Imperial College London

COURSEWORK 2

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Mathematics for Machine Learning

Author:
Arnold Cheung (CID: 01184493)

Date: October 24, 2019

1 Differentiation

(a) First, expand the completed square form:

$$\begin{aligned}&(\boldsymbol{x}-\boldsymbol{c})^T\boldsymbol{C}(\boldsymbol{x}-\boldsymbol{c})+c_0\\=&\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}-\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{c}-\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{x}+\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{c}+c_0\\=&\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}-2\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{x}+\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{c}+c_0\end{aligned}$$

Then compare coefficients with $f_1(x)$, for $\boldsymbol{C}$:

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}&=\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x}-\boldsymbol{x}^T\boldsymbol{x}\\\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}&=\boldsymbol{x}^T(\boldsymbol{B}-\boldsymbol{I})\boldsymbol{x}\\=>\boldsymbol{C}&=\boldsymbol{B}-\boldsymbol{I}\\&=\begin{bmatrix}4&-2\\-2&4\end{bmatrix}-\begin{bmatrix}1&0\\0&4\end{bmatrix}\\&=\begin{bmatrix}3&-2\\-2&3\end{bmatrix}\end{aligned}$$

For $\boldsymbol{c}$:

$$\begin{aligned}-2\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{x}&=\boldsymbol{a}^T\boldsymbol{x}-\boldsymbol{b}^T\boldsymbol{x}\\-2\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{x}&=(\boldsymbol{a}^T-\boldsymbol{b}^T)\boldsymbol{x}\\-2\boldsymbol{c}^T\boldsymbol{C}&=(\boldsymbol{a}^T-\boldsymbol{b}^T)\\\boldsymbol{c}^T&=(\frac{\boldsymbol{b}^T-\boldsymbol{a}^T}{2})\boldsymbol{C}^{-1}\\\boldsymbol{c}&=((\frac{\boldsymbol{b}^T-\boldsymbol{a}^T}{2})\boldsymbol{C}^{-1})^T\\\boldsymbol{c}&=(\frac{1}{5}(\frac{[-2\quad 0]-[0\quad 1]}{2})\begin{bmatrix}3&2\\2&3\end{bmatrix})^T\\\boldsymbol{c}&=(\frac{1}{5}([-1\quad 0])\begin{bmatrix}3&2\\2&3\end{bmatrix})^T\\\boldsymbol{c}&=\frac{1}{5}\begin{bmatrix}-3\\-2\end{bmatrix}\end{aligned}$$

For c_0:

$$\begin{aligned}
\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{c} + c_0 &= 0 \\
c_0 &= -\boldsymbol{c}^T\boldsymbol{C}\boldsymbol{c} \\
c_0 &= -\frac{1}{25}\begin{bmatrix}-3 & -2\end{bmatrix}\begin{bmatrix}3 & -2\\ -2 & 3\end{bmatrix}\begin{bmatrix}-3\\ -2\end{bmatrix} \\
c_0 &= -\frac{3}{5}
\end{aligned}$$

Therefore, $\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{x} + 2\boldsymbol{a}^T\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x}$ can be written as:

$$(\boldsymbol{x} - \frac{1}{5}\begin{bmatrix}-3\\ -2\end{bmatrix})^T\begin{bmatrix}3 & -2\\ -2 & 3\end{bmatrix}(\boldsymbol{x} - \frac{1}{5}\begin{bmatrix}-3\\ -2\end{bmatrix}) - \frac{3}{5}$$

(b) f_1 has minimum point can be deduced from $\boldsymbol{C}$ being positive definite. The minimum point is located at $x = \boldsymbol{c} = \begin{bmatrix}-0.6\\ -0.4\end{bmatrix}$

