Optimization Project, Alternative 2
Quasi-Newton methods, DFP and BFGS

Osman Sibai

1 Introduction

This project delves into the world of Quasi-Newton methods, particularly the DFP (Davidon-Fletcher-Powell) and BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithms, and their application in optimization problems. The primary goal is to implement these algorithms in MATLAB, test their performance under various conditions, and analyze their effectiveness in solving complex optimization challenges, such as the Rosenbrock function and constrained optimization problems using penalty functions.

Quasi-Newton methods, widely used for nonlinear optimization, offer computational efficiency by approximating the Hessian matrix using gradient evaluations instead of direct computation. The DFP and BFGS algorithms stand out in this category for their robustness and are explored in detail in this project.

Our objectives include implementing these methods in MATLAB, testing their consistency with theoretical models, analyzing the effects of different starting points on algorithm performance, and comparing DFP and BFGS under various settings.

1.1 Analysis of Optimization Functions

We focus on two main types of functions:

- **The Rosenbrock Function:** Defined as $f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. This function, known for its difficult 'topography' featuring a narrow, curved valley, is used to test the performance of optimization algorithms.
- **Constrained Optimization with Penalty Function:** A complex problem minimized using a penalty function approach. Formulated as:

$$\begin{aligned} \text{minimize} \quad & e^{x_1 x_2 x_3 x_4 x_5} && (1)\\ \text{subject to} \quad & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10, && (2)\\ & x_2 x_3 = 5 x_4 x_5, && (3)\\ & x_1^3 + x_3^3 = -1. && (4) \end{aligned}$$

The function (1) is of the form $g \circ f$ where g is strictly increasing. According to theorem 6.32 (Diehl, 2023) the solution will be unchanged if we minimize f instead. So in this case we can minimize $x_1x_2x_3x_4x_5$. To be able to minimize the given function with the constraints 2 - 4 we will use penalty functions. We define the auxiliary function $\varphi(\mathbf{x}) = f(\mathbf{x}) + \mu P(\mathbf{x})$ where $f(\mathbf{x}) = x_1x_2x_3x_4x_5$, μ is the penalty parameter and $P(x)$ is the penalty function. The penalty function is defined as $P(\mathbf{x}) = \sum_{i=1}^{3} h_i(\mathbf{x})^2$. The $h_i(\mathbf{x})$ is our constraints re-written on the form $h_i(\mathbf{x}) = 0$, we get $h_1(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10$, $h_2(\mathbf{x}) = x_2x_3 - 5x_4x_5$ and $h_3(\mathbf{x}) = x_1^3 + x_3^3 + 1$. So the function that will be minimized is $\varphi(\mathbf{x})$.

The analysis includes comparing the DFP and BFGS algorithms under various conditions to evaluate their effectiveness in solving these complex problems.

2 Theory

2.1 Hessian, and how it is used

As we know for the one variable case, the second derivative is of immense importance when evaluating stationary points. Local minimums have positive second derivative and local maximums have negative second derivative. Similarly, it turns out that second derivatives are important in several variable optimization questions, as we are mainly looking for local/global minimums/maximums. However, now are partial derivatives in charge instead of "single variable" derivative. Therefore we introduce the "Hessian matrix" or simply the "Hessian" of a function f as:

The Hessian matrix $\mathbf{H}$ of a function f with respect to the vector of variables $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is defined as:

$$\mathbf{H} = \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

As mentioned before, the "sign" of the second derivative is key to differentiate between min and max when stationary points. In the one variable case, it is trivial. But in the several variable case, we consider another formulation to intemperate the "sign" of the second derivative matrix i.e. the Hessian. We therefore consider the "Definiteness" of a matrix as similar interpretation as the sign in the scalar case:

Definiteness of a Quadratic Form: Let $\mathbf{A}$ be a symmetric matrix and $\mathbf{x}$ be a column vector. The definiteness of the quadratic form associated with $\mathbf{A}$ is determined by the sign of the expression $\mathbf{x}^T\mathbf{A}\mathbf{x}$.

