HW Goals:

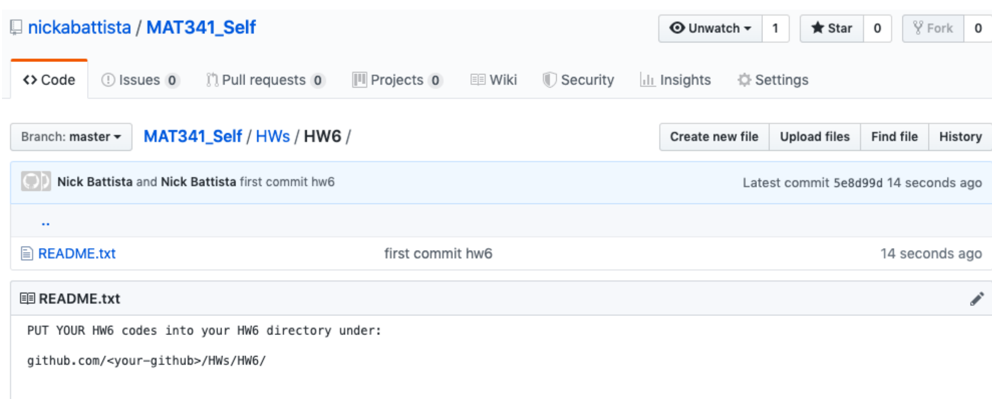
The purpose of this homework is to continue familiarizing ourselves with the Gradient Descent Method for minimizing a multivalued function. In particular, we will explore how different methods to change the step-size, γ, affects the number of iterations it takes the algorithm to converge to within a desired error tolerance.

HW DUE:

Tuesday November 19 by 11:59pm

WHERE:

Put all of the codes for this homework into your personal GitHub: `MAT341_Self`→`HWs`→`HW6`. 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 `Gradient_Descent_1.m` that takes in two inputs, *tol* and *gamma*, and returns the *number of iterations*, N, necessary to achieve a certain the error tolerance, *tol*, using a specific ***fixed*** step-size *gamma*, e.g.,

   function **N** = Gradient_Descent_1(tol,gamma)

   Within this script, implement the *Gradient Descent 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*, and step-size, *gamma*, that are inputted.

Use the following point as the initial guess:

$$\mathbf{x}_1 = (x_1, y_1) = (1.0, 1.5)$$

Define the err **for the** `while-loop` **to be the** $l^2 - Norm$**:**

$$err = \left|\left|\mathbf{x}_{n+1} - \mathbf{x}_n\right|\right|_2 = \sqrt{(\mathbf{x}_{n+1} - \mathbf{x}_n)^T(\mathbf{x}_{n+1} - \mathbf{x}_n)}$$

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 point above, how many iterations does it take to achieve $1e-10$ accuracy using $gamma = 0.5$?

(b) Using the initial point above, how many iterations does it take to achieve $1e-10$ accuracy using $gamma = 0.9$?

(c) Using the initial point above, how many iterations does it take to achieve $1e-10$ accuracy using $gamma = 1.5$?

2. Write a script called vary_StepSize_Gamma_To_Optimize.m that will run your Gradient_Descent_1 code from Problem 1 for a variety of step-sizes, *gamma*, to find which gamma seems to minimize the total number of iterations needed to achieve $1e-10$ accuracy, e.g.,

```
function vary_StepSize_Gamma_To_Optimize()
```

Recall that Gradient_Descent_1 outputs the number of iterations, N, it takes to achieve a particular error tolerance, *tol* with given fixed step-size, *gamma*. Using this code, find what value of *gamma* seems to minimize the total number of iterations necessary.

Do the following:

(a) Run your code for a variety of *gamma*, store the corresponding number of iterations in a vector

(b) Only what the value of *gamma* that seems to minimize the number of iterations to within a tolerance of 0.02.

(c) Make a plot of **# of iterations** *vs. gamma* **(step-size)**. Be sure to:
   - *# of Iterations, N* you calculate as the dependent variable (vertical axis) and the *gamma (step-size)* as the independent variable (horizontal axis)
   - Label the axes
   - Change the linecolor to blue.
   - Change the thickness of the line (e.g., line width) to 5.
   - Add a figure legend; call this `Fixed Step`.
   - Use a logarithmic axis for the x-axis

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

(a) What does the "best" step-size, *gamma*, seem to be for this particular function, $f(x,y) = -(\sin(x) + \cos(y))$?

Hints:

- Do not write an optimization scheme for this problem
- Approach this problem the way that we've done before when varying *error tolerances*, etc.

3. Write a script called Gradient_Descent_2.m that takes in one inputs, *tol* and returns the *number of iterations*, N, necessary to achieve a certain the error tolerance, *tol*, using a specific ***fixed*** step-size *gamma*, e.g.,

---

function $\mathbf{N}$ = Gradient_Descent_2(tol)

Within this script, implement the *Gradient Descent algorithm* using the **Barzilai-Borwein step-size** for *gamma*, 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 point as the initial guess:

$$\mathbf{x}_1 = (x_1, y_1) = (1.0, 1.5)$$

Use the following *gamma* value as an initial *gamma*:

$$gamma = 0.5$$

Recall that the **Barzilai-Borwein step-size** requires two successive approximations, $\mathbf{x}_{n+1}, \mathbf{x}_n$, and then calculates

$$\gamma_k = \frac{\Big(\mathbf{x}_{n+1} - \mathbf{x}_n\Big)^T \Big(\nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n)\Big)}{\Big(\nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n)\Big)^T \Big(\nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n)\Big)}$$

Define the *err* for the `while-loop` to be the $l^2 - Norm$:

$$err = \Big|\Big|\mathbf{x}_{n+1} - \mathbf{x}_n\Big|\Big|_2 = \sqrt{(\mathbf{x}_{n+1} - \mathbf{x}_n)^T (\mathbf{x}_{n+1} - \mathbf{x}_n)}$$

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 point above how many iterations does it take to achieve $1e-6$ accuracy using the Barzilai-Borwein step-size?

(b) Using the initial point above, how many iterations does it take to achieve $1e-10$ accuracy using the Barzilai-Borwein step-size?

4. Create a script called `vary_Error_Tolerances_To_Compare.m` that takes no inputs (nor returns anything), e.g.,

    function vary_Error_Tolerances_To_Compare()

   to compare the iteration counts for both versions of the Gradient Descent script. Recall that both `Gradient_Descent_1` and `Gradient_Descent_2` output the number of iterations, N, it takes to achieve a particular error tolerance, *tol*. Run this algorithm for each error tolerance in the following vector:

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

   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.

   Recall that for `Gradient_Descent_1` you also need to specify the what fixed step-size you want. **Use the "best" step-size you found in Problem 2.**

   Make **two** plots that illustrates the 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 both 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 Fixed Step-Size line color blue, and the Barzilai-Borwein step-size red.
   (c) Change the thickness of the line (e.g., line width) to 5.
   (d) Make a figure legend, e.g., `legend('Fixed Step','Barzilai-Borwein')`.
   (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 to converge faster to the minimum?

(b) What happens if you change the **fixed step-size** to $gamma = 0.5$? Which algorithm converges quicker?

(c) What is an advantage of using the **Barzilai-Borwein** step-size?

(d) If you were to modify your code to minimize a different function, $f(x, y)$, which step-size method would you choose to use and why?