In [1]:

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

In [2]:

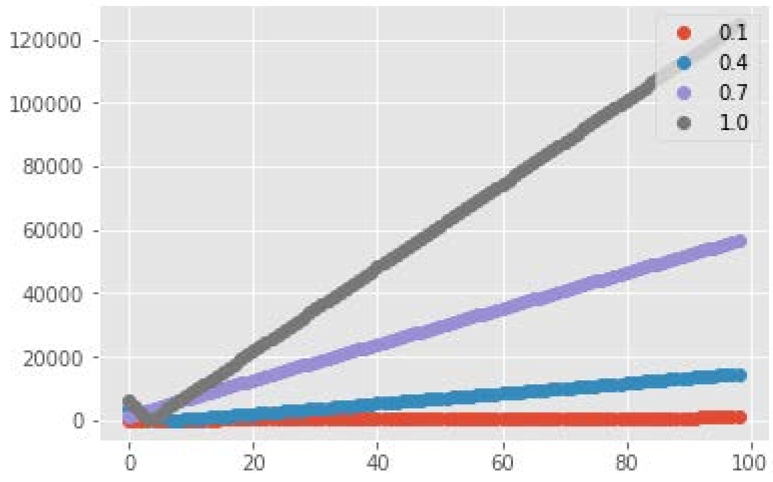
```
plt.style.use(['ggplot'])
```

In [3]:

```
alphas = np.linspace(0.1,1,4)
```

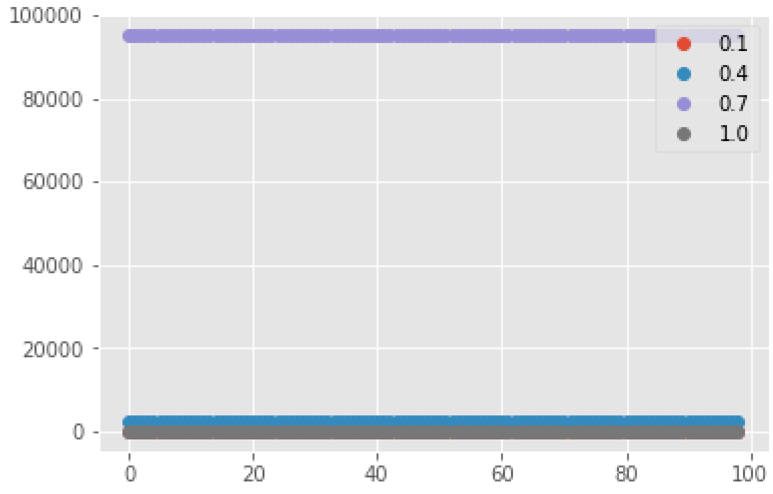
In [4]:

```
"""
Example 1: The vectors c i have the form c i = (ξ i , ζ i ), where
ξ i , ζ i , as well as b i , are chosen randomly and independently
from [−100, 100] according to a uniform distribution.
"""
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(-100,100,m)
    b = np.random.choice(dist, 1)
    c = np.random.choice(dist, 2)
    for k in range(1,m):
        if (c*x[k]>b).all:
            x[k] = x[k-1] - alpha * np.sum(c)
        else:
            x[k] = x[k-1]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```



In [34]:

```
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(-100,100,m)
    b = np.random.choice(dist, 1)
    c = np.random.choice(dist, 2)
    phi = np.random.rand(m,m)
    for k in range(1,m):
        for i in range(1,m):
            if (c*phi[i-1,k]>b).all:
                phi[i,k]=phi[i-1,k] - alpha * np.sum(c)
            else:
                phi[i,k]=phi[i-1,k]
            x[k]=phi[i,k]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```

