

```
In [ ]: import autograd.numpy as anp
        from autograd import grad
        import matplotlib.pyplot as plt
```

Question 3.5

```
In [ ]: def gradient_descent(alpha, max_its, w_init):

    # Define the cost and its gradient functions
    def g(w):
        return 1/50 * (w**4 + w**2 + 10*w)

    def grad_g(w):
        return 1/50 * (4*w**3 + 2*w + 10)

    # Initialize variables
    w = w_init
    cost_history = [g(w)]

    for _ in range(max_its):
        w -= alpha * grad_g(w) # Update step
        cost_history.append(g(w)) # Store the cost

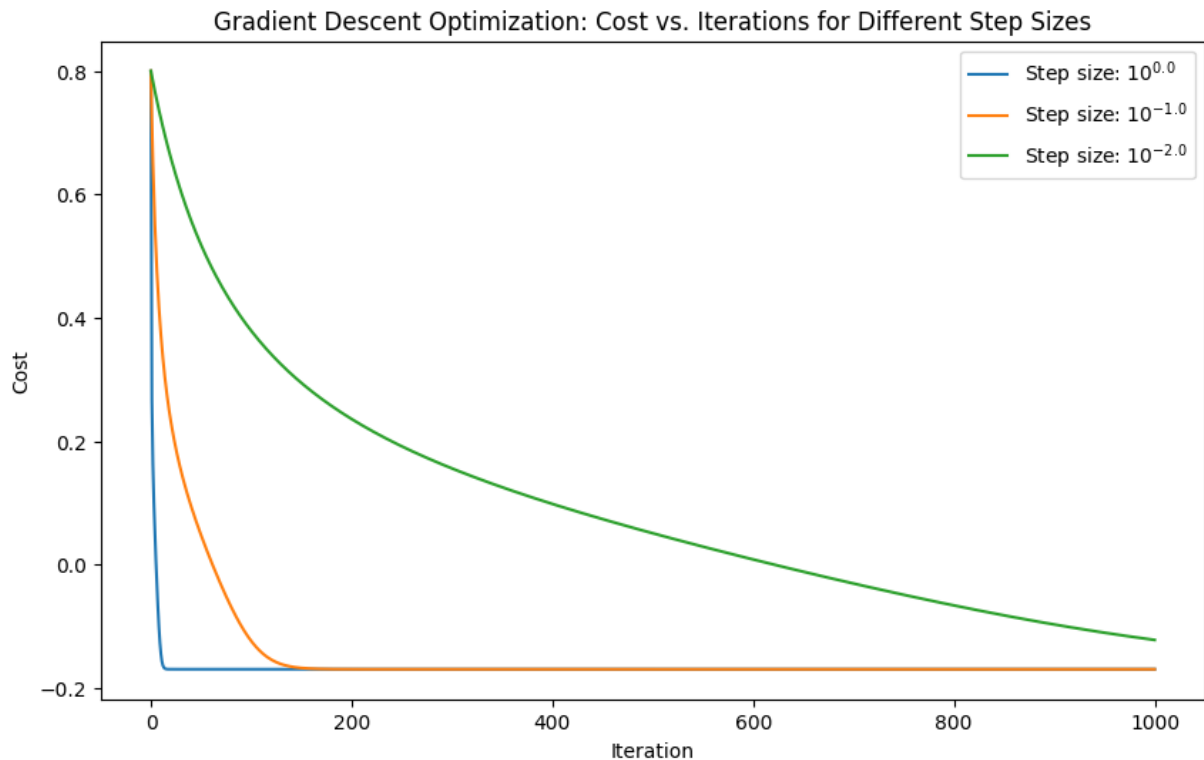
    return cost_history
```

```
In [ ]: initial_weight = 2.0
        iterations = 1000
        step_sizes = [10**0, 10**(-1), 10**(-2)]
        histories = []

        for step_size in step_sizes:
            histories.append(gradient_descent(step_size, iterations, initial_weight))

        plt.figure(figsize=(10, 6))
        for index, history in enumerate(histories):
            plt.plot(history, label=f'Step size:  $10^{\{\{np.log10(step\_sizes[index])\}}$ ')

        plt.xlabel('Iteration')
        plt.ylabel('Cost')
        plt.title('Gradient Descent Optimization: Cost vs. Iterations for Different')
        plt.legend()
        plt.show()
```



