

Optimisation Algorithms for Data Analysis - Assignment 3

CS7DS2

Maurice Kiely - 23350601

23/03/2024

Questions

a) i)

```
1 def SGD(optimizer, f, starting_point, data, batch_size=32,
2         max_iters=1000, **kwargs):
3     optimizer = optimizer.strip().lower()
4     if optimizer not in ['constant', 'polyak', 'rmsprop', 'hb',
5                           'adam']:
6         raise ValueError("Optimizer argument not valid")
7
8     if optimizer == 'constant':
9         alpha = kwargs.get('alpha', 0.01)
10        tol = kwargs.get('tol', 1e-8)
11        path, f_vals = constant(f=f, starting_point=starting_point,
12                               data=data, alpha=alpha, tol=tol, max_iters=max_iters,
13                               batch_size=batch_size)
14        print("Stochastic-Gradient-Descent using Constant Optimiser")
15        return path, f_vals
16
17    elif optimizer == 'polyak':
18        ...
19        ...
20        ...
21
22    else optimizer == 'adam':
23        alpha = kwargs.get('alpha', 0.001)
24        beta1 = kwargs.get('beta1', 0.9)
25        beta2 = kwargs.get('beta2', 0.999)
26
27        epsilon = kwargs.get('epsilon', 1e-8)
28        tol = kwargs.get('tol', 1e-5)
29
30        path, f_vals = adam(f=f, starting_point=starting_point,
31                             data=data, alpha=alpha, beta1=beta1, beta2=beta2,
32                             epsilon=epsilon, max_iters=max_iters, tol=tol,
33                             batch_size=32)
34        print("Stochastic-Gradient-Descent using Adam Optimiser")
35        return path, f_vals
```

The function I implemented for SGD takes an argument to specify which optimization algorithm to use. Once this is done, the function uses kwargs to take in the necessary hyperparameters for each function.

Each function is essentially the same as in the previous assignment, however, there are 2 key differences due to the lack of an explicit function.

Firstly, the training data is broken into batches with the mini_batch function and secondly, the gradient is calculated using finite-difference for each mini-batch.

Both of these functions can be seen here as follows:

```

1 def mini_batch(data, batch_size):
2     batches = []
3     np.random.shuffle(data)
4     n = len(data)
5     for i in np.arange(0, n, batch_size):
6         batch = data[i:(i + batch_size)] if i + batch_size < n
7             else data[i:n]
8         batches.append(batch)
9     return batches

```

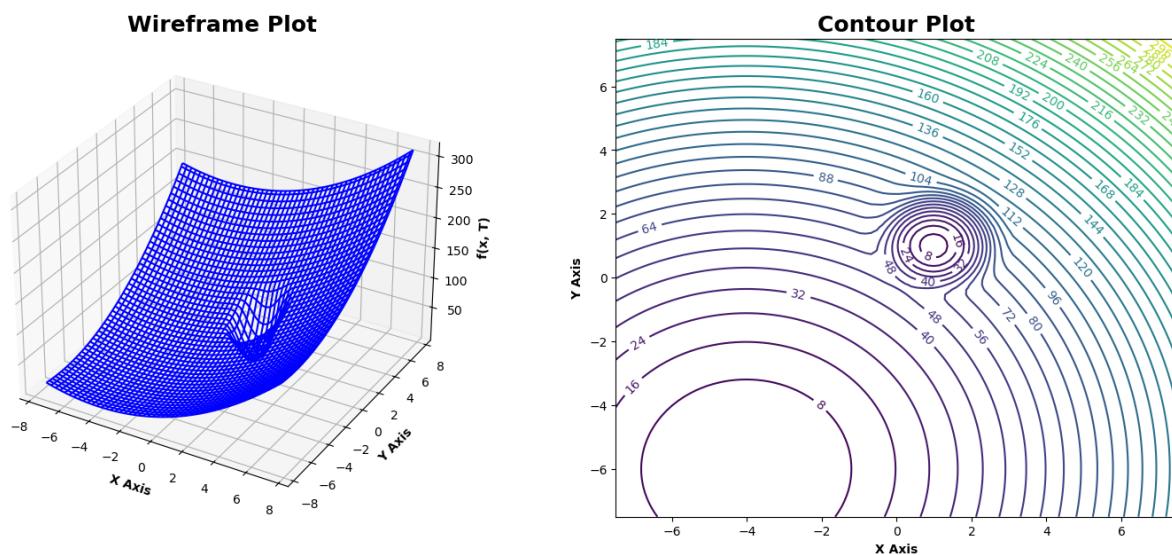
```

1 def batch_finite_diff(f, pt, batch, h=1e-8):
2     batch_dx = (f([pt[0] + h, pt[1]], batch) - f(pt, batch))/h
3     batch_dy = (f([pt[0], pt[1] + h], batch) - f(pt, batch))/h
4     return batch_dx, batch_dy

```

In each of the optimizing algorithms, the batches are used to get both a function value and gradient.

a) ii)



I chose to plot the function from $x, y \in [-7.5, 7.5]$ as outside of these values the function is well-behaved and clearly bowl-shaped. The most important feature I thought to capture was the valley at $(x, y) \approx (1.8, 1.8)$. This is where many of our SGD algorithms will converge. The code used to calculate the grid is seen below:

```

1 training_data = generate_trainingdata()
2 x = np.linspace(-7.5, 7.5, 100)
3 y = np.linspace(-7.5, 7.5, 100)
4 z = []
5 for i in x:
6     for j in y:
7         z.append(f([i, j], training_data))

```

```

8     z = np.array(z).reshape(100, 100)
9     x, y = np.meshgrid(x, y)

```

a) iii)

The method I used to find the derivative at each point was that of finite difference.

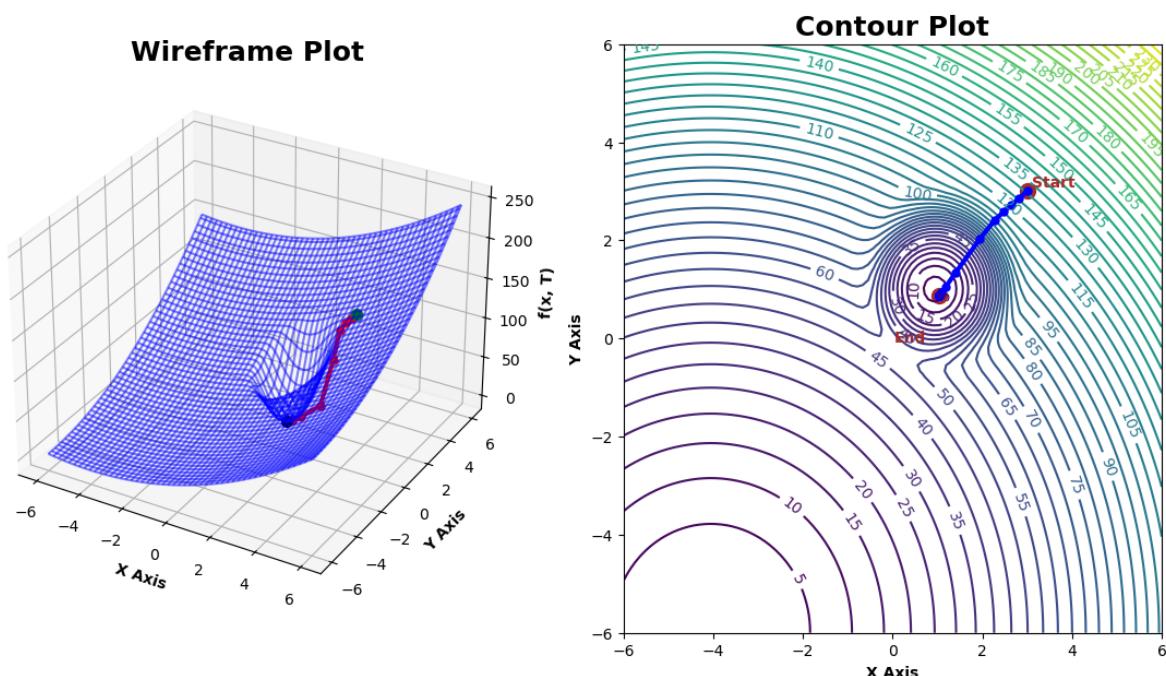
```

1 def batch_finite_diff(f, pt, batch, h=1e-8):
2     batch_dx = (f([pt[0] + h, pt[1]], batch) - f(pt, batch))/h
3     batch_dy = (f([pt[0], pt[1] + h], batch) - f(pt, batch))/h
4     return batch_dx, batch_dy

```

The code implements the function: $f'(x, N) \approx \lim_{h \rightarrow 0} \frac{f(x + h, N) - f(x, N)}{h}$ for each $x_i \in \vec{x}$ which is the derivative. This calculates the rate of change of the loss function based on the points in the batch.

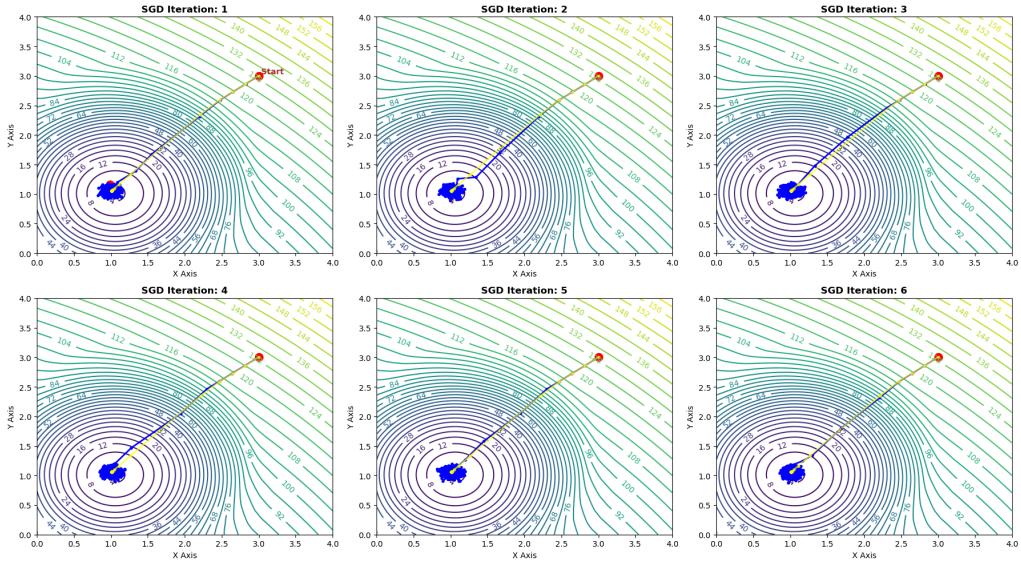
b) i)



Here, the step size that I chose to use was $\alpha = 0.01$ for 100 iterations. I chose alpha as it will converge relatively quickly and won't overshoot the valley for a steep gradient.

