f(\boldsymbol{x}^{(k)} + \eta^{(k)} \boldsymbol{p}^{(k)}) \leq f(\boldsymbol{x}^{(k)}) + c_1 \eta^{(k)} \boldsymbol{p}^{(k)\top} f(\boldsymbol{x}^{(k)}), \tag{25}$$

$$-\boldsymbol{p}^{(k)\top} \nabla f(\boldsymbol{x}^{(k)} + \eta^{(k)} \boldsymbol{p}^{(k)}) \leq -c_2 \boldsymbol{p}^{(k)\top} f(\boldsymbol{x}^{(k)}), \tag{26}$$

  where $0 < c_1 < c_2 < 1$ are the parameters of Wolfe conditions. It is recommended in [12] to have $c_1 = 10^{-4}$ and $c_2 = 0.9$.
- The first condition is the **Armijo condition** [13] which ensures the step size $\eta^{(k)}$ decreases the function value sufficiently.
- The second condition is the **curvature condition** which ensures the step size $\eta^{(k)}$ decreases the function slope sufficiently. In quasi-Newton's method (introduced later), the curvature condition makes sure the approximation of Hessian matrix remains positive definite.
- The Armijo and curvature conditions give an upper-bound and lower-bound on the step size, respectively.

Wolfe Conditions and Line-Search in Newton's Method

- The Wolfe conditions were:

$$f(\boldsymbol{x}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)}) \leq f(\boldsymbol{x}^{(k)}) + c_1\eta^{(k)}\boldsymbol{p}^{(k)\top}\nabla f(\boldsymbol{x}^{(k)}), \tag{27}$$

$$-\boldsymbol{p}^{(k)\top}\nabla f(\boldsymbol{x}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)}) \leq -c_2\boldsymbol{p}^{(k)\top}\nabla f(\boldsymbol{x}^{(k)}), \tag{28}$$

- There also exists a **strong curvature condition**:

$$|\boldsymbol{p}^{(k)\top}\nabla f(\boldsymbol{x}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)})| \leq c_2|\boldsymbol{p}^{(k)\top}\nabla f(\boldsymbol{x}^{(k)})|, \tag{29}$$

  which can be used instead of the curvature condition.
- Note that the Wolfe conditions can also be used for line-search in first-order methods.

Fast Solving System of Equations in Newton's Method

Fast Solving System of Equations in Newton's Method

- In unconstrained Newton's method, the update of solution which is Eq. (8):

$$\nabla^2 f(\boldsymbol{x})\,\Delta\boldsymbol{x} := -\nabla f(\boldsymbol{x}),$$

  is in the form of a **system of linear equations**.

- In constrained Newton's method (and hence, in the interior-point method), the update of solution which is Eq. (14):

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{x})^\top & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -\nabla f(\boldsymbol{x}) \\ \boldsymbol{0} \end{bmatrix},$$

  is also in the form of a system of linear equations.

- Therefore, every iteration of all optimization methods is reduced to a system of equations such as:

$$\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}, \tag{30}$$

  where we need to calculate $\boldsymbol{z}$. Therefore, the optimization toolboxes, such as CVX [9], solve a system of equations to find the solution at every iteration.

- If the dimensionality of data or the number of constraints is large, the number of equations and the size of matrices in the system of equations increase. Solving the large system of equations is very time-consuming. Some methods have been developed to accelerate solving the system of equations. Here, we review some of them.

Decomposition Methods: LU decomposition

- We can use various matrix decomposition/factorization methods [14] for decomposing the coefficient matrix $\boldsymbol{M}$ and solving the system of equations (30), $\boldsymbol{Mz} = \boldsymbol{q}$. We review some of them here.
- **LU decomposition:** We use LU decomposition to decompose $\boldsymbol{M} = \boldsymbol{PLU}$. We have:

$$\boldsymbol{Mz} = \boldsymbol{P}\,\boldsymbol{L}\underbrace{\underbrace{\boldsymbol{Uz}}_{=\boldsymbol{w}_2}}_{=\boldsymbol{w}_1} = \boldsymbol{q},$$

where we define $\boldsymbol{w}_2 := \boldsymbol{Uz}$ and $\boldsymbol{w}_1 := \boldsymbol{Lw}_2$. Hence, we can solve the system of equations as:

1. LU decomposition: $\boldsymbol{M} = \boldsymbol{PLU}$
2. permutation: Solve $\boldsymbol{Pw}_1 = \boldsymbol{q}$ to find $\boldsymbol{w}_1$
3. forward subtraction: Solve $\boldsymbol{Lw}_2 = \boldsymbol{w}_1$ to find $\boldsymbol{w}_2$
4. backward subtraction: Solve $\boldsymbol{Uz} = \boldsymbol{w}_2$ to find $\boldsymbol{z}$

Decomposition Methods: Cholesky decomposition

- **Cholesky decomposition:** In most cases of optimization, the coefficient matrix is positive definite. For example, in Eq. (8), $\nabla^2 f(\boldsymbol{x})\,\Delta\boldsymbol{x} := -\nabla f(\boldsymbol{x})$, the coefficient matrix is the Hessian which is positive definite. Therefore, we can use Cholesky decomposition to decompose $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^\top$. We have:

$$\boldsymbol{M}\boldsymbol{z} = \boldsymbol{L}\underbrace{\boldsymbol{L}^\top \boldsymbol{z}}_{=\boldsymbol{w}_1} = \boldsymbol{q},$$

  where we define $\boldsymbol{w}_1 := \boldsymbol{L}^\top \boldsymbol{z}$.