Question 3.8

```
In [ ]: def gradient_descent(function_to_optimize, step_size, max_iterations, initial_weight):
    compute_gradient = grad(function_to_optimize)

    weights_history = [initial_weight]
    costs_history = [function_to_optimize(initial_weight)]

    for _ in range(max_iterations):
        current_gradient = compute_gradient(weights_history[-1])

        updated_weight = weights_history[-1] - step_size * current_gradient

        weights_history.append(updated_weight)
        costs_history.append(function_to_optimize(updated_weight))

    return weights_history, costs_history
```

```
In [ ]: def squared_norm(w):
    return np.dot(w.T, w)[0, 0]
```

```
In [ ]: def g(w):
    return squared_norm(w)

N = 10
initial_weight = 10 * np.ones((N, 1))
max_iterations = 100
step_sizes = [10**0, 10**(-1), 10**(-2)]
```

```

results = []
for alpha in step_sizes:
    weights_history, costs_history = gradient_descent(g, alpha, max_iteratic
    results.append(costs_history)

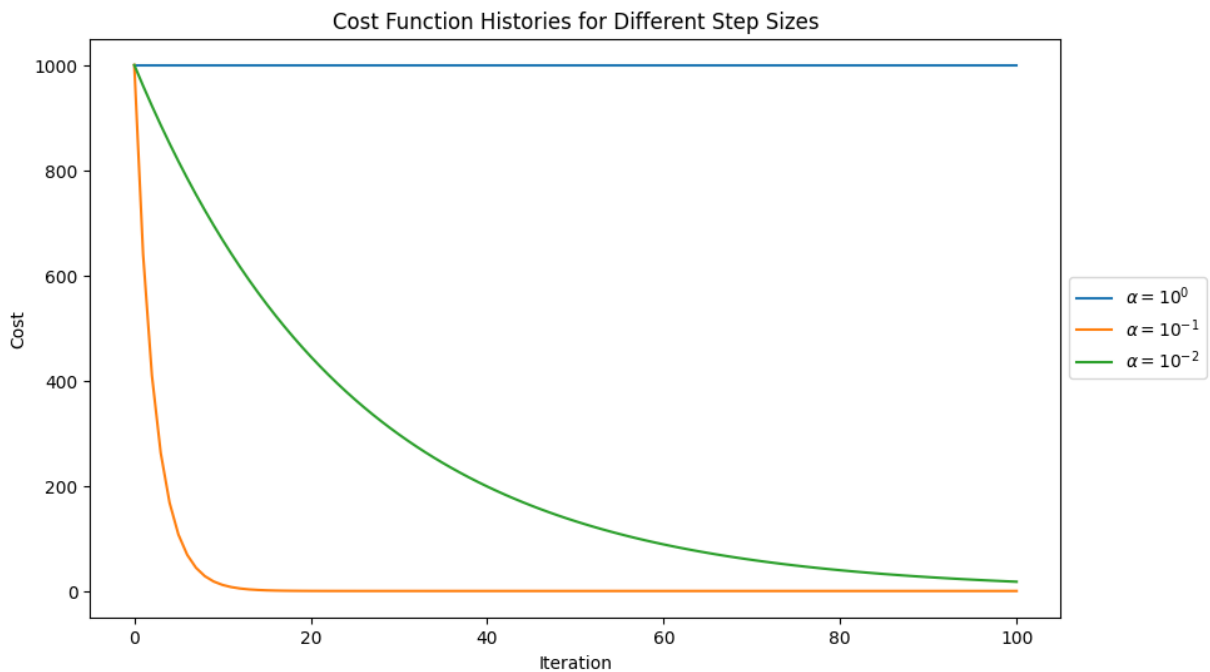
def plot_cost_histories(cost_histories, labels):

    plt.figure(figsize=(10, 6))
    for history, label in zip(cost_histories, labels):
        plt.plot(history, label=label)
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xlabel('Iteration')
    plt.ylabel('Cost')
    plt.title('Cost Function Histories for Different Step Sizes')
    plt.show()

labels = [r'$\alpha = 10^{\{0\}}$', r'$\alpha = 10^{\{-1\}}$', r'$\alpha = 10^{\{-2\}}$']

plot_cost_histories(results, labels)