This gradient-descent was done by using the SGD function shown above with a constant optimizer and `batch_size = training_data`

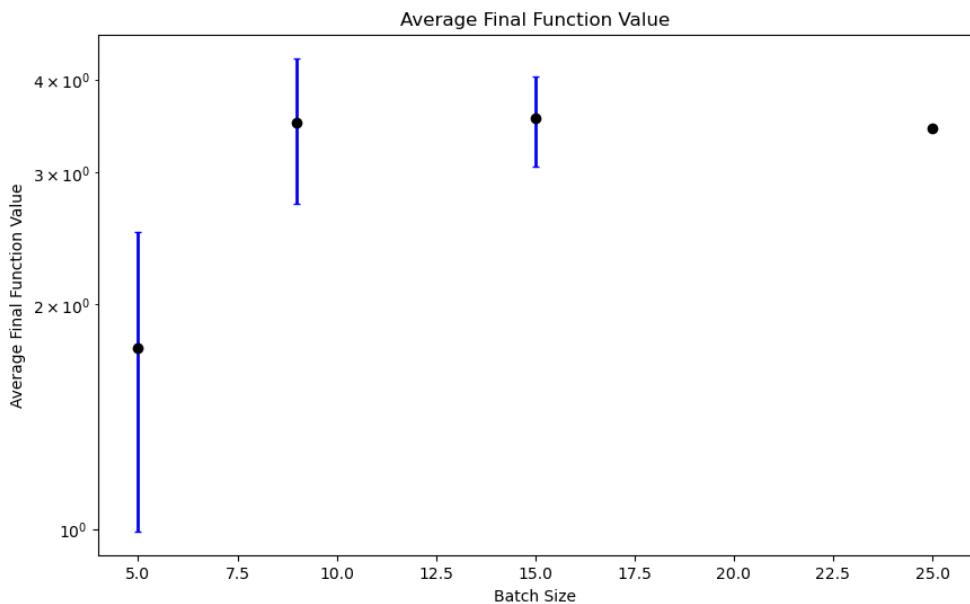
b) ii)



In this image, we can see the path taken by 6 different iterations of the same function. This is simply a SGD with a batch size of 5. The path taken here is the blue line, with the yellow being that of the gradient-descent calculated above. It can be seen that the blue lines seem to have far more points. This is due to x updating 5 times per epoch compared to the 1 as before. This leads to quicker convergence. The stochasticity can be seen by comparing the blue lines in each graph. The path taken to the minimum is slightly different in each iteration. This happens due to the batch used giving marginally different values for $f(x, N)$ and $f'(x, N)$.

b) iii)

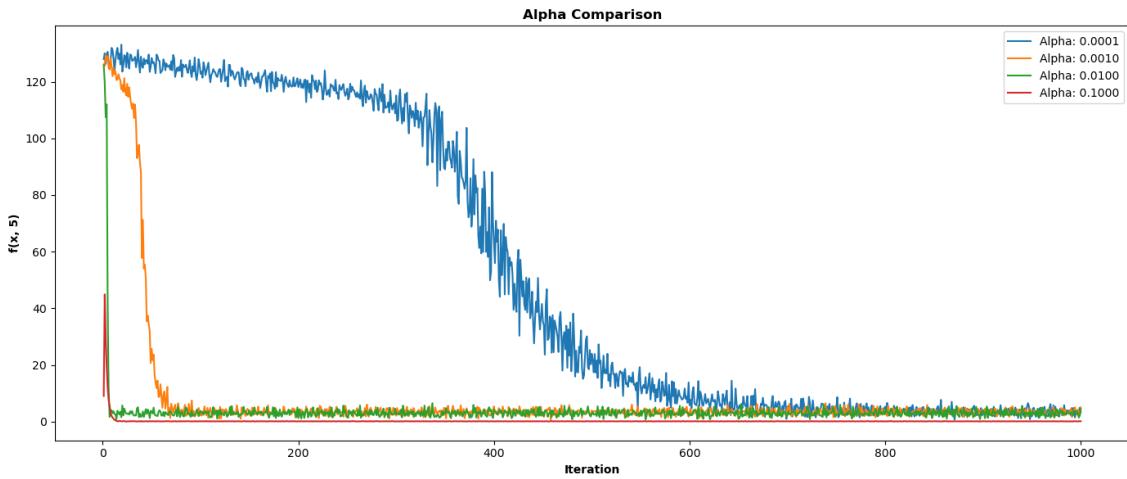
Using a fixed α of 0.01, I performed a few iterations of constant SGD for each batch_size. These returned the following



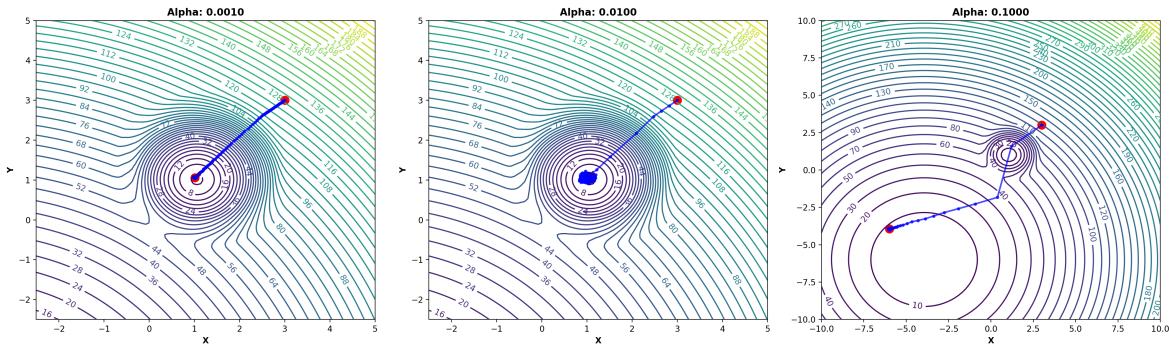
We can see that as the batch size increases, the noise decreases. This is simply an affect of the Law of Large numbers. The more samples used, the less variance is present. A trade-off of the higher variance is quicker convergence due to x updating after each iteration rather than each epoch. This stochasticity from different evaluating points leads to marginally different convergence points: batch_size: 5 finished at [0.9962, 0.9397]. batch_size: 9 finished at [0.9466, 0.9463]. batch_size: 15 finished at [1.0008, 0.9440]. batch_size: 25 finished at [1.0062, 0.9455].

b) iv)

I chose to perform analysis of $\alpha \in [0, 0.0001, 0.001, 0.01, 0.1]$. There were 2 noticeable differences in the results of varying alphas. The first was the speed of convergence. Higher alpha converged to a minimum far quicker than lower values. This is to be expected due to the bowl-shape of the loss function. We can see this in the plot below:



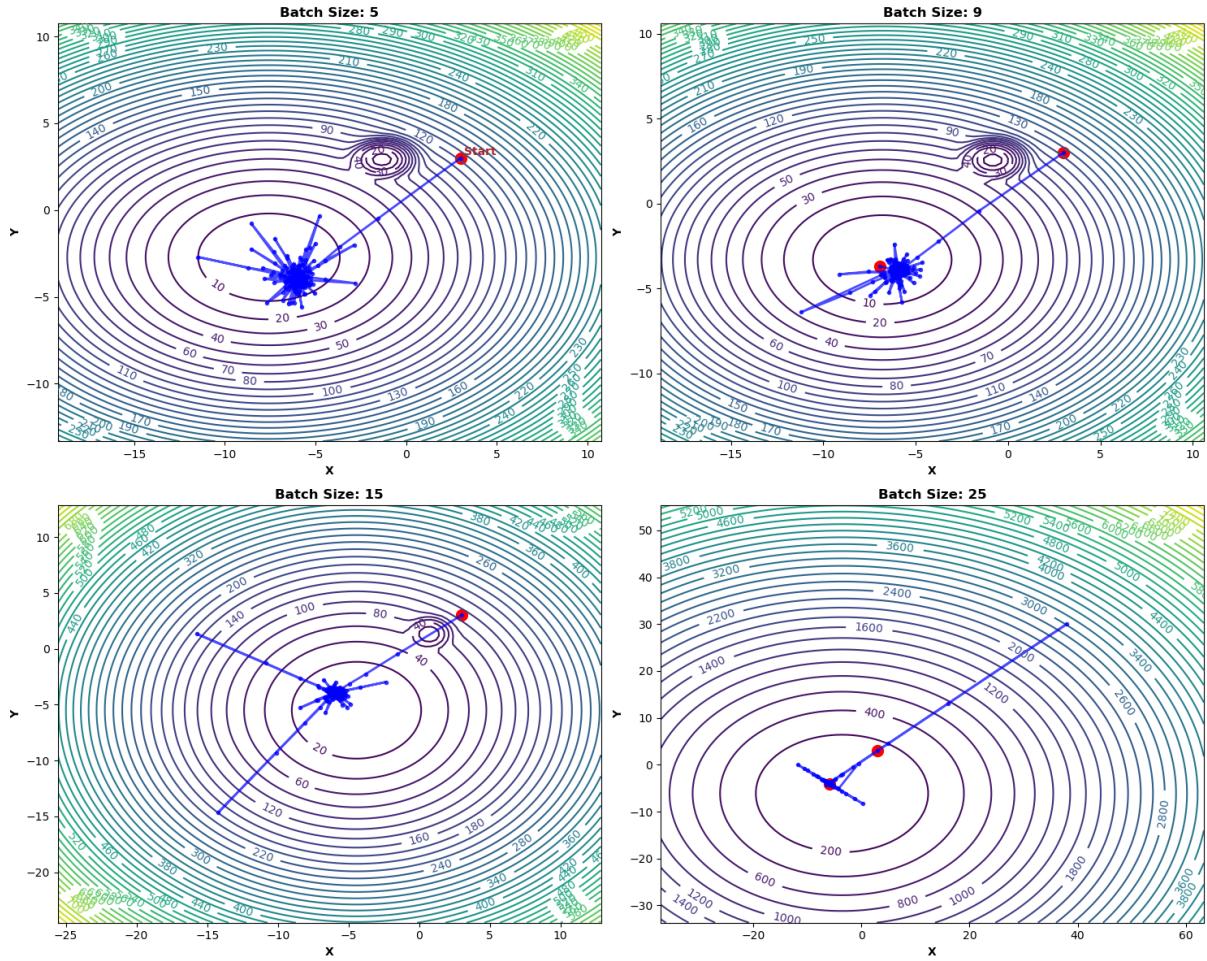
The second thing worth noting is the point of convergence of the different values of α . A larger value will lead the SGD function to overshoot the minimum in the valley and converge to the minimum value in the centre of the bowl, however, the smaller α values will settle in the valley. Although a larger step-size generally results in more noise, here this doesn't seem to be the case.



