COMPSCI 371 Homework 3

Group Members: Phillip Sievers, Jose Pablo Rivera, Gordon Liang

Problem 0 (3 points)

Part 1: Notation

Problem 1.1 (Exam Style)

a = 0. b = (1,0)

Problem 1.2 (Exam Style)

A circle in the real plane with radius 2 centered at point (1,0).

Problem 1.3 (Exam Style)

1. c < 0.
2. c = 0. point is (1,0).

Problem 1.4 (Exam Style)

An annulus in the real plane with inner radius 2 and outer radius 3 centered at (1,0).

i.e. the locus of points with Euclidean distance between 2 and 3, inclusive, from point (1,0).

Part 2: Momentum

Problem 2.1 (Exam Style)

$$\nabla q(z) = \begin{bmatrix} z_0 - 3 \\ 2z_1 \end{bmatrix}$$

Problem 2.1 (Exam Style)

$$v_{k+1} = \mu_k v_k - \alpha_k \nabla q(z_k)$$

$$z_{k+1} = z_k + v_{k+1}$$

$$v_0 = [0, 0]$$

$$z_0 = [4, 2]$$

$$v_1 = 0.5 \cdot [0, 0] - 0.1 \cdot [1, 4] = [-0.1, -0.4]$$

$$z_1 = [4, 2] + [-0.1, -0.4] = [3.9, 1.6]$$

$$v_2 = 0.5 \cdot [-0.1, -0.4] - 0.1 \cdot [0.9, 3.2] = [-0.14, -0.52]$$

$$z_2 = [3.9, 1.6] + [-0.14, -0.52] = [3.76, 1.08]$$

Problem 2.2

In [26]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [27]:

```
def gradient_descent(g, z0, mu = 0.5, alpha=0.1, delta=1.e-6, k_max=30,
    v_old = np.zeros(len(z0))
    z, z_old = z0, z0.copy()
    history = [z] if store else None
    for k in range(k_max):
        v = mu * v_old - alpha * g(z_old)
        z = z_old + v
        if store:
            history.append(z)
        if np.linalg.norm(z - z_old) <= delta:
            return (z, history) if store else z
        v_old = v
        z_old = z
    print('warning: maximum iterations exceeded')
    return (z, history) if store else z
```

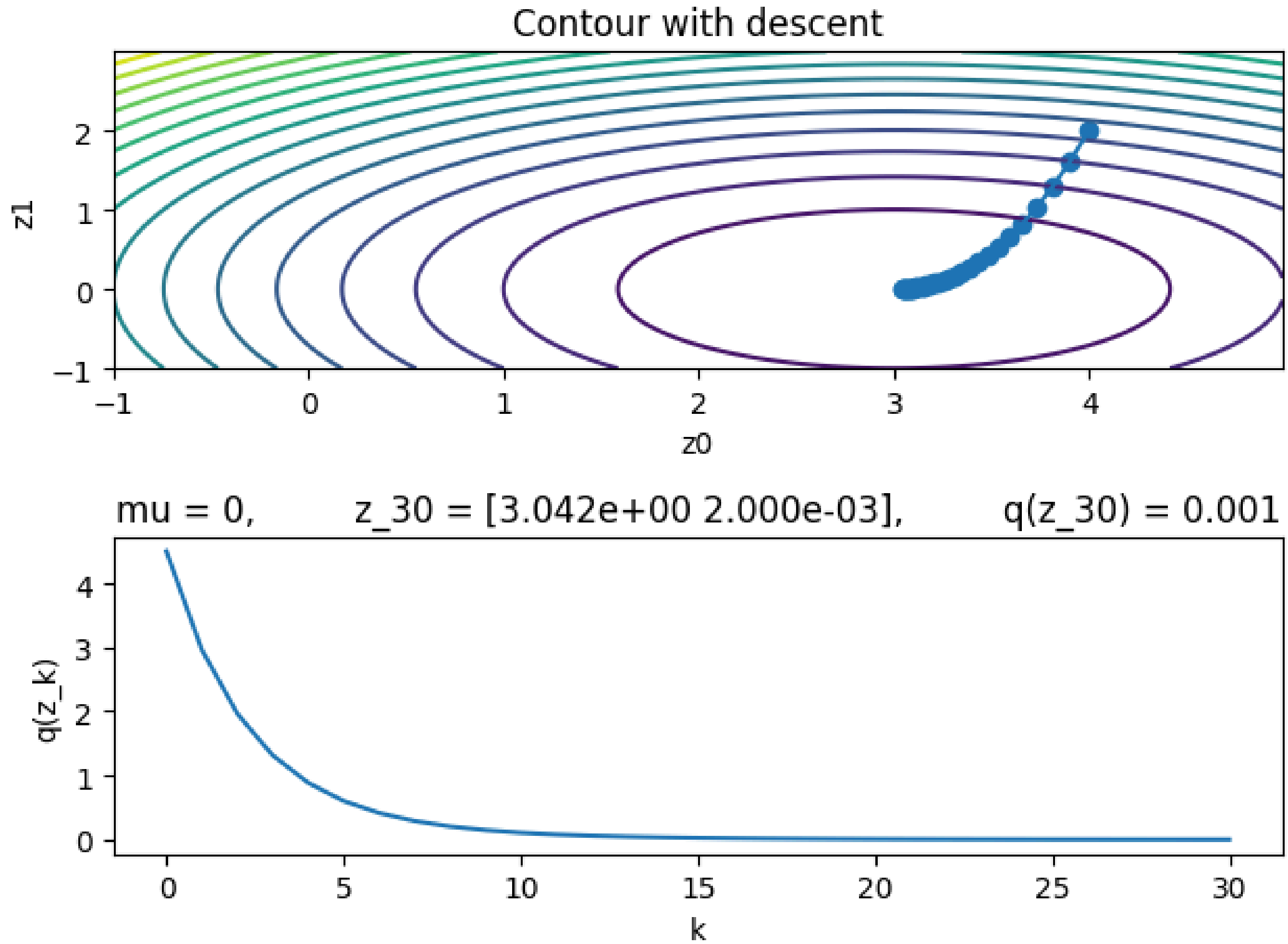
In [28]:

```
def g(z):
    return np.array([z[0] - 3, 2*z[1]])
```

