COMPSCI 371D Homework 4

Problem 0 (3 points)

Jihyeon Je (jj271), Rahul Prakash (rp221), Tim Ho (th265)

Part 1: Stochastic Gradient Descent

In [2]:
```
from urllib.request import urlretrieve
from os import path as osp


def retrieve(file_name, semester='fall21', course='371d', homework=4):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci{}/homework/{}
/{}'
        url = fmt.format(semester, course, homework, file_name)
        urlretrieve(url, file_name)
        print('Downloaded file {}'.format(file_name))
```

In [3]:
```
import pickle


data_file_name = 'samples.pkl'
code_file_names = ('decorators.py', 'function.py')
for name in (data_file_name, *code_file_names):
    retrieve(name)

from decorators import HistoryTracker
from function import f, phi

with open(data_file_name, 'rb') as file:
    data = pickle.load(file)
```

```
Downloaded file samples.pkl
Downloaded file decorators.py
Downloaded file function.py
```

In [4]:
```
import autograd.numpy as np
from autograd import grad

z_0 = np.array((0.4, 0.1))
f_0 = f(z_0, data)
plot_info = {
        'z_0': z_0,
        'f_0': f_0,
        'z_ast': phi.z_ast,
        'x_range': [-1., 2.5],
        'y_range': [-2.2, 1.],
}
```

In [5]:
```
def plot_contours(fct, samples, info, title=None, hist=None,
                  grid_samples=101, marker_size=15, font_size=20):
    x = np.linspace(info['x_range'][0], info['x_range'][1], grid_samples
)
    y = np.linspace(info['y_range'][0], info['y_range'][1], grid_samples
)
    box = (x[0], x[-1], y[0], y[-1])
    x_grid, y_grid = np.meshgrid(x, y)
    z_grid = np.stack((x_grid, y_grid), axis=0)
    fct_grid = fct(z_grid, samples)
    fig = plt.figure(figsize=(13, 12), tight_layout=True)
    img = plt.imshow(fct_grid, interpolation='bilinear',
               origin='lower', extent=box, cmap=cm.hot)
    plt.contour(x_grid, y_grid, fct_grid, 50, colors='w', linewidths=1)
    plt.plot(info['z_0'][0], info['z_0'][1], 'bo', markersize=marker_siz
e)
    plt.plot(info['z_ast'][0], info['z_ast'][1], 'go', markersize=marker
_size)
    plt.axis('scaled')
    plt.xticks(fontsize=font_size)
    plt.yticks(fontsize=font_size)
    label_size = (font_size * 1.5) // 1
    plt.xlabel('$z_0$', fontsize=label_size)
    plt.ylabel('$z_1$', fontsize=label_size)
    bar = fig.colorbar(img, shrink=0.72)
    bar.ax.tick_params(labelsize=font_size)
    if hist is not None:
        hz = hist['z']
        plt.plot(hz[:, 0], hz[:, 1], color='w', linewidth=2)
    if title is not None:
        plt.title(title, fontsize=font_size)
    plt.show()
```

In [6]:
```
from matplotlib import pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
```

In [7]:
```
from numpy import random as npr


def batch_index_generator(n_samples, batch_size):
    rg = npr.default_rng(15)
    batch = rg.permutation(n_samples)
    start, stop = 0, batch_size
    while stop < n_samples:
        yield batch[start:stop]
        start += batch_size
        stop += batch_size
    stop = min(stop, n_samples)
    yield batch[start:stop]


@HistoryTracker
def step(fct, samples, z, alpha):
    def fct_batch(u):
        return fct(u, samples)
    z_prime = z - alpha * grad(fct_batch)(z)
    f_prime = fct_batch(z_prime)
    return z_prime, f_prime
```

Problem 1.1 (Exam-Style)

