# COMPARING COMPACT QUASI-NEWTON FORMULATIONS

BY

JINXIN XIA

A Thesis Submitted to the Graduate Faculty of

WAKE FOREST UNIVERSITY GRADUATE SCHOOL OF ARTS AND SCIENCES

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF ARTS

Mathematics and Statistics

May 2021

Winston-Salem, North Carolina

Approved By:

Jennifer Erway, Ph.D., Advisor

Kenneth S. Berenhaut, Ph.D., Chair

Robert Erhardt, Ph.D.

# Acknowledgments

There are many people who helped me to make this thesis possible. I would like to express my sincere thanks to my advisor Dr. Erway for her continuous support of my thesis during my study at Wake Forest University. She always encourages me in trying new ideas that I come up with and gives me enlightening suggestions. Without her support, I would never be able to write this thesis. I would also express my thanks to the rest of my thesis committee: Dr. Berenhaut and Dr. Erhardt for their kind help during the process of my graduation.

In the meantime, I am extremely grateful to my parents for their unconditional love, caring and sacrifices during my hardest moment in last spring semester. My special thanks go to Eva Wu for her kind care during this global pandemic situation.

# Table of Contents

# Abstract

Jinxin Xia

The computational power growth nowadays does not match the demands of the data size growth. Efficient algorithms are needed in almost every large numerical optimization task. Decades ago Broyden, Fletcher, Goldfarb, and Shanno discovered the Broyden–Fletcher–Goldfarb–Shanno algorithm based on Newton's algorithm. This algorithm latter was modified as limited-memory BFGS algorithm that has a prominent record in computational efficiency. Limited-memory BFGS can have different forms based on the compact formulas of the BFGS. In this thesis, two different compact BFGS formulas, David Ek and Anders Forsgren (2020) and Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994), are compared in terms of computational costs under different situations. The compact BFGS formula from David Ek and Anders Forsgren (2020) is converted into limited-memory BFGS. Numerical experiments are conducted based on this limited-memory BFGS. According to the experiment results and analysis, we find that this limited-memory BFGS does not guarantee positive definite property as other limited-memory BFGS normally do. To solve this problem more research in combining line search and trust region methods are needed.

# Chapter 1: Introduction

Optimization is such a common thing in our life. Let's take the lunch time during a rough day as an example. When you are busy and you have to get a quick lunch, then among all the food options you might want to choose a sandwich over a Thai Pho because it requires less time to eat and you want to minimize the time you spend on lunch. But then you realize that the closest sandwich shop is 25 miles away, and it's too time consuming to get there. So now you have to choose a nearby food option that requires short lunch time. Then after a careful selecting you decide to order pizza delivery even though you think it's not healthy.

Now you see that the problem about determining what to eat for lunch on a rough day is an optimization problem. The lunch time and how unhealthy the lunch is are the values that you want to minimize, and to get the time value you break it down to ordering time, waiting time, and eating time. Since health is also valuable to you, serious penalty will be put on fast food even though they are fast. Then you find all the options that are feasible to you for lunch so that you will not waste time on finding authentic Chinese food. After the this process you put options into your lunch calculating function to get the overall decision value. Among all those feasible options, you find that sandwich needs least lunch time and it is also very healthy to eat. However, the closest sandwich shop is 25 miles away. Then you have to give the input a constraint which is the food option should be within 5 miles away from you. At the end of this process, you order the pizza delivery even though a big penalty has already put on it, but it is fast enough to overcome the penalty.

Not all the optimization problems are as intuitive as the lunch problem. Therefore, we need more rigorous mathematical way to present optimization problems and its

constraints. We call the function that we want to minimize or maximize as the objective function. To make it convenient, we only talk about minimizing since we can easily convert maximizing problem to minimizing problem by introduce a negative sign in front of the objective function. The formal way to represent an optimization problem with objective function $f(x)$ and its constrains $c_i(x)$ is

$$\begin{aligned} &\underset{x}{\text{minimize}} && f(x) \\ &\text{subject to} && c_i(x) \leq b_i,\ i = 1, \ldots, m. \end{aligned}$$

Since optimization is such a useful tool and it is studied in different areas using different terms, we define the terms that will be used in this thesis below:

$f(x)$: objective function that will be optimized

$c(x)$: constraint equations

$x_k$: current point

$p_k$: difference between $x_{k+1}$ and $x_k$, also called the search direction

$x^*$: minimizer of the objective function

$x_0$: the initial point of the optimization process

$\nabla f(x_k)$: first derivative of a high-dimensional objective function

$\nabla^2 f(x_k)$: second derivative of a high-dimensional objective function

$B_k$: Hessian matrix approximation

$H_k$ : the inverse of the Hessian matrix approximation $B_k$

### 1.0.1 Newton's Method

Let's begin with the simplest function $f(x) = ax + b$. The minimizer or maximizer occur at the edges of the domain, given $a \neq 0$, since $f(x)$ is just a straight line. For a simple quadratic function, like $f(x) = ax^2 + bx + c$, we need more information from the function to determine the minimizer or maximizer.

Many methods and algorithms are developed to solve quadratic functions. The idea of Newton's method in this simple one-dimensional quadratic function situation is to fit a tangent line to the objective function at current point, and choose the point where the tangent line has a zero as the next iteration point. The tangent line at current point is

$$f(x_k) - f(a) = f'(x_k)(x_k - a). \tag{1.1}$$

The next point in this optimization process will be the point that makes the function value be 0. Therefore, the next point will have the following equation

$$f(x_k) - f(x_{k+1}) = f'(x_k)(x_k - x_{k+1}),$$

where $f(x_{k+1}) = 0$, and $x_{k+1}$ is

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{1.2}$$

Equation (1.2) works well for convex one-dimension optimization problems. However, for higher dimensions and more complicated optimization problems we will need to use more information from the objective function and change our minimizer searching strategy.

For higher-dimensional functions, instead of using the tangent line to search for next iteration point, we use a quadratic model to approximate the objective function's local

behavior. We prefer quadratic model among all other models because a quadratic model is the well-studied model form that has many nice properties for optimization purposes such as it is easy to find global minimum and easy to construct. The next step is to construct a quadratic model that fits our need. Here we use Taylor expansion to approximate that quadratic model, and the second order Taylor expansion for a lower dimensional objective function is

$$f(x) = f(x_k) + f'(x_k)(x_k - x) + \frac{f''(x_k)}{2!}(x_k - x)^2 + \varepsilon, \tag{1.3}$$

where $\varepsilon$ is the rest of the Taylor expansion. Based on equation (1.3), we develop a quadratic model $m_k(p)$ for a high dimensional objective function at current point

$$m_k(x) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T\nabla^2 f(x_k)(x - x_k). \tag{1.4}$$

Since we are searching for the next iteration point $x_{k+1}$, which is also a minimizer of equation (1.4), we will use $p$ the searching direction to represent $(x - x_k)$ and this leads to the following equation

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2}p^T\nabla^2 f(x_k)p \tag{1.5}$$

The minimum (only if $\nabla^2 f(x_k)$ is positive definite) of equation (1.5) is

$$p_k = -\nabla^2 f(x_k)^{-1}\nabla f(x_k). \tag{1.6}$$

However, we might need to modify the search direction $p_k$ by giving it some scalar to satisfy `Wolfe conditions`, to ensure the overall method will converge. So the next iteration point will be

$$x_{k+1} = x_k + \alpha_k p_k,$$

with `Wolfe conditions`

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \nabla f_k^T p_k,$$

$$c_2 \nabla f_k^T p_k \leq \nabla f(x_k + \alpha_k p_k)^T p_k,$$

where $0 < c_1 < c_2 < 1$.

If the Hessian matrix $\nabla^2 f(x_k)$ in equation (1.5) is positive definite, then $p_k$ here is a descent direction which means $p_k^T \nabla f(x_k) < 0$. This descent direction $p_k$ will guarantee next iteration point $x_{k+1} = x_k + p_k$ will reduce the value of the objective function since the objective function value change from $x_k$ to $x_{k+1}$ can be approximated as $p_k^T \nabla f(x_k) < 0$.

However, in real world applications, the Hessian matrix $\nabla^2 f(x_k)$ in equation (1.5) is not always positive definite and the iterates may not converge. Besides the positive definite constraint on the Hessian matrix $\nabla^2 f(x_k)$, computational cost is also a big problem when $x$ lives in a high-dimensional space. For example, if $x_0 \in \mathbb{R}^{100,000}$ then the Hessian matrix $\nabla^2 f(x_k) \in \mathbb{R}^{100,000 \times 100,000}$ which has 10 billion entries in the matrix and it is too large to compute and store. Also, more and more data extracted from biology, medicine, physics, and all other subjects, lead to larger and larger optimization problems that put a serious urgency in computationally efficient methods.

### 1.0.2 Quasi-Newton Method

Now we see there are two problems with Newton's method: The Hessian matrix $\nabla^2 f(x_k)$ might not always be positive definite and it is not always possible or efficient to compute the Hessian matrix given the technology that we have. Therefore to solve these problems, instead of computing the Hessian matrix directly, quasi-Newton methods replace the Hessian matrix using a well-defined approximate matrix which is positive definite and less computationally expensive.

Quasi-Newton methods are powerful tools in optimization field that are based on Newton's method. We now follow the derivation in Numerical Optimization (Jorge Nocedal and Stephen J. Wright) to derive the secant equation. The idea of the method is to approximate Hessian matrices at each iteration point rather than directly compute them. Recall from the Newton method at current point we have a quadratic function(model) and its minimizer

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p.$$

$$p_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k).$$

To make the equation easy to read, we use $f_k$ to represent $f(x_k)$ and the same rule for gradient or Hessian matrix. Since we are not computing the Hessian matrix, we will use $B_k$ to approximate $\nabla^2 f(x_k)$. Therefore, we can rewrite above equations as

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k f_k p, \tag{1.7}$$

$$p_k = -B_k^{-1} \nabla f_k. \tag{1.8}$$

Based on this quadratic model, the next quadratic function will be

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p. \tag{1.9}$$

After taking derivative of equation (1.9) with respect to $p$, we have the following equation

$$\nabla m_{k+1}(p) = \nabla f_{k+1} + B_k p. \tag{1.10}$$

According to the Numerical Optimization book written by Jorge Nocedal and Stephen J. Wright, equation (1.10) should satisfy that its gradient match with the objective function's gradient at point $x_k$ and point $x_{k+1}$.

