# IOE 511/MATH 562: Homework № 3 Part 2

Due: March 9th, 2023

## [1]

Function 2 Problem:

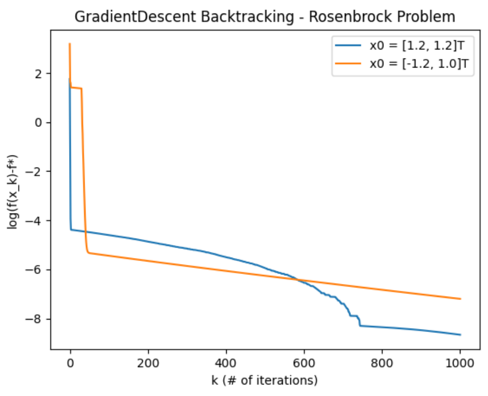
$$\begin{aligned}
f(x) &= \sum_{i=1}^{3} (y_i - w(1-z^i))^2, \text{where } x = [w\ z]^T \in R^2 \\
&= ((y_1 - w(1-z^1))^2 + (y_2 - w(1-z^2))^2 + (y_3 - w(1-z^3))^2) \\
\text{Gradient: } \nabla_x f(x) &= \begin{bmatrix} \sum_{i=1}^3 2(w(1-z^i) - y_i)(1-z^i) \\ \sum_{i=1}^3 2w(iz^{i-1})(y_i - w(1-z^i)) \end{bmatrix} \\
\text{Hessian: } \nabla_x^2 f(x) &= \begin{bmatrix} \sum_{i=1}^3 2(1-z^i)^2 & \sum_{i=1}^3 2(iz^{i-1})(y_i - 2w(1-z^i)) \\ \sum_{i=1}^3 2(iz^{i-1})(y_i - 2w(1-z^i)) & 30w^2z^4 + 12w^2z^2 + 12w(y_3 - w)z - 2w^2 + 4y_2w \end{bmatrix}
\end{aligned}$$

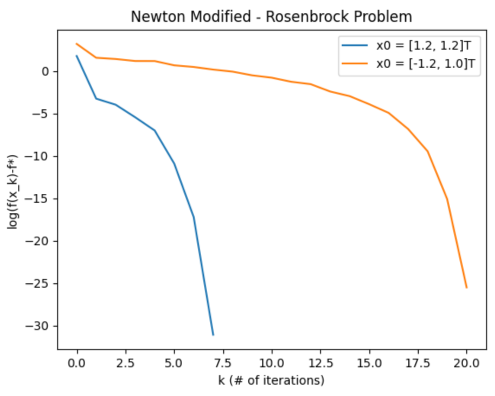
Function 3 Problem:

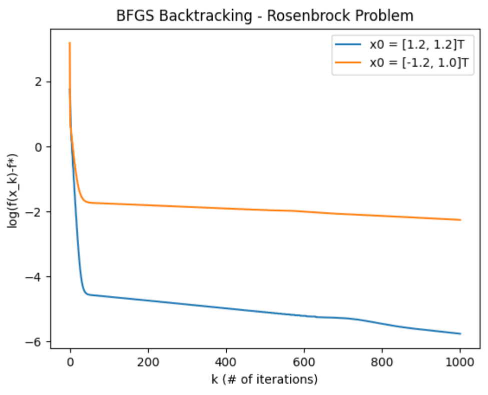
$$\begin{aligned}
f(x) &= \frac{e^{x_0} - 1}{e^{x_0} + 1} + 0.1e^{-x_0} + \sum_{i=1}^{n} (x_i - 1)^4 \\
\text{Gradient: } \nabla_x f(x) &= \left[ \frac{2e^{x_0}}{(e^{x_0}+1)^2}, 4(x_1-1)^3, 4(x_2-1)^3, ..., 4(x_n-1)^3 \right]^T \\
\text{Hessian: } \nabla_x^2 f(x) &= \left[ \frac{2e^{x_0}(1-e^{x_0})}{(e^{x_0}+1)^3} + 0.1e^{-x_0}, 12(x_1-1)^2, 12(x_2-1)^2, ..., 12(x_n-1)^2 \right]^T
\end{aligned}$$