- If $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$, then the quadratic form is *positive definite*.
- If $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for all vectors $\mathbf{x}$, then the quadratic form is *positive semidefinite*.
- If $\mathbf{x}^T\mathbf{A}\mathbf{x} < 0$ for all nonzero vectors $\mathbf{x}$, then the quadratic form is *negative definite*.
- If $\mathbf{x}^T\mathbf{A}\mathbf{x} \leq 0$ for all vectors $\mathbf{x}$, then the quadratic form is *negative semidefinite*.
- If there exist vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ such that $\mathbf{x}_1^T\mathbf{A}\mathbf{x}_1 > 0$ and $\mathbf{x}_2^T\mathbf{A}\mathbf{x}_2 < 0$, then the quadratic form is *indefinite*.

Necessary Conditions for a Minimum in $\mathbb{R}^n$:
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. If $\mathbf{x}^*$ is a local minimum of f, then the following conditions must hold:

1. **First-order condition:** The gradient of f at $\mathbf{x}^*$ is zero.
$$\nabla f(\mathbf{x}^*) = \mathbf{0}$$
2. **Second-order condition:** The Hessian matrix of f at $\mathbf{x}^*$ is positive semidefinite.
$$\nabla^2 f(\mathbf{x}^*) \succeq \mathbf{0}$$
This means that for any vector $\mathbf{v} \in \mathbb{R}^n$, the quadratic form $\mathbf{v}^T\nabla^2 f(\mathbf{x}^*)\mathbf{v} \geq 0$.

One can mention this useful and well known lemma to easily identify positive definite matrices:

Sylvester's Criterion for Positive Definiteness: Let $\mathbf{A}$ be a symmetric matrix. The matrix $\mathbf{A}$ is positive definite if and only if all the leading principal minors have positive determinants. The kth leading principal minor of $\mathbf{A}$ is the determinant of the upper-left $k \times k$ submatrix of $\mathbf{A}$.

Mathematically:

1. $\det(\mathbf{A}_1) > 0$, where $\mathbf{A}_1$ is the 1×1 matrix formed by the first element of $\mathbf{A}$.
2. $\det(\mathbf{A}_2) > 0$, where $\mathbf{A}_2$ is the 2×2 matrix formed by the first two rows and columns of $\mathbf{A}$.
3. $\det(\mathbf{A}_3) > 0$, where $\mathbf{A}_3$ is the 3×3 matrix formed by the first three rows and columns of $\mathbf{A}$.

$\vdots$

4. $\det(\mathbf{A}_n) > 0$, where $\mathbf{A}_n$ is the $n \times n$ matrix, i.e., the whole matrix $\mathbf{A}$.

These second order derivatives are used in several numerical optimization methods, as the Newton's method. It may be valuable to introduce it before going further with the quasi-newton method.

Newton's Method in Optimization:
Newton's method is an iterative optimization algorithm used to find the minimum (or maximum) of a differentiable function. The update formula for Newton's method is given by:

Update Formula:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\nabla^2 f(\mathbf{x}_k)\right)^{-1} \nabla f(\mathbf{x}_k)$$

where:

- $\mathbf{x}_k$ is the current estimate of the minimum,
- $\nabla f(\mathbf{x}_k)$ is the gradient (first-order partial derivatives) of the objective function at $\mathbf{x}_k$,
- $\nabla^2 f(\mathbf{x}_k)$ is the Hessian matrix (second-order partial derivatives) of the objective function at $\mathbf{x}_k$, and
- $\left(\nabla^2 f(\mathbf{x}_k)\right)^{-1}$ is the inverse of the Hessian matrix.

Note: The Hessian matrix should be positive definite to be sure the inverse exist.

The advantage of Newton's method is its fast convergence. The main drawbacks are that convergence only occurs for starting points close to minimizer, that second order derivatives are needed, that matrices need to be stored. We introduce therefore Quasi-Newton Methods and use them in this project.

