Gradient Descent 의 단점 해결 방안들에 대해 알아보겠습니다.

오늘 다룰 내용은 SGD, SGD with momentum, AdaGrad, RMSprop, Adam입니다.

In [41]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
```

우선 함수는 아래와 같이 주어집니다.

In [42]:
```
def f(x):
    return x[0]**2/20 + x[1]**2
```

위 함수의 최소값과 최소값을 만드는 입력값은 무엇입니까?

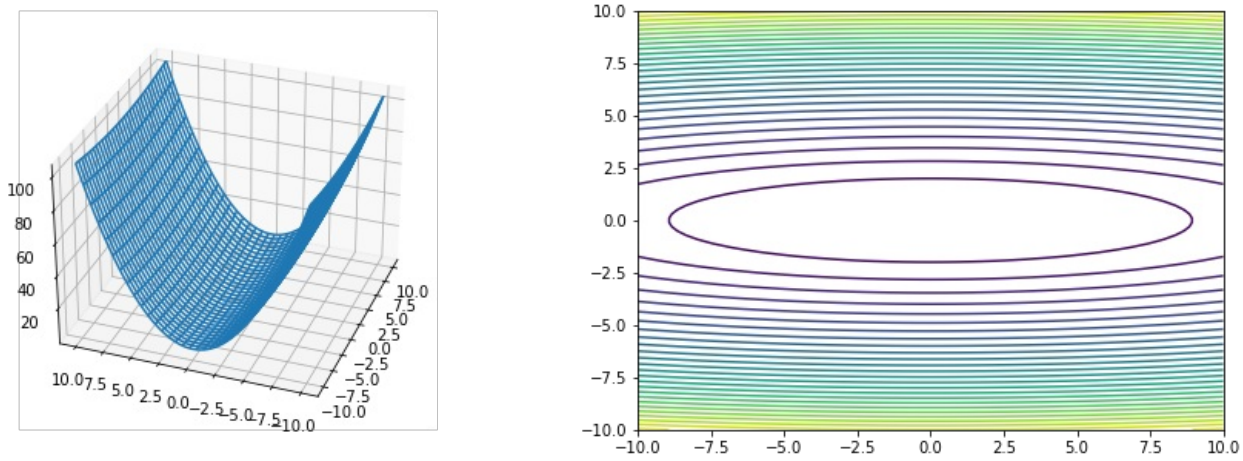
정답) 최소값 : 0, 최소값을 만드는 입력값 : [0,0]

위 함수의 3D wireframe과 2D Contour를 그리는 함수를 완성합니다.

In [43]:
```
def plot_points_on_contour(f,points=None):
    fig = plt.figure(figsize=(15,5))

    ax1 = fig.add_subplot(121, projection='3d')
    ax2 = fig.add_subplot(122)

    x = [np.linspace(-10,10,100), np.linspace(-10,10,100)]

    X = np.meshgrid(x[0],x[1])
    Z = f(X)


    # Plot a basic wireframe.
    ax1.plot_wireframe(X[0],X[1],Z,rstride=3, cstride=3)
    ax1.view_init(30,200)

    ax2.contour(X[0],X[1],Z,30)

    if points is not None:
        ax2.plot(points[0],points[1],marker='o')

    plt.show()
```

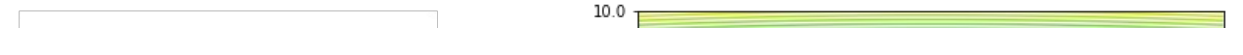
함수 f만 입력시 wireframe과 contour 그림이 출력됩니다.

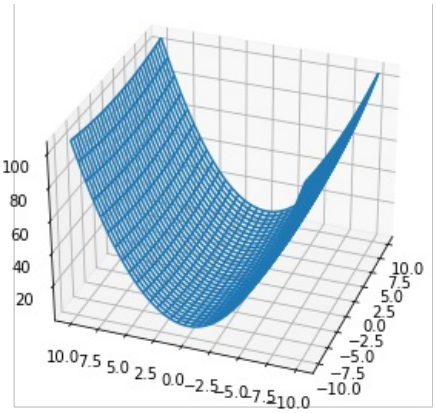
In [44]:
```
plot_points_on_contour(f)
```

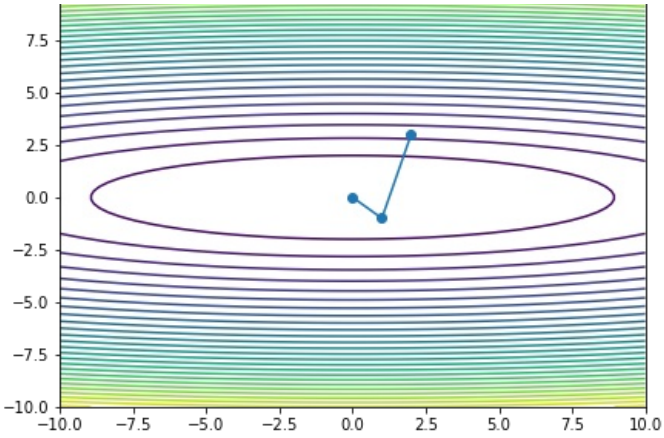


함수와 각 점의 x,y축을 입력하면 그 부분이 contour 그림에 표시됩니다.

In [45]:
```
plot_points_on_contour(f, [[0,1,2],[0,-1,3]])
```







각각의 방식으로 주어진 함수를 최소로 만드는 x,y값을 찾아봅시다.

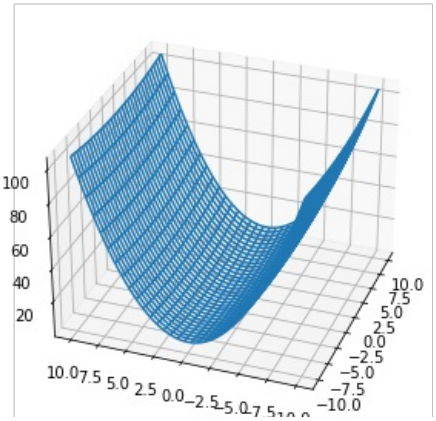
즉 위에서 계산한 입력값 x,y를 각 방법으로 찾을 수 있어야 합니다.

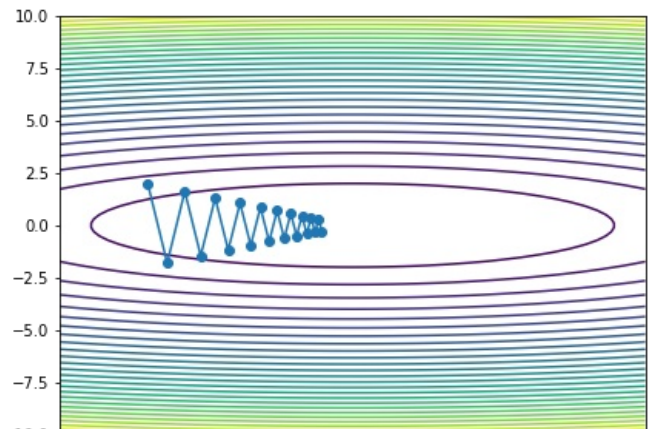
아래 함수 GD는 함수, torch.Tensor 객체, learning_rate, num_step 를 입력으로 받아 3D wireframe과 2D contour에 x값이 어떻게 변화는지를 표시하여줍니다.

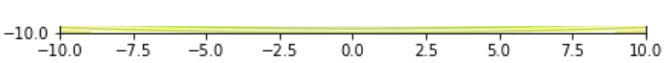
In [46]:
```
# t = torch.tensor([-7,2], requires_grad = True, dtype=torch.float32)