When $p = 0$ in equation (1.10), $\nabla m_{k+1}(p) = \nabla f_{k+1}$ and the current point is $x_{k+1}$. Since $x_{k+1} = x_k + \alpha_k p_k$, we know that when the input is $-\alpha_k p_k$ in equation (1.10) the current point is $x_k$. Based on the requirement in the Numerical Optimization book we have

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_k p_k = \nabla f_k, \tag{1.11}$$

This equation uses interpolation technique which is setting the first derivative of $m_{k+1}$ equals to the first derivative of $f$ at point $x_k$. Since $m_{k+1}(0) = f_{k+1}$, we can treat $-\alpha_k p_k$ as $x_k$ in $m_{k+1}$. Therefore, we have the above equation.

Rewriting the above equations, we get

$$\alpha_k B_k p_k = \nabla f_{k+1} - \nabla f_k. \tag{1.12}$$

Let $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$, then we can get

$$(\textit{secant equation}) \quad B_{k+1} s_k = y_k. \tag{1.13}$$

We call this simple expression the secant equation since this equation is generated from information that related to points $x_k$ and $x_{k+1}$.

So far, all the notation used is standard notation for quasi-Newton method, the next step is to introduce the Broyden class and Broyden, Fletcher, Goldfard, and Shanno (BFGS) algorithm update.

### 1.0.3 The Broyden class and BFGS

In 1965 Charles George Broyden published the famous paper "A Class of Methods for Solving Nonlinear Simultaneous Equations" that updates an approximate Hessian inside a Newton method. The original paper uses Jacobian matrix instead of Hessian matrix to do the approximation update. By using the same idea to the Hessian matrix approximation, they introduce the Broyden class as

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} + \phi_k (s_k^T B_k s_k) v_k v_k^T,$$

where $\phi_k$ is a scalar and $v_k = [\frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}]$. By setting

$$\omega_k = (s_k^T B_k s_k)^{\frac{1}{2}} (\frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}),$$

we can also write the Broyden class as

$$\text{(Broyden Class) } B_{k+1} = B_k - \frac{B_k s_k (B_k s_k)^T}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} + \phi_k \omega_k \omega_k^T. \tag{1.14}$$

Based on this Broyden class method Broyden, Fletcher, Goldfard, and Shanno further studied the details of quasi-Newton low rank update method. Around 1970, they all published their works in a quasi-Newton optimization method that used similar

techniques. Those methods were then combined as a new famous method named as Broyden class and Broyden, Fletcher, Goldfard, and Shanno (BFGS) method, and the inverse Hessian approximation version is

$$(BFGS) \quad H_{k+1} = (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}, \tag{1.15}$$

where $H_{k+1}$ is the inverse of the objective function's Hessian approximation at point $x_{k+1}$.

We can also get the Hessian approximation $B_k$ update from this solution by applying the Sherman Morrison Woodbury formula:

$$(BFGS) \quad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{1.16}$$

BFGS requires $y_k^T s_k > 0$ to make sure $B_k$ is positive definite. This is because if we have $y_k^T s_k > 0$, then based on the secant equation we have

$$s_k^T B_{k+1} s_k = s_k^T y_k = y_k^T s_k > 0,$$

and since $s_k$ is not fixed then $B_{k+1}$ is positive definite.

Notice if we set $\phi_k = 0$ in equation (1.13) then we get the BFGS update. For all members of the Broyden class, we do not need to compute the Hessian matrix and store it as long as we have the stored pairs that are needed.

### 1.0.4 Positive definite property of BFGS

Recall BFGS in section one has two formulas, the Hessian approximation inverse version

$$(BFGS) \quad H_{k+1} = (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k},$$

and the Hessian approximation version

$$(BFGS) \quad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}.$$

Given $x \in \mathbb{R}^n$ and $x \neq 0$, we have

$$x^T H_{k+1} x = x^T (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) x + \frac{x^T s_k s_k^T x}{y_k^T s_k}. \tag{1.17}$$

We know that $(I - \frac{s_k y_k^T}{y_k^T s_k}) \neq 0$, $H_k$ is positive definite, and $y_k^T s_k > 0$. Therefore,

$$x^T H_{k+1} x = x^T (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) x + \frac{x^T s_k s_k^T x}{y_k^T s_k} > 0, \tag{1.18}$$

which means $H_{k+1}$ is positive definite as long as $H_k$ is positive definite and $y_k^T s_k > 0$. Since the matrix inverse of a positive definite matrix is also positive definite, we know that $B_{k+1}$ is positive definite as long as $B_k$ is positive definite and $y_k^T s_k > 0$.

### 1.0.5 Compact representations of BFGS

With more information and data coming from biology, chemistry and physics, high-dimensional data significantly slows the optimization speed of algorithm, given the limited read and write speed on computer CPUs. However, we can use linear algebra to reshape the BFGS into a compact format which only requires small matrices and vectors in each iteration update. The computing result is the same, but the total computational cost differs from the regular BFGS. With proper linear algebra rearrangement, the computational cost will be significant reduced as well as the storage requirement and data movement between storage and computer CPU.

We begin our research on a new compact formula for Broyden class from DeGuchy O, Erway JB, Marcia RF (2018), and then narrow down to compact BFGS within that Broyden class which is from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994). Another compact formulated BFGS studied in this thesis is from David Ek and Anders Forsgren (2020 arXiv). Besides these two compact formulated BFGS, another compact formulated BFGS from David Ek and Anders Forsgren (2020 arXiv) is also provided in Appendix A as reader's interest. Both compact formulations in this thesis use the following notations, and we will begin with the Broyden class compact formula from DeGuchy O, Erway JB, Marcia RF (2018). Let

$$S_k = \begin{pmatrix} s_0 & s_1 & s_2 & \dots & s_k \end{pmatrix}, \tag{1.19}$$

$$Y_k = \begin{pmatrix} y_0 & y_1 & y_2 & \dots & y_k \end{pmatrix}, \text{ and} \tag{1.20}$$

$$S_k^T Y_k = L_k + D_k + R_k, \tag{1.21}$$

where $L_k$, $D_k$, and $R_k$ separate $S_k^T Y_k$ into three parts: strictly lower triangular, diagonal, and strictly upper triangular respectively. In DeGuchy O, Erway JB, Marcia RF (2018), the permutation matrix $\Pi_k \in \mathbb{R}^{2(k+1)\times 2(k+1)}$ with $\Pi_0 = I_2$ is

$$\hat{\Pi}_k = \begin{pmatrix} I_k & 0 & 0 & 0 \\ 0 & 0 & I_k & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.22}$$

They also proposed $\Xi_k$, with $\Xi_0 = E_0$, which is recursively defined as

$$\Xi_k = \begin{pmatrix} \Pi_{k-1}^T \Xi_{k-1} & 0 \\ 0 & E_k \end{pmatrix}, \tag{1.23}$$

where

$$E_k = \begin{cases} (-1 \quad 1)^T & \text{if } \phi_k = \phi_k^{SR1} \\ I_2 & \text{otherwise.} \end{cases}$$

Here $\phi_k^{SR1}$ is defined as $\phi_k^{SR1} = \frac{s_k^T y_k}{s_k^T y_k - s_k^T B_k s_k}$ and $\Gamma_k \in \mathbb{R}^{(k+1)\times(k+1)}$ is a diagonal matrix as

$$\Gamma_k = \underset{0\le j\le k}{\text{diag}}(\gamma_j), \tag{1.24}$$

where

$$\gamma_j = \begin{cases} \phi_j(-\frac{1-\phi_j}{s_j^T B_j s_j} - \frac{\phi_j}{s_j^T y_j}) & \text{if } \phi_j = \phi_j^{SR1} \\ 0 & \text{otherwise.} \end{cases}$$

With all these notation set up, they found a compact formulation for the Broyden class update as

$$B_{k+1} = B_0 + \hat{\Psi}_k \hat{M}_k \hat{\Psi}_k^T, \tag{1.25}$$

where

$$\hat{M}_k = \begin{pmatrix} \Xi_k^T \Pi_k & \begin{pmatrix} -S_k^T B_0 S_k + \Gamma_k & -L_k + \Gamma_k \\ -L_k^T + \Gamma_k & D_k + \Gamma_k \end{pmatrix} & \Pi_k^T \Xi_k \end{pmatrix}^{-1}, \tag{1.26}$$

$$\hat{\Psi}_k = \Psi_k \Pi_k^T \Xi_k, \tag{1.27}$$

and

$$\Psi_k = Y_k - B_0 S_k. \tag{1.28}$$

The proof of this compact formula of Broyden class is clearly presented in the paper, so we will not prove this again. We will set $\Gamma_k = 0$ in latter section to get the BFGS from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994).

