20171664 소프트웨어학부 이범석

HW6 p167 ~ p225 In and Out Practice

실습 결과

1. numpy와 matplotlib을 import했습니다.

   또한 나이와 몸무게의 인공 데이터를 생성했습니다.

In [1]:
```
#리스트 5-1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(seed=1)
X_min=4
X_max=30
X_n=16
X=5+25*np.random.rand(X_n)
Prm_c=[170,108,0.2]
T=Prm_c[0]-Prm_c[1]*np.exp(-Prm_c[2]*X)\
+4*np.random.randn(X_n)
np.savez('ch5_data.npz', X=X, X_min=X_min, X_max=X_max, X_n=X_n, T=T)
```

2. X의 내용(나이)을 표시했습니다.

In [2]:
```
#리스트 5-1-(2)
print(X)
```

```
[15.42555012 23.00811234  5.00285937 12.55831432  8.66889727  7.30846487
  9.65650528 13.63901818 14.91918686 18.47041835 15.47986286 22.13048751
 10.11130624 26.95293591  5.68468983 21.76168775]
```

3. np.round 함수를 사용해 소수점 이하 표시를 반올림하였습니다.

In [3]:
```
#리스트 5-1-(3)
print(np.round(X,2))
```

```
[15.43 23.01  5.   12.56  8.67  7.31  9.66 13.64 14.92 18.47 15.48 22.13
 10.11 26.95  5.68 21.76]
```

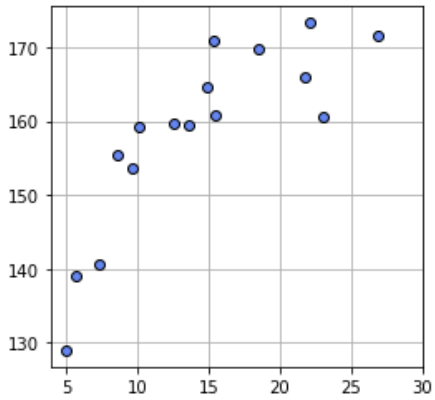
4. T의 내용(키)도 확인하였습니다.

In [4]:
```
#리스트 5-1-(4)
print(np.round(T,2))
```

```
[170.91 160.68 129.   159.7  155.46 140.56 153.65 159.43 164.7  169.65
 160.71 173.29 159.31 171.52 138.96 165.87]
```

5. X와 T를 그래프로 표시하였습니다.

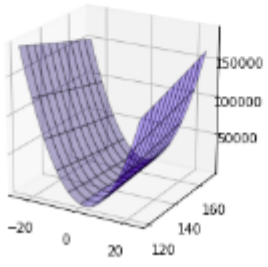
In [5]:
```
#리스트 5-1-(5)
plt.figure(figsize=(4,4))
plt.plot(X,T,marker='o', linestyle='None',
        markeredgecolor='black', color='cornflowerblue')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

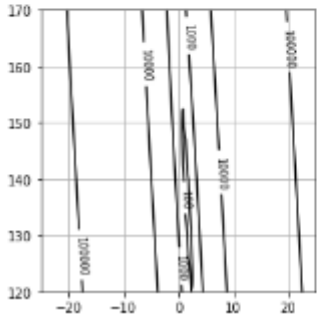


6. 기울기를 나타내는 w0과 절편을 나타내는 w1의 값을 결정하고 평균 제곱 오차 J를 계산한 그래프를 나타냈습니다.

In [6]:

```
#리스트 5-1-(6)
from mpl_toolkits.mplot3d import Axes3D

def mse_line(x,t,w):
    y = w[0]*x+w[1]
    mse = np.mean((y-t)**2)
    return mse

xn = 100
w0_range=[-25,25]
w1_range=[120,170]
x0=np.linspace(w0_range[0], w0_range[1], xn)
x1=np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0,x1)
J=np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X,T,(x0[i0], x1[i1]))


plt.figure(figsize = (9.5, 4))
plt.subplots_adjust(wspace = 0.5)

ax = plt.subplot(1,2,1, projection='3d')
ax.plot_surface(xx0, xx1, J, rstride=10, cstride=10, alpha=0.3,
                color='blue', edgecolor='black')
ax.set_xticks([-20,0,20])
ax.set_yticks([120,140,160])
ax.view_init(20,-60)

plt.subplot(1,2,2)
cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=[100,1000,10000,100000])
cont.clabel(fmt='%1.0f',fontsize=8)
plt.grid(True)
plt.show()
```





7. 인수 데이터를 전달하여 기울기를 반환하는 함수를 만들었습니다.

In [7]:

```
#리스트 5-1-(7)
def dmse_line(x,t,w):
    y=w[0]* x + w[1]
    d_w0=2*np.mean((y-t) *x)
    d_w1=2*np.mean(y-t)
    return d_w0, d_w1
```

8. w = [10, 165] 의 기울기를 구해보았습니다.

In [8]:
```
#리스트 5-1-(8)
d_w=dmse_line(X,T,[10,165])
print(np.round(d_w,1))
```

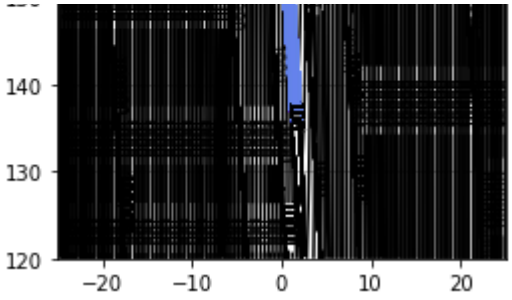
```
[5046.3  301.8]
```

9. 경사 하강법의 함수를 구현하고, 기울기의 각 요소의 절댓값이 eps = 0.1보다 작을 때까지 구현하고, 마지막으로 얻어진 w값을 표시하고, w의 갱신 내역을 그래프로 표현했습니다.

In [9]:

```
#리스트 5-1-(9)
def fit_line_num(x, t):
    w_init = [10.0, 165.0]
    alpha = 0.001
    i_max=100000
    eps = 0.1
    w_i = np.zeros([i_max, 2])
    w_i[0, :] = w_init
    for i in range(1,i_max):
        dmse = dmse_line(x, t, w_i[i-1])
        w_i[i, 0] = w_i[i - 1, 0] - alpha* dmse[0]
        w_i[i, 1] = w_i[i -1 , 1] - alpha*dmse[1]
        if max(np.absolute(dmse))<eps:
            break
    w0 = w_i[i, 0]
    w1 = w_i[i, 1]
    w_i = w_i[:i, :]
    return w0, w1, dmse, w_i



plt.figure(figsize=(4,4))
xn=100
w0_range=[-25,25]
w1_range=[120,170]
x0=np.linspace(w0_range[0], w0_range[1], xn)
x1=np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J=np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X,T,(x0[i0], x1[i1]))
    cont =plt.contour(xx0, xx1, J, 30, colors='black',
                      levels=(100,1000,10000,100000))
    cont.clabel(fmt='%1.0f', fontsize=8)
    plt.grid(True)



W0, W1, dMSE, W_history = fit_line_num(X,T)

print('반복 횟수 {0}'.format(W_history.shape[0]))
print('W=[{0:.6f},{1:.6f}]'.format(W0, W1))
print('dMSE=[{0:.6f},{1:.6f}]'.format(dMSE[0], dMSE[1]))
print('MSE={0:.6f}'.format(mse_line(X,T,[W0, W1])))
plt.plot(W_history[:,0], W_history[:, 1], '.-',
        color='gray',markersize=10, markeredgecolor='cornflowerblue')
plt.show()
```

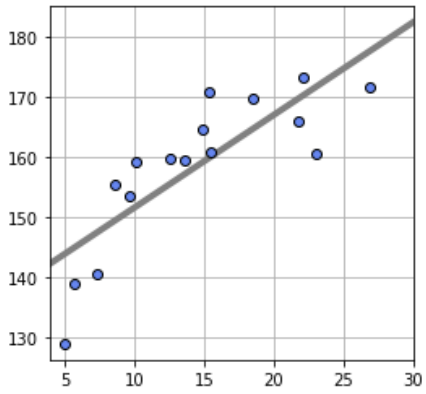
```
반복 횟수 13820
W=[1.539947,136.176160]
dMSE=[-0.005794,0.099991]
MSE=49.027452
```



10. 구한 W0과 W1의 값을 직선 식에 대입하여 데이터 분포에 겹쳐서 그려 보았습니다.

In [10]:
```
#리스트 5-1-(10)

def show_line(w):
    xb = np.linspace(X_min, X_max, 100)
    y = w[0] * xb + w[1]
    plt.plot(xb, y, color= (.5, .5, .5), linewidth=4)

plt.figure(figsize= (4,4))
W = np.array([W0,W1])
mse = mse_line(X,T,W)
print("w0={0:.3f}, w1={1:.3f}".format(W0, W1))
print("SD={0:.3f} cm".format(np.sqrt(mse)))
show_line(W)
plt.plot(X,T,marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=1.540, w1=136.176
SD=7.002 cm
```

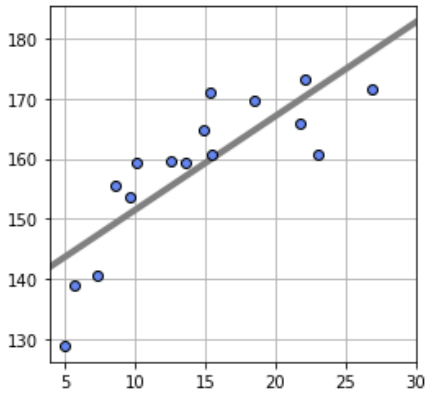


11. 입력 데이터 X와 목표 데이터 T의 값을 식에 넣어 w(해석해)를 찾았습니다.

    경사 하강법과 근사한 결과가 얻어졌습니다.

In [11]:
```
#리스트 5-1-(11)
def fit_line(x,t):
    mx = np.mean(x)
    mt = np.mean(t)
    mtx = np.mean(t*x)
    mxx = np.mean(x*x)
    w0 =(mtx-mt*mx)/(mxx - mx**2)
    w1 = mt - w0 * mx
    return np.array([w0, w1])