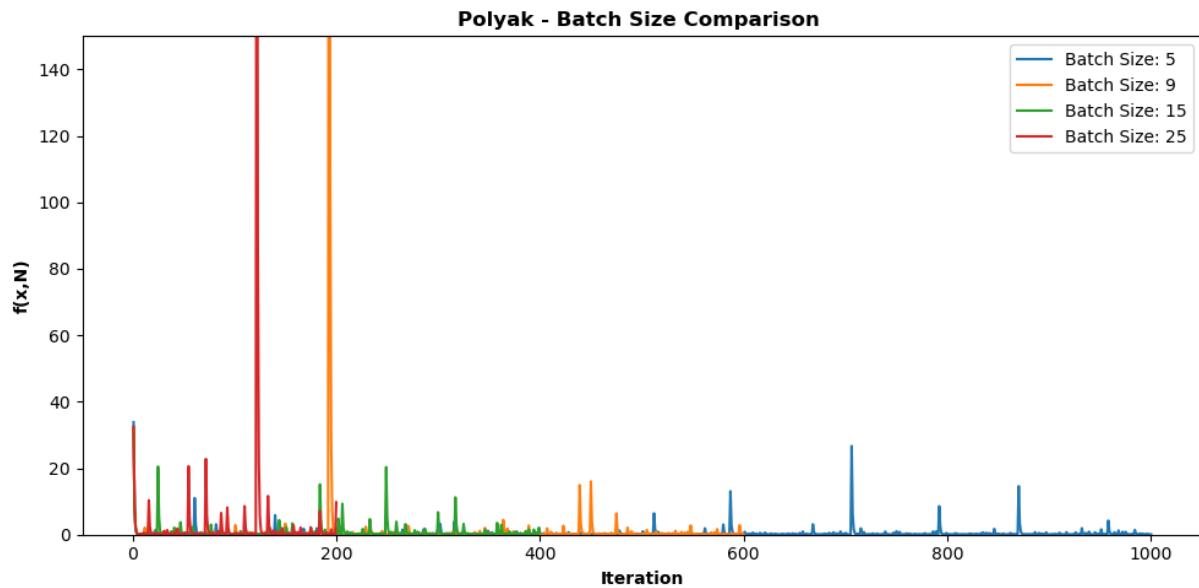
c) i)

Due to the large starting value of the function, the initial step-size overshoots the valley. Because of this, the polyak function converges to the minimum at the middle of the bowl-shaped surface.

The convergence of the polyak to this minimum is quite random with a lot of oscillation.



We can see that the stochasticity involved with a smaller batch size amplifies the oscillations although the increased number of point updates also contributes to this.

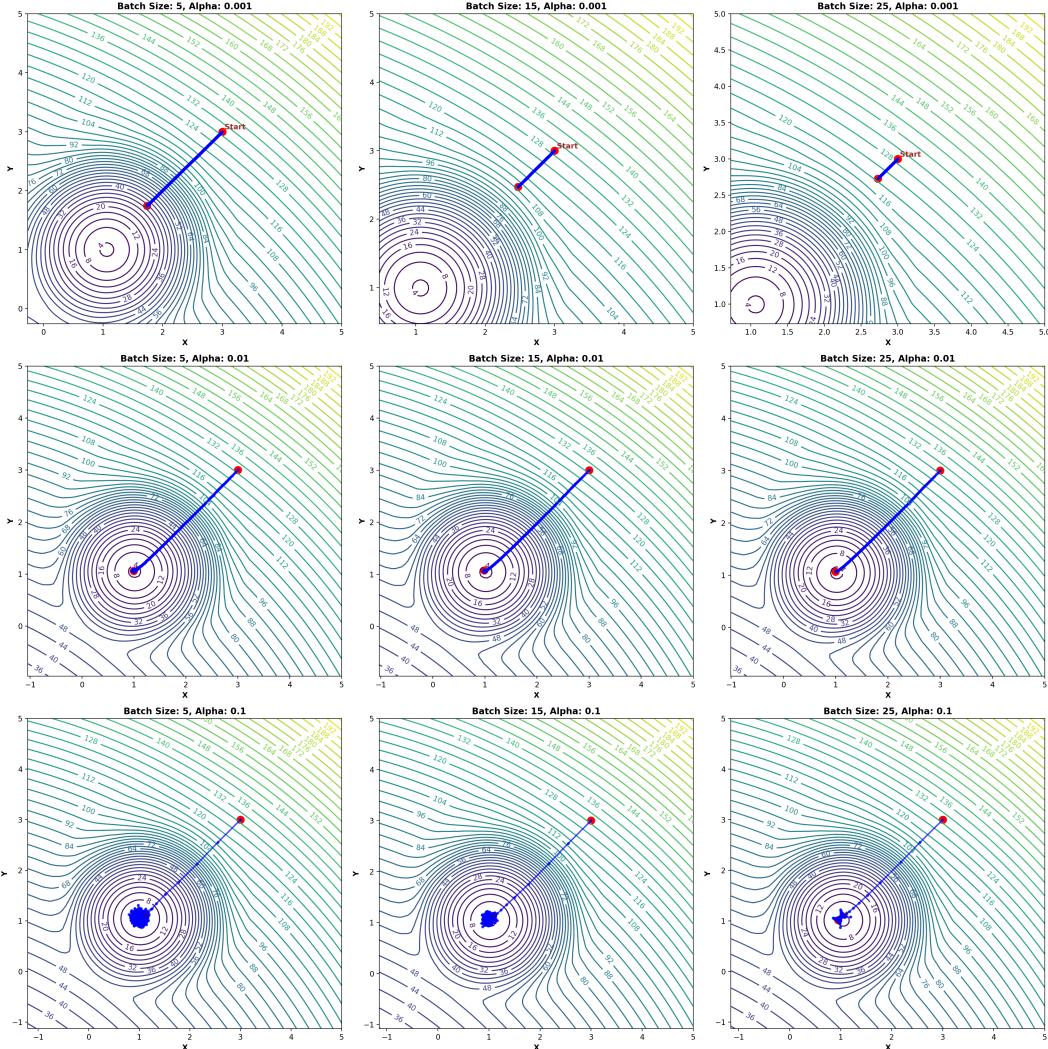


Although Polyak converges quickly, it is unstable due to the lack of momentum and therefore

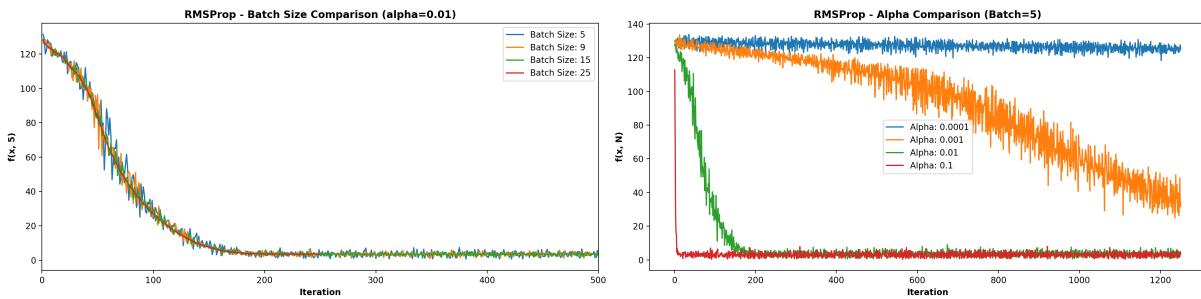
exposure to randomness in the SGD algorithm. The baseline algorithm from b performs better.

c) ii)

The RMSProp algorithm is more suited to SGD than Polyak. Since the gradient used to calculate α has components from past gradients, some of the randomness that comes with a smaller batch size is mitigated. I used a few different values of β but settled on 0.9.



We can see from these plots that again, a larger alpha converges faster, however it also comes with more oscillation around the minimum.

