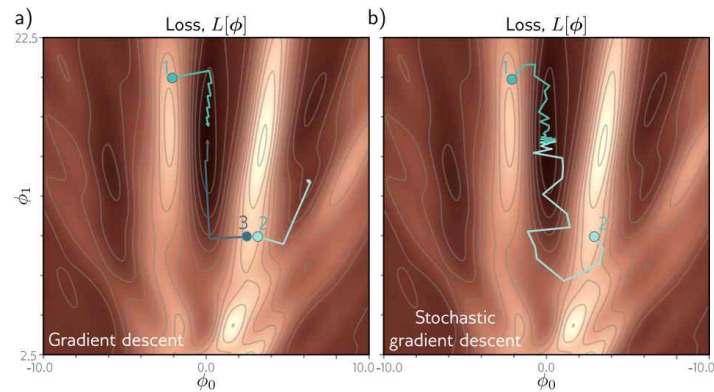


Notebook 6.3 Stochastic_Gradient_Descent

Figure 6.5

* 경사하강법 및 확률적 경사하강법 구현



1. 데이터 및 Gabor model 정의

30쌍의 $\{x_i, y_i\}$ 훈련 데이터 Gabor model에 적합시키기

```
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]])
```

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

모델 정의

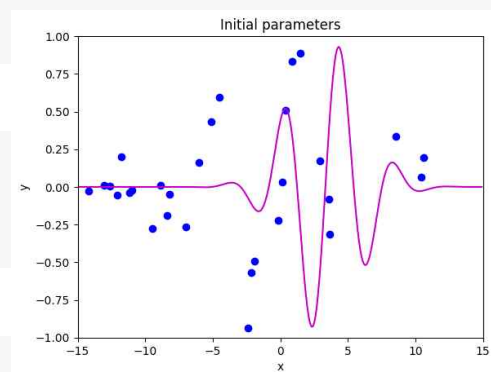
```
def model(phi,x):
    sin_component = np.sin(phi[0] + 0.06* phi[1] * x)
    gauss_component = np.exp(-(phi[0] + 0.06* phi[1] * x) * (phi[0] + 0.06* phi[1] * x) / 32)
    y_pred= sin_component * gauss_component
    return y_pred
```

- 파라미터 초기 설정

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

- 훈련 데이터에 대한 제곱합 손실 계산

```
def compute_loss(data_x, data_y, model, phi):
    # 모델 예측값 계산
    y_pred = model(phi, data_x)
    # 제곱 오차 계산
    squared_diff = (y_pred - data_y) ** 2
    # 제곱 오차의 합 계산
    loss = np.sum(squared_diff)
```



```

    return loss

# 제곱합 손실 함수의 표현식 및 phi0과 phi1의 미분
def gabor_deriv_phi0(data_x,data_y,phi0, phi1):
    x = 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 = 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):
    x = 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])
    # gradient 반환
    return np.array([[dl_dphi0],[dl_dphi1]])

```

- 그래디언트 하강법 수행

```

def loss_function_1D(dist_prop, data, model, phi_start, gradient):
    # 거리 이동 후 손실 반환
    return compute_loss(data[0:], data[1:], model, phi_start+ gradient * dist_prop)
def line_search(data, model, phi, gradient, thresh=.00001, max_dist0.1, max_iter15):
    a = 0
    b = 0.33* max_dist
    c = 0.66* max_dist
    d = 1.0* max_dist
    n_iter = 0
    while np.abs(b-c) > thresh and n_iter < max_iter:
        n_iter = n_iter+1
        lossa = loss_function_1D(a, data, model, phi,gradient)
        lossb = loss_function_1D(b, data, model, phi,gradient)
        loss_c = loss_function_1D(c, data, model, phi,gradient)
        lossd = loss_function_1D(d, data, model, phi,gradient)

        # Rule #1 If point A is less than points B, C, and D then halve points B,C, and D
        if np.argmin((lossa,lossb,loss_c,lossd))==0:
            b = b/2
            c = c/2