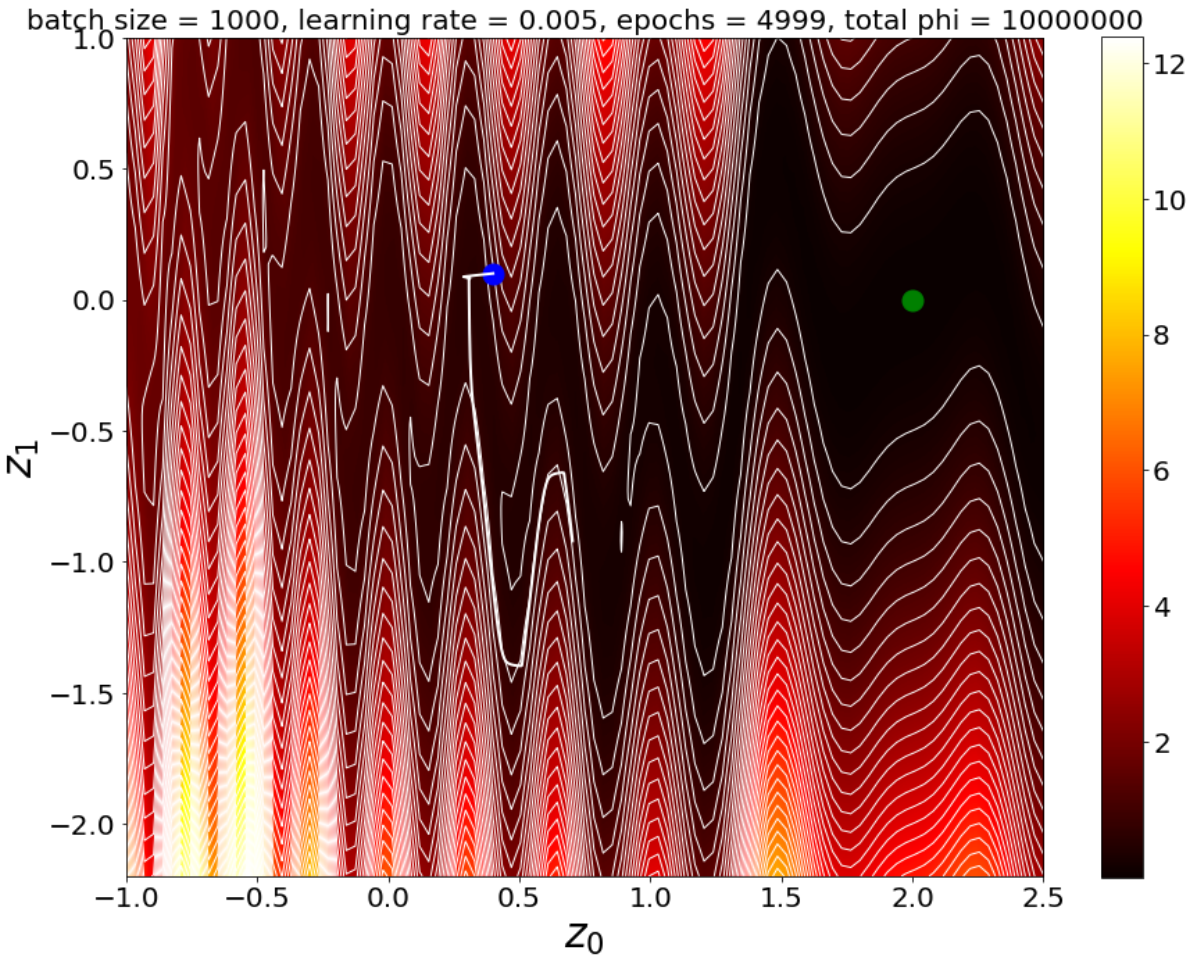
No, the number of phi in a single epoch does not depend on the size of the mini-batch since phi has to be evalulated for every single data point. Because every sample in the whole dataset has to be evaluated, the size of the mini-batch does not matter, since the total number of samples stay the same no matter the size of the mini-batch.

