HW Goals:

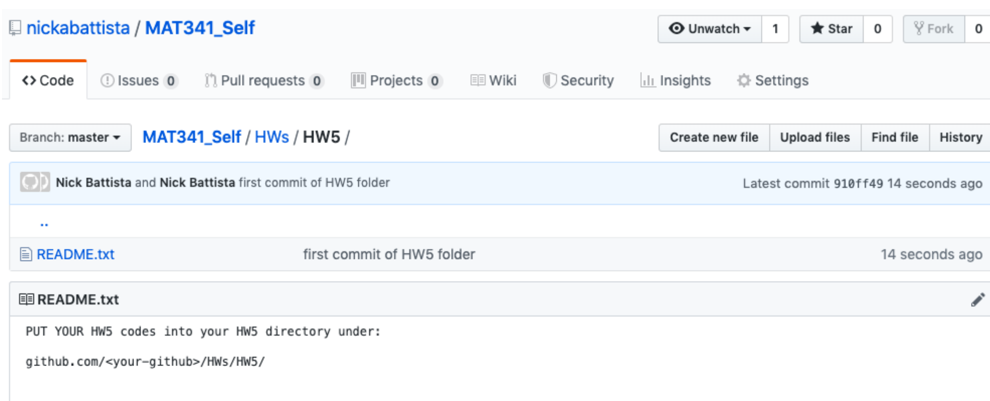
The purpose of this homework is to continue familiarizing ourselves with different algorithms for optimization in 1 or more dimensions - the ***Nelder-Mead Algorithm*** *and* ***Newton's Method Algorithms****.*

HW DUE:

Friday, November 8 by 11:59pm

WHERE:

Put all of the codes for this homework into your personal GitHub: `MAT341_Self`→`HWs`→`HW5`. That is, you will have to `add`, `commit` and `push` these scripts to your GitHub for me to pull, see below for example:



1. Write a script called `Nelder_Mead.m` that takes in one input, *tol*, and returns the *number of iterations*, N, necessary to achieve a certain the error tolerance, *tol*, e.g.,

$$\text{function } \mathbf{N} = \text{Nelder\_Mead(tol)}$$

   Within this script, implement the *Nelder-Mead algorithm* as discussed in class.

   Use your algorithm to find the a minimum of $f(x, y) = -(\sin(x) + \cos(y))$.

   Have it return (output) the number of iterations, N, it takes to achieve the specific error tolerance, *tol*, that is inputted.

Use the following 3 points as input:

(a) $(x, y) = (0.35, 2.8)$

(b) $(x, y) = (-0.25, 0.3)$

(c) $(x, y) = (1.5, 0.5)$

Use the *err* **for the** `while-loop` **defined as follows:**

$$err = \left| f(\vec{x}_1) - f(\vec{x}_3) \right| = \left| f(x_1, y_1) - f(x_3, y_3) \right|.$$

Note that local minima for this function occur when: $\sin(x) = 1$ **and** $\cos(y) = 1$.

Answer the following and write your answers as comments at the bottom of the script:

(a) Using the initial points above, how many iterations does it take to achieve $1e - 8$ accuracy?

(b) For the initial points listed above, what point does it appear to converge to? What is the true (x, y) point where this minima is located? Does it look like $1e - 8$ accuracy? Why or why not?

(c) Change the second initial point from $(-0.25, 0.3) \Rightarrow (1.75, 0.10)$. How many iterations did it take to achieve $1e - 8$ accuracy? Which minima did it locate? Changing the second initial value to $(1.75, 0.10)$ actually puts this point closer to the local minima you found in part(a), comment on what you think happened here with the number of iterations compared to (a).

(d) Change the second initial point and third initial points to $(-0.25, 0.3) \Rightarrow (4, 4)$ and $(1.5, 0.5) \Rightarrow (4.5, 4.5)$, respectively. What minima does the algorithm find?

2. Write a script called `Newtons_1D_Opt.m` that takes in one input, *tol* and returns the *number of iterations*, N, necessary to achieve a certain the error tolerance, *tol*, e.g.,

$$\text{function } \mathbf{N} = \text{Newtons\_1D\_Opt(tol)}$$

Within this script, implement *Newton's Method* to find a **local minimum** (NOT A ROOT) with initial guess $x_1 = 0.25$.

Use your algorithm to find the minimum of $f(x) = 0.5 - xe^{-x^2}$ within the interval $[0, 2]$. Have it return (output) the number of iterations, N, it takes to achieve the specific error tolerance, *tol*, that is inputted.

