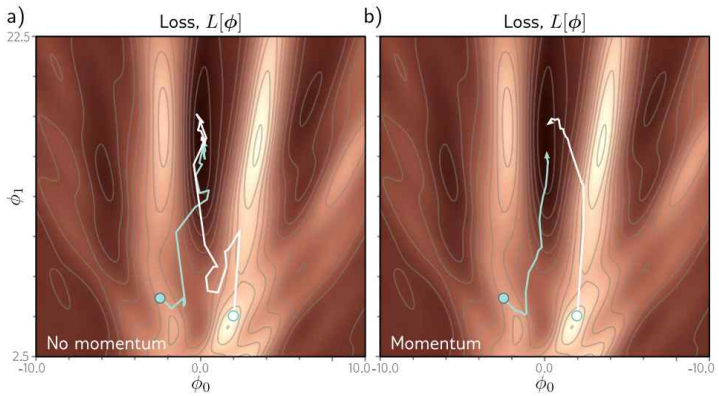
Notebook 6.4 Momentum

Figure 6.7

* momentum 알고리즘 구현



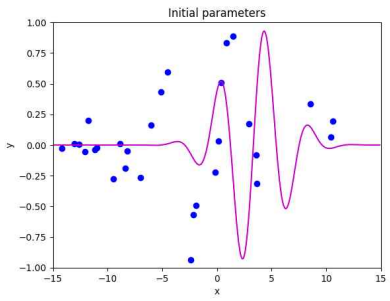
1. 모델 정의

$$f[x,\phi] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{32.0}\right)$$

```
# model 정의(Gabor 모델 적합)
def model(phi,x):
  sin_component = np.sin(phi[0] + 0.06* phi[1] * x)
  gauss_component = np.exp(-(phi[0] + 0.06* phi[1] * x) * (phi[0] + 0.06* phi[1] * x) / 32)
  y_pred = sin_component * gauss_component
 return y_pred
```

2. parameters 초기화 및 model 시각화

```
phi = np.zeros((2,1))
phi[0] = -5     # Horizontal offset
phi[1] = 25     # Frequency
draw_model(data,model,phi, "Initial parameters")
```



3. 손실 제곱합 계산

```
def compute_loss(data_x, data_y, model, phi):
  pred_y = model(phi, data_x)
  loss = np.sum((pred_y-data_y)*(pred_y-data_y))
 return loss
```

4. 주어진 parameters에 대해 gradient vector를 계산

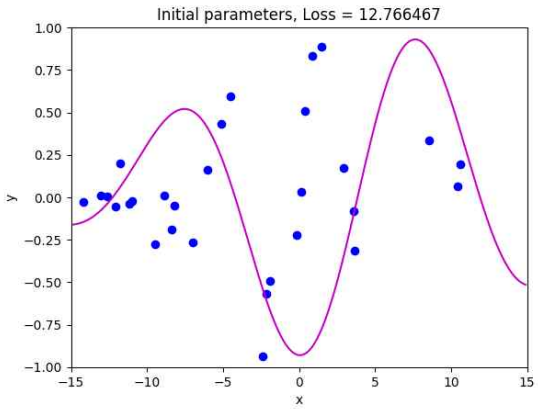
```
def gabor_deriv_phi0(data_x,data_y,phi0, phi1):
   0.06* phi1 * data_x + phi0
   y = data_y
cos_component = np.cos(x)
   sin_component = np.sin(x)
```

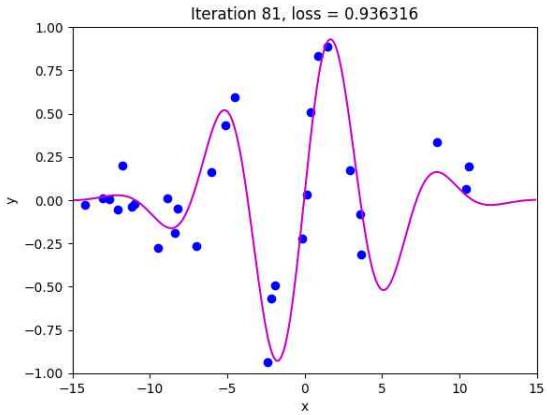
$$\frac{\partial L}{\partial \phi} = \begin{bmatrix} \frac{\partial L}{\partial \phi_0} \\ \frac{\partial L}{\partial \phi_1} \end{bmatrix}$$