Problem 1.2

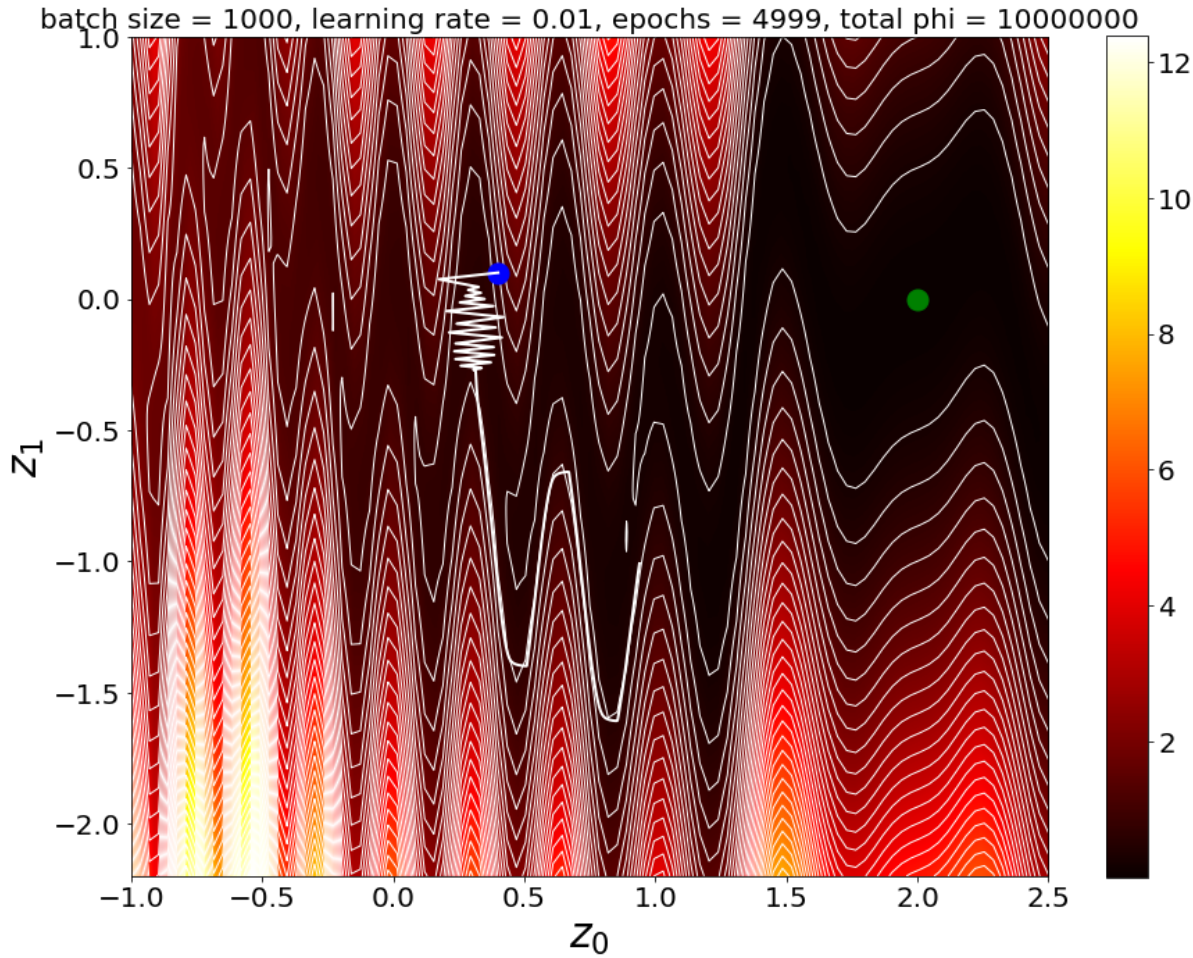
In [121]:
```
def sgd(fct, samples, z, batch_size, alpha, min_step=1.e-5, max_epochs=5
000):
    for i in range(0,max_epochs):
        for batch_indices in batch_index_generator(len(samples), batch_s
ize):
            z_next,_ = step(fct, samples[batch_indices], z, alpha)

            if ((np.linalg.norm(z_next - z)< min_step) or (i>max_epochs
)):
                #print("warning: max epochs exceeded")
                return z, i

            z = z_next
    return z, i
```

In [122]:
```
def experiment(batch_size, alpha, info):
    step.restart(info['z_0'],info['f_0'])
    phi.reset()
    z_new, tot_epochs = sgd(f, data, info['z_0'], batch_size, alpha, min
_step=1.e-5, max_epochs=5000)
    tot_phi = phi.calls()
    plot_contours(f, data[0], plot_info, title = f'batch size = {batch_s
ize}, learning rate = {alpha}, epochs = {tot_epochs}, total phi = {tot_p
hi}',hist=step.history())
```