2.2 Quasi-Newton Methods

Quasi-Newton methods provide efficient algorithms for numerical optimization by iteratively updating approximations to the Hessian matrix of a function. Two of the most well-known methods in this category are the Davidon-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithms. Both of them utilizing Hessian approximation to not have to compute the $\nabla^2 f$ each iteration.

2.2.1 Davidon-Fletcher-Powell (DFP)

The DFP method refines an approximation of the inverse Hessian matrix, D_k, which is updated at each iteration using the formula:

$$D_{k+1} = D_k + \frac{p_k p_k^T}{p_k^T q_k} - \frac{D_k q_k q_k^T D_k}{q_k^T D_k q_k}$$

where $p_k = x_{k+1} - x_k$ represents the step taken in the variable space, and $q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ is the change in the gradient of the function. The DFP method aims to approximate the inverse of the Hessian matrix based on the observed changes in gradients, thereby informing the search direction without direct computation of the Hessian (Wikipedia, 2023).

2.2.2 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Similarly, the BFGS method updates the inverse Hessian approximation, D_k, according to the formula:

$$D_{k+1} = D_k + \left(1 + \frac{q_k^T D_k q_k}{p_k^T q_k}\right) \frac{p_k p_k^T}{p_k^T q_k} - \frac{p_k q_k^T D_k + D_k q_k p_k^T}{p_k^T q_k}$$

with p_k and q_k defined as in the DFP method. BFGS is designed to satisfy the secant equation $D_{k+1} q_k = p_k$, which ensures the updated inverse Hessian approximation reflects the curvature information along the direction of p_k.

Both methods share the goal of using curvature information to improve the search direction, yet they differ in how they update their respective Hessian approximations. They are used to solve unconstrained optimization problems, where they iteratively adjust the approximation to the Hessian matrix, compute a search direction, perform a line search, and then update the variable vector x until convergence is achieved (Wikipedia, 2023).

2.3 Some theory on Restart

When running Quasi Newton's algorithm, it is useful to consider a restart by taking a steepest descent direction every n iterations. That is because iterations might stuck with short step lengths because of bad search directions due to nearly singular D_k. Also, when being far from a minimizer, the hessian may not be positive definite and there is no interest in estimating it. We know that search directions should be strict descent anyway. So to take efficient Newton steps, the hessian should be estimated when approaching the minimizer.

2.4 Line Search Method

We opted for the golden section method, with bracketing, as our line search technique in our optimization algorithms, prioritizing its balance between simplicity and efficiency. Also we had already implemented this before. This approach, simpler than more complex strategies like the Armijo rule or Wolfe conditions,

eliminates the need for gradient information, finding derivatives, and additional parameter tuning. By focusing on interval reduction, the golden section method provides a clear and understandable process, even though it might require more iterations compared to other methods. Its straightforward implementation and reliability make it particularly suitable for our project's needs. Adhering to the principle of 'keep it simple, stupid' (KISS), we found the golden section method to be an ideal fit, ensuring ease of debugging and predictability, essential for handling complex optimization tasks effectively.

3 Optimization Results

3.1 Consistency testing

To test the consistency of our program we used a randomly generated quadratic function of the form $q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$. Where the A matrix is created by generating a random $n \times n$ matrix, lets call it B, where the elements of B, $b_{i,j} \in [-1, 1]$. Then $A = (B + B^T)/2 + nI$, this way A will be a positive definite matrix, and q will have a minimizer.

This function was tested for $n = 10, 25, 50, 100$ and both methods worked well for all of them with a random starting point where $x_i \in [-n, n]$ in the starting point, see table 1. The number of function evaluations was about the same for the different methods and most of the variation came from different matrices A and different starting points.

n	Iterations	Function evaluations
10	7-9	300-400
25	9-10	700-800
50	10-13	1300-1700
100	13-15	3000-3500

Table 1: Optimization results for q. Where n is the dimension of the function, iterations and function evaluations denotes the approximate ranges of the respective variables in our testing.

