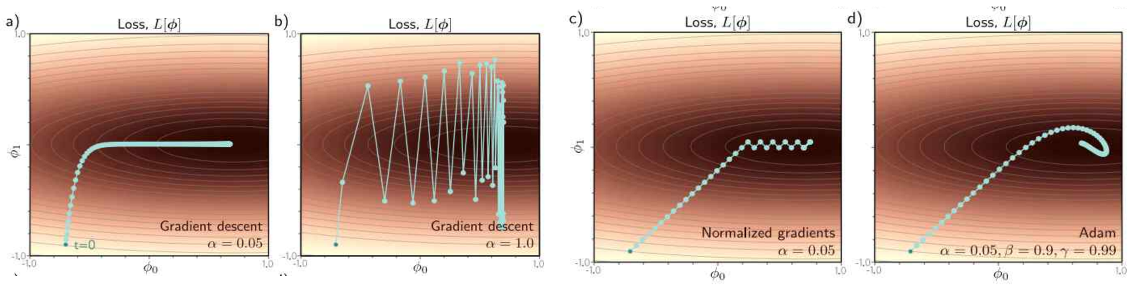
Notebook 6.5 Adam

Figure 6.9

* Adam 알고리즘 구현



1. 손실함수 정의

- loss 함수를 통해 주어진 phi0, phi1값을 사용하여 height 계산 후 손실 값 반환

$$\text{height} = \exp\left(-0.5 \cdot (\phi_1^2) \cdot 4.0\right) \cdot \exp\left(-0.5 \cdot (\phi_0 - 0.7)^2 / 4.0\right)$$

```
def loss(phi0, phi1):
    # phi1과 phi0를 사용하여 height 값을 계산
    height = np.exp(-0.5* (phi1 * phi1)*4.0)
    height = height * np. exp(-0.5* (phi0-0.7) *(phi0-0.7)/4.0)
    return 1.0-height
```

2. 손실함수의 기울기 계산

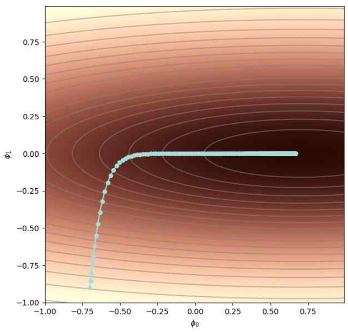
```
# 손실 함수의 기울기(gradient) 계산
def get_loss_gradient(phi0, phi1):
    delta_phi = 0.00001;
    gradient = np.zeros((2,1));
    gradient[0] = (loss(phi0+delta_phi/2.0, phi1) - loss(phi0-delta_phi/2.0, phi1))/delta_phi
    gradient[1] = (loss(phi0, phi1+delta_phi/2.0) - loss(phi0, phi1-delta_phi/2.0))/delta_phi
    # 1차원 배열로 변환하여 반환
    return gradient[:,0];
```

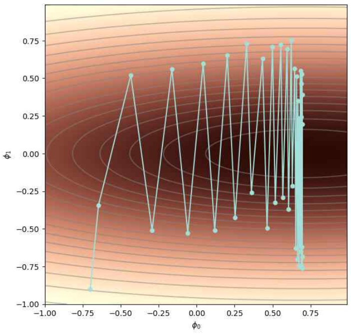
```
# 단순 경사 하강법 함수 정의
def grad_descent(start_posn, n_steps, alpha):
    grad_path = np.zeros((2, n_steps+1));
    grad_path[:,0] = start_posn[:,0];
    for c_step in range(n_steps):
      this_grad = get_loss_gradient(grad_path[0,c_step], grad_path[1,c_step]);
      grad_path[:,c_step+1] = grad_path[:,c_step] - alpha * this_grad
    return grad_path;
```

• 고정 step size를 사용하여 경사 하강법 실행

```
loss_function, phi0mesh, phi1mesh = get_loss_function_for_plot() ;
start_posn = np.zeros((2,1));
start_posn[0,0] = -0.7; start_posn[1,0] = -0.9
```

```
# 경사 하강법 실행 (첫 번째 실행: alpha=0.08, n_steps=200)
grad_path1 = grad_descent(start_posn, n_steps=200, alpha = 0.08)
draw_function(phi0mesh, phi1mesh, loss_function, my_colormap, grad_path1)
# 경사 하강법 실행 (두 번째 실행: alpha=1.0, n_steps=40)
grad_path2 = grad_descent(start_posn, n_steps=40, alpha= 1.0)
draw_function(phi0mesh, phi1mesh, loss_function, my_colormap, grad_path2)
```

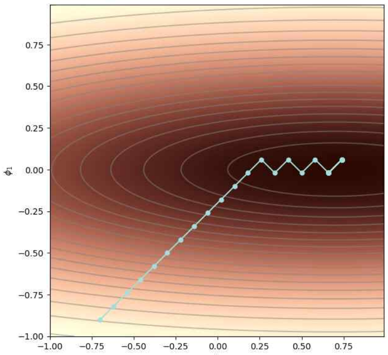





```
def normalized_gradients(start_posn, n_steps, alpha, epsilon=1e-20):
grad_path = np.zeros((2, n_steps + 1))
grad_path[:, 0] = start_posn[:, 0]
    for c_step in range(n_steps):
        # 현재 위치에서 손실 함수의 기울기 계산(식 6.13 첫 번째)
         m = get_loss_gradient(grad_path[0, c_step], grad_path[1, c_step])
        # 기울기의 제곱 계산(식 6.13 두 번째)
         v = m ** 2  # Square each component of the gradient vector
        # 기울기를 정규화하여 업데이트 (식 6.14)
         grad_path[:, c_step + 1] = grad_path[:, c_step] - alpha * m / (np.sqrt(v) + epsilon)
    return grad_path
```