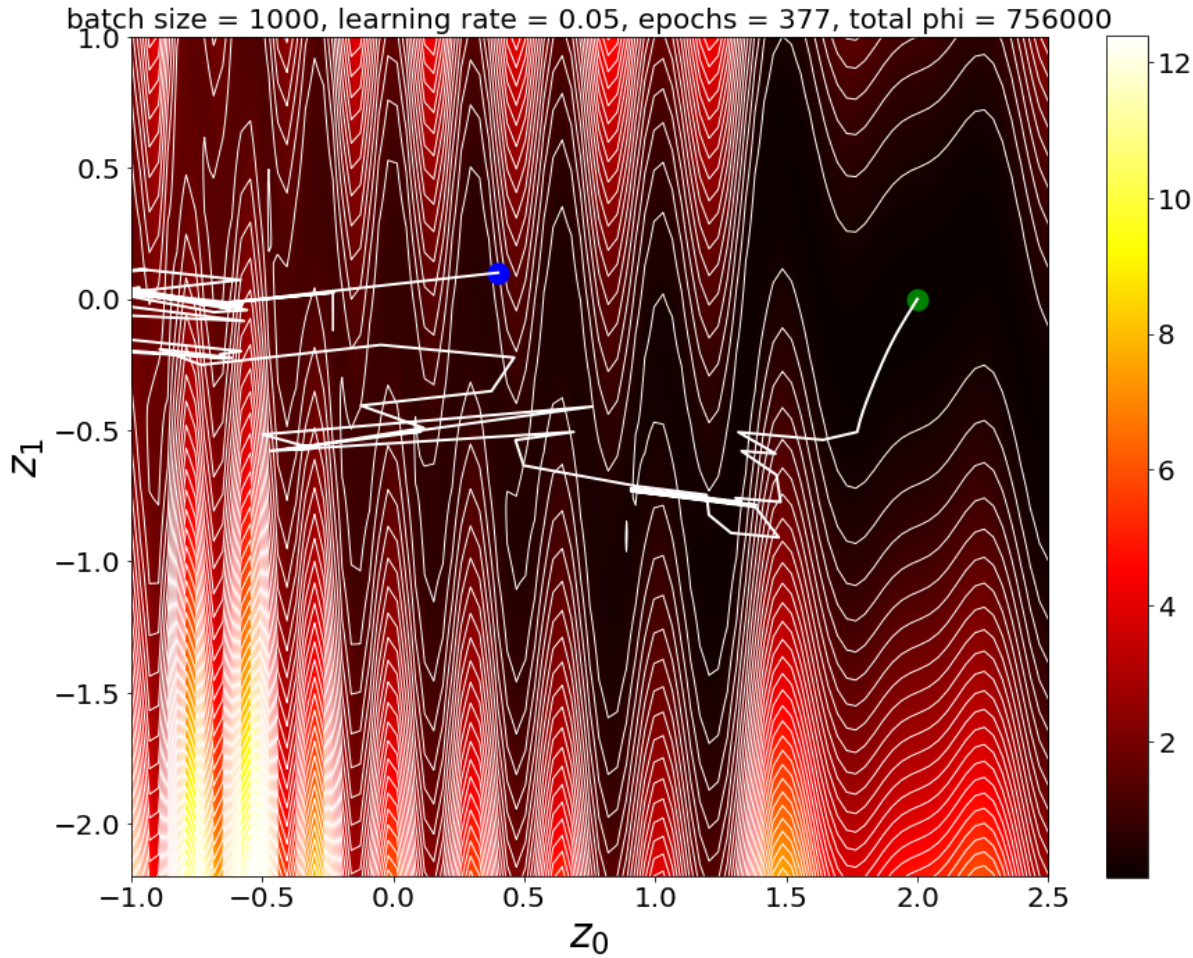
In [123]:
```
alpha = (0.005, 0.01, 0.05)
experiment(1000, alpha[0], plot_info)
```