3.2 Problem 1 - The Rosenbrock Function

Table 2 to 5 summarizes the optimization results for the Rosenbrock function using first the DFP and then the BFGS method across a range of initial points. The tolerance in the tables refers to the termination criteria where the method terminates if $\|\nabla f(\mathbf{x})\| <$ tolerance.

Note: Only the tests without restart were run with the corrected program, but the tests with restart are expected to have increased function evaluations of similar degree.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Evaluations
-2 2	1.0000 1.0000	9.4325e-12	15	505
0 0	1.0000 1.0000	1.1950e-12	13	380
2 2	1.0000 1.0000	1.4055e-10	26	867
-1 3	1.0000 1.0000	1.0246e-11	26	854
12 -9	1.0000 1.0000	1.3138e-10	51	2227
-100 100	75 5653	6.7732e+03	1000	35028
200 -100	169 28438	2.8792e+04	1000	27449

Table 2: Optimization results for the Rosenbrock function using the DFP method with tolerance 0.001 without restart and max iterations 1000.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Restart After N Iterations	Evaluations
-2 2	1.0000 1.0000	2.0117e-11	22	2	330
-2 2	1.0000 1.0000	3.7808e-10	20	6	437
0 0	1.0000 1.0000	1.3020e-09	21	2	327
0 0	1.0000 1.0000	8.9012e-14	12	6	259
2 2	1.0001 1.0001	3.4177e-09	25	2	361
2 2	0.9999 0.9998	5.9029e-09	27	6	580
-1 3	1.0000 0.9999	1.3218e-09	39	2	573
-1 3	1.0000 1.0000	8.8142e-14	29	6	629
12 -9	1.0000 0.9999	6.7984e-10	115	2	1716
12 -9	1.0000 1.0000	4.2021e-13	70	6	1558
-100 100	66 4293	4.1645e+03	1000	2	15887
-100 100	1.0001 1.0002	1.0414e-08	252	6	6321
200 -100	-160 25738	2.6062e+04	1000	2	16164
200 -100	1.0000 1.0000	1.3967e-14	516	6	13243

Table 3: Optimization results for the Rosenbrock function using the DFP method with tolerance 0.001 with restart and max iterations 1000.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Evaluations
-2 2	1.0000 1.0000	1.4210e-11	16	457
0 0	1.0000 1.0000	4.5336e-10	12	337
2 2	1.0000 1.0000	3.8824e-12	27	762
-1 3	1.0000 1.0000	3.2150e-11	26	727
12 -9	1.0000 1.0000	4.0774e-15	56	1575
-100 100	1.0000 1.0000	1.1709e-11	155	4365
200 -100	1.0000 1.0000	1.0776e-10	264	7489

Table 4: Optimization results for the Rosenbrock function using the BFGS method with tolerance 0.001 without restart and max iterations 1000.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Restart After N Iterations	Evaluations
-2 2	1.0000 1.0001	1.1614e-09	25	2	367
-2 2	1.0000 1.0000	4.3790e-13	21	6	427
0 0	1.0000 0.9999	1.3020e-09	21	2	323
0 0	1.0000 1.0000	4.9997e-10	11	6	229
2 2	1.0001 1.0001	4.0048e-09	25	2	359
2 2	1.0002 1.0005	5.9405e-08	25	6	496
-1 3	1.0000 0.9999	1.3309e-09	39	2	572
-1 3	1.0000 1.0000	9.7067e-14	29	6	589
12 -9	1.0000 1.0001	8.3677e-10	97	2	1447
12 -9	1.0000 1.0000	3.5916e-12	69	6	1403
-100 100	66 4283	4.1548e+03	1000	2	15886
-100 100	1.0000 1.0000	6.1213e-12	250	6	5253
200 -100	-160 25707	2.6031e+04	1000	2	16168
200 -100	1.0000 1.0000	5.2183e-11	381	6	8066

Table 5: Optimization results for the Rosenbrock function using the BFGS method with tolerance 0.001 with restart and max iterations 1000.