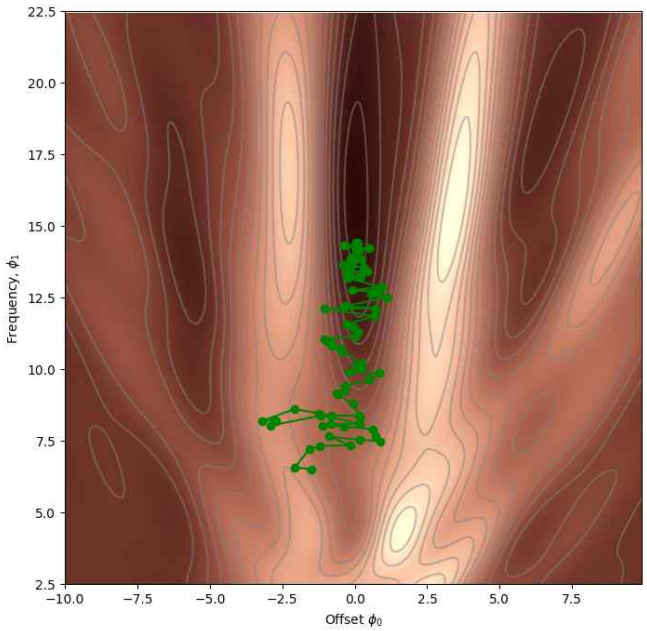
```
    gauss_component = np.exp(-0.5* x *x / 16)
    deriv = cos_component * gauss_component - sin_component * gauss_component * x / 16
    deriv = 2* deriv * (sin_component * gauss_component - y)
    return np.sum(deriv)

def gabor_deriv_phi1(data_x, data_y,phi0, phi1):
    0.06* phi1 * data_x + phi0
    y = data_y
    cos_component = np.cos(x)
    sin_component = np.sin(x)
    gauss_component = np.exp(-0.5* x *x / 16)
    deriv = 0.06* data_x * cos_component * gauss_component - 0.06* data_x*sin_component *
            gauss_component * x / 16
    deriv = 2*deriv * (sin_component * gauss_component - y)
    return np.sum(deriv)

def compute_gradient(data_x, data_y, phi):
     dl_dphi0 = gabor_deriv_phi0(data_x, data_y, phi[0],phi[1])
     dl_dphi1 = gabor_deriv_phi1(data_x, data_y, phi[0],phi[1])
    # Return the gradient
    return np.array([[dl_dphi0],[dl_dphi1]])
```

* 표준 확률적 경사 하강법

```
np.random.seed(1)
n_steps = 81
batch_size = 5
alpha = 0.6
phi_all = np.zeros((2,n_steps+1))
phi_all[0,0] = -1.5
phi_all[1,0] = 6.5
loss = compute_loss(data[0,:], data[1,:], model, phi_all[:,0:1])
draw_model(data,model,phi_all[:,0:1], "Initial parameters, Loss = %f"%(loss))

for c_step in range (n_steps):
batch_index = np.random.permutation(data.shape[1])[0:batch_size]
gradient = compute_gradient(data[0,batch_index], data[1,batch_index], phi_all[:,c_step:c_step+1] )
phi_all[:,c_step+1:c_step+2] = phi_all[:,c_step:c_step+1] - alpha * gradient
loss = compute_loss(data[0,:], data[1,:], model, phi_all[:,c_step+1:c_step+2])
draw_model(data,model,phi_all[:,c_step+1], "Iteration %d, loss = %f"%(c_step+1,loss))
draw_loss_function(compute_loss, data, model,phi_all)
```

