Numerical Methods

**Assignment1: Solving a Non-Linear Equation**

**Name: Hong Se Hyun**  **ID: 21700791**

**Instructions:**

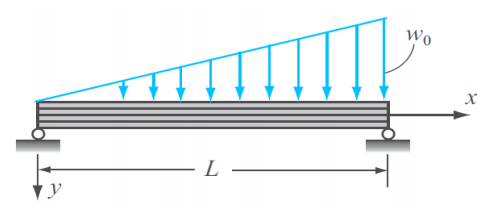
Declare and define the numerical method functions in your header file “myNM.h”, “myNM.c”, respectively.

The main program file, ‘Assignment\_nonlinear.c’, contains the main function and calls yours NM functions to solve the assignment problems.

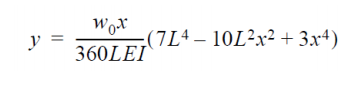
You must submit, report file and program file: “myNM.h”, “myNM.c”, “Assignment\_nonlinear.c” on Hisnet.

**Problem: Solve the following non-linear equation [20pt]**

A simply supported I-beam is loaded with a distributed load, as shown. The deflection, y, of the center line of the beam as a function of the position, x, is given by the equation:



From the solution of the equation of motion for this model, the steady-state up-and-down motion of the car (mass) is given by x(t), due to the wheel motion of y(t). The ratio between amplitude X and amplitude Y is given by:



where L=4m is the length, E=70 GPa is the elastic modulus, I=52.9\*10-6 m4 is the moment of inertia, and w0=20kN/m.

Find the position x where the deflection at the point where the deflection of the beam is maximum and determine the deflection at this point.

HINT: The maximum deflection is at the point where dy/dx=0=f(x).

a. Solve for the solution using MATLAB’s functions of fzero( )

MATLAB을 통해, 문제에서 주어진 함수가 어느 부분에서 Deflection이 가장 심하게 일어나는지 확인할 수 있다.

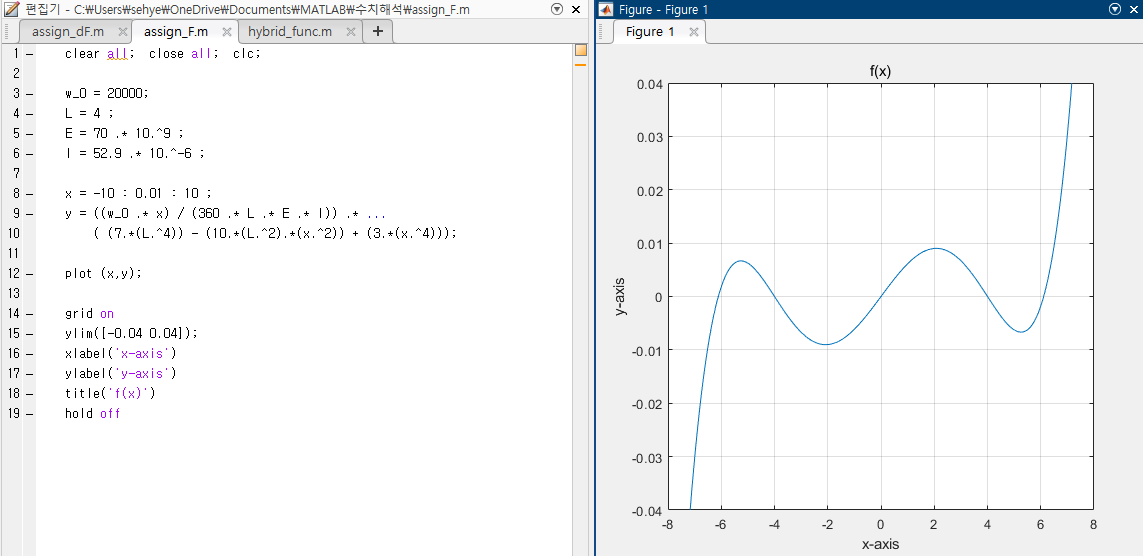


Figure 1.1. Graph of given function F

Beam의 길이가 4인 것을 생각해 보면, 위 그래프에 의해 값이 약 2 정도에서 Deflection이 가장 심한 것을 확인할 수 있다. Deflection이 심하게 일어나는 지점의 정확한 좌표를 찾기 위해서는, 주어진 함수의 기울기가 0이 되는 순간 지점을 찾아주면 된다.

즉, 주어진 함수를 미분하여 를 찾아주고, 의 해를 구간 0부터 4사이에서 찾아주면 되는 것이다.

