AMATH 301 – Autumn 2019
Homework 5

Due: 7:00pm, November 6, 2019.

Instructions for submitting:

- Scorelator problems are submitted as a MATLAB script (`.m` file). You should **NOT** upload your `.dat` files. The `.dat` files should be created by your script. You have 5 attempts on Scorelator.
- Writeup problems are submitted to Gradescope as a single `.pdf` file that contains text and plots. Put the problems in order and label each writeup problem. When you submit, **you must indicate which problem is which on Gradescope**. **All code you used for this part of the assignment should be included either at the end of the problem or at the end of your .pdf file.**

Scorelator problems

Note that the Scorelator portion is shorter than the Gradescope portion of the assignment this week: there are only 8 .dat files to be submitted this week. Please plan accordingly.

1. A seminal breakthrough in HIV treatment was the use of multiple-drug combination therapy. The idea behind the therapy is that multiple drugs, introduced to the body at specific times, fight the virus off better than one drug alone. A key step in the development of multiple-drug therapy is determining the best time to take the next drug. One way this is done is to take the next drug when the previous drug is at its maximum in the body (and beginning to decrease from there). Drug absorption and elimination in the body is modeled using *compartmental models*.

   Consider a model whose solution is given by

   $$x(t) = \frac{10}{9}\left(e^{-t/10} - e^{-t}\right), \tag{1}$$

   where $x(t)$ is the amount of the drug in the body at time t. Assume that we want to administer the next drug when $x(t)$ is at its maximum. We call this time $t_{\max}$ in what follows. Recall that we have only talked about **minimizing** functions so you will have to restate the problem as a minimization problem.

(a) Use Golden Section search to find $t_{\max}$. Use an initial interval of $[0, 10]$ and stop when $b - a \leq 10^{-8}$. Create a 1×3 row vector that contains the number of iterations in the first component, $t_{\max}$ in the second component, and $x(t_{\max})$ in the third component. Save the vector in `A1.dat`.

(b) Use `fminbnd` with the same initial interval as above to find $t_{\max}$. Create a 1×2 row vector that contains the $t_{\max}$ in the first component and $x(t_{\max})$ in the second component. Save this vector in `A2.dat`.

2. The function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

is known as *Himmelblau's function* which is a function designed for testing performance of optimization algorithms (you can see more examples here). You may want to do Gradescope Problem 1 before this problem to help with visualizing this function. Himmelblau's function has multiple *local* minima. In this problem we will find one of those local minima.

(a) Create a MATLAB function that computes f using only one input: a column vector `[x; y]`. Use `fminsearch` with an initial guess of `[3;4]` to find the column vector `[x; y]` that minimizes f and save it in the file `A3.dat`.

(b) Create a MATLAB function (with only one input) that calculates the gradient $\nabla f(x, y)$. The gradient is given by the following formula

$$\nabla f(x, y) = \begin{pmatrix} 4x^3 - 42x + 4xy + 2y^2 - 14 \\ 4y^3 - 26y + 4xy + 2x^2 - 22 \end{pmatrix}. \tag{2}$$

Recall that the gradient of f should be zero at a local minimum. Calculate $\nabla f(x, y)$ at the x and y values found in part (a) and save the norm to `A4.dat`. In other words, if you found $(x, y) = (x_1, y_1)$ above, save

$$\|\nabla f(x_1, y_1)\|_2$$

in `A4.dat`.

(c) Recall from class that we wrote the following code to perform one step of gradient descent.

```
p = [6; 4]; % Choose an initial guess
grad = fgrad(p); % Find which direction to go
phi = @ (t) p - t*grad; % Define the path
f_of_phi = @ (t) f(phi(t)); % Create a function of "heights along path"
tmin = fminbnd(f_of_phi,0,1); % Find time it takes to reach min height
p = phi(tmin); % Find the point on the path and update your guess
```

Adapt the code so that it performs up to 10000 iterations of gradient descent and stops if the infinity norm of the vector `grad` is less than a tolerance of 10^{-5}.

(d) Now use your gradient descent code to find the column vector `[x; y]` that minimizes the function $f(x, y)$. Use an initial guess of `[3; 4]`. Save the final guess in `A5.dat` and the number of iterations in `A6.dat`. The initial guess does not count as an iteration.

3. Recall that there are two different possible outputs when solving a minimization problem. First we can find the value F such that $f(x) \geq F$ for all x. Second, we can find the value y such that $f(x) \geq f(y)$ for all $x \neq y$. The first of these is called the *minimum* of $f(x)$ and is denoted $F = \min_x(f(x))$. The second is called the *argmin* of $f(x)$ and is denoted $y = \text{argmin}_x(f)$. In other words, the argmin is the value of x at which the minimum occurs, $F = f(y)$. The *max* and *argmax* are defined similarly and have to do with the maximum of f.

   Which of the following is true? Save your answer in `A7.dat`. **If more than one is true, save a string containing both letters with the earliest letter in the alphabet coming first. For example, if the correct answers are G and J, use** `a7 = 'GJ'; save('A7.dat','a7','-ascii')`.

   A $\min_x(f(x)) = -\max_x(-f(x))$

   B $\text{argmin}_x(f(x)) = -\text{argmax}_x(f(x))$

   C $\min_x(f(x)) = \max_x(-f(x))$

   D $\text{argmin}_x(f(x)) = -\text{argmax}_x(-f(x))$

   E $\text{argmin}_x(f(x)) = \text{argmax}_x(-f(x))$

   F $\min_x(f(x)) = -\max_x(f(x))$

4. Which of the following is true about the connection between optimization problems and root-finding algorithms? Save your answer in `A8.dat`.