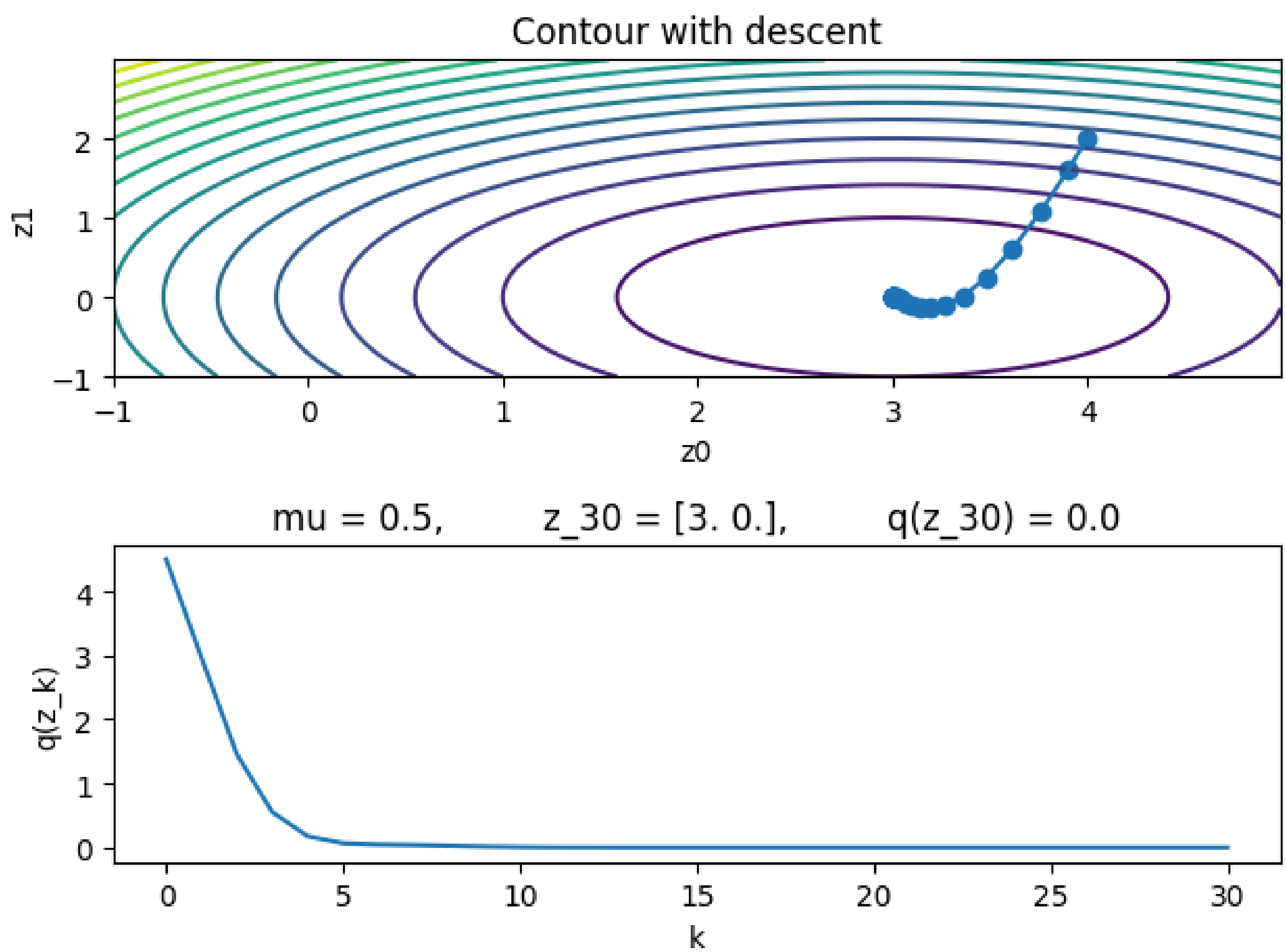
```python
def q(z):
    return 1/2 * ((z[0] - 3)**2 + 2*z[1]**2)
```

In [29]:
```python
z0 = np.array([4,2])

z0s = np.arange(-1, 5, 0.01)
z1s = np.arange(-1, 3, 0.01)
Z0, Z1 = np.meshgrid(z0s, z1s)
Q = q([Z0, Z1])

for i, mu in enumerate([0, 0.5, 0.9]):
    fig, ax = plt.subplots(2)

    z, history = gradient_descent(g, z0, mu)
    z_last = history[len(history) - 1]

    ks = np.arange(len(history))
    qs = list(map(q, history))

    ax[0].contour(Z0, Z1, Q, levels=20)
    ax[0].set_xlabel("z0")
    ax[0].set_ylabel("z1")
    ax[0].set_title("Contour with descent")

    xs, ys = zip(*history)
    ax[0].scatter(xs, ys)
    ax[0].plot(xs, ys)

    ax[1].plot(ks, qs)
    ax[1].set_title(
        f"mu = {mu}, \
        z_{len(history) - 1} = {np.round(z_last, 3)}, \
        q(z_{len(history) - 1}) = {np.round(q(z_last),3)}"
    )
    ax[1].set_xlabel("k")
    ax[1].set_ylabel("q(z_k)")

    plt.tight_layout()
    plt.show()
```

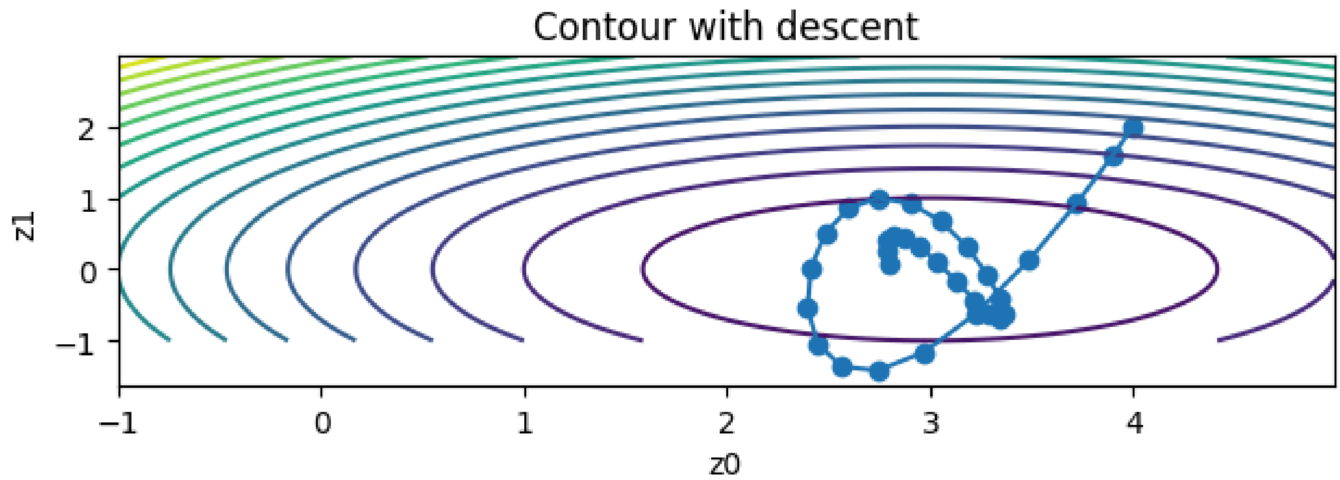
```
warning: maximum iterations exceeded
```