```



4. |

a) For any vector z , the curvature function of a matrix C with non-negative eigenvalues d_n is

$$\psi(z) = z^T C z = z^T \left(\sum_{n=1}^N e_n e_n^T d_n \right) z = \sum_{n=1}^N (e_n^T z)^2 d_n$$
$$\psi(z) \geq 0$$

This expression is non-negative due to the non-negativity of $(e_n^T z)^2$ and d_n , confirming that C is positive semidefinite.

b) Conversely, if C is positive semidefinite,

its curvature function $\psi(z)$ must satisfy:

$$\psi(z) = z^T C z \geq 0$$

This requirement is inherently fulfilled by sum of squares form in its eigen decomposition, which implies that all eigenvalues d_n are non-negative. Any negative eigenvalue would violate the positive semidefiniteness by producing a negative curvature function for z aligned with its corresponding eigenvector

c), The matrix $C = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, possessing eigenvalues $d_1 = 0$, $d_2 = 2$, is thus positive semidefinite, indicating the convexity of the associated function g .

(d) Enhancing matrix C with $\lambda I_{n \times n}$, where

λ is a scalar, modifies the curvature function

to:

$$\psi(z) = \sum_{n=1}^N (e_n^T z)^2 (d_n + \lambda)$$

To ensure semipositivity, λ must be chosen to offset the smallest negative eigenvalue of C .

This augmentation guarantees that $\psi(z)$ remains non-negative for any z , solidifying the positive semidefinite status of $C + \lambda I_{n \times n}$

4.5

(a) set $g(w) = \log(1 + e^{w^T w})$ to zero

$$\nabla g(w) = \frac{2e^{w^T w}}{1 + e^{w^T w}} = 0_{N \times 1}$$

As $\frac{2e^{w^T w}}{1 + e^{w^T w}} \geq 1$ only when $w = 0_{N \times 1}$

(b) The Hessian of g ,

$$\nabla^2 g(w) = \left(\frac{4e^{w^T w}}{(1 + e^{w^T w})^2} w w^T + \frac{2e^{w^T w}}{1 + e^{w^T w}} I_{N \times N} \right)$$

Fixing w

$$\begin{aligned} z^T \nabla^2 g(w) z &= z^T \left(\frac{4e^{w^T w}}{(1 + e^{w^T w})^2} w w^T + \frac{2e^{w^T w}}{1 + e^{w^T w}} I_{N \times N} \right) z \\ &= z^T \nabla^2 g(w) z = z^T \frac{4e^{w^T w}}{(1 + e^{w^T w})^2} w w^T z + z^T \frac{2e^{w^T w}}{1 + e^{w^T w}} I_{N \times N} z \\ &= \frac{4e^{w^T w}}{(1 + e^{w^T w})^2} z^T w w^T z + \frac{2e^{w^T w}}{1 + e^{w^T w}} z^T z \\ &= \frac{4e^{w^T w}}{(1 + e^{w^T w})^2} (z^T w)^2 + \frac{2e^{w^T w}}{1 + e^{w^T w}} \|z\|_2^2 \end{aligned}$$

