AMATH 301
Homework 7
Due: Saturday, February 23, 2019

Beale Function

The function,

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

is called the Beale function. It is often used as a benchmarking test for optimization algorithms.

Plotting the Beale function

The Beale function is fairly difficult to interpret without a visual aid, so we will begin by plotting the function to get an idea of what it looks like. We will make two types of plots: a surface plot and a contour plot.

- A *surface plot* depicts the function as a sheet whose height indicates the value of the function $f(x, y)$ at each point (x, y).
- A *contour plot* shows curves (contours) on which the function is a constant value, $f(x, y) = c$ for several values of c. Contour plots are like topographic maps.

Problem 1 (Writeup).
Use the `surf` function to create a surface plot of the Beale function.

To do this, you will need to define the function $f(x, y)$ in MATLAB as a function of two variables. Remember to use component-wise operations. You will also need the `meshgrid` function, which is demonstrated in the video lecture on gradient descent, and which we will talk about in class. Plot within the domain $-5 \le x \le 5$ and $-5 \le y \le 5$. Use 31 linearly-spaced points in both directions.

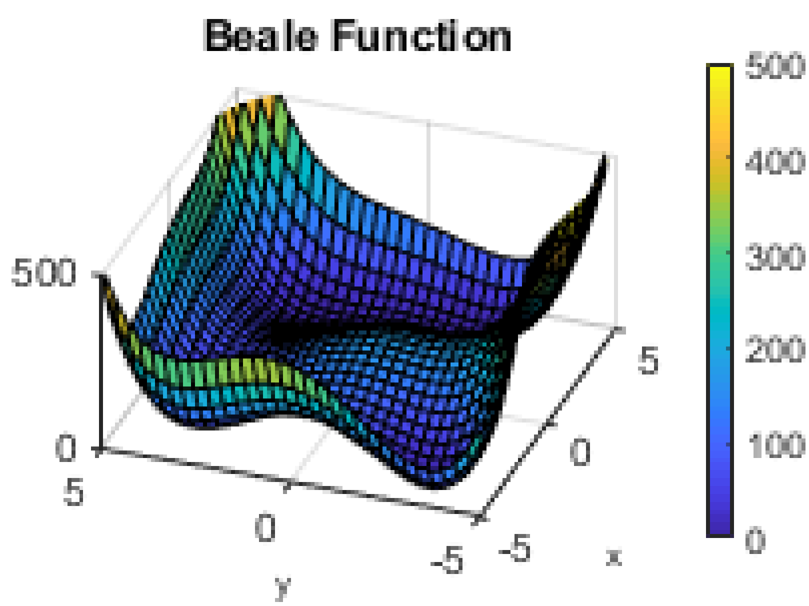

Figure 1: A low-resolution version of `beale_surf.png`.

The plot will not look very good with default settings. You should:

- Set the limits on the z-axis so that $0 \leq z \leq 500$.
- Set the limits on the x- and y-axis to match the domain we are plotting on (it may not match by default).
- Set the limits of the *color axis* to match the z-axis (look at the `caxis` command).
- Add a *color bar* that indicates what values the colors correspond to.
- Set the **view**ing angle of the plot to be `AZ = -70` and `EL = 30` – these are essentially angles of spherical coordinates.
- Use the `daspect` function to set the aspect ratio of the data shown in the plot to `[1 1 100]`. This will make one unit in the x-direction take up the same length as one unit in the y-direction and one hundred units in the z-direction.

Save your figure as `beale_surf.png`.

Problem 2 (Writeup).
Create a *contour plot* of the Beale function.

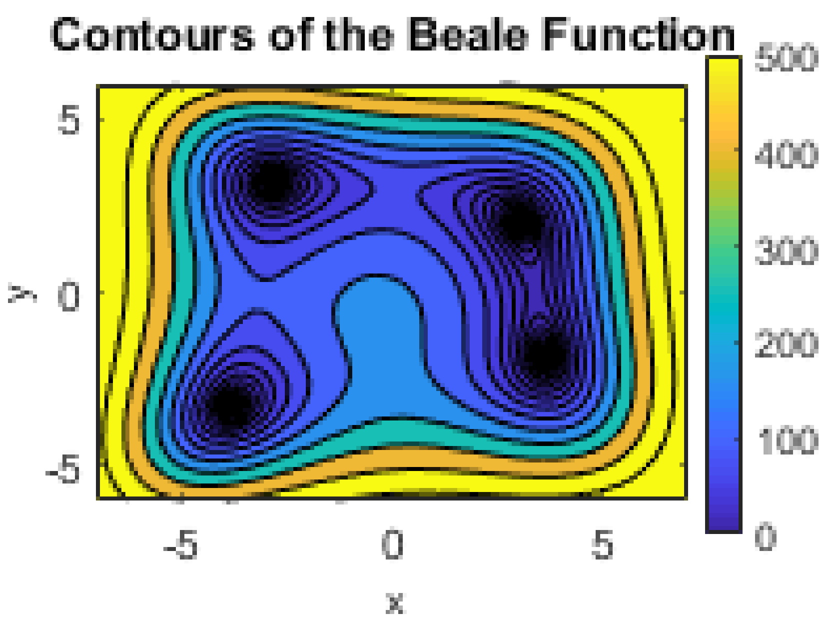

Figure 2: A low-resolution version of `beale_contour.png`. Make sure your plot looks better than this!

