In this coursework, I am going to implement three algorithms which are used to find the minimum of the target functions in Python. These three algorithms, respectively, are the quasi-Newton DFP algorithm with an exact line search, the Steepest Descent method with an exact line search and the quasi-Newton DFP algorithm with an iterative line search method. The first two algorithms are used for a quadratic function, and the third algorithm is used for a Rosenbrock's function.

I start with the quasi-Newton DFP algorithm with an exact line search [1]. I define a function as "newton_quasi" and initially set $k = 0,\ x^{(0)} = init,\ C^{(0)} = I_n$ and $accuracy = \left|\left|\nabla f(x^{(0)})\right|\right|$ where k represents the iteration, $x^{(0)}$ represents the initial estimate, $C^{(0)}$ is an identity matrix and $accuracy$ represents the norm of the gradient at $x^{(k)}$. Then I create a while loop which terminates as $accuracy = \left|\left|\nabla f(x^{(k)})\right|\right| < 10^{-6}$. In the while loop, the direction, $d^{(k)} = - C^{(k)} g^{(k)}$ where $g^{(k)} = \nabla f(x^{(k)})$, is calculated every iteration. The point at which we cease using the direction $d^{(k)}$ is determined via the exact line search which determines the step length parameter α which satisfies that $f(x^{(k)} + \alpha^{(k)} d^{(k)}) \approx min_{\alpha} f(x^{(k)} + \alpha^{(k)} d^{(k)})$. The derivative of such function is $f'(x^{(k)} + \alpha^{(k)} d^{(k)}) = (\nabla f(x^{(k)} + \alpha^{(k)} d^{(k)}))^T (d^{(k)})$ and so $f'(x^{(k)} + \alpha^{(k)} d^{(k)}) = 0 \Rightarrow \alpha^{(k)} = \frac{-d^{(k)^T} g^{(k)}}{d^{(k)^T} \cdot A^T \cdot d^{(k)}}$ where A is is symmetric from the quadratic target function $f(x) = \frac{1}{2} x^T A x + b^T x$. Hence, for a quadratic target function, an exact line search is achieved in the quasi-Newton algorithm by setting $\alpha^{(k)} = \frac{-d^{(k)^T} g^{(k)}}{d^{(k)^T} \cdot A^T \cdot d^{(k)}}$. Inputting $\alpha^{(k)}$ and $d^{(k)}$, the new estimate, $x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$, is calculated. Next step, the matrix $C^{(k+1)}$ is updated by satisfying Davidon-Fletcher-Powell (DFP) formula. By imposing the quasi-Newton condition and the approximation $C^{(k)}$ to $A^{-1}(x^{(k)})$, respectively $C^{(k+1)} \Delta g^{(k)} = \Delta x^{(k)}$ and $A(x^{(k)})^{-1} \Delta g^{(k)} \approx \Delta x^{(k)}$, I have the form

$C^{(k+1)} = C^{(k)} + \mu aa^T + \nu bb^T \Rightarrow C^{(k+1)} = C^{(k)} + \frac{\Delta x^{(k)} \Delta x^{(k)^T}}{\Delta x^{(k)^T} \Delta g^{(k)}} - \frac{C^{(k)} \Delta g^{(k)} \Delta g^{(k)^T} C^{(k)}}{\Delta g^{(k)^T} C^{(k)} \Delta g^{(k)}}$ where

$\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$, $\Delta g^{(k)} = g(x^{(k+1)}) - g(x^{(k)})$. As iterations move on, $C^{(k)}$ is then made to approximate more and more closely to $A^{-1}(x^{(k)})$. The last step, I set k add itself by 1 each iteration and calculate $accuracy = \left|\left|\nabla f(x^{(k+1)})\right|\right|$ at new estimate $x^{(k+1)}$. The while loop so ends while $accuracy$ reaches the stopping criterion which is 10^{-6} and the outputs of this function, "newton_quasi", are a list of $x^{(k)}$ and the number of iterations k.