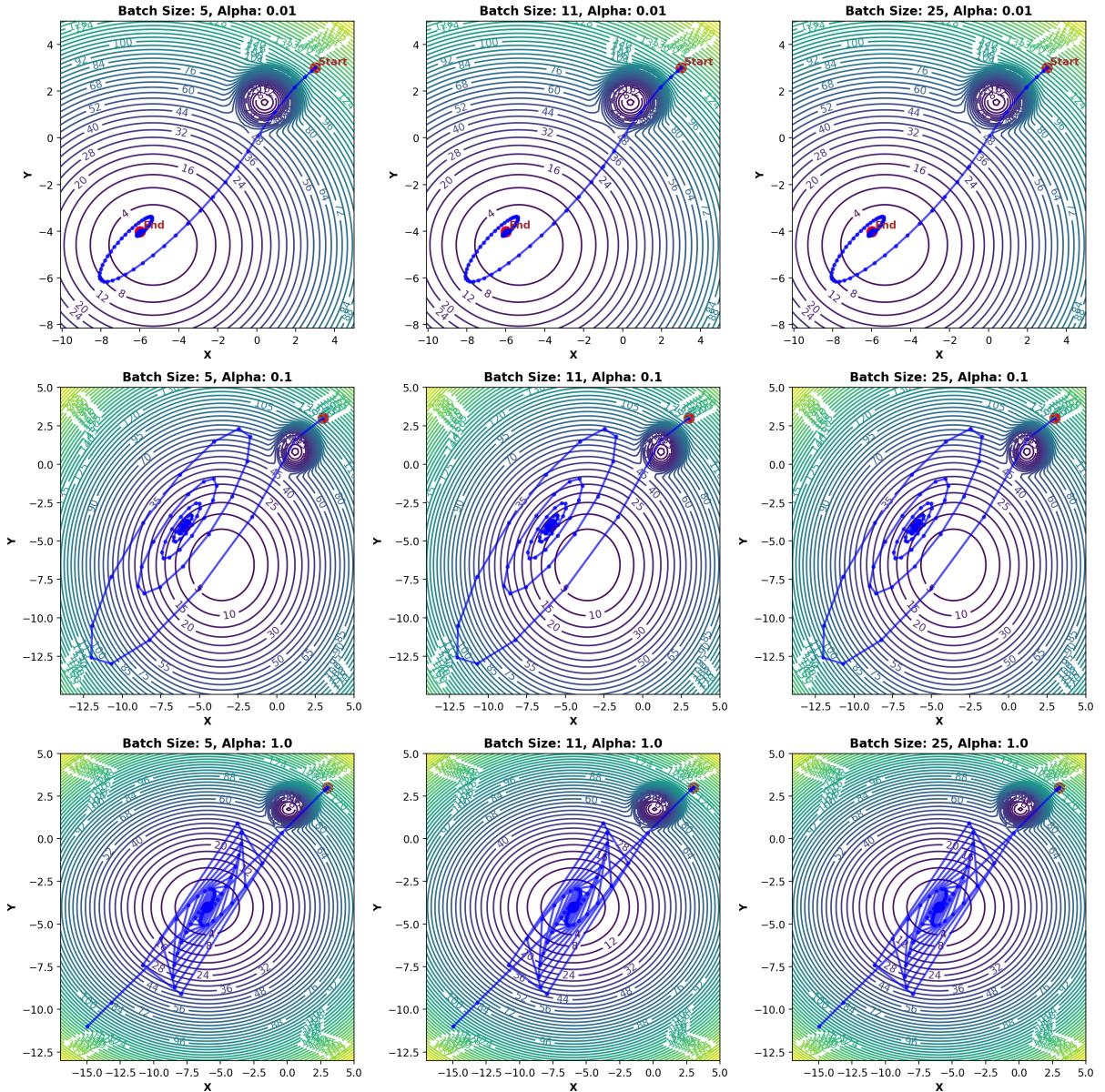


Looking at the different batch sizes, we can see that a lower batch size results in far quicker convergences, however, it also comes with increased noise. It is worth noting that in the plot of convergence with varying batch_size, the iterations for smaller batches are performed $\approx \frac{\text{data}}{\text{batch_size}}$ quicker than training-data. The baseline still appears to perform better

c) iii)

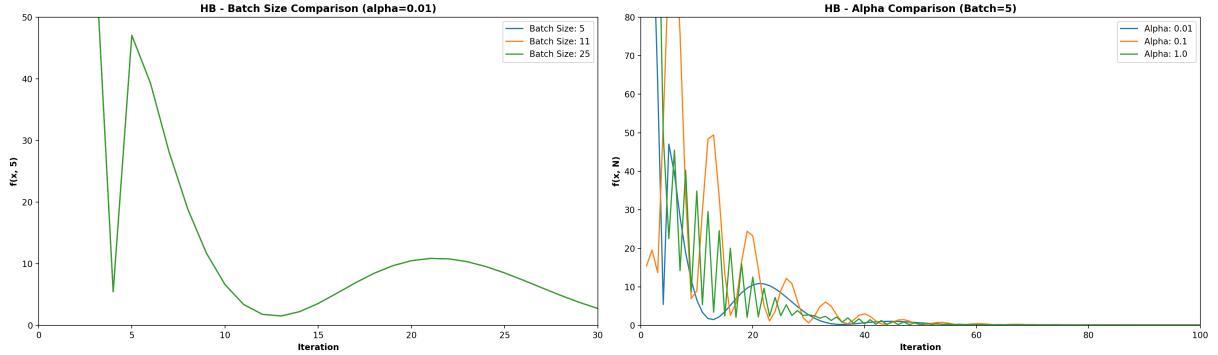
For the heavy-ball implementation, I chose a $\beta = 0.9$ as once again, it seemed to give the best results with everything else held constant. I chose to iterate over $\alpha \in [0.01, 0.1, 1]$ with batch sizes of 5, 11, 25.

Because heavy-ball has a momentum component, we can increase the learning rate alpha as long as the β value is kept relatively large.



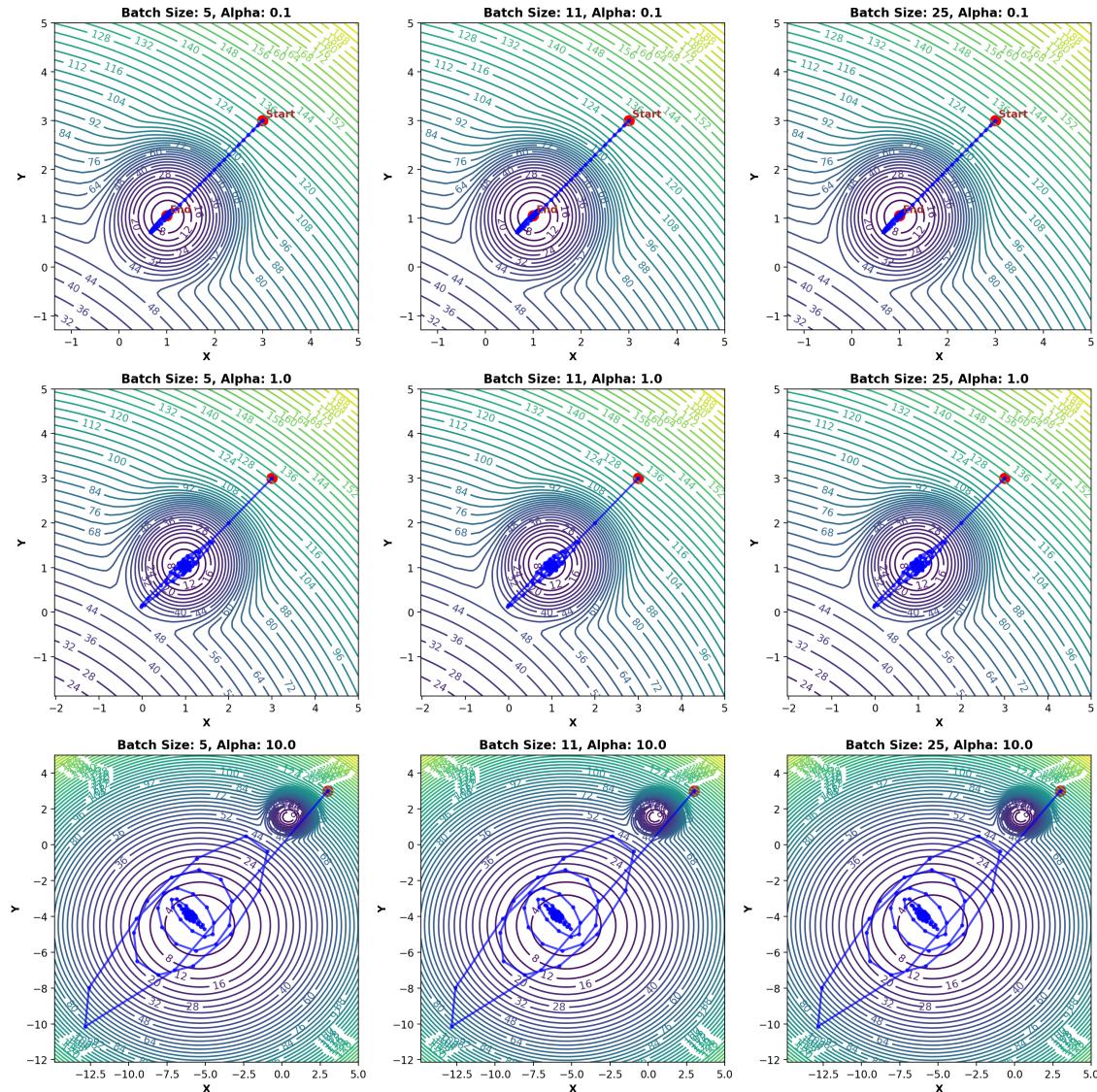
We can see that the momentum component for $\beta = 0.9$ means that the noise from different batch sizes is made almost irrelevant. In the left plot, there are 3 batch sizes however they

are almost indistinguishable. Increasing alpha increases the size of the oscillations to the point where convergence for each alpha is essentially the same.

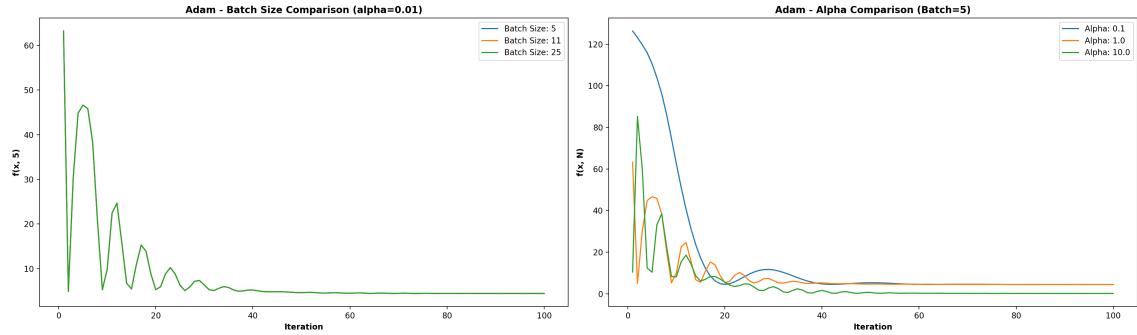


c) iv)

For the Adam algorithm, I chose $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Due to the momentum of this algorithm, it is not affected drastically by the noise associated with SGD



The speed of convergence becomes quicker with larger α , however, it also leads to further instability.