(c) The gradient of each function is given by the first derivative with respect to x, the respective derivatives are:

$$\begin{aligned}
\frac{df_1}{dx} &= 2\boldsymbol{x}^T\boldsymbol{B} - 2\boldsymbol{x}^T + \boldsymbol{a}^T + \boldsymbol{b}^T \\
\frac{df_2}{dx} &= -2\sin[(\boldsymbol{x}-\boldsymbol{b})^T(\boldsymbol{x}-\boldsymbol{b})](\boldsymbol{x}-\boldsymbol{b})^T + 2(\boldsymbol{x}-\boldsymbol{a})\boldsymbol{B} \\
\frac{df_3}{dx} &= 2\exp[(\boldsymbol{x}-\boldsymbol{a})^T(\boldsymbol{x}-\boldsymbol{a})](\boldsymbol{x}-\boldsymbol{a})^T + 2\exp[(\boldsymbol{x}-\boldsymbol{b})^T\boldsymbol{B}(\boldsymbol{x}-\boldsymbol{b})](\boldsymbol{x}-\boldsymbol{b})^T\boldsymbol{B} \\
&\quad + \frac{1}{10}tr((\frac{1}{100}\boldsymbol{I} + \boldsymbol{x}x^T)^{-1}\frac{d\boldsymbol{x}\boldsymbol{x}^T}{d\boldsymbol{x}})
\end{aligned}$$

The code implementations can be found in the coding_ansewrs.py file.

(d) The continuous colour represents the value of the function at a given point, and the discrete contours represent the magnitude of the gradient of the function at a given point. All of the gradient descents below have a starting position of $\begin{bmatrix} 0.3 \\ 0 \end{bmatrix}$ with 50 iterations, shown by the red line on both plots:

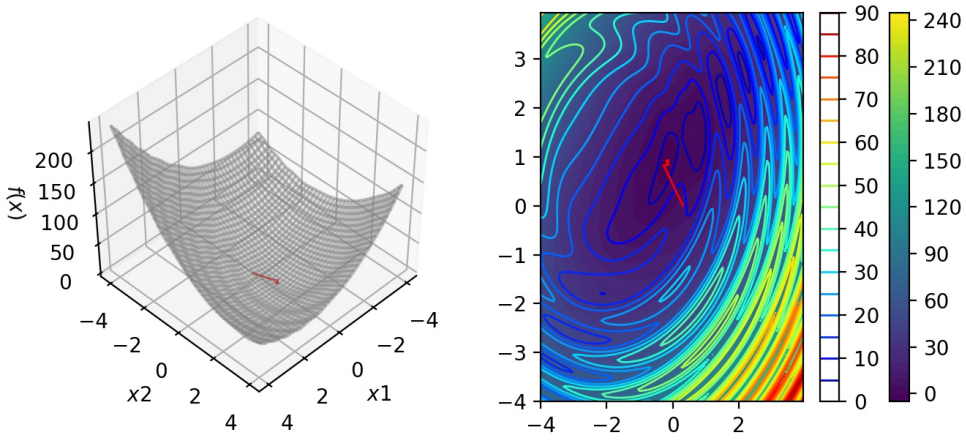

Figure 1: The 3D and contour plot of f_2, step size = 0.07.

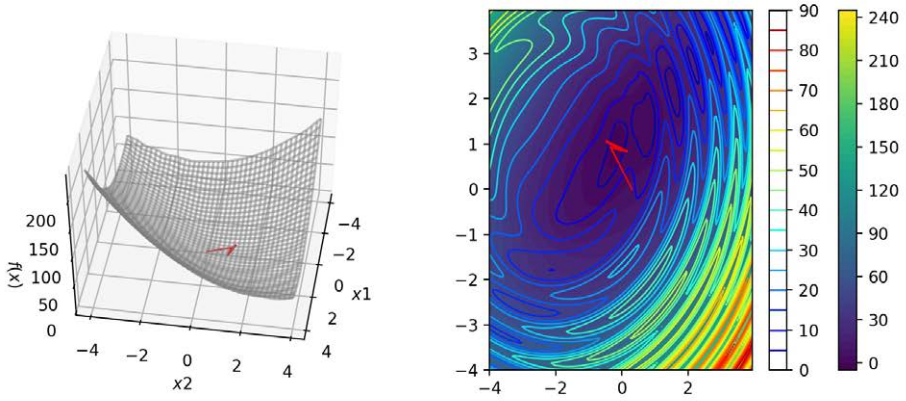

Figure 2: The 3D and contour plot of f_2, step size = 0.1.