3.3 Problem 2 - Constrained Optimization with Penalty Function

Table 6 to 9 summarizes the results of the optimization of the constrained optimization problem. In all of the tests the penalty parameter was set to $\mu = 100$. A number of different minimizers where found. There was three different function values reached in these minimizers with the lowest function value being 0.0538. Some of the function values was reached in different points with symmetry in some variables. The tolerance in the tables refers to the termination criteria where the method terminates if $\|\nabla f(\mathbf{x})\| <$ tolerance.

Note: Only the tests without restart were run with the corrected program, but the tests with restart are expected to have increased function evaluations of similar degree.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Evaluations
-2 2 2 -1 -1	-1.7173 1.8274 1.5960 -0.7642 -0.7642	0.0538	58	2650
4 -2 1 4 5	-1.7173 -1.8274 1.5960 -0.7642 -0.7642	0.0538	48	2155
4 -2 1 -4 5	-0.6999 -2.7892 -0.8709 -0.6972 -0.6973	0.4382	44	2126
52 -75 -41 12 -76	-0.6999 -2.7892 -0.8709 0.6972 0.6973	0.4382	115	5728

Table 6: Optimization results for the exponential function using the DFP method with tolerance 0.001 without restart and max iterations 1000.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Restart After N Iterations	Evaluations
-2 2 2 -1 -1	-1.7173 1.8274 1.5960 -0.7642 -0.7642	0.0538	74	5	1304
-2 2 2 -1 -1	-1.7173 1.8274 1.5960 -0.7642 -0.7642	0.0538	53	15	1386
4 -2 1 4 5	-1.7173 -1.8274 1.5960 0.7642 -0.7642	0.0538	66	5	1135
4 -2 1 4 5	-1.7173 -1.8274 1.5960 0.7642 -0.7642	0.0538	45	15	1198
4 -2 1 -4 5	-0.6999 -2.7892 -0.8709 -0.6972 -0.6973	0.4382	129	5	2334
4 -2 1 -4 5	-0.6999 -2.7892 -0.8709 -0.6972 -0.6972	0.4382	51	15	1331
52 -75 -41 12 -76	-0.6999 -2.7892 -0.8710 0.6971 0.6975	0.4382	99	5	1621
52 -75 -41 12 -76	-0.6999 -2.7892 -0.8709 0.6972 0.6972	0.4382	142	15	3849

Table 7: Optimization results for the exponential function using the DFP method with tolerance 0.001 with restart and max iterations 1000.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Evaluations
-2 2 2 -1 -1	-1.7173 1.8274 1.5960 -0.7642 -0.7642	0.0538	48	1636
4 -2 1 4 5	-1.7173 -1.8274 1.5960 -0.7642 0.7642	0.0538	51	1748
4 -2 1 -4 5	-0.6999 -2.7892 -0.8709 -0.6972 -0.6972	0.4382	73	2544
52 -75 -41 12 -76	-0.6999 2.7892 -0.8709 0.6972 -0.6973	0.4382	137	4882

Table 8: Optimization results for the exponential function using the BFGS method with tolerance 0.001 without restart and max iterations 1000.

Initial Point	$\mathbf{x}_{optimal}$	$f(\mathbf{x}_{optimal})$	Iterations	Restart After N Iterations	Evaluations
-2 2 2 -1 -1	-1.7173 1.8274 1.5960 -0.7642 -0.7642	0.0538	55	5	957
-2 2 2 -1 -1	-1.7173 1.8274 1.5960 -0.7642 -0.7642	0.0538	53	15	1272
4 -2 1 4 5	-1.7173 -1.8274 1.5960 -0.7642 0.7642	0.0538	95	5	1617
4 -2 1 4 5	-1.7173 -1.8274 1.5960 -0.7642 0.7642	0.0538	45	15	1097
4 -2 1 -4 5	-0.6999 -2.7892 -0.8709 -0.6973 -0.6973	0.4382	44	5	760
4 -2 1 -4 5	-0.6999 -2.7892 -0.8709 -0.6973 -0.6972	0.4382	72	15	1787
52 -75 -41 12 -76	2.1916 0.0000 -2.2589 0.3074 -0.0000	1.0000	60	5	917
52 -75 -41 12 -76	-0.6999 -2.7892 -0.8709 0.6972 -0.6972	0.4382	72	15	1798

Table 9: Optimization results for the exponential function using the BFGS method with tolerance 0.001 with restart and max iterations 1000.