- Hence, we can solve the system of equations as:
  1. Cholesky decomposition: $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^\top$
  2. forward subtraction: Solve $\boldsymbol{L}\boldsymbol{w}_1 = \boldsymbol{q}$ to find $\boldsymbol{w}_1$
  3. backward subtraction: Solve $\boldsymbol{L}^\top \boldsymbol{z} = \boldsymbol{w}_1$ to find $\boldsymbol{z}$

Decomposition Methods: Schur complement

- Assume the system of equations can be divided into blocks:

$$\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q} \implies \boldsymbol{M}\boldsymbol{z} = \begin{bmatrix} \boldsymbol{M}_{11} & \boldsymbol{M}_{12} \\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \end{bmatrix}.$$

From this, we have:

$$\begin{aligned} &\boldsymbol{M}_{11}\boldsymbol{z}_1 + \boldsymbol{M}_{12}\boldsymbol{z}_2 = \boldsymbol{q}_1 \implies \boldsymbol{z}_1 = \boldsymbol{M}_{11}^{-1}(\boldsymbol{q}_1 - \boldsymbol{M}_{12}\boldsymbol{z}_2). \\ &\boldsymbol{M}_{21}\boldsymbol{z}_1 + \boldsymbol{M}_{22}\boldsymbol{z}_2 = \boldsymbol{q}_2 \\ &\implies \boldsymbol{M}_{21}(\boldsymbol{M}_{11}^{-1}(\boldsymbol{q}_1 - \boldsymbol{M}_{12}\boldsymbol{z}_2)) + \boldsymbol{M}_{22}\boldsymbol{z}_2 = \boldsymbol{q}_2 \\ &\implies (\boldsymbol{M}_{22} - \boldsymbol{M}_{21}\boldsymbol{M}_{11}^{-1}\boldsymbol{M}_{12})\boldsymbol{z}_2 = \boldsymbol{q}_2 - \boldsymbol{M}_{21}\boldsymbol{M}_{11}^{-1}\boldsymbol{q}_1. \end{aligned}$$

The term $(\boldsymbol{M}_{22} - \boldsymbol{M}_{21}\boldsymbol{M}_{11}^{-1}\boldsymbol{M}_{12})$ is the Schur complement [15] of block matrix $\boldsymbol{M}_{11}$ in matrix $\boldsymbol{M}$.

- We assume that the block matrix $\boldsymbol{M}_{11}$ is not singular so its inverse exists. We use the Schur complement to solve the system of equations as:
  1. Calculate $\boldsymbol{M}_{11}^{-1}\boldsymbol{M}_{12}$ and $\boldsymbol{M}_{11}^{-1}\boldsymbol{q}_1$
  2. Calculate $\widetilde{\boldsymbol{M}} := \boldsymbol{M}_{22} - \boldsymbol{M}_{21}(\boldsymbol{M}_{11}^{-1}\boldsymbol{M}_{12})$ and $\widetilde{\boldsymbol{q}} := \boldsymbol{q}_2 - \boldsymbol{M}_{21}(\boldsymbol{M}_{11}^{-1}\boldsymbol{q}_1)$
  3. Solve $\widetilde{\boldsymbol{M}}\boldsymbol{z}_2 = \widetilde{\boldsymbol{q}}$ (as derived above) to find $\boldsymbol{z}_2$
  4. Solve $\boldsymbol{M}_{11}\boldsymbol{z}_1 = \boldsymbol{q}_1 - \boldsymbol{M}_{12}\boldsymbol{z}_2$ (as derived above) to find $\boldsymbol{z}_1$

Conjugate Gradient Method

- The **conjugate gradient** method, proposed in 1952 [16], iteratively solves Eq. (30), $\boldsymbol{Mz} = \boldsymbol{q}$, faster than the regular solution. Its pacing shows off better when the matrices are very large. A good book on conjugate gradient is [17, Chapter 2].
- Note that **truncated Newton's methods** [18], which approximate the Hessian for large-scale optimization, usually use conjugate gradient as their approximation method for calculating the search direction.
- The solution to Eq. (30), $\boldsymbol{Mz} = \boldsymbol{q}$, should satisfy $\boldsymbol{z} = \boldsymbol{M}^{-1}\boldsymbol{q}$. The conjugate gradient method approximates this solution. According to the first-order optimizality condition, $\nabla f(\boldsymbol{z}^*) = \boldsymbol{0}$, the solution to Eq. (30) minimizes the function:

$$\nabla f(\boldsymbol{z}^*) = \boldsymbol{0} \implies \boldsymbol{Mz} - \boldsymbol{q} = \boldsymbol{0} \implies f(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{Mz} - \boldsymbol{z}^\top \boldsymbol{q},$$

because Eq. (30) is the gradient of this function, i.e.,

$$\nabla f(\boldsymbol{z}) = \boldsymbol{Mz} - \boldsymbol{q}. \tag{31}$$

Conjugate Gradient Method

- Conjugate gradient iteratively solves Eq. (30), $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}$, as:

$$\boldsymbol{z}^{(k+1)} := \boldsymbol{z}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)}. \tag{32}$$

- It starts by moving toward minus gradient as gradient descent does. Then, it uses the conjugate of gradient. This is the reason for the name of this method.
- At iteration k, the **residual (error)** for fulfilling Eq. (30), $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}$, is:

$$\boldsymbol{r}^{(k)} := \boldsymbol{q} - \boldsymbol{M}\boldsymbol{z}^{(k)} \overset{(31)}{=} -\nabla f(\boldsymbol{z}^{(k)}). \tag{33}$$

- We also have:

$$\begin{aligned}\boldsymbol{r}^{(k+1)} - \boldsymbol{r}^{(k)} &\overset{(33)}{=} \boldsymbol{q} - \boldsymbol{M}\boldsymbol{z}^{(k+1)} - \boldsymbol{q} + \boldsymbol{M}\boldsymbol{z}^{(k)} \\ &\overset{(32)}{=} \boldsymbol{M}(-\boldsymbol{z}^{(k)} - \eta^{(k)}\boldsymbol{p}^{(k)} + \boldsymbol{z}^{(k)}) = -\eta^{(k)}\boldsymbol{M}\boldsymbol{p}^{(k)}.\end{aligned}$$

- Initially, the direction is this residual, $\boldsymbol{p}^{(0)} = \boldsymbol{r}^{(0)} = -\nabla f(\boldsymbol{z}^{(0)})$ as in gradient descent.
- If we take derivative of $f(\boldsymbol{z}^{(k+1)}) \overset{(32)}{=} f(\boldsymbol{z}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)})$ w.r.t. $\eta^{(k)}$, we have the learning rate:

$$\frac{\partial}{\partial \eta^{(k)}} f(\boldsymbol{z}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)}) \overset{\text{set}}{=} 0 \implies \eta^{(k)} = \frac{\boldsymbol{p}^{(k)\top}(\boldsymbol{q} - \boldsymbol{M}\boldsymbol{x}^{(k)})}{\boldsymbol{p}^{(k)\top}\boldsymbol{M}\boldsymbol{p}^{(k)}} \overset{(33)}{=} \frac{\boldsymbol{p}^{(k)\top}\boldsymbol{r}^{(k)}}{\boldsymbol{p}^{(k)\top}\boldsymbol{M}\boldsymbol{p}^{(k)}}.$$

Conjugate Gradient Method

- The direction of update $\boldsymbol{p}$ is found by a linear combination of the residual (which was initialized by negative gradient as in gradient descent) and the previous direction:

$$\boldsymbol{p}^{(k+1)} := \boldsymbol{r}^{(k+1)} + \beta^{(k+1)} \boldsymbol{p}^{(k)}. \tag{34}$$

  The weight of previous direction in this linear combination is β:

$$\beta = \frac{\boldsymbol{r}^{(k+1)\top} \boldsymbol{r}^{(k+1)}}{\boldsymbol{r}^{(k)\top} \boldsymbol{r}^{(k)}}, \tag{35}$$

  which gets smaller if the residual of this step is much smaller than the residual of previous iteration.

- The conjugate gradient method returns a $\boldsymbol{z}$ as an approximation to the solution to Eq. (30), $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}$.

Conjugate Gradient Method: Algorithm

Initialize: $\boldsymbol{z}^{(0)}$

$\boldsymbol{r}^{(0)} := -\nabla f(\boldsymbol{z}^{(0)}) = \boldsymbol{q} - \boldsymbol{M}\boldsymbol{z}^{(0)}$, $\boldsymbol{p}^{(0)} := \boldsymbol{r}^{(0)}$

for *iteration* $k = 0, 1, \ldots$ **do**

  $\eta^{(k)} = \frac{\boldsymbol{p}^{(k)\top}\boldsymbol{r}^{(k)}}{\boldsymbol{p}^{(k)\top}\boldsymbol{M}\boldsymbol{p}^{(k)}}$

  $\boldsymbol{z}^{(k+1)} := \boldsymbol{z}^{(k)} + \eta^{(k)}\boldsymbol{p}^{(k)}$

  $\boldsymbol{r}^{(k+1)} := \boldsymbol{r}^{(k)} - \eta^{(k)}\boldsymbol{M}\boldsymbol{p}^{(k)}$

  if $\|\boldsymbol{r}^{(k+1)}\|_2$ *is small* **then**

    Break the loop.

