1.

$$f(x_t) - f(x^*) \leq (1 - \mu \cdot c)^t \left(f(x_0) - f(x^*)\right)$$

$$\left[\mu = \frac{1}{2} = \frac{1}{4}\right]\left[c = 2\right]\left\{f(x_0) - f(x^*) \leq 5\right\}$$

$$f(x_t) - f(x^*) \leq \left(1 - \frac{1}{2}\right)^t \cdot 5$$

$$f(x_t) - f(x^*) \leq \frac{5}{2^t} \leq 10^{-4}$$

$$5 \cdot 10^4 \leq 2^t \rightarrow \log(5 \cdot 10^4) \leq t \cdot \log(2)$$

$$t \geq \frac{\log(5 \cdot 10^4)}{\log(2)} \approx 15,6$$

Answer: 16 times

2. a)

① Identify set of indices with the largest entries.

$$I = \underset{|I|=s}{\operatorname{argmax}} \sum_{i \in I} x_i$$

$$\overline{\Omega}_B(x)_i = \begin{cases} \max(x_i, 0) & \text{if } i \in I \\ 0 & \text{else} \end{cases}$$

$$\begin{cases} x_{t+1} = \Omega_B(x) \end{cases}$$

---

2 (6) $\nabla f(x) = A^T(Ax-b)$

$$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{t+1} = \overline{\Omega}_B\left[x_t - \eta \cdot (A^T(Ax-b))\right] \end{cases}$$

2c) $\nabla f(x) = A^T(Ax - b)$

Let $x_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \in B$

$$\nabla f(x_0) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\Rightarrow \nabla f(x_0) = \begin{bmatrix} -4 \\ 0 \end{bmatrix}$$

$x_1 = \bar{\Omega}_B(p)$ where $p = x_0 - \mu \nabla f(x_0) \rightarrow p = \begin{bmatrix} 4\mu \\ 1 \end{bmatrix}$

let $\mu = \frac{1}{8} \Rightarrow p = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}$

$$\bar{\Omega}_B\left(\begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = x_1$$

This means we have a cycle!

$f\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) < f\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \Rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ is not global minimizer, which means the projected GD does not converge for all $x_0 \in B$

3.

error = 2.985671286817389e-08

nonzero indices are 0 and 14 in the x-array of length 50.
(i made the x-array of length 50 because otherwise i couldn't multiply it with A)

```
x = [0.99997799 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         2.9999611  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         ]
```