Make the following adjustments to the plot:

- Specify the levels of the contour plot to be 21 **logarithmically-spaced** points from 10^{-1} to 10^{3}. Look up the `logspace` command to do this.
- Set the limits of the *color axis* to match the surface plot — 0 to 500.
- Add a *color bar* that indicates what values the colors correspond to.

Use the `contourf` function to produce this plot. You will again need the `meshgrid` command. This time plot within the domain $-7 \le x \le 7$ and $-6 \le y \le 6$ with 100 linearly-spaced points in both directions. The finer grid spacing is required here to get smooth-looking contours.

Save your figure as `beale_contour.png`.

Gradient Descent

We will next implement the gradient descent method to find the minimum.

Problem 3 (Scorelator).

Write a MATLAB function that computes $f(x, y)$ using only one input (i.e. one variable: a vector). This is a packaging problem. Your input will need to be a vector of length 2. Use this function to calculate $f(1, 1)$ and save the result to **A1.dat**.

Problem 4 (Scorelator).

Find a formula for the gradient $\nabla f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$. Recall from calculus that the gradient is defined as the 2×1 vector

$$\nabla f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial f}{\partial x}(x, y) \\ \frac{\partial f}{\partial y}(x, y) \end{bmatrix}.$$

The gradient is a function from $\mathbb{R}^2$ to $\mathbb{R}^2$: it takes in a vector (a position (x, y)) and returns a vector (the gradient at the point (x, y)).

Write your gradient function as a function of one input $\vec{p} = (x, y)$. Calculate $\nabla f(\vec{p})$ for the point $\vec{p} = (1, 1)$ and save the resulting 2×1 vector in **A2.dat**.

The gradient of f should be zero at a local minimum. Find the infinity-norm of the gradient at $\vec{p} = (1, 1)$ and confirm that it is not particularly close to zero. Save the result in **A3.dat**.

Problem 5 (Scorelator).

The vector $-\nabla f(\vec{p})$ (the negative gradient) points from the point $\vec{p}$ in the direction where f decreases the fastest. In other words, it points in the direction of steepest descent. Write a function in MATLAB defined by

$$\phi(t) = \vec{p} - t\nabla f(\vec{p}).$$

The function $\phi(t)$ defines a line, starting from the point $\vec{p}$, that moves in the direction of the negative gradient as t is increased. The function ϕ goes from $\mathbb{R}$ (the input t) to $\mathbb{R}^2$ (the space of x–y points).

Calculate $\phi(0.1)$ and save the resulting 2×1 vector in **A4.dat**. Calculate $f(\phi(0.1))$ and save the resulting value in **A5.dat**.

Problem 6 (Scorelator).

The function $f(\phi(t))$ is decreasing for small values of t (in the direction of descent) but at some point it starts increasing again. This means that $f(\phi(t))$ has a minimum at some point $t^* > 0$. We could implement golden section search or Newton's method to find this minimum, but we will make things easy by just using the built-in MATLAB function `fminbnd`. Use `fminbnd` to find t^* that minimizes $f(\phi(t))$ on the interval $0 < t < 0.1$. Save the resulting t^* in **A6.dat** and the resulting 2×1 vector $\phi(t^*)$ in **A7.dat**.

You just completed one step of the gradient descent iterative method. Starting with an initial guess of $\vec{p} = (1, 1)$, we found that the gradient of f at this point was not zero — we were not at a minimum — and so we calculated a new guess. To find the new guess, we found the direction of steepest descent, $-\nabla f(x, y)$, and then we found the minimum value of f in this direction using the `fminbnd` command. The resulting point $\vec{p}_{\text{new}} = \phi(t^*)$ is our new guess.

Problem 7 (Scorelator).

Now we will repeat the process from Problems 4–6 over and over again until we find the minimum of f. This is an iterative method, so the core of this process should be a loop: every time we go through the loop, we calculate a new guess.

Write code to repeat the gradient descent iteration for up to 1000 iterations. Add a stopping condition so that the iteration halts if the infinity-norm of the gradient at the current iterate is less than 10^{-4}. Re-start from the point $x = y = 1$ — so the first iteration of your code should in effect repeat the calculations from Problems 4–6.

Things to keep in mind:

1. Remember that the gradient must be evaluated at the coordinates of the new guess at each step.
2. The function $\phi(t)$ must also be updated based on the new guess. The numbers $x = 1, y = 1$ in Problem 4 need to be replaced based on the new guess.

Save the final iterate as a 2×1 vector to the file **A8.dat**. Save the number of iterates to **A9.dat**. Remember that the initial guess does not count as an iterate.

Minimizing with built-in functions

Problem 8 (Scorelator).

Use `fminsearch` repeatedly to find the minima (the multiple local minimums) of f. Use appropriate initial guesses (based on what you see in the plots you created in the previous problems) to find one local minimum in each of the four x–y quadrants.

Save the minima from the first, second, third, and fourth quadrants in a 2×4 matrix where each column is a minimum. Save this result to file as **A10.dat**

Problem 9 (Writeup).

Copy your contour plot code from Problem 2. Add code to plot:

- The local minima you found with `fminsearch` as open red circles.
- The local minimum you found with gradient descent as a yellow star (`'*'`).
- The global mimimum as a green plus sign (`'+'`).

Change the title of the figure to "Minima of the Beale Function', and save your figure as `beale_minima.png`.