# t_loss = f(t)
# t_loss.backward()
# delta_loss = t.grad
# t = t - 0.95*delta_loss
# print(t)
```

In [47]:
```
def GD(f,init_x, learning_rate, num_step):

    x_rlt = []
    y_rlt = []

    for i in range(num_step):

        x_rlt.append(init_x[0].item())
        y_rlt.append(init_x[1].item())

        # loss 함수를 정의하고 미분하시오
        loss = f(init_x)
        loss.backward()
        delta_loss = init_x.grad

        with torch.no_grad():
            # 채우시오
            init_x = init_x - learning_rate * delta_loss


        init_x.requires_grad = True

    plot_points_on_contour(f, [x_rlt,y_rlt])
```

In [48]:
```
x = torch.tensor([-7,2], requires_grad = True, dtype=torch.float32)

GD(f,x,.95, 20)
```







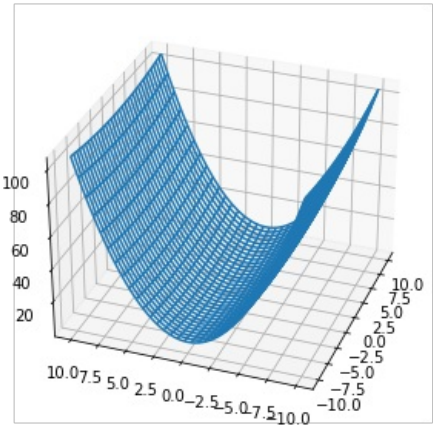
위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

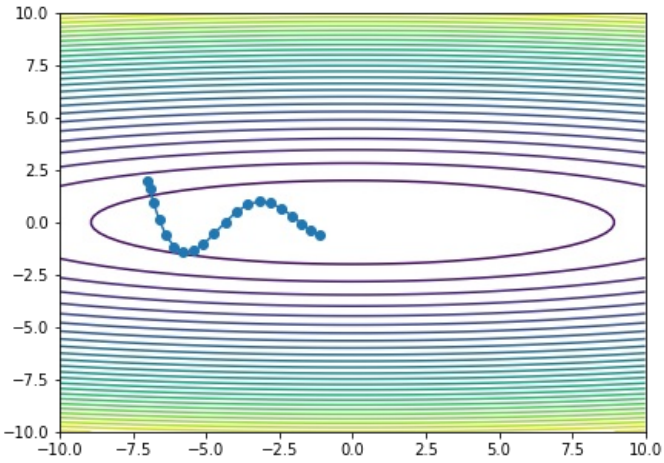
정답) learning_rate를 낮게 잡을 수록 변화하는 크기(보폭)이 줄어든다. 따라서 lr 값이 작을 수록 여러방향으로 튀는 경향(잡음)은 적지만 최종목표 (0,0)에는 근접하지 않는다. 0.95로 했을 때 위아래로 값이 튀면서 수렴하지만 그래도 lr이 낮은 거에 비해서 최종목표값이 가깝게 수렴한다.

아래 함수 GD_momentum는 함수, torch.Tensor 객체, learning_rate, num_step, momentum 계수를 입력으로 받아 3D wireframe과 2D contour에 x값이 어떻게 변화는지를 표시하여줍니다.

In [49]:
```
def GD_momentum(f, init_x, learning_rate, num_step, mom):

    v = 0
    x_rlt = []
    y_rlt = []

    for i in range(num_step):

        x_rlt.append(init_x[0].item())
        y_rlt.append(init_x[1].item())

        # loss 함수를 정의하고 미분하시오
        loss = f(init_x)
        loss.backward()
        delta_loss = init_x.grad


        with torch.no_grad():

            # 채우시오
            v = v*mom - learning_rate*delta_loss
            init_x = init_x + v

        init_x.requires_grad = True

    plot_points_on_contour(f, [x_rlt,y_rlt])
```

In [50]:
```
x = torch.tensor([-7,2], requires_grad = True, dtype=torch.float32)