For the second compact formula of Broyden class in David Ek and Anders Forsgren (2020) arXiv paper, they proposed a new compact formula for Broyden class for use with quadratic problems. Together with $B_k p_k = -g_k$ for all $k$, they can find this simple compact formula in the following theorem:

**Theorem 1.1.**

$$B_k = B_0 + G_k T_k G_k^T, \tag{1.29}$$

*where*

$$G_k = [g_0 \quad g_1 \quad g_2 \quad ... \quad g_{k-1} \quad g_k],$$

*and* $T_k \in \mathcal{R}^{(K+1)\times(K+1)}$ *is a symmetric tridiagonal matrix on the form* $T_k = T_k^C + T_k^\phi$ *where*

$$T_k^C(1,1) = \frac{1}{g_0^T p_0} + \frac{1}{\alpha_0 (g_1 - g_0)^T p_0}$$

$$T_k^C(i,i) = \frac{1}{g_i^T p_i} + \frac{1}{\alpha_{i-1}(g_i - g_{i-1})^T p_{i-1}} + \frac{1}{\alpha_i (g_{i+1} - g_i)^T p_i} \qquad i = 1, \quad ..., \quad k-1$$

$$T_k^C(i+1,i) = T_k^C(i,i+1) = -\frac{1}{\alpha_{i-1}(g_i - g_{i-1})^T p_{i-1}} \qquad i = 1, \quad ..., \quad k$$

$$T_k^C(k+1,k+1) = -\frac{1}{\alpha_{k-1}(g_k - g_{k-1})^T p_{k-1}}$$

$$T_k^\phi(1,1) = -\phi_0 g_0^T p_0 (\frac{1}{(g_1 - g_0)^T p_0} + \frac{1}{g_0^T p_0})^2$$

$$T_k^\phi(i,i) = -\phi_{i-1} g_{i-1}^T p_{i-1} (\frac{1}{(g_i - g_{i-1})^T p_{i-1}})^2 - \phi_i g_i^T p_i (\frac{1}{(g_{i+1} - g_i)^T p_i} + \frac{1}{g_i^T p_i})^2$$

$i = 1, \quad ..., \quad k-1$

$$T_k^\phi(i+1,i) = T_k^\phi(i,i+1) = \phi_{i-1} g_{i-1}^T p_{i-1} (\frac{1}{(g_i - g_{i-1})^T p_{i-1}})^2 + \phi_{i-1} \frac{1}{(g_i - g_{i-1})^T p_{i-1}}$$

$i = 1, \quad ..., \quad k$

$$T_k^\phi(k+1,k+1) = -\phi_{k-1} g_{k-1}^T p_{k-1} (\frac{1}{(g_k - g_{k-1})^T p_{k-1}})^2.$$

In the original paper, David Ek and Anders Forsgren did not give a detailed proof of this compact formula. To better understand this compact formula's strengths and weaknesses, the proof is listed below. For notational simplicity, we prove this for the BFGS case only, but the full Broyden class case can be proved use the similar proof below.

*Proof.* Recall equation (1.21) the Broyden class

$$B_k = B_{k-1} - \frac{B_{k-1}s_{k-1}(B_{k-1}s_{k-1})^T}{s_{k-1}^T B_{k-1}s_{k-1}} + \frac{y_{k-1}y_{k-1}^T}{y_{k-1}^T s_{k-1}} + \phi_{k-1}\omega_{k-1}\omega_{k-1}^T,$$

where

$$\omega_{k-1} = (s_{k-1}^T B_{k-1}s_{k-1})^{\frac{1}{2}}(\frac{y_{k-1}}{y_{k-1}^T s_{k-1}} - \frac{B_{k-1}s_{k-1}}{s_{k-1}^T B_{k-1}s_{k-1}}).$$

Substitute $B_i p_i = -g_i$ to the Broyden class equation, we get

$$B_k = B_{k-1} + [\frac{1}{g_{k-1}^T p_{k-1}} g_{k-1}g_{k-1}^T + \frac{1}{\alpha_i(g_k - g_{k-1})^T}(g_k - g_{k-1})(g_k - g_{k-1})^T + \phi_{k-1}\omega_{k-1}\omega_{k-1}^T]. \tag{1.30}$$

Then substitute $B_i p_i = -g_i$ again to $B_{k-1}$ in the above equation we can get

$$B_k = B_{k-2} + \sum_{i=k-2}^{k-1} [\frac{1}{g_i^T p_i} g_i g_i^T + \frac{1}{\alpha_i(g_{i+1} - gi)^T}(g_{i+1} - gi)(g_{i+1} - gi)^T + \phi_i \omega_i \omega_i^T]. \tag{1.31}$$

By repeating this process, we can finally get the following equation

$$B_k = B_0 + \sum_{i=0}^{k-1} [\frac{1}{g_i^T p_i} g_i g_i^T + \frac{1}{\alpha_i(g_{i+1} - gi)^T}(g_{i+1} - gi)(g_{i+1} - gi)^T + \phi_i \omega_i \omega_i^T], \tag{1.32}$$

where

$$\omega_i = (-g_i^T p_i)^{\frac{1}{2}}(\frac{1}{(g_{i+1} - gi)^T p_i}(g_{i+1} - gi) - \frac{1}{g_i^T p_i} g_i).$$

We use induction to prove equation (1.28) can be written as equation (1.27). Let k = 1, then

$$B_1 = B_0 - \frac{B_0 s_0 (B_0 s_0)^T}{s_0^T B_0 s_0} + \frac{y_0 y_0^T}{y_0^T s_0} + \phi_0 \omega_0 \omega_0^T,$$

where

$$\omega_0 = (s_0^T B_0 s_0)^{\frac{1}{2}} (\frac{y_0}{y_0^T s_0} - \frac{B_0 s_0}{s_0^T B_0 s_0}).$$

Substitute $B_i s_i = \alpha_i p_i$ and $s_i = x_{i+1} - x_i = \alpha_i p_i$, where $i = 0, 1, \ldots, k-1$ into the above equation to obtain

$$\omega_0 = (-g_0^T p_0)^{\frac{1}{2}} (\frac{1}{(g_1 - g_0)^T p_0}(g_1 - g_0) - \frac{1}{g_0^T p_0} g_0).$$

Therefore, we have

$$B_1 = B_0 - \frac{B_0 s_0 (B_0 s_0)^T}{s_0^T B_0 s_0} + \frac{y_0 y_0^T}{y_0^T s_0} + \phi_0 \omega_0 \omega_0^T \tag{1.33}$$

$$= B_0 - \frac{-\alpha g_0 (-\alpha g_0^T)}{-\alpha g_0^T \alpha p_0} + \frac{y_0 y_0^T}{y_0^T s_0} + \phi_0 \omega_0 \omega_0^T \tag{1.34}$$

$$= B_0 + \frac{g_0 g_0^T}{g_0^T p_0} + \frac{y_0 y_0^T}{y_0^T s_0} + \phi_0 \omega_0 \omega_0^T, \tag{1.35}$$

$$\tag{1.36}$$

where

$$\phi_0 \omega_0 \omega_0^T = \phi_0 (\frac{g_0 p_0^T (-g_1 g_1^T + g_1 g_0^T + g_0 g_1^T - g_0 g_0^T)}{((g_1 - g_0)^T p_0)^2} + \frac{2 g_1 g_0^T + 2 g_0 g_1^T}{(g_1 - g_0^T) p_0 g_0^T p_0} + \frac{g_0 g_0^T}{g_0^T p_0}).$$

The compact formula is $B_1 = B_0 + G_1 T_1 G_1^T$ with $G_1 = \begin{pmatrix} g_0 & g_1 \end{pmatrix}$, and the $T_k$ matrix is decomposed as

$$T_1^C = \begin{pmatrix} \frac{1}{g_0^T p_0} + \frac{1}{\alpha_0 (g_1 - g_0)^T p_0} & \frac{1}{\alpha_0 (g_1 - g_0)^T p_0} \\ \frac{1}{\alpha_0 (g_1 - g_0)^T p_0} & -\frac{1}{\alpha_0 (g_1 - g_0)^T p_0} \end{pmatrix}, \tag{1.37}$$

and

$$T_1^{\phi} = \begin{pmatrix} -g_0^T p_0 (\frac{1}{(g_1-g_0)^T p_0} + \frac{1}{g_0^T p_0})^2 & \phi_0 \frac{g_0^T p_0}{[(g_1-g_0)^T p_0]^2} + \phi_0 \frac{g_0^T p_0}{(g_1-g_0)^T p_0} \\ \phi_0 \frac{g_0^T p_0}{[(g_1-g_0)^T p_0]^2} + \phi_0 \frac{g_0^T p_0}{(g_1-g_0)^T p_0} & -\phi_1 g_1^T p_1 (\frac{1}{(g_2-g_1)^T p_1})^2 \end{pmatrix}. \tag{1.38}$$

After the linear algebra calculation we get

$$\begin{aligned} B_1 &= B_0 + G_1 T_1 G_1^T \\ &= B_0 + G_1 T_1^C G_1^T + G_1 T_1^{\phi} G_1^T \\ &= B_0 + \begin{pmatrix} g_0 & g_1 \end{pmatrix} \begin{pmatrix} \frac{1}{g_0^T p_0} + \frac{1}{\alpha_0 (g_1-g_0)^T p_0} & \frac{1}{\alpha_0 (g_1-g_0)^T p_0} \\ \frac{1}{\alpha_0 (g_1-g_0)^T p_0} & -\frac{1}{\alpha_0 (g_1-g_0)^T p_0} \end{pmatrix} \begin{pmatrix} g_0 & g_1 \end{pmatrix}^T \\ &+ \begin{pmatrix} g_0 & g_1 \end{pmatrix} \begin{pmatrix} -g_0^T p_0 (\frac{1}{(g_1-g_0)^T p_0} + \frac{1}{g_0^T p_0})^2 & \phi_0 \frac{g_0^T p_0}{[(g_1-g_0)^T p_0]^2} + \phi_0 \frac{g_0^T p_0}{(g_1-g_0)^T p_0} \\ \phi_0 \frac{g_0^T p_0}{[(g_1-g_0)^T p_0]^2} + \phi_0 \frac{g_0^T p_0}{(g_1-g_0)^T p_0} & -\phi_1 g_1^T p_1 (\frac{1}{(g_2-g_1)^T p_1})^2 \end{pmatrix} \begin{pmatrix} g_0 & g_1 \end{pmatrix}^T \\ &= B_0 + \frac{g_0 g_0^T}{g_0^T p_0} + \frac{g_0 g_0^T}{\alpha_0 (g_1-g_0)^T p_0} + \frac{g_0 g_1^T + g_1 g_0^T}{\alpha_0 (g_1-g_0)^T p_0} - \frac{g_1 g_1^T}{\alpha_0 (g_1-g_0)^T p_0} + \phi_0 \omega_0 \omega_0^T \\ &= B_0 + \frac{g_0 g_0^T}{g_0^T p_0} + \frac{y_0 y_0^T}{y_0^T s_0} + \phi_0 \omega_0 \omega_0^T. \end{aligned}$$

Therefore, we proved when k=1 the statement in the theorem is correct. Since we only focus on the BFGS part, therefore we will drop the term leading with $\phi$ to make the Broyden class become BFGS compact formula from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994)..