  $\beta^{(k+1)} := \frac{\boldsymbol{r}^{(k+1)\top}\boldsymbol{r}^{(k+1)}}{\boldsymbol{r}^{(k)\top}\boldsymbol{r}^{(k)}} = \frac{\|\boldsymbol{r}^{(k+1)}\|_2^2}{\|\boldsymbol{r}^{(k)}\|_2^2}$

  $\boldsymbol{p}^{(k+1)} := \boldsymbol{r}^{(k+1)} + \beta^{(k+1)}\boldsymbol{p}^{(k)}$

Return $\boldsymbol{z}^{(k+1)}$

Algorithm : The conjugate gradient method

Relation of Conjugate Gradient Method to Krylov subspace

Definition (Conjugate vectors)

Two non-zero vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are **conjugate** if $\boldsymbol{x}^\top \boldsymbol{M}\boldsymbol{y} = \boldsymbol{0}$ where $\boldsymbol{M} \succ \boldsymbol{0}$.

Definition (Krylov subspace [19])

The order-r Krylov subspace, denoted by $\mathcal{K}_r$, is spanned by the following bases:

$$\mathcal{K}_r(\boldsymbol{M}, \boldsymbol{q}) = \textbf{span}\{\boldsymbol{q}, \boldsymbol{M}\boldsymbol{q}, \boldsymbol{M}^2\boldsymbol{q}, \ldots, \boldsymbol{M}^{r-1}\boldsymbol{q}\}. \tag{36}$$

- The solution to Eq. (30), $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}$, should satisfy $\boldsymbol{z} = \boldsymbol{M}^{-1}\boldsymbol{q}$. Therefore, the solution to Eq. (30) lies in the Krylov subspace.
- The conjugate gradient method approximates this solution lying in the Krylov subspace. Every iteration of conjugate gradient can be seen as projection onto the Krylov subspace.

Nonlinear Conjugate Gradient Method

- As we saw, conjugate gradient is for solving linear equations, such as Eq. (30), $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}$. **Nonlinear Conjugate Gradient (NCG)** generalizes conjugate gradient to nonlinear functions.
- Recall that conjugate gradient solves Eq. (30):

$$\boldsymbol{M}\boldsymbol{z} = \boldsymbol{q} \implies \boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{z} = \boldsymbol{M}^\top \boldsymbol{q} \implies 2\boldsymbol{M}^\top(\boldsymbol{M}\boldsymbol{z} - \boldsymbol{q}) = \boldsymbol{0}.$$

- The goal of NCG is to find the minimum of a quadratic function:

$$f(\boldsymbol{z}) = \|\boldsymbol{M}\boldsymbol{z} - \boldsymbol{q}\|_2^2, \tag{37}$$

  using its gradient. The gradient of this function is $\nabla f(\boldsymbol{z}) = 2\boldsymbol{M}^\top(\boldsymbol{M}\boldsymbol{z} - \boldsymbol{q})$ which was found above.
- The NCG method is very similar to the conjugate gradient method. It uses steepest descent for updating the solution:

$$\eta^{(k+1)} := \arg\min_{\eta} f(\boldsymbol{z}^{(k+1)} + \eta\, \boldsymbol{p}^{(k+1)}).$$

- The direction for update is found by a linear combination of the residual, initialized by negative gradient as in gradient descent, and the direction of previous iteration (as in the conjugate gradient method):

$$\boldsymbol{p}^{(k+1)} := \boldsymbol{r}^{(k+1)} + \beta^{(k+1)}\boldsymbol{p}^{(k)}. \tag{38}$$

Nonlinear Conjugate Gradient Method

- We said: $\boldsymbol{p}^{(k+1)} := \boldsymbol{r}^{(k+1)} + \beta^{(k+1)}\boldsymbol{p}^{(k)}$.
- Several formulas exist for the weight β for the previous direction in the linear combination:

$$
\begin{aligned}
\beta_1^{(k+1)} &:= \frac{\boldsymbol{r}^{(k+1)\top}\boldsymbol{r}^{(k+1)}}{\boldsymbol{r}^{(k)\top}\boldsymbol{r}^{(k)}} = \frac{\|\boldsymbol{r}^{(k+1)}\|_2^2}{\|\boldsymbol{r}^{(k)}\|_2^2}, \\
\beta_2^{(k+1)} &:= \frac{\boldsymbol{r}^{(k+1)\top}\left(\boldsymbol{r}^{(k+1)} - \boldsymbol{r}^{(k)}\right)}{\boldsymbol{r}^{(k)\top}\boldsymbol{r}^{(k)}}, \\
\beta_3^{(k+1)} &:= -\frac{\boldsymbol{r}^{(k+1)\top}\left(\boldsymbol{r}^{(k+1)} - \boldsymbol{r}^{(k)}\right)}{\boldsymbol{p}^{(k)\top}\left(\boldsymbol{r}^{(k+1)} - \boldsymbol{r}^{(k)}\right)}, \\
\beta_4^{(k+1)} &:= -\frac{\boldsymbol{r}^{(k)\top}\boldsymbol{r}^{(k)}}{\boldsymbol{p}^{(k)\top}\left(\boldsymbol{r}^{(k+1)} - \boldsymbol{r}^{(k)}\right)}.
\end{aligned}
\tag{39}
$$