W = fit_line(X,T)
print("w0={0:3f}, w1={1:.3f}".format(W[0], W[1]))
mse= mse_line(X,T,W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.figure(figsize=(4,4))
show_line(W)
plt.plot(X,T,marker='o', linestyle='None',
        color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=1.557575, w1=135.872
SD=7.001 cm
```



12. 몸무게는 키의 제곱에 비례한다는 식을 만들었습니다.

    원래 나이 x를 x0, 몸무게의 데이터를 X1로 추가했습니다.

In [12]:
```
#리스트 5-1-(12)
X0=X
X0_min = 5
X0_max=30
np.random.seed(seed=1)
X1 = 23*(T/100)**2 + 2*np.random.randn(X_n)
X1_min=40
X1_max=75
```

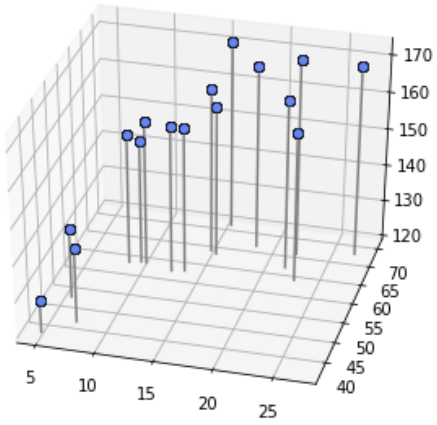
13. 데이터를 표시했습니다.

In [13]:
```
#리스트 5-1-(13)
print(np.round(X0,2))
print(np.round(X1,2))
print(np.round(T,2))
```

```
[15.43 23.01  5.   12.56  8.67  7.31  9.66 13.64 14.92 18.47 15.48 22.13
 10.11 26.95  5.68 21.76]
[70.43 58.15 37.22 56.51 57.32 40.84 57.79 56.94 63.03 65.69 62.33 64.95
 57.73 66.89 46.68 61.08]
[170.91 160.68 129.   159.7  155.46 140.56 153.65 159.43 164.7  169.65
 160.71 173.29 159.31 171.52 138.96 165.87]
```

14. 16명의 X0, X1, T를 3차원 플롯 그래프로 표시했습니다.

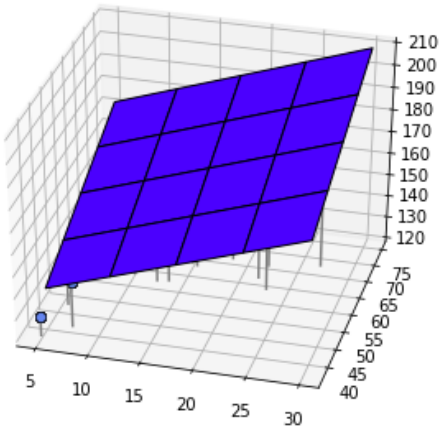
In [14]:
```
#리스트 5-1-(14)
def show_data2(ax,x0,x1,t):
    for i in range(len(x0)):
        ax.plot([x0[i], x0[i]], [x1[i], x1[i]],
            [120, t[i]], color='gray')
        ax.plot(x0, x1, t, 'o',
                color='cornflowerblue', markeredgecolor='black',
                markersize=6, markeredgewidth=0.5)
        ax.view_init(elev=35, azim=-75)


plt.figure(figsize=(6,5))
ax = plt.subplot(1,1,1,projection='3d')
show_data2(ax,X0,X1,T)
plt.show()
```



15. 임의의 w에 대해 면을 그리는 함수와 평균 제곱 오차를 계산하는 함수를 만들었습니다.
    그리고 면의 함수를 3차원으로 나타냈습니다.

In [15]:
```
#리스트 5-1-(15)
def show_plane(ax,w):
    px0 = np.linspace(X0_min,X0_max, 5)
    px1 = np.linspace(X1_min,X1_max, 5)
    px0, px1 = np.meshgrid(px0, px1)
    y = w[0]*px0+w[1]*px1 + w[2]
    ax.plot_surface(px0, px1, y, rstride=1, cstride=1,
                    color='blue', edgecolor='black')