Now assume when k = n the statement is also true. That is $B_n = B_0 + G_n T_n^C G_n^T$. Let k = n+1, we have

$$B_{n+1} = B_n + \frac{g_n g_n^T}{g_n^T p_i} + \frac{(g_{n+1}-g_n)(g_{n+1}-g_n)^T}{\alpha_n (g_{n+1}-g_n)^T p_n}, \tag{1.39}$$

Also, notice that $G_{n+1} = \begin{pmatrix} g_0 & g_1 & \dots & g_{n+1} \end{pmatrix} = \begin{pmatrix} G_n & g_{n+1} \end{pmatrix}$ and

$$T_{n+1}^C = \begin{pmatrix} T_n^C & -\frac{1}{\alpha_n(g_n - g_{n-1}^T)p_{n-1}} \\ -\frac{1}{\alpha_n(g_n - g_{n-1}^T)p_{n-1}} & \frac{1}{\alpha_n(g_n - g_{n-1}^T)p_{n-1}} \end{pmatrix}.$$

Therefore, we know that

$$\begin{aligned} G_{n+1}T_{n+1}^C G_{n+1}^T &= \begin{pmatrix} G_n & g_{n+1} \end{pmatrix} T_{n+1}^C \begin{pmatrix} G_n & g_{n+1} \end{pmatrix}^T \\ &= G_n T_n^C G_n^T + \frac{g_n g_n^T}{g_n^T p_i} + \frac{(g_{n+1} - g_n)(g_{n+1} - g_n)^T}{\alpha_n (g_{n+1} - g_n)^T p_n}. \end{aligned}$$

Since $B_n = B_0 + G_n^C T_n^C G_n^T$ we know that

$$B_0 + G_{n+1}T_{n+1}^C G_{n+1}^T = B_0 + G_n T_n^C G_n^T + \frac{g_n g_n^T}{g_n^T p_i} + \frac{(g_{n+1} - g_n)(g_{n+1} - g_n)^T}{\alpha_n (g_{n+1} - g_n)^T p_n},$$

which can also be written as

$$B_0 + G_{n+1}T_{n+1}^C G_{n+1}^T = B_n + \frac{g_n g_n^T}{g_n^T p_i} + \frac{(g_{n+1} - g_n)(g_{n+1} - g_n)^T}{\alpha_n (g_{n+1} - g_n)^T p_n}. \tag{1.40}$$

$$= B_{n+1} \tag{1.41}$$

Therefore, by induction we prove that $B_k = B_0 + G_k T_k^C G_k^T$ is true for $k \geq 1$

□

# Chapter 2: Compact updates of BFGS Comparsion

Now we have two compact updates of BFGS. Both of them are well designed and can save storage cost and computational cost comparing with regular BFGS. To further test which method is better in saving computational cost and storage cost, we will ask the following questions: What are the storage costs and flops needed of each method? Which is the fastest factorization to use to solve linear systems of the form $Ax = b$, where A is a quasi-Newton matrix? Which is the fastest factorization to use when multiplying the matrix times a vector? Which factorization is easier to update when the quasi-Newton matrix is update? We begin our analysis by answering the first question.

### 2.0.1 Storage cost and flops analysis

From the two compact BFGS update methods, we find the dimensions of the core matrix in the two methods are different. The first method has a core matrix $\hat{M}$ with dimension $\mathbb{R}^{2(k+1)\times 2(k+1)}$, and the core matrix's dimension of the second method is $\mathbb{R}^{(k+1)\times(k+1)}$. According to these information, we might guess that the compact update from David Ek and Anders Forsgren (2020) might be a less computational expensive than the compact update from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) since they share the same structure $BAB^T$ where $A$ is the core matrix and $BAB^T \in \mathbb{R}^{n\times n}$ . We will use the form from DeGuchy O, Erway JB, Marcia RF (2018) to represent the BFGS originally from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994). We will use the big $\mathcal{O}$ notation here to check the order of each method in updating and storage cost. Big $\mathcal{O}$ is a term that describe the leading or dominant

term in a series. For example, if the computational cost for an algorithm is $8n^2+100n$ then the big $\mathcal{O}$ term for this computational cost is $\mathcal{O}(n^2)$ since the $n^2$ term here will dominate the computational cost as $n$ increasing. If the big $\mathcal{O}$ notation cannot give us a clear answer, then we can go to details of the flops count.

Recall the update method from DeGuchy O, Erway JB, Marcia RF (2018) is

$$B_{k+1} = B_0 + \hat{\Psi}_k \hat{M}_k \hat{\Psi}_k^T,$$

with

$$\hat{M}_k = \begin{pmatrix} \Xi_k^T \Pi_k & \begin{pmatrix} -S_k^T B_0 S_k + \Gamma_k & -L_k + \Gamma_k \\ -L_k^T + \Gamma_k & D_k + \Gamma_k \end{pmatrix} & \Pi_k^T \Xi_k \end{pmatrix}^{-1}.$$

Since $\Xi_k$ and $\Pi_k$ are permutation matrix ($\Xi_k$ might not be permutation matrix but its complexity is similar to a permutation matrix). They do not require flops. Therefore, the computational cost to obtain $\hat{M}_k$ is the number of flops need to compute

$$\begin{pmatrix} -S_k^T B_0 S_k + \Gamma_k & -L_k + \Gamma_k \\ -L_k^T + \Gamma_k & D_k + \Gamma_k \end{pmatrix}^{-1}.$$

Since we are looking at BFGS, $\Gamma_k = 0$.

Notice the core matrix in the second part of this simplified compact update is a dense matrix with a dimension of $\mathbb{R}^{2(k+1)\times 2(k+1)}$. Therefore, the computational cost of each update under this simplified version is

$$8n(k+1)(k+3/4) + n^2(4k+3) + 2(2n-1)(k+1)^2 + \frac{40}{3}(k+1)^3. \tag{2.1}$$

Specifically, the $\hat{\Psi}_k \hat{M}_k{}^{-1}$ part the computational cost is $8n(k+1)(k+3/4)$, and for the $(\hat{\Psi}_k \hat{M}_k{}^{-1})\hat{\Psi}_k^T$ part the computational cost is $n^2(4k+3)$. The $2(2n-1)(k+1)^2$

term is from the computational cost of $S_k^T Y_k$ and $-S_k^T B_0 S_k$ (assuming $B_0 = I$) to get $\hat{M}_k$, and $\frac{40(k+1)^3}{3}$ is the computational cost of the inverse calculation. We can further simplify equation (2.3) as

$$4n^2k + 3n^2 + 12nk^2 + 22nk + 10n - 2(k+1)^2 + \frac{40}{3}(k+1)^3.$$

In the second BFGS compact update from David Ek and Anders Forsgren (2020), we have the update as

$$B_k = B_0 + G_k T_k^C G_k^T.$$

In this compact update, $G_k$ is a dense matrix with dimension $\mathbb{R}^{n\times(k+1)}$, and $T_k$ is a tridiagonal matrix with dimension $\mathbb{R}^{(k+1)\times(k+1)}$. In particular, using the fact that $T_k$ is diagonal, the computational cost for the $G_k T_k$ part is $5n(k+\frac{4}{5})$ and the computational cost for the $(G_k T_k)G_k^T$ part is $n^2(2k+1)$ since $(G_k T_k)$ is a dense matrix. Therefore, considering the tridiagonal structure of the core matrix $T_k$ in this compact formula we know the computational cost without considering the cost of getting $T_k$ is

$$5n(k+\frac{4}{5}) + n^2(2k+1). \tag{2.2}$$

Besides the computational cost in the compact formula, we also need to count the cost of updating elements in the core matrix $T_k$. Since we only focus on the BFGS update, so we only count the $T_k^C$ part. Recall $T_k^C$ has the following definition

$T_k^C(1,1) = \frac{1}{g_0^T p_0} + \frac{1}{\alpha_0 (g_1 - g_0)^T p_0}$

$T_k^C(i,i) = \frac{1}{g_i^T p_i} + \frac{1}{\alpha_{i-1}(g_i - g_{i-1})^T p_{i-1}} + \frac{1}{\alpha_i (g_{i+1} - g_i)^T p_i} \quad i = 1, \ \ldots, \ k-1$

$$T_k^C(i+1,i) = T_k^C(i,i+1) = -\frac{1}{\alpha_{i-1}(g_i - g_{i-1})^T p_{i-1}} \quad i = 1, \quad ..., \quad k$$

$$T_k^C(k+1,k+1) = -\frac{1}{\alpha_{k-1}(g_k - g_{k-1})^T p_{k-1}}.$$

All the computational of the elements of matrix $T_k^C$ is vector times vector. Without too much work, we can get the flops of computing matrix $T_k^C$ is

$$10n - 2 + (k-1)(9n-1) + (k-2)(4n-1). \tag{2.3}$$

Thus, this compact update method has a total computational cost at each iteration of $n(k+1)(2k+1) + n^2(2k+1) + 5n(k+\frac{4}{5}) + 10n - 2 + (k-1)(9n-1) + (k-2)(4n-1)$, which simplifies to

$$2n^2k + n^2 + 2nk^2 + 21nk - 2n - 2k + 1.$$

Comparing with the computational cost at each iteration from the first compact update, which is

$$4n^2k + 3n^2 + 12nk^2 + 22nk + 10n - 2(k+1)^2 + \frac{40}{3}(k+1)^3,$$

we can see that the compact update method from David Ek and Anders Forsgren (2020) is relatively less computationally expensive at each iteration.

The storage cost of two methods has the same storage cost in the big $\mathcal{O}$ notation. Both methods share the similar structure as $B_k = B_0 + BAB^T$ which means the storage costs will go to $B$ and $A$. In the first method, $A \in \mathbb{R}^{n \times 2(k+1)}$ and $B \in \mathbb{R}^{2(k+1) \times 2(k+1)}$, which means the storage cost of this method is

$$\mathcal{O}((k+1)^2) + \mathcal{O}(n(k+1)). \tag{2.4}$$

In the second method, $A \in \mathbb{R}^{n\times(k+1)}$ and $B \in \mathbb{R}^{(k+1)\times(k+1)}$, which means the storage cost of this method is

$$\mathcal{O}((k+1)^2) + \mathcal{O}(n(k+1)). \tag{2.5}$$

However, the exact storage costs of this method are smaller since T is tridiagonal and the size of A and B are twice as small. Even though the storage cost for the second method is smaller than the first one, but the big O notation is the same.