  The β_1, β_2, β_3, and β_4 are the formulas for **Fletcher-Reeves** (1964) [20], **Polak-Ribière** (1969) [21], **Hestenes-Stiefel** (1952) [16], and **Dai-Yuan** (1999) [22] methods, respectively.
- Fletcher-Reeves (β_1) was also used in the conjugate gradient method.
- In all these formulas, β gets smaller if the residual of next iteration gets much smaller than the previous residual.
- The NCG method returns a $\boldsymbol{z}$ as an approximation to the minimizer of the nonlinear function in Eq. (37).

Nonlinear Conjugate Gradient Method: Algorithm

1 Initialize: $\boldsymbol{z}^{(0)}$
2 $\boldsymbol{r}^{(0)} := -\nabla f(\boldsymbol{z}^{(0)})$, $\boldsymbol{p}^{(0)} := \boldsymbol{r}^{(0)}$
3 **for** *iteration* $k = 0, 1, \ldots$ **do**
4 $\quad \boldsymbol{r}^{(k+1)} := -\nabla f(\boldsymbol{z}^{(k+1)})$
5 $\quad$ Compute $\beta^{(k+1)}$ by one of Eqs. (217)
6 $\quad \boldsymbol{p}^{(k+1)} := \boldsymbol{r}^{(k+1)} + \beta^{(k+1)} \boldsymbol{p}^{(k)}$
7 $\quad \eta^{(k+1)} := \arg\min_{\eta} f(\boldsymbol{z}^{(k+1)} + \eta\, \boldsymbol{p}^{(k+1)})$
8 $\quad \boldsymbol{z}^{(k+1)} := \boldsymbol{z}^{(k)} + \eta^{(k+1)} \boldsymbol{p}^{(k+1)}$
9 **Return** $\boldsymbol{z}^{(k+1)}$

Algorithm : The NCG method

Quasi-Newton's Methods

Hessian Approximation

- Similar to what we did in Eq. (6) for Taylor approximation of function, we can approximate the function at the updated solution by its second-order Taylor series:

$$f(\boldsymbol{x}^{(k+1)}) = f(\boldsymbol{x}^{(k)} + \boldsymbol{p}^{(k)}) \approx f(\boldsymbol{x}^{(k)}) + \nabla f(\boldsymbol{x}^{(k)})^\top \boldsymbol{p}^{(k)} + \frac{1}{2}\boldsymbol{p}^{(k)\top} \boldsymbol{B}^{(k)} \boldsymbol{p}^{(k)}.$$

  where $\boldsymbol{p} = \Delta \boldsymbol{x}$ is the step size and $\boldsymbol{B}^{(k)} = \nabla^2 f(\boldsymbol{x}^{(k)})$ is the Hessian matrix at iteration k.
- Taking derivative from this equation w.r.t. $\boldsymbol{p}$ gives:

$$\nabla f(\boldsymbol{x}^{(k)} + \boldsymbol{p}^{(k)}) \approx \nabla f(\boldsymbol{x}^{(k)}) + \boldsymbol{B}^{(k)} \boldsymbol{p}^{(k)}. \tag{40}$$

  This equation is called the **secant equation**.
- Setting this derivative to zero, for optimization, gives:

$$\boldsymbol{B}^{(k)} \boldsymbol{p}^{(k)} = -\nabla f(\boldsymbol{x}^{(k)}) \implies \boldsymbol{p}^{(k)} = -\boldsymbol{H}^{(k)} \nabla f(\boldsymbol{x}^{(k)}), \tag{41}$$

  where $\boldsymbol{H}^{(k)} := (\boldsymbol{B}^{(k)})^{-1}$ is the inverse of Hessian matrix. This equation is the previously found Eqs. (7) and (8) for Newton's method.
- Note that although the letter H seems to be used for Hessian, literature uses $\boldsymbol{H}$ to denote the (approximation of) inverse of Hessian.