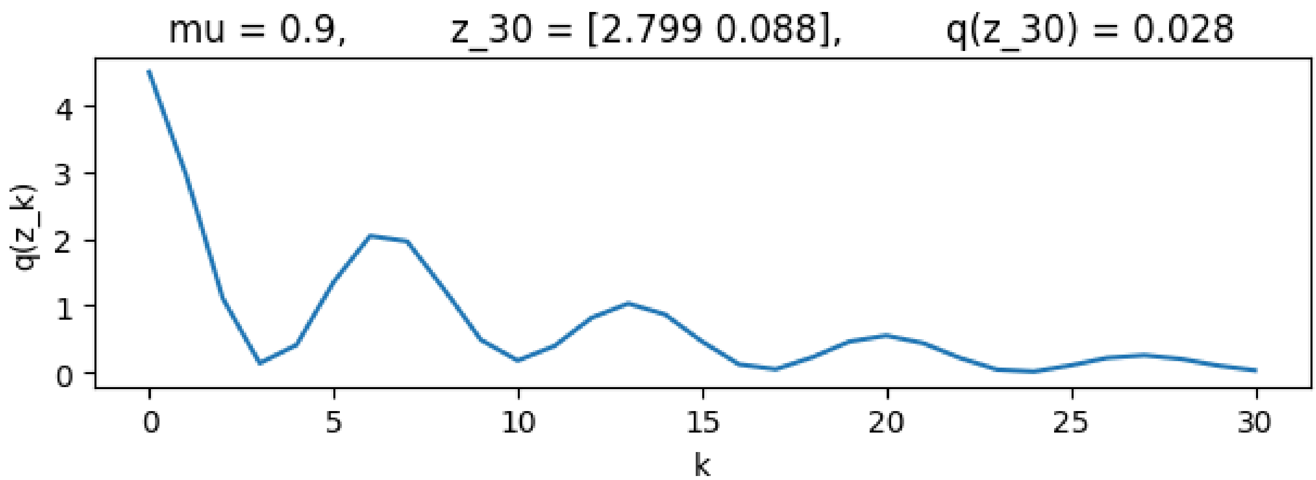



```
warning: maximum iterations exceeded
```



```
warning: maximum iterations exceeded
```





Problem 2.3 (Exam Style)

1. $z_{true} = (3, 0)$. $q(z_{true}) = 0$.
2. yes. $\mu = 0.5$ is more efficient than $\mu = 0$.

- contour shows faster approach to $z_{true} = (3, 0)$
- graph shows steeper approach to $q(z_{true}) = 0$
- overcome spurious local minima

3. no. $\mu = 0.9$ is less efficient than $\mu = 0$.

- contour shows overshoot of $z_{true} = (3, 0)$
- graph shows oscillation above $q(z_{true}) = 0$
- i.e. at $k = 10$, $q(z_k) < 1$ for $\mu = 0$ but $q(z_k) > 1$ for $\mu = 0.9$

Part 3: Automatic Differentiation

In [30]:
```python
from autograd import numpy as anp
```

```python
from autograd import elementwise_grad as gradient
```

Problem 3.1 (Exam Style except for the Plots)

$$h(x) = \frac{1}{1 + ae^{-bx}}$$

$$h'(x) = \frac{-1}{(1 + ae^{-bx})^2} \cdot ae^{-bx} \cdot -b = \frac{abe^{-bx}}{(1 + ae^{-bx})^2}$$

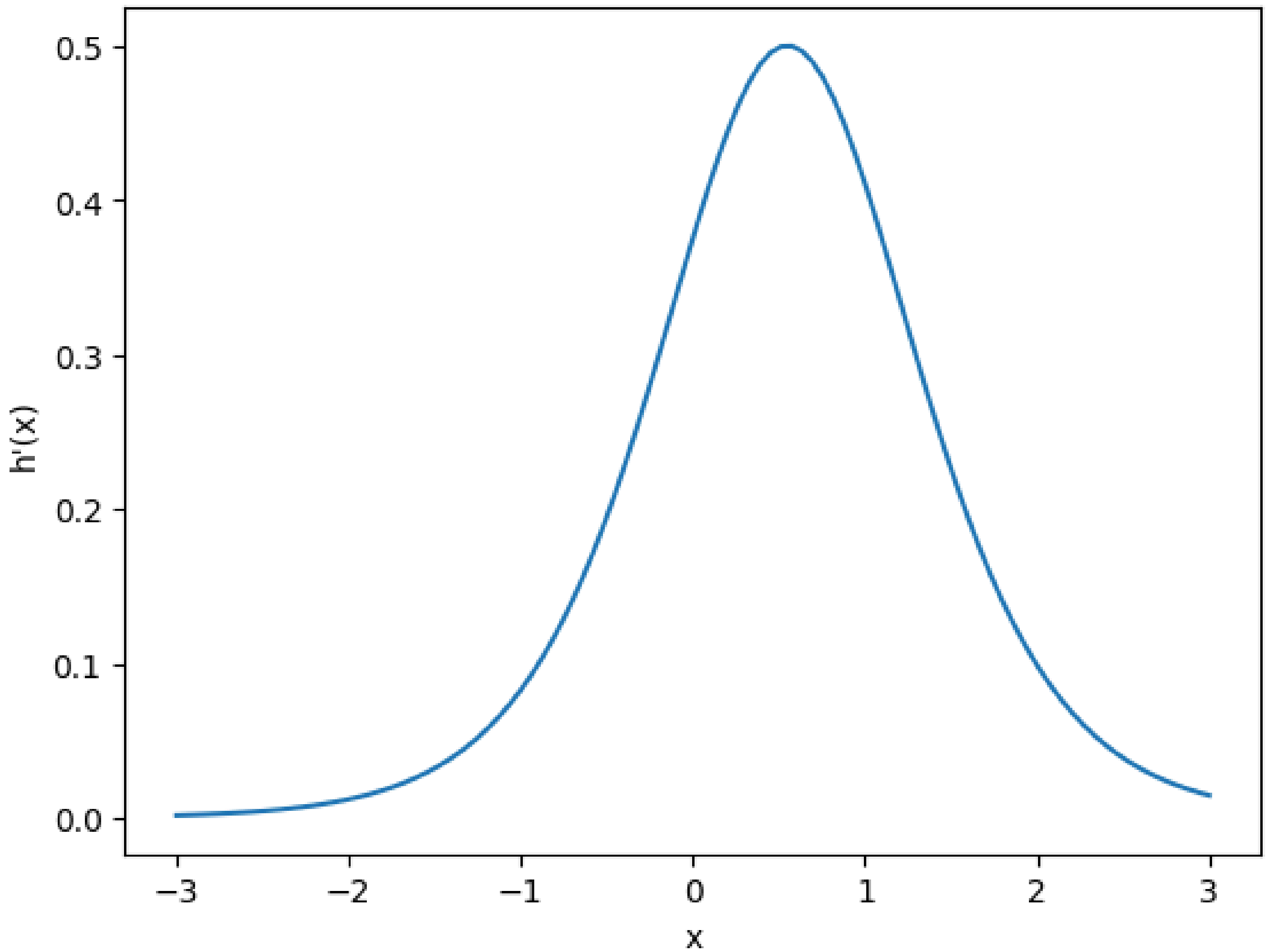
$$g(a, b) = \nabla_{a,b} h(x) = \begin{bmatrix} \frac{-1}{(1 + ae^{-bx})^2} \cdot e^{-bx} \\ \frac{-1}{(1 + ae^{-bx})^2} \cdot ae^{-bx} \cdot -x \end{bmatrix} = \begin{bmatrix} \frac{-e^{-bx}}{(1 + ae^{-bx})^2} \\ \frac{xae^{-bx}}{(1 + ae^{-bx})^2} \end{bmatrix}$$

In [31]:

```python
def h(x, z):
    a, b = z[0], z[1]
    return 1 / (1 + a * anp.exp(-b * x))

def h_prime(x, z):
    a, b = z[0], z[1]
    return a*b*anp.exp(-b * x) / (1 + a * anp.exp(-b * x))**2
```

In [32]:

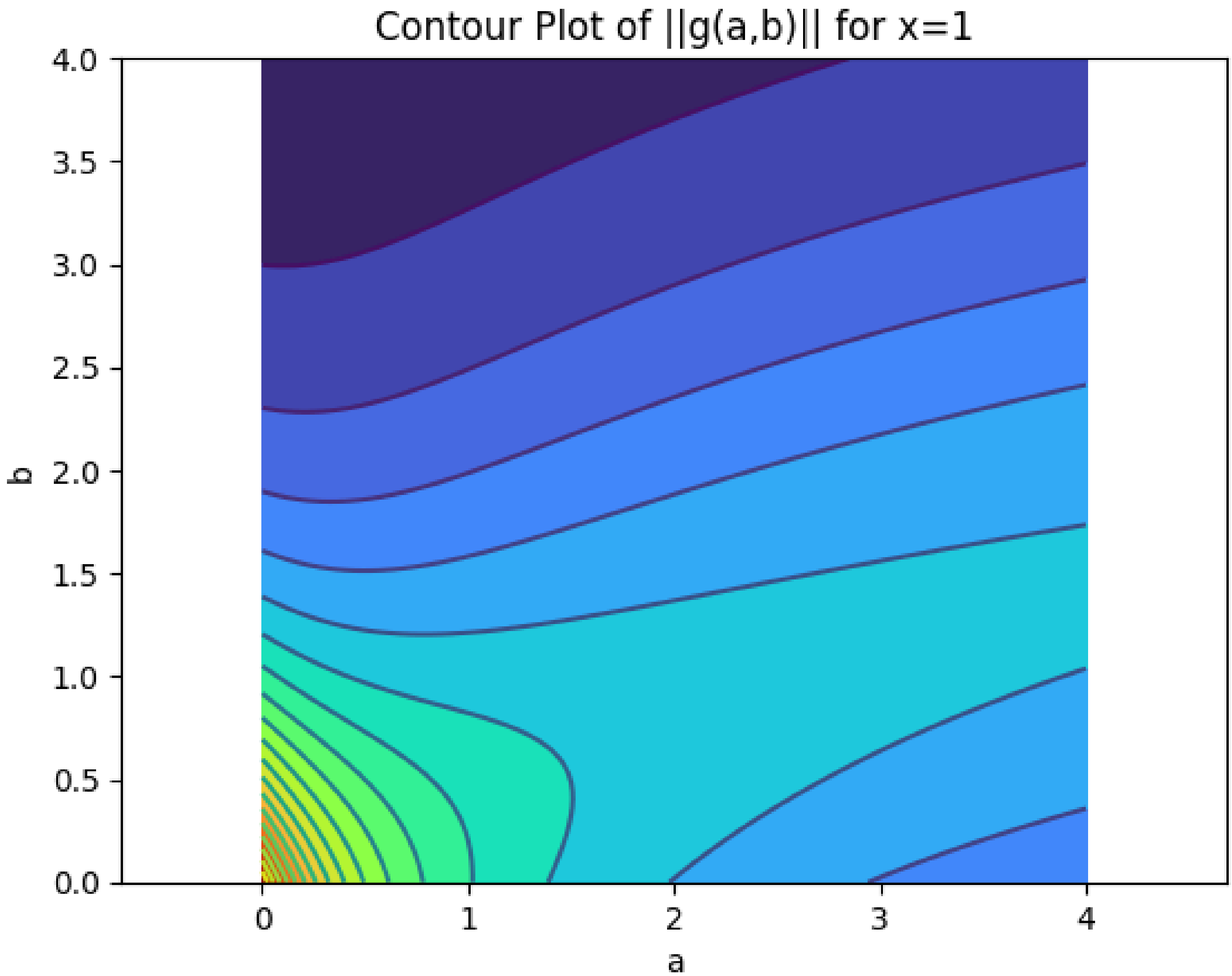
```python
def plot(xs, z, hp):
    fig, ax = plt.subplots()

    ys = [ hp(x, z) for x in xs ]

    ax.set_xlabel("x")
    ax.set_ylabel("h'(x)")
    ax.plot(xs, ys)

xs = anp.linspace(-3, 3, 100)
a, b = 3, 2
plot(xs, [a, b], h_prime)
```



In [33]:
```
def gamma(a,b,x):
    dhda = -anp.exp(-b*x) / (1 + a * anp.exp(-b * x))**2
    dhdb = x * a * anp.exp(-b * x) / (1 + a * anp.exp(-b * x))**2
    return (dhda**2+dhdb**2)**0.5
```

In [34]:
```
fig, ax = plt.subplots()
x = 1
a = anp.linspace(0, 4, 100)
b = anp.linspace(0, 4, 100)
A, B = anp.meshgrid(a, b)

Z = gamma(A,B,x)

ax.set_xlabel("a")
ax.set_ylabel("b")
ax.contour(A, B, Z, levels=20)
ax.contourf(A, B, Z, levels=20, cmap="turbo")
ax.set_title('Contour Plot of ||g(a,b)|| for x=1')
plt.axis('equal')
```

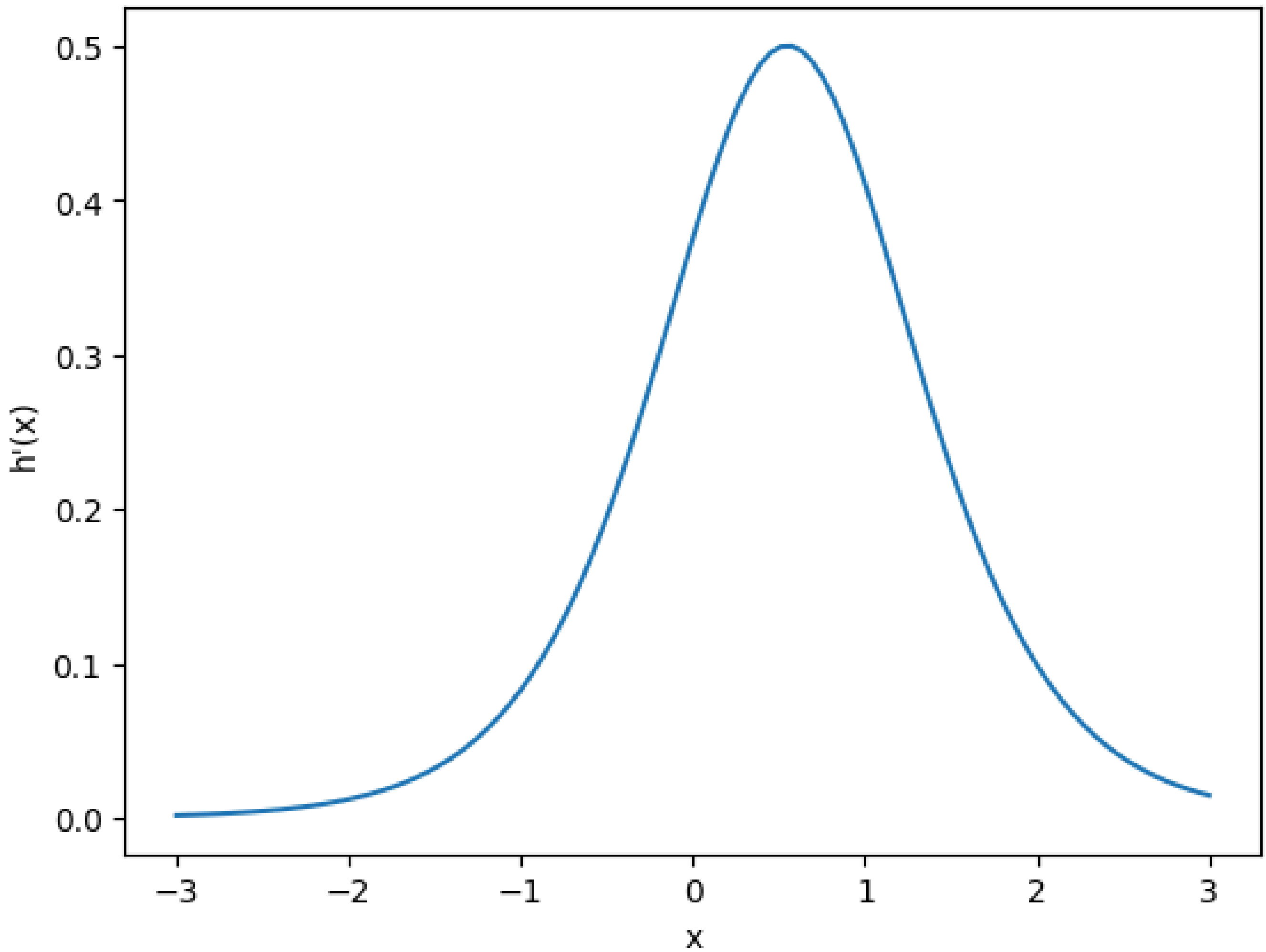
Out[34]:
```
(np.float64(0.0), np.float64(4.0), np.float64(0.0), np.float64(4.0))
```