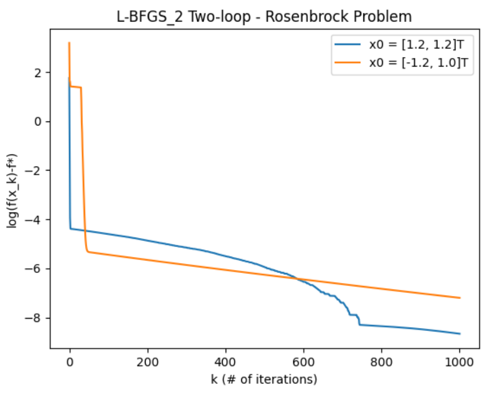
## [4] Algorithm Implementation Report

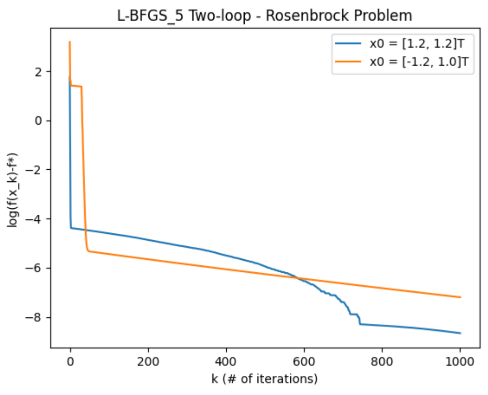
## 1 Rosenbrock Problem

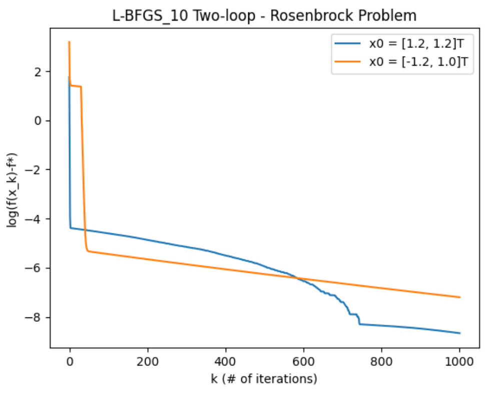












1. Convergence:

   In Rosenbrock Problem, Modified Newton's method converged below the termination tolerance value, which is $10^{-6}$, but the other six algorithms did not converge within 1000 iterations yet. They all behave as converging algorithms, so increasing the number of iterations will lead most of these algorithms to converge within the maximum iterations. Newton's method, which is the only converged algorithm in the Rosenbrock problem within 1000 iterations, x* at x0 = $[1.2, 1.2]^T$ is $[1.000, 1.000]^T$ with f* = $3.227 \times 10^{-14}$ in 8 iterations, and x*, f* at $x0 = [-1.2, -1]^T$ are $[0.999999, 0.999998]^T$ and $8.517 \times 10^{-12}$, respectively, and it converged in 21 iterations. As it converges, x* is almost identical to the theoretically ideal x*.

   To be specific about the converging status of the other five algorithms with two given starting points at the 1000th iteration (k=1000), s1: when x0 = $[1.2, 1.2]^T$, s2: $x0 = [-1.2, -1]^T$

Gradient Descent Backtracking: (s1) $x_k$ = [1.01276 1.02537], $f_k$ = 0.00017322 // (s2) $x_k$ = [0.97278 0.94602], $f_k$ = 0.00074842
BFGS Backtracking: (s1) $x_k$ = [1.0561 1.1152] , $f_k$ = 0.0031485 // (s2) $x_k$ = [1.3234 1.7508], $f_k$ = 0.10463
L-BFGS (memory length m=2): (s1) $x_k$ = [1.01276 1.02537], $f_k$ = 0.00017322 // (s2) $x_k$ = [0.97278 0.94602] , $f_k$ = 0.00074842
L-BFGS (m=5): (s1) $x_k$ = [1.0127633 1.02536825] , $f_k$ = 0.00017322276919479591 // (s2) $x_k$ = [0.97277594 0.9460233] , $f_k$ = f = 0.00074842
L-BFGS (m=10): (s1) $x_k$ = [1.0127633 1.02536825], $f_k$ = 0.00017322 // (s2) $x_k$ = [0.97277594 0.9460233], $f_k$ = 0.00074842

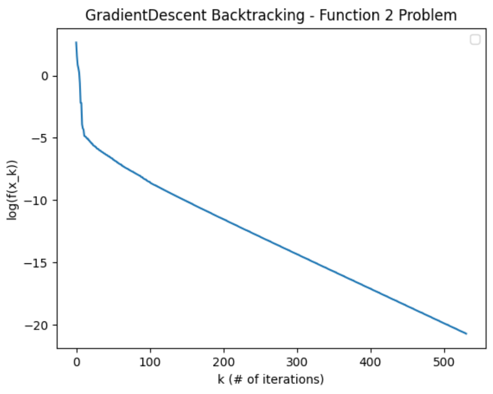
2. Modified Newton's method:
   No iteration was modified in the Modified Newton's method in Rosenbrock problem.