Next, I check if this quasi-Newton DFP algorithm works on the quadratic target function, $f(x) = \frac{1}{2} x^T Ax + b^T x$. The gradient of $f(x)$ is $\nabla f(x) = g(x) = \frac{1}{2} A^T x + \frac{1}{2} Ax - b = Ax - b$ since A is symmetric in this function. By inputting f, g, A and initial coordinate, the last element in the output list is an array, [[21],[-13],[8],[-5]]. I notice that such quadratic function has a unique local minimum [2] $x^* = -A^T b = -Ab$ which is [[21],[-13],[8],[-5]]. Hence, this algorithm successfully finds the extreme point and the minimum of target function is -159.5.

```
array([[ 21.],
       [-13.],
       [  8.],
       [ -5.]])
```

Figure 1. the last element in the output list

```
array([[ 21.],
       [-13.],
       [  8.],
       [ -5.]])
```

Figure 2. calculated by -Ab

The second algorithm is the Steepest Descent method with an exact line search [3]. I define a function as "steepest" and initially set $k = 0$, $x^{(0)} = init$ and $accuracy = \left|\left|\nabla f(x^{(0)})\right|\right|$ where k represents the iteration, $x^{(0)}$ represents initial estimate and $accuracy$ represents the norm of the gradient at $x^{(k)}$. Then I create a while loop which terminates as $accuracy = \left|\left|\nabla f(x^{(k)})\right|\right| < 10^{-6}$. In the while loop, Steepest

Descent Method starts by selecting the search direction to be in the opposite direction to the gradient vector i.e. $d^{(k)} = -\nabla f(x^{(k)}) = -g(x^{(k)})$ where g is the gradient of function f each iteration. The point at which we cease using the direction $-g(x^{(k)})$ is determined via the exact line search which determines the step length parameter α which satisfies that $f(x^{(k)} - \alpha^{(k)} g^{(k)}) \approx min_{\alpha} f(x^{(k)} - \alpha^{(k)} g^{(k)})$. The derivative of such function is $f'(x^{(k)} - \alpha^{(k)} g^{(k)}) = (\nabla f(x^{(k)} - \alpha^{(k)} g^{(k)}))^T (-g^{(k)}) = -g^{(k)^T} g^{(k)} + \alpha^{(k)} g^{(k)^T} A^T g^{(k)}$ and so $f'(x^{(k)} - \alpha^{(k)} g^{(k)}) = 0 \Rightarrow \alpha^{(k)} = \frac{\|g^{(k)}\|^2}{g^{(k)^T} \cdot A^T \cdot g^{(k)}}$ where A is is symmetric from the quadratic target function $f(x) = \frac{1}{2} x^T A x + b^T x$. Hence, for a quadratic target function, an exact line search is achieved in the steepest descent algorithm by setting $\alpha^{(k)} = \frac{\|g^{(k)}\|^2}{g^{(k)^T} \cdot A \cdot g^{(k)}}$. Such α > 0 and minimises $f(\alpha) = f(x^{(k)} - \alpha g(x^{(k)}))$. The new estimate $x^{(k+1)} = x^{(k)} - \alpha^{(k)} g^{(k)}$ is calculated. Finally, we set the $k = k + 1$ and calculate $accuracy = \left\|\nabla f(x^{(k+1)})\right\|$ at new estimate $x^{(k+1)}$ each iteration. The while loop terminates while reaching the stopping criterion which is 10^{-6} and the outputs of this function, "steepest", are a list of $x^{(k)}$ and the number of iterations k.

Then I check this algorithm. The last element in the output list is an array, [[20.999],[-12.999],[7.999],[-4.999]]. Therefore, the second algorithm works as well.

```
array([[ 20.99991612],
       [-12.99999453],
       [  7.99998909],
       [ -4.99997584]])
```

Figure 3. the last element in the output list

```
array([[ 21.],
       [-13.],
       [  8.],
       [ -5.]])
```

Figure 4. calculated by -Ab

By comparing the first two algorithms used on the same quadratic function, I have some conclusions. Firstly, for this quadratic function and initial guess on the first coordinate, the number of iterations under quasi-Newton DFP algorithm is 4 and the number of iterations under Steepest Descent method is 8260, which means using quasi-Newton DFP algorithm is much more computationally efficient than using Steepest Descent method. This is usually true that Newton's method minimizes a quadratic in a few steps because the quadratic approximation is the same as the original function. Even if the search direction under Steepest Descent is always a descent direction, the convergence rate of which is $O(\frac{1}{k})$ where k is the number of iterations and so results in lack of monotone convergence; however, by using $C^{(k)}$ to approach $A^{-1}(x^{(k)})$, without the need of second derivatives, quasi-Newton is still able to maintain the properties i.e. converging quadratically and quadratic termination. Secondly, although none of these two algorithms requires second derivatives (i.e., the Hessian matrix), quasi-Newton is a little bit more complicated than Steepest Descent since $C^{(k)}$should be updated every iteration while using quasi-Newton.