In [124]:
```
experiment(1000, alpha[1], plot_info)
```



In [125]:
```
experiment(1000, alpha[2], plot_info)
```



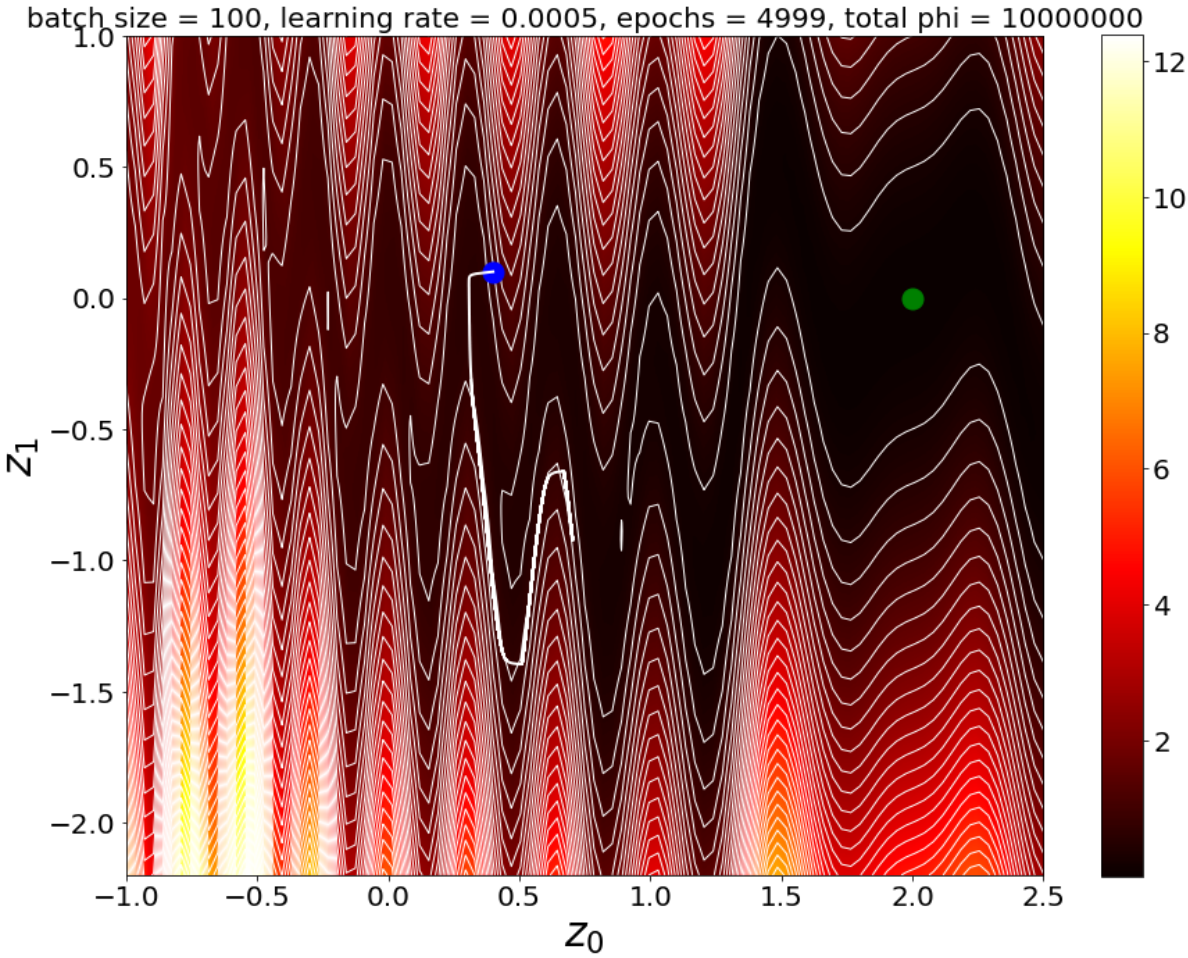
Problem 1.3 (Exam Style)

At the lower learning rates 0.005 and 0.01, trajectory follows the local gradient contours more closely. This is because the step size is smaller as alpha decreases. However, these cases reach the max number of epochs prior to convergence. Especially, with the learning rate of 0.01, the trajectory oscillates at a local minimum due to the larger step size. As we increase the learning rate to 0.05, the trajectory does not follow the local gradient contour as closely. It seems to change its directions quickly since it reponds strongly to the local gradient. However, this case achieves convergence prior to reaching the max epochs.