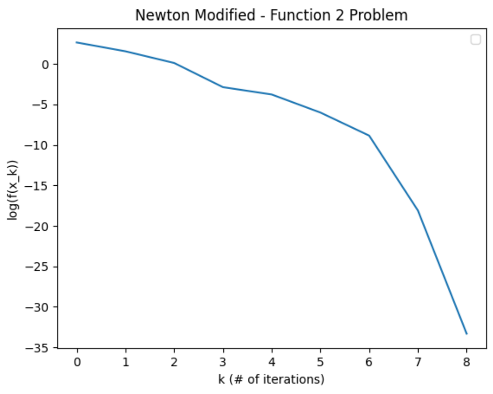
3. BFGS method:
   No iteration has a skip update of sufficiently positive, so there was no skipped iteration.

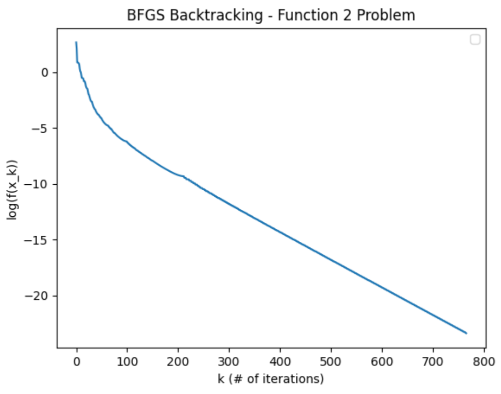
4. Convergence rate:
   All algorithms are found with linear + superlinear convergence rate based on log scale plot, especially Newton's method has nearly a perfect superlinear convergence rate, as we can guess from the plots.

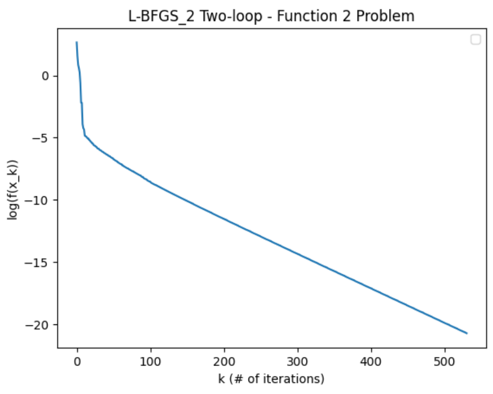
5. Computation time: For implementing two cases of starting points, each algorithm takes approximately 0.5s, 0.2s, 0.8s, 0.4s, 0.4s, and 0.4s. As Newton's method converges the earliest, its computation time is the shortest (It also has a superlinear convergence rate that quadratically converges, which has a fast convergence speed).