Hessian Approximation

- We found:

$$\boldsymbol{p}^{(k)} = -\boldsymbol{H}^{(k)} \nabla f(\boldsymbol{x}^{(k)}).$$

- Considering the step size, we can write the step $\boldsymbol{s}^{(k)}$ as:

$$\mathbb{R}^d \ni \boldsymbol{s}^{(k)} := \Delta \boldsymbol{x} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} = -\eta^{(k)} \boldsymbol{H}^{(k)} \nabla f(\boldsymbol{x}^{(k)}), \tag{42}$$

  where the step size η is found by the Wolfe conditions in line-search.
- Computation of the Hessian matrix or its inverse is usually expensive in Newton's method.
- One way to approximate the Newton's solution at every iteration is the conjugate gradient method, which was introduced before.
- Another approach for approximating the solution is the **quasi-Newton's methods** which approximate the (inverse) Hessian matrix. They approximate the Hessian based on the step $\boldsymbol{p}^{(k)}$, the difference of gradients between iterations:

$$\mathbb{R}^d \ni \boldsymbol{y}^{(k)} := \nabla f(\boldsymbol{x}^{(k+1)}) - \nabla f(\boldsymbol{x}^{(k)}), \tag{43}$$

  and the previous approximated Hessian $\boldsymbol{B}^{(k)}$ or its inverse $\boldsymbol{H}^{(k)}$.

Hessian Approximation

- Some papers approximate the Hessian by a diagonal matrix [23, Appendix C.1], [24]. In this situation, the inverse of approximated Hessian is simply the inverse of its diagonal elements.
- Some methods use a dense approximation for Hessian matrix. Examples for these are BFGS, DFP, Broyden, and SR1 methods, which will be introduced in the following. Some other methods, such as LBFGS introduced later, approximate the Hessian matrix only by a scalar.
- The most well-known algorithm for quasi-Newton's method is **Broyden-Fletcher-Goldfarb-Shanno (BFGS)** (1987, 1996) [25, 26].
- **Limited-memory BFGS (LBFGS)** (1980, 1989) [27, 28] is a simplified version of BFGS which utilizes less memory.
- Some other quasi-Newton's methods are **Davidon-Fletcher-Powell (DFP)** (1991, 1987) [29, 25], **Broyden method** (1965) [30], and **Symmetric Rank-one (SR1)** (1991) [31].
- In the following, we review the approximations of Hessian matrix and its inverse by different methods. More explanation on these methods can be found in Chapter 6 of Nocedal's book [12, Chapter 6].

Quasi-Newton's Algorithms

- We define:

$$\mathbb{R} \ni \rho^{(k)} := \frac{1}{\boldsymbol{y}^{(k)\top}\boldsymbol{s}^{(k)}}, \tag{44}$$

$$\mathbb{R}^{d\times d} \ni \boldsymbol{V}^{(k)} := \boldsymbol{I} - \rho^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{s}^{(k)\top} = \boldsymbol{I} - \frac{\boldsymbol{y}^{(k)}\boldsymbol{s}^{(k)\top}}{\boldsymbol{y}^{(k)\top}\boldsymbol{s}^{(k)}}, \tag{45}$$

  where $\boldsymbol{I}$ is the identity matrix.

- **Broyden-Fletcher-Goldfarb-Shanno (BFGS):** The approximations in BFGS method are:

$$\begin{aligned}\boldsymbol{B}^{(k+1)} &:= \boldsymbol{B}^{(k)} + \rho^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)\top} - \frac{\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\top}\boldsymbol{B}^{(k)\top}}{\boldsymbol{s}^{(k)\top}\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}},\\ \boldsymbol{H}^{(k+1)} &:= \boldsymbol{V}^{(k)\top}\boldsymbol{H}^{(k)}\boldsymbol{V}^{(k)} + \rho^{(k)}\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\top}.\end{aligned} \tag{46}$$

- **Davidon-Fletcher-Powell (DFP):** The approximations in DFP method are:

$$\begin{aligned}\boldsymbol{B}^{(k+1)} &:= \boldsymbol{V}^{(k)}\boldsymbol{B}^{(k)}\boldsymbol{V}^{(k)\top} + \rho^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)\top},\\ \boldsymbol{H}^{(k+1)} &:= \boldsymbol{H}^{(k)} + \rho^{(k)}\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\top} - \frac{\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)\top}\boldsymbol{H}^{(k)\top}}{\boldsymbol{y}^{(k)\top}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}.\end{aligned} \tag{47}$$

- Comparing Eqs. (46) and (47) shows that the BFGS and DFP methods are dual of each other. Experiments have shown that BFGS often outperforms DFP [32].

Quasi-Newton's Algorithms

- **Broyden method:** The approximations in Broyden method are:

$$\begin{aligned} \boldsymbol{B}^{(k+1)} &:= \boldsymbol{B}^{(k)} + \frac{(\boldsymbol{y}^{(k)} - \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)})\,\boldsymbol{s}^{(k)\top}}{\boldsymbol{s}^{(k)\top}\boldsymbol{s}^{(k)}}, \\ \boldsymbol{H}^{(k+1)} &:= \boldsymbol{H}^{(k)} + \frac{(\boldsymbol{s}^{(k)} - \boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)})\boldsymbol{s}^{(k)\top}\boldsymbol{H}^{(k)}}{\boldsymbol{s}^{(k)\top}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}. \end{aligned} \tag{48}$$