```

```

d = d/2
    continue;
    # Rule #2 If point b is less than point c then
    #           point d becomes point c, and
    #           point b becomes 1/3 between a and new d
    #           point c becomes 2/3 between a and new d
    if lossb < lossd:
d = c
b = a+ (d-a)/3
c = a+ 2*(d-a)/3
    continue
    # Rule #2 If point c is less than point b then
    #           point a becomes point b, and
    #           point b becomes 1/3 between new a and d
    #           point c becomes 2/3 between new a and d
a = b
b = a+ (d-a)/3
c = a+ 2*(d-a)/3
    # Return average of two middle points
    return(b+c)/2.0

```

- 동적 학습률

line_search(alpha)를 이용하여 학습률을 동적으로 설정

```

def gradient_descent_step(phi, data, model):
    # gradient 계산
    gradient = compute_gradient(data[0:],data[1:], phi)
    # parameters 업데이트 - 음의 방향으로 탐색
    alpha = line_search(data, model, phi, gradient*-1, max_dist = 2.0)
    phi = phi - alpha * gradient
    return phi

```

```

n_steps = 21
phi_all = np.zeros((2,n_steps+1))
phi_all[0,0] = -1.5
phi_all[1,0] = 8.5
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))
# gradient 하강법 스텝 수행
for c_step in range (n_steps):
    phi_all[:,c_step+1:c_step+2] = gradient_descent_step(phi_all[:,c_step:c_step+1],data, model)
    # 매 5번째 스텝마다 손실 측정 및 모델 시각화
    if c_step % 5== 0:
        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)

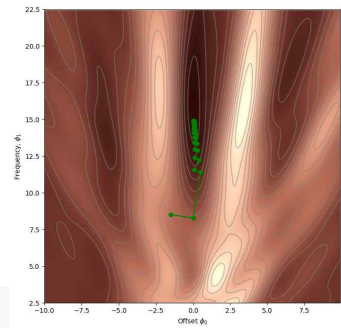
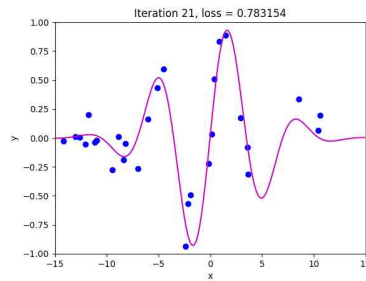
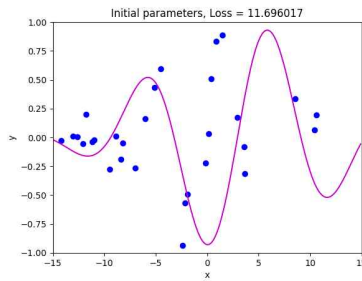
```

- 고정된 학습률

```

def gradient_descent_step_fixed_learning_rate(phi, data, alpha):

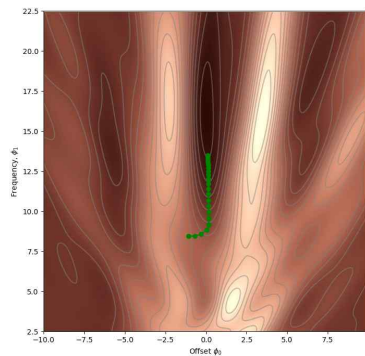
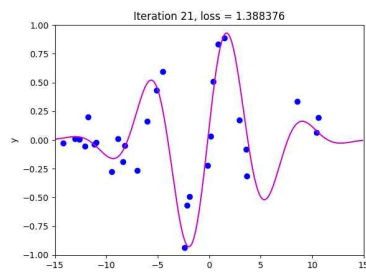
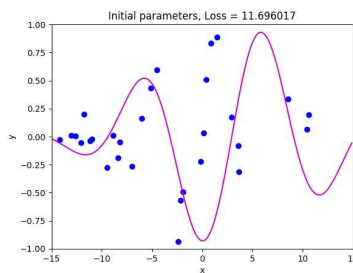
```



```
# 기울기 계산
grad = compute_gradient(data[0, :], data[1, :], phi)
# 파라미터 업데이트
phi = phi - alpha * grad
return phi
```

```
n_steps = 21
phi_all = np.zeros((2,n_steps+1))
phi_all[0,0] = -1.5
phi_all[1,0] = 8.5
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):
    phi_all[:,c_step+1:c_step+2] = gradient_descent_step_fixed_learning_rate(phi_all[:,c_step:c_step+1],data, alpha
=0.2)
    if c_step % 5== 0:
        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]