In this section we only discuss the regular BFGS update situation for cost analysis. In the next chapter we will discuss the cost analysis for this two method based on the limited-memory BFGS situation.

### 2.0.2 Computational speed in solving linear system

Here the linear system is

$$B_k x = b, \tag{2.6}$$

where $B_k$ is the Hessian approximation has dimension of $\mathbb{R}^{n\times n}$ and $x$ and $b$ are vectors that have the same dimension $\mathbb{R}^{n\times 1}$. We solve this linear system by directly using the Sherman-Morrison-Woodbury formula to get the inverse of $B_k$ and therefore to get the following solution for $x$:

$$x = B_k^{-1} b. \tag{2.7}$$

Applying the Sherman-Morrison-Woodbury formula to both of the compact update

methods and fixing the different indexing issue, we get the compact update from DeGuchy O, Erway JB, Marcia RF (2018) as

$$B_{k+1}^{-1} = B_0^{-1} + B_0^{-1}\hat{\Psi}_k(-\hat{M}_k^{-1} - \hat{\Psi}_k^T B_0^{-1}\hat{\Psi}_k)\hat{\Psi}_k^T B_0^{-1}, \tag{2.8}$$

and from David Ek and Anders Forsgren (2020) as

$$B_{k+1}^{-1} = B_0^{-1} + B_0^{-1}G_k(-T_k^{-1} - G_k^T B_0^{-1}G_k)G_k^T B_0^{-1}. \tag{2.9}$$

Then multiply $b$ to both sides of the equations we get the compact update from as

$$B_{k+1}^{-1}b = B_0^{-1}b + B_0^{-1}\hat{\Psi}_k(-\hat{M}_k^{-1} - \hat{\Psi}_k^T B_0^{-1}\hat{\Psi}_k)\hat{\Psi}_k^T B_0^{-1}b, \tag{2.10}$$

and from David Ek and Anders Forsgren(2020) as

$$B_{k+1}^{-1}b = B_0^{-1}b + B_0^{-1}G_k(-T_k^{-1} - G_k^T B_0^{-1}G_k)G_k^T B_0^{-1}b. \tag{2.11}$$

$B_0$ can be chosen as we want, and it is not unreasonably to use an identity matrix. Thus we can drop the $B_0^{-1}$ part since any vectors times an identity matrix lead to the same vector. Therefore the big difference are the terms,

$$\hat{\Psi}_k(-\hat{M}_k^{-1} - \hat{\Psi}_k^T\hat{\Psi}_k)\hat{\Psi}_k^T b, \tag{2.12}$$

from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) and DeGuchy O, Erway JB, Marcia RF (2018) and the terms,

$$G_k(-T_k^{-1} - G_k^T G_k)G_k^T b, \tag{2.13}$$

from David Ek and Anders Forsgren (2020).

In equation (2.12), the computational cost of $\hat{\Psi}_k^T\hat{\Psi}_k$ is

$$4(k+1)^2(2n-1), \tag{2.14}$$

where $\hat{\Psi}_k \in \mathbb{R}^{n\times 2(k+1)}$. Assuming $\hat{M}_k^{-1}$ has already been stored, the computational cost of $(-\hat{M}_k^{-1} - \hat{\Psi}_k^T\hat{\Psi}_k)$ is

$$4(k+1)^2(2n-1) + 4(k+1)^2, \tag{2.15}$$

the computational cost of $\hat{\Psi}_k(-\hat{M}_k^{-1} - \hat{\Psi}_k^T\hat{\Psi}_k)$ is

$$4(k+1)^2(2n-1) + 4(k+1)^2 + 2n(k+1)(4k+3), \tag{2.16}$$

and the computational cost of $\hat{\Psi}_k^T b$ is

$$2(k+1)(2n-1). \tag{2.17}$$

Therefore, the computational cost of $\hat{\Psi}_k(-\hat{M}_k^{-1} - \hat{\Psi}_k^T\hat{\Psi}_k)\hat{\Psi}_k^T b$ is $4(k+1)^2(2n-1) + 4(k+1)^2 + 2n(k+1)(4k+3) + 2(k+1)(2n-1) + n(4k+3)$, which simplifies to

$$8n(k+1)^2 + 2n(k+1)(4k+3) + 8n(k+\frac{7}{8}) - 2(k+1),$$

and we can further simplify it as

$$16nk^2 + 38nk + 21n - 2k - 2. \tag{2.18}$$

In the update method from David Ek and Anders Forsgren (2020), we have

$$G_k(-(T_k^C)^{-1} - G_k^T G_k)G_k^T b.$$

The computational cost of $G_k^T G_k$ is

$$(k+1)^2(2n-1), \tag{2.19}$$

the computational cost of $(-(T_k^C)^{-1} - G_k^T G_k)$, assuming $(T_k^C)^{-1}$ has already been stored, is

$$(k+1)^2(2n-1) + (k+1)^2, \tag{2.20}$$

the computational cost of $G_k(-(T_k^C)^{-1} - G_k^T G_k)$ is

$$(k+1)^2(2n-1) + (k+1)^2 + n(k+1)(2k+1), \tag{2.21}$$

and the computational cost of $G_k^T b$ is

$$(k+1)(2n-1). \tag{2.22}$$

Thus, the computational cost of $G_k(-(T_k^C)^{-1} - G_k^T G_k)G_k^T b$ is $(k+1)^2(2n-1) + (k+1)^2 + n(k+1)(2k+1) + (k+1)(2n-1) + n(2k+1)$, which simplifies to

$$2n(k+1)^2 + n(k+1)(2k+1) + 4n(k+\frac{3}{4}) - (k+1),$$

and we can further simplify it as

$$4nk^2 + 11nk + 6n - k - 1. \tag{2.23}$$

Comparing with the computational cost in solving a linear system in the first method equation (2.18), we see that the cost from David Ek and Anders Forsgren (2020) is relatively less computationally expensive.

### 2.0.3 Computational cost analysis in multiplying a vector

Multiplying a vector $x \in \mathbb{R}^{n\times 1}$ to the compact update in eGuchy O, Erway JB, Marcia RF (2018) , we get

$$B_{k+1}x = B_0 x + \hat{\Psi}_k \hat{M}_k \hat{\Psi}_k^T x, \tag{2.24}$$

and in David Ek and Anders Forsgren (2020), we have

$$B_{k+1}x = B_0 x + G_{k+1} T_{k+1} G_{k+1}^T x. \tag{2.25}$$

Since $B_0$ is an identify matrix, we can focus on

$$\hat{\Psi}_k \hat{M}_k \hat{\Psi}_k^T x \tag{2.26}$$

in equation (2.28) and

$$G_{k+1} T_{k+1} G_{k+1}^T x \tag{2.27}$$

in equation (2.29).

Therefore, in equation (2.26) , we get the computational cost of $\hat{\Psi}_k^T x$

$$2(k+1)(2n-1), \tag{2.28}$$

the computational cost of $\hat{M}_k \hat{\Psi}_k^T x$

$$2(k+1)(2n-1) + 2(k+1)(4k+3), \tag{2.29}$$

and the computational cost of $\hat{\Psi}_k \hat{M}_k \hat{\Psi}_k^T x$ is $2(k+1)(2n-1)+2(k+1)(4k+3)+n(4k+3)$, which simplifies to

$$4n(k+1) + 4n(k+\frac{3}{4}) + 8k(k+1) + 4(k+1). \tag{2.30}$$

In David Ek and Anders Forsgren (2020), we can get the computational cost of $G_{k+1}^T x$

$$(k+1)(2n-1), \tag{2.31}$$

the computational cost of $T_{k+1} G_{k+1}^T x$ (notice $T_{k+1}$ here is tridiagonal)

$$(k+1)(2n-1) + 5(k-1) + 2 \times 3, \tag{2.32}$$

and the computational cost of $G_{k+1} T_{k+1} G_{k+1}^T x$ is $(k+1)(2n-1) + 5(k-1) + 2 \times 3 + n(2k+1)$, which simplifies to

$$2n(k+1) + 2n(k+\frac{1}{2}) + 4k. \tag{2.33}$$

Comparing the computational costs of two update methods in multiplying a vector, we see that method from David Ek and Anders Forsgren (2020) is relatively less expensive.

Based on the computational cost analysis we have so far, we can answer that the compact update method in David Ek and Anders Forsgren (2020) has smaller cost in multiplying a vector. Also, it is easier to update in each iteration because it requires less computational cost and has a relative simple structure. Therefore, we will use it as the compact formula in next Chapter.

# Chapter 3: Limited-memory BFGS

So far we have reviewed the Hessian matrix approximation $B_k$ at each iteration. However, in some complicated optimization problem, it might take hundreds or thousands iteration before converging to an optimal point. Under this situation, even the compact formulated BFGS needs a significant amount of storage space. It is not hard to see that if the entire minimizer searching takes thousand iterations then the costs of storing Hessian matrix approximation updates are too huge to be satisfied. Taking the $x_0 \in \mathbb{R}^{100,000}$ example again, if we use the compact BFGS update method from David Ek and Anders Forsgren (2020) and we need 10,000 iterates to converge, then we have $G \in \mathbb{R}^{100,000\times 10,000}$ which requires storage for 1 billion entries.

### 3.0.1 Limited memory BFGS

To solve the problem in the compact formulated BFGS update, applied mathematicians proposed a modified compact update method – limited-memory BFGS. In an optimization process, in general, older points contribute less valuable information about the Hessian matrix at the current iterate than the more recently computed points. Therefore, we can limit the information from previous points that are stored in current iteration by deleting some previous points to keep a bound on the storage costs. We will set up the general compact formulated BFGS and then give examples of how to update the compact BFGS under a limited memory condition.

Assume we have a general compact BFGS update as

$$B_k = B_0 + Z_k^T D_k Z_k, \tag{3.1}$$

where

$$Z_k = \begin{pmatrix} z_0 & z_1 & z_2 & \dots & z_k \end{pmatrix} \tag{3.2}$$

is a matrix that store previous points information. For example, $z_1$ stores the information from the first iteration and $z_k$ stores the information from the $k$th iteration. The core matrix $D_k$ is a matrix with $\mathbb{R}^{(k+1)\times(k+1)}$ dimension.

After setting up the notation, we can discuss how we can update this compact BFGS (equation (3.1)) under a limited-memory situation. Based on the general rules about how previous points influence current point, it is natural to come up with the idea of getting rid of some of the previous stored vectors and entries in $Z_k$ and $D_k$ since they only have very small influence on the Hessian at the current point. Therefore, given a limited memory situation which can only store $k+1$ columns of $z_i$, $i = 0, 1, 2, \dots, k$, and $(k+1) \times (k+1)$ entries of $D_k$, we can get rid of the first vector in $Z_k$ and first column and first row in $D_k$ when current iteration step reach the maximal memory limitation. After modification, we have

$$Z_k^* = \begin{pmatrix} z_1 & z_2 & \dots & z_k & z_{k+1} \end{pmatrix} \tag{3.3}$$

and

$$D_k^* = \begin{pmatrix} D_k(2:k+1, 2:k+1) & d_{k+1}^{vec}(1:k) \\ d_{k+1}^{row}(1:k) & d_{k+1}(k+1, k+1) \end{pmatrix}, \tag{3.4}$$

where $d_{k+1}^{vec}(1:k)$, $d_{k+1}^{row}(1:k)$, and $d_{k+1}(k+1, k+1)$ are the new update entries in the core matrix. We can update BFGS under the limited-memory situation by applying this update routine to all later iterations once we reach the maximal memory limitation.

Besides the method just discussed, we might also use another approach that might

be a more efficient way to update BFGS under the limited-memory situation. When current iteration reaching the maximum memory, instead of just getting rid of the first column in $Z_k$ and $D_k$, and first row in $D_k$, we can get rid of all the columns in $Z_k$ except the last column and restore the $D_k$ to $D_0$. The idea behind this approach is that we restart the limited-memory process using the last iterate as the initial point. By getting rid of all the information from previous points except the very last iteration point, the new optimization cycle begins with the last iterate as a new first iterate.This method works only when each iteration pushes the last point closer to the minimum of objective function.

### 3.0.2 Converting compact BFGS to limited memory version

In Chapter 2, through the computational cost analysis we conclude that the compact update of BFGS from David Ek and Anders Forsgren (2020) is relatively less computational expensive comparing with the compact update from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) . However, in L-BFGS we will reuse some of the elements in the compact update which might influence the cost analysis results we get in section 2.0.1. Therefore, we will discuss the computational cost after the compact updates reaching the max memory requirement and then decide which compact update to convert to limited memory version.

When convert to limited memory BFGS, no matter it is the compact update from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) or the compact update from David Ek and Anders Forsgren (2020) the computational cost of the core part $ABA$ is unchanged. This is because the size and structures of the matrices in the formula is unchanged. Therefore, the only part that will change with regard to computational cost is updating A and B.

Recall the BFGS compact update from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) is represented in DeGuchy O, Erway JB, Marcia RF (2018) as

$$B_{k+1} = B_0 + \hat{\Psi}_k \hat{M}_k \hat{\Psi}_k^T.$$

$\hat{M}_k$ has an inverse operation. Therefore, we need to recalculate $\hat{M}_k$ even if there are only few changes in updating it when reach the maximum memory requirement. Since $\hat{\Psi}_k = \Psi_k \Pi_k^T \Xi_k$ and $\Psi_k = Y_k - B_0 S_k$, if $B_0 = I$ then we only need to update the last column in $\hat{\Psi}_k$. However, if $B_0 \neq 0$ then we need to recalculate $\hat{\Psi}_k$ by using $\hat{\Psi}_k = \Psi_k \Pi_k^T \Xi_k$. In this thesis, we set $B_0 = I$ so we do not need to update the whole

part in $\Psi_k$ which will save $n(k+1)(2k+1)$ flops.

Recall the compact update from David Ek and Anders Forsgren (2020) is

$$B_k = B_0 + G_k T_k G_k^T.$$

When converting this compact update formula to limited-memory BFGS, we will drop the first column in $G_k$ and add one new column at the end of $G_k$. The new added column is the gradient of the new point so the updating cost is based on the objective function. We will do the similar thing to $T_k$ plus removing the first row and adding one new row to the bottom. Therefore, we only need to calculate the new added elements in $T_k$ which are the last three tridiagonal elements. Recall in section 2.0.1, we know that the updating cost is $3 \times (4n-1)$ and we can keep all other elements which will save $13nk - 19n - 2k + 12$ flops.

No matter it is the compact update from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) or the compact update from David Ek and Anders Forsgren (2020), the saved computational cost is not big enough to change the comparison results in Chapter 2. Therefore, we will use the compact update from David Ek and Anders Forsgren (2020) in later discussion.

From Chapter 2, we know the compact BFGS formula in David Ek and Anders Forsgren(2020) is more computational efficient. Therefore, we will convert this formula to the limited memory BFGS by using the regular approach mentioned in previous section. Before converting, we need to make sure each iteration after reaching the maximum memory is positive definite.

Recall the compact BFGS update from David Ek and Anders Forsgren (2020) is

$$B_k = B_0 + G_k T_k^C G_k^T,$$

where

$T_k^C(1,1) = \frac{1}{g_0^T p_0} + \frac{1}{\alpha_0 (g_1 - g_0)^T p_0}$

$T_k^C(i,i) = \frac{1}{g_i^T p_i} + \frac{1}{\alpha_{i-1}(g_i - g_{i-1})^T p_{i-1}} + \frac{1}{\alpha_i (g_{i+1} - g_i)^T p_i} \; i = 1, \quad ..., \quad k-1$

$T_k^C(i+1,i) = T_k^C(i,i+1) = -\frac{1}{\alpha_{i-1}(g_i - g_{i-1})^T p_{i-1}} \; i = 1, \quad ..., \quad k$

$T_k^C(k+1,k+1) = -\frac{1}{\alpha_{k-1}(g_k - g_{k-1})^T p_{k-1}}.$

Assuming we have a limit index k = 3, the optimization problem is

$$\underset{x}{\text{minimize}} \quad f(x),$$

with $g_i$ as the gradient and $p_i$ as the search direction at $i$th iteration, where $i = 0, 1, 2, ...$ . We begin the optimization iteration from point $X_0$ and we have G = $\begin{pmatrix} g_0 & g_1 & g_2 & g_3 \end{pmatrix}$, P = $\begin{pmatrix} p_0 & p_1 & p_2 \end{pmatrix}$ with the memory maximum hit.

At the 3rd iteration, the first two columns of $T_3$ is

$$T_3 = \begin{pmatrix} \frac{1}{g_0^T p_0} + \frac{1}{(g_1 - g_0)^T p_0} & -\frac{1}{(g_1 - g_0)^T p_0} \\ -\frac{1}{(g_1 - g_0)^T p_0} & \frac{1}{g_1^T p_1} + \frac{1}{(g_2 - g_1)^T p_1} + \frac{1}{(g_1 - g_0)^T p_0} \\ & -\frac{1}{(g_2 - g_1)^T p_1} \end{pmatrix},$$

and the second two columns of $T_3$ is

$$T_3 = \begin{pmatrix} & -\frac{1}{(g_2 - g_1)^T p_1} \\ \frac{1}{g_2^T p_2} + \frac{1}{(g_3 - g_2)^T p_2} + \frac{1}{(g_2 - g_1)^T p_1} & -\frac{1}{(g_3 - g_2)^T p_2} \\ -\frac{1}{(g_3 - g_2)^T p_2} & \frac{1}{(g_3 - g_2)^T p_2} \end{pmatrix}.$$

After reaching the maximum index 3, we update $G$ and $T$ by getting rid of some of the row and columns. The updated $G$ and $P$ are

$$G_4 = \begin{pmatrix} g_1 & g_2 & g_3 & g_4 \end{pmatrix}, \tag{3.5}$$

$$P_4 = \begin{pmatrix} p_1 & p_2 & p_3 \end{pmatrix}. \tag{3.6}$$

The first two columns of $T_4$ is

$$T_4 = \begin{pmatrix} \frac{1}{g_1^T p_1} + \frac{1}{(g_2-g_1)^T p_1} & -\frac{1}{(g_2-g_1)^T p_1} \\ -\frac{1}{(g_2-g_1)^T p_1} & \frac{1}{g_2^T p_2} + \frac{1}{(g_3-g_2)^T p_2} + \frac{1}{(g_2-g_1)^T p_1} \\ & -\frac{1}{(g_3-g_2)^T p_2} \end{pmatrix},$$

and the second two columns of $T_4$ is

$$T_4 = \begin{pmatrix} & -\frac{1}{(g_3-g_2)^T p_2} & \\ \frac{1}{g_3^T p_3} + \frac{1}{(g_4-g_3)^T p_3} + \frac{1}{(g_3-g_2)^T p_2} & & -\frac{1}{(g_4-g_3)^T p_3} \\ & -\frac{1}{(g_4-g_3)^T p_3} & \frac{1}{(g_4-g_3)^T p_3} \end{pmatrix}.$$

As we can see $T_4$ keeps the last three rows and columns from $T_3$, and we only need to update $T_4(1,1), T_4(4,4), T_4(3,3), T_4(3,4)$, and $T_4(4,3)$. Now we can use $X_3$ as previous iteration point, even though $X_3$ is as if we started from $X_1$. To update $X_4$ given that we can calculate $B_4$ using $G, P$ and $T_4$, which is

$$B_4 = B_0 + G_4 T_4 G_4^T. \tag{3.7}$$

Notice $B_4$ here has nothing to do with $g_0$ or $p_0$. It only involves $g1$ and $p1$ as starting condition, and therefore $B_4$ or later $B_k$ might not satisfy the positive definite assumption. By definition if $B_4$ is positive definite, then for any $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{x} \neq 0$, given $G_k$ has full column rank, we have

$$\mathbf{x}^T B_4 \mathbf{x} = \mathbf{x}^T (B_0 + G_4 T_4^C G_4^T)\mathbf{x} > 0. \tag{3.8}$$

Showing $T_4$ is positive definite is equivalent to show that $B_4$, which is the first iteration result after reaching the maximum memory, is positive definite. We first start with $T_3$ using linear algebra approach to check whether it is positive definite or not and then move to $T_4$.

Let $x \in \mathbb{R}^n$ and $\mathbf{x} \neq 0$, then $x^T G_3 = \begin{pmatrix} x^T g_0 & x^T g_1 & x^T g_2 & x^T g_3 \end{pmatrix}$ with all the elements in $\mathbb{R}$. Denote $W = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix}^T = \begin{pmatrix} x^T g_0 & x^T g_1 & x^T g_2 & x^T g_3 \end{pmatrix}^T$, then

$$W^T T_3 W = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix} T_3 \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix}^T \tag{3.9}$$

$$= \frac{w_1^2}{g_0^T p_0} + \frac{w_2^2}{g_1^T p_1} + \frac{w_3^2}{g_2^T p_2} + \frac{(w_1 - w_2)^2}{(g_1 - g_0)^T p_0} + \frac{(w_2 - w_3)^2}{(g_2 - g_1)^T p_1} + \frac{(w_3 - w_4)^2}{(g_3 - g_2)^T p_2}. \tag{3.10}$$

We can further simplify the above equation by combine the first term with fourth term, second term with fifth term and third term with the last term. Take the combination of first and fourth term as an example therefore we have

$$\frac{w_1^2}{g_0^T p_0} + \frac{(w_1 - w_2)^2}{(g_1 - g_0)^T p_0} = \frac{w_1^2 (g_1 - g_0)^T p_0}{g_0^T p_0 (g_1 - g_0)^T p_0} + \frac{g_0^T p_0 (w_1 - w_2)^2}{g_0^T p_0 (g_1 - g_0)^T p_0} \tag{3.11}$$

$$= \frac{g_1^T p_0 w_1^2 - g_0^T p_0 w_1^2}{g_0^T p_0 (g_1 - g_0)^T p_0} + \frac{g_0^T p_0 w_1^2 - 2 g_0^T p_0 w_1 w_2 + g_0^T p_0 w_2^2}{g_0^T p_0 (g_1 - g_0)^T p_0} \tag{3.12}$$

$$= \frac{g_1^T p_0 w_1^2 - 2 g_0^T p_0 w_1 w_2 + g_0^T p_0 w_2^2}{g_0^T p_0 (g_1 - g_0)^T p_0}. \tag{3.13}$$

Then use the similar calculation the equation (1.18) it can be shown that

$$W^T T_3 W = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix} T_3 \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix}^T \tag{3.14}$$

$$= \frac{g_1^T p_0 w_1^2 - 2 g_0^T p_0 w_1 w_2 + g_0^T p_0 w_2^2}{g_0^T p_0 (g_1 - g_0)^T p_0} + \frac{g_2^T p_1 w_2^2 - 2 g_1^T p_1 w_2 w_3 + g_1^T p_1 w_3^2}{g_1^T p_1 (g_2 - g_1)^T p_1} \tag{3.15}$$

$$+ \frac{g_3^T p_2 w_3^2 - 2 g_2^T p_2 w_3 w_4 + g_2^T p_2 w_4^2}{g_2^T p_2 (g_3 - g_2)^T p_2}. \tag{3.16}$$

By assumption $(g_1 - g_0)^T p_0 = g_1^T p_0 - g_0^T p_0 > 0$, therefore $g_1^T p_0 > g_0^T p_0$ then we can have the following inequality given $g_0^T p_0 < 0$

$$\frac{g_1^T p_0 w_1^2 - 2 g_0^T p_0 w_1 w_2 + g_0^T p_0 w_2^2}{g_0^T p_0 (g_1 - g_0)^T p_0} < \frac{g_0^T p_0 w_1^2 - 2 g_0^T p_0 w_1 w_2 + g_0^T p_0 w_2^2}{g_0^T p_0 (g_1 - g_0)^T p_0} \tag{3.17}$$

$$= \frac{g_0^T p_0 (w_1 - w_2)^2}{g_0^T p_0 (g_1 - g_0)^T p_0} \tag{3.18}$$

$$= \frac{(w_1 - w_2)^2}{(g_1 - g_0)^T p_0} \geqslant 0 \tag{3.19}$$

We know that $B_3$ is positive definite since it follows exactly the BFGS update procedures. However the above inequality cannot give us a clear answer about whether $B_3$ is positive definite or not. Therefore, we cannot use this approach to show that $B_4$ is positive definite. But we can switch $B_3$ back to classic BFGS formula as

$$(BFGS) \quad H_{k+1} = (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k},$$

then use the same proof in section 3.0.2 we can see that $B_3$ is positive definite.

In the compact update from David Ek and Anders Forsgren (2020), when we drop items in $G_k$ and $T_k$ we cannot switch $B_k$ to the above classic BFGS formula since the

updated $B_k$ is no longer BFGS. Recall equation (1.29), we can get the BFGS update as

$$B_k = B_0 + \sum_{i=0}^{k-1}[\frac{1}{g_i^T p_i} g_i g_i^T + \frac{1}{\alpha_i (g_{i+1} - gi)^T}(g_{i+1} - gi)(g_{i+1} - gi)^T]. \tag{3.20}$$

When we update the limited-memory BFGS, we are actually changing the summation index in equation (3.22). For example, after first hitting the memory maximum we are changing the summation index from 0 to $k-1$ to from 1 to $k$, which becomes

$$B_{k+1} = B_0 + \sum_{i=1}^{k}[\frac{1}{g_i^T p_i} g_i g_i^T + \frac{1}{\alpha_i (g_{i+1} - gi)^T}(g_{i+1} - gi)(g_{i+1} - gi)^T]. \tag{3.21}$$

But equation (3.23) is no longer a BFGS update which means we cannot convert it back to classic BFGS formula. Unfortunately, this limited-memory version is not guaranteed to be positive definite. However, in future work we can use trust region methods to solve the lack of positive definite problem since trust region method does not require the Hessian approximation to be positive definite.

# Chapter 4: Numerical Experimental Results

Other than the analysis and proofs, numerical experiments are also conducted on the limited-memory BFGS in section 3.0.3. We apply the numerical optimization problems from the CUTEst optimization problem test set (Gould, N.I.M., Orban, D. & Toint, P.L. (2015)) under the OBR2 category. The objective function of all the testing optimization problems is nonlinear and twice continuous differentiable with bounds on the variables which is the definition of the OBR2 category. In each experiment we get the initial point, objective function and its gradient from the CUTEst set as input in the limited-memory BFGS and check its positive definite property and its convergence.

Since this compact formula has a unique update method, when we implement the limited memory version into MATLAB, we need to make clear about the indices. For example, we begin with $B_0, x_0$. Then from the input, we can get $g_0$ and $p_0$ based on $B_i * p_i = -g_i$. Now we have $x_0, g_0$, and $p_0$, which also give us $x_1 = x_0 + p_0$ and $g_1$. So now we have $x_0, g_0, p_0, x_1$ and $g_1$. For the next iteration, we can update $B_1$ based on the inputs we have now. Then from $B_1 * p_1 = -g_1$ we can get $p_1$, $x_2 = x_1 + p_1$ and $g_2$. $B_k$ is the approximations to the Hessian and the bounds within the objective functions are not touched in each experiment. The $B_0$ is defined as identity matrix. A result table is listed below for all tested problems. The second column states whether the Hessian approximation $B_i$ is positive definite before reaching the maximum memory limitation, and the third column states whether the Hessian approximation $B_i$ is positive definite after hitting the maximum memory limitation. *max* means the maximum number of iterations that the maximum memory requirement can hold.

| CUTEst tested problems | | |
|---|---|---|
| Name | Positive Definite $k \leq max$ | Positive Definite $k > max$ |
| CHARDIS1 | Yes | No |
| ALLINIT | Yes | No |
| MAXLIKA | Yes | No |
| MCCORMCK | Yes | No |
| MINSURFO | Yes | No |
| MINSURFO | Yes | No |
| BDEXP | Yes | Yes |

The experimental results follow the conclusion in Chapter 3 which we cannot guarantee the positive definite of $B_k$ after hitting the maximum memory limitation. Whether $B_k$ is positive definite or not is based on the optimization problem rather than the method itself.

# Chapter 5: Conclusion and Future Work

In this thesis, we first introduce how quasi Newton's method evolves from Newton's method, and then introduce compact formulated quasi Newton's method and limited-memory BFGS. After a careful study of the computational cost and storage cost of the two compact formulated BFGS from Byrd, R.H., Nocedal, J. & Schnabel, R.B. (1994) and David Ek and Anders Forsgren (2020). We further investigate in the compact formulated BFGS in David Ek and Anders Forsgren (2020) by converting it to the limited-memory version. However, because its positive definite property of $B_k$ is not guaranteed after hitting the maximum memory limitation, a trust region method is needed.

Future work can integrate the compact formulation from David Ek and Anders Forsgren (2020) into trust-region methods.

# Bibliography


[1] DeGuchy, O, Erway, JB, Marcia, RF. Compact representation of the full Broyden class of quasi-Newton updates. Numer Linear Algebra Appl. 2018; 25:e2186. https://doi.org/10.1002/nla.2186

[2] David Ek and Anders Forsgren. Exact linesearch limited-memory quasi-Newton methods for minimizing a quadratic function. arXiv, math.OC 1809.10590, 2020.