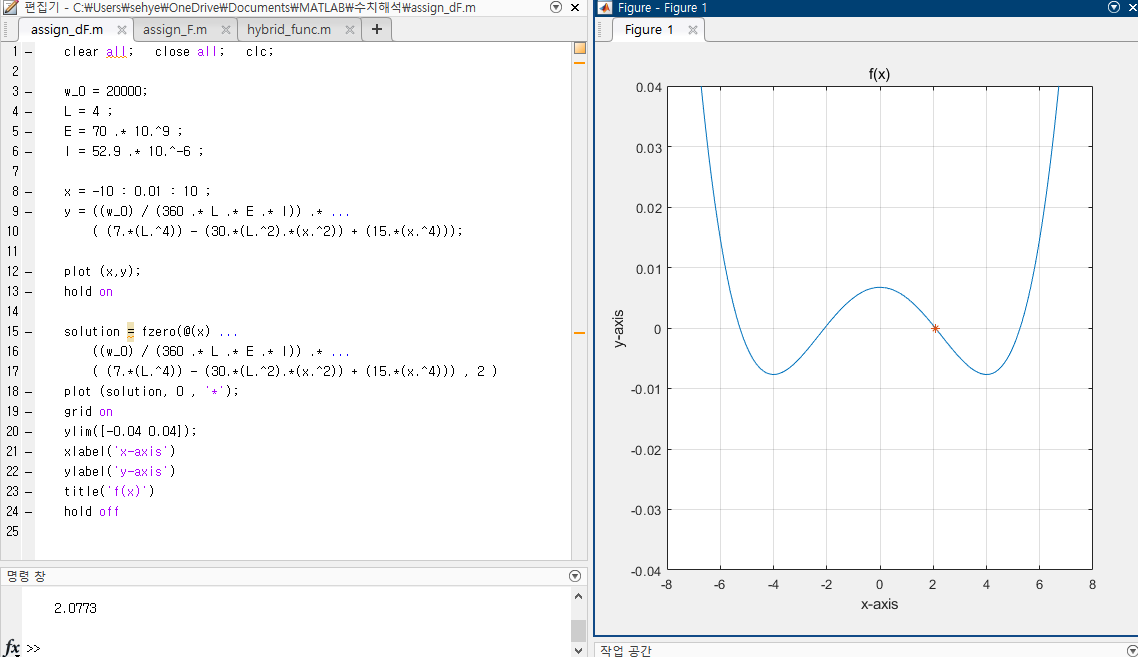


Figure 1.2. Graph of given function dF

매틀랩을 통해 Deflection이 최대가 되는 지점의 x좌표가 2.07임을 확인할 수 있다.

이제는, Bisection method와 Newton-Raphson method를 적용하여 정말로 해가 2.07에 근사하게 나오는지 확인해보고 Matlab의 결과와 비교해 주면 된다.

b. Use your defined Newton-Rhapson method to solve for the solution.

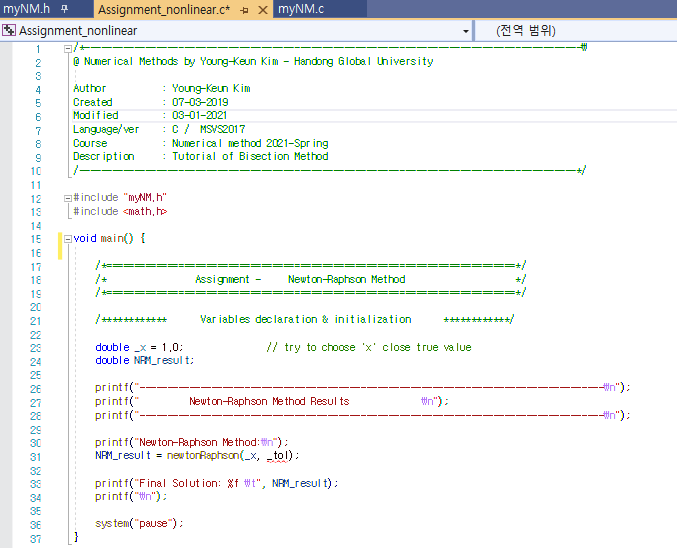


Figure 2.1. Main Code of Newton-Raphson Method

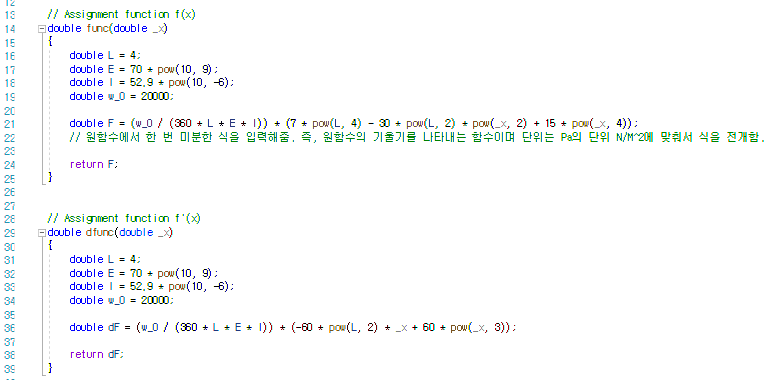


Figure 2.2. Function Code of Newton-Raphson Method

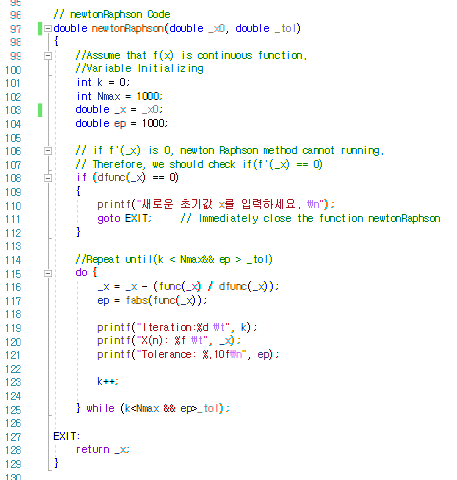


Figure 2.3. Code of Newton-Raphson Method

인 점에서는 접선을 그어도 에 만날 수가 없게 되기 때문에, Newton-Raphson method를 이용하기 위해서는 108행과 같이 반드시 Error Checking Code를 넣어줘야 한다.

