Engineering Optimization Lab1

Muhammad Umer

March 2019

Consider a function as follows

$$f(x) = \frac{x^2}{10} - 2sin(x) \tag{1}$$

The above function can be plotted as follows using Python

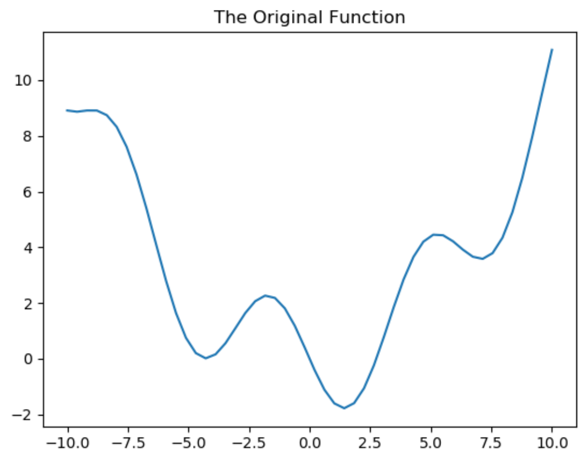

Figure 1: $f(x) = \frac{x^2}{10} - 2sin(x)$

Now let's try to find the minimum of this function analytically by setting the derivative equal to zero as follows

$$\frac{df(x)}{dx} = \frac{x}{5} - 2cos(x) = 0 \tag{2}$$

As it can be seen from above that the minimum for this simple 1D function can not be found analytically and we need to use the numerical optimization techniques to find the optimal solution.

1 Steepest Descent

1.1 Steepest Descent With fixed Step Size

Python Code

```
def GD(x0, alpha, eps,max_iter):
    iter = 0
    y = x0
    y_new = np.zeros((max_iter,1))
    y_new[iter,] = y
    iter += 1
    y = y - alpha * df(y)
    y_new[iter,] = y
    while (np.abs(y_new[iter,] - y_new[iter-1,]) > eps and iter < max_iter):
        iter += 1
        if (iter == max_iter):
            break
        y = y - alpha* df(y)
        y_new[iter,] = y
    return y,iter,y_new
```

x_o = -5 and α = 0.1 The algorithm converges to a local minimum value of -4.27113696 in 44 iterations

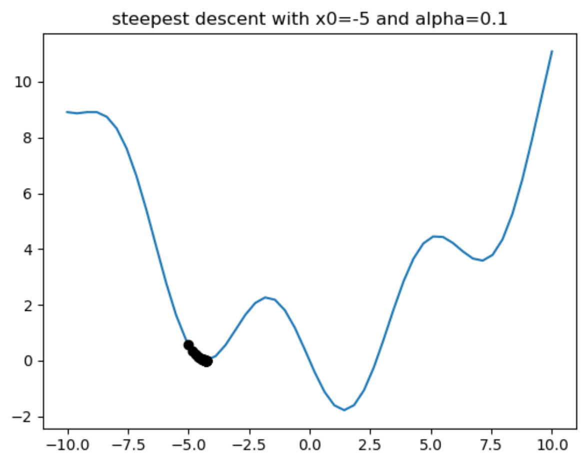

Figure 2: Steepest Descent with x_o=-5 and α=0.1

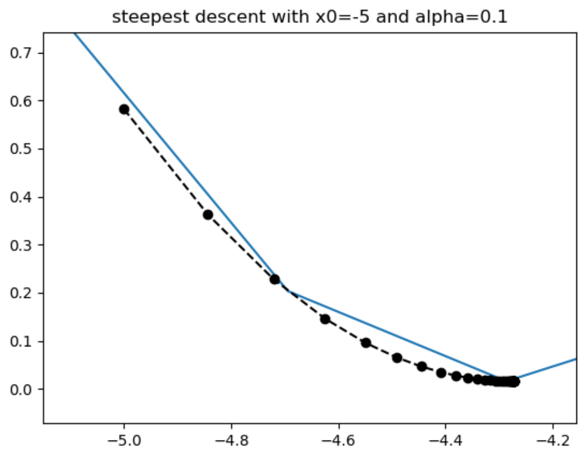

Figure 3: Zoomed in version of Steepest Descent with x_o=-5 and α=0.1

x_o = -5 and α = 0.5 The algorithm converges to a local minimum value of -4.27109712 in 4 iterations. It converges faster as compared to the previous case.