def mse_plane(x0, x1, t, w):
    y=w[0] * x0+w[1]*x1 + w[2]
    mse = np.mean((y-t)**2)
    return mse

plt.figure(figsize=(6,5))
ax=plt.subplot(1,1,1,projection='3d')
W=[1.5,1,90]
show_plane(ax, W)
show_data2(ax, X0, X1, T)
mse = mse_plane(X0, X1, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.show()
```

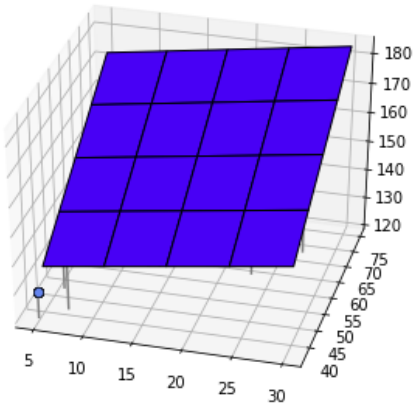
```
SD=12.876 cm
```



16. 해석해에 의한 평면 모델의 피팅 결과를 나타냈습니다.

In [16]:
```
#리스트 5-1-(16)
def fit_plane(x0,x1,t):
    c_tx0 = np.mean(t*x0) - np.mean(t) *np.mean(x0)
    c_tx1 = np.mean(t*x1) - np.mean(t) * np.mean(x1)
    c_x0x1 = np.mean(x0*x1) - np.mean(x0) *np.mean(x1)
    v_x0 = np.var(x0)
    v_x1 = np.var(x1)
    w0 = (c_tx1*c_x0x1 - v_x1 *c_tx0)/(c_x0x1**2 - v_x0*v_x1)
    w1 = (c_tx0*c_x0x1 - v_x0 *c_tx1)/(c_x0x1**2 - v_x0*v_x1)
    w2 = -w0 * np.mean(x0) - w1*np.mean(x1) + np.mean(t)
    return np.array([w0,w1,w2])
plt.figure(figsize=(6,5))
ax = plt.subplot(1,1,1, projection = '3d')
W=fit_plane(X0, X1,T)
print("w0={0:.1f}, w1={1:.1f}".format(W[0], W[1],W[2]))
show_plane(ax,W)
show_data2(ax,X0,X1,T)
mse=mse_plane(X0,X1,T,W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.show()
```

```
w0=0.5, w1=1.1
SD=2.546 cm
```



16. 위에서 작성한 데이터를 로드했습니다.

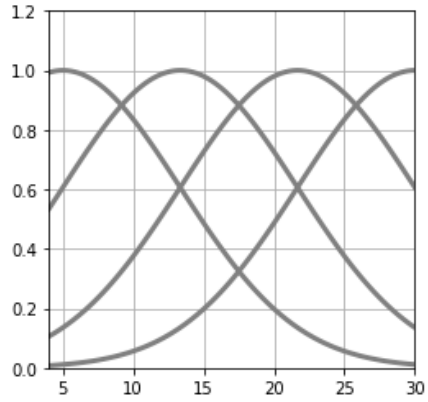
In [17]:
```
outfile = np.load('ch5_data.npz')
X = outfile['X']
X_min= outfile['X_min']
X_max= outfile['X_max']
X_n=outfile['X_n']
T=outfile['T']
```

17. 가우스 함수를 정의했습니다.

In [18]:
```
#리스트 5-2-(2)
def gauss(x, mu, s):
    return np.exp(-(x-mu)**2/(2*s**2))
```

18. 4개의 가우스 함우를 나이의 범위 5~30으로 정하고 이를 나타냈습니다.

In [19]:
```
#리스트 5-2-(3)
M=4
plt.figure(figsize=(4,4))
mu=np.linspace(5,30,M)
s=mu[1]-mu[0]
xb=np.linspace(X_min, X_max, 100)
for j in range(M):
    y=gauss(xb, mu[j], s)
    plt.plot(xb, y, color='gray', linewidth=3)
plt.grid(True)
plt.xlim(X_min, X_max)
plt.ylim(0,1.2)
plt.show()
```



19. 선형 기저 함수 모델을 정의했습니다.

In [20]:
```
#리스트 5-2-(4)
def gauss_func(w,x):
    m=len(w)-1
    mu=np.linspace(5,30,m)
    s=mu[1]-mu[0]
    y=np.zeros_like(x)
    for j in range(m):
        y=y+w[j] * gauss(x, mu[j], s)
    y=y+w[m]
    return y
```

20. 평균 제곱 오차를 계산하는 함수를 만들었습니다. 이는 피팅의 수준을 산출합니다.

In [21]:
```
#리스트 5-2-(5)
def mse_gauss_func(x, t, w):
    y = gauss_func(w,x)
    mse = np.mean((y-t)**2)
    return mse
```

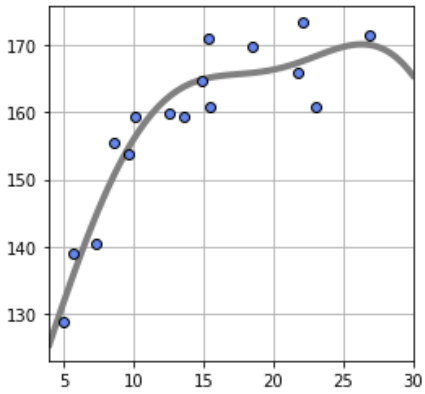
21. 선형 기저 함수 모형의 매개 변수의 해석해를 제공하는 함수를 만들었습니다.

In [22]:
```
#리스트 5-2-(6)
def fit_gauss_func(x, t, m):
    mu = np.linspace(5, 30, m)
    s= mu[1] - mu[0]
    n = x.shape[0]
    psi = np.ones((n, m+1))
    for j in range(m):
        psi[:, j] = gauss(x, mu[j], s)
    psi_T = np.transpose(psi)