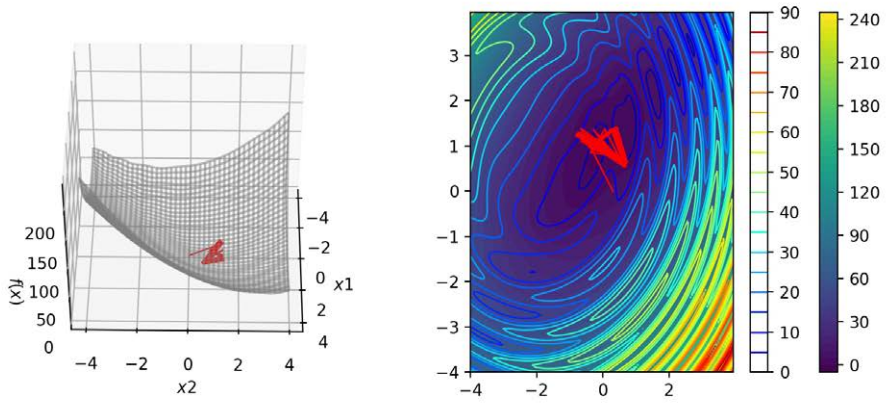

Figure 3: The 3D and contour plot of f_2, step size = 0.15.

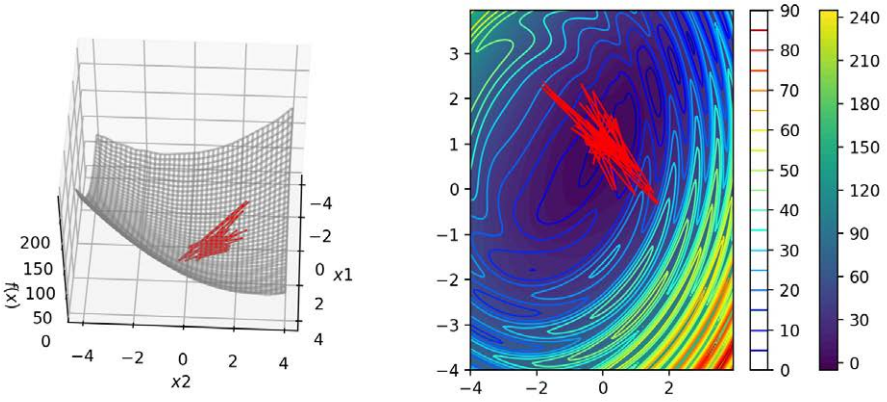

Figure 4: The 3D and contour plot of f_2, step size = 0.173.

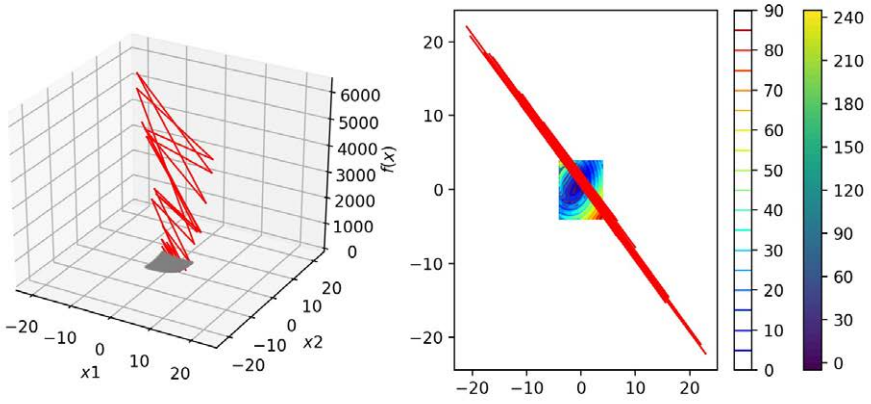

Figure 5: The 3D and contour plot of f_2, step size = 0.174.

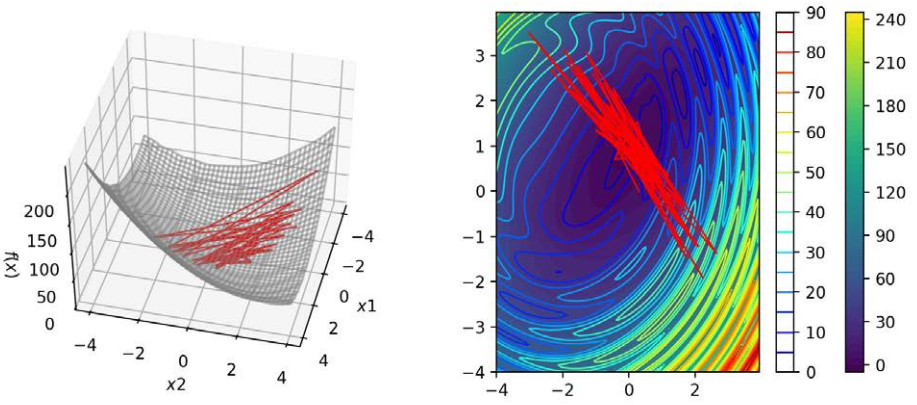

Figure 6: The 3D and contour plot of f_2, step size = 0.1747.