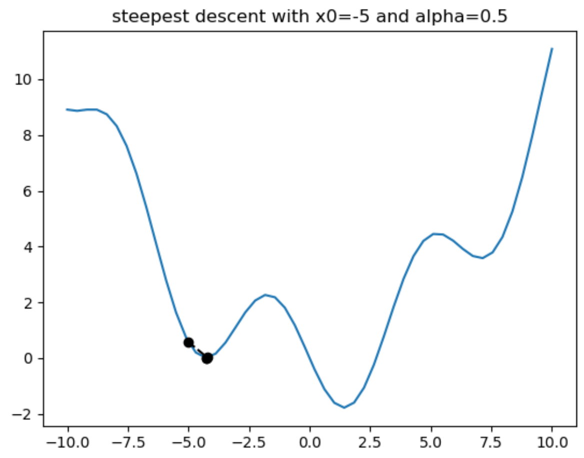

Figure 4: Steepest Descent with x_o=-5 and α=0.5

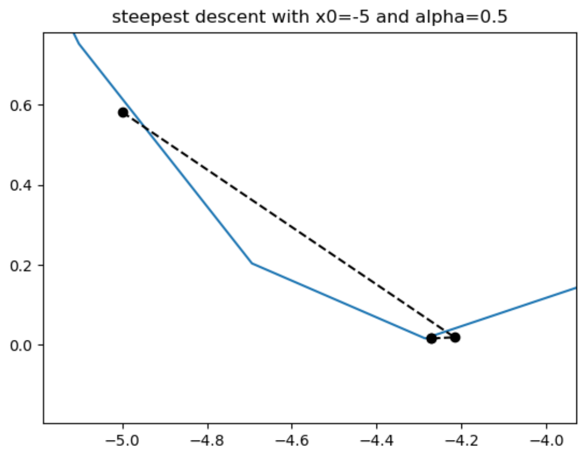

Figure 5: Zoomed in version of Steepest Descent with x_o=-5 and α=0.5

x_o = -5 and α = 1, the algorithm does not converge to a local minimum, it zigzags around a local minimum without converging.

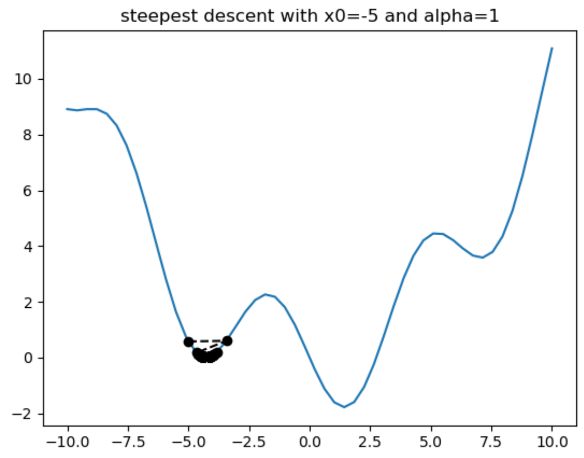

Figure 6: Steepest Descent with x_o=-5 and α=1

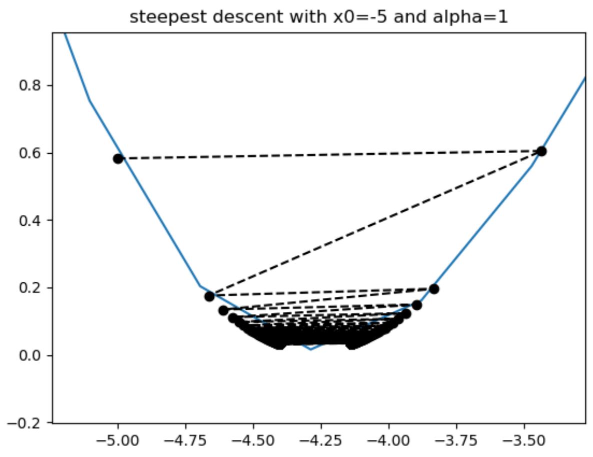

Figure 7: Zoomed in version of Steepest Descent with x_o=-5 and α=1

Effects of Step size: When the step size is too small i.e. 0.1, the algorithm converges to local minimum but it need to iterate many times to get to the solution i.e. 44 iterations, therefore slower convergence. On the other side, when the step size is too big i.e. 1, the algorithm zigzags around the solution and never converges to local minimum. But when we set α=0.5, the algorithm converges to a local minimum in just 4 iterations. Therefore, step size needs to be selected intelligently in order to get faster convergence and avoid overshooting.