1 Code

```
1 import numpy as np
2
3 #Downloaded Functions
4 def generate_trainingdata(m=25):
5     return np.array([0,0])+0.25*np.random.randn(m,2)
6
7 def f(x, minibatch):
8     # loss function sum_{w in training data} f(x,w)
9     y=0; count=0
10    for w in minibatch:
11        z=x-w-1
12        y=y+min(31*(z[0]**2+z[1]**2), (z[0]+7)**2+(z[1]+5)**2)
13        count=count+1
14    return y/count
15
16
17
18 def mini_batch(data, batch_size):
19     batches = []
20     np.random.shuffle(data)
21     n = len(data)
22     for i in np.arange(0, n, batch_size):
23         batch = data[i:(i + batch_size)] if i + batch_size < n else
24             data[i:n]
25         batches.append(batch)
26
27
28 def batch_finite_diff(f, pt, batch, h=1e-8):
29     batch_dx = (f([pt[0] + h, pt[1]], batch) - f(pt, batch))/h
30     batch_dy = (f([pt[0], pt[1] + h], batch) - f(pt, batch))/h
31     return batch_dx, batch_dy
32
33
34 def constant(f, starting_point, data, alpha, tol, max_iters=1000,
35 batch_size=32):
36     x, y = starting_point[0], starting_point[1]
37     path=[(x, y)]
38     f_vals=[]
39
40     for _ in range(max_iters):
41         for batch in mini_batch(data, batch_size=batch_size):
42             dx, dy = batch_finite_diff(f, [x,y], batch, h=1e-8)
43             x -= alpha * dx
44             y -= alpha * dy
```

```

45     path.append((x,y))
46     f_vals.append(f([x, y], batch))
47
48     if np.sqrt(dx**2 + dy**2) < tol:
49         break
50
51 return path, f_vals
52
53
54
55
56 def Polyak(f, starting_point, data, epsilon=1e-8, f_star=0,
57             max_iters=1000, batch_size=32):
58     x, y = starting_point[0], starting_point[1]
59     path=[(x, y)]
60     f_vals=[]
61
62     for _ in range(max_iters):
63         for batch in mini_batch(data, batch_size=batch_size):
64             dx, dy = batch_finite_diff(f, [x,y], batch, h=1e-8)
65             alpha = (f([x, y], batch) - f_star)/(dx**2 + dy**2 +
66                                         epsilon)
67             x -= alpha * dx
68             y -= alpha * dy
69
70             path.append((x,y))
71             f_vals.append(f([x, y], batch))
72
73     return path, f_vals
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89

```

```

90     step_size_x = alpha0 / np.sqrt(sum_dx + epsilon)
91     step_size_y = alpha0 / np.sqrt(sum_dy + epsilon)
92
93     # Update x and y
94     x -= step_size_x * dx
95     y -= step_size_y * dy
96
97     path.append((x, y))
98     f_vals.append(f([x, y], batch))
99
100    return path, f_vals
101
102 def heavy_ball(f, starting_point, data, alpha, beta, max_iters=1000,
103 tol=1e-5, batch_size=32, stopping=True):
104     # Initialize x,y, momentum variables and path,f(x,y) lists
105     x, y = starting_point[0], starting_point[1]
106     ux, uy = 0, 0
107     path = [(x, y)]
108     f_vals = []
109
110     for _ in range(max_iters):
111         for batch in mini_batch(data, batch_size=batch_size):
112             # Take gradients for current x, y
113             dx, dy = batch_finite_diff(f, [x,y], batch, h=1e-8)
114
115             # Update momentum variable with current and past momentum
116             ux = beta * ux + alpha * dx
117             uy = beta * uy + alpha * dy
118
119             # Update x and y
120             x -= ux
121             y -= uy
122
123             # Store the new values
124             path.append((x, y))
125             f_vals.append(f([x, y], batch))
126
127             if f([x, y], batch) > 1e10:
128                 break
129
130             # Check if the norm of the gradient is below the tolerance
131             if np.sqrt(dx**2 + dy**2) < tol and stopping == True:
132                 break
133
134
135
136    return path, f_vals

```

```

137 def adam(f, starting_point, data, alpha, beta1, beta2, epsilon=1e-6,
138     max_iters=1000, tol=1e-5, batch_size=32,):
139     # Initialize x,y, momentum variables and path,f(x,y) lists
140     x, y = starting_point[0], starting_point[1]
141     m_x, m_y = 0, 0
142     v_x, v_y = 0, 0
143     path = [(x, y)]
144     f_vals = []
145
146     for t in range(1, max_iters + 1):
147         for batch in mini_batch(data, batch_size=batch_size):
148             # Take gradients for current x, y
149             dx, dy = batch_finite_diff(f, [x,y], batch, h=1e-8)
150
151             # Update momentum variable for gradient
152             m_x = beta1 * m_x + (1 - beta1) * dx
153             m_y = beta1 * m_y + (1 - beta1) * dy
154             # Update momentum variable for squared gradient
155             v_x = beta2 * v_x + (1 - beta2) * dx**2
156             v_y = beta2 * v_y + (1 - beta2) * dy**2
157             # Compute bias-corrected estimates
158             m_x_hat = m_x / (1 - beta1**t)
159             m_y_hat = m_y / (1 - beta1**t)
160
161             v_x_hat = v_x / (1 - beta2**t)
162             v_y_hat = v_y / (1 - beta2**t)
163
164             # Update variables
165             x -= alpha * m_x_hat / (np.sqrt(v_x_hat) + epsilon)
166             y -= alpha * m_y_hat / (np.sqrt(v_y_hat) + epsilon)
167
168             path.append((x, y))
169             f_vals.append(f([x, y], batch))
170             if f([x, y], batch) > 1e10:
171                 break
172
173     return path, f_vals
174
175 def const_GD(f, starting_point, alpha, training_data, max_iters=1000,
176     tol=1e-6):
177     x, y = starting_point[0], starting_point[1]
178     path=[(x, y)]
179     f_vals=[]
180     for _ in range(max_iters):
181         dx, dy = batch_finite_diff(f, [x,y], training_data, h=1e-8)
182         x -= alpha * dx

```

```

183     y -= alpha * dy
184
185     path.append((x,y))
186     f_vals.append(f([x, y], training_data))
187
188
189     return path, f_vals
190
191
192 def SGD(optimizer, f, starting_point, data, batch_size=32,
193         max_iters=1000, **kwargs):
194     optimizer = optimizer.strip().lower()
195     if optimizer not in ['constant', 'polyak', 'rmsprop', 'hb',
196                          'adam']:
197         raise ValueError("Optimizer argument not valid")
198
199     if optimizer == 'constant':
200         alpha = kwargs.get('alpha', 0.01)
201         tol = kwargs.get('tol', 1e-8)
202         path, f_vals = constant(f=f, starting_point=starting_point,
203                                 data=data, alpha=alpha, tol=tol, max_iters=max_iters,
204                                 batch_size=batch_size)
205         print("Stochastic-Gradient-Descent using Constant Optimiser")
206         return path, f_vals
207
208
209     elif optimizer == 'polyak':
210         f_star = kwargs.get('f_star', 0.0)
211         epsilon = kwargs.get('epsilon', 1e-8)
212
213         path, f_vals = Polyak(f=f, starting_point=starting_point,
214                               data=data, f_star=f_star, epsilon=epsilon,
215                               max_iters=max_iters, batch_size=batch_size)
216         print("Stochastic-Gradient-Descent using Polyak Optimiser")
217         return path, f_vals
218
219
220     elif optimizer == 'rmsprop':
221         alpha0 = kwargs.get('alpha0', 0.01)
222         beta = kwargs.get('beta', 0.9)
223         epsilon = kwargs.get('epsilon', 1e-8)
224
225         path, f_vals = RMSProp(f=f, starting_point=starting_point,
226                                data=data, alpha0=alpha0, beta=beta, epsilon=epsilon,
227                                max_iters=max_iters, batch_size=batch_size)
228         print("Stochastic-Gradient-Descent using RMSProp Optimiser")
229         return path, f_vals
230
231
232     elif optimizer == 'hb':
233         alpha = kwargs.get('alpha', 0.01)

```

```

223     beta = kwargs.get('beta', 0.9)
224     tol = kwargs.get('tol', 1e-5)
225
226     path, f_vals = heavy_ball(f=f, starting_point=starting_point,
227                               data=data, alpha=alpha, beta=beta, max_iters=max_iters,
228                               tol=tol, batch_size=32)
229     print("Stochastic-Gradient-Descent using Heavy-Ball Optimiser")
230     return path, f_vals
231
232
233
234
235
236
237
238
239
240
241

```

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from optimizer_functions import *
5
6 from IPython.display import clear_output

```

```

1 training_data = generate_trainingdata()
2 x = np.linspace(-7.5, 7.5, 100)
3 y = np.linspace(-7.5, 7.5, 100)
4 z = []
5 for i in x:
6     for j in y:
7         z.append(f([i,j], training_data))
8
9 z = np.array(z).reshape(100, 100)
10 x, y = np.meshgrid(x, y)
11
12 fig = plt.figure(figsize=(18, 7))
13
14 ax1 = fig.add_subplot(121, projection='3d')
15 ax1.plot_wireframe(x, y, z, color='blue')
16 ax1.set_title('Wireframe Plot', fontsize=18, fontweight='bold')

```

```

17 ax1.set_xlabel('X Axis', fontweight='bold')
18 ax1.set_ylabel('Y Axis', fontweight='bold')
19 ax1.set_zlabel('f(x, T)', fontweight='bold')
20
21
22 ax2 = fig.add_subplot(122)
23 contour = ax2.contour(x, y, z, 50)
24 plt.clabel(contour, inline=1, fontsize=10)
25 ax2.set_title('Contour Plot', fontsize=18, fontweight='bold')
26 ax2.set_xlabel('X Axis', fontweight='bold')
27 ax2.set_ylabel('Y Axis', fontweight='bold')
28
29 plt.show()

```

```

1 path, f_vals = SGD('constant', f, starting_point=[3,3],
2                     data=training_data, alpha=0.01, max_iters=100, tol=-0.1,
3                     batch_size=len(training_data))
4 path_x, path_y = zip(*path)
5 path_z = f_vals
6
7 training_data = generate_trainingdata()
8 x = np.linspace(-6, 6, 100)
9 y = np.linspace(-6, 6, 100)
10 z = []
11 for i in x:
12     for j in y:
13         z.append(f([i,j], training_data))
14
15 z = np.array(z).reshape(100, 100)
16 x, y = np.meshgrid(x, y)
17
18 fig = plt.figure(figsize=(14, 7))
19
20 ax1 = fig.add_subplot(121, projection='3d')
21 ax1.plot_wireframe(x, y, z, color='blue', alpha=0.5)
22 ax1.scatter(path_x[0], path_y[0], path_z[0], color='green', s=50)
23 ax1.plot(path_x[0:-1], path_y[0:-1], path_z, color='red', linewidth=3,
24             marker='o', markersize=5)
25 ax1.scatter(path_x[-1], path_y[-1], path_z[-1], color='black', s=50)
26 ax1.set_title('Wireframe Plot', fontsize=18, fontweight='bold')
27 ax1.set_xlabel('X Axis', fontweight='bold')
28 ax1.set_ylabel('Y Axis', fontweight='bold')
29 ax1.set_zlabel('f(x, T)', fontweight='bold')
30
31 ax2 = fig.add_subplot(122)
32 contour = ax2.contour(x, y, z, 50, cmap='viridis')
33 ax2.plot(path_x, path_y, color='blue', linewidth=3, marker='o',

```