all component is nonnegative whatever z is

so $z^T \nabla^2 g(w) z \geq 0$

for all w and z , g is indeed convex

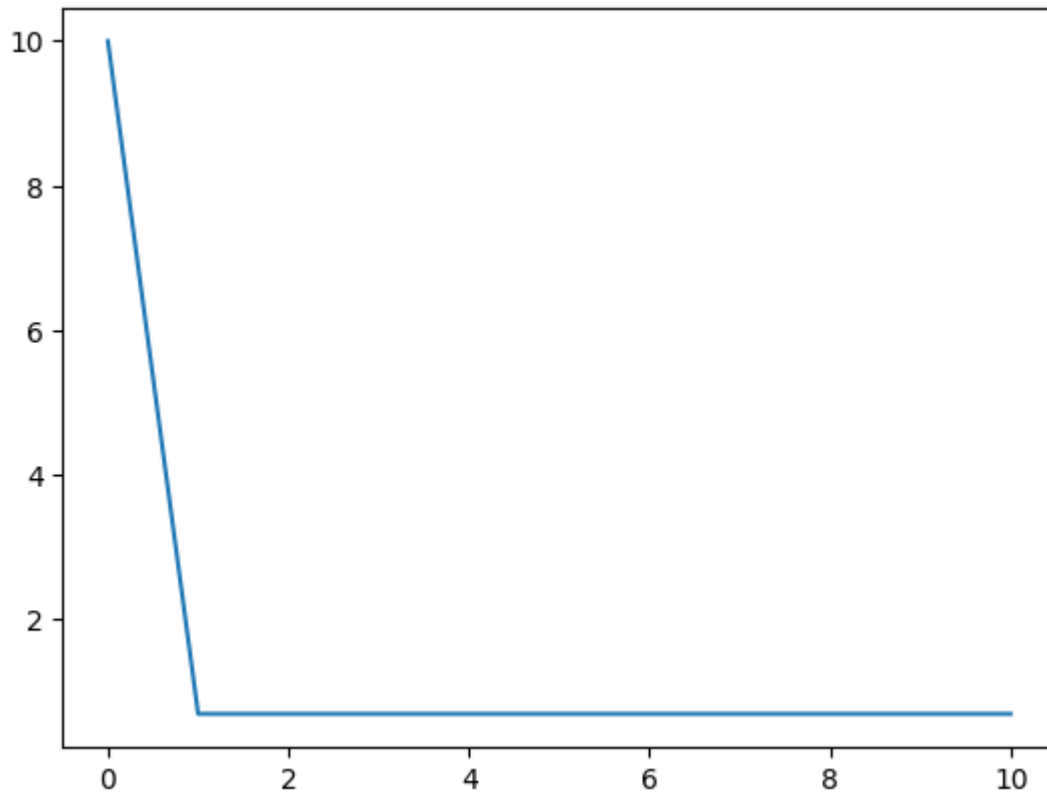
Question4.5

```
In [ ]: from autograd import grad
        from autograd import hessian
        import matplotlib.pyplot as plt
        from autograd import numpy as np

        def g(w):
            return np.log(1+np.exp(np.dot(w.T, w)))

        def newton_method(g, max_its, w, **kwargs):
            gradient = grad(g)
            hess = hessian(g)
            epsilon = 10**(-7)
            if 'epsilon' in kwargs:
                epsilon = kwargs['epsilon']
            weight_history = [w]
            cost_history = [g(w)]
            for k in range(max_its):
                grad_eval = gradient(w)
                hess_eval = hess(w)
                hess_eval.shape = (int((np.size(hess_eval))*(0.5)), int((np.size(hess_eval))*(0.5)))
                A = hess_eval + epsilon*np.eye(w.size)
                b = grad_eval
                w = np.linalg.solve(A, np.dot(A, w) - b)
                weight_history.append(w)
                cost_history.append(g(w))
            cost_history = [np.squeeze(val) for val in cost_history]
            return weight_history, cost_history

        N = 10
        w0 = np.ones((N, 1))
        weights, cost_history = newton_method(g, 10, w0)
        plt.plot(cost_history)
        plt.show()
```