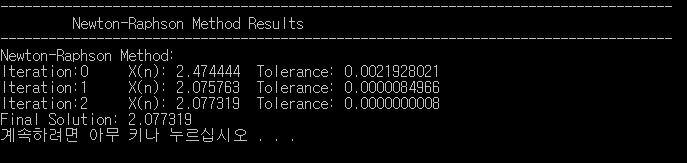


Figure 2.4. Result of Newton-Raphson Method

c. Compare the result with MATLAB’s solution

[Figure 1.2.]를 보면, Matlab으로 얻은 해는 2.0773임을 확인할 수 있으며, [Figure 2.4.]를 통해서도 Newton-Raphson method를 통해 얻은 해 역시 2.0773이 나왔다.

즉, 두 결과값이 거의 일치하다고 봐도 무방하며 Newton-Raphson method를 통해서 실제 값과 아주 근사한 해를 찾을 수 있음을 확인할 수 있다.

d. Compare the results of method a) with bisection method: (final answer, number of iterations, tolerance)

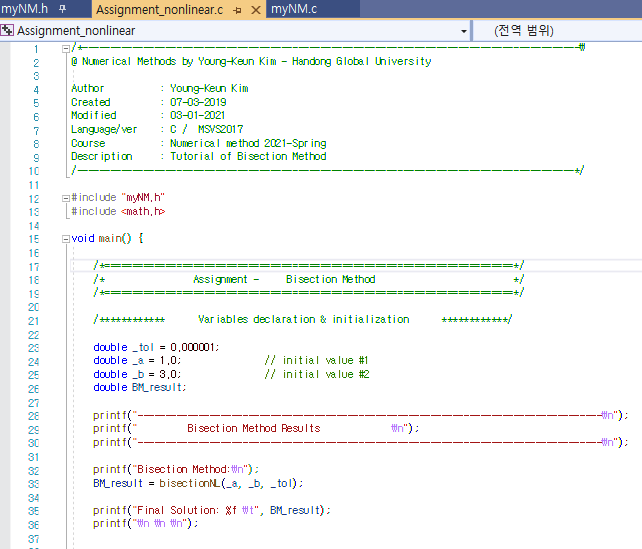


Figure 3.1. Main Code of Bisection Method

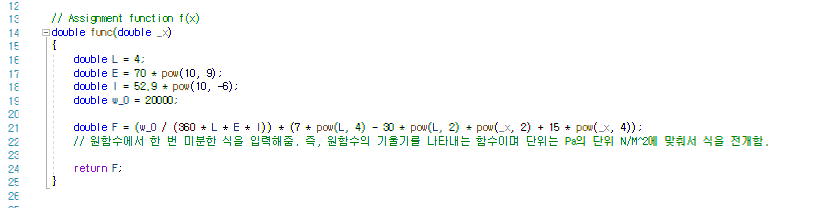


Figure 3.2. Function Code of Bisection Method

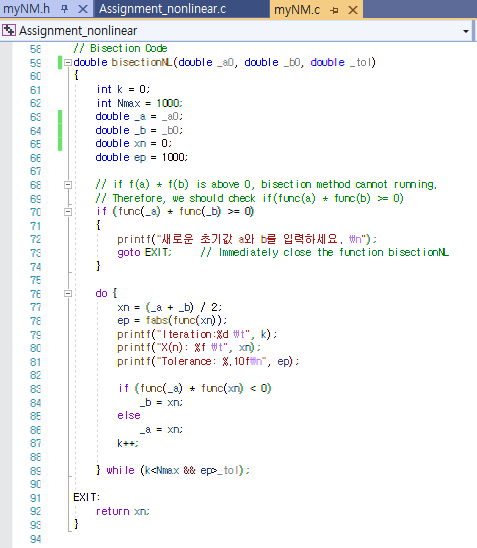


Figure 3.3. Code of Bisection Method

인 구간에서는 Bisection Algorithm의 특성상, 제대로 된 기능을 할 수 없게 되기 때문에 70행과 같이 반드시 Error Checking Code를 넣어줘야 한다.