```

# model시각화
def draw_model(data,model,phi,title=None):
    x_model = np.arange(-15,15,0.1)
    y_model = model(phi,x_model)
    fig, 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')
    if title is not None:
        ax.set_title(title)
    plt.show()

def draw_loss_function(compute_loss, data, model, phi_iters=None):
    # my_colormap_vals_hex : 색상 맵을 정의하는데 사용
    my_colormap_vals_hex = ('2a0902', '2b0a03', '2c0b04', '2d0c05', '2e0c06', '2f0d07', '300e08', '310f09', '32100a', '33110b', '34120c', '35130d', '36140e', '37150f', '381610', '391711', '3a1812', '3b1913', '3c1a14', '3d1b15', '3e1c16', '3f1d17', '401e18', '411f19', '42201a', '43211b', '44221c', '45231d', '46241e', '47251f', '482620', '492721', '4a2822', '4b2923', '4c2a24', '4d2b25', '4e2c26', '4f2d27', '502e28', '512f29', '52302a', '53312b', '54322c', '55332d', '56342e', '57352f', '583630', '593731', '5a3832', '5b3933', '5c3a34', '5d3b35', '5e3c36', '5f3d37', '603e38', '613f39', '62403a', '63413b', '64423c', '65433d', '66443e', '67453f', '684640', '694741', '6a4842', '6b4943', '6c4a44', '6d4b45', '6e4c46', '6f4d47', '704e48', '714f49', '72504a', '73514b', '74524c', '75534d', '76544e', '77554f', '785650', '795751', '7a5852', '7b5953', '7c5a54', '7d5b55', '7e5c56', '7f5d57', '805e58', '815f59', '82605a', '83615b', '84625c', '85635d', '86645e', '87655f', '886660', '896761', '8a6862', '8b6963', '8c6a64', '8d6b65', '8e6c66', '8f6d67', '906e68', '916f69', '92706a', '93716b', '94726c', '95736d', '96746e', '97756f', '987660', '997761', '9a7862', '9b7963', '9c7a64', '9d7b65', '9e7c66', '9f7d67', 'a07e68', 'a17f69', 'a2806a', 'a3816b', 'a4826c', 'a5836d', 'a6846e', 'a7856f', 'a88660', 'a98761', 'aa8862', 'ab8963', 'ac8a64', 'ad8b65', 'ae8c66', 'af8d67', 'b08e68', 'b18f69', 'b2906a', 'b3916b', 'b4926c', 'b5936d', 'b6946e', 'b7956f', 'b89660', 'b99761', 'ba9862', 'bb9963', 'bc9a64', 'bd9b65', 'be9c66', 'bf9d67', 'c09e68', 'c19f69', 'c2a06a', 'c3a16b', 'c4a26c', 'c5a36d', 'c6a46e', 'c7a56f', 'c8a660', 'c9a761', 'caa862', 'caba63', 'cbab64', 'ccac65', 'cdad66', 'ceae67', 'cfaf68', 'd0b069', 'd1b16a', 'd2b26b', 'd3b36c', 'd4b46d', 'd5b56e', 'd6b66f', 'd7b760', 'd8b861', 'd9b962', 'dab063', 'dcb164', 'ddb265', 'deb366', 'dfb467', 'e0b568', 'e1b669', 'e2b76a', 'e3b86b', 'e4b96c', 'e5ba6d', 'e6bb6e', 'e7bc6f', 'e8bd60', 'e9be61', 'eabf62', 'ecbf63', 