    b = np.linalg.inv(psi_T.dot(psi))
    c = b.dot(psi_T)
    w = c.dot(t)
    return w
```

22. 선형 기저 함수 모델의 피팅 결과를 나타냈습니다.

In [23]:
```
#리스트 5-2-(7)
def show_gauss_func(w):
    xb = np.linspace(X_min, X_max, 100)
    y = gauss_func(w,xb)
    plt.plot(xb, y, c= [.5, .5, .5], lw=4)


plt.figure(figsize = (4,4))
M = 4
W = fit_gauss_func(X,T,M)
show_gauss_func(W)
plt.plot(X,T,marker='o', linestyle ='None',
        color='cornflowerblue',  markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
mse = mse_gauss_func(X,T,W)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

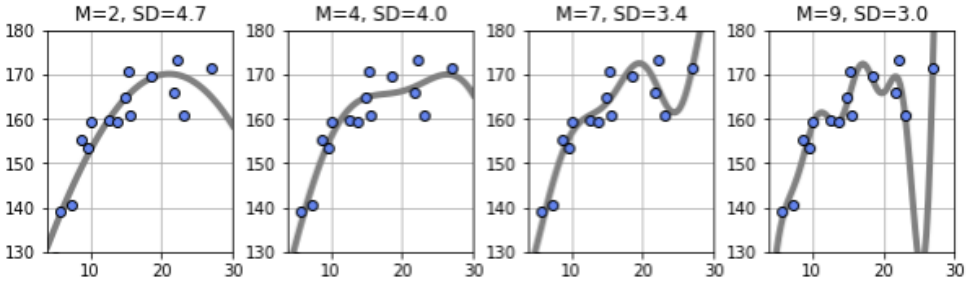
```
SD=3.98 cm
```



23. M을 늘리며 표준 편차를 줄여보았지만 가우스 기저 함수가 변했습니다.

    과적합이 일어났습니다.

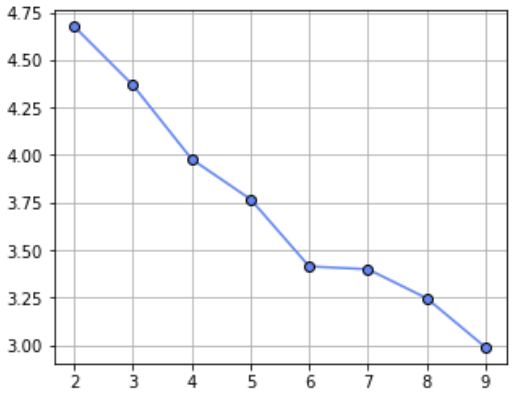
In [24]:
```
#리스트 5-2-(8)
plt.figure(figsize=(10, 2.5))
plt.subplots_adjust(wspace = 0.3)
M = [2, 4, 7, 9]
for i in range(len(M)):
    plt.subplot(1, len(M), i+1)
    W = fit_gauss_func(X,T,M[i])
    show_gauss_func(W)
    plt.plot(X,T,marker='o', linestyle = 'None',
            color =  'cornflowerblue', markeredgecolor='black')
    plt.xlim(X_min, X_max)
    plt.grid(True)
    plt.ylim(130,180)
    mse = mse_gauss_func(X,T,W)
    plt.title("M={0:d}, SD={1:.1f}".format(M[i], np.sqrt(mse)))
plt.show()
```



24. M이 2일때부터 9까지의 표준 편차를 계산하고 그래프로 나타냈습니다.

    M이 증가함에 따라 표준 편차는 감소했습니다. 하지만 이는 최적의 M의 기준으로는 볼 수 없습니다.