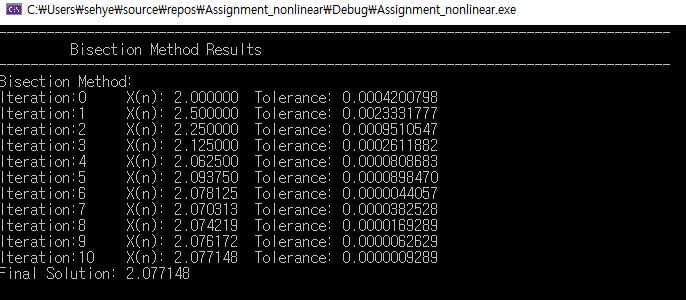


Figure 3.4. Result of Bisection Method

[Figure 1.2.], [Figure 2.4.], [Figure 3.4.]을 살펴보면, 순서대로 Matlab의 결과값과 Newton-Raphson의 결과값과 Bisection의 결과값이 거의 일치한다고 볼 수 있다.

즉, Newton-Raphson method뿐만 아니라 Bisection method를 통해서도 2.077이라는 값을 얻음으로써 실제 값과 아주 근사한 해를 찾을 수 있음을 확인할 수 있다.

한편, number of iterations의 관점에서 Newton-Raphson method와 Bisection method를 비교해보면, Newton-Raphson method에서 값이 더 빨리 수렴함을 확인할 수 있다.

물론, 어느 한 점에서 접선을 그어가며 해를 찾는 방식인 Newton-Raphson method와, 어느 두 점의 중점을 기준으로 대소 관계를 비교하면서 해를 찾는 방식인 Bisection method의 특성상, 어느 초기값을 주는지에 따라서 혹은 어느 범위를 입력해 주는지에 따라서 Iteration은 언제든지 바뀔 수 있다. 하지만, 계속해서 초기값을 바꿔가며 Newton-Raphson method와 Bisection method의 수렴 속도를 비교해도, 결과는 동일하게 Newton-Raphson이 더 빠르게 해를 찾았다.

[Figure 4.1.]과 [Figure 4.2.]를 참고하도록 한다. [Figure 4.1.]에서는 Bisection의 초기 범위로 1.5와 2.5를 입력했으며 Newton-Raphson method의 초기값으로 1.5를 입력했다. [Figure 4.2.]에서는 Bisection의 초기 범위로 2.0과 2.1을 입력했으며 Newton-Raphson method의 초기값으로 2.0을 입력했다. 해가 2.0773임을 생각해 보면 Bisection의 범위로 2.0과 2.1을 준 것은 굉장히 파격적임에도 불구하고, Newton-Raphson method의 수렴 속도를 따라가지 못함을 확인해볼 수 있다.

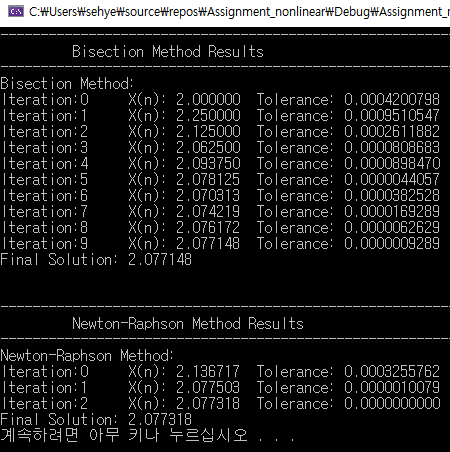
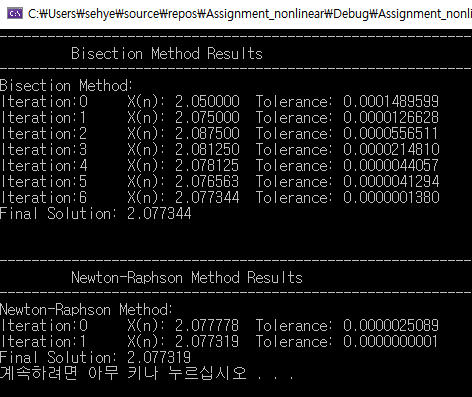


Figure 4.1. changing initial values (1) Figure 4.2. changing initial values (2)

마지막으로, tolerance의 관점에서 Newton-Raphson method와 Bisection method를 비교해보면, [Figure 4.1.], [Figure 4.2.]에서도 확인할 수 있듯이 Newton-Raphson method에서 오차가 더 작게 나타나며 해에 보다 더 근사하게 수렴하는 것을 확인해 볼 수 있다.

**Advanced Problem: Bonus Point [10pt]**

Q. You can combine bisection method and Newton-Raphson for a fail-safe routine. This hybrid algorithm can be designed as

Hybrid method 코드를 작성하기에 앞서, 왜 우리에게 hybrid 코드가 필요한지에 대해 생각해 보았다.

Newton-Raphson method의 경우에는 기울기가 너무 작을 경우, 무한대로 발산하는 경우가 생길 수 있다는 약점이 있다. 또한, 기울기가 너무 클 경우에는 해를 구하기까지 Iteration이 과도하게 커질 수 있다. 이럴 경우, Bisection method를 통해 기울기가 극단적으로 작거나 큰 경우를 회피하도록 하여 안정적으로 해를 구할 수 있도록 할 수 있다. 즉, Bisection method를 통해 Newton-Raphson method의 단점을 보완함과 동시에, 오차와 Iteration이 큰 Bisection method의 단점을 Newton-Raphson method를 통해 보완할 수 있게 된다.