4 Analysis

4.1 Program Consistency and Testing

The consistency was tested for "nice" quadratic functions with positive definite hessian matrices (see section 3.1). In our tests there were no problems to minimize these functions even up to 100 dimensions. A quadratic function with positive definite hessian matrix is not a challenging function to minimize but we are satisfied that the program works for an arbitrary dimension of reasonable size. According to theory Quasi-Newton methods will find the minimizer of a quadratic function in at most $n+1$ iterations (Diehl, 2023, p.72-73). As we can see in table 1 we where within this range for all of the tests.

4.2 General Analysis

We tested different initial points for the given functions, both the BFGS and the DFP method, with and without restart. First of all, we can say that almost all of the tested starting points resulted in a minimizer being reached within a 1000 iterations, with a few exceptions for the Rosenbrock function with different methods. We can see in table 2 and 3 that the DFP method can reach a minimizer from more starting points if an appropriate restart condition is set. In our tests we tested to restart every n and $3n$ iterations where n is the dimension of the problem. In this case to restart after $n = 2$ iterations did not improve the consistency, however to restart after $3n = 6$ iterations seemed a more appropriate restart condition. For the BFGS method we can see in table 4 and 5 that to restart to often, every $n = 2$ iterations, will make the consistency worse.

4.3 Problem 1 - The Rosenbrock Function

If we start by considering the case with no restart in table 2 and 4 we can see that there seem to be some correlation between number of iterations and distance to the minimizer. However, it is not so simple so that the closest starting point always leads to fewest iterations. See for example the point (-2, 2) and (2, 2) where (2, 2) is the closest to the minimizer in (1, 1) yet it takes fewer iterations from (-2, 2) to reach it. But we can say from this that generally it is a good idea to pick a starting point near the minimizer if you have a rough idea of where it is, this holds true for both DFP and BFGS. If we also consider the case where we are allowed to restart the same still holds true. However, whether restarts are a good idea for this problem is hard to answer. For DFP the robustness can be improved with restart after 6 iterations and a fewer number of function evaluations is required for most starting points. But for BFGS the robustness can actually be worse when restart is used the method restarts after 2 iterations. In our tests the BFGS method without restart worked best, we had no problems with robustness and for most of the starting points the

number of function evaluations was lower or about equal to the other methods, with some exceptions.

4.4 Problem 2 - Constrained Optimization with Penalty Function

This problem was a bit more complex and had multiple minimizers. Some minimizers was symmetric in some variables and thus produced the same function values. If we look at the function to minimize this is expected. Similar to the Rosenbrock function, the choice of the initial point plays a crucial role in the convergence of the optimization algorithms. The distance from the initial point to the optimal solution affects the number of iterations and evaluations required for convergence. For example, starting points closer to the optimal solution generally lead to faster convergence. However, here we face the problem of different minimizers. A number of different starting points are tested and the minimum function value 0.0538 is found. What minimizer we converge towards depend on the starting point but also the algorithm. If we look at tables 6 to 9 we see that no convergence problem occurred for this problem. Compared to the Rosenbrock function there was a bigger difference between the BFGS and DFP method for this problem. For both with and without restart BFGS out preformed DFP in respect to the number of function evaluations. There was also a bigger difference between with and without restart for this problem. For both DFP and BFGS it worked better in most cases with restart. The most successful method with respect to number of function evaluations was BFGS with restart after 5 iterations. But it is not as easy to compare for this problem since different methods reach different minimizers.