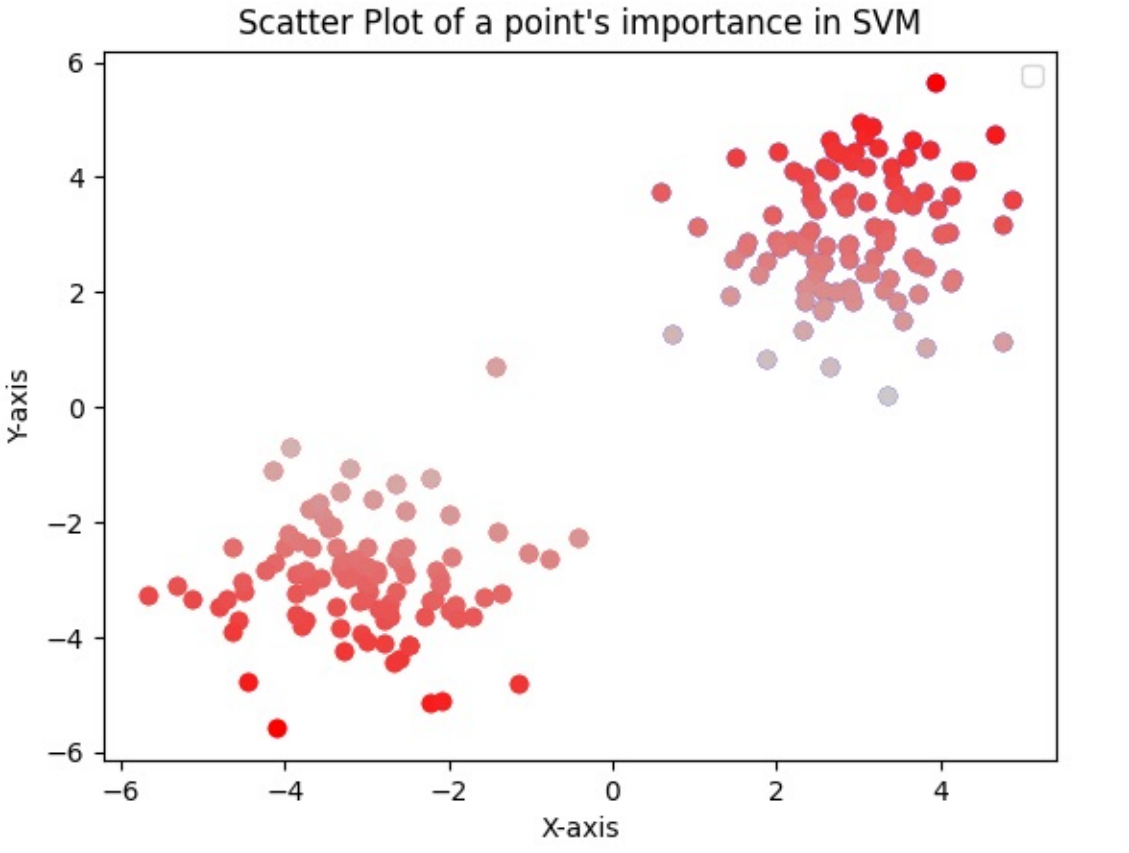
```python
from matplotlib import cm
from matplotlib.colors import LinearSegmentedColormap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('HW7Q4.csv', header=None)
blue = df.iloc[:100, :2].to_numpy().astype(float)
red = df.iloc[100:, :2].to_numpy().astype(float)
all_points = df.iloc[:].to_numpy().astype(float)
start = np.array([-1,1],dtype=float)


# plt.scatter(blue[:, 0], blue[:, 1], color='blue', label='Blue')
# plt.scatter(red[:, 0], red[:, 1], color='red', label='Red')

# # Add labels and title
# plt.xlabel('X-axis')
# plt.ylabel('Y-axis')
# plt.title('Scatter Plot of Blue and Red Points')
# plt.legend()
# plt.savefig('scatter_plt.png')
# plt.show()

def indicator(weights:np.ndarray, index:int):
    if 1 - np.dot(all_points[index][:2], weights) * all_points[index][2] > 0:
          return 1
    return 0
```