- Set the bounds [a, b] on a root () as in the bisection method

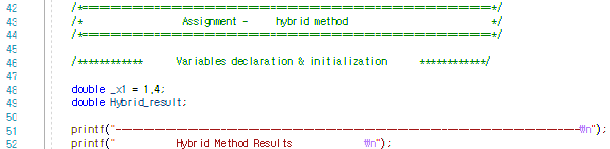


Figure 5.1. Variable Initializing at Hybrid Method

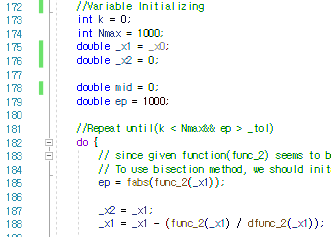


Figure 5.2. Variable Initializing at Hybrid Method

앞에서 확인했듯이, Newton-Raphson method가 Iteration 측면에서도 그렇고, tolerance의 측면에서도 Bisection method보다 뛰어나기에 Hybrid method에서 최종적으로 해를 찾아주는 과정은 Newton-Raphson method를 사용한다. 그렇기 때문에 변수 설정에 있어서, Newton-Raphson method와 동일하게 초기값 한 개만 입력을 받고, 두 번째 변수의 경우 Newton-Raphson method를 1회 실행하여 얻어진 접선의 절편의 값으로 설정해 줬다.

- For each iteration,

If Newton-Raphson gives the solution out of bounds, use bisection method for the next estimation of x(n+1)

If Newton-Raphson is not decreasing fast enough, or seems to be diverging, use bisection method for x(n+1).

Otherwise use Newton-Raphson

- Check your algorithm by solving for . Use x0=1.4 for the initial point. The exact solution is x=0.5.

- First, use Newton-Raphson method and analyze the solution. Use your hybrid method to find the solution. Compare these two solutions.

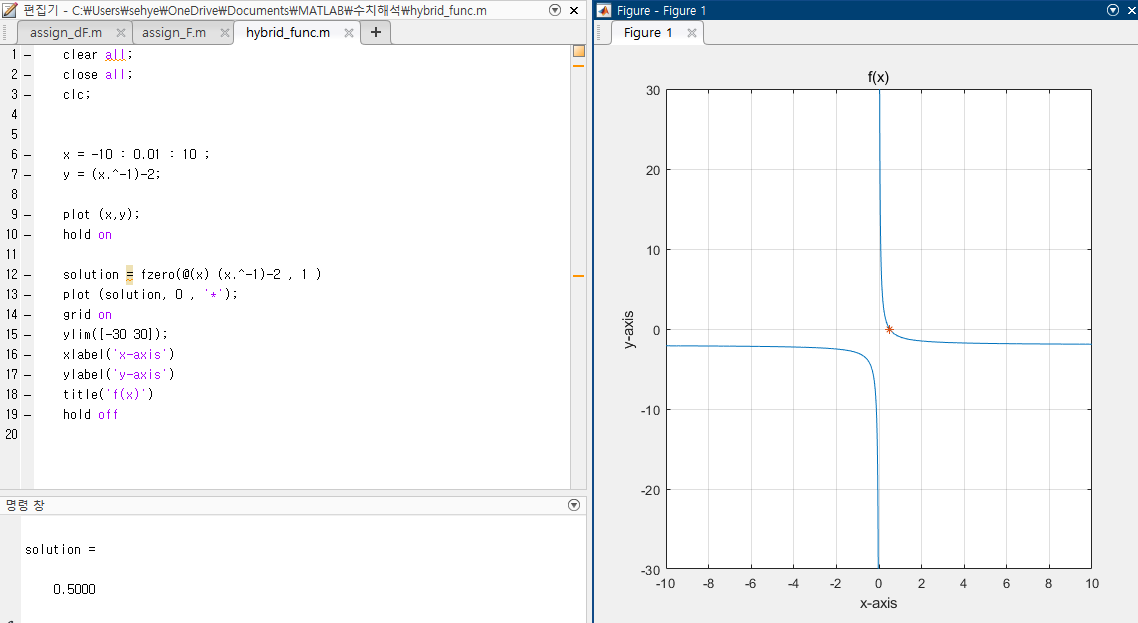
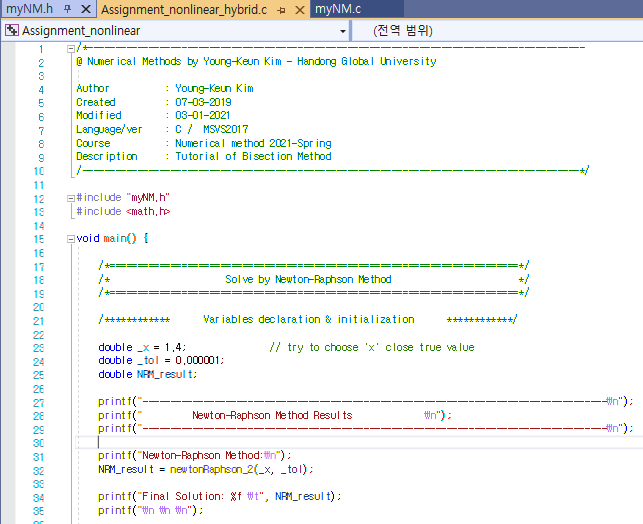


Figure 6.1. Graph of Hybrid Function

코드를 작성하기에 앞서, Matlab으로 의 그래프를 그려보았다. 지수함수의 특성상, 의 값이 약 1을 넘어가는 순간부터는 기울기가 급격히 작아지며 음의 방향으로 발산할 것 같이 보인다.