- **Symmetric Rank-one (SR1):** The approximations in SR1 method are:

$$\begin{aligned} \boldsymbol{B}^{(k+1)} &:= \boldsymbol{B}^{(k)} + \frac{(\boldsymbol{y}^{(k)} - \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)})(\boldsymbol{y}^{(k)} - \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)})^\top}{(\boldsymbol{y}^{(k)} - \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)})^\top\boldsymbol{s}^{(k)}}, \\ \boldsymbol{H}^{(k+1)} &:= \boldsymbol{H}^{(k)} + \frac{(\boldsymbol{s}^{(k)} - \boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)})(\boldsymbol{s}^{(k)} - \boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)})^\top}{(\boldsymbol{s}^{(k)} - \boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)})^\top\boldsymbol{y}^{(k)}}. \end{aligned} \tag{49}$$

Quasi-Newton's Algorithms: LBFGS

- The above methods, including BFGS, approximate the inverse Hessian matrix by a dense $(d \times d)$ matrix. For large d, storing this matrix is very memory-consuming.
- Hence, **Limited-memory BFGS (LBFGS)** [27, 28] was proposed, by Nocedal et al. in 1980's, which uses much less memory than BFGS.
- In LBFGS, the inverse of Hessian is a scalar multiplied by identity matrix, i.e., $\boldsymbol{H}^{(0)} := \gamma^{(k)}\boldsymbol{I}$; therefore, it approximates the $(d \times d)$ matrix by a scalar.
- It uses a memory of m previous variables and recursively calculates the updating direction of solution. In other words, it has recursive unfoldings which approximate the descent directions in optimization.
- The number of recursions is a small integer, for example $m = 10$; hence, not much memory is used.
- By m times recursion on Eq. (46), LBFGS approximates the inverse Hessian as [28, Algorithm 2.1]:

$$\begin{aligned}\boldsymbol{H}^{(k+1)} :=& (\boldsymbol{V}^{(k)\top} \dots \boldsymbol{V}^{(k-m)\top})\boldsymbol{H}^{(0)}(\boldsymbol{V}^{(k-m)} \dots \boldsymbol{V}^{(k)}) \\ &+ \rho^{(k-m)}(\boldsymbol{V}^{(k)\top} \dots \boldsymbol{V}^{(k-m+1)\top})\boldsymbol{s}^{(k-m)}\boldsymbol{s}^{(k-m)\top}(\boldsymbol{V}^{(k-m+1)} \dots \boldsymbol{V}^{(k)}) \\ &+ \rho^{(k-m+1)}(\boldsymbol{V}^{(k)\top} \dots \boldsymbol{V}^{(k-m+2)\top})\boldsymbol{s}^{(k-m+1)}\boldsymbol{s}^{(k-m+1)\top} \\ &\qquad (\boldsymbol{V}^{(k-m+2)} \dots \boldsymbol{V}^{(k)}) + \dots + \rho^{(k)}\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\top}.\end{aligned}$$

Quasi-Newton's Algorithms: LBFGS

- The LBFGS algorithm can be implemented as shown in the following algorithm [33] which is based on the algorithm in Nocedal's book [12, Chapter 6].
- As this algorithm shows, every iteration of optimization calls a **recursive function** for up to m recursions and uses the stored previous m memory to calculate the direction $\boldsymbol{p}$ for updating the solution.
- As Eq. (41) states, the initial updating direction is the Newton's method direction, which is $\boldsymbol{p} = -\boldsymbol{H}^{(0)} \nabla f(\boldsymbol{x}^0)$.

Initialize the solution $\boldsymbol{x}^{(0)}$
$\boldsymbol{H}^{(0)} := \frac{1}{\|\nabla f(\boldsymbol{x}^{(0)})\|_2} \boldsymbol{I}$
for $k = 0, 1, \ldots$ *(until convergence)* **do**
  $\boldsymbol{p}^{(k)} \leftarrow \text{GetDirection}(-\nabla f(\boldsymbol{x}^{(k)}), k, 1)$
  $\eta^{(k)} \leftarrow$ Line-search with Wolfe conditions
  $\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - \eta^{(k)} \boldsymbol{p}^{(k)}$
  $\boldsymbol{s}^{(k)} := \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} = \eta^{(k)} \boldsymbol{p}^{(k)}$
  $\boldsymbol{y}^{(k)} := \nabla f(\boldsymbol{x}^{(k+1)}) - \nabla f(\boldsymbol{x}^{(k)})$
  $\gamma^{(k+1)} := \frac{\boldsymbol{s}^{(k)\top} \boldsymbol{y}^{(k)}}{\boldsymbol{y}^{(k)\top} \boldsymbol{y}^{(k)}}$
  $\boldsymbol{H}^{(k+1)} := \gamma^{(k+1)} \boldsymbol{I}$
  Store $\boldsymbol{y}^{(k)}$, $\boldsymbol{s}^{(k)}$, and $\boldsymbol{H}^{(k+1)}$
return $\boldsymbol{x}^{(k+1)}$