Problem 3.2

In [35]:
```
def ah(x, z):
    a, b = z[0], z[1]
    return 1 / (1 + a * anp.exp(-b * x))
```

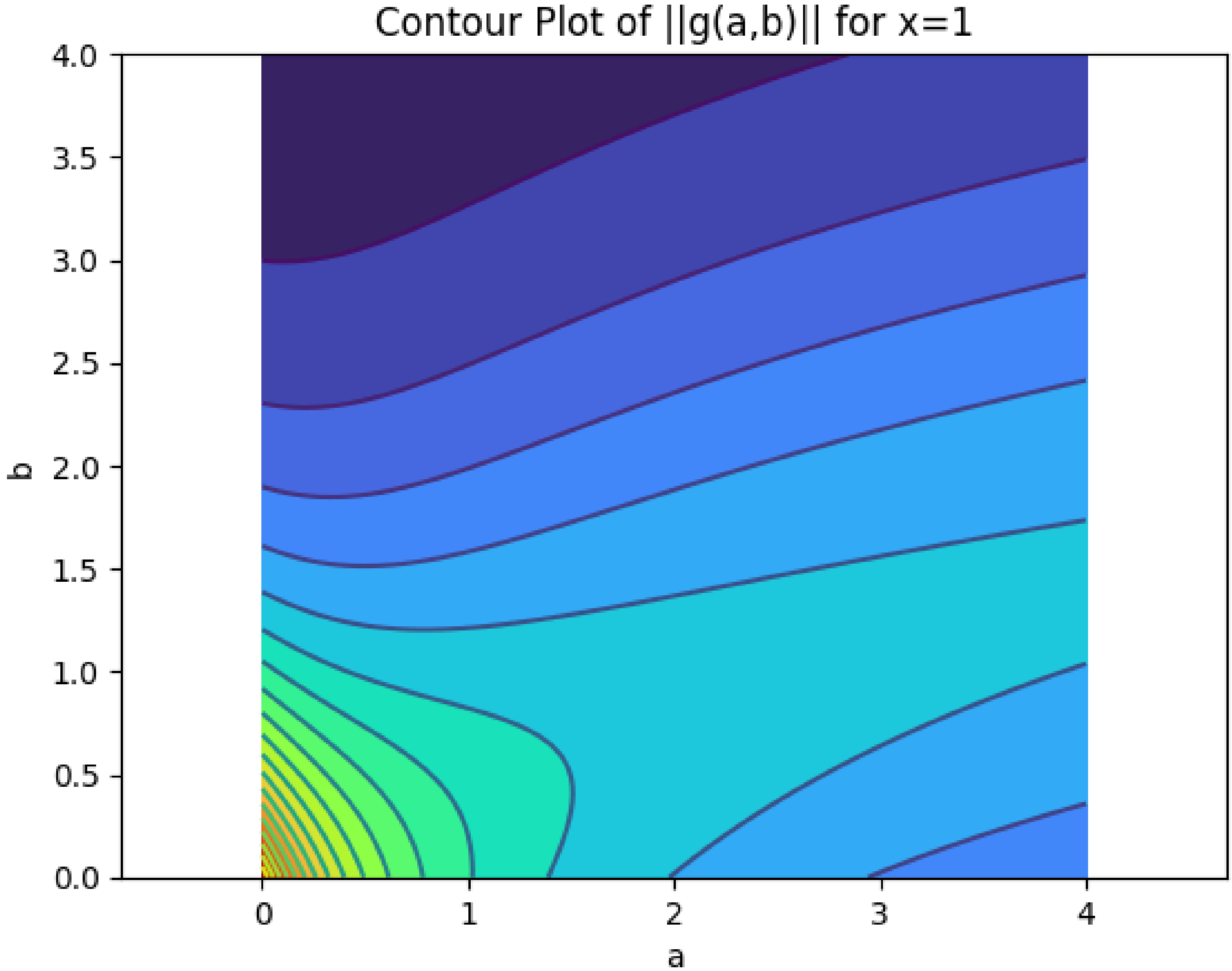
In [36]:
```
xs = anp.linspace(-3, 3, 100)
a, b = 3, 2
plot(xs, [a, b], gradient(ah))
```



In [37]:
```
def a_gamma(x, z):
    A, B = z[0], z[1]
    return np.linalg.norm(g(x,[A,B]), axis=0)
```

In [38]:
```
fig, ax = plt.subplots()
x = 1
a = anp.linspace(0, 4, 100)
b = anp.linspace(0, 4, 100)
A, B = anp.meshgrid(a, b)

g = gradient(ah,1)
Z = a_gamma(x, [A, B])

ax.set_xlabel("a")
ax.set_ylabel("b")
ax.contour(A, B, Z, levels=20)
ax.contourf(A, B, Z, levels=20, cmap="turbo")
ax.set_title('Contour Plot of ||g(a,b)|| for x=1')
plt.axis('equal')
```

Out[38]:
```
(np.float64(0.0), np.float64(4.0), np.float64(0.0), np.float64(4.0))
```



Part 4: Stochastic Gradient Descent

In [39]:
```
def batch_index_generator(n_samples, batch_size, rg):
    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]
```

Problem 4.1 (Exam Style)

$$
\begin{aligned}
L_T(\vec{z}) &= \frac{1}{N}\sum_{n=0}^{N-1} l(y_n, h_z(x_n)) && (1)\\
&= \frac{1}{N}\sum_{n=0}^{N-1} (h_z(x_n) - y_n)^2 && (2)\\
\nabla L_T(\vec{z}) &= \frac{1}{N}\sum_{n=0}^{N-1} \nabla (h_z(x_n) - y_n)^2 && (3)\\
&= \frac{1}{N}\sum_{n=0}^{N-1} 2g_z(x_n)\cdot(h_z(x_n) - y_n) && (4)
\end{aligned}
$$