   A Successive Parabolic Interpolation is most like the Bisection method.

   B Golden Section Search is most like Newton's method, it always converges quickly.

   C Successive Parabolic Interpolation is most like Newton's method, it always converges.

   D Gradient Descent is most like the Bisection method.

   E Gradient Descent is most like Newton's method.

   F Successive Parabolic Interpolation is most like Newton's method, it does not always converge, but when it does it does so quickly.

Gradescope problems

1. Recall Himmelblau's function defined in Scorelator Problem 2. In this problem you will create plots of this function to aid in visualization.

   You only need to turn in the two plots that are created: the surface plot in part (a) and the contourplot with the markers on it created in parts (b)-(d).

   (a) First we will create a surface plot using `surf`.

   i. Define the function $f(x, y)$ in MATLAB as a function of two variables. Remember to use component-wise operations.

   ii. Use the `meshgrid` function to create a mesh with 20 equally spaced points from -5 to 5 in the x direction and 20 equally spaced points from -5 to 5 in the y direction.

   iii. Use the `surf` command to plot $f(x, y)$ versus x and y.

   iv. Notice that the plot does not look good with default settings. You should

   - Change the limits on the z-axis from 0 to 200 (see `zlim`).
   - Set the limits on the x- and y-axes (see `xlim` and `ylim`) to match the range of x and y values.
   - Set the limits of the *color axis* to be the same as the z-axis limits (see `caxis`).
   - Add a *color bar* that indicates what values the colors correspond to (see `colorbar`).
   - Set the viewing angle of the plot using `view(30,30)`.
   - Use `daspect[1 1 70]` to change the aspect ratio of your plot so that 1 x and y unit corresponds to 70 z units.
   - Use `colormap('jet')` to change the coloring of the plot.
   - Label your x, y, and $f(x, y)$ axes.

   (b) Next you will create a *contour plot* of the function.

   i. Use the same function f defined in part (a).

   ii. Define a meshgrid with 100 linearly spaced points between -5 and 5 along the x-axis and 100 linearly spaced points between -5 and 5 along the y-axis.

   iii. Use the `contourf` command to plot the contours of $f(x, y)$ vs x and y. Your plot should have 22 logarithmically spaced contours between 10^{-1} and 10^3. Look up `logspace` to see how this is accomplished.

   iv. Change the color axis on the plot (`caxis`) to go from 0 to 200.

   v. Change the colormap to `jet` as was done in part (a).

   vi. Label your x and y axes.

   (c) Next you will use `fminsearch` repeatedly to find all of the local minima (the multiple local minimums) of f. Use the plots and appropriate initial guesses (based on what you see in the plots you created in the previous problems) to find all 4 local minima.

(d) Now you will add visual markers to the plots to show where the minima are, according to the different algorithms used. You should add the following to your contour plot (part (b) above).

i. Plot the local minimum found in Scorelator Problem 2 (a) with `fminsearch` as an open red circle. You should use `'MarkerSize',15` and `'Linewidth',3` to make this circle more visible.

ii. Plot the local minimum you found with gradient descent in Scorelator Problem 2 (d) as a green square. You should use `'MarkerSize',15` to make this marker more visible.

iii. Plot the local minima found in part (c) with yellow stars. You should use `'MarkerSize',15` to make these markers more visible.

2. This problem is about comparing the two different ways to do gradient descent. You should re-familiarize yourself with Scorelator Problem 2 and the details therein before moving on.

Instead of calculating `tmin` with `fminbnd` for every step of gradient descent, you can use a fixed value of `t`, which we will call `tstep`, to calculate the next guess in gradient descent. In order to do this, you can replace the lines of code

```
phi = @ (t) p - t*grad; % Define the path
f_of_phi = @ (t) f(phi(t)); % Create a function of "heights along path"
tmin = fminbnd(f_of_phi,0,1); % Find time it takes to reach min height
p = phi(tmin); % Find the point on the path and update your guess
```

with the single line of code

```
p = p - tstep*grad;
```

where the variable `tstep` must be defined previously in the code.

In each of the following solves using gradient descent, use an initial guess of `[3; 4]`, a tolerance of 10^{-5} and a maximum of 10000 iterations to try to find the column vector `[x; y]` that minimizes the function $f(x, y)$.

You do not need to answer these questions individually, you only need to turn in the table created in part (e) and the comments in part (f).

(a) Use gradient descent with `fminbnd` (as you did on Scorelator Problem 2). Use tic and toc to determine how long it takes the algorithm to complete and save the number of iterations required to converge, if it does converge.

(b) Use gradient descent with `tstep = 0.01` (described above). Use tic and toc to determine how long it takes the algorithm to complete and save the number of iterations required to converge, if it does converge.

(c) Use gradient descent with `tstep = 0.02` (described above). Use tic and toc to determine how long it takes the algorithm to complete and save the number of iterations required to converge, if it does converge.

(d) Use gradient descent with `tstep = 0.025` (described above). Use tic and toc to determine how long it takes the algorithm to complete and save the number of iterations required to converge, if it does converge.

(e) Record your results in the following table

	Number Iterations	Time	Converged (Yes/No)
`tstep=0.01`			
`tstep=0.02`			
`tstep=0.025`			
`fminbnd`			

(f) Comment on the results in the table.

- Did the Gradient Descent method always converge? For which settings/step sizes (using `fminbnd` vs. `tstep`) did it not converge? In the cases for which it did not converge, explain why it did not converge.
- For which settings (`fminbnd` vs. `tstep` and different step sizes) did the method converge the quickest?
- Which converged in the fewest number of iterations?
- Is your answer to the above two questions (fastest and fewest iterations) the same? If so, why? If not, why? What slows down the algorithm?