In [25]:
```
#리스트 5-2-(9)
plt.figure(figsize=(5,4))
M = range(2,10)
mse2 = np.zeros(len(M))
for i in range(len(M)):
    W = fit_gauss_func(X,T,M[i])
    mse2[i]=np.sqrt(mse_gauss_func(X,T,W))
plt.plot(M, mse2, marker='o',
         color='cornflowerblue', markeredgecolor='black')
plt.grid(True)
plt.show()
```



25. 최적의 M을 찾기 위하여 X와 t의 1/4을 테스트 데이터로, 나머지 3/4를 훈련 데이터로 나누고,

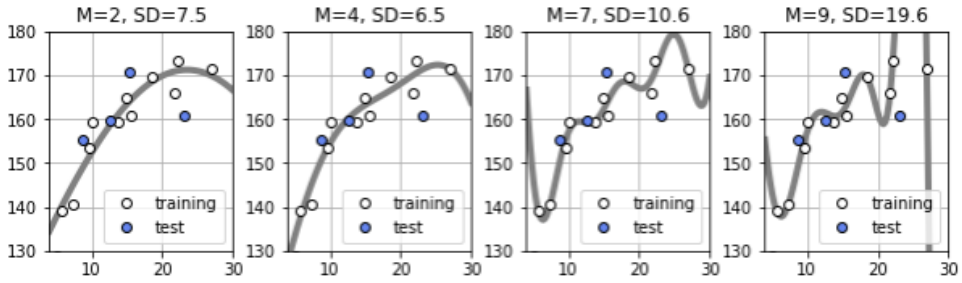
    w는 훈련 데이터만을 사용하여 최적화합니다. 훈련 데이터의 평균 제곱 오차를 최소화하도록 w를 선택합니다.

    w를 사용하여 테스트 데이터의 평균 제곱 오차를 계산하고, M의 평가 기준으로 삼습니다.

    이처럼 훈련에 이용하지 않은 미지의 데이터에 대한 예측 오차로 M을 평가하는 것을 홀드 아웃 검증이라고 합니다.

    이 데이터에 대한 그래프를 표현했습니다.

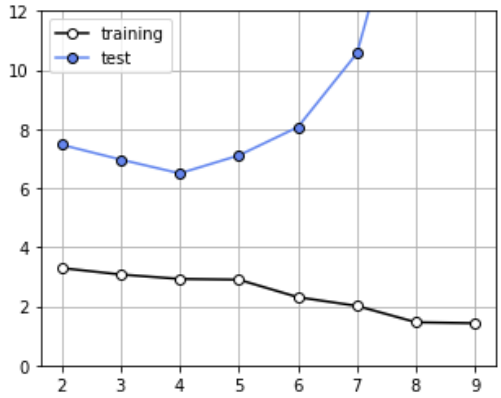
In [26]:
```
#리스트 5-2-(10)
X_test = X[:int(X_n/4+1)]
T_test = T[:int(X_n/4+1)]
X_train = X[int(X_n/4+1):]
T_train = T[int(X_n/4+1):]

plt.figure(figsize=(10,2.5))

plt.subplots_adjust(wspace=0.3)
M=[2,4,7,9]
for i in range(len(M)):
    plt.subplot(1,len(M), i + 1)
    W = fit_gauss_func(X_train, T_train, M[i])
    show_gauss_func(W)
    plt.plot(X_train, T_train, marker='o',
             linestyle='None', color='white',
             markeredgecolor='black', label='training')
    plt.plot(X_test,T_test,marker='o',  linestyle ='None',
             color='cornflowerblue',
             markeredgecolor='black', label='test')
    plt.legend(loc='lower right', fontsize=10, numpoints=1)
    plt.xlim(X_min, X_max)
    plt.ylim(130, 180)
    plt.grid(True)
    mse=mse_gauss_func(X_test, T_test, W)
    plt.title("M={0:d}, SD={1:.1f}".format(M[i], np.sqrt(mse)))
plt.show()
```



26. 훈련 데이터와 데스트 데이터의 오차를 그래프로 나타냈습니다.

    M = 4의 경우가 데이터에 가장 적합하다는 결론이 나왔습니다.

In [27]:
```
#리스트 5-2-(11)
plt.figure(figsize=(5,4))
M =range(2, 10)
mse_train = np.zeros(len(M))
mse_test = np.zeros(len(M))
for i in range(len(M)):
    W = fit_gauss_func(X_train, T_train, M[i])
    mse_train[i] = np.sqrt(mse_gauss_func(X_train, T_train, W))
    mse_test[i] = np.sqrt(mse_gauss_func(X_test,T_test, W))
plt.plot(M,mse_train, marker='o', linestyle='-',
         markerfacecolor='white', markeredgecolor='black',
         color='black', label='training')
plt.plot(M,mse_test, marker='o', linestyle = '-',
         color='cornflowerblue', markeredgecolor='black',
         label='test')
plt.legend(loc='upper left',fontsize=10)
plt.ylim(0,12)
plt.grid(True)
plt.show()
```



27. 데이터를 K 분할하여 각각의 평균 제곱 오차를 출력하는 함수를 만들었습니다.