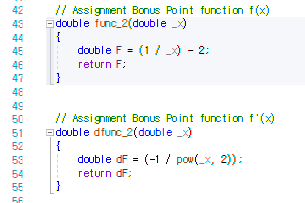


Figure 7.1. Main Code of Newton-Raphson Method for Hybrid Figure 7.2. Function Code of Hybrid

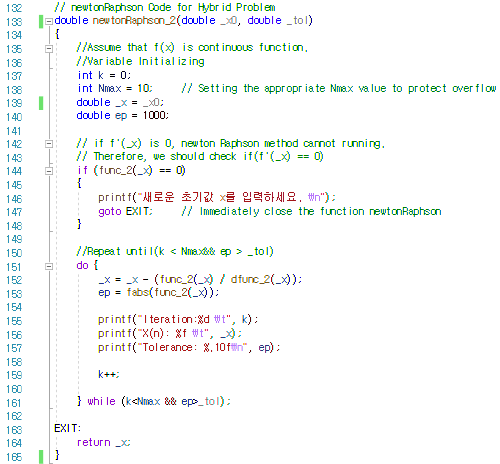


Figure 7.3. Code of Newton-Raphson Method for Hybrid

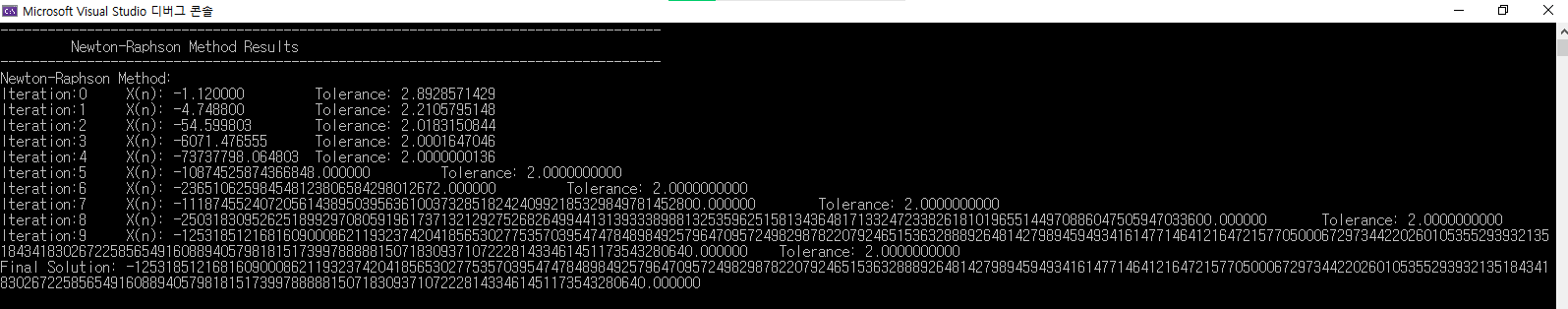


Figure 7.4. Result of Newton-Raphson method

실제로 Newton-Raphson method를 통해서 주어진 함수의 해를 구하려고 시도해보면, [Figure 7.4.]와 같이 음의 무한대로 발산하고 있음을 확인할 수 있으며, 해를 구할 수 없게 된다.

[Figure 5.2.]의 Hybrid Method에서의 변수 선언 방식을 살펴보면 초기값 1.4와 그 초기값에 대해 Newton-Raphson method를 실행하여 얻어진 좌표의 값이 두 번째 변수가 된다. 이후에는, 반복문을 통해 두 변수가 서로 초기화되는 과정을 반복한다.

예를 들어, Newton-Raphson method 1회 실행된 값을 보면 -1.12에 찍힌 것을 확인할 수 있다. 이 경우, 에는 초기값 1.4가 저장되었고 에는 -1.12가 저장된다. 그리고 반복문으로 다시 들어올 때, 에는 -1.12로 값이 초기화되고, 에는 -1.12에서 Newton-Raphson method를 적용하여 얻어진 새로운 좌표를 저장한다. 여기서 문제가 발생한다. [Figure 6.1.]의 그래프를 살펴보면 영역에서는 Newton-Raphson method를 쓰면 쓸수록 로 발산하게 되는 것이다. 즉, 이 부분에서 Bisection method를 통해 영역에 변수가 찍히는 경우를 막아야 목표로 하는 결과값 0.5를 얻을 수 있을 것이라 판단이 되었다.