GD_momentum(f,x,.1, 20, 0.9)
```





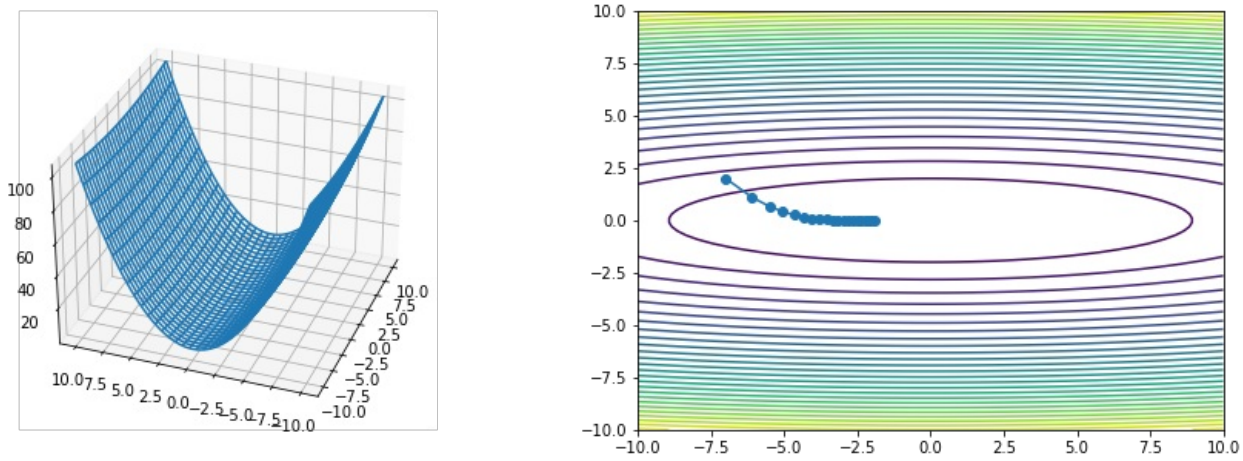
위 결과에 learning_rate와 초기값, 모멘텀 계수을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

정답) learning_rate 값이 커지면 보폭이 커지므로 최솟값(0,0)에 수렴하지 않고 발산하는 형태를 나타낸다. 반면 너무 작을 경우 보폭이 너무 작아지므로 최솟값으로 향하는 방향성은 맞지만 최종 결과값이 0과 먼 곳에서 멈추게 된다. 모멘텀 계수는 learning_rate와 반비례하는 결과를 나타낸다. 값이 작아지면 발산하고 값이 커지면 0과 먼 곳에서 멈추게 된다. 이는 v의 결과값이 v*mom-lr*delta_loss 인 것을 보면 lr과 mom이 반비례 관계임을 예측할 수 있다.

In [51]:
```
def GD_AdaGrad(f, init_x, learning_rate, num_step):

    h = 0
    x_rlt = []
    y_rlt = []
```

```
    for i in range(num_step):

        x_rlt.append(init_x[0].item())
        y_rlt.append(init_x[1].item())

        # loss 함수를 정의하고 미분하시오
        loss = f(init_x)
        loss.backward()
        delta_loss = init_x.grad


        with torch.no_grad():
            h = h + delta_loss * delta_loss
            init_x = init_x - learning_rate/(h**0.5) * delta_loss

        init_x.requires_grad = True

    plot_points_on_contour(f, [x_rlt,y_rlt])
```

In [52]:
```
x = torch.tensor([-7,2], requires_grad = True, dtype=torch.float32)

GD_AdaGrad(f,x,.9, 20)
```

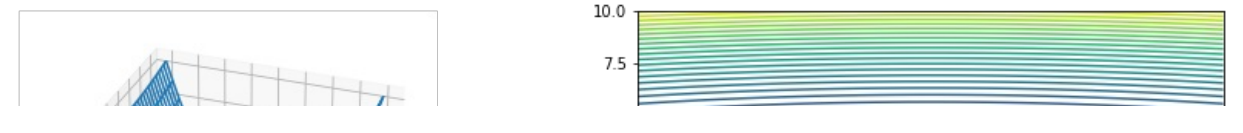


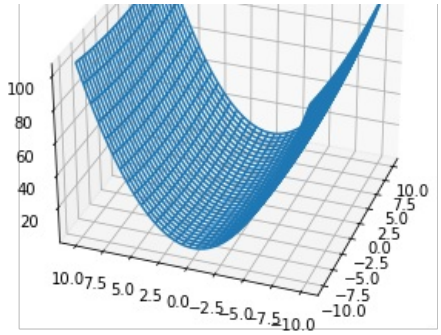
위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

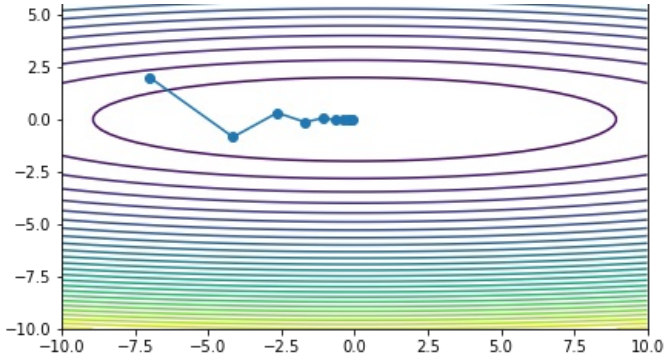
정답)learning_rate를 낮게 잡을 수록 변화하는 크기(보폭)이 줄어든다. 따라서 lr 값이 작을 수록 여러방향으로 튀는 경향(잡음)은 적지만 최종목표 (0,0)에는 근접하지 않는다. 0.9로 했을 때 발산하지 않고 그래도 lr이 낮은 거에 비해서 최종목표값이 가깝게 수렴한다.

In [53]:
```
def GD_RMSprop(f,init_x, learning_rate, num_step, gamma):

    g = 0
    x_rlt = []
    y_rlt = []

    for i in range(num_step):

        x_rlt.append(init_x[0].item())
        y_rlt.append(init_x[1].item())

        # loss 함수를 정의하고 미분하시오
        loss = f(init_x)
        loss.backward()
        delta_loss = init_x.grad

        with torch.no_grad():
            g = gamma*g + (1-gamma)*delta_loss*delta_loss
            init_x = init_x - learning_rate/(g**0.5) * delta_loss

        init_x.requires_grad = True

    plot_points_on_contour(f, [x_rlt,y_rlt])
```

In [54]:
```
x = torch.tensor([-7,2], requires_grad = True, dtype=torch.float32)