In [28]:
```
#리스트 5-2-(12)
def kfold_gauss_func(x,t,m,k):
    n = x.shape[0]
    mse_train = np.zeros(k)
    mse_test =np.zeros(k)
    for i in range(0, k):
        x_train=x[np.fmod(range(n), k)!=i]
        t_train=t[np.fmod(range(n), k)!=i]
        x_test=x[np.fmod(range(n), k)==i]
        t_test=t[np.fmod(range(n), k)==i]
        wm=fit_gauss_func(x_train, t_train, m)
        mse_train[i] = mse_gauss_func(x_train, t_train, wm)
        mse_test[i] = mse_gauss_func(x_test, t_test, wm)
    return mse_train, mse_test
```

28. n을 k로 나눈 나머지를 출력하는 함수의 결과를 출력했습니다.

In [29]:
```
#리스트 5-2-(13)
np.fmod(range(10),5)
```

Out[29]:
```
array([0, 1, 2, 3, 4, 0, 1, 2, 3, 4])
```

29. 기저 수 M = 4, 분할 수 K = 4 로 함수를 실행해 보았습니다.

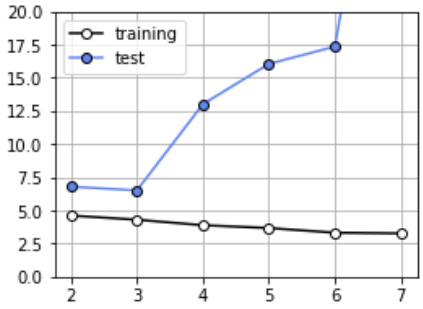
In [30]:
```
#리스트 5-2-(14)
M=4
K=4
kfold_gauss_func(X,T,M,K)
```

Out[30]:
```
(array([12.87927851,  9.81768697, 17.2615696 , 12.92270498]),
 array([ 39.65348229, 734.70782017,  18.30921743,  47.52459642]))
```

30. 분할 수를 16으로 하고, 2에서 7까지의 M의 오차 평균을 계산하여 그래프로 나타냈습니다.

    M이 3일 때 테스트 데이터의 오차가 가장 작은 것을 알 수 있었습니다.

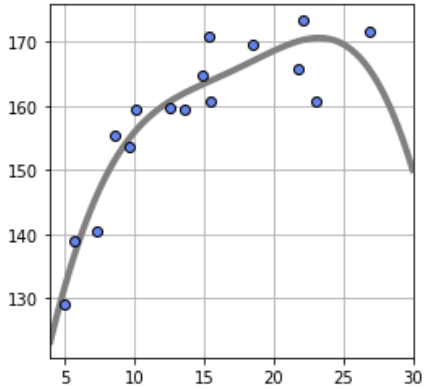
In [31]:
```
#리스트 5-2-(15)
M= range (2, 8)
K = 16
Cv_Gauss_train = np.zeros((K, len(M)))
Cv_Gauss_test = np.zeros((K, len(M)))

for i in range(0, len(M)):
    Cv_Gauss_train[:, i], Cv_Gauss_test[:, i] =\
    kfold_gauss_func(X, T, M[i], K)
mean_Gauss_train = np.sqrt(np.mean(Cv_Gauss_train, axis=0))
mean_Gauss_test = np.sqrt(np.mean(Cv_Gauss_test, axis=0))


plt. figure(figsize=(4, 3))
plt.plot(M, mean_Gauss_train, marker='o', linestyle='-',
         color='k', markerfacecolor='w', label='training')
plt.plot(M, mean_Gauss_test, marker='o', linestyle='-',
         color='cornflowerblue', markeredgecolor='black', label='test')
plt. legend (loc='upper left', fontsize=10)
plt.ylim(0, 20)
plt.grid(True)
plt.show()
```



31. M = 3이 최적이므로 해당 결과를 대입하고, w의 모든 데이터를 사용해 계산했습니다.

In [32]:
```
#리스트 5-2-(16)
M=3
plt.figure(figsize=(4,4))
W=fit_gauss_func(X,T,M)
show_gauss_func(W)
plt.plot(X,T,marker='o', linestyle='None',
         color='cornflowerblue',markeredgecolor='black')
plt.xlim([X_min, X_max])
plt.grid(True)
mse=mse_gauss_func(X,T,W)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