• 정규화된 경사 하강법

```
# normalized gradients
start_posn = np.zeros((2,1));
start_posn[0,0] = -0.7; start_posn[1,0] = -0.9
# gradient descent
grad_path1 = normalized_gradients(start_posn, n_steps=40, alpha = 0.08)
draw_function(phi0mesh, phi1mesh, loss_function, my_colormap, grad_path1)
```

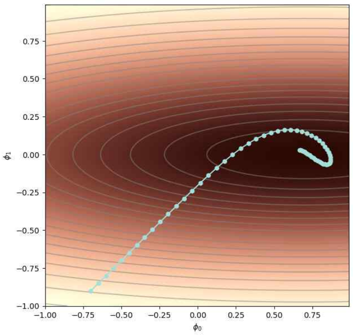



```
def adam(start_posn, n_steps, alpha, beta=0.9, gamma=0.99, epsilon=1e-20):
grad_path = np.zeros((2, n_steps + 1))
grad_path[:, 0] = start_posn[:, 0]
m = np.zeros_like(grad_path[:, 0])
```

```python
v = np.zeros_like(grad_path[:, 0])
    for c_step in range(n_steps):
        # Measure the gradient
        grad = get_loss_gradient(grad_path[0, c_step], grad_path[1, c_step])
        # 모멘텀 기반 기울기 추정 업데이트 (식 6.15 첫 번째)
        m = beta * m + (1- beta) * grad
        # # 모멘텀 기반 제곱 기울기 추정 업데이트 (식 6.15 두 번째)
        v = gamma * v + (1- gamma) * grad ** 2
        # 편향 보정 적용 (식 6.16)
        m_tilde = m / (1- beta ** (c_step + 1))
        v_tilde = v / (1- gamma ** (c_step + 1))
        # 업데이트 규칙 적용 (식 6.17)
        grad_path[:, c_step + 1] = grad_path[:, c_step] - alpha * m_tilde / (np.sqrt(v_tilde) + epsilon)
    return grad_path
```

```python
# Adam algorithm
start_posn = np.zeros((2,1));

# Adam 알고리즘을 사용하여 경사 하강법 실행 (60 스텝, 학습률 0.05)
start_posn[0,0] = -0.7; start_posn[1,0] = -0.9
grad_path1 = adam(start_posn, n_steps=60, alpha = 0.05)
draw_function(phi0mesh, phi1mesh, loss_function, my_colormap, grad_path1)
```



[참고. 시각화 코드 구현]

```
# 손실함수의 시각화를 위한 함수
def get_loss_function_for_plot():
    grid_values = np.arange(-1.0,1.0,0.01);
    phi0mesh, phi1mesh = np.meshgrid(grid_values, grid_values)
    loss_function = np.zeros((grid_values.size, grid_values.size))
    for idphi0, phi0 in enumerate(grid_values):
      for idphi1, phi1 in enumerate(grid_values):
        loss_function[idphi0, idphi1] = loss(phi1,phi0)
  return loss_function, phi0mesh, phi1mesh
```

```
my_colormap_vals_hex =('2a0902', '2b0a03', '2c0b04', '2d0c05', '2e0c06', '2f0d07', '300d08', '310e09', '320f0a',
                                        ⋮
'fff0d1', 'fff2d2', 'fff3d3', 'fff4d5', 'fff6d6', 'fff7d8', 'fff8d9', 'fffada', 'fffbdc', 'fffcdd', 'fffedf', 'ffffe0')
my_colormap_vals_dec = np.array([int(element,base=16) forelement inmy_colormap_vals_hex])
r = np.floor(my_colormap_vals_dec/(256*256))
g = np.floor((my_colormap_vals_dec - r *256*256)/256)
b = np.floor(my_colormap_vals_dec - r * 256*256- g * 256)
my_colormap_vals = np.vstack((r,g,b)).transpose()/255.0
my_colormap = ListedColormap(my_colormap_vals)
def draw_function(phi0mesh, phi1mesh, loss_function, my_colormap, opt_path):
   fig = plt.figure();
   ax = plt.axes();
   fig.set_size_inches(7,7)
   ax.contourf(phi0mesh, phi1mesh, loss_function, 256, cmap=my_colormap);
   ax.contour(phi0mesh, phi1mesh, loss_function, 20, colors=['#80808080'])
   ax.plot(opt_path[0,:], opt_path[1,:],'-', color='#a0d9d3ff')
   ax.plot(opt_path[0,:], opt_path[1,:],'.', color='#a0d9d3ff',markersize=10)
   ax.set_xlabel(r"$\phi_{0}$")
   ax.set_ylabel(r"$\phi_{1}$")
   plt.show()
```