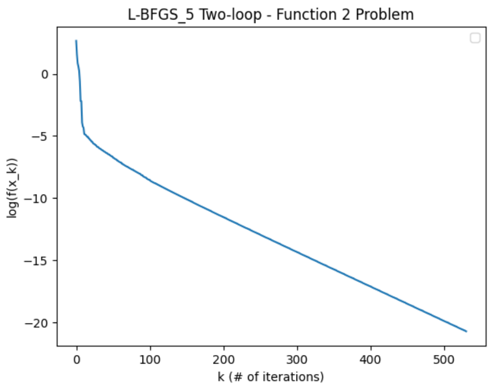
## 2 Function 2 Problem

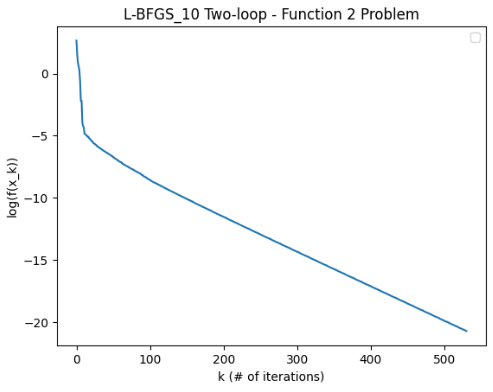












1. Convergence:

   Contrary to the Rosenbrock problem, all the algorithms using Function 2 converges within maximum iterations of 1000. Gradient Descent backtracking converges in 530 iterations (k = 530), with final x and f values of $x_k$ = [2.9999 0.49998], $f_k = 1.0069 \times 10^{-9}$, Newton's method converges in 8 iterations (k = 8) which is notably fast in converging rate, with $x_k$ = [3.0000 0.50000] and $f_k = 3.3979 \times 10^{-15}$. BFGS backtracking converges in 765 iterations (k = 765), $x_k$ =[2.9999802 0.49999565] and $f_k = 7.01357 \times 10^{-11}$, and L-BFGS (m=2) converges in 530 iterations (k = 530) with $x_k$ = [2.9999 0.49998] and $f_k = 1.00695 \times 10^{-9}$. For L-BFGS with memory lengths of 5 and 10, they both converge within 530 iterations (k = 530) with similar $x_k$ and $f_k$ values, approximately $x_k$ = [3.0000 0.50000] and $f_k$ $1.0070 \times 10^{-9}$.

2. Modified Newton's method:

   There was no modified iteration.

3. BFGS method:

   There were no skipped iterations.

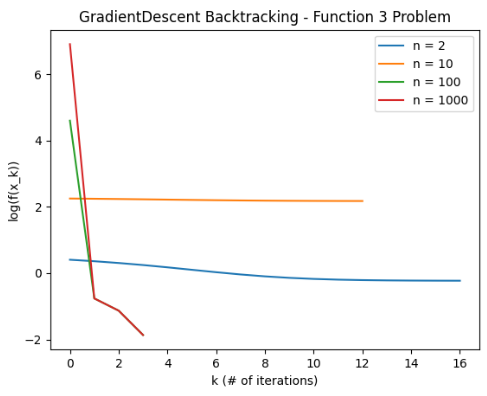
4. Convergence rate:

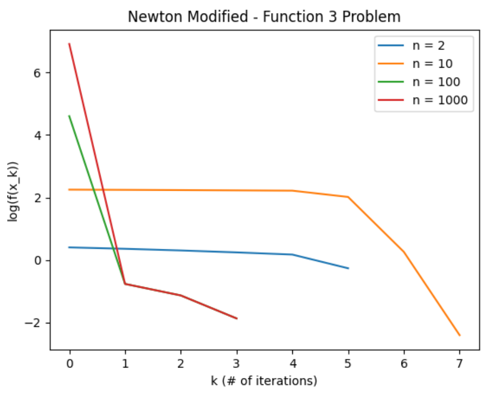
   All five algorithms except for Newton's method seem roughly linearly converging with some nature of sublinear convergence. Likely from the Rosenbrock problem, Newton's method is superlinearly converging, which means its convergence rate is faster than other algorithms. We can find this from the number of iterations and executed time of code (computation time).

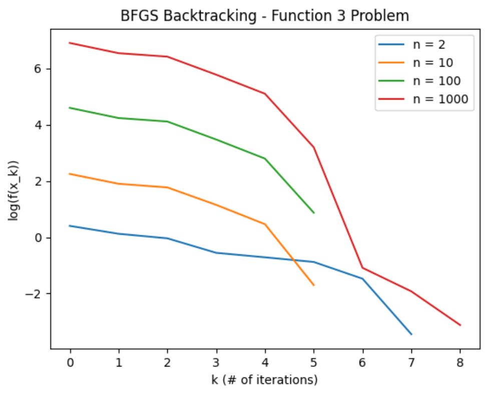
5. Computation time:

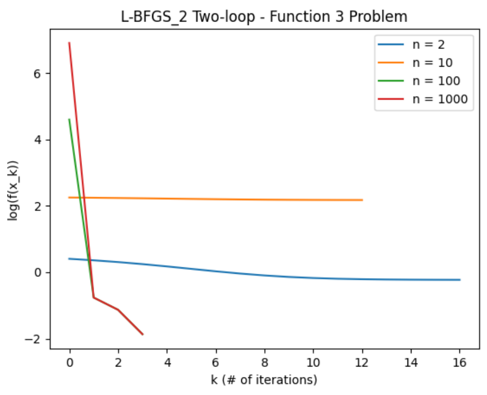
   Similarly to the Rosenbrock problem, Newton's method computes the fastest in 0.2s, and it follows by GD backtracking and Limited Memory-BFGS (m = 2, 5, 10) with all 0.3s, and the latest in 0.5s, which is BFGS with non-limited memory.