	Advantages	Disadvantages
quasi-Newton	Fast convergence. Quadratic termination No need of second derivatives	A little bit more complicated Needs good initial estimate Descent search direction not guaranteed
Steepest Descent	Search direction is always a descent direction No need of second derivatives	Slow convergence

Figure 5. Advantages and Disadvantages

The Third algorithm is the quasi-Newton DFP algorithm with an iterative line search method. I define a function as “newton_quasi_ILS”. The only difference here is that I use an iterative line search method [4] instead of using exact line search. I define a function inside as “ILS” which returns the step length parameter α which minimizes $f(x^{(k)} + \alpha^{(k)} d^{(k)})$. I write a for loop and impose Armijo (Sufficient Decrease) Condition, $f(x^{(k)} + \alpha^{(k)} d^{(k)}) \leq f(x^{(k)}) + c \cdot \alpha^{(k)} d^{T} \nabla f(x^{(k)})$ where the initial values are $c = 10^{-3}$, $\alpha = 1 \; and \; \rho = 0.5$. This condition ensures the computed step length can sufficiently decrease the objective function. If Armijo Condition does not satisfy, which means the minimum has not been reached, the new alpha, $\alpha = \alpha \cdot \rho$ will be used to test Armijo Condition in the next iteration. As Armijo Condition is satisfied, the alpha which minimizes $f(x^{(k)} + \alpha^{(k)} d^{(k)})$ will be returned.

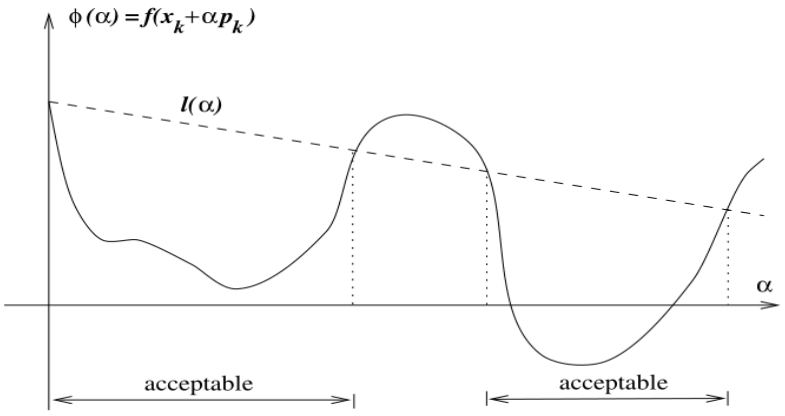

Figure 6. Armijo Condition

Then I check this algorithm. The last element in the output list is an array, [[0.999],[0.999]]. The function, $f(x_1, x_2) = 100(x_2 - x_1^{\,2})^2 + (1 - x_1)^2$, has the minimum at the point [[1],[1]]. Therefore, the quasi-Newton DFP algorithm with an iterative line search method works. The number of iterations is 35 and the minimum of the target function is 0. The figure below is the plot of Rosenbrock's function and the red path is the plot of the output list of coordinates under the algorithm. I see that except the first

iteration which is orthogonal to the contours since $C^{(0)} = I_n$, the quasi-Newton's method spheres the contours in every next step and moves in a direction orthogonal to the sphered contours, however, steepest descent, as a contrast, moves in a direction orthogonal to the last path in every next step.