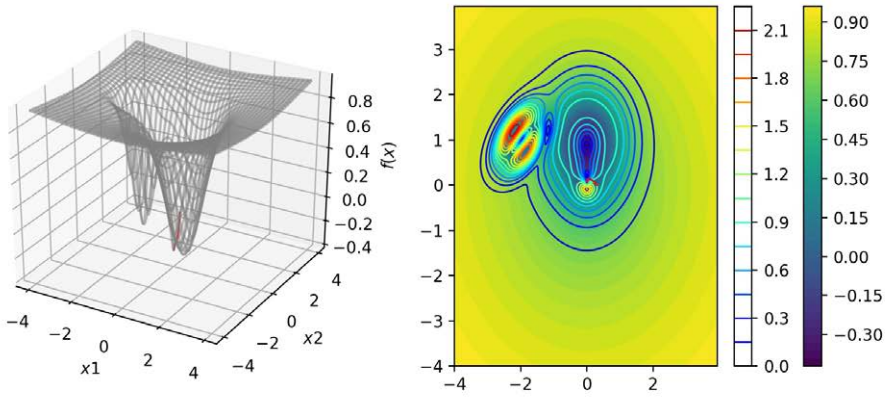

Figure 7: The 3D and contour plot of f_3, step size = 0.07.

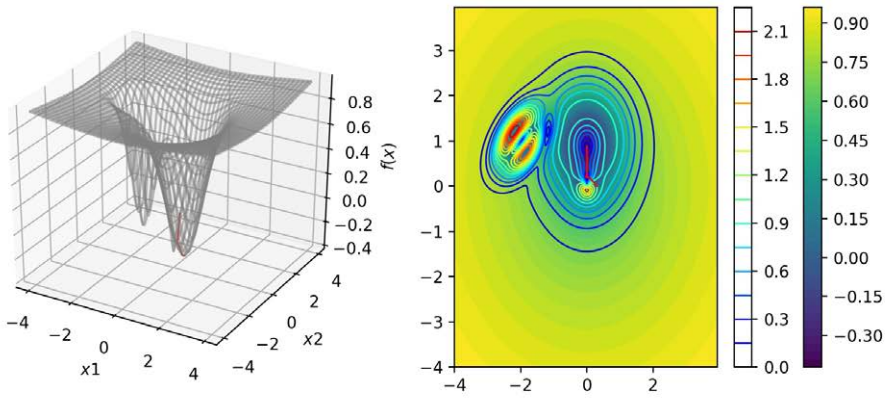

Figure 8: The 3D and contour plot of f_3, step size = 0.26.

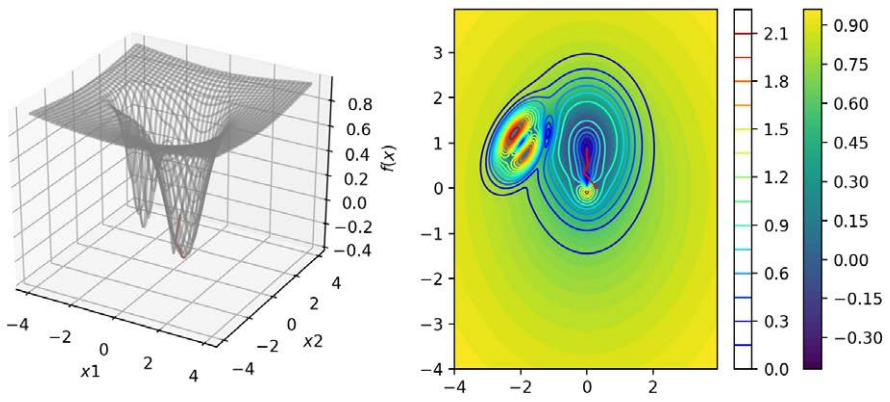

Figure 9: The 3D and contour plot of f_3, step size = 0.5.