// recursive function:
Function GetDirection($\boldsymbol{p}, k, n\_\text{recursion}$)
if $k > 0$ **then**
  // do up to m recursions:
  if $n\_recursion > m$ **then**
    return $\boldsymbol{p}$
  $\rho^{(k-1)} := \frac{1}{\boldsymbol{y}^{(k-1)\top} \boldsymbol{s}^{(k-1)}}$
  $\tilde{\boldsymbol{p}} := \boldsymbol{p} - \rho^{(k-1)} (\boldsymbol{s}^{(k-1)\top} \boldsymbol{p}) \boldsymbol{y}^{(k-1)}$
  $\widehat{\boldsymbol{p}} := \text{GetDirection}(\tilde{\boldsymbol{p}}, k-1, n\_\text{recursion} + 1)$
  return $\widehat{\boldsymbol{p}} - \rho^{(k-1)} (\boldsymbol{y}^{(k-1)\top} \widehat{\boldsymbol{p}}) \boldsymbol{s}^{(k-1)} + \rho^{(k-1)} (\boldsymbol{s}^{(k-1)\top} \boldsymbol{s}^{(k-1)}) \boldsymbol{p}$
else
  return $\boldsymbol{H}^{(0)} \boldsymbol{p}$

Acknowledgement

- Some slides of this slide deck are inspired by the lectures of Prof. Stephen Boyd at the Stanford University.
- Some slides of this slide deck, especially the Quasi-Newton's methods, are inspired by the teachings of Prof. Reshad Hosseini at the University of Tehran.
- Our tutorial also has the materials of this slide deck: [10]

References

[1] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, vol. 12. Springer Science & Business Media, 2013.

[2] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[3] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of computational and applied mathematics*, vol. 124, no. 1-2, pp. 281–302, 2000.

[4] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[5] M. Wright, "The interior-point revolution in optimization: history, recent developments, and lasting consequences," *Bulletin of the American mathematical society*, vol. 42, no. 1, pp. 39–56, 2005.

[6] I. Dikin, "Iterative solution of problems of linear and quadratic programming," in *Doklady Akademii Nauk*, vol. 174, pp. 747–748, Russian Academy of Sciences, 1967.

[7] A. V. Fiacco and G. P. McCormick, "The sequential unconstrained minimization technique (SUMT) without parameters," *Operations Research*, vol. 15, no. 5, pp. 820–827, 1967.

[8] Y. Nesterov, *Lectures on convex optimization*, vol. 137. Springer, 2018.

References (cont.)

[9] M. Grant, S. Boyd, and Y. Ye, "CVX: Matlab software for disciplined convex programming," 2009.

[10] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "KKT conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey," *arXiv preprint arXiv:2110.01858*, 2021.

[11] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[12] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2 ed., 2006.

[13] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[14] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU press, 2013.

[15] F. Zhang, *The Schur complement and its applications*, vol. 4. Springer Science & Business Media, 2006.

[16] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, vol. 49. NBS Washington, DC, 1952.

References (cont.)

[17] C. T. Kelley, *Iterative methods for linear and nonlinear equations*. SIAM, 1995.

[18] S. G. Nash, "A survey of truncated-Newton methods," *Journal of computational and applied mathematics*, vol. 124, no. 1-2, pp. 45–59, 2000.

[19] A. Krylov, "On the numerical solution of equation by which are determined in technical problems the frequencies of small vibrations of material systems," *News Acad. Sci. USSR*, vol. 7, pp. 491–539, 1931.

[20] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[21] E. Polak and G. Ribiere, "Note sur la convergence de méthodes de directions conjuguées," *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, vol. 3, no. R1, pp. 35–43, 1969.

[22] Y.-H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM Journal on optimization*, vol. 10, no. 1, pp. 177–182, 1999.

[23] J. A. Lee and M. Verleysen, *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[24] N. Andrei, "A diagonal quasi-Newton updating method for unconstrained optimization," *Numerical Algorithms*, vol. 81, no. 2, pp. 575–590, 2019.

References (cont.)

[25] R. Fletcher, *Practical methods of optimization*.
John Wiley & Sons, 1987.

[26] J. E. Dennis Jr and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*.
SIAM, 1996.

[27] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980.

[28] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[29] W. C. Davidon, "Variable metric method for minimization," *SIAM Journal on Optimization*, vol. 1, no. 1, pp. 1–17, 1991.

[30] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of computation*, vol. 19, no. 92, pp. 577–593, 1965.

[31] A. R. Conn, N. I. Gould, and P. L. Toint, "Convergence of quasi-newton matrices generated by the symmetric rank one update," *Mathematical programming*, vol. 50, no. 1, pp. 177–195, 1991.

[32] M. Avriel, *Nonlinear programming: analysis and methods*.
Courier Corporation, 2003.

References (cont.)

[33] R. Hosseini and S. Sra, "An alternative to EM for Gaussian mixture models: batch and stochastic Riemannian optimization," *Mathematical Programming*, vol. 181, no. 1, pp. 187–223, 2020.