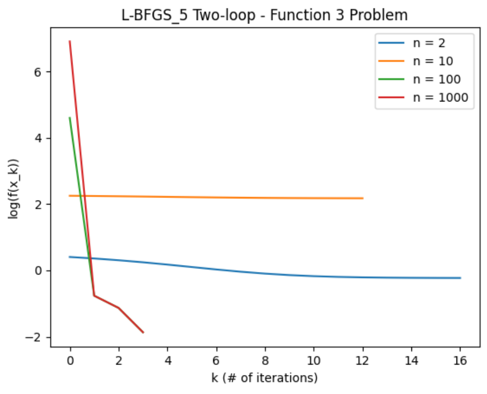
## 3 Function 3 Problem

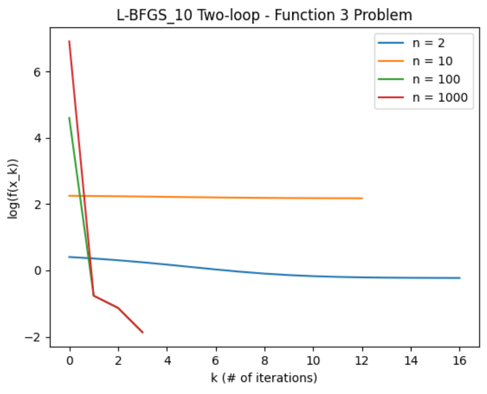












1. Convergence:
   (Values of x and f at convergence are all printed on *script.ipynb* file.) With any given dimensions of n = 2, 10, 100, 1000 of $x$, they all converge before they reach the maximum iteration counts, and convergence speed in Function 3 problem is comparatively faster than other problems. For instance, GD Backtracking with n = 100 and 1000 converge in 4 iterations, with terminated value of $x_{n=100}$ = [-1.24477 1.0000], $x_{n=1000}$ = [-1.24477 1.0000 ... 1.0000], $f_{n=100} = f_{n=1000} = -0.205573$. For n = 2 and 10, they converge slower, which are in 17 and 13 iterations with having all same x[0] value and different size of x vector filled with 1.000s. Second, in Newton's method, $x_{n=2}$ = [-1.24476995 0.99261379] with $f$ of -0.205572. Regardless of the dimension (n), they all converges to the same f values and the dimension of 2, 10, 100, 100 are respectively convering in 6, 8, 4, and 4 iterations, and the convergence speed seems the faster in higher dimension of x. In BFGS without memory limit, this is the only algorithm out of six that all x with different dimensions behave very similarly with each others. In this case, n = 2 and 1000 converge in 8-9 iterations, and both 10 and 100 converge in 6 iterations. Interestingly, n = 2 and 1000 has the similar tendency. For Limited memory BFGS with three different memory lengths are all converged and here also observed that x with higher dimension converges notably faster than the other two.

2. Modified Newton's method:
   There were modified iterations.

3. BFGS method:
   There were skipped iterations.

4. Convergence rate:
   Let me first talk about five algorithms' behaviors besides BFGS backtracking. Their convergence speeds of higher dimensions of x (n = 100, 1000) are substantially greater than using x of dimension size 2 and 10. Thus, in the plots, we can observe that n = 10 and 100 converges linearly (or lightly superlinear). In contrast, x of n = 100 and 1000 converges linear sublinearly, which means the speed of convergence in each iteration is slower but it actually converges earlier than n = 10 and 100. And now, let's look at BFGS backtracking algorithm which behaves the most distinctively different, their speed of convergence (shape of the line on the plot) look alike until the sixth iteration. However, in cases of n = 2 and 1000 (each side of extreme), their convergence rates' gradient behave reversely from its tendency until k = 7, which is interesting. In short, they all seem to have convergence rate of superlinear.

5. Computation time:
   To generate four plot lines of different n values in the same graph, each takes computation times to execute as 0.3s, 1.6s, 115.9s, 0.3s, 0.4s, 0.4s in order of LRTB in page 5. GD backtracking and L-BFGS algorithms computed in the fastest time frames similar to previous problems (computation time of Newton's method also increased, but I can guess that this is because of implementing for the multiple x cases). However, there is a noticeable difference in computation time in BFGS backtracking. This can be induced from that BFGS does not have the memory limit, especially for n = 100 and 1000 (bigger input parameter size) which may cause a huge increase in execution time of algorithm implementation. Just to be more detailed, as expected, computation time of each x

with n = 2, 10, 100, and 1000 are 0.089, 0.089, 0.084, 0.138 seconds that I can say the time are roughly proportional to the size of parameters.