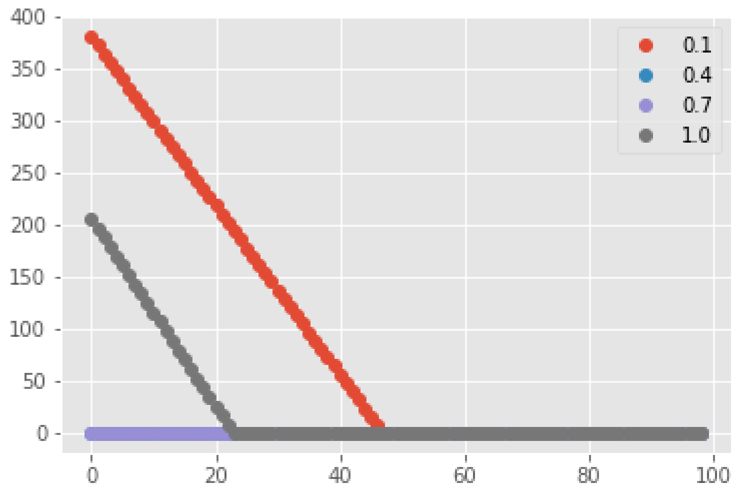


In [6]:

"""

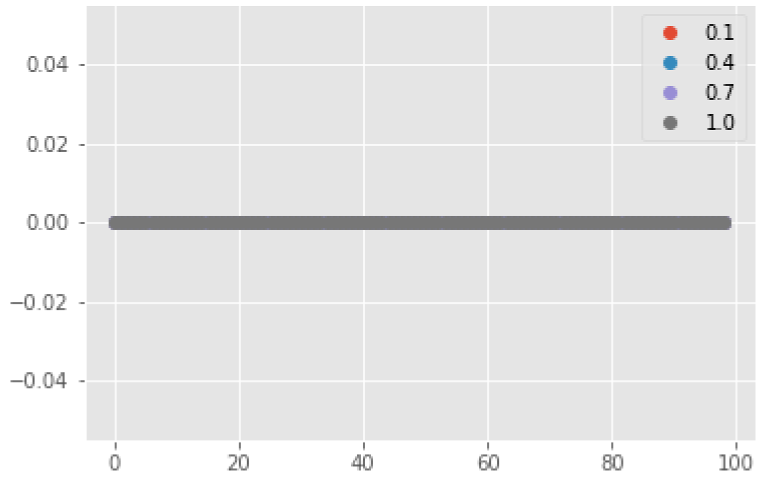
Example 2: The vectors c i have the form c i = (ξ i , ζ i), where ξ i

```
and ζ i are chosen randomly and independently from [−10, 10], while b
i is chosen randomly and independently within [0, 1000] according to a
uniform distribution.
"""
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(0,1000,m)
    b = np.random.choice(dist, 1)
    c = np.array([np.random.randint(-10,10),np.random.randint(-10,10)])
    for k in range(1,m):
        if (c*x[k]>b).all:
            x[k] = x[k-1] - alpha * np.sum(c)
        else:
            x[k] = x[k-1]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```



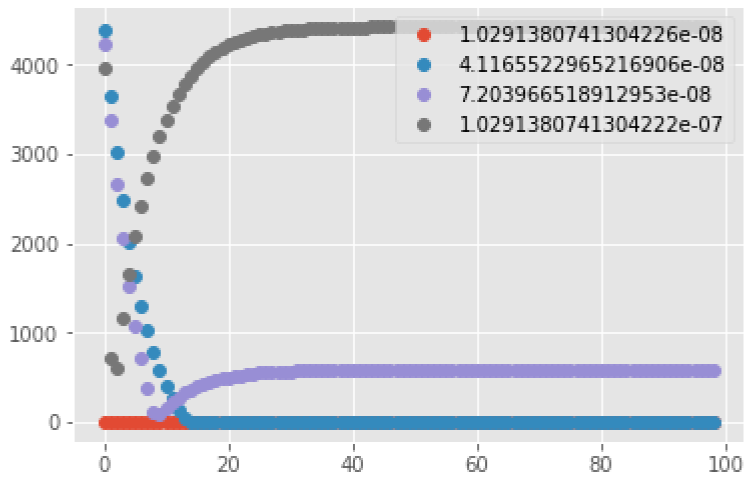
In [33]:

```
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(0,1000,m)
    b = np.random.choice(dist, 1)
    c = np.array([np.random.randint(-10,10),np.random.randint(-10,10)])
    phi = np.random.rand(m,m)
    for k in range(1,m):
        for i in range(1,m):
            if (c*phi[i-1,k]>b).all:
                phi[i,k]=phi[i-1,k] - alpha * np.sum(c)
            else:
                phi[i,k]=phi[i-1,k]
            x[k]=phi[i,k]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```