$x_o = 0.5$ and $\alpha = 0.1$ The algorithm converges to a global minimum value of 1.42751529 in 43 iterations

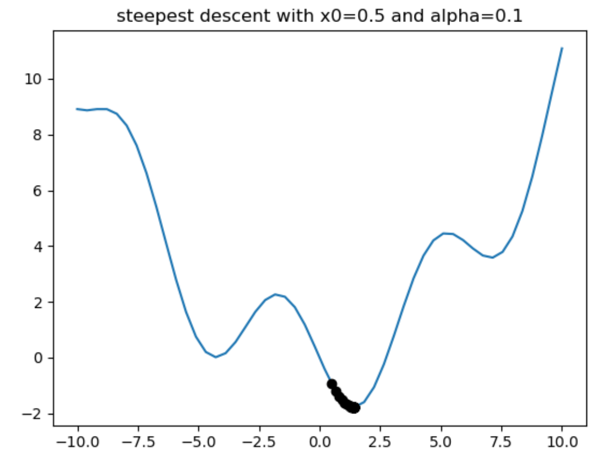

Figure 8: Steepest Descent with x_o=0.5 and α=0.1

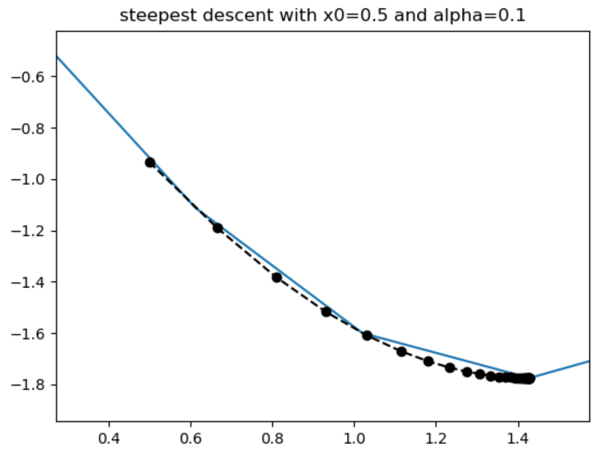

Figure 9: Zoomed in version of Steepest Descent with x_o=0.5 and α=0.1

x_o = 0.5 and α = 0.5 The algorithm converges to a global minimum value of 1.42754589 in 6 iterations

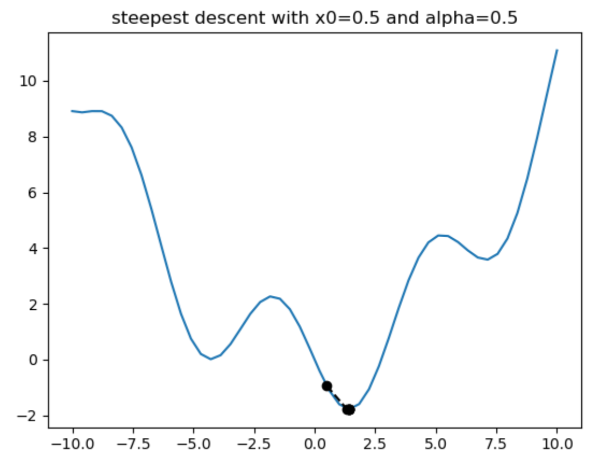

Figure 10: Steepest Descent with x_o=0.5 and α=0.5

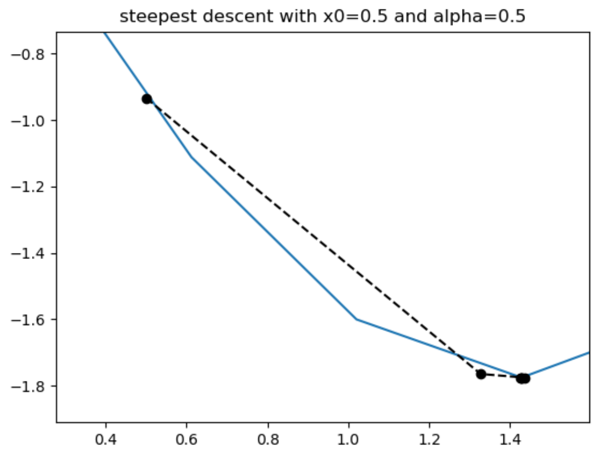

Figure 11: Zoomed in version of Steepest Descent with x_o=0.5 and α=0.5

$x_o = 0.5$ and $\alpha = 1$, the algorithm again does not converge to a local or a global minimum, it starts zigzagging around a global minimum without converging.