```

    markersize=5)
32 ax2.scatter(path_x[0], path_y[0], color='brown', s=100)
33 ax2.scatter(path_x[-1], path_y[-1], color='brown', s=100)
34 ax2.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3, 3),
35             textcoords='offset points', color='brown', weight='bold')
36 ax2.annotate('End', xy=(path_x[-1], path_y[-1]), xytext=(-30, -30),
37             textcoords='offset points', color='brown', weight='bold')
38 plt.clabel(contour, inline=1, fontsize=10)
39 ax2.set_title('Contour Plot', fontsize=18, fontweight='bold')
40 ax2.set_xlabel('X Axis', fontweight='bold')
41 ax2.set_ylabel('Y Axis', fontweight='bold')
42
43 plt.show()

```

```

1 x = np.linspace(0, 4, 100)
2 y = np.linspace(0, 4, 100)
3 z = []
4 for i in x:
5     for j in y:
6         z.append(f([i,j], training_data))
7
8 z = np.array(z).reshape(100, 100)
9 x, y = np.meshgrid(x, y)
10
11 final_points = {i:[0.0,0.0] for i in range(1,7)}
12
13 fig, axes = plt.subplots(2, 3, figsize=(18, 10))
14
15 starting_point = [3, 3]
16 alpha = 0.01
17 batch_size = 5
18 max_iters = 100
19
20 for i in range(2):
21     for j in range(3):
22         ax = axes[i, j]
23         path, f_vals = SGD('constant', f, starting_point=[3,3],
24                             data=training_data, alpha=alpha, max_iters=max_iters,
25                             tol=1e-4, batch_size=batch_size)
26         true_path, true_f_vals = SGD('constant', f,
27                                       starting_point=[3,3], data=training_data, alpha=0.01,
28                                       max_iters=100, tol=-0.1, batch_size=len(training_data))
29
30         path_x, path_y = zip(*path)
31         true_path_x, true_path_y = zip(*true_path)
32
33         final_points[3 * i + j + 1][0] += path_x[-1]

```

```

31     final_points[3 * i + j + 1][1] += path_y[-1]
32
33
34     contour = ax.contour(x, y, z, 50, cmap='viridis')
35     ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
36             markersize=3, alpha=1)
37     ax.plot(true_path_x, true_path_y, color='yellow', linewidth=2,
38             marker='o', markersize=3, alpha=0.7)
39
40     ax.scatter(path_x[0], path_y[0], color='red', s=100)
41     if i == 0 and j == 0:
42         ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3,
43                               3),
44                     textcoords='offset points', color='brown',
45                     weight='bold')
46
47
48     ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
49     ax.clabel(contour, inline=1, fontsize=10)
50     ax.set_title(f'SGD Iteration: {3 * i + j + 1}', fontweight='bold')
51     ax.set_xlabel('X Axis')
52     ax.set_ylabel('Y Axis')
53
54 plt.tight_layout()
55 plt.show()

56
57 for k, v in final_points.items():
58     print(f"Iteration: {k} finished at [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1 x = np.linspace(0, 4, 100)
2 y = np.linspace(0, 4, 100)
3 z = []
4 for i in x:
5     for j in y:
6         z.append(f([i,j], training_data))
7
8 z = np.array(z).reshape(100, 100)
9 x, y = np.meshgrid(x, y)
10
11 final_points = {i:[0.0,0.0] for i in range(1,7)}
12
13 fig, axes = plt.subplots(2, 3, figsize=(18, 10))
14
15 starting_point = [3, 3]
16 alpha = 0.01
17 batch_size = 5

```

```

18 max_iters = 100
19
20 for i in range(2):
21     for j in range(3):
22         ax = axes[i, j]
23         path, f_vals = SGD('constant', f, starting_point=[3,3],
24                             data=training_data, alpha=alpha, max_iters=max_iters,
25                             tol=1e-4, batch_size=batch_size)
26         true_path, true_f_vals = SGD('constant', f,
27                                       starting_point=[3,3], data=training_data, alpha=0.01,
28                                       max_iters=100, tol=-0.1, batch_size=len(training_data))
29
30         path_x, path_y = zip(*path)
31         true_path_x, true_path_y = zip(*true_path)
32
33         final_points[3 * i + j + 1][0] += path_x[-1]
34         final_points[3 * i + j + 1][1] += path_y[-1]
35
36         contour = ax.contour(x, y, z, 50, cmap='viridis')
37         ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
38                 markersize=3, alpha=1)
39         ax.plot(true_path_x, true_path_y, color='yellow', linewidth=2,
40                 marker='o', markersize=3, alpha=0.7)
41
42         ax.scatter(path_x[0], path_y[0], color='red', s=100)
43         if i == 0 and j == 0:
44             ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3,
45                         3),
46                         textcoords='offset points', color='brown',
47                         weight='bold')
48
49         ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
50         ax.clabel(contour, inline=1, fontsize=10)
51         ax.set_title(f'SGD Iteration: {3 * i + j + 1}', fontweight='bold')
52         ax.set_xlabel('X Axis')
53         ax.set_ylabel('Y Axis')
54
55 plt.tight_layout()
56 plt.show()
57
58 for k, v in final_points.items():
59     print(f"Iteration: {k} finished at [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1      starting_point = [3, 3]
2      alpha = 0.01
3      batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=4,
4          base=10).astype(int)
5
6      fig, axes = plt.subplots(4, 3, figsize=(15, 20))
7
8      final_points = {i.astype(int):[0.0, 0.0] for i in batch_sizes}
9
10     for i in range(4):
11         for j in range(3):
12             ax = axes[i, j]
13             batch_size = batch_sizes[i]
14
15             path, f_vals = SGD('constant', f, starting_point,
16                                 data=training_data,
17                                 alpha=alpha, max_iters=100, tol=1e-6,
18                                 batch_size=batch_size)
19
20             path_x, path_y = zip(*path)
21             path_z = f_vals
22             final_points[batch_size][0] += path_x[-1]/3
23             final_points[batch_size][1] += path_y[-1]/3
24
25             contour = ax.contour(x, y, z, 50, cmap='viridis')
26             ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
27                     markersize=3, alpha=0.7)
28             ax.scatter(path_x[0], path_y[0], color='red', s=100)
29             if i == 0 and j == 0:
30                 ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3,
31                                         3),
32                             textcoords='offset points', color='brown',
33                             weight='bold')
34
35             ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
36             ax.clabel(contour, inline=1, fontsize=10)
37             ax.set_title(f'Batch Size: {batch_size}, Iteration: {j+1}')
38             ax.set_xlabel('X')
39             ax.set_ylabel('Y')
40
41             if i == 0 and j == 0:
42                 ax.legend()

```

```

42 plt.tight_layout()
43 plt.show()
44
45 for k, v in final_points.items():
46     print(f"batch_size: {k} finished at [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1      starting_point = [3, 3]
2 alpha = 0.01
3 iterations = 5
4 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=4,
5   base=10).astype(int)
6
7 final_f_vals = {batch_size: [] for batch_size in batch_sizes}
8
9 for batch_size in batch_sizes:
10    for _ in range(iterations):
11        path, f_vals = SGD('constant', f, starting_point,
12          data=training_data,
13          alpha=alpha, max_iters=100, tol=1e-6,
14          batch_size=batch_size)
15        final_f_vals[batch_size].append(f_vals[-1])
16 avg_f_vals = {batch_size: np.mean(vals) for batch_size, vals in
17   final_f_vals.items()}
18 std_f_vals = {batch_size: np.std(vals) for batch_size, vals in
19   final_f_vals.items()}
20
21 fig, ax = plt.subplots(figsize=(10, 6))
22
23 batch_sizes_list = list(batch_sizes)
24 avg_vals_list = [avg_f_vals[bs] for bs in batch_sizes_list]
25 std_vals_list = [std_f_vals[bs] for bs in batch_sizes_list]
26
27 ax.errorbar(batch_sizes_list, avg_vals_list, yerr=std_vals_list,
28   fmt='o', color='black', ecolor='blue', elinewidth=2, capsize=2)
29
30 ax.set_title('Average Final Function Value')
31 ax.set_xlabel('Batch Size')
32 ax.set_ylabel('Average Final Function Value')
33
34 plt.show()

```

```

1 starting_point = [3, 3]
2 alphas = np.logspace(-3, -1, num=3, base=10)
3 batch_size = 5
4 max_iters = 200
5
6 fig, axes = plt.subplots(1, 3, figsize=(20, 6), dpi=200)
7

```

```

8  final_points = {alpha: [0.0, 0.0] for alpha in alphas}
9
10 for i, alpha in enumerate(alphas):
11     if alpha < 0.05:
12         x_range = np.linspace(-2.5, 5, 100)
13         y_range = np.linspace(-2.5, 5, 100)
14     else:
15         x_range = np.linspace(-10, 10, 100)
16         y_range = np.linspace(-10, 10, 100)
17
18     z = [f([i, j], training_data) for i in x_range for j in y_range]
19     z = np.array(z).reshape(100, 100)
20     x, y = np.meshgrid(x_range, y_range)
21
22     ax = axes[i]
23     path, f_vals = SGD('constant', f, starting_point,
24                          data=training_data,
25                          alpha=alpha, max_iters=max_iters, tol=1e-6,
26                          batch_size=batch_size)
27     path_x, path_y = zip(*path)
28     final_points[alpha][0] += path_x[-1]
29     final_points[alpha][1] += path_y[-1]
30
31     contour = ax.contour(x, y, z, 50, cmap='viridis')
32     ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
33              markersize=3, alpha=0.7)
34     ax.scatter(path_x[0], path_y[0], color='red', s=100)
35     if i == 0 and j == 0:
36         ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3, 3),
37                     textcoords='offset points', color='brown',
38                     weight='bold')
39
40     ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
41     ax.clabel(contour, inline=1, fontsize=10)
42     ax.set_title(f'Alpha: {alpha:.4f}', fontweight='bold')
43     ax.set_xlabel('X', fontweight='bold')
44     ax.set_ylabel('Y', fontweight='bold')
45
46     plt.tight_layout()
47     plt.show()