'ecbf64', 'ecbf65', 'ecbf66', 'ecbf67', 'ecbf68', 'ecbf69', 'ecbf6a', 'ecbf6b', 'ecbf6c', 'ecbf6d', 'ecbf6e', 'ecbf6f', 'ecbf70', 'ecbf71', 'ecbf72', 'ecbf73', 'ecbf74', 'ecbf75', 'ecbf76', 'ecbf77', 'ecbf78', 'ecbf79', 'ecbf7a', 'ecbf7b', 'ecbf7c', 'ecbf7d', 'ecbf7e', 'ecbf7f', 'ecbf80', 'ecbf81', 'ecbf82', 'ecbf83', 'ecbf84', 'ecbf85', 'ecbf86', 'ecbf87', 'ecbf88', 'ecbf89', 'ecbf8a', 'ecbf8b', 'ecbf8c', 'ecbf8d', 'ecbf8e', 'ecbf8f', 'ecbf90', 'ecbf91', 'ecbf92', 'ecbf93', 'ecbf94', 'ecbf95', 'ecbf96', 'ecbf97', 'ecbf98', 'ecbf99', 'ecbf9a', 'ecbf9b', 'ecbf9c', 'ecbf9d', 'ecbf9e', 'ecbf9f', 'ecbf100', 'ecbf101', 'ecbf102', 'ecbf103', 'ecbf104', 'ecbf105', 'ecbf106', 'ecbf107', 'ecbf108', 'ecbf109', 'ecbf10a', 'ecbf10b', 'ecbf10c', 'ecbf10d', 'ecbf10e', 'ecbf10f', 'ecbf110', 'ecbf111', 'ecbf112', 'ecbf113', 'ecbf114', 'ecbf115', 'ecbf116', 'ecbf117', 'ecbf118', 'ecbf119', 'ecbf11a', 'ecbf11b', 'ecbf11c', 'ecbf11d', 'ecbf11e', 'ecbf11f', 'ecbf120', 'ecbf121', 'ecbf122', 'ecbf123', 'ecbf124', 'ecbf125', 'ecbf126', 'ecbf127', 'ecbf128', 'ecbf129', 'ecbf12a', 'ecbf12b', 'ecbf12c', 'ecbf12d', 'ecbf12e', 'ecbf12f', 'ecbf130', 'ecbf131', 'ecbf132', 'ecbf133', 'ecbf134', 'ecbf135', 'ecbf136', 'ecbf137', 'ecbf138', 'ecbf139', 'ecbf13a', 'ecbf13b', 'ecbf13c', 'ecbf13d', 'ecbf13e', 'ecbf13f', 'ecbf140', 'ecbf141', 'ecbf142', 'ecbf143', 'ecbf144', 'ecbf145', 'ecbf146', 'ecbf147', 'ecbf148', 'ecbf149', 'ecbf14a', 'ecbf14b', 'ecbf14c', 'ecbf14d', 'ecbf14e', 'ecbf14f', 'ecbf150', 'ecbf151', 'ecbf152', 'ecbf153', 'ecbf154', 'ecbf155', 'ecbf156', 'ecbf157', 'ecbf158', 'ecbf159', 'ecbf15a', 'ecbf15b', 'ecbf15c', 'ecbf15d', 'ecbf15e', 'ecbf15f', 'ecbf160', 'ecbf161', 'ecbf162', 'ecbf163', 'ecbf164', 'ecbf165', 'ecbf166', 'ecbf167', 'ecbf168', 'ecbf169', 'ecbf16a', 'ecbf16b', 'ecbf16c', 'ecbf16d', 'ecbf16e', 'ecbf16f', 'ecbf170', 'ecbf171', 'ecbf172', 'ecbf173', 'ecbf174', 'ecbf175', 'ecbf176', 'ecbf177', 'ecbf178', 'ecbf179', 'ecbf17a', 'ecbf17b', 'ecbf17c', 'ecbf17d', 'ecbf17e', 'ecbf17f', 'ecbf180', 'ecbf181', 'ecbf182', 'ecbf183', 'ecbf184', 'ecbf185', 'ecbf186', 'ecbf187', 'ecbf188', 'ecbf189', 'ecbf18a', 'ecbf18b', 'ecbf18c', 'ecbf18d', 'ecbf18e', 'ecbf18f', 'ecbf190', 'ecbf191', 'ecbf192', 'ecbf193', 'ecbf194', 'ecbf195', 'ecbf196', 'ecbf197', 'ecbf198', 'ecbf199', 'ecbf19a', 'ecbf19b', 'ecbf19c', 'ecbf19d', 