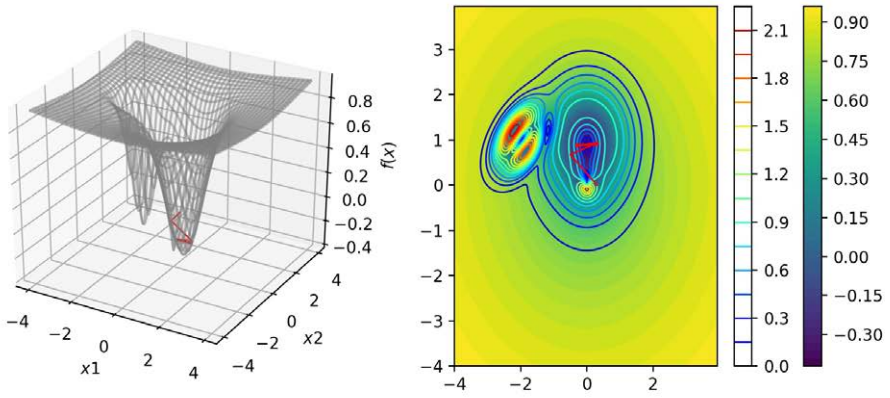

Figure 10: The 3D and contour plot of f_3, step size = 1.0.

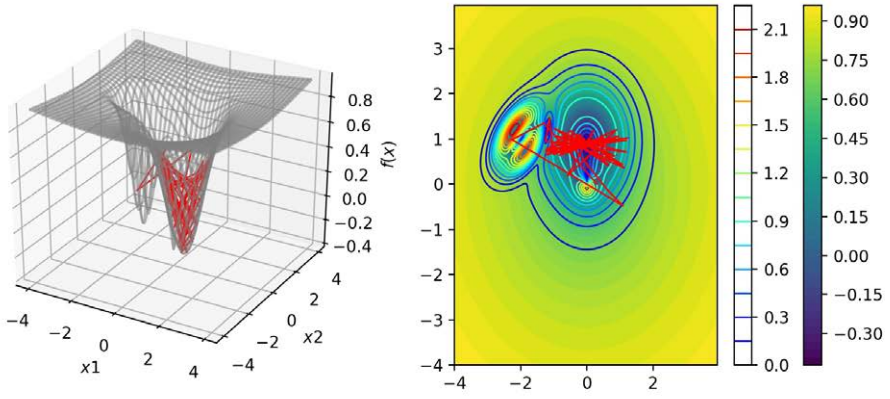

Figure 11: The 3D and contour plot of f_3, step size = 2.0.

(e) For f_2, for step sizes small enough (smaller than 0.1), the gradient descent algorithm reliably locates the minimum around [-0.16, 0.93] of the function as shown in Figure 1. However, the algorithm begins to lose its convergence from step size around 0.1 and above, from Figure 2 to Figure 4, it can be seen that the output position of the algorithm fluctuates increasingly with an increasing step size (from 0.1 to 0.173). The result eventually diverges at step size = 0.174 as shown in Figure 5, where the algorithm is "climbing" out of the valley to a very high value, outside the range of the predefined axes. However, surprisingly when step size is increased to 0.1747 the algorithm is again contained within the predefined axes, as shown in Figure 6. This effect is likely due to the fact that when step size = 0.174, the algorithm lands on the right side of the periodic wave where the gradient points towards the minimum. And for step sizes beyond 1.0, the algorithm simply outputs values that increases exponentially.

For f_3, for step sizes small enough (smaller than 0.26), the gradient descent algorithm lands inside the local minimum of the small valley right next to the largest valley around [0,0] as shown in Figure 7. Step size = 0.26 is the first time it skips the small valley in to the lower minimum in the largest valley around [0, 0.89]. This remains true for increasing step sizes, although the lowest minimum cannot be reached for step sizes too big, such as 1.0 shown in Figure 10. For step sizes beyond 1.0, for example at 2.0 shown in Figure 11, although the algorithm is still contained within the big valley, it is increasingly unstable and even reaching positions in the second biggest valley occasionally.