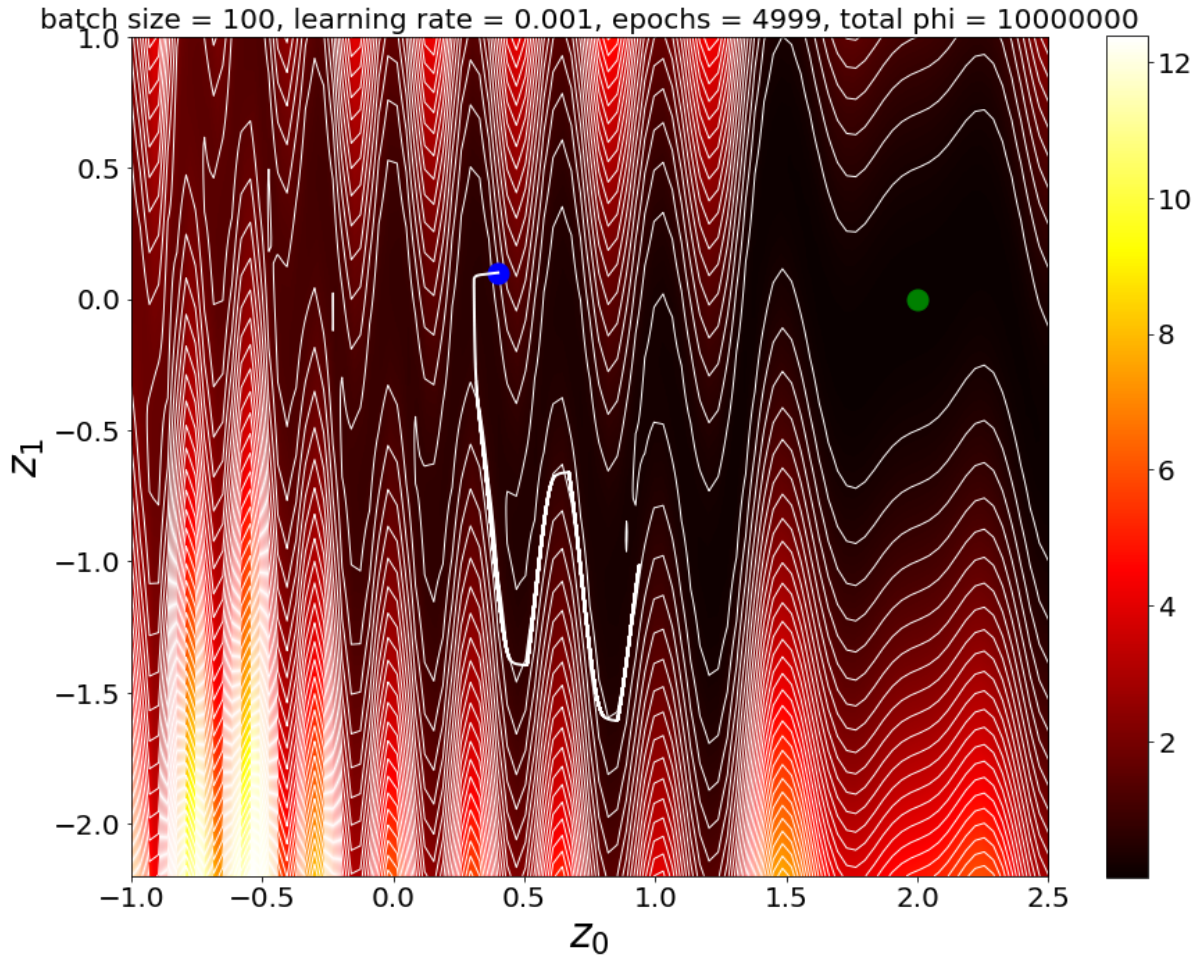
However, these results are not reliable especially when $\mathbf{z} \in \mathbb{R}^d$ with d much greater than 2. This is because gradient descent is sensitive to the shape of the local gradient and starting point. The choice of a "good" learning rate is therefore dependent on these specific conditions as well. Furthermore, the minimization may get trapped/converge to a local minima or follow a non-optimal path even though a monotonically decreasing path from z0 to z *exists. As dimensionality increases, the number of possible trajectories from a given z increases greatly. Therefore, the chances of the steepest descent trajectory being along the monotonically decreasing path (z0 to z*) at each step are quite low, making true convergence very rare.

Problem 1.4

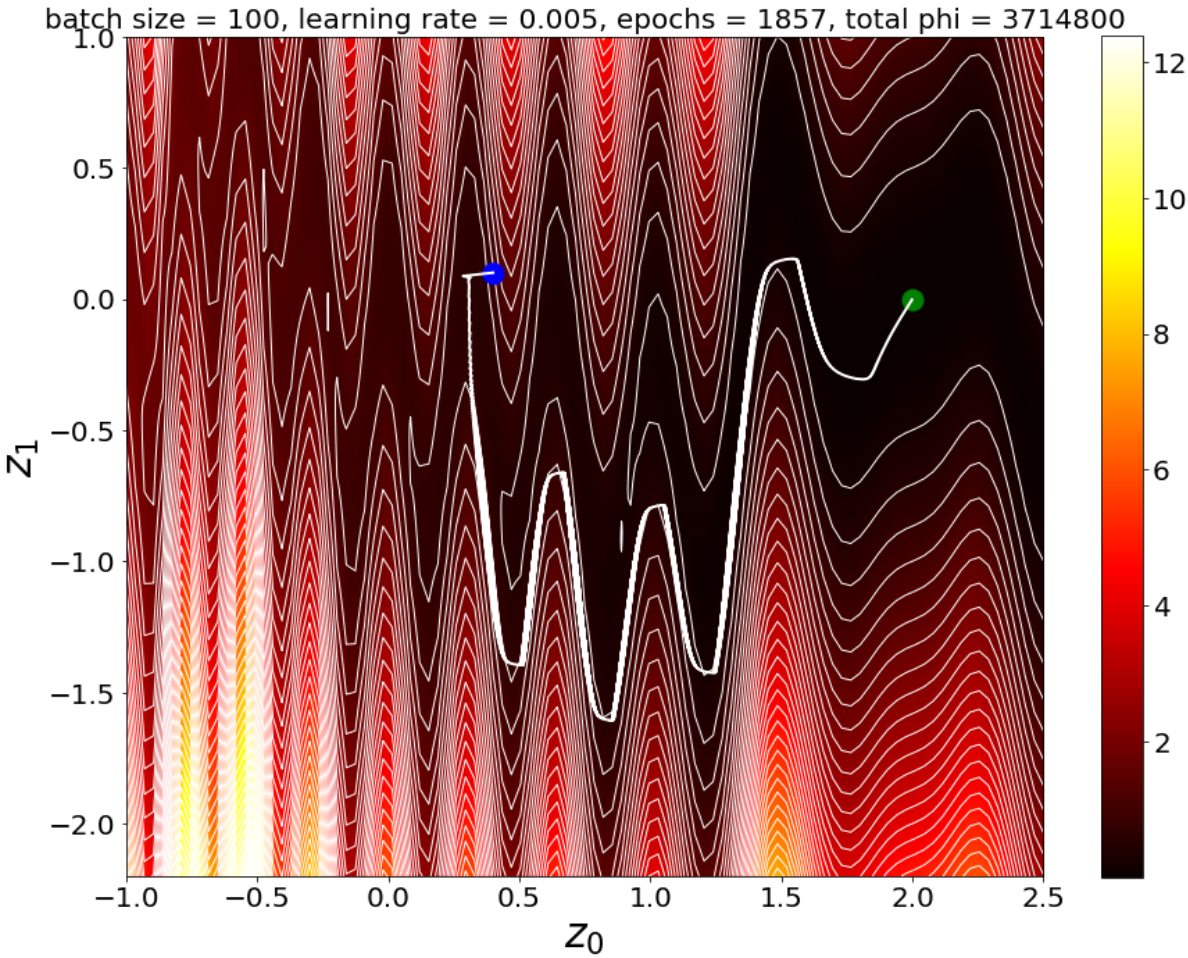
In [126]:
```
alpha = (0.0005, 0.001, 0.005, 0.01)
experiment(100, alpha[0], plot_info)
```