48 for k, v in final_points.items():
49     print(f"Alpha: {k}, finished at [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1 starting_point = [3, 3]
2 alphas = np.logspace(-4, -1, num=4, base=10)
3 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=4,
4                           base=10).astype(int)

```

```

4 max_iters = 200
5
6 fig, axs = plt.subplots(1, 1, figsize=(14, 6))
7
8 for alpha in alphas:
9     _, f_vals = SGD('constant', f, starting_point, data=training_data,
10                     alpha=alpha, max_iters=max_iters, tol=1e-6,
11                     batch_size=5)
12     iterations = list(range(1, len(f_vals) + 1))
13     axs.plot(iterations, f_vals, label=f'Alpha: {alpha:.4f}')
14
15 axs.set_title('Alpha Comparison', fontweight='bold')
16 axs.set_xlabel('Iteration', fontweight='bold')
17 axs.set_ylabel('f(x, 5)', fontweight='bold')
18 axs.legend()
19
20 plt.tight_layout()
21 plt.show()

```

```

1 training_data = generate_trainingdata()
2
3 starting_point = [3, 3]
4 alpha = 0.01
5 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=4,
6                           base=10).astype(int)
7 max_iters = 100
8
9 fig, axes = plt.subplots(2, 2, figsize=(15, 12))
10 axes = axes.flatten()
11
12 final_points = {batch: [0.0, 0.0] for batch in batch_sizes}
13
14 for i, batch_size in enumerate(batch_sizes):
15     ax = axes[i]
16     path, f_vals = SGD('polyak', f=f, starting_point=starting_point,
17                         data=training_data, epsilon=1e-6, max_iters=max_iters,
18                         batch_size=batch_size)
19     path_x, path_y = zip(*path)
20     final_points[batch_size][0] += path_x[-1]
21     final_points[batch_size][1] += path_y[-1]
22
23     size = max((max(path_x) - min(path_x)), (max(path_y) -
24                 min(path_y))) + 1
25
26     x = np.linspace(min(path_x) - size / 2, max(path_x) + size / 2,
27                     100)
28     y = np.linspace(min(path_y) - size / 2, max(path_y) + size / 2,
29

```

```
100)
24 z = [f([a, b], training_data) for a in x for b in y]
25 z = np.array(z).reshape(100, 100)
26 x, y = np.meshgrid(x, y)
27
28 contour = ax.contour(x, y, z, 50, cmap='viridis')
29 ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
30         markersize=3, alpha=0.7)
31 ax.scatter(path_x[0], path_y[0], color='red', s=100)
32
33 ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3, 3),
34             textcoords='offset points', color='brown',
35             weight='bold')
36
37 ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
38 ax.clabel(contour, inline=1, fontsize=10)
39 ax.set_title(f'Batch Size: {batch_size}', fontweight='bold')
40 ax.set_xlabel('X', fontweight='bold')
41 ax.set_ylabel('Y', fontweight='bold')
42
43 plt.tight_layout()
44 plt.show()
45
46 for k, v in final_points.items():
47     print(f"Batch size: {k}, finished at [{v[0]:0.4f}, {v[1]:0.4f}]")
```

```

1 starting_point = [3, 3]
2 alphas = np.logspace(-4, -1, num=4, base=10)
3 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=4,
4                           base=10).astype(int)
5 max_iters = 200
6
7
8
9 for batch_size in batch_sizes:
10    _, f_vals = SGD('polyak', f=f, starting_point=starting_point,
11                      data=training_data, epsilon=1e-6, max_iters=max_iters,
12                      batch_size=batch_size)
13    iterations = list(range(len(f_vals)))
14    plt.plot(iterations, f_vals, label=f'Batch Size: {batch_size}')
15
16 plt.title('Polyak - Batch Size Comparison ', fontweight='bold')
17 plt.xlabel('Iteration', fontweight='bold')
18 plt.ylim(0, 150)
19 plt.ylabel('f(x, N)', fontweight='bold')
20 plt.legend()

```

```
20 plt.tight_layout()
21 plt.show()
```

```
1 training_data = generate_trainingdata()
2
3 starting_point = [3, 3]
4 alphas = np.logspace(-3, -1, num=3, base=10)
5 beta = 0.95
6
7 batch_sizes = np.linspace(5, 25, 3).astype(int)
8 max_iters = 250
9
10 fig, axes = plt.subplots(3, 3, figsize=(20, 20), dpi=200)
11
12 final_points = {batch: [0.0, 0.0] for batch in batch_sizes}
13
14 for j, batch_size in enumerate(batch_sizes):
15     for i, alpha0 in enumerate(alphas):
16         ax = axes[i, j]
17         path, f_vals = SGD('rmsprop', f=f,
18                             starting_point=starting_point, data=training_data,
19                             alpha0=alpha0, beta=beta, epsilon=1e-6,
20                             max_iters=max_iters, batch_size=batch_size)
21         path_x, path_y = zip(*path)
22         final_points[batch_size][0] += path_x[-1]
23         final_points[batch_size][1] += path_y[-1]
24
25         size = max((max(path_x) - min(path_x)), (max(path_y) -
26                     min(path_y))) + 2
27
28         x = np.linspace(min(path_x)-2, max(path_x)+2, 100)
29         y = np.linspace(min(path_y)-2, max(path_y)+2, 100)
30         z = [f([a, b], training_data) for a in x for b in y]
31         z = np.array(z).reshape(100, 100)
32         x, y = np.meshgrid(x, y)
33
34         contour = ax.contour(x, y, z, 50, cmap='viridis')
35         ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
36                 markersize=3, alpha=0.7)
37         ax.scatter(path_x[0], path_y[0], color='red', s=100)
38
39         if i == 0:
40             ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3,
41                         3),
42                         textcoords='offset points', color='brown',
43                         weight='bold')
44
45         ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
```

```

39     ax.clabel(contour, inline=1, fontsize=10)
40     ax.set_title(f'Batch Size: {batch_size}, Alpha: {alpha0}', fontweight='bold')
41     ax.set_xlabel('X', fontweight='bold')
42     ax.set_ylabel('Y', fontweight='bold')
43
44 plt.tight_layout()
45 plt.show()
46
47 for k, v in final_points.items():
48     print(f"Batch size: {k}, finished at [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1 starting_point = [3, 3]
2 alphas = np.logspace(-4,-1, num=4, base=10)
3 beta = 0.95
4 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=4,
5     base=10).astype(int)
max_iters = 250
6
7
8 fig, axs = plt.subplots(1, 2, figsize=(20, 5), dpi=200)
9
10
11 for batch_size in batch_sizes:
12     _, f_vals = SGD('rmsprop', f=f, starting_point=starting_point,
13         data=training_data, alpha0=0.01, beta=beta, epsilon=1e-6,
14         max_iters=max_iters, batch_size=batch_size)
15     iterations = list(range(1, len(f_vals) + 1))
16     axs[0].plot(iterations, f_vals, label=f'Batch Size: {batch_size}')
17
18     axs[0].set_title('RMSProp - Batch Size Comparison (alpha=0.01)', fontweight='bold')
19     axs[0].set_xlabel('Iteration', fontweight='bold')
20     axs[0].set_xlim(0, 500)
21     axs[0].set_ylabel('f(x, 5)', fontweight='bold')
22     axs[0].legend()
23
24 for alpha in alphas:
25     _, f_vals = SGD('rmsprop', f=f, starting_point=starting_point,
26         data=training_data, alpha0=alpha, beta=beta, epsilon=1e-6,
27         max_iters=max_iters, batch_size=5)
28     iterations = list(range(1, len(f_vals) + 1))
29     axs[1].plot(iterations, f_vals, label=f'Alpha: {alpha}')
30
31     axs[1].set_title('RMSProp - Alpha Comparison (Batch=5)', fontweight='bold')
32     axs[1].set_xlabel('Iteration', fontweight='bold')

```

```

30     axs[1].set_ylabel('f(x, N)', fontweight='bold')
31     axs[1].legend()
32
33 plt.tight_layout()
34 plt.show()

```

```

1 training_data = generate_trainingdata()
2
3 starting_point = [3, 3]
4 alphas = np.logspace(-2, 0, num=3, base=10)
5 beta = 0.9
6
7 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=3,
   base=10).astype(int)
8 max_iters = 250
9
10 fig, axes = plt.subplots(3, 3, figsize=(15, 15), dpi=200)
11
12 final_points = {batch: {
13     alpha: [0.0, 0.0] for alpha in alphas
14 } for batch in batch_sizes}
15
16 for j, batch_size in enumerate(batch_sizes):
17     for i, alpha in enumerate(alphas):
18         ax = axes[i, j]
19         path, f_vals = SGD('hb', f=f, starting_point=starting_point,
20             data=training_data, alpha=alpha, beta=beta, epsilon=1e-6,
21             max_iters=max_iters, tol=-0.1, batch_size=batch_size)
22         path_x, path_y = zip(*path)
23         final_points[batch_size][alpha][0] = path_x[-1]
24         final_points[batch_size][alpha][1] = path_y[-1]
25
26         size = max((max(path_x) - min(path_x)), (max(path_y) -
27             min(path_y))) + 2
28
29         x = np.linspace(min(path_x)-2, max(path_x)+2, 100)
30         y = np.linspace(min(path_y)-2, max(path_y)+2, 100)
31         z = [f([a, b], training_data) for a in x for b in y]
32         z = np.array(z).reshape(100, 100)
33         x, y = np.meshgrid(x, y)
34
35         contour = ax.contour(x, y, z, 50, cmap='viridis')
36         ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
37             markersize=3, alpha=0.7)
38         ax.scatter(path_x[0], path_y[0], color='red', s=100)
39
40         if i == 0:
41             ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3,

```

```

3),
38             textcoords='offset points', color='brown',
39             weight='bold')
40 ax.annotate('End', xy=(path_x[-1], path_y[-1]), xytext=(3,
41             3),
42             textcoords='offset points', color='brown',
43             weight='bold')