4.5 Comparison of Quasi-Newton Methods

When using the DFP and the BFGS method it is not easy to come up with any general settings or choice of method to be objectively best in all cases. We will try to summarize our findings. Probably the most important choice is the starting point, for two reasons. It is important because starting near a minimizer will require fewer function evaluations to reach the minimiser. It is also important, as can be seen in problem 2, because different starting point will sometimes result in different minimizers. If you know roughly where a global minimizer is it likley to be a good idea to start around there.

The BFGS method seems to be a bit more reliable for the Rosenbrock function and required fewer function evaluations for problem 2. So if we where to recommend any one method it would be BFGS. As for restarts it seems like a good idea to use some form of restart from looking at the results. We would recommend to restart every $3n$ iterations since it improves robustness for the DFP method on the Rosenbrock function and still decreases the number of required function evaluations in problem 2. However, as these where not the optimal

settings for any one problem it shows the difficulty in making general choices that works well for every problem.

5 Example Minimization Printout

Here is an example of a printout for the Rosenbrock function where the BFGS method is used without restart, tolerance = 0.001 and initial point (-2,2):

Iteration	$\mathbf{x}$	$f(\mathbf{x})$	norm($\nabla f(\mathbf{x})$)	function evals	λ
1	2.0520 3.0092	145.4650	1017.0835	5	0.0025
2	1.9925 3.9082	1.3661	52.6521	7	0.0029
3	1.9888 3.9557	0.9777	1.5168	20	1.0119
4	1.8301 3.3238	0.7548	21.0568	27	79.2897
5	1.7030 2.8653	0.6167	26.2009	20	1.8713
6	1.6462 2.7189	0.4257	4.9811	18	0.7342
7	1.5315 2.3295	0.3081	11.3212	18	0.7463
8	1.4562 2.0960	0.2676	15.8925	20	1.3188
9	1.1399 1.2940	0.0226	2.9747	21	5.3584
10	1.1440 1.3073	0.0210	1.0082	14	0.1073
11	1.0695 1.1384	0.0078	2.6965	21	4.1371
12	1.0339 1.0709	0.0015	0.8427	21	2.3886
13	1.0115 1.0223	0.0002	0.3764	17	0.4457
14	0.9992 0.9980	0.0000	0.1475	21	2.3108
15	1.0002 1.0004	0.0000	0.0036	18	0.6692
16	1.0000 1.0000	0.0000	0.0002	20	1.1687

References

[1] Diehl, Stefan. (2023). Optimization A Basic Course. *Studentliteratur*

[2] Wikipedia contributors. (2023). BFGS method. In *Wikipedia, The Free Encyclopedia*. Retrieved 2023-12-14, from https://en.wikipedia.org/wiki/BFGS_method

[3] Wikipedia contributors. (2023). Davidon–Fletcher–Powell formula. In *Wikipedia, The Free Encyclopedia*. Retrieved 2023-12-14, from https://en.wikipedia.org/wiki/DavidonFletcherPowell_formula