GD_RMSprop(f,x,.9, 10, 0.9)
```





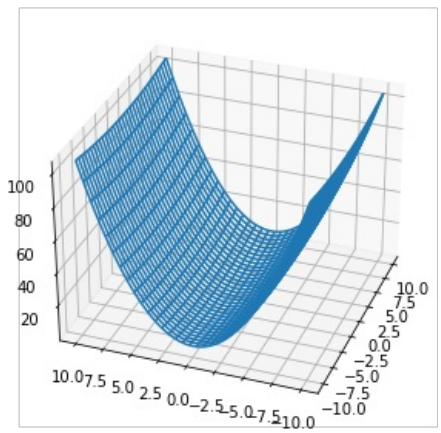


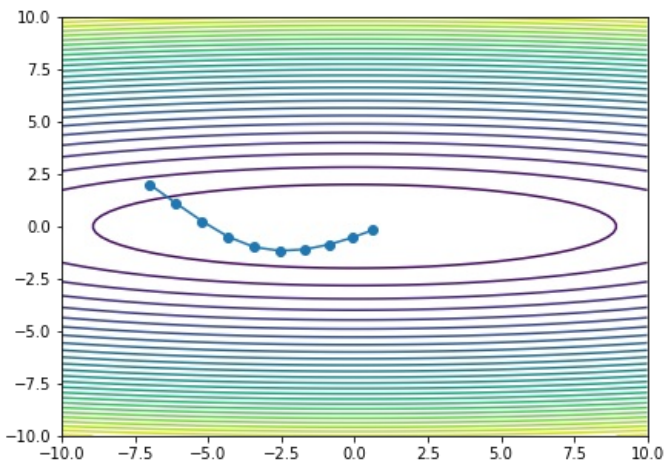
위 결과에 learning_rate와 초기값, gamma 을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

정답)learning_rate는 마찬가지로 크면 발산, 작으면 수렴하게 된다. 해당 모델에서는 0.5부터 0.9까지 범위가 최적이라고 판단된다. 초기값은 어디서 시작하던지 -10~10 내에서 설정하게 되면 최솟값을 찾는데 큰 문제가 되지 않는다. RMSprop의 특성상 앞의 값과 다음값의 가중평균을 구하여 그 다음 위치를 예측하는 것이다. 따라서 gamma를 너무 작게 설정하면 이전 값들을 사용하여 다음 값을 제대로 예측할 수 없게된다. 결과적으로 최종값에 다와서도 0에 수렴하지않고 계속 발산하는 모습을 보여준다. 또한 1이상이되면 계산값이 변화가 없어지기 때문에 반드시 1 미만의 값을 설정해줘야 한다. 해당 모델에서는 수렴하기 위해서 0.9가 최적값으로 판단된다.

In [55]:
```
def GD_Adam(f,init_x, learning_rate, num_step, beta_1, beta_2):

    m = 0
    v = 0
    t = 1
    x_rlt = []
    y_rlt = []

    for i in range(num_step):

        x_rlt.append(init_x[0].item())
        y_rlt.append(init_x[1].item())

        # loss 함수를 정의하고 미분하시오
        loss = f(init_x)
        loss.backward()
        delta_loss = init_x.grad


        with torch.no_grad():

            # 채우시오
            m = beta_1*m + (1-beta_1)*delta_loss
            v = beta_2*v + (1-beta_2)*delta_loss*delta_loss
            m_hat = m/(1-beta_1**t)
            v_hat = v/(1-beta_2**t)
            t += 1

            init_x = init_x - m_hat*learning_rate/((v_hat+1e-7)**0.5)

        init_x.requires_grad = True

    plot_points_on_contour(f, [x_rlt,y_rlt])
```

In [56]:
```
x = torch.tensor([-7,2], requires_grad = True, dtype=torch.float32)