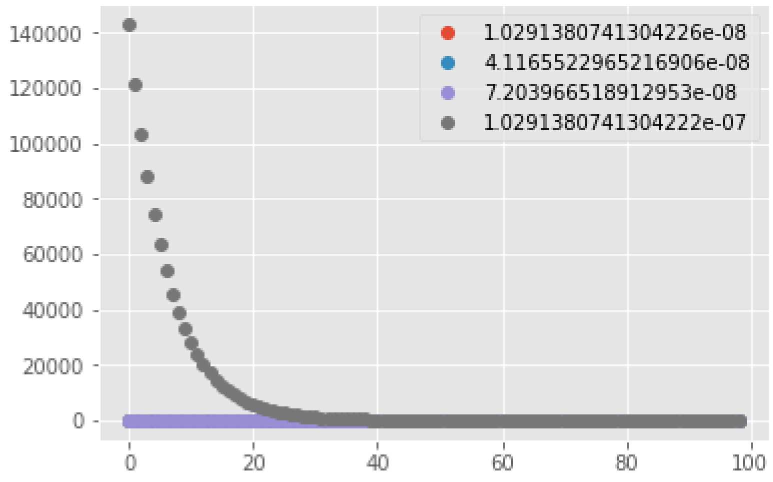
In [8]:

```
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(-100,100,m)
    b = np.random.choice(dist, 1)
    c = np.random.choice(dist, 2)
    for k in range(1,m):
        alpha = alpha*0.85
        if (c*x[k]>b).all:
            x[k] = x[k-1] - alpha * np.sum(c)
        else:
            x[k] = x[k-1]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```



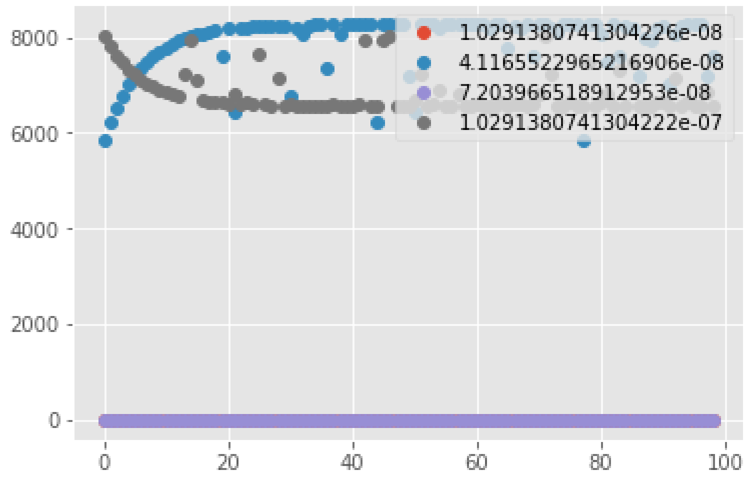
In [9]:

```
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(-100,100,m)
    b = np.random.choice(dist, 1)
    c = np.random.choice(dist, 2)
    phi = np.random.rand(m,m)
    for k in range(1,m):
        alpha = alpha*0.85
        for i in range(1,m):
            if (c*phi[i-1,k]>b).all:
                phi[i,k]=phi[i-1,k]-alpha * c[0]-alpha * c[1]
            else:
                phi[i,k]=phi[i-1,k]
            x[k]=phi[i,k]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```



In [32]:

```
for alpha in alphas:
    y = []
    m = 100
    x = np.zeros([m,1])+100
    dist = np.random.uniform(-100,100,m)
    b = np.random.choice(dist, 1)
    c = np.random.choice(dist, 2)
    arr=np.arange(1,m)
    for k in random.sample(list(arr),len(arr)):
        alpha = alpha*0.85
        if (c*x[k]>b).all:
            x[k] = x[k-1] - alpha * np.sum(c)
        else:
            x[k] = x[k-1]
        y.append(np.max([0,c[0]*x[k]-b])+np.max([0,c[1]*x[k]-b]))
    plt.plot(y,'o',label=alpha)
    plt.legend(loc='upper right')
```



The Example 1 cases seem to perform better than example 2 cases when diminishing step sizes are used. Constant step sizes seems to not work well at all! The SGD seems to perform better when cyclic is used vs. randomized. Stochastic GD doesn't work at all (maybe it is implemented wrong?) if constant step sizes are used, but workjs well if diminishing is used.