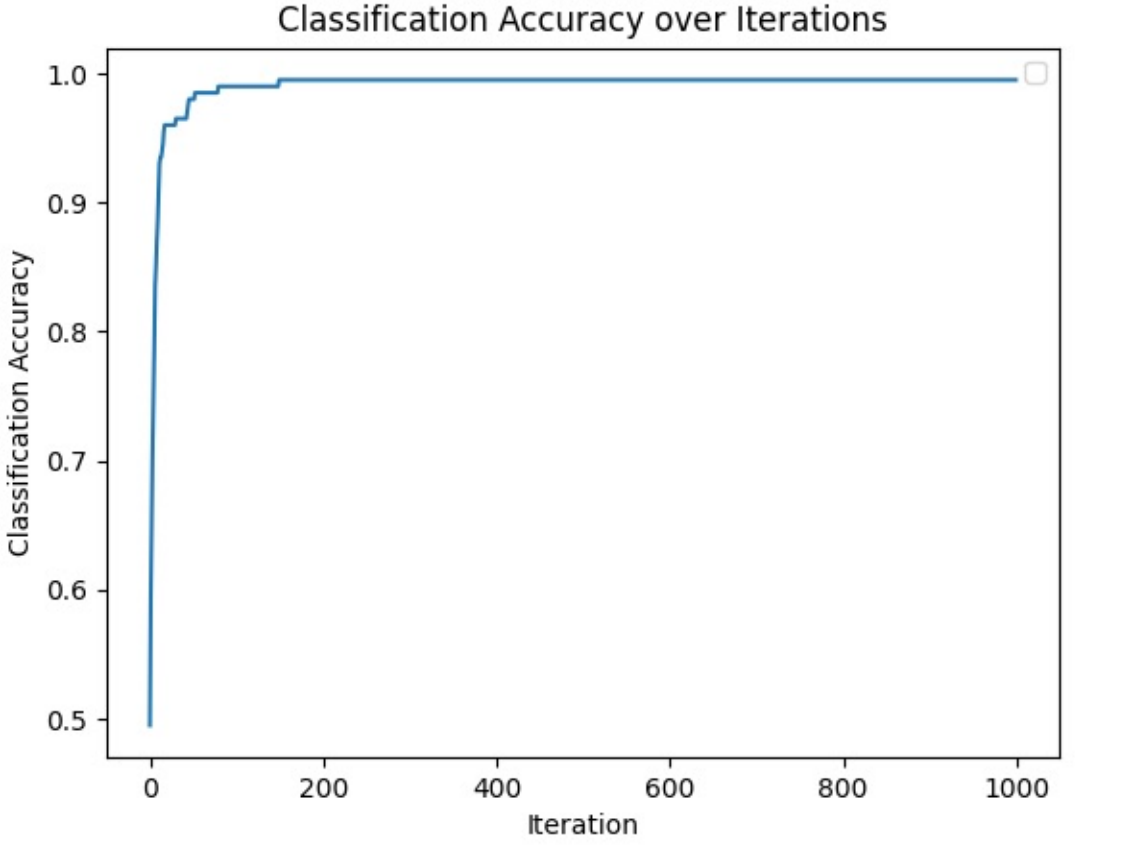
```python
def loss(w:np.ndarray) -> float:
    term = 0.5 * np.linalg.norm(w)**2
    for i in range(200):
        inner_product = np.dot(all_points[i][:2], w) * all_points[i][2]
        term += max(0, 1 - inner_product)
    return term

def loss_grad(weight:np.ndarray) -> float:
    term = weight.copy()
    for i in range(200):
        term -= all_points[i][2] * all_points[i][:2] * indicator(index=i, weights=weight)
    return term.astype(float)

def classification(w:np.ndarray) -> float:
    num = 0
    tot = 0
    for index in range(200):
        num += 1 if np.dot(all_points[index][:2], w) * all_points[index][2] > 0 else 0
        tot += 1
    return num / tot


def gd(w:np.array= start, lr:float=1e-4,iterations:int=10**3):
    errorList = [classification(w)]
    margin = [2 / np.linalg.norm(w)]
    for i in range(iterations-1):
        w = w - lr * loss_grad(w)
```

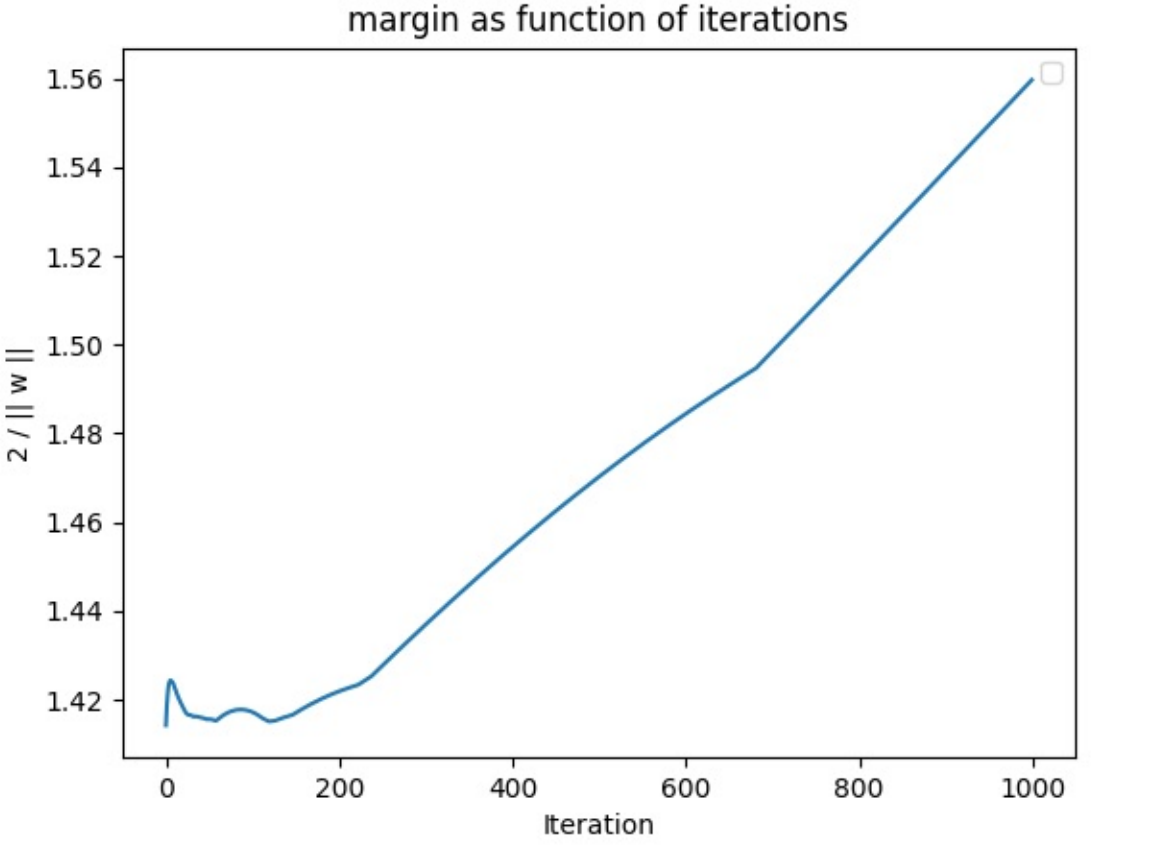
```python
        errorList.append(classification(w))
        margin.append(2 / np.linalg.norm(w))
        # if i % 100 == 0:
            # print(loss(w))
    return w, errorList, margin

def importance_function(w:np.ndarray) -> float:
    # print(1 - all_points[index][2] * np.dot(all_points[index][:2],w))
    list = []
    for index in range(200):
        list.append(np.abs(1 - all_points[index][2] * np.dot(all_points[index][:2],w)))
    maxval = max(list)
    for item in range(200):
        list[item] /= maxval
    return list

w, errorList, margin = gd()
iterationList = [i for i in range(10**3)]

plt.plot(iterationList, errorList)
plt.xlabel('Iteration')
plt.ylabel('Classification Accuracy')
plt.title('Classification Accuracy over Iterations')
plt.legend()
plt.savefig('accuracy.png')
plt.show()

plt.plot(iterationList, margin)
plt.xlabel('Iteration')
plt.ylabel('2 / || w ||')
```

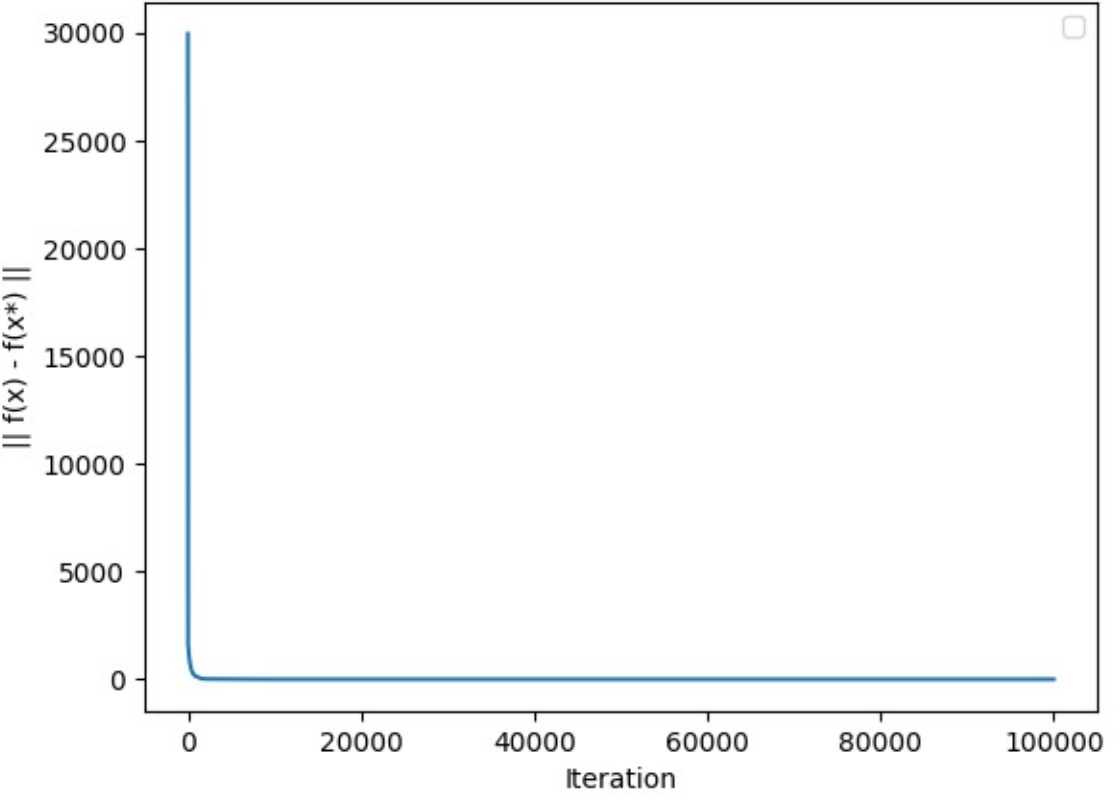
```python
plt.title('margin as function of iterations')
plt.legend()
plt.savefig('margin.png')
plt.show()


#scatter plot
colors = [(0.8, 0.8, 0.8), (1, 0, 0)]  # Light grey to red
n_bins = 100  # Discretize the colormap into 100 bins
cmap_name = 'custom_cmap'
custom_cmap = LinearSegmentedColormap.from_list(cmap_name, colors, N=n_bins)
plt.scatter(blue[:, 0], blue[:, 1], color='blue')
plt.scatter(red[:, 0], red[:, 1], color='red')

colorList = importance_function(w=w)
print(colorList)

for i in range(len(all_points)):
    plt.scatter(all_points[i][0], all_points[i][1], color=custom_cmap(np.abs(colorList[i])))


# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Color indicates a point\'s importance in SVM')
plt.legend()
plt.savefig('scatter_plt.png')
plt.show()
```

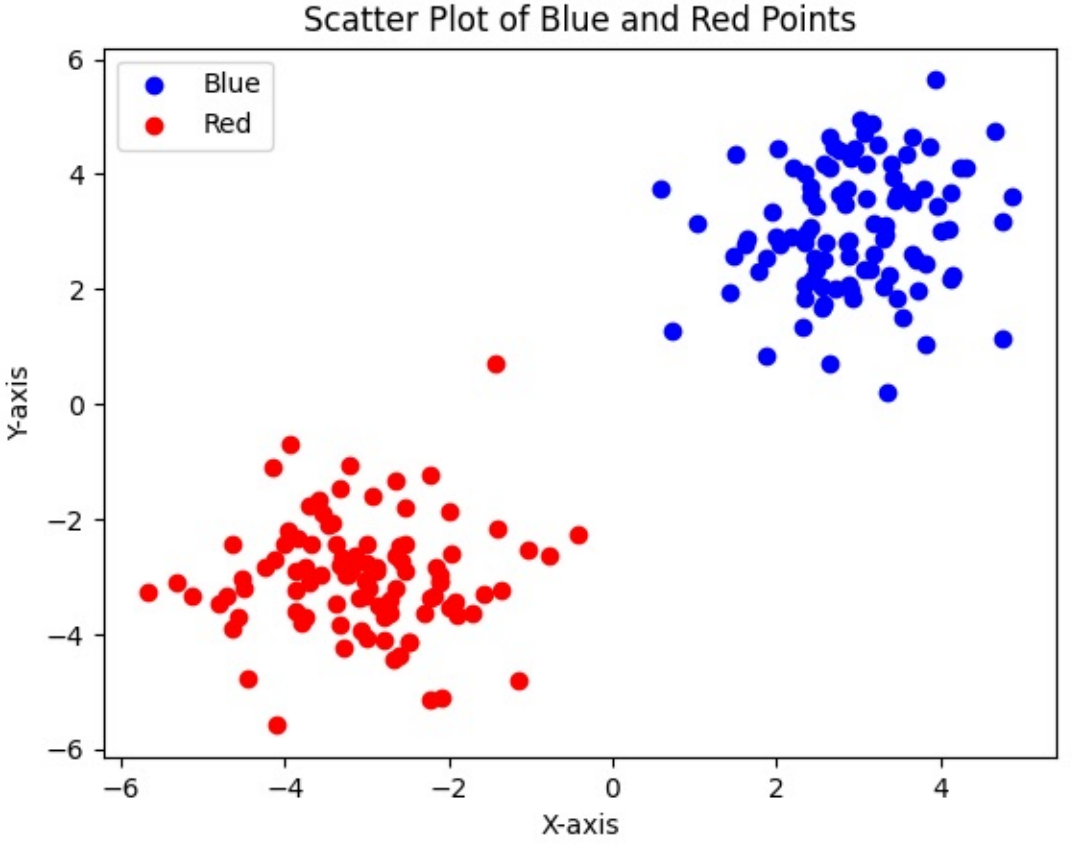
```python
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('HW7Q3.csv', header=None)


A = df.iloc[:, :50].to_numpy()
b = df.iloc[:, 50:].to_numpy().flatten()
s = 48
iterations = 10**5
x_optimal = np.zeros(50, dtype=float)
x_optimal[0] = 1
x_optimal[14] = 3

def function(x:np.ndarray) -> np.ndarray:
    return 0.5*(np.linalg.norm(np.dot(A, x) - b)**2)

def gradient(x:np.ndarray) -> np.ndarray:
    return np.dot(A.T, (A @ x) - b)

def projection(x:np.ndarray, s:int) -> np.ndarray:
    arr = x.copy()
    for _ in range(s):
        min_index = np.argmin(arr)
        x[min_index] = 0
        arr[min_index] = np.finfo(np.float64).max
    return x

def gradient_descent(x:np.ndarray, lr:float=1e-4) -> float:
    f_optimal = function(x_optimal)
    ylist = [np.abs(function(x) - f_optimal)]
    for i in range(iterations):
        x = projection(x - lr * gradient(x), s)
        ylist.append(np.abs(function(x) - f_optimal))
        # if i % 100 == 0:
    print(function(x))
    print(x)
    return ylist

ylist = gradient_descent(np.array([np.random.uniform(0,10) for _ in range(50)]))
iterationList = [i for i in range(iterations+1)]
plt.plot(iterationList, ylist)
plt.xlabel('Iteration')
plt.ylabel('|| f(x) - f(x*) ||')
plt.legend()
plt.savefig('q3_plot.png')
plt.show()
```


Scatter Plot of a point's importance in SVM
Y-axis
X-axis
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4


Classification Accuracy over Iterations
Classification Accuracy
Iteration
1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000


margin as function of iterations
1.56
1.54
1.52
1.50
1.48
1.46
1.44
1.42
2 / || w ||
0
200
400
600
800
1000
Iteration


30000
25000
20000
15000
10000
5000
0
|| f(x) - f(x*) ||
0
20000
40000
60000
80000
100000
Iteration


Scatter Plot of Blue and Red Points
Blue
Red
Y-axis
X-axis
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4