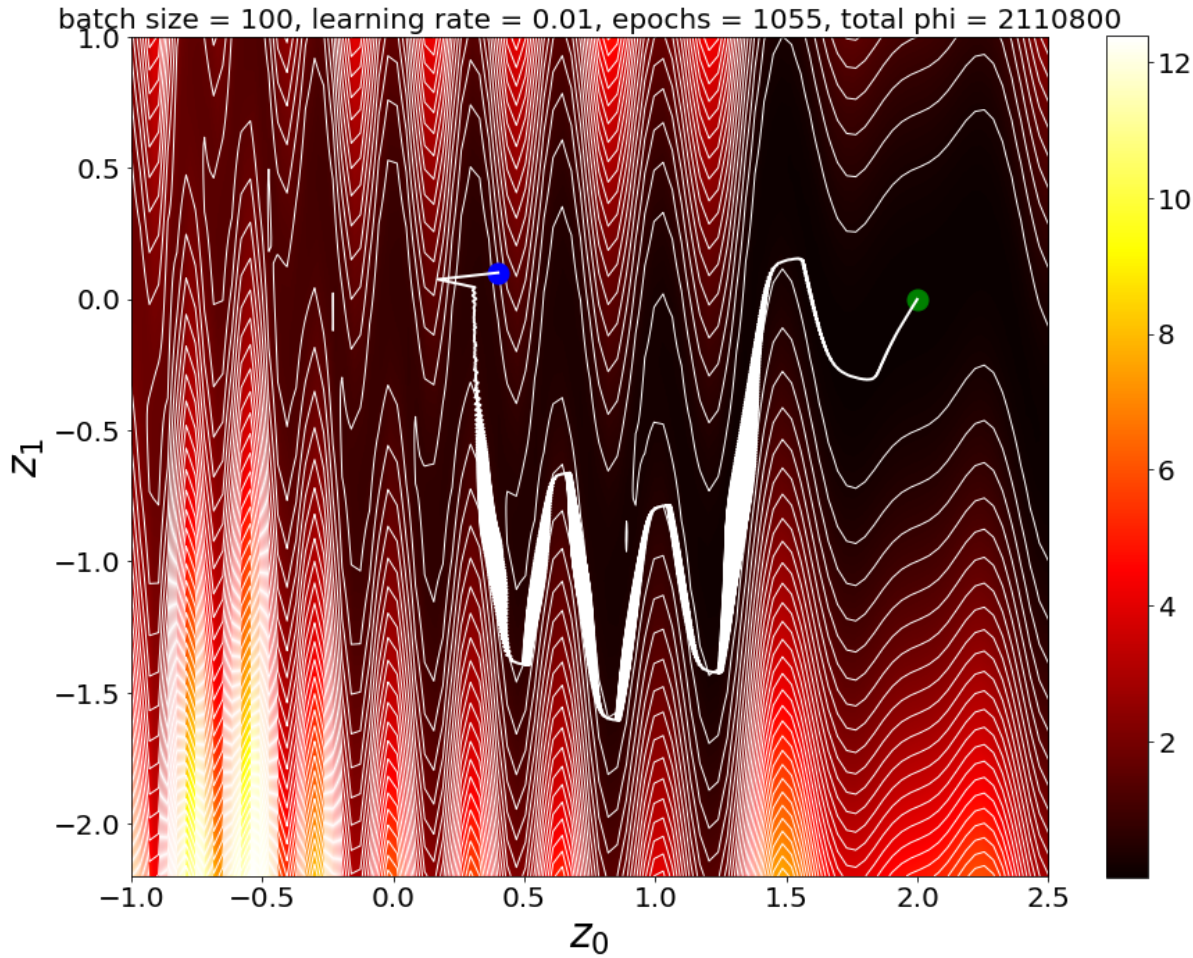
In [127]:
```
experiment(100, alpha[1], plot_info)
```