A Program Code

```
function [x_optimal, fval, iter, N_eval, normg] =
    nonlinearmin(func, x0, method, tol, restart,
    printout)
    % Initialize variables
    if isrow(x0)
    % Convert row vector to column vector
        x = x0';
    else
        % If it's already a column vector or not a
            vector, leave it unchanged
        x = x0;
    end
    iter_restart = 3*length(x0);
    maxIter = 1000;
    H = eye(length(x0)); % Hessian inverse
       approximation
    iter = 0;
    [gradient, N_fun_eval] = grad(func, x);
    N_eval = N_fun_eval;

        % Print header if printout is enabled
    if printout
        fprintf('iteration\t x\t\t f(x)\t\t norm(grad)
           \t ls fun evals\t lambda\n');
    end

    % Main loop
    while true
        % Check convergence
        if norm(gradient) < tol
            break;
        end

        % Determine search direction
        d = -H * gradient;
```

```
% Line search
F =@(lambda) func(x+lambda*d);
[alpha, fval, N_fun_eval] = lineSearch(F, tol)
    ; % Sample parameters
N_eval = N_eval + N_fun_eval;

% Update variables
s = alpha * d;
x_new = x + s;
[grad_new, N_fun_eval] = grad(func, x_new);
N_eval = N_eval + N_fun_eval;
q = grad_new - gradient;
p=x_new-x; % (Could be removed since s = p)

% Update Hessian approximation
if strcmpi(method, 'BFGS')
    H = H + 1/(p'*q)*((1+(q'*H*q)/(p'*q))*(p*p
        ') - H*q*p' - p*q'*H);
elseif strcmpi(method, 'DFP')
    H = H + (p*p')/(p'*q) - ((H*(q*q')*H)/(q'*
        H*q));
end

% Update for next iteration
x = x_new;
gradient = grad_new;
iter = iter + 1;

% Print current iteration details if printout
    is enabled
if printout
    fprintf('%d\t\t %.4f\t\t %.4f\t\t %.4f\t\t
        %d\t\t %.4f\n', iter, x(1), fval, norm
        (gradient), N_fun_eval, alpha);
    for i = 2:length(x)
        fprintf('\t\t %.4f\n', x(i));
    end
end


% Check for max iterations
if iter >= maxIter
    break;
end
```

```
        % Restart logic
        if restart && mod(iter, iter_restart) == 0
            H = eye(length(x)); % R eset Hessian
                approximation
        end
    end
    % Prepare output
    x_optimal = x;
    iter;
    fval;
    N_eval;
    normg = norm(gradient);
end

%%
function [x, fval, N_eval] = lineSearch(F,tol)
b = 2;
k_max=10000;
alpha=2;
F_0 = F(0);
N_eval = 1;

[b, N] = bracketing(b,F,alpha,k_max,F_0);
N_eval = N_eval + N;

[x, fval, N] = golden_section(F, 0, b, tol);
F_x = F(x);
N_eval = N_eval + 1 + N;
if isnan(F_x) || F_x>F_0
    error('Bad job of the line search!')
end
end

%%
function [b, N] = bracketing(b,F,alpha,k_max, F_0)

N = 0;
while F(b)>F_0
    b=b/alpha;
    N = N + 1;
end
k=0;
while F(b)<F_0 && k < k_max
    k=k+1;
    b=b*alpha;
    N = N + 1;
```

```
end
if k==k_max
    error('minimizer may not exist')
end

%%
function [x, fval, N] = golden_section(F, a, b, tol)
    % Initialize the variables
    N = 0; % Function evaluation counter
    X = []; % Matrix to store the interval updates
    tau = (sqrt(5) - 1) / 2; % Golden ratio constant

    % Calculate the initial points
    x1 = a + (1 - tau) * (b - a);
    x2 = a + tau * (b - a);
    % Evaluate the function at x1 and x2
    f1 = F(x1);
    f2 = F(x2);

    % Update the counter two times beacuse call
       function two time lol
    N = N + 2;

    while (b - a) > tol
        % now only have to call them once each
           iteration=good
        if f1 < f2
            b = x2;
            x2 = x1;
            f2 = f1;
            x1 = a + (1 - tau) * (b - a);
            f1 = F(x1);
        else
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = a + tau * (b - a);
            f2 = F(x2);
        end

        % Update the counter
        N = N + 1;

        % Store the current a, b, and (b-a)
        X = [X; a, b, b - a];
```

```
    end
    x = (a+b)/2;
    fval=F(x);
end

%%
function [g, N_fun_eval] = grad(f,x)
% g = grad(f,x)
%
% Calculates the gradient (column) vector of the
   function f at x.

lx = length(x);
g = zeros(lx,1);
N_fun_eval = 0;
for i = 1:lx
   xplus = x;
   xminus = x;
   xplus(i) = x(i) + 1.e-8;
   xminus(i) = x(i) - 1.e-8;
   g(i,1) = ( f(xplus) - f(xminus) )/2.e-8;
   N_fun_eval = N_fun_eval + 2;
end
```