[3] R. Fletcher. A new approach to variable metric algorithms. The Computer Journal, Volume 13, Issue 3, 1970, Pages 317–322, https://doi.org/10.1093/comjnl/13.3.317

[4] Shanno, David F. Conditioning of quasi-Newton methods for function minimization. Mathematics of computational, July 1970, 24 (111): 647–656,

[5] Broyden, C. G. A Class of Methods for Solving Nonlinear Simultaneous Equations. Mathematics of computational. American Mathematical Society. (October 1965).19 (92): 577–593. doi:10.1090/S0025-5718-1965-0198670-6

[6] Jorge Nocedal and Stephen J. Wright. Numerical Optimization. Springer. 2006

[7] Gould, N.I.M., Orban, D. & Toint, P.L. CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. Comput Optim Appl 60. 545–557 (2015). https://doi.org/10.1007/s10589-014-9687-3

[8] Byrd, R.H., Nocedal, J. & Schnabel, R.B. Representations of quasi-Newton matrices and their use in limited

memory methods..Mathematical Programming 63, 129–156 (1994). https://doi.org/10.1007/BF01582063

# Appendix A: BFGS update using $\gamma$

**Theorem A.0.1.** *Let $\phi$ equals zero. Then $B_K$ is BFGS update with structure*

$$B_k^{BFGS} = B_{k-1} - \frac{B_{k-1}s_{k-1}s_{k-1}^T B_{k-1}^T}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1}y_{k-1}^T}{y_{k-1}^T s_{k-1}}, \tag{A.1}$$

*which can be written as*

$$B_k^{BFGS} = B_0 + \gamma_{k-1} D_{k-1} \gamma_{k-1}^T \tag{A.2}$$

*with*

$$\gamma_{k-1} = \begin{pmatrix} g_0 & y_0 & \dots & g_{k-1} & y_{k-1} \end{pmatrix}$$

$$D_{k-1} = \begin{pmatrix} \frac{1}{g_0^T p_0} & & & & \\ & \frac{1}{y_0^T s_0} & & & \\ & & \cdots & & \\ & & & \frac{1}{g_{k-1}^T p_{k-1}} & \\ & & & & \frac{1}{y_{k-1}^T s_{k-1}} \end{pmatrix}.$$

*Proof.* We can use induction to prove this theorem. Let k = 1, then we have

$$B_1^{BFGS} = B_0 + \gamma_1 D_1 \gamma_1^T, \tag{A.3}$$

which can be written as

$$B_1^{BFGS} = B_0 + \begin{pmatrix} g_0 & y_0 \end{pmatrix} \begin{pmatrix} \frac{1}{g_0^T p_0} & \\ & \frac{1}{y_0^T s_0} \end{pmatrix} \begin{pmatrix} g_0 & y_0 \end{pmatrix}^T = B_0 + \frac{y_0 y_0^T}{y_0^T s_0} + \frac{g_0 g_0^T}{g_0^T p_0}. \tag{A.4}$$

The original $B_1^{BFGS}$ formula is

$$B_1^{BFGS} = B_0 - \frac{B_0 s_0 s_0^T B_0^T}{s_0^T B_0 s_0} + \frac{y_0 y_0^T}{y_0^T s_0}. \tag{A.5}$$

We know that $s_i = x_{i+1} - x_i = \alpha_i p_i$, $B_i p_i = -g_i$ and $B_i$ is symmetric, therefore 1.5 can be written as

$$B_1^{BFGS} = B_0 - \frac{B_0 s_0 s_0^T B_0^T}{s_0^T B_0 s_0} + \frac{y_0 y_0^T}{y_0^T s_0} \tag{A.6}$$

$$= B_0 - \frac{-\alpha g_0(-\alpha g_0^T)}{-\alpha g_0^T \alpha p_0} + \frac{y_0 y_0^T}{y_0^T s_0} \tag{A.7}$$

$$= B_0 + \frac{g_0 g_0^T}{g_0^T p_0} + \frac{y_0 y_0^T}{y_0^T s_0}, \tag{A.8}$$

which is equal to 1.4. Thus we proved when k=1 the theorem statement is true. Now assume k = n is true, that is

$$B_n^{BFGS} = B_{n-1} - \frac{B_{n-1} s_{n-1} s_{n-1}^T B_{n-1}^T}{s_{n-1}^T B_{n-1} s_{n-1}} + \frac{y_{n-1} y_{n-1}^T}{y_{n-1}^T s_{n-1}} \tag{A.9}$$

$$= B_0 + \sum_{i=0}^{n-1} \frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i} \tag{A.10}$$

$$= B_0 + \gamma_n D_n \gamma_n. \tag{A.11}$$

$$\tag{A.12}$$

Then for k = n+1,

$$B_{n+1}^{BFGS} = B_n - \frac{B_n s_n s_n^T B_n^T}{s_n^T B_n s_n} + \frac{y_n y_n^T}{y_n^T s_n} \tag{A.13}$$

$$= B_0 + \sum_{i=0}^{n-1}(\frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i}) - \frac{B_n s_n s_n^T B_n^T}{s_n^T B_n s_n} + \frac{y_n y_n^T}{y_n^T s_n} \tag{A.14}$$

$$= B_0 + \sum_{i=0}^{n-1}(\frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i}) - \frac{-\alpha g_n(-\alpha g_n^T)}{-\alpha g_n^T \alpha p_n} + \frac{y_n y_n^T}{y_n^T s_n} \tag{A.15}$$

$$= B_0 + \sum_{i=0}^{n-1}(\frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i}) + \frac{g_n g_n^T}{g_n^T p_n} + \frac{y_n y_n^T}{y_n^T s_n} \tag{A.16}$$

$$= B_0 + \sum_{i=0}^{n}(\frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i}). \tag{A.17}$$

Notice that

$$\gamma_{n+1} D_{n+1} \gamma_{n+1} = \gamma_n D_n \gamma_n + \frac{g_n g_n^T}{g_n^T p_n} + \frac{y_n y_n^T}{y_n^T s_n} \tag{A.18}$$

$$= \sum_{i=0}^{n-1}(\frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i}) + \frac{g_n g_n^T}{g_n^T p_n} + \frac{y_n y_n^T}{y_n^T s_n} \tag{A.19}$$

$$= \sum_{i=0}^{n}(\frac{y_i y_i^T}{y_i^T s_i} + \frac{g_i g_i^T}{g_i^T p_i}). \tag{A.20}$$

Therefore, $B_{n+1}^{BFGS} = B_0 + \gamma_{n+1} D_{n+1} \gamma_{n+1}$, and by induction theorem 1.1 is proved. □

## Appendix B: Proof of positive definite of BFGS using the Hessian approximation version

$$(BFGS) \quad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{B.1}$$

*Proof.* Recall Cauchy-Schwarz inequality is

$$< x, y >^2 \leq < x, x >< y, y >, \tag{B.2}$$

which can also be written as

$$(x^T A y)^2 \leq x^T A x y^T A y. \tag{B.3}$$

Assume $B_k$ is positive definite, then give $x \in \mathbb{R}^n$ and $x \neq 0$, we have

$$\begin{aligned} x^T B_{k+1} x =& x^T B_k x - \frac{x^T B_k s_k s_k^T B_k x}{s_k^T B_k s_k} + \frac{x^T y_k y_k^T x}{y_k^T s_k} \\ =& x^T B_k x - \frac{(x^T B_k s_k)^2}{s_k^T B_k s_k} + \frac{(x^T y_k)^2}{y_k^T s_k} \\ >& x^T B_k x - \frac{(s^T B_k s_k)(x^T B_k x_k)}{s_k^T B_k s_k} + \frac{(x^T y_k)^2}{y_k^T s_k} \\ >& \frac{(x^T y_k)^2}{y_k^T s_k}. \end{aligned}$$

Since $y_k^T s_k > 0$, therefore we know that

$$x^T B_{k+1} x > \frac{(x^T y_k)^2}{y_k^T s_k} \geq 0. \tag{B.4}$$

Therefore, $B_{k+1}$ is positive definite as long as $y_k^T s_k > 0$ and $B_k$ is positive definite. □

## Appendix C: Curriculum Vitae

*Curriculum Vitae*


# Jinxin Xia

Personal Website: jinxinxia.github.io

Email: xiajinxin27@gmail.com

Phone: 336-918-8763


---

### Education

| | |
|---|---|
| 2020 Winter | M.A. in Mathematical Statistic, GPA: 3.92/4.00<br>Wake Forest University, Winston-Salem, NC<br>Thesis Title: "L-BFGS Optimization Methods for Large-scale Problems" |
| 2018 | B.S. in Statistics, Finance (Dual Degree program with Sichuan University), Colorado State University, Fort Collins, CO, GPA: 3.76/4.00 |
| 2016 | B.M. in Labor and Social Security, GPA: 3.31/4.00<br>Sichuan University, Chengdu, China |

### Research

| | |
|---|---|
| May - Aug 2019 | Genotype Quality Control of Genetic Markers<br>*Department of Biostatistics and Data Science*<br>*Wake Forest University*<br>Mentor: Dr. Carl Langefeld |

| | |
|---|---|
| May - Aug 2020 | Batch Effects for Microbiome and Metabolomic Datah<br>*Department of Biostatistics and Data Science*<br>*Wake Forest University*<br>Mentor: Dr. Carl Langefeld |
| May - Aug 2020 | Efficient and Scalable Algorithm for Clustering via Partitioned Local Depth<br>*Department of Computer Science*<br>*Wake Forest University*<br>Mentor: Dr. Grey Ballard |
| May 2019 - Present | Optimization Methods for Large-scale Problems in Computational Genomics<br>*Department of Mathematics and Statistics*<br>*Wake Forest University (WFU)*<br>Mentor(Thesis Advisor): Dr. Jennifer Erway (NSF IIS-1741264) |

**Teaching**

| | |
|---|---|
| May 2019 - Present | Teaching Assistant<br>*Department of Mathematics and Statistics, WFU*<br>MST 656 Numerical Methods, MST 652 Numerical Linear Algebra, MST 107 Explorations in Mathematics |
| Jan 2019 - Present | Math and Statistics Tutor, Lead Tutor since Jan 2020<br>*Department of Mathematics and Statistics, WFU* |

**Technical Skills**

*Software & Programming*

R, Linux shell, Python, MATLAB, Git, Conda, LaTeX, C/C++

**Honors and awards**

Wake Forest University Partial Scholarship (2019)

Global Association of Risk Professional Research Fellowship (2017)

IMA Financial Group, Inc. Scholarship at Colorado State University (2016)

First Scholarship at Sichuan University (2013)