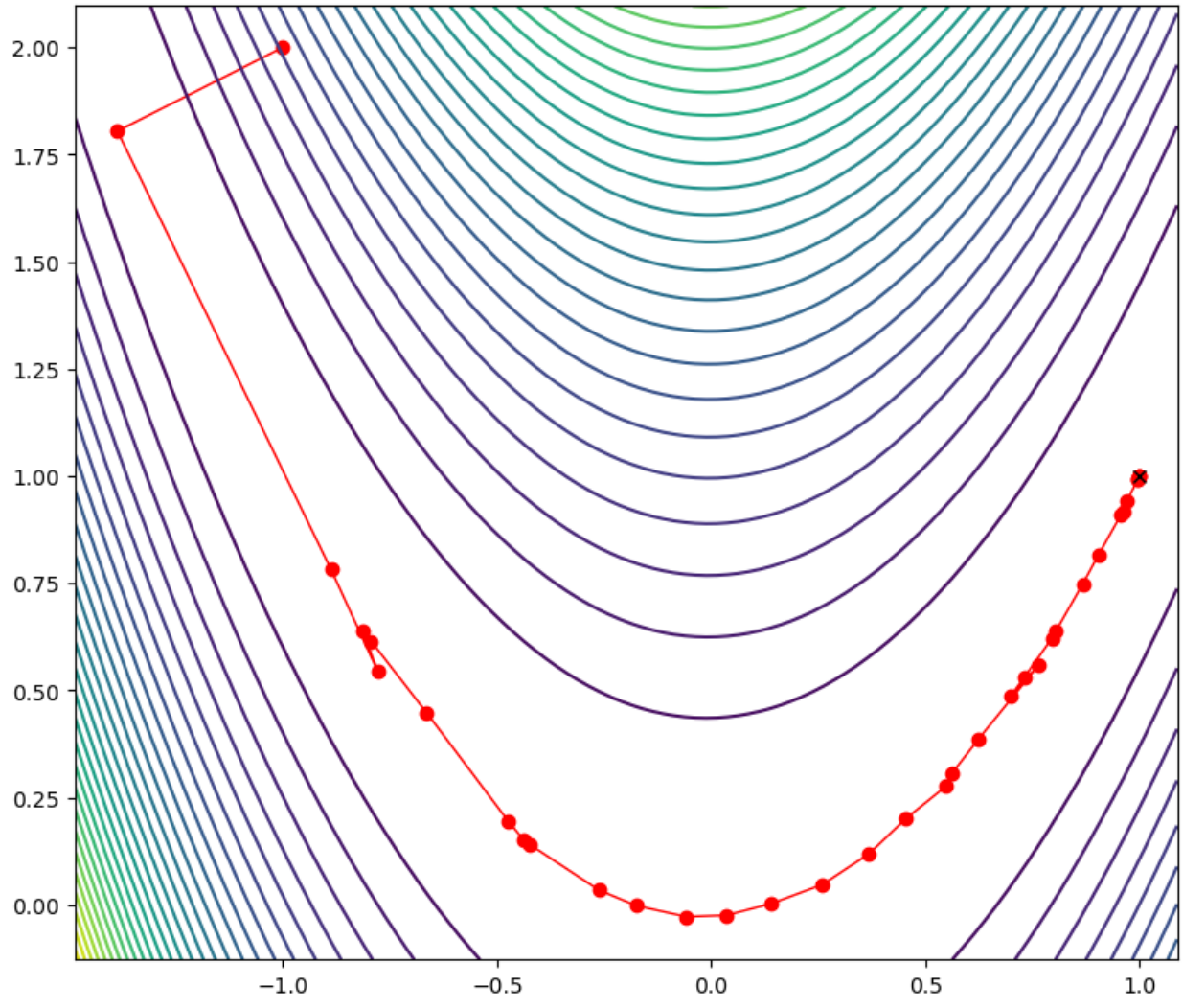

Figure 7. Contour surface and red path

References

1. Mats Vermeeren†. (2023). *Static and Dynamic Optimisation, pp. 56–59*. Available at: https://learn.lboro.ac.uk/pluginfile.php/2433667/mod_resource/content/7/chapter4-0320.pdf (Accessed: 1 May 2023).
2. Mats Vermeeren†. (2023). *Static and Dynamic Optimisation, pp. 38*. Available at: https://learn.lboro.ac.uk/pluginfile.php/2433667/mod_resource/content/7/chapter4-0320.pdf (Accessed: 1 May 2023).
3. Mats Vermeeren†. (2023). *Static and Dynamic Optimisation, pp. 37–39*. Available at: https://learn.lboro.ac.uk/pluginfile.php/2433667/mod_resource/content/7/chapter4-0320.pdf (Accessed: 1 May 2023).
4. Lihe Cao, Zhengyi Sui, Jiaqi Zhang, Yuqing Yan, and Yuhui Gu. (Fall 2021). *Line search methods*. Available at: https://optimization.cbe.cornell.edu/index.php?title=Line_search_methods (Accessed: 1 May 2023).