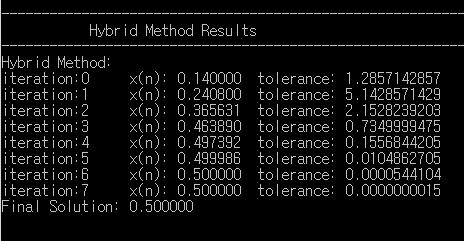
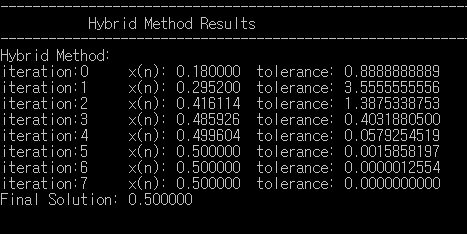
처음에는 이 부분을 미처 파악하지 못하고, 코드를 작성하였다. 문제에서 요구하는 대로 초기값 1.4를 줬을 때, 0.5에 수렴하도록 하는 Hybrid 코드를 작성하는 데는 성공하였지만, 특정 초기값에 대해서는 여전히 로 발산하는 결과값이 나왔다. Figure 설명에 있어서 Initial Value를 줄여서 I.V.로 표기하도록 한다.

Figure 8.1. Result of Failed-Hybrid at I.V. 1.4 Figure 8.2. Result of Failed-Hybrid at I.V. 0.9

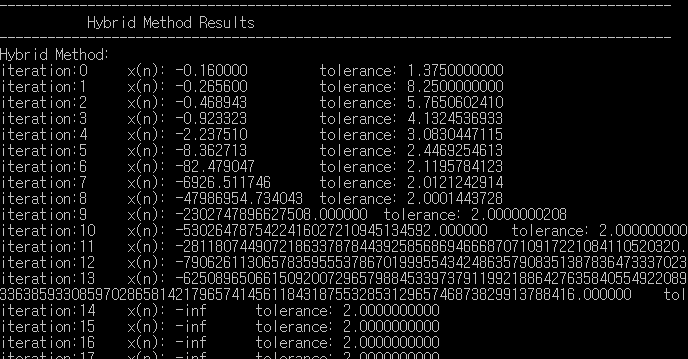


Figure 8.3. Result of Failed-Hybrid at I.V. 1.6

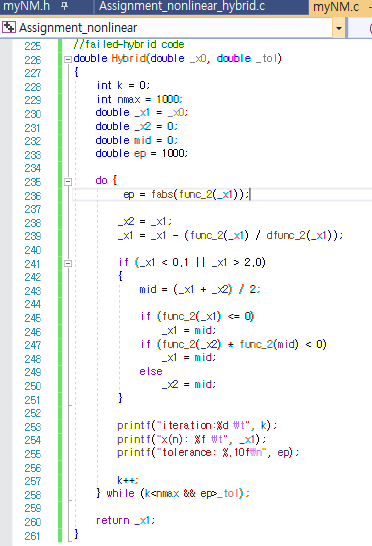


Figure 8.4. Code of Failed-Hybrid method

오류의 원인이 변수를 지정하는 부분에 있음을 디버깅을 통해 확인할 수 있었고, 앞서 설명했듯이 영역에 변수가 찍히는 경우를 막아야 목표로 하는 결과값 0.5를 얻을 수 있다고 판단했기에, 영역에 변수가 찍히는 것을 완전히 막을 수 있는 코드를 추가함으로써 어떠한 양수 범위의 입력에 대해서 항상 0.5라는 값을 구할 수 있는 Hybrid Method를 완성하였다. 코드에 대한 자세한 설명은 주석을 참고하도록 한다.