```
SD=4.37 cm
```



32. 새로운 모델 A를 정의하고, 표시용 함수와 MSE 출력 함수를 정의했습니다.

In [33]:
```
#리스트 5-2-(17)
def model_A(x,w):
    y=w[0]-w[1]*np.exp(-w[2]*x)
    return y


def show_model_A(w):
    xb = np.linspace(X_min, X_max, 100)
    y=model_A(xb, w)
    plt.plot(xb, y , c=[.5,.5,.5], lw=4)


def mse_model_A(w,x,t):
    y=model_A(x,w)
    mse=np.mean((y-t)**2)
    return mse
```

33. 매개 변수의 최적화 함수를 만들었습니다.

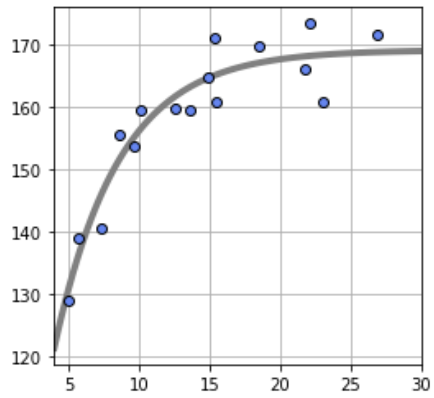
In [34]:
```
#리스트 5-2-(18)
from scipy.optimize import minimize

def fit_model_A(w_init, x, t):
    res1=minimize(mse_model_A, w_init, args=(x,t), method="powell")
    return res1.x
```

34. 최적화 함수를 움직여 보았습니다.

    오차의 표준 편차가 직선 모델과 선형 기저 함수 모델보다 훨씬 낮아졌습니다.

In [35]:
```
#리스트 5-2-(19)
plt.figure(figsize= (4,4))
W_init=[100,0,0]
W=fit_model_A(W_init, X, T)
print("w0={0:.1f}, w1={1:.1f}, w2={2:.1f}".format(W[0],W[1],W[2]))
show_model_A(W)
plt.plot(X,T,marker='o', linestyle='None',
        color='cornflowerblue',markeredgecolor='black')
plt.xlim(X_min,X_max)
plt.grid(True)
mse=mse_model_A(W,X,T)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

```
w0=169.0, w1=113.7, w2=0.2
SD=3.86 cm
```

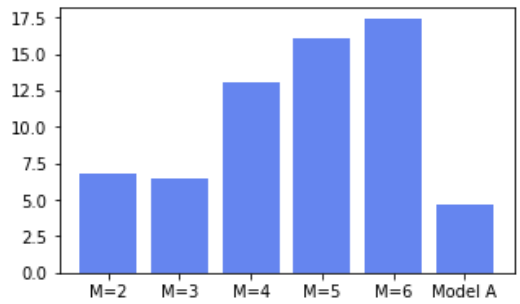


35. 선형 기저 함수 모델과 모델 A의 LOOCV를 실시하여 비교한 그래프를 나타냈습니다.

    모델 A가 데이터에 잘 어울린다고 생각할 수 있습니다.

In [36]:
```
#리스트 5-2-(20)

def kfold_model_A(x, t, k):
    n = len(x)
    mse_train = np.zeros(k)
    mse_train = np.zeros(k)
    mse_test = np.zeros(k)
    for i in range(0, k):
        x_train = x[np.fmod(range(n), k) != i]
        t_train = t[np.fmod(range(n), k) != i]
        x_test = x[np.fmod(range(n), k) == i]
        t_test = t[np.fmod(range(n), k) == i]
        wm = fit_model_A(np.array([169, 113, 0.2]), x_train, t_train)
        mse_train[i] = mse_model_A(wm, x_train, t_train)
        mse_test[i] = mse_model_A(wm, x_test, t_test)
    return mse_train, mse_test

K = 16
Cv_A_train, Cv_A_test = kfold_model_A(X, T, K)
mean_A_test = np.sqrt(np.mean(Cv_A_test))
print("Gauss (M=3) SD={0:.2f} cm", format(mean_Gauss_test[1]))
print("Model A SD={0:.2f} cm", format(mean_A_test))
SD = np.append(mean_Gauss_test[0:5], mean_A_test)
M = range(6)
label = ["M=2", "M=3", "M=4", "M=5", "M=6", "Model A"]
plt.figure(figsize=(5,3))
plt.bar(M,SD,tick_label=label, align="center",
        facecolor="cornflowerblue")
plt.show()
```

```
Gauss (M=3) SD={0:.2f} cm 6.514042205279559
Model A SD={0:.2f} cm 4.720687846139534
```



수행 소감

수많은 트러블슈팅을 통해 최적화된 모델을 구현하는 방법이 여러 가지라는 것을 알게 되었습니다.

수많은 데이터들을 효율적으로 해석할 수 있는 방법이었다고 생각합니다.

긴 글 읽어 주셔서 감사합니다.

이범석

국민대학교 소프트웨어학부, 20171664

Mobile 010-6401-6042

qpwoeiru6486@gmail.com

ijkoo16@kookmin.ac.kr