GD_Adam(f,x,0.9, 10, 0.9, 0.9)
```





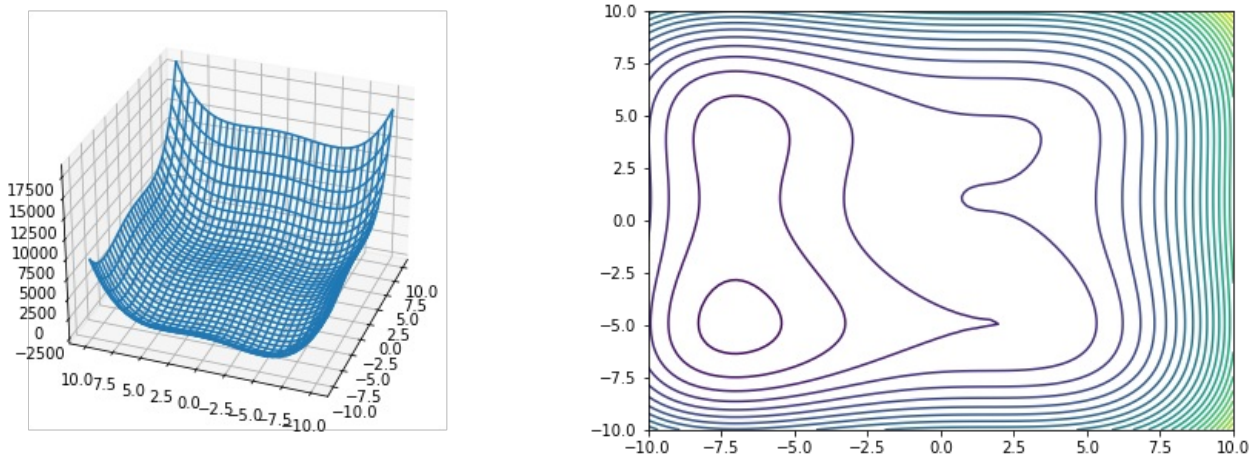
위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

정답)learning_rate는 마찬가지로 크면 발산, 작으면 수렴하게 된다. 해당 모델에서는 0.7정도가 최적이라고 판단된다. 초기값은 어디서 시작하던지 -10~10 내에서 설정하게 되면 최솟값을 찾는데 큰 문제가 되지 않는다.

새로운 함수에 대해 위 작업을 반복해보자

In [57]:
```
def g(x):
    return x[0]**4 + x[1]**4 + 5*x[0]**3  - 40*(x[0]-1)**2 - 40*(x[1]-1)**2
```

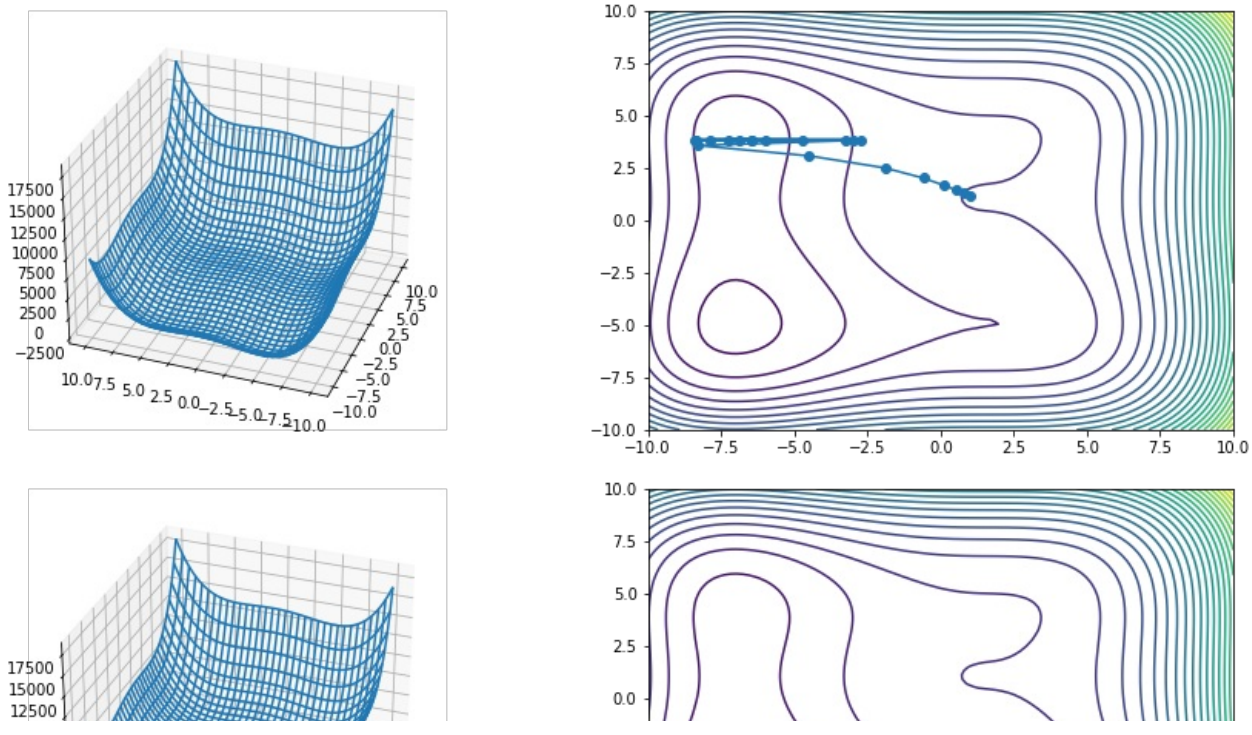
In [58]:
```
plot_points_on_contour(g)
```

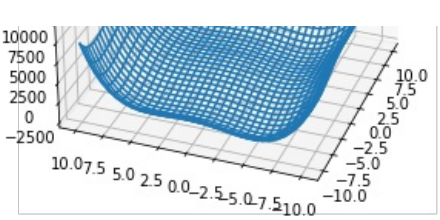
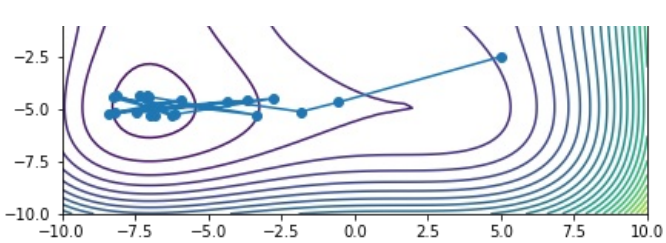
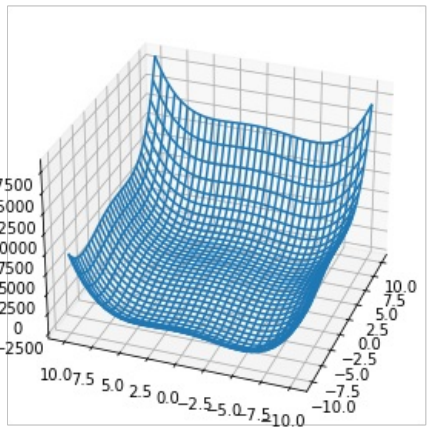
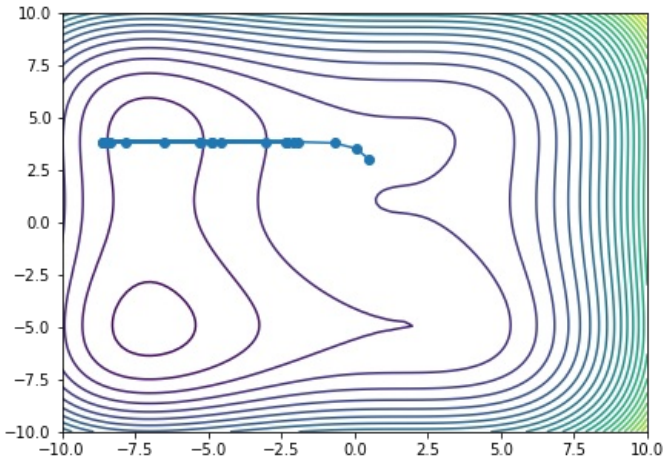


위 함수의 최소값를 갖는 입력값은 대략 어디인가?

정답)[-7,-5]

In [59]:
```
x = torch.tensor([1,1.2], requires_grad = True, dtype=torch.float32)
GD(g,x,.01, 20)
x = torch.tensor([5,-2.5], requires_grad = True, dtype=torch.float32)
GD(g,x,.01, 20)
x = torch.tensor([0.5,3], requires_grad = True, dtype=torch.float32)
GD(g,x,.01, 20)
```



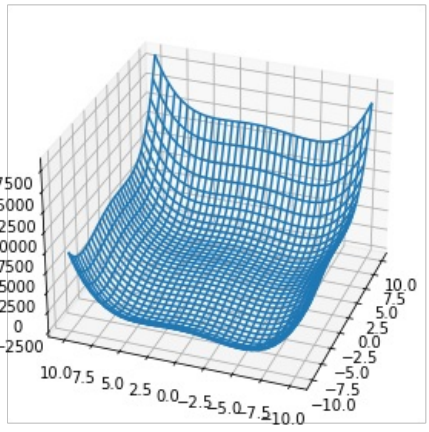
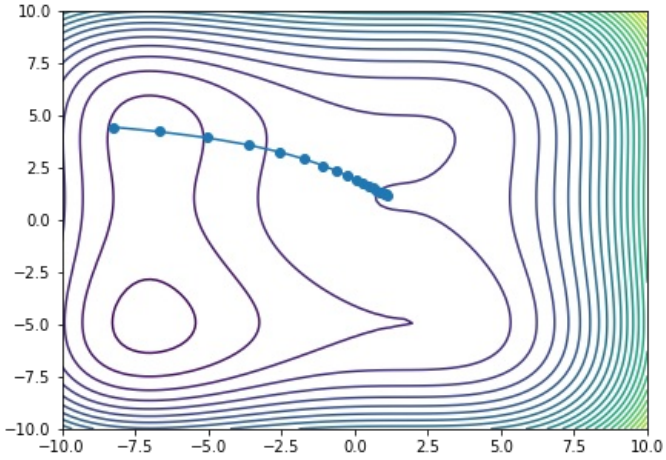
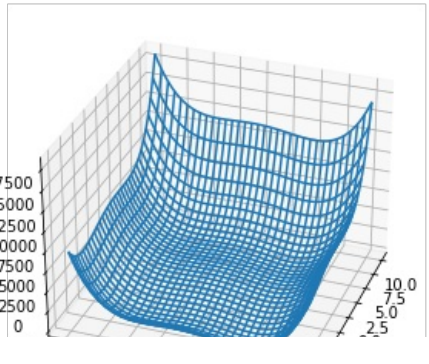
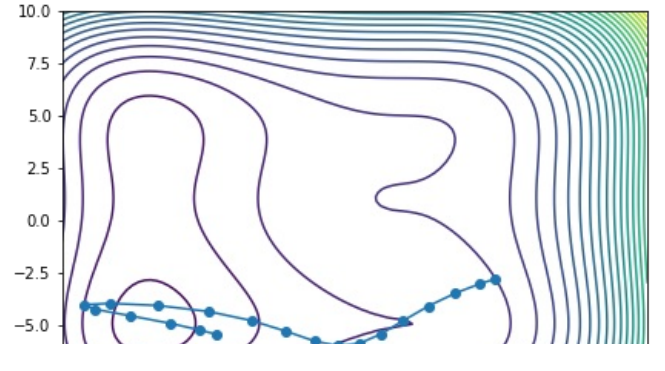





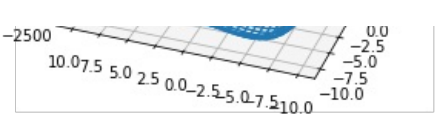
위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

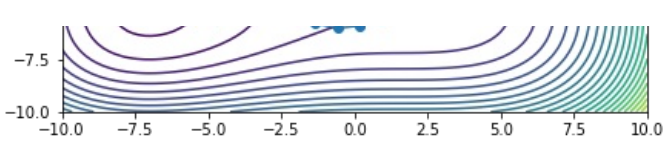
정답)lr은 앞의 모델에서와 같이 작으면 수렴, 크면 발산한다. 하지만 g함수가 f함수보다 차원이 높고 기울기가 훨씬 가파르므로 lr이 조금만 커져도 아주 크게 발산해버린다. 따라서 앞의 모델(f)의 경우에는 대체적으로 lr을 0.1~0.9의 값으로 설정했다면 g에서는 0.01~0.1 정도의 값에서 조정해서 -10부터 10사이의 값으로 수렴한다. 또한 변화율이 커서인지 초기값에 따라서 수렴하는 모양이 많이 달라짐을 확인하였다. 특정 위치에서는 최솟값에 근접하게 수렴하지만 조금만 위치가 달라져도 다른 곳으로 수렴함을 확인할 수 있었다.

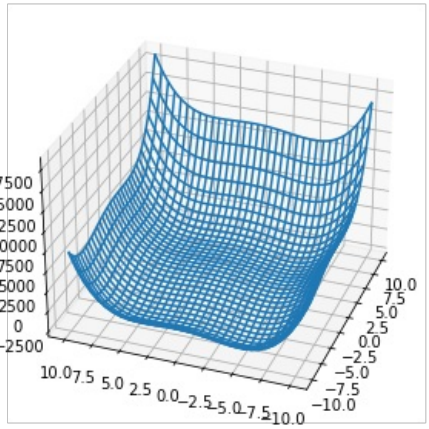
In [60]:

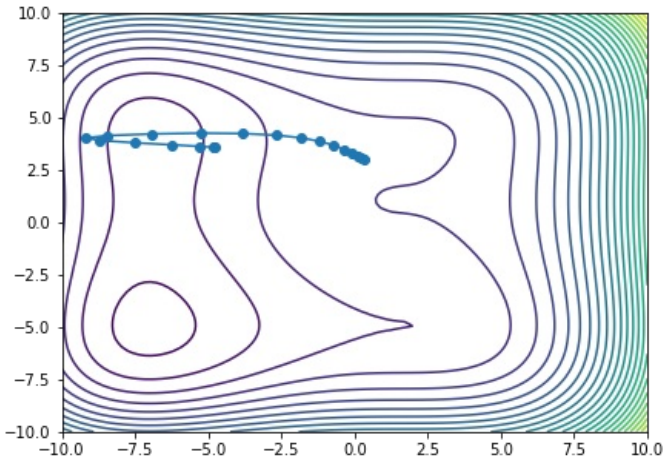
```
x = torch.tensor([1.1,1.2], requires_grad = True, dtype=torch.float32)