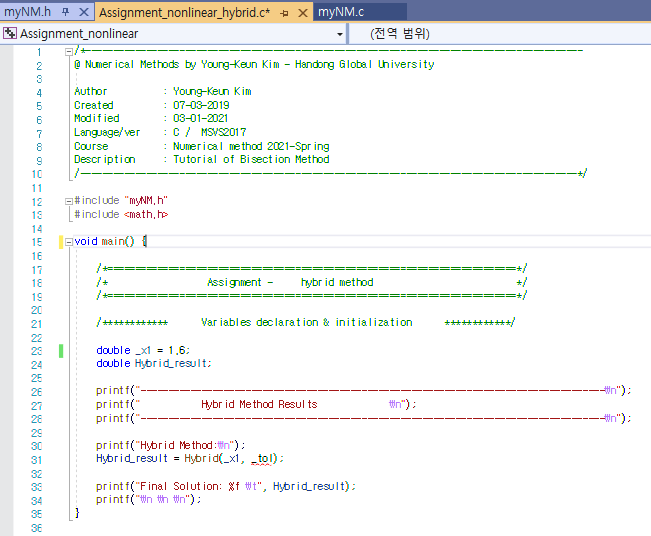


Figure 9.1. Main Code of Hybrid Method

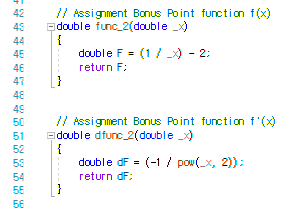


Figure 9.2. Function Code of Hybrid Method

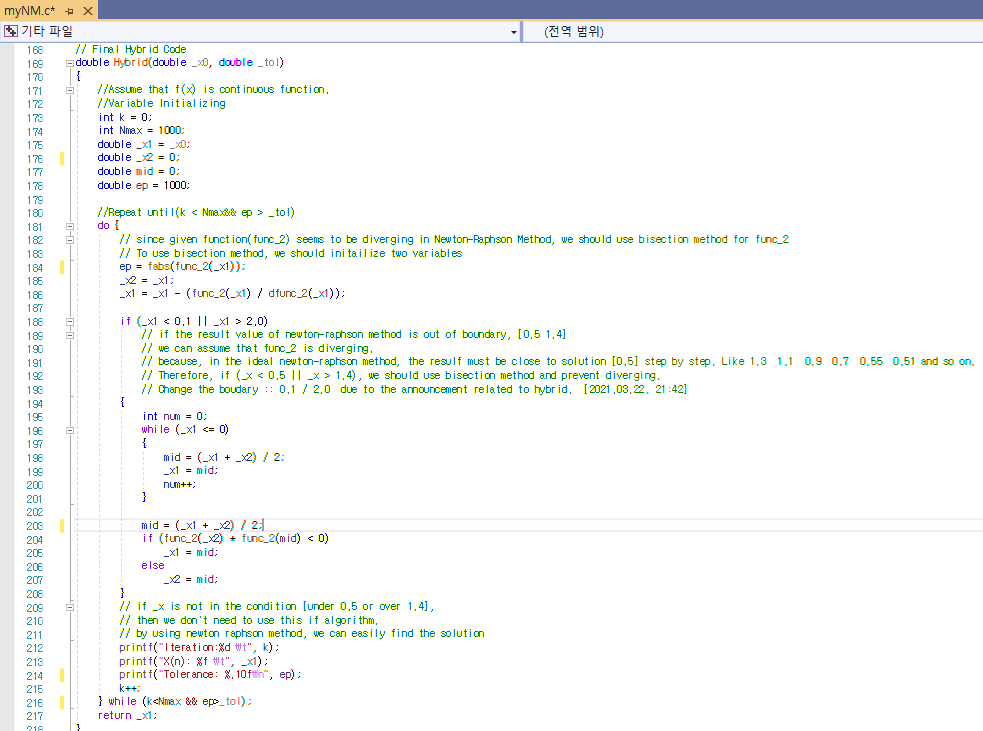


Figure 9.3. Code of Hybrid Method

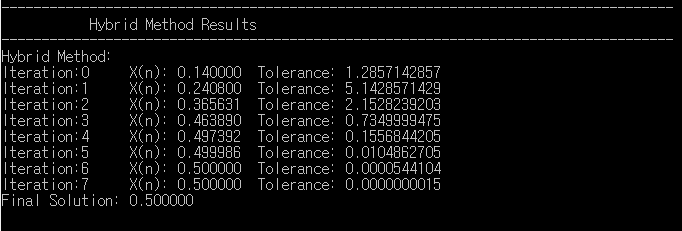


Figure 9.4. Result of Hybrid Method

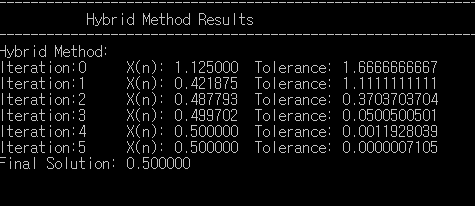


Figure 9.5. Result of Hybrid Method at I.V. 3

**Conclusion**

Newton-Raphson method를 이용하여 해를 찾는 과정이 Bisection method를 이용하여 해를 찾는 과정보다 더 빠르다고 볼 수 있다.

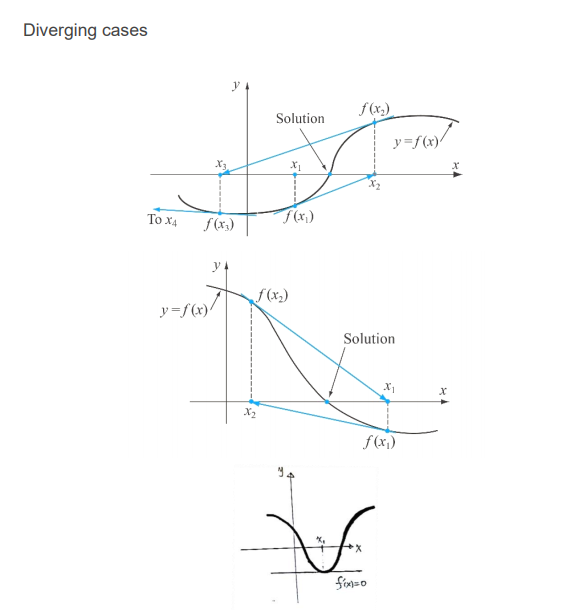


Figure 10.1. Diverging Cases in Newton-Raphson method

그러나, Newton-Raphson method는 [Figure 10.1.]에서 확인할 수 있듯이 무한대로 발산하거나 특정 Loop에 갇히는 경우가 발생하여 해를 구하는데 실패할 때가 발생할 수 있다. 이를 방지하기 위해 Newton-Raphson method에 Bisection method를 합쳐서 Hybrid method를 만들어 사용하는 것이다.