Answer the following and write your answers as comments at the bottom of the script:

(a) How many iterations does it take to achieve $1e{-8}$ accuracy with the initial guesses as described above?

(b) Change the initial guess from $x_1 = 0.25$ to $x_1 = 1.5$. What happened? (While Newton's Method is known for its fast convergence rate, it all depends on what initial guess you provide!)

Change the initial guess x_1 back to $x_1 = 0.25$ for Problem 3.

3. For this problem, copy and paste your previous two $1D$ optimization codes (`golden_Search.m` and `successive_Parabolic_Interpolation.m`) as well as your `vary_Error_Tolerances_To_Compare.m` script into the **HW5 folder**.

   Modify the script called `vary_Error_Tolerances_To_Compare.m` that takes no inputs (nor returns anything), e.g.,

   ```
   function vary_Error_Tolerances_To_Compare()
   ```

   to call the previous two algorithms you've made (`golden_Search` and `successive_Parabolic_Interpolation`) and now your `Newtons_1D_Opt.m` code to run for a variety of error tolerances. Recall that each of those scripts outputs the number of iterations, N, it takes to achieve a particular error tolerance. Run all of these algorithms for each error tolerance in the following vector:

   $$errTolVec = [1e{-1}\ 1e{-2}\ 1e{-3}\ 1e{-4}\ 1e{-5}\ 1e{-6}\ 1e{-7}\ 1e{-8}\ 1e{-9}\ 1e{-10}\ 1e{-11}\ 1e{-12}],$$

   e.g., you will loop over every component of the vector $errTolVec$. Save each algorithm's number of iterations into a storage vector. Use a different vector for each algorithm.

   Make **two** plots that illustrate each algorithms number of iterations, N vs. specific error tolerances, *tol*. For one plot, use logarithmic axis in the horizontal direction only (e.g., `semilogx`) while in the other use logarithmic axis for both (e.g., `loglog`). On each figure, plot all sets of data. That is, you want to see a comparison of each algorithm's number of iterations vs. error tolerances.

   Make sure to use the *# of Iterations, N* you calculate as the dependent variable (vertical axis) and the *error tolerance, tol* as the independent variable (horizontal axis).

   Make sure to:

   (a) Label the axes

   (b) Make the Golden Search's line color blue, Successive Parabolic Interpolation's line color red, and the **Newton's Method line in black**.

   (c) Change the thickness of the line (e.g., line width) to 5.

   (d) Make a figure legend, e.g., `legend('Golden Search','Succ. Para. Interp.', 'Newton Method')`. (if you listed the Golden Search first in the figure)

(e) Recall to plot multiple sets of data on the same plot, use the `hold on` command after the plotting statement.

(f) You can explicitly make multiple figures, by using the command `figure(1)` and `figure(2)` before anything related to its plot.

Answer the following and write your answers as comments at the bottom of the script:

(a) Which algorithm appears converges faster to the minimum for less accurate tolerances?

(b) What happens when you increase the accuracy threshold? Does that algorithm always converge quicker?

(c) What could change the convergence rates for these algorithms? (e.g., what did they depend on to get started?)

4. Write a script called Newtons_2D_Opt.m that takes in one input, *tol* and returns the *number of iterations*, N, necessary to achieve a certain the error tolerance, *tol*, e.g.,

$$\text{function } \mathbf{N} = \text{Newtons\_2D\_Opt(tol)}$$

Within this script, implement the *Multivariable Newton's Method* to find a **local minimum** (NOT A ROOT) with initial guess $\vec{x}_1 = (x_1, y_1) = (-0.25, 0.25)$.

Use your algorithm to find a minimum of $f(x,y) = -(\sin(x) + \cos(y))$. Have it return (output) the number of iterations, N, it takes to achieve the specific error tolerance, *tol*, that is inputted.

Use a tolerance of ***tol***$=1e-8$ and **define your error** to be the l^2**-error**, e.g.,

$$err = \sqrt{(\vec{x}_{n+1} - \vec{x}_n)^T(\vec{x}_{n+1} - \vec{x}_n)}.$$

Answer the following and write your answers as comments at the bottom of the script:

(a) With the initial guess above and an error tolerance of $1e-8$, how many iterations does it take to find a minima? Which minima did it find?

(b) Change your initial guess to $\vec{x}_1 = (x_1, y_1) = (-4.5, 4.5)$. Which minima did it find? How many iterations did it take?