In [128]:
```
experiment(100, alpha[2], plot_info)
```



In [129]:
```
experiment(100, alpha[3], plot_info)
```



Problem 1.5 (Exam Style)

For alpha = 0.0005, the trajectory follows the local gradient very closely, but reaches the max number of epochs before converging as it takes very small steps. For alpha = 0.001, the trajectory again follows the local gradient very closely, but reaches the max number of epochs before converging. However, it was closer to convergence compared to alpha = 0.0005 case. for alpha = 0.005, the trajectory reached convergence after 1857 epochs as it took bigger steps. With this learning rate, the trajectory followed the local gradient more loosely and begins to oscillate. for alpha = 0.01, the trajectory reached convergence after 1055 epochs, but oscillated much greatly throughout the path as it bounces around a local minimum because of its larger stepsize.

Concerning the two cases for $\alpha = 0.005$ and $\alpha = 0.01$, one case is not preferable to the other when applying SGD in general. The learning rate must be chosen to suit the particular landscape of the gradient and thus we cannot say one alpha value is better than the other in general. The drawbacks of using a smaller alpha value ($\alpha = 0.005$) is that you run more epochs while the drawbacks of using a larger alpha value ($\alpha = 0.01$) is that you risk the danger of deviating away from the global minimum when the gradient is not smooth.

It would be extremely hard to select a good fixed learning rate when z lives in a space with many dimensions because it would not be possible to plot the gradient/trajectory. As we cannot visualize the gradient in multiple dimensions, we would have to make the judgement based off of numerical evaluation metrics(i.e. accuracy) which makes it extremely hard to select a good learning rate.

Problem 1.6 (Exam Style)

SGD may be preferable to the ordinary version of the Gradient Descent as it is more flexible compared to GD. SGD uses a stochastically chosen batch of training data for the gradient computation for each step. This gives it more flexibility in terms of size and directions of the steps taken in one epoch as the gradient varies depending on the distribution of the data in the chosen batch. SGD also results in a greater number of steps in an epoch. These characteristics make SGD more likely converge faster than GD when using large datasets. SGD will use smaller batches approximations of the true data, thereby computing each step faster and converging faster.

Part 2: Hyperplanes

Problem 2.1 (Exam Style)

let Q an arbitrary point on the plane

$$\vec{n} = [2, 2, 1]$$
$$Q = (0, 0, 3)^T$$
$$P = (1, 0, 4)^T$$
$$\overrightarrow{PQ} = [1, 0, 1]^T$$
$$||n|| = \sqrt{(9)} = 3$$
$$distance = |proj_n \overrightarrow{PQ}| = \frac{\overrightarrow{PQ} \cdot n}{||n||} = \frac{3}{3} = 1$$

Problem 2.2 (Exam Style)

distance from point P to the origin:

$$\sqrt{1^2 + 4^2} = \sqrt{17}$$

distance from plane II to the origin:

$$\vec{n} = [2, 2, 1]$$
$$Q = (0, 0, 3)^T$$
$$O = (0, 0, 0)^T$$
$$\overrightarrow{QO} = [0, 0, 3]^T$$
$$||n|| = \sqrt{(9)} = 3$$
$$distance = |proj_n \overrightarrow{QO}| = \frac{\overrightarrow{QO} \cdot n}{||n||} = \frac{3}{3} = 1$$

Therefore, the distance from point P to the origin is farther away than the distance from plane II to the origin.

Problem 2.3 (Exam Style)

Plane Π' that is parallel to Π has the form $2x + 2y + z = c$

Since Plane Π' goes through point p, $2(1) + 2(0) + 1(4) = 6$, and therefore $c = 6$

We can write the equation in the form $-6 + 2x + 2y + z = 0$ where $w_x = 2, w_y = 2, w_z = 1$