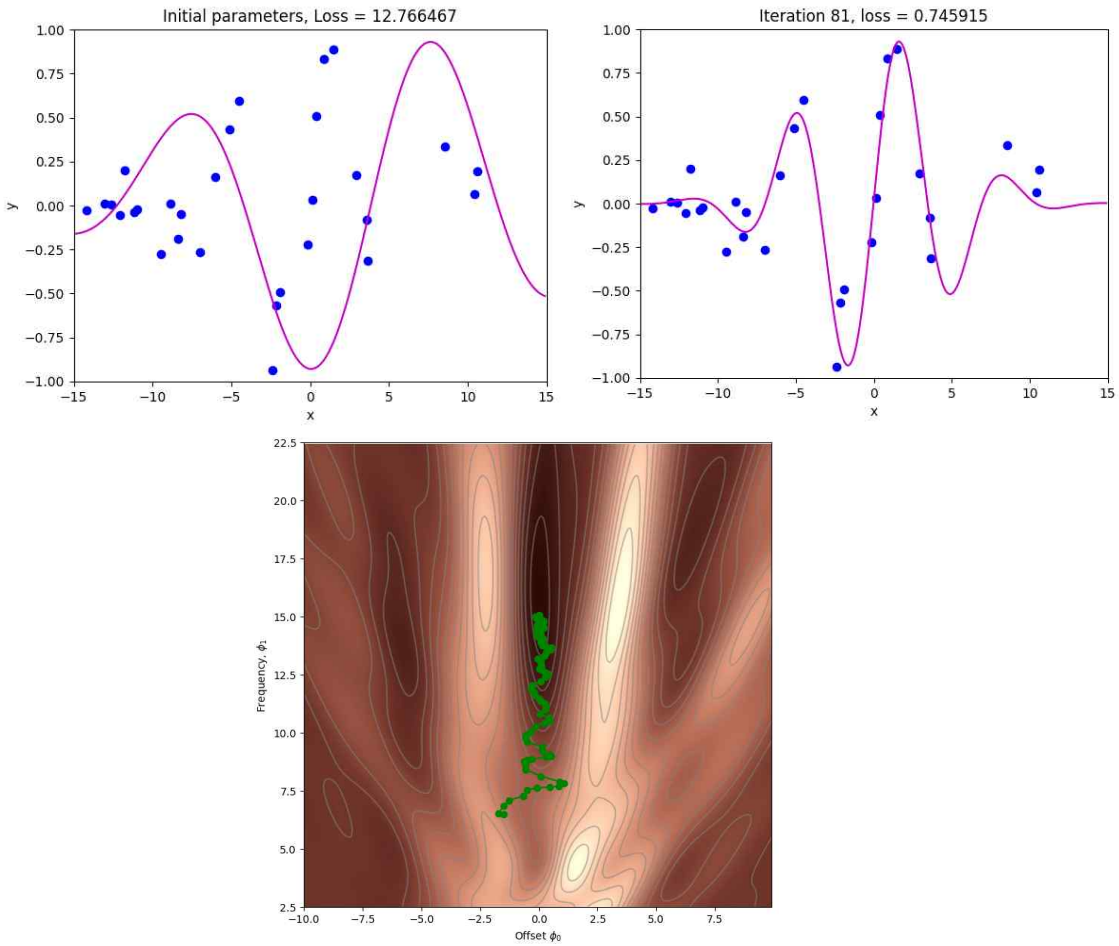






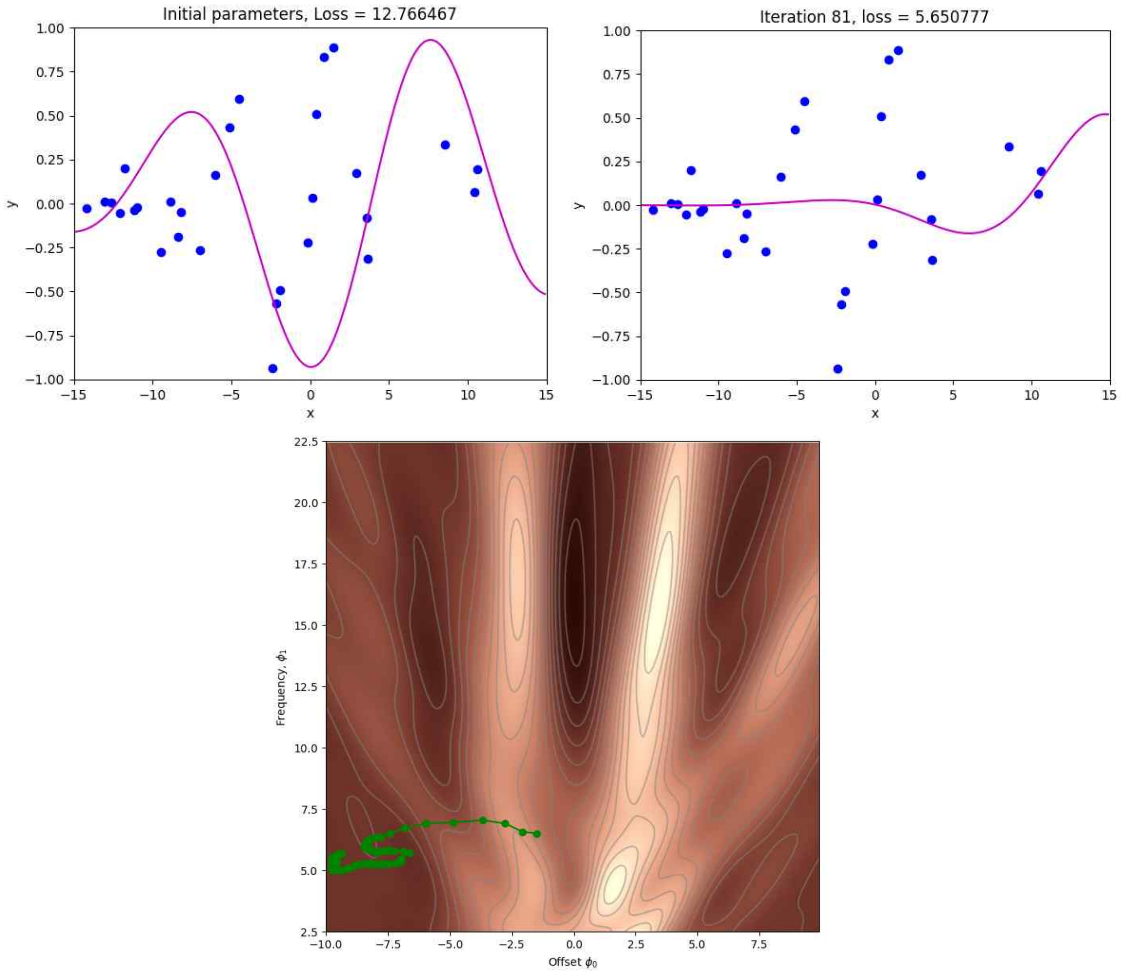
* Momentum 추가

```
np.random.seed(1)
n_steps = 81
batch_size = 5
alpha = 0.6
beta = 0.6
momentum = np.zeros([2,1])
phi_all = np.zeros((2,n_steps+1))
phi_all[0,0] = -1.5
phi_all[1,0] = 6.5
loss = compute_loss(data[0,:], data[1,:], model, phi_all[:,0:1])
draw_model(data,model,phi_all[:,0:1], "Initial parameters, Loss = %f"%(loss))
forc_step inrange (n_steps):
batch_index = np.random.permutation(data.shape[1])[0:batch_size]
gradient = compute_gradient(data[0,batch_index], data[1,batch_index], phi_all[:,c_step:c_step+1])
 # # 모멘텀을 계산
momentum = beta * momentum + (1- beta) * gradient
 # 파라미터 업데이트
phi_all[:,c_step+1:c_step+2] = phi_all[:,c_step:c_step+1] - alpha * momentum
loss = compute_loss(data[0,:], data[1,:], model, phi_all[:,c_step+1:c_step+2])
draw_model(data,model,phi_all[:,c_step+1], "Iteration %d, loss = %f"%(c_step+1,loss))
draw_loss_function(compute_loss, data, model,phi_all)
```



* Nesterov Momentum 시도

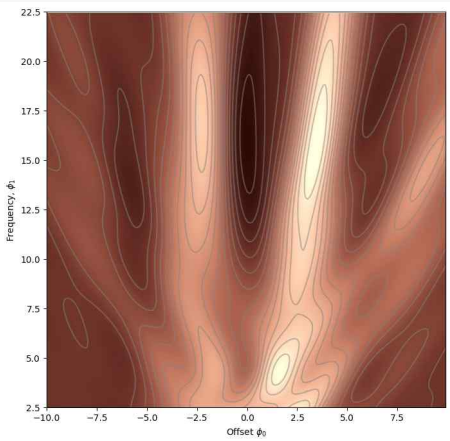
```
np.random.seed(1)
n_steps = 81
batch_size = 5
alpha = 0.6
beta = 0.6
momentum = np.zeros([2, 1])
phi_all = np.zeros((2, n_steps + 1))
phi_all[0, 0] = -1.5
phi_all[1, 0] = 6.5
loss = compute_loss(data[0, :], data[1, :], model, phi_all[:, 0:1])
draw_model(data, model, phi_all[:, 0:1], "Initial parameters, Loss = %f"% (loss))
for c_step in range(n_steps):
batch_index = np.random.permutation(data.shape[1])[0:batch_size]
    # Nesterov momentum : 현재 위치에서 momentum 사용하여 미리 나아간 위치에서 기울기 계산
gradient = compute_gradient(data[0, batch_index], data[1, batch_index], phi_all[:, c_step:c_step + 1] - alpha
* beta * momentum)
momentum = beta * momentum + gradient
phi_all[:, c_step + 1:c_step + 2] = phi_all[:, c_step:c_step + 1] - alpha * momentum
loss = compute_loss(data[0, :], data[1, :], model, phi_all[:, c_step + 1:c_step + 2])
draw_model(data, model, phi_all[:, c_step + 1], "Iteration %d, loss = %f"% (c_step + 1, loss))
draw_loss_function(compute_loss, data, model, phi_all)
```



[참고. 시각화 코드 구현]

```
# 30개의 {x_i, y_i} 쌍으로 구성된 학습 데이터 생성
data = np.array([[-1.920e+00,-1.422e+01,1.490e+00,-1.940e+00,-2.389e+00,-5.090e+00,
                                                    ⋮
                  -1.119e+01,2.902e+00,-8.220e+00,-1.179e+01,-8.391e+00,-4.505e+00],
                 [-1.051e+00,-2.482e-02,8.896e-01,-4.943e-01,-9.371e-01,4.306e-01,
                                                    ⋮
                  -3.666e-02,1.709e-01,-4.805e-02,2.008e-01,-1.904e-01,5.952e-01]])
```

```
# model 시각화
def draw_model(data,model,phi,title=None):
x_model = np.arange(-15,15,0.1)
y_model = model(phi,x_model)
fix, ax = plt.subplots()
ax.plot(data[0,:],data[1,:],'bo')
ax.plot(x_model,y_model,'m-')
ax.set_xlim([-15,15]);ax.set_ylim([-1,1])
ax.set_xlabel('x'); ax.set_ylabel('y')
  iftitle is not None:
ax.set_title(title)
plt.show()
```




```
def draw_loss_function(compute_loss, data,  model, phi_itersNone):
my_colormap_vals_hex =('2a0902', '2b0a03', '2c0b04', '2d0c05', '2e0c06', '2f0d07',
                                                    ⋮
 '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 = ListedColormap(np.vstack((r,g,b)).transpose()/255.0)
offsets_mesh, freqs_mesh = np.meshgrid(np.arange(-10,10.0,0.1), np.arange(2.5,22.5,0.1))
loss_mesh = np.zeros_like(freqs_mesh)
  for idslope, slope innp.ndenumerate(freqs_mesh):
loss_mesh[idslope] = compute_loss(data[0,:], data[1,:], model, np.array([[offsets_mesh[idslope]], [slope]]))
fig,ax = plt.subplots()
fig.set_size_inches(8,8)
ax.contourf(offsets_mesh,freqs_mesh,loss_mesh,256,cmap=my_colormap)
ax.contour(offsets_mesh,freqs_mesh,loss_mesh,20,colors=['#80808080'])
  if phi_iters is not None:
ax.plot(phi_iters[0,:], phi_iters[1,:],'go-')
ax.set_ylim([2.5,22.5])
ax.set_xlabel('Offset $\phi_{0}$'); ax.set_ylabel('Frequency, $\phi_{1}$')
plt.show()

draw_loss_function(compute_loss, data, model)
```