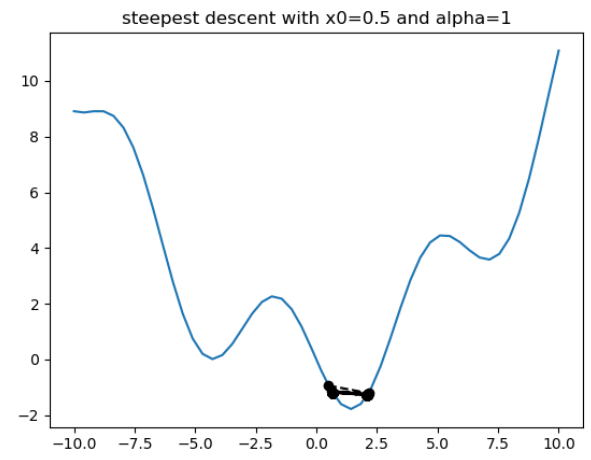

Figure 12: Steepest Descent with x_o=0.5 and α=1

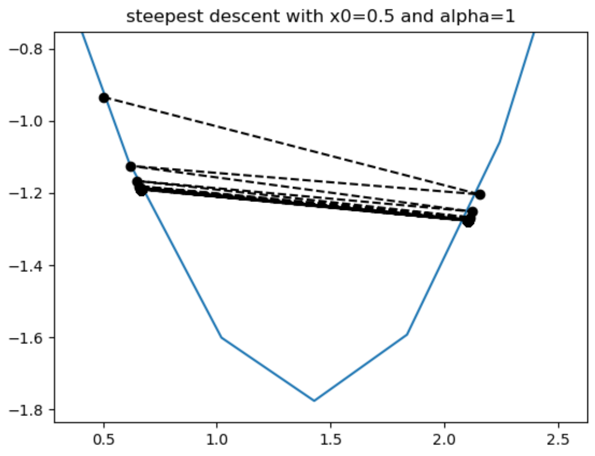

Figure 13: Zoomed in version of Steepest Descent with x_o=0.5 and α=1

1.2 Steepest Descent With an Adaptive Step Size Reduction Rule

Python Code

```
def SGD(x0, alpha, eps,max_iter,gamma):
    iter = 0
    y = x0
    y_new = np.zeros((max_iter,1))
    y_new[iter,] = y
    iter += 1
    alpha = gamma * alpha
    y = y - alpha * df(y)
    y_new[iter,] = y
    while(np.abs(y_new[iter,] - y_new[iter-1,]) > eps and iter < max_iter):
        iter += 1
        if(iter == max_iter):
            break
        y = y - alpha* df(y)
        y_new[iter,] = y
        alpha = gamma * alpha
    return y,iter,y_new
```

Using a forgetting factor of γ=0.9, initial starting point $x_o = 0.5$ and step size $\alpha = 1$, the algorithm converges to a global minimum of value 1.42754209 in 10 iterations after zigzagging a little bit around the global minimum. The forgetting factor decreases the step size at each iteration intelligently to avoid overshooting and it helps in convergence.

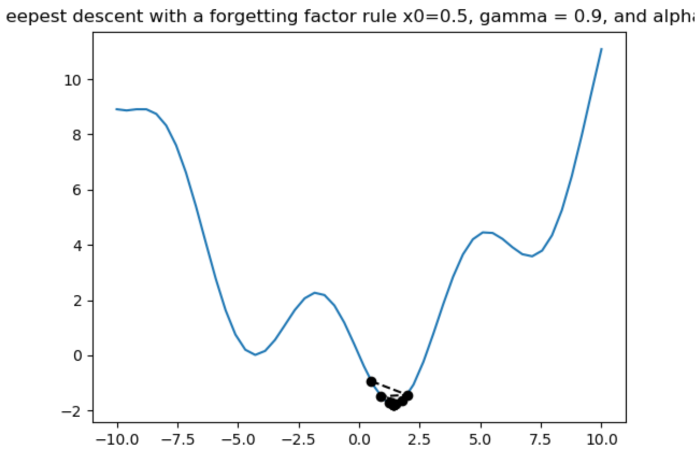

Figure 14: Steepest Descent with γ =0.9, x_o=0.5 and α=1

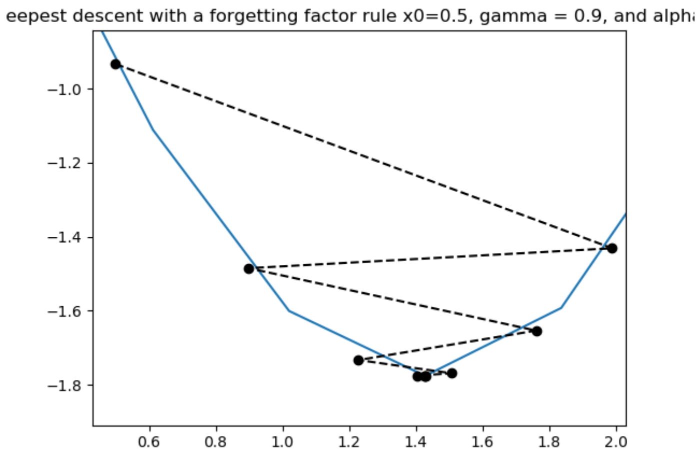

Figure 15: Zoomed in version of Steepest Descent with γ =0.9, x_o=0.5 and α=1

Using a forgetting factor of γ=0.5, initial starting point $x_o = 0.5$ and step size $\alpha = 1$, the algorithm converges to a value of 1.42958162, which is close to a global minimum value of 1.42754209 in again 10 iterations but this time more smoothly i.e. without overshooting.

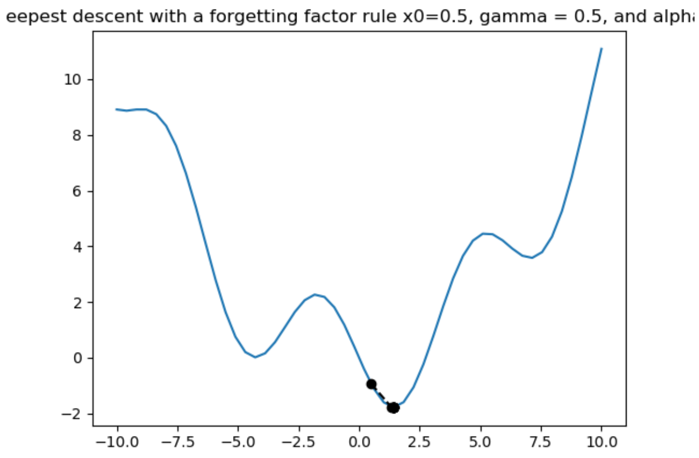

Figure 16: Steepest Descent with γ =0.5, x_o=0.5 and α=1

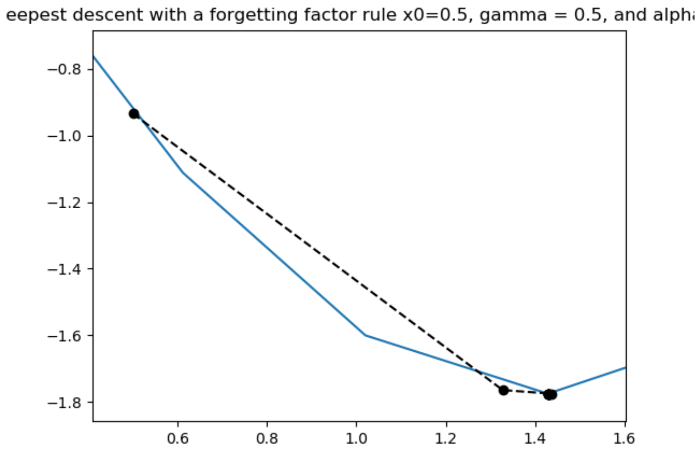

Figure 17: Zoomed in version of Steepest Descent with γ =0.5, x_o=0.5 and α=1

Proposed Adaptive Step Size
Python Code

```
def SGD_proposed(x0, alpha, eps, max_iter, gamma):
    iter = 0
    y = x0
    y_new = np.zeros((max_iter,1))
    y_new[iter,] = y
    iter += 1
    alpha = (gamma**(iter)) * alpha
    y = y - alpha * df(y)
    y_new[iter,] = y
    while(np.abs(y_new[iter,] - y_new[iter-1,]) > eps and iter < max_iter) :
        iter += 1
        if(iter == max_iter):
            break
        y = y - alpha* df(y)
        y_new[iter,] = y
        alpha = (gamma**(iter)) * alpha
    return y, iter, y_new
```

We propose using the following step size reduction rule i.e. use $\alpha = \gamma^k * \alpha$ where, k=0,1,... and $\gamma < 1$. Now using x_o=0.5, α=1, and γ=0.95, the algorithm converges to global minimum of value 1.42748933 in 11 iterations after zigzagging a little bit around the global minimum.

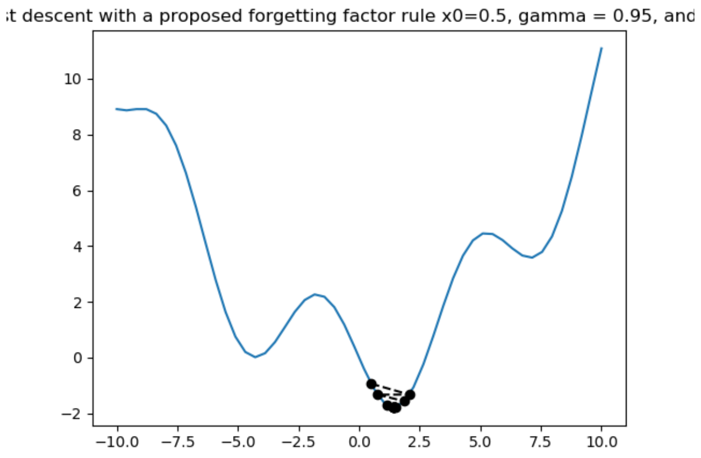

Figure 18: Steepest Descent with proposed γ =0.95, x_o=0.5 and α=1

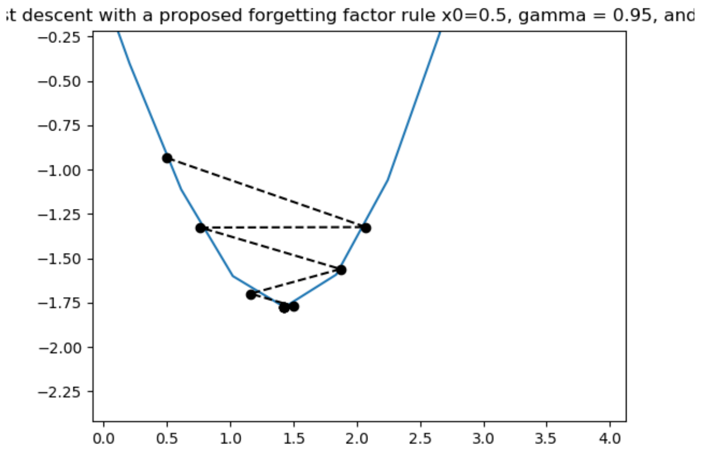

Figure 19: Zoomed in version of Steepest Descent with proposed γ =0.95, x_o=0.5 and α=1

1.3 Steepest Descent With Armijo Rule Step Size Reduction

Python Code

```
def SGD_w_armijo(x0, s, sig, beta, max_iter):
```

```
iter = 0
y = x0
alpha = s
y_new = np.zeros((max_iter,1))
y_new[iter,] = y
for i in range(1,max_iter):
    alpha = s * (beta ** (i))
    y = y - alpha * df(y)
    print(y)
    y_new[i,] = y
    if ((f(y_new[i-1,]) - f(y_new[i,]))
    <= (1*sig *s *(beta ** (i)) *(df(y_new[i-1,]))**2)):
        break
return y, i, y_new
```

With the initial starting point of $x_o = 0.5$, we use the initial step sizes of s=0.1, 0.5, and 1. We tried the values of σ = 1e-5, 1e-3, 1e-1 and values of β = 0.5, 0.7, and 0.9 respectively for each s.
For s = 0.1
σ = 1e-5 and β = 0.5 (No convergence)
σ = 1e-5 and β = 0.7 (No convergence)
σ = 1e-5 and β = 0.9 (No convergence)
σ = 1e-3 and β = 0.5 (No convergence)
σ = 1e-3 and β = 0.7 (No convergence)
σ = 1e-3 and β = 0.9 (No convergence)
σ = 1e-1 and β = 0.5 (No convergence)
σ = 1e-1 and β = 0.7 (No convergence)
σ = 1e-1 and β = 0.9 (No convergence)

For s = 0.5
σ = 1e-5 and β = 0.5 (No convergence)
σ = 1e-5 and β = 0.7 (No convergence)
σ = 1e-5 and β = 0.9 (Converges to a global minimum of value 1.42755134 in 77 iterations)
σ = 1e-3 and β = 0.5 (No convergence)
σ = 1e-3 and β = 0.7 (No convergence)
σ = 1e-3 and β = 0.9 (Converges to a global minimum of value 1.42755134 in 77 iterations)
σ = 1e-1 and β = 0.5 (No convergence)
σ = 1e-1 and β = 0.7 (No convergence)
σ = 1e-1 and β = 0.9 (Converges to a global minimum of value 1.42755134 in 77 iterations)

For s = 1
σ = 1e-5 and β = 0.5 (No convergence)
σ = 1e-5 and β = 0.7 (Converges to a value 1.42702495 which is closer to the

global minimum in 66 iterations)
σ = 1e-5 and β = 0.9 (Converges to a global minimum of value of 1.42755178 in 16 iterations)
σ = 1e-3 and β = 0.5 (No convergence)
σ = 1e-3 and β = 0.7 (Converges to a value 1.42702495 which is closer to the global minimum in 66 iterations)
σ = 1e-3 and β = 0.9 (Converges to a global minimum of value of 1.42755178 in 16 iterations)
σ = 1e-1 and β = 0.5 (No convergence)
σ = 1e-1 and β = 0.7 (Converges to a value 1.42702495 which is closer to the global minimum in 66 iterations)
σ = 1e-1 and β = 0.9 (Converges to a global minimum of value of 1.42755178 in 16 iterations)

1.4 Newton Method

Python Code

```
def NM(x0, eps, max_iter):
        iter = 0
        y = x0
        y_new = np.zeros((max_iter, 1))
        y_new[iter,] = y
        iter += 1
        y = y - df(y)/ddf(y)
        y_new[iter,] = y
        while (np.abs(y_new[iter,] - y_new[iter - 1,])
        > eps and iter < max_iter):
            iter += 1
            if (iter == max_iter):
                break
            y = y - df(y)/ddf(y)
            y_new[iter,] = y
        return y, iter, y_new
```

x_o=-5 converges to a local minimum of value -4.27109534 in 4 iterations

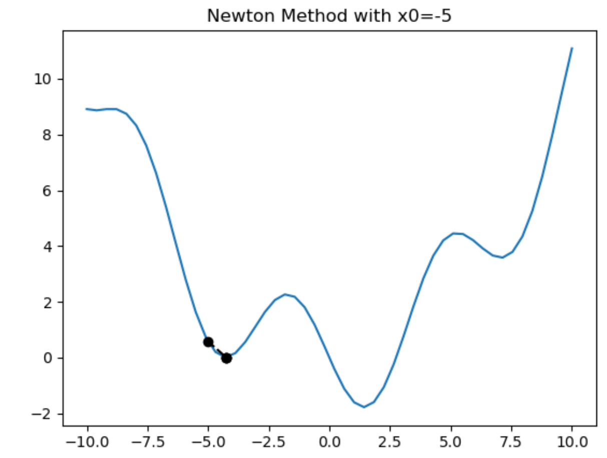

Figure 20: Newton Method with x_o=-5

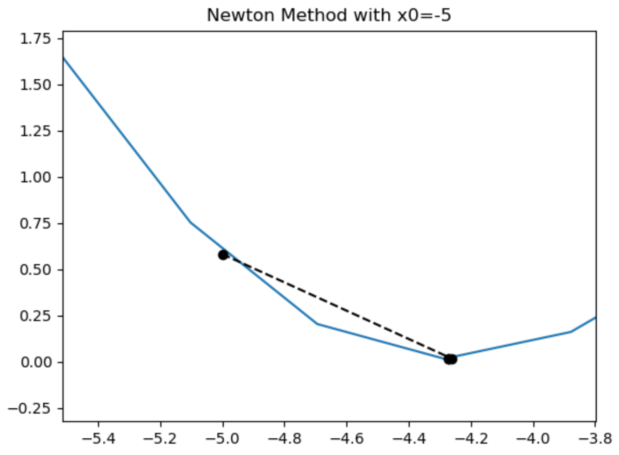

Figure 21: Zoomed in version of Newton Method with x_o=-5

x_o=0.5 converges to a global minimum of value 1.42755178 in 5 iterations

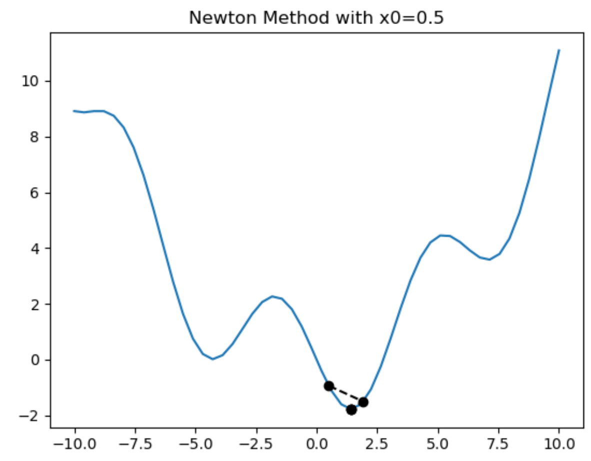

Figure 22: Newton Method with x_o=0.5

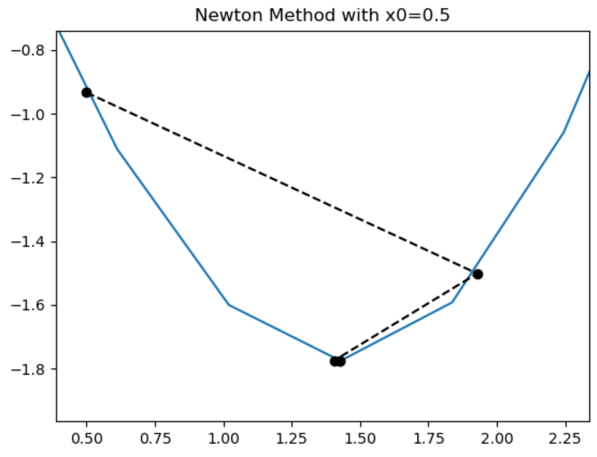

Figure 23: Zoomed in version of Newton Method with x_o=0.5

x_o=0 It neither converges to a global or a local minimum instead it first goes to a global maximum and from there it converges to a stationary point of value 5.26711643.

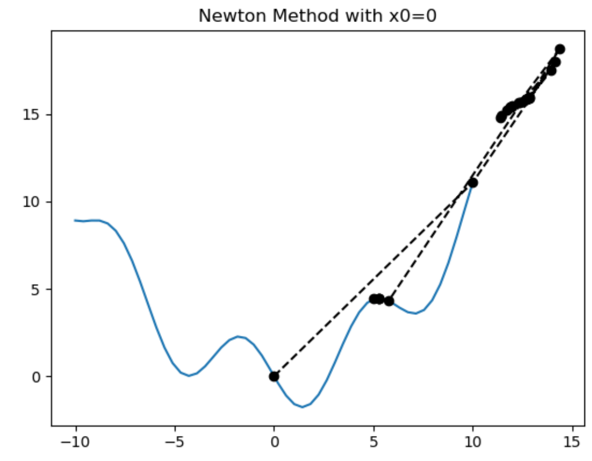

Figure 24: Newton Method with x_o=0

x_o=-6 It neither converges to a global or a local minimum instead it converges to a stationary point of value -1.74632832 i.e. local maximum.

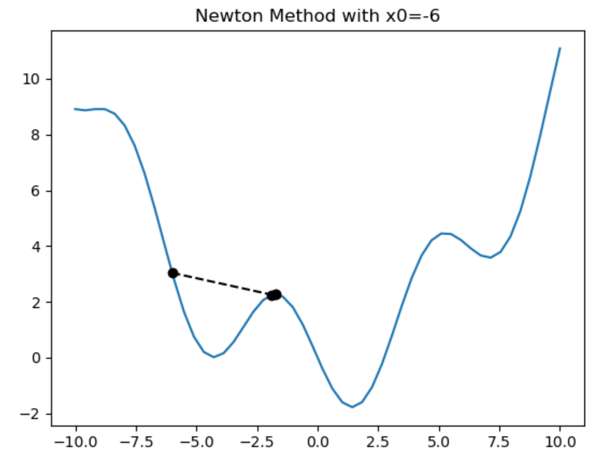

Figure 25: Newton Method with x_o=0

Comments: If we pick initial point closer to a local or global minimum, the Newton's method converges faster with less number of iterations as compared to the steepest descent. However, if we pick the initial point far from the local or global minimum, the algorithm may converge to either a some different stationary point or global maximum