Homework 6

Problem 1: Rosenbrock's Banana

The function

$$f(x_1, x_2) = (a - x_1)^2 + b(x_2 - x_1^2)^2$$

is somewhat whimsically known as Rosenbrock's banana function. It is often used as a benchmarking test for optimization algorithms because it is easy to find the minimum analytically but often very tricky numerically. In this problem we will explore a method called *gradient descent* to find this minimum. Throughout the problem, we will use the values $a = 1$ and $b = 10$. You can plot this function using the following code:

```
x = -3:0.1:3;
y = -10:0.2:10;
[X, Y] = meshgrid(x, y);
Z = (1 - X).^2 + b*(Y - X.^2).^2;
surf(X, Y, Z)
```

(a) Write a Matlab function that computes f using only one input. (That is, your function should only use one variable.) Use this function to calculate $f(-1, 8)$ and save the result in `A1.dat`.

(b) Use `fminsearch` to find the minimum of f and save it as a 2×1 vector $[x_1; x_2]$ in the file `A2.dat`. Use an initial guess of $[-1;\ 8]$.

Things to think about: Can you confirm the minimum of f analytically? (This should be easy.) Try timing `fminsearch`. How long does it take to find this minimum?

(c) Find a formula for the gradient $\nabla f(x_1, x_2)$. Recall from calculus that the gradient is defined as the 2×1 vector

$$\nabla f(x_1, x_2) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2) \\ \frac{\partial f}{\partial x_2}(x_1, x_2) \end{pmatrix}.$$

Calculate $\nabla f(-1, 8)$ and save the resulting 2×1 vector in `A3.dat`.

Remember from calculus that the gradient of f is zero at a minimum. Find the infinity norm of $\nabla f(-1, 8)$ and confirm that it is not particularly close to zero. Save the result in `A4.dat`

(d) The vector $-\nabla f(x_1, x_2)$ points from $[x_1;\ x_2]$ in the direction where f decreases as quickly as possible. (In other words, it points in the direction of steepest descent.) Write a function in Matlab defined by

$$\phi(t) = \begin{pmatrix} -1 \\ 8 \end{pmatrix} - t\nabla f(-1, 8) = \begin{pmatrix} -1 - t\frac{\partial f}{\partial x_1}(-1, 8) \\ 8 - t\frac{\partial f}{\partial x_2}(-1, 8) \end{pmatrix}.$$

Calculate $\phi(0.1)$ and save the resulting 2×1 vector in `A5.dat`. Calculate $f(\phi(0.1))$ and save the resulting number in `A6.dat`.

(e) Notice that the function $f(\phi(t))$ is decreasing at $t = 0$, but that at some point it starts to increase again. This means that $f(\phi(t))$ has a minimum at some $t > 0$. We could implement the golden section search or Newton's method to find this minimum, but Matlab has a builtin function called `fminbnd` that does this for us. (Technically, it uses a combination of golden section search and parabolic interpolation, which is described in the video lectures.) Use `fminbnd` to minimize $f(\phi(t))$ on the interval $0 < t < 0.1$. (You should look at the documentation for `fminbnd` to see how to call it.) Save the resulting time t in `A7.dat` and the resulting 2×1 vector $\phi(t)$ in `A8.dat`.

We have just completed one step in the iterative method *gradient descent*. We made an initial guess $[x_1; x_2] = [-1;\ 8]$. Since the gradient of f at this point was not zero, we knew we needed a new guess, so we found the direction of steepest descent $-\nabla f(x_1, x_2)$, then found how far we needed to travel in this direction to minimize $f(\phi(t))$. The resulting point $\phi(t)$ is our new guess.

Things to think about: The video on gradient descent describes how to find the minimum of ϕ (for a different function f) analytically? Can you do that here? Try implementing Newton's method for this problem. Does it run faster? If so, is the boost in speed worth it?

Try plotting your initial guess and this point on the surface (with the `plot3` command). Confirm that the gradient of f gives the direction from the initial guess to this new point. You should see that $\phi(t)$ is in the bottom of a (banana shaped) valley. Does this mean that we are close to the minimum of f? Can you find a better direction than the gradient so that we would get closer to the minimum? Can you see a way to generalize that idea so that it works for any initial guess? (The last question is *very* difficult.)

(f) Finally, we will repeat the process from steps (c) - (e) over and over again to find the minimum of f. You can use the following template as a guide:

```
% Make initial guess

while % (infinity norm of gradient is >= tolerance)
    % Make the function phi(t) = x - t*grad f(x)
    % Find the minimum of f(phi(t)) between t=0 and t=0.1
    % Make new guess = phi(t)
end
```

You should use a tolerance of 10^{-4} and an initial guess of $[-1;\ 8]$. Save your final guess (which should be a 2×1 vector) in `A9.dat`. Save the total number of steps in `A10.dat`. (The initial guess of $[-1;\ 8]$ does not count as a step, but every other new guess does.)

Things to think about: Time this process. How does it compare to `fminsearch`? The function `fminbnd` is performing (mostly) golden section search, so it also has a loop. If this loop only performed golden section search (which is the worst-case scenario), how many steps would it take every time we called `fminbnd`? How many steps does this inner loop take in all of part (f)? This is one of the worst possible functions for gradient descent. Try some other functions $f(x, y)$ and see how many steps it takes to find their minima. In particular, try $f(x, y) = x^2+y^2$.

The slow part of this process is where we find the minimum of $f(\phi(t))$. What happens if you just choose a constant t value instead of recalculating this minimum at every step? Be sure to try values that are too large and values that are too small. Try this with the original f as well as $f(x, y) = x^2 + y^2$.