Problem 4.2

In [40]:
```
def gradient_descent(g, z0, alpha=0.01, delta=1.e-6, k_max=100000, stor
    z, z_old = z0, z0.copy()
    history = [z] if store else None
    for k in range(k_max):
        z = z_old - alpha * g(z_old)
        if store:
            history.append(z)
        if np.linalg.norm(z - z_old) <= delta:
            return (z, history) if store else z
        z_old = z
    print('warning: maximum iterations exceeded')
    return (z, history) if store else z
```

In [41]:
```
import urllib.request
import ssl
from os import path as osp
import shutil
import pickle
```

In [42]:
```
def retrieve(file_name, semester='fall25', homework=3):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        context = ssl._create_unverified_context()
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci371/homework/
        url = fmt.format(semester, homework, file_name)
        with urllib.request.urlopen(url, context=context) as response:
            with open(file_name, 'wb') as file:
                shutil.copyfileobj(response, file)
        print('Downloaded file {}'.format(file_name))
```

In [43]:
```
file_name = 'logistic_samples.pkl'
retrieve(file_name)
```

```
with open(file_name, "rb") as file:
    training_set = pickle.load(file)
```

```
Using previously downloaded file logistic_samples.pkl
```

Given Code for the contour plot

In [44]:
```
def hz(z, x):
    a, b = np.array(z[0]), np.array(z[1])
    return 1 / (1 + a[..., np.newaxis] * np.exp(- b[..., np.newaxis] *
```

In [45]:
```
def risk(z, x, y):
    y_hat = hz(z, x)
    return np.squeeze(np.mean((y[np.newaxis, np.newaxis, ...] - y_hat)
```

In [46]:
```
def risk_on_t(z):
    return risk(z, training_set['x'], training_set['y'])


box, grid_samples = [0, 4, 0, 4], 200
z0, z1 = np.meshgrid(
    np.linspace(box[0], box[1], grid_samples),
    np.linspace(box[2], box[3], grid_samples)
)

risk_array = risk_on_t([z0, z1])
```

Code for the SGD

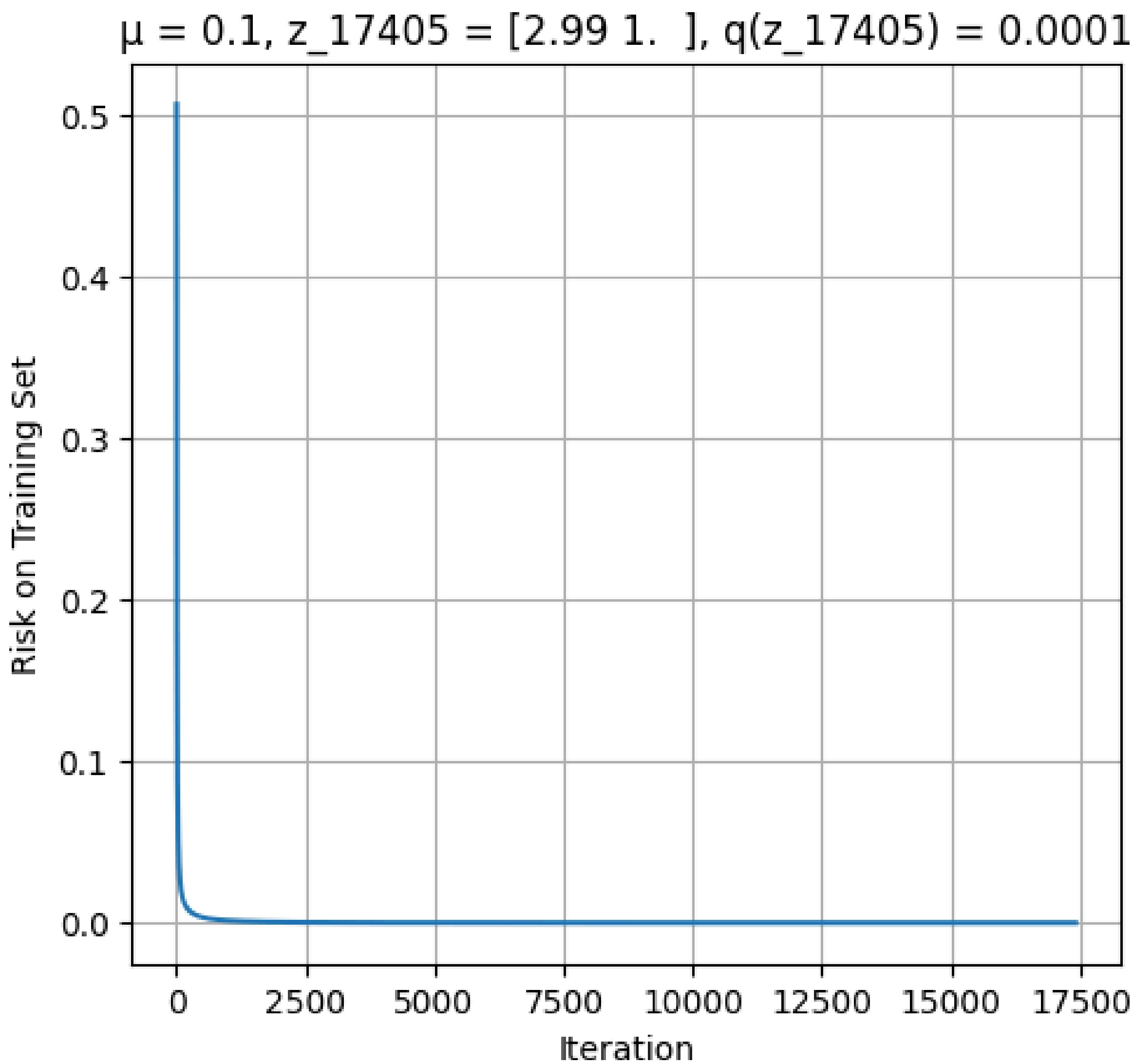
In [47]:
```
def gz(z, x):
    a, b = np.array(z[0]), np.array(z[1])
    return np.array([
        -np.exp(-b * x) / (1 + a*np.exp(-b * x))**2,
        a*x*np.exp(-b * x) / (1 + a*np.exp(-b * x))**2
    ])

def risk_gradient(z, x, y):
    y_hat = hz(z, x)
    return np.squeeze(np.mean(2 * gz(z, x) * (y_hat - y), axis=-1))
```

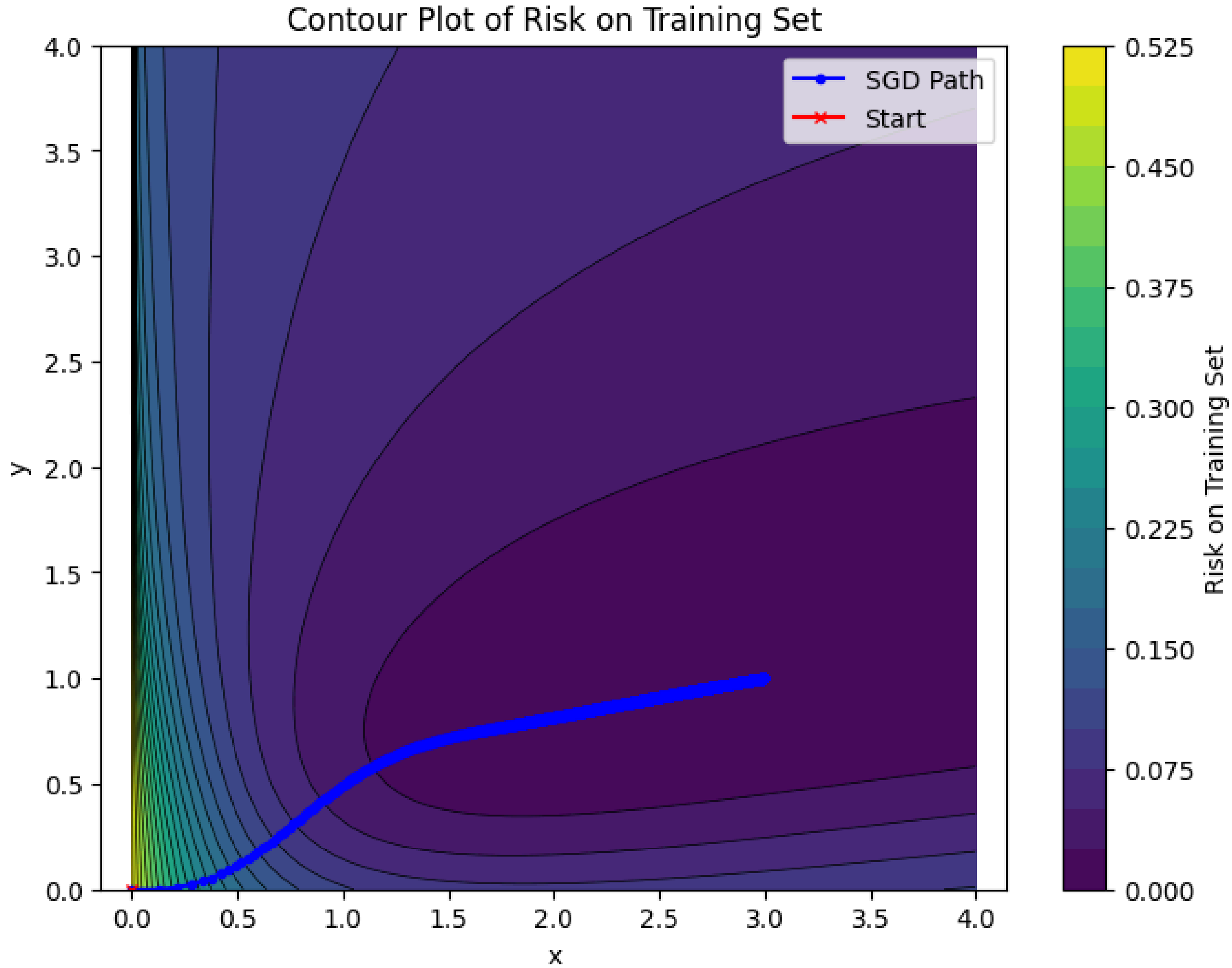
In [48]:
```
def sgd(g, t, z0, alpha=0.1, delta=1.e-6, k_max=100000, batch_size=None
    z, z_old = np.array(z0, dtype=float), np.array(z0, dtype=float)
    X, Y = t['x'], t['y']
    history = [z.copy()] if store else None
    n_samples = len(t['y'])

    rg = np.random.default_rng(3)
    batch_size = batch_size if batch_size is not None else n_samples
    batch_generator = batch_index_generator(n_samples, batch_size, rg)
```

```python
    for k in range(k_max):
        try:
            indices = next(batch_generator)
        except StopIteration:
            batch_generator = batch_index_generator(n_samples, batch_si
            indices = next(batch_generator)

        grad = g(z_old, X[indices], Y[indices])
        z = z_old - alpha * grad

        if store:
            history.append(z.copy())
        if np.linalg.norm(z - z_old) <= delta:
            return (z, history) if store else z
        z_old = z
    print('warning: maximum iterations exceeded')
    return (z, history) if store else z
```

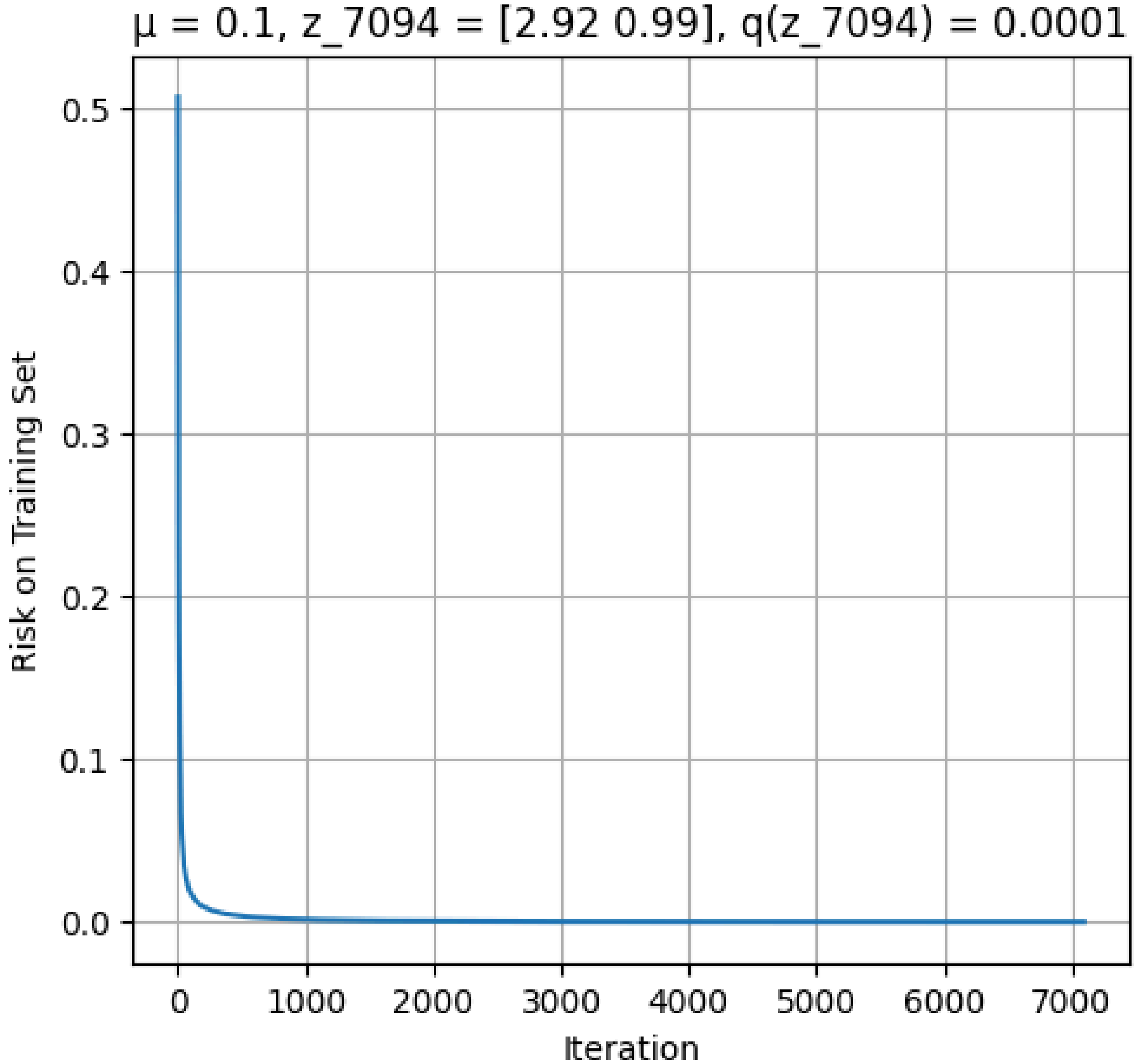
In [49]:
```python
def contour_path_plot(x, y, z, history):
    plt.figure(figsize=(8, 6))
    contour = plt.contourf(x, y, z, levels=20, cmap='viridis')
    plt.colorbar(contour, label='Risk on Training Set')
    plt.contour(x, y, z, levels=20, colors='black', linewidths=0.5)
    plt.axis('equal')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Contour Plot of Risk on Training Set')
    history = np.array(history)
    plt.plot(history[:, 0], history[:, 1], marker='o', color='blue', ma
    plt.plot(history[0, 0], history[0, 1], marker='x', color='red', mar
    plt.legend()
    plt.show()

def risk_iteration_plot(history, alpha=0.1):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    risk_history = [risk_on_t(p) for p in history]
    plt.plot(risk_history)
    plt.xlabel('Iteration')
    plt.ylabel('Risk on Training Set')

    last_step = len(history) - 1
    z_last = history[-1]
    q_z_last = risk_history[-1]
    title = (f'μ = {alpha}, z_{last_step} = {np.round(z_last, 2)}, '
             f'q(z_{last_step}) = {q_z_last:.4f}')
    plt.title(title)

    plt.grid(True)
```

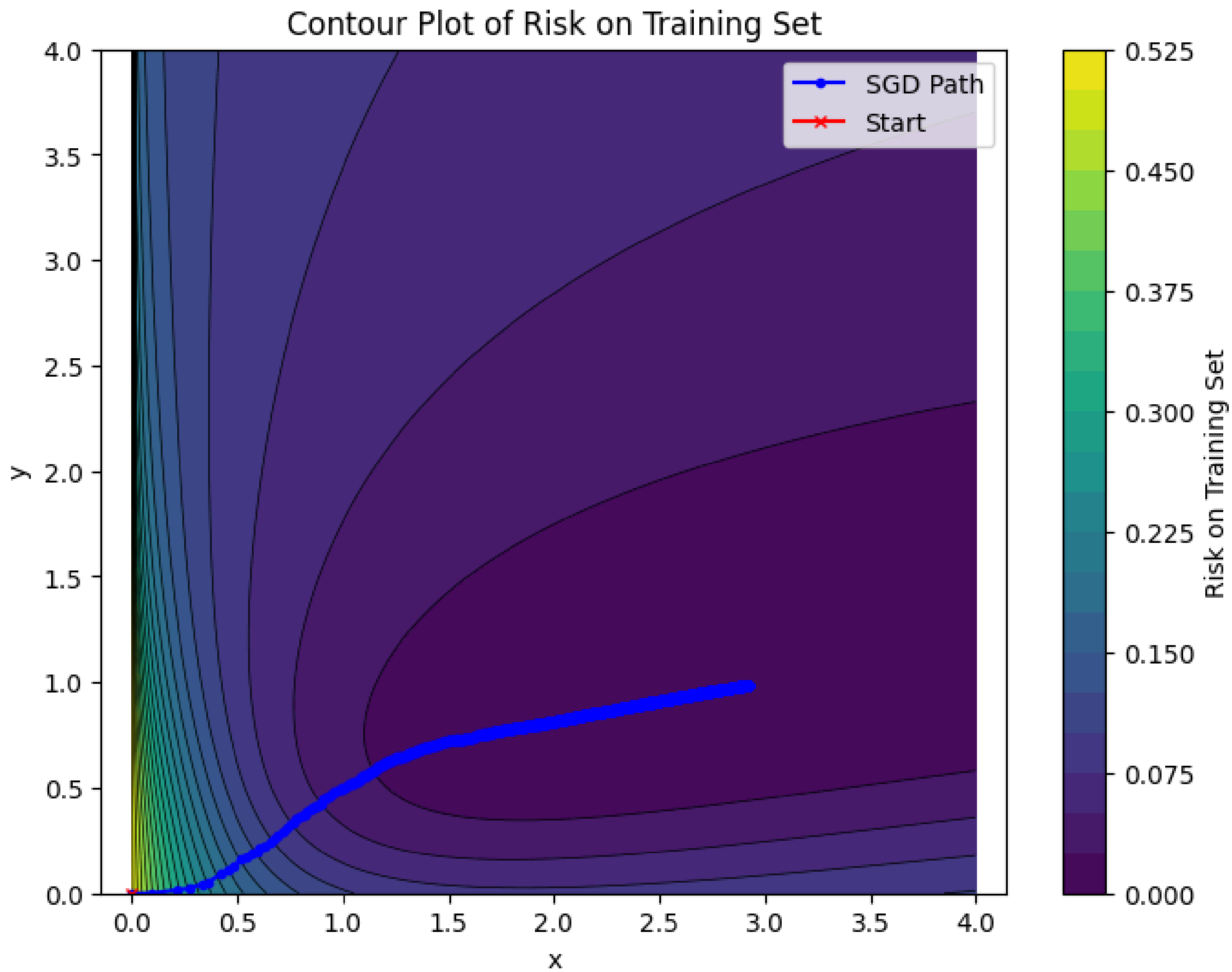
```
z_start = [0, 0]
z_final_batch, history_batch = sgd(risk_gradient, training_set, z_start
risk_iteration_plot(history_batch)
contour_path_plot(z0, z1, risk_array, history_batch)
```





In [50]:
```
z_final_batch, history_batch = sgd(risk_gradient, training_set, z_start
risk_iteration_plot(history_batch)
contour_path_plot(z0, z1, risk_array, history_batch)
```


μ = 0.1, z_7094 = [2.92 0.99], q(z_7094) = 0.0001
Risk on Training Set
0.5
0.4
0.3
0.2
0.1
0.0
0
1000
2000
3000
4000
5000
6000
7000
Iteration



Problem 4.3 (Exam Style)

> How does SGD compare with regular Gradient Descent (GD) when `batch_size` is equal to the number of samples in the training set?

Answer 1.

Since we are using a single batch we form the problem into our original GD and have the same performance.

> State the total number of batches (in a full run of `sgd`, not per epoch) that were processed with `batch_size` set to `None` and with `batch_size` set to `10`, respectively.

Asnwer 2.

- `batch_size = None` $\Rightarrow$ `last = 17404`
- `batch_size = 10` $\Rightarrow$ `last = 7094`

> Convert those numbers to cost units. That is, what is the total cost of your run when `sgd` is called with `batch_size` set to `None`? And when `batch_size` is `10`?

Answer 3

Processing a batch with k samples cost k cost units

For `batch_size = None` We default to processing the whole batch. This means `batch_size = 500` and we processed `10000` batches, one per iteration. Therefore we get

- `cost units = 17404 * 500 = 8,702,500`

For `batch_size = 10` We proces batches of 10. We processed `7094` batches, one per iteration. Therefore we get

- `cost units = 7094 * 10 = 70,940`

> Can SGD beat GD in terms of computational efficiency?

Answer 4

SGD clearly beats our version with `batch_size = None` which is equivalent to GD. Therefore it is a lot more cost efficient, a lot less cost units for training.