44     ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
45     ax.clabel(contour, inline=1, fontsize=10)
46     ax.set_title(f'Batch Size: {batch_size}, Alpha: {alpha}',
47                 fontweight='bold')
48     ax.set_xlabel('X', fontweight='bold')
49     ax.set_ylabel('Y', fontweight='bold')

50 plt.tight_layout()
51 plt.show()

52 for k, a in final_points.items():
53     for x, v in a.items():
54         print(f"Batch size: {k}, Alpha: {x}\t finished at
55             [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1 starting_point = [3, 3]
2 alphas = np.logspace(-2, 0, num=3, base=10)
3 beta = 0.9
4 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=3,
5     base=10).astype(int)
6 max_iters = 250
7
8 fig, axs = plt.subplots(1, 2, figsize=(20, 6), dpi=200)
9
10 for batch_size in batch_sizes:
11     _, f_vals = SGD('hb', f=f, starting_point=starting_point,
12                     data=training_data, alpha=0.01, beta=beta, epsilon=1e-6,
13                     max_iters=max_iters, tol=-0.1, batch_size=batch_size)
14     iterations = list(range(1, len(f_vals) + 1))
15     axs[0].plot(iterations, f_vals, label=f'Batch Size: {batch_size}')
16
17 axs[0].set_title('HB - Batch Size Comparison (alpha=0.01)',
18                 fontweight='bold')
19 axs[0].set_xlabel('Iteration', fontweight='bold')
20 axs[0].set_xlim(0, 30)
21 axs[0].set_ylim(0, 50)
22 axs[0].set_ylabel('f(x, 5)', fontweight='bold')
23 axs[0].legend()
24
25 for alpha in alphas:
26     _, f_vals = SGD('hb', f=f, starting_point=starting_point,
27                     data=training_data, alpha=alpha, beta=beta, epsilon=1e-6,
28                     max_iters=max_iters, tol=-0.1, batch_size=batch_size)
29     iterations = list(range(1, len(f_vals) + 1))
30     axs[1].plot(iterations, f_vals, label=f'Alpha: {alpha}')
31
32 axs[1].set_title('HB - Batch Size Comparison (batch_size=16)')
33 axs[1].set_xlabel('Iteration', fontweight='bold')
34 axs[1].set_xlim(0, 30)
35 axs[1].set_ylim(0, 50)
36 axs[1].set_ylabel('f(x, 5)', fontweight='bold')
37 axs[1].legend()
38
39 for i in range(2):
40     plt.subplot(2, 1, i+1)
41     plt.grid(True)
42     plt.title(f'Batch Size Comparison (alpha={alphas[i]})')
43     plt.xlabel('Iteration', fontweight='bold')
44     plt.ylabel('f(x, 5)', fontweight='bold')
45     plt.legend()
46
47 plt.tight_layout()
48 plt.show()

```

```

22     _, f_vals = SGD('hb',f=f, starting_point=starting_point,
23                         data=training_data, alpha=alpha, beta=beta, epsilon=1e-6,
24                         max_iters=max_iters, tol=-0.1, batch_size=5)
25     iterations = list(range(1, len(f_vals) + 1))
26     axs[1].plot(iterations, f_vals, label=f'Alpha: {alpha}')
27
28     axs[1].set_title('HB - Alpha Comparison (Batch=5)', fontweight='bold')
29     axs[1].set_xlabel('Iteration', fontweight='bold')
30     axs[1].set_ylabel('f(x, N)', fontweight='bold')
31     axs[1].set_xlim(0,100)
32     axs[1].set_ylim(0,80)
33     axs[1].legend()
34
35 plt.tight_layout()
36 plt.show()

```

```

1 starting_point = [3, 3]
2 alphas = np.logspace(-2, 0, num=3, base=10)
3 beta = 0.9
4 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=3,
5                           base=10).astype(int)
6 max_iters = 250
7
8 fig, axs = plt.subplots(1, 2, figsize=(20, 6), dpi=200)
9
10 for batch_size in batch_sizes:
11     _, f_vals = SGD('hb',f=f, starting_point=starting_point,
12                       data=training_data, alpha=0.01, beta=beta, epsilon=1e-6,
13                       max_iters=max_iters, tol=-0.1, batch_size=batch_size)
14     iterations = list(range(1, len(f_vals) + 1))
15     axs[0].plot(iterations, f_vals, label=f'Batch Size: {batch_size}')
16
17     axs[0].set_title('HB - Batch Size Comparison (alpha=0.01)',
18                      fontweight='bold')
19     axs[0].set_xlabel('Iteration', fontweight='bold')
20     axs[0].set_xlim(0,30)
21     axs[0].set_ylim(0,50)
22     axs[0].set_ylabel('f(x, 5)', fontweight='bold')
23     axs[0].legend()
24
25 for alpha in alphas:
26     _, f_vals = SGD('hb',f=f, starting_point=starting_point,
27                       data=training_data, alpha=alpha, beta=beta, epsilon=1e-6,
28                       max_iters=max_iters, tol=-0.1, batch_size=5)
29     iterations = list(range(1, len(f_vals) + 1))
30     axs[1].plot(iterations, f_vals, label=f'Alpha: {alpha}')
31
32     axs[1].set_title('HB - Alpha Comparison (Batch=5)', fontweight='bold')

```

```

27 |     axs[1].set_xlabel('Iteration', fontweight='bold')
28 |     axs[1].set_ylabel('f(x, N)', fontweight='bold')
29 |     axs[1].set_xlim(0,100)
30 |     axs[1].set_ylim(0,0)
31 |     axs[1].legend()
32 |
33 |     plt.tight_layout()
34 |     plt.show()

```

```

1 training_data = generate_trainingdata()

2

3 starting_point = [3, 3]
4 alphas = np.logspace(-1, 1, num=3, base=10)
5 beta1 = 0.9
6 beta2 = 0.999
7
8 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=3,
    base=10).astype(int)
9 max_iters = 250
10
11 fig, axes = plt.subplots(3, 3, figsize=(15, 15), dpi=200)
12
13 final_points = {batch: {
14     alpha: [0.0, 0.0] for alpha in alphas
15 } for batch in batch_sizes}
16
17 for j, batch_size in enumerate(batch_sizes):
18     for i, alpha in enumerate(alphas):
19         ax = axes[i, j]
20         path, f_vals = SGD('adam', f=f, starting_point=starting_point,
21             data=training_data, alpha=alpha, beta1=beta1, beta2=beta2,
22             epsilon=1e-6, max_iters=max_iters, tol=-0.1,
23             batch_size=batch_size)
24         path_x, path_y = zip(*path)
25         final_points[batch_size][alpha][0] = path_x[-1]
26         final_points[batch_size][alpha][1] = path_y[-1]
27
28         size = max((max(path_x) - min(path_x)), (max(path_y) -
29             min(path_y))) + 2
30
31         x = np.linspace(min(path_x)-2, max(path_x)+2, 100)
32         y = np.linspace(min(path_y)-2, max(path_y)+2, 100)
33         z = [f([a, b], training_data) for a in x for b in y]
34         z = np.array(z).reshape(100, 100)
35         x, y = np.meshgrid(x, y)
36
37         contour = ax.contour(x, y, z, 50, cmap='viridis')
38         ax.plot(path_x, path_y, color='blue', linewidth=2, marker='o',
39

```

```

    markersize=3, alpha=0.7)
35 ax.scatter(path_x[0], path_y[0], color='red', s=100)

36
37 if i == 0:
38     ax.annotate('Start', xy=(path_x[0], path_y[0]), xytext=(3,
39         3),
40             textcoords='offset points', color='brown',
41             weight='bold')
42     ax.annotate('End', xy=(path_x[-1], path_y[-1]), xytext=(3,
43         3),
44             textcoords='offset points', color='brown',
45             weight='bold')

46     ax.scatter(path_x[-1], path_y[-1], color='red', s=100)
47     ax.clabel(contour, inline=1, fontsize=10)
48     ax.set_title(f'Batch Size: {batch_size}, Alpha: {alpha}',
49         fontweight='bold')
50     ax.set_xlabel('X', fontweight='bold')
51     ax.set_ylabel('Y', fontweight='bold')

52 plt.tight_layout()
53 plt.show()

54 for k, a in final_points.items():
55     for x, v in a.items():
56         print(f"Batch size: {k}, Alpha: {x}\t finished at
57             [{v[0]:0.4f}, {v[1]:0.4f}]")

```

```

1     starting_point = [3, 3]
2 alphas = np.logspace(-1, 1, num=3, base=10)
3 beta1 = 0.9
4 beta2 = 0.999
5 batch_sizes = np.logspace(0.75, np.log10(len(training_data)), num=3,
6     base=10).astype(int)
7 max_iters = 100
8
9 fig, axs = plt.subplots(1, 2, figsize=(20, 6), dpi=200)
10
11
12 for batch_size in batch_sizes:
13     _, f_vals = SGD('adam', f=f, starting_point=starting_point,
14         data=training_data, alpha=1, beta1=beta1, beta2=beta2,
15         epsilon=1e-6, max_iters=max_iters, tol=-0.1,
16         batch_size=batch_size)
17     iterations = list(range(1, len(f_vals) + 1))
18     axs[0].plot(iterations, f_vals, label=f'Batch Size: {batch_size}', )

```

```

16
17 axs[0].set_title('Adam - Batch Size Comparison (alpha=0.01)',
18   fontweight='bold')
18 axs[0].set_xlabel('Iteration', fontweight='bold')
19 axs[0].set_ylabel('f(x, 5)', fontweight='bold')
20 axs[0].legend()
21
22
23 for alpha in alphas:
24     _, f_vals = SGD('adam', f=f, starting_point=starting_point,
25       data=training_data, alpha=alpha, beta1=beta1, beta2=beta2,
26       epsilon=1e-6, max_iters=max_iters, tol=-0.1, batch_size=5)
27     iterations = list(range(1, len(f_vals) + 1))
28     axs[1].plot(iterations, f_vals, label=f'Alpha: {alpha}')
29
30 axs[1].set_title('Adam - Alpha Comparison (Batch=5)',
31   fontweight='bold')
31 axs[1].set_xlabel('Iteration', fontweight='bold')
32 axs[1].set_ylabel('f(x, N)', fontweight='bold')
32 axs[1].legend()
33
34 plt.tight_layout()
34 plt.show()

```