'ecbf19e', 'ecbf19f', 'ecbf200', 'ecbf201', 'ecbf202', 'ecbf203', 'ecbf204', 'ecbf205', 'ecbf206', 'ecbf207', 'ecbf208', 'ecbf209', 'ecbf20a', 'ecbf20b', 'ecbf20c', 'ecbf20d', 'ecbf20e', 'ecbf20f', 'ecbf210', 'ecbf211', 'ecbf212', 'ecbf213', 'ecbf214', 'ecbf215', 'ecbf216', 'ecbf217', 'ecbf218', 'ecbf219', 'ecbf21a', 'ecbf21b', 'ecbf21c', 'ecbf21d', 'ecbf21e', 'ecbf21f', 'ecbf220', 'ecbf221', 'ecbf222', 'ecbf223', 'ecbf224', 'ecbf225', 'ecbf226', 'ecbf227', 'ecbf228', 'ecbf229', 'ecbf22a', 'ecbf22b', 'ecbf22c', 'ecbf22d', 'ecbf22e', 'ecbf22f', 'ecbf230', 'ecbf231', 'ecbf232', 'ecbf233', 'ecbf234', 'ecbf235', 'ecbf236', 'ecbf237', 'ecbf238', 'ecbf239', 'ecbf23a', 'ecbf23b', 'ecbf23c', 'ecbf23d', 'ecbf23e', 'ecbf23f', 'ecbf240', 'ecbf241', 'ecbf242', 'ecbf243', 'ecbf244', 'ecbf245', 'ecbf246', 'ecbf247', 'ecbf248', 'ecbf249', 'ecbf24a', 'ecbf24b', 'ecbf24c', 'ecbf24d', 'ecbf24e', 'ecbf24f', 'ecbf250', 'ecbf251', 'ecbf252', 'ecbf253', 'ecbf254', 'ecbf255')
    # 16진수 색상 값을 10진수로 변환하여 각 색상 구성 요소 (r, g, b)를 추출 후 my_colormap으로 결합하여 색상 맵 생성
    my_colormap_vals_dec = np.array([int(element,base=16) for element in my_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)
    # 시각화를 위한 offset/frequency values의 그리드 생성
    offsets_mesh, freqs_mesh = np.meshgrid(np.arange(-10,10,0.01), np.arange(2.5,22.5,0.1))
    loss_mesh = np.zeros_like(freqs_mesh)
    # 모든 파라미터 집합에 대한 손실 계산
    for idslope, slope in np.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)
```

Your loss = 16.419, Correct loss = 16.419

정확한 계산여부는 **finite differences**, 을 통해 알 수 있음. 함수를 평가한 후 파라미터 중 하나를 매우 작은 양만큼 변경하고 그 양으로 정규화하면 그레디언트에 대한 근사를 얻을 수 있음

$$\frac{\partial L}{\partial \phi_0} \approx \frac{L[\phi_0 + \delta, \phi_1] - L[\phi_0, \phi_1]}{\delta}$$
$$\frac{\partial L}{\partial \phi_1} \approx \frac{L[\phi_0, \phi_1 + \delta] - L[\phi_0, \phi_1]}{\delta}$$

parameters가 많은 경우 사용 불가능(parameters가 많을 경우 손실함수의 평가의 횟수도 증가하기 때문, gradients를 직접 계산하는 것이 더 효율적)

```
# finite differences 활용
delta = 0.0001
dl_dphi0_est = (compute_loss(data[0,:],data[1:],model,phi+np.array([[delta],[0]])) - \
compute_loss(data[0,:],data[1:],model,phi))/delta
dl_dphi1_est = (compute_loss(data[0,:],data[1:],model,phi+np.array([[0],[delta]])) - \
compute_loss(data[0,:],data[1:],model,phi))/delta
print("Approx gradients: (%3.3f,%3.3f)"%(dl_dphi0_est,dl_dphi1_est))
```