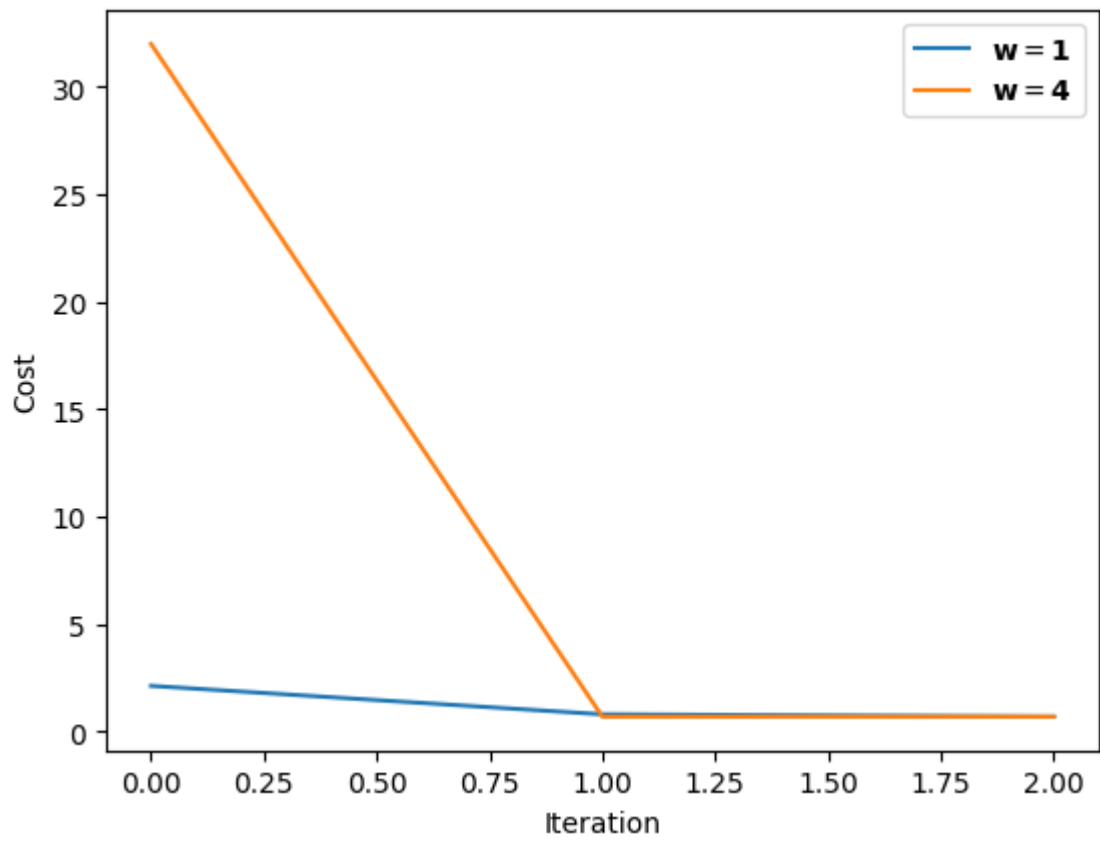


```
In [ ]: import matplotlib.pyplot as plt

def plot_cost_histories(cost_histories, labels, start=0, points=False):
    for cost_history, label in zip(cost_histories, labels):
        if points:
            plt.scatter(range(start, len(cost_history)), cost_history[start:])
        else:
            plt.plot(range(start, len(cost_history)), cost_history[start:],

    plt.xlabel('Iteration')
    plt.ylabel('Cost')
    plt.legend()
    plt.show()

plot_cost_histories([cost_history, cost_history_2], labels=[r'$\mathbf{w}=\mathbf{n}$', r'$\mathbf{w}=\mathbf{m}$'])
```



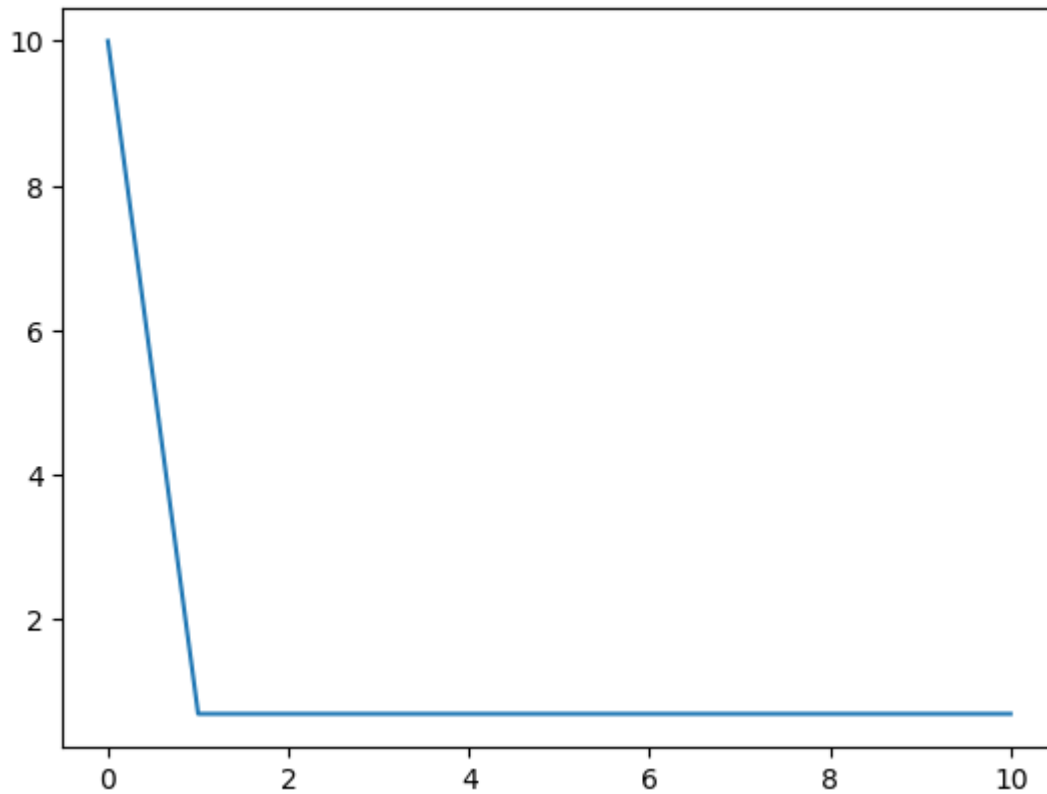
Question4.5

```
In [ ]: from autograd import grad
        from autograd import hessian
        import matplotlib.pyplot as plt
        from autograd import numpy as np

        def g(w):
            return np.log(1+np.exp(np.dot(w.T, w)))

        def newton_method(g, max_its, w, **kwargs):
            gradient = grad(g)
            hess = hessian(g)
            epsilon = 10**(-7)
            if 'epsilon' in kwargs:
                epsilon = kwargs['epsilon']
            weight_history = [w]
            cost_history = [g(w)]
            for k in range(max_its):
                grad_eval = gradient(w)
                hess_eval = hess(w)
                hess_eval.shape = (int((np.size(hess_eval))*(0.5)), int((np.size(hess_eval))*(0.5)))
                A = hess_eval + epsilon*np.eye(w.size)
                b = grad_eval
                w = np.linalg.solve(A, np.dot(A, w) - b)
                weight_history.append(w)
                cost_history.append(g(w))
            cost_history = [np.squeeze(val) for val in cost_history]
            return weight_history, cost_history

        N = 10
        w0 = np.ones((N, 1))
        weights, cost_history = newton_method(g, 10, w0)
        plt.plot(cost_history)
        plt.show()
```



```
In [ ]: import matplotlib.pyplot as plt

def plot_cost_histories(cost_histories, labels, start=0, points=False):
    for cost_history, label in zip(cost_histories, labels):
        if points:
            plt.scatter(range(start, len(cost_history)), cost_history[start:])
        else:
            plt.plot(range(start, len(cost_history)), cost_history[start:],

            plt.xlabel('Iteration')
            plt.ylabel('Cost')
            plt.legend()
            plt.show()

w = np.ones((2,)); max_its = 2;
weight_history, cost_history = newtons_method(g, max_its, w)

w = 4*np.ones((2,)); max_its = 2;
weight_history_2, cost_history_2 = newtons_method(g, max_its, w)

plot_cost_histories([cost_history, cost_history_2], labels=[r'$\mathbf{w}=\mathbf{n}$', r'$\mathbf{w}=\mathbf{4n}$'])
```