GD_momentum(g,x,.0011, 20, 0.9)

x = torch.tensor([4.8,-2.8], requires_grad = True, dtype=torch.float32)

GD_momentum(g,x,.0011, 20, 0.9)

x = torch.tensor([0.3,3], requires_grad = True, dtype=torch.float32)

GD_momentum(g,x,.0011, 20, 0.9)
```



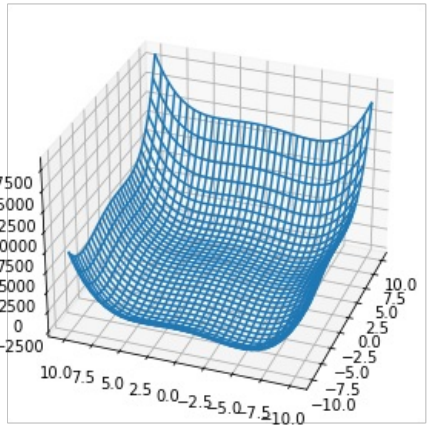


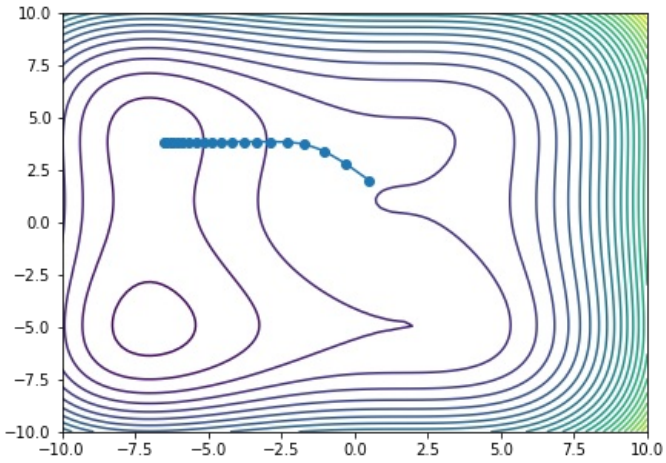


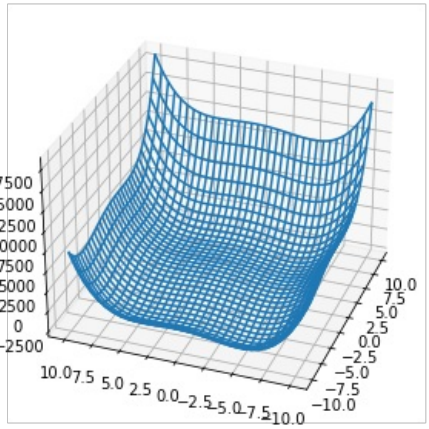


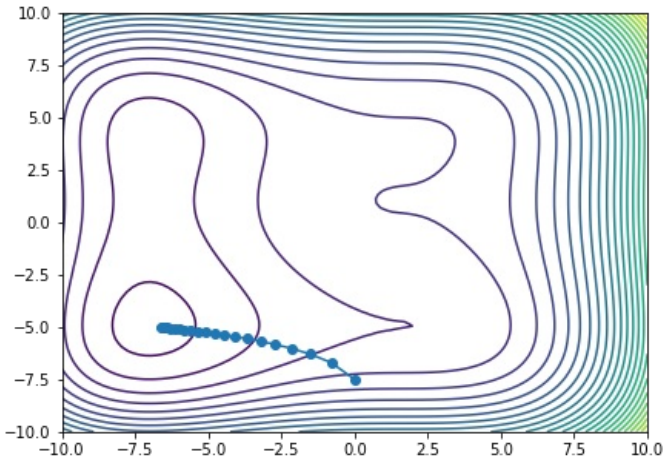
In [61]:

```
x = torch.tensor([0.5,2], requires_grad = True, dtype=torch.float32)

GD_AdaGrad(g,x,.8, 20)

x = torch.tensor([0,-7.5], requires_grad = True, dtype=torch.float32)

GD_AdaGrad(g,x,.8, 20)
```









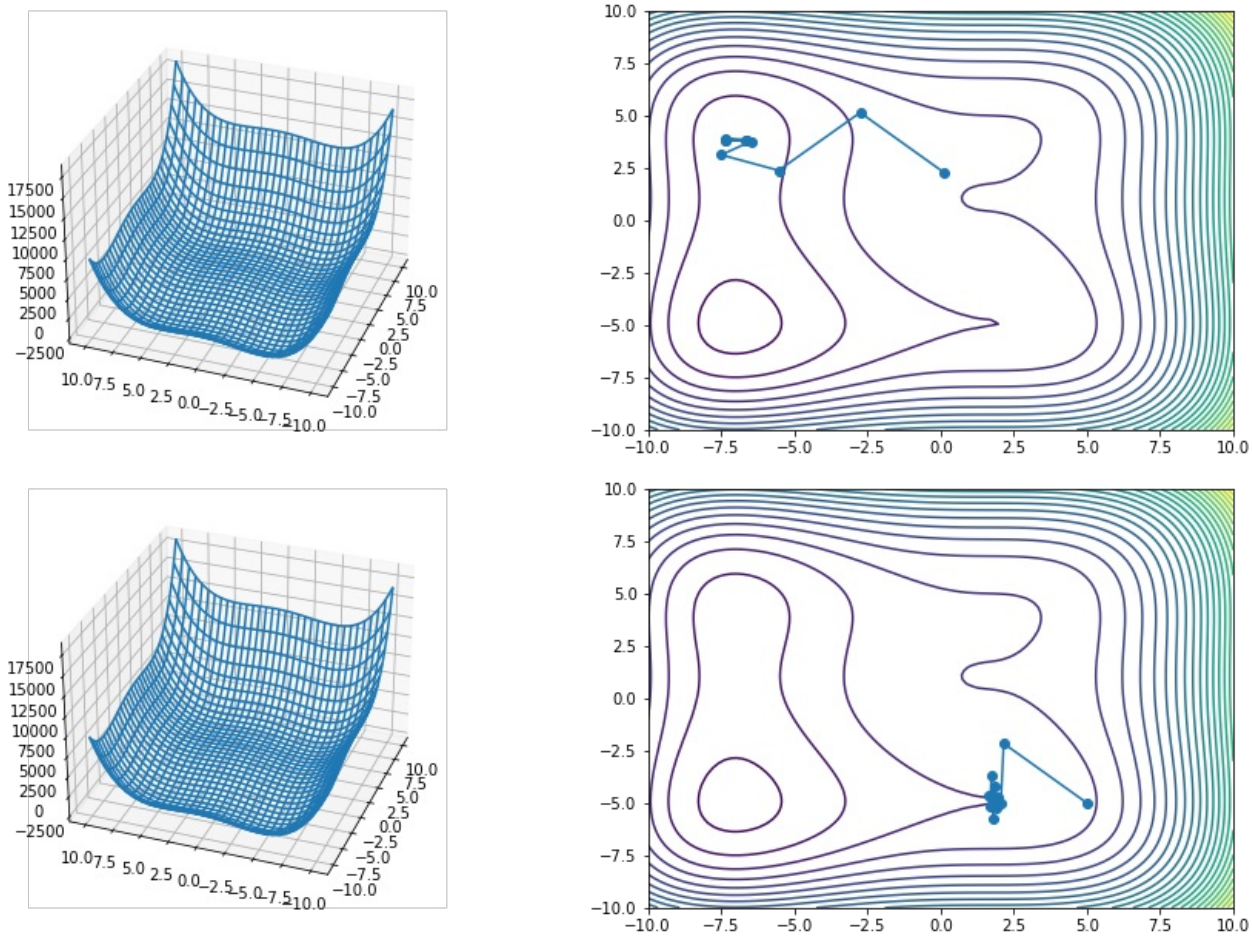
위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

정답) 일반 GD와 모멘텀에 비해서 lr 값을 높여줘야(0.5에서 0.9사이)수렴이 잘되는 것을 확인할 수 있었다. 그리고 위의 예시에서 볼 수 있듯이 같은 lr 값을 주었음에도 초기값에 따라서 0에 수렴하는 정도가 확연히 차이남을 확인할 수 있다.

In [62]:

```
x = torch.tensor([0.1,2.3], requires_grad = True, dtype=torch.float32)
```

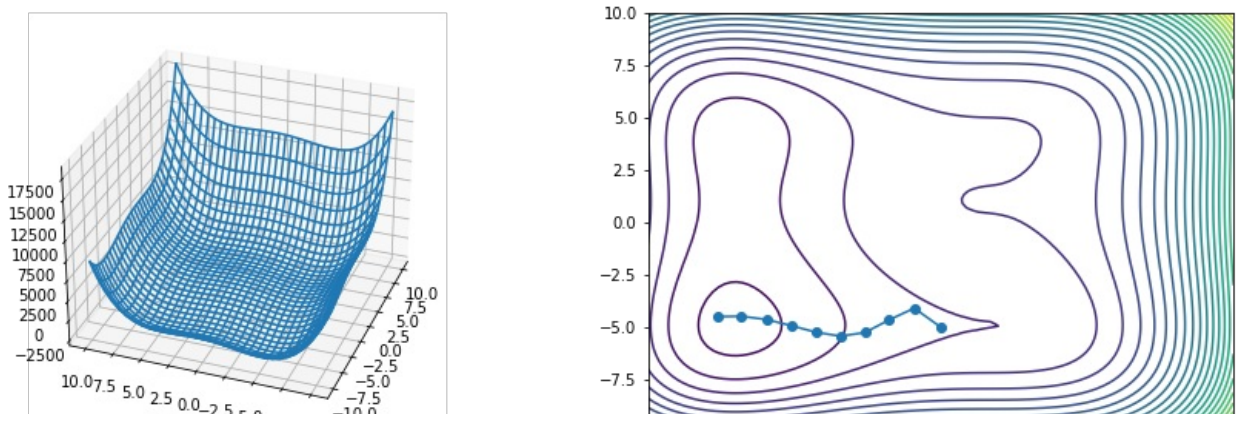
```
GD_RMSprop(g,x,.9, 10, 0.9)

x = torch.tensor([5,-5], requires_grad = True, dtype=torch.float32)
GD_RMSprop(g,x,.9, 10, 0.9)
```

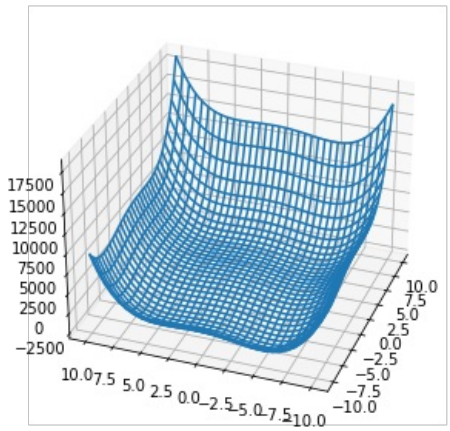


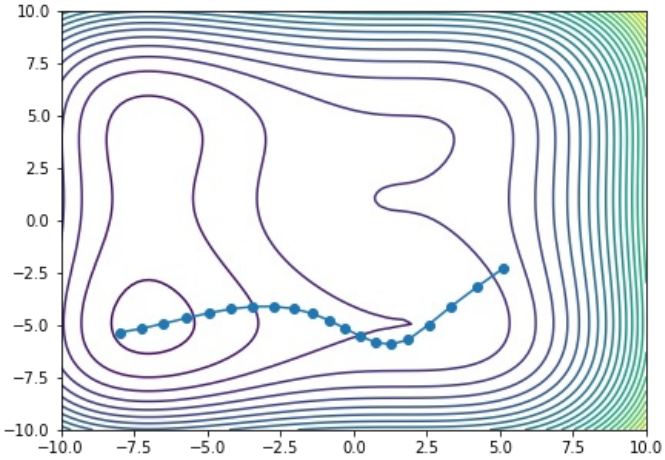
위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

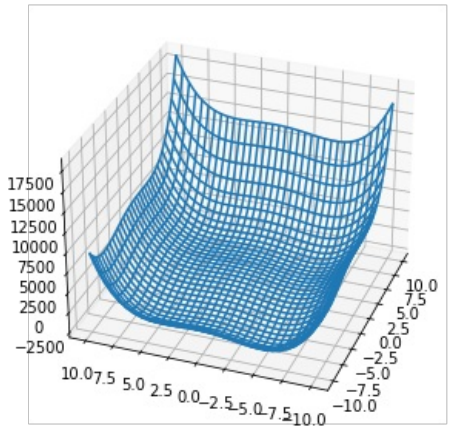
정답) 해당 모델에서도 lr이 크면 발산, 작으면 수렴하지만 대체적으로 최솟값에 제대로 수렴하지 않는다. 초기값에 따라서 수렴하는 위치와 방향, 그리고 정도가 확연하게 차이남을 확인하였다.

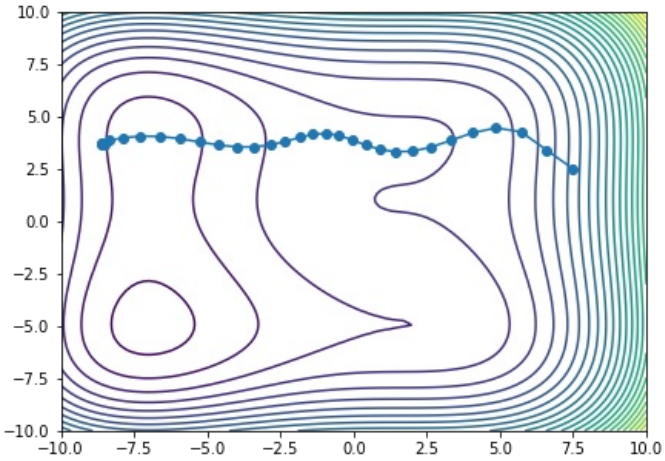
In [63]:
```
x = torch.tensor([0,-5], requires_grad = True, dtype=torch.float32)
GD_Adam(g,x,0.9, 10, 0.9, 0.9)

x = torch.tensor([5.1,-2.3], requires_grad = True, dtype=torch.float32)
GD_Adam(g,x,0.9, 20, 0.9, 0.9)

x = torch.tensor([7.5,2.5], requires_grad = True, dtype=torch.float32)
GD_Adam(g,x,0.9, 30, 0.9, 0.9)
```











위 결과에 learning_rate와 초기값을 체계적으로 변경하여 보고 각각 무슨 영향을 미치는지 서술하세요.

정답) 해당 모델에서는 lr을 높여도 진행속도(보폭)가 상대적으로 작았기 때문에 반복횟수를 30으로 늘려서 최솟값 근처에 수렴할 수 있도록 하였다. 또한 위의 예시와 같이 0근처에서 초기값을 설정하며 학습하면 최솟값 근처로 수렴하지만 그렇지 않을 경우 최종 수렴 위치가 최솟값과 떨어져 있음을 확인할 수 있다.

In [ ]: