In [1]:
```
#PART-1
```

In [2]:
```
#train data whole
import numpy as np
x = np.array([-1, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.
1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
t = np.array([5.12, 4.97, 4.92, 4.83,4.90, 5.06, 5.29, 5.34, 5.36, 5.
76, 5.99, 6.30, 6.66, 6.70, 7.49, 7.92, 8.48, 9.09, 9.70, 10.30, 10.9
8])
```

In [3]:
```
train_set_x = np.array((x[::3], x[1::3], x[2::3]))
train_set_t = np.array((t[::3], t[1::3], t[2::3]))
print(train_set_x)
```

```
[[-1.  -0.7 -0.4 -0.1  0.2  0.5  0.8]
 [-0.9 -0.6 -0.3  0.   0.3  0.6  0.9]
 [-0.8 -0.5 -0.2  0.1  0.4  0.7  1. ]]
```

In [4]:
```
#defining the basis function
# def guassian_basis(x,mean,var):
#     return np.exp((abs(x-mean)**2)/(2*var))

# #mean
# def mean(numbers):
#     return float(sum(numbers))/max(len(numbers),1)
```

In [5]:
```
#basis
# mean_1 = mean(train_set_x[0])
# var = np.var(train_set_x[0])
# basis_1 = [guassian_basis(x,mean_1,var) for x in train_set_x[0]]
# mean_2 = mean(train_set_x[1])
# var = np.var(train_set_x[1])
# basis_2 = [guassian_basis(x,mean_2,var) for x in train_set_x[1]]
# mean_3 = mean(train_set_x[2])
# var = np.var(train_set_x[2])
# basis_3 = [guassian_basis(x,mean_3,var) for x in train_set_x[2]]

def basis_1(count):
    return np.ones(count)

def basis_2(train_set):
    return [np.exp(-((x-0.5)**2/0.1)) for x in train_set]

def basis_3(train_set):
    return [np.exp(-((x+0.5)**2/0.1)) for x in train_set]
```

In [6]:
```
basis = np.array((basis_1(len(train_set_x[0])),basis_2(train_set_x[0
]),basis_3(train_set_x[0])))
```

In [7]:
```
#calculate w
from numpy.linalg import inv
def calculate_w(basis, reg_parameter, train_set_t):
    return np.matmul(np.matmul(inv(np.add(np.matmul(basis,basis.trans
pose()),reg_parameter*np.identity(3))),basis), train_set_t)
```

In [8]:
```
#1. Formulate matrix using training data 1
#2. Solve for [w1,w2,w3] using lambda = 0
regularization_parameter = 0
w = calculate_w(basis,regularization_parameter, train_set_t[0])
print(w)
```

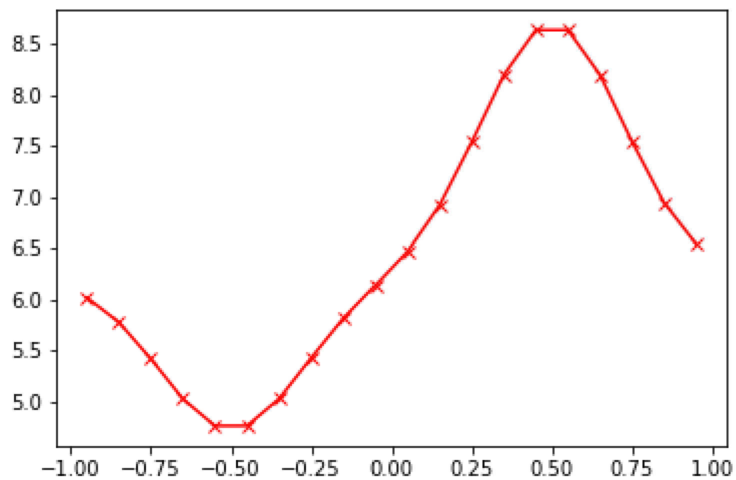
```
[ 6.21319522  2.48369193 -1.49159905]
```

In [9]:
```
#3.plot graphs :)
#x train1 vs t train1
import matplotlib.pyplot as plt

N = len(train_set_x[0])
colors = np.random.rand(N)
area = np.pi * 4**2
plt.scatter(train_set_x[0],train_set_t[0],s=area,c = colors, alpha = 
0.8)
plt.show()

#x train1 vs phiT*xTrain1 * w
y = np.matmul(basis.transpose(), w)
plt.plot(train_set_x[0], y, 'xb-')
plt.show()
```

```
<Figure size 640x480 with 1 Axes>

<Figure size 640x480 with 1 Axes>
```

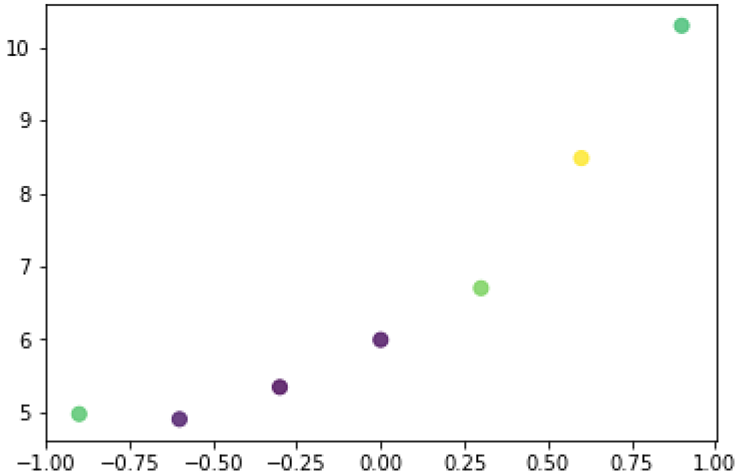
In [10]:
```
#plot x_out vs phiT * x_out * w
x_out = np.array([-0.95, -0.85, -0.75, -0.65, -0.55, -0.45, -0.35, -
0.25, -0.15, -0.05, 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75,
0.85, 0.95 ])
basis = np.array((basis_1(len(x_out)), basis_2(x_out), basis_3(x_out
)))
y = np.matmul(basis.transpose(), w)
plt.plot(x_out, y , 'xr-')
```

Out[10]: [<matplotlib.lines.Line2D at 0x7f332980dac8>]

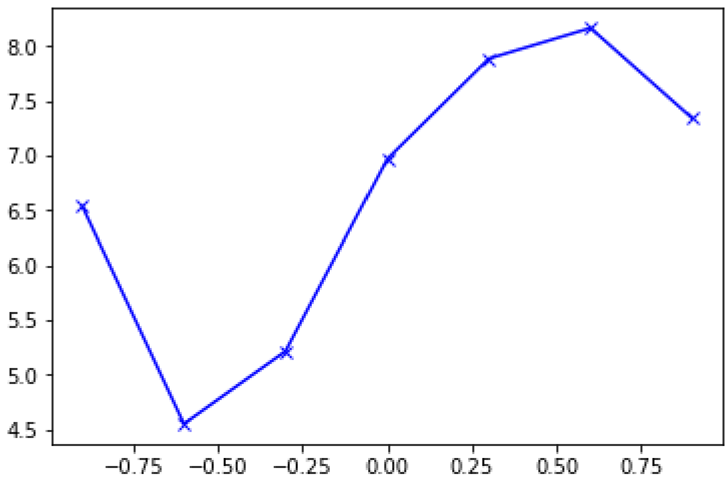


In [11]:
```
def train(train_set_x, train_set_t, regularization_parameter):
    basis = np.array((basis_1(len(train_set_x)),basis_2(train_set_x),
basis_3(train_set_x)))

    w = calculate_w(basis,regularization_parameter, train_set_t)

    N = len(train_set_x)
    colors = np.random.rand(N)
    area = np.pi * 4**2
    plt.scatter(train_set_x,train_set_t,s=area,c = colors, alpha = 0.
8)
    plt.show()

    #x train1 vs phiT*xTrain1 * w
    y = np.matmul(basis.transpose(), w)
    plt.plot(train_set_x, y, 'xb-')
    plt.show()

    basis = np.array((basis_1(len(x_out)), basis_2(x_out), basis_3(x_
out)))
    y = np.matmul(basis.transpose(), w)
    plt.plot(x_out, y , 'xr-')
    plt.show()
```

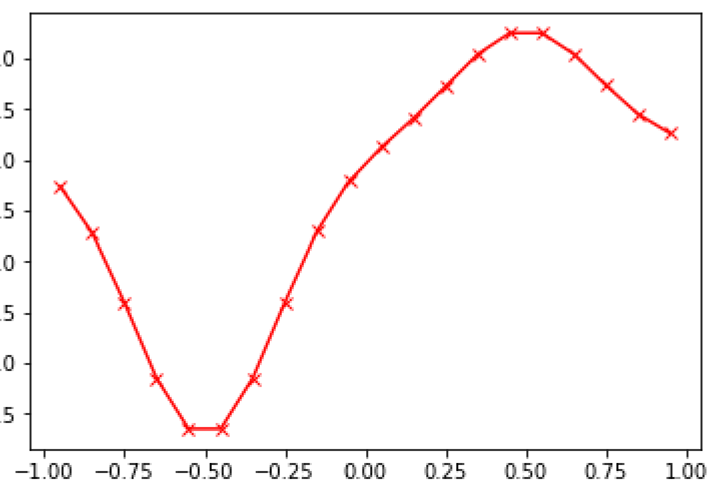
In [12]:
```
#4. calculate for train_2 and train_3
train(train_set_x[1], train_set_t[1], 0)
train(train_set_x[2], train_set_t[2], 0)
```



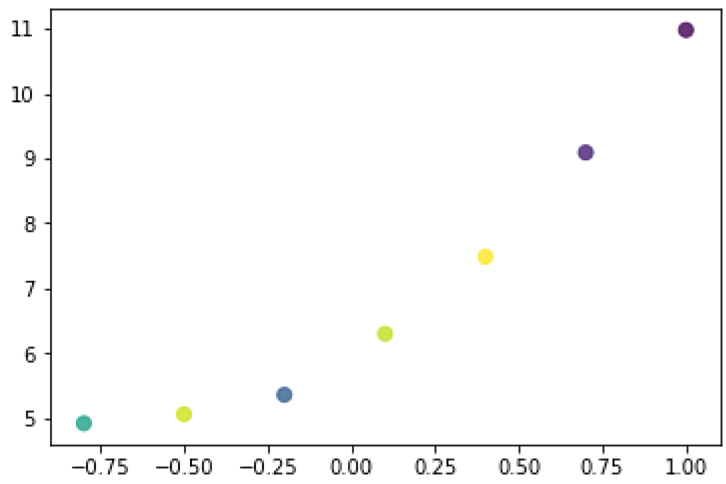
10
9
8
7
6
5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00



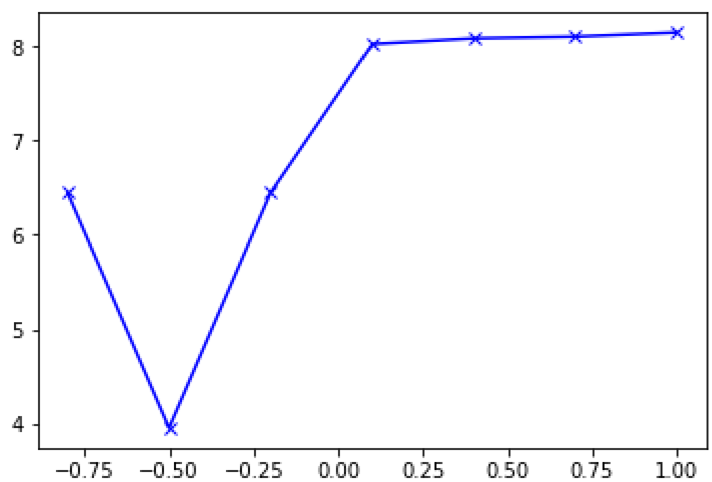
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75



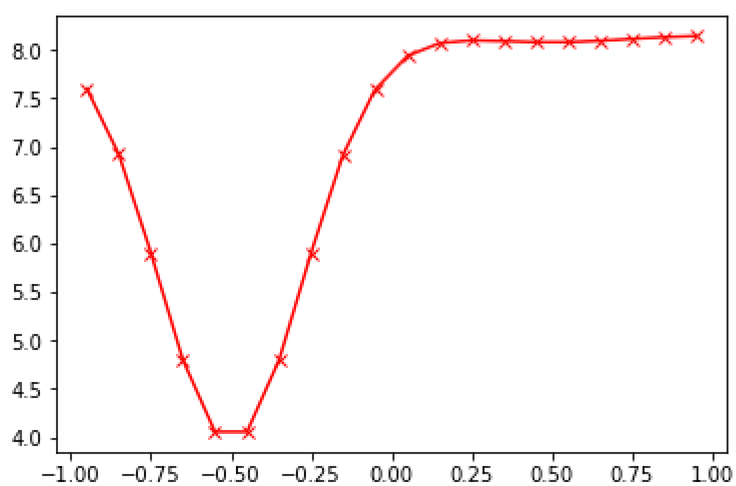
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00



11
10
9
8
7
6
5
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



8
7
6
5
4
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

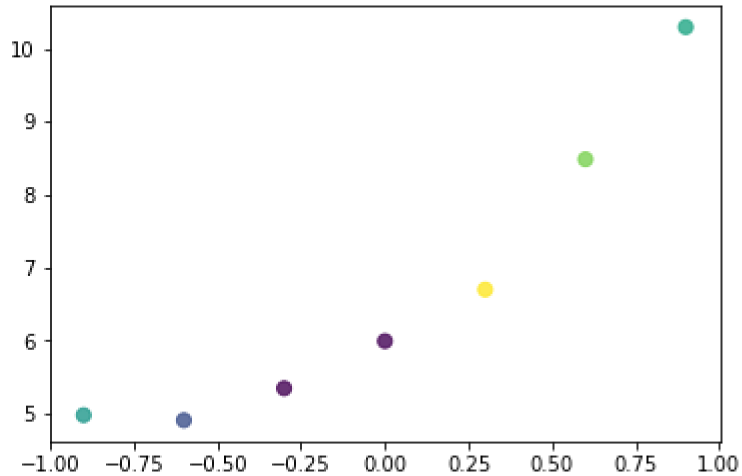


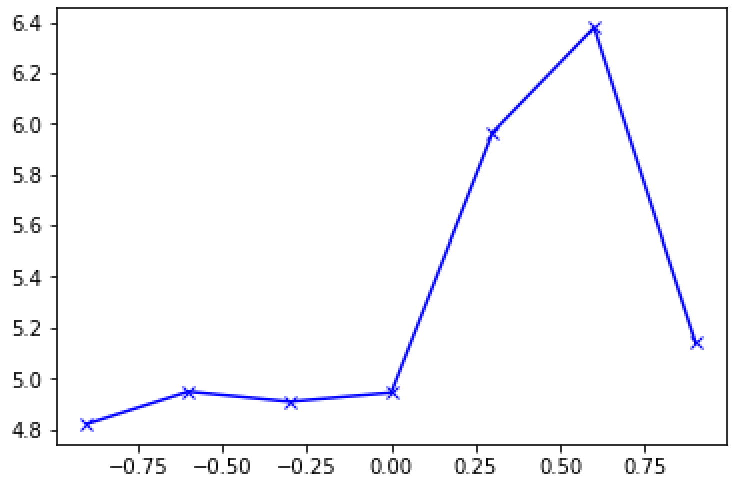
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

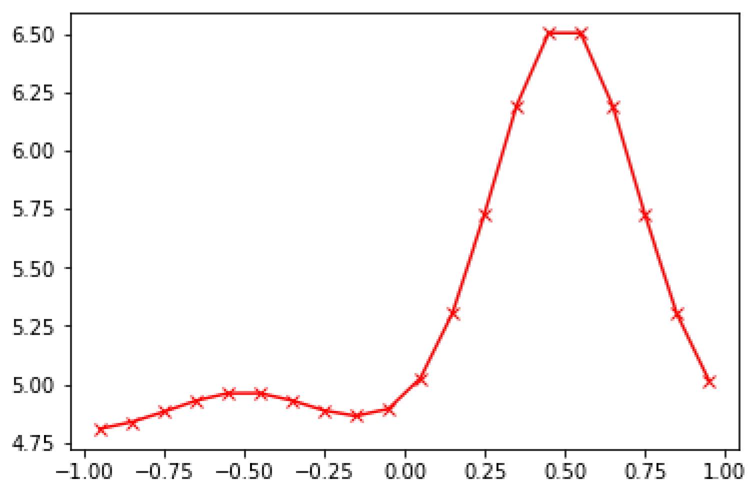
In [13]:
```
#var and bias
x_true = [5, 4.92, 4.88,4.88, 4.92, 5.00, 5.12, 5.28, 5.48, 5.72, 6.0
0, 6.32, 6.68, 7.08,7.52, 8.00, 8.52, 9.08, 9.68, 10.32, 11.00]
true_set_x = np.array((x_true[::3], x_true[1::3], x_true[2::3]))

def train_y(train_set_x, train_set_t, regularization_parameter):
    basis = np.array((basis_1(len(train_set_x)),basis_2(train_set_x),
basis_3(train_set_x)))
    w = calculate_w(basis,regularization_parameter, train_set_t)
    return np.matmul(basis.transpose(), w)


def bias(regularization_parameter):
    avg_prediction_bias = (1/3) * (np.add(np.add(train_y(train_set_x[
0], train_set_t[0], regularization_parameter), train_y(train_set_x[1
], train_set_t[1], 0)), train_y(train_set_x[2], train_set_t[2], regul
arization_parameter)))
    return (1/len(true_set_x[0])) * (np.sum(np.square(np.subtract(avg
_prediction_bias, true_set_x[0]))))

def var(regularization_parameter):
    avg_prediction_bias = (1/3) * (np.add(np.add(train_y(train_set_x[
0], train_set_t[0], regularization_parameter), train_y(train_set_x[1
], train_set_t[1], 0)), train_y(train_set_x[2], train_set_t[2], regul
arization_parameter)))
    return (1/len(true_set_x[0])) * (np.sum((1/3) * (np.add(np.square
(np.subtract(train_y(train_set_x[0], train_set_t[0], regularization_p
arameter),avg_prediction_bias)), np.add(np.square(np.subtract(train_y
(train_set_x[1], train_set_t[1], regularization_parameter),avg_predic
tion_bias)), np.square(np.subtract(train_y(train_set_x[2], train_set_
t[2], regularization_parameter), avg_prediction_bias)))))))

print(bias(0))
print(var(0))
```

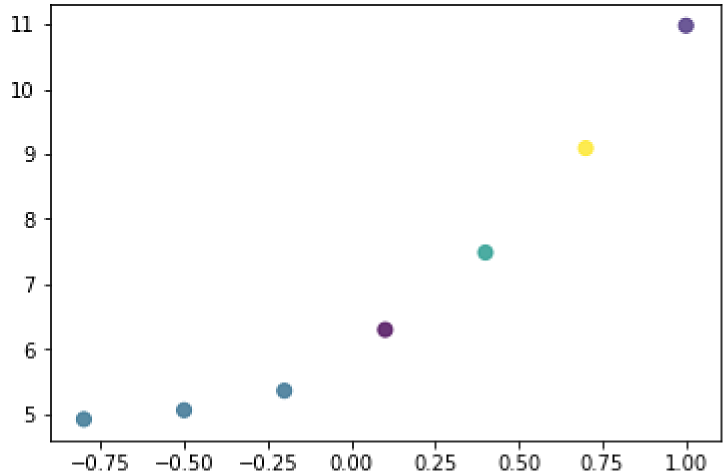
```
1.336553192697737
0.2623341191100613
```

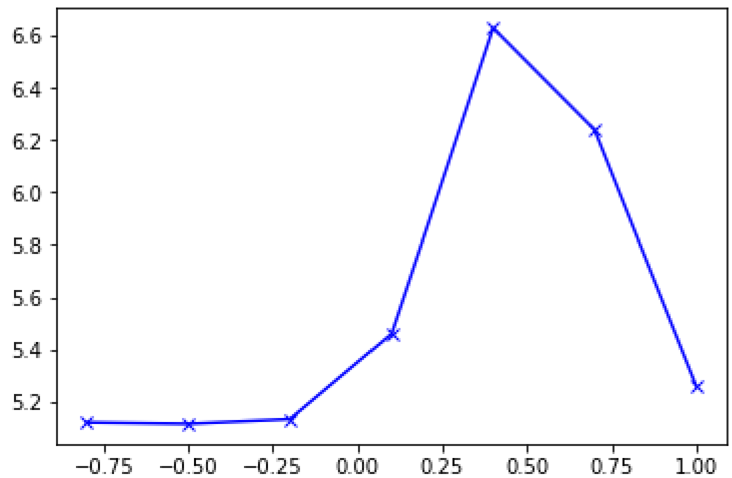
In [14]:
```
#7. plot lambda vs var & bias2
lamb = np.array([0])
variance = np.array([var(0)])
bias2 = np.array([bias(0)])

for i in [2, 4, 5, 6, 10]:
    train(train_set_x[1], train_set_t[1], i)
    train(train_set_x[2], train_set_t[2], i)
    lamb = np.append(lamb, i)
    variance = np.append(variance, var(i))
    bias2 = np.append(bias2, bias(i))

plt.plot(lamb, variance, 'xb-')
plt.plot(lamb, bias2 , 'xr-')
plt.show()
```

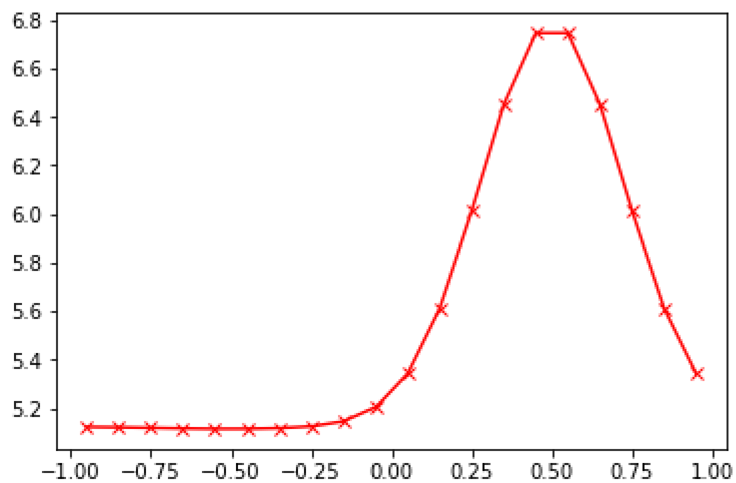

10
9
8
7
6
5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

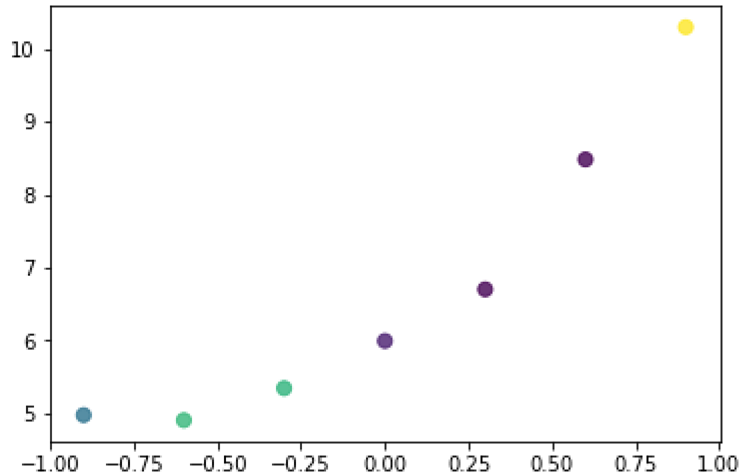

6.4
6.2
6.0
5.8
5.6
5.4
5.2
5.0
4.8
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75

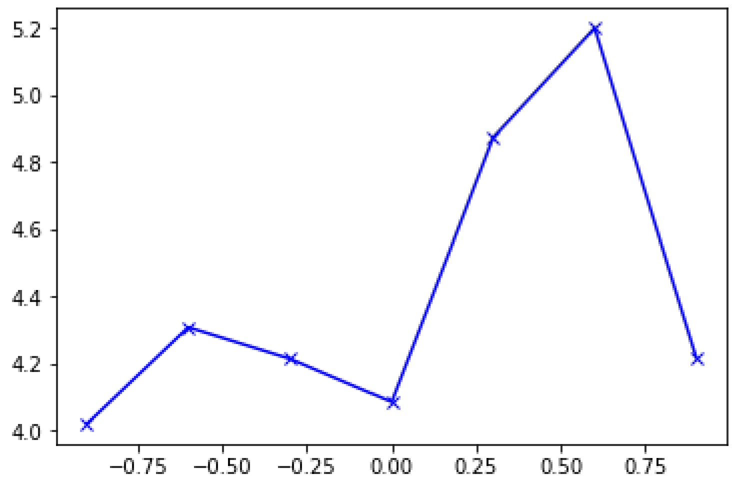

6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

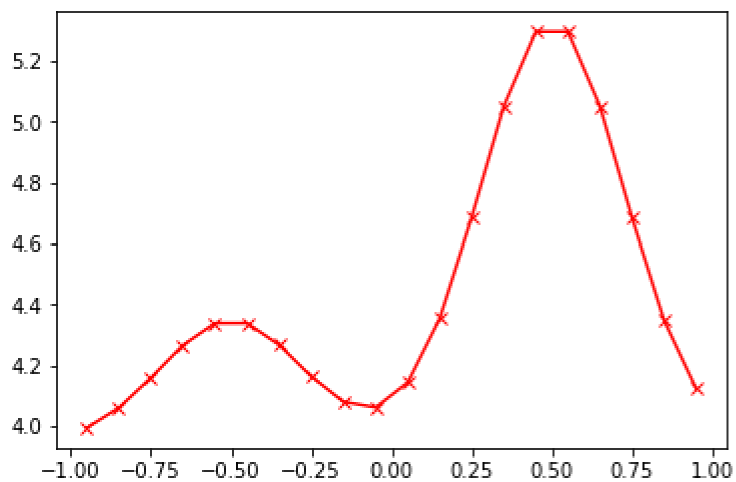

11
10
9
8
7
6
5
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

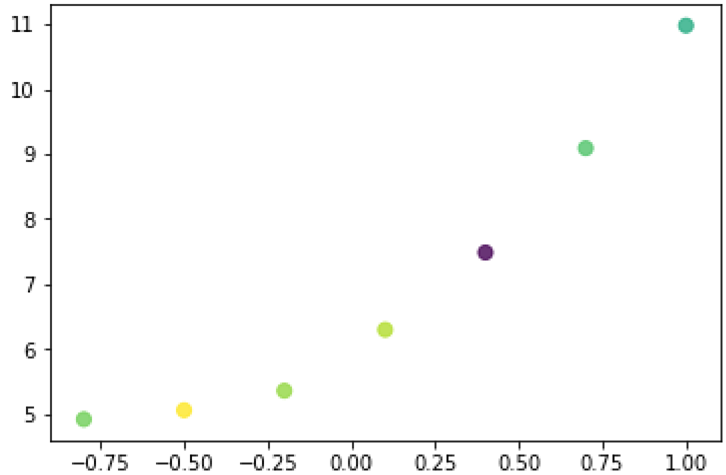

6.6
6.4
6.2
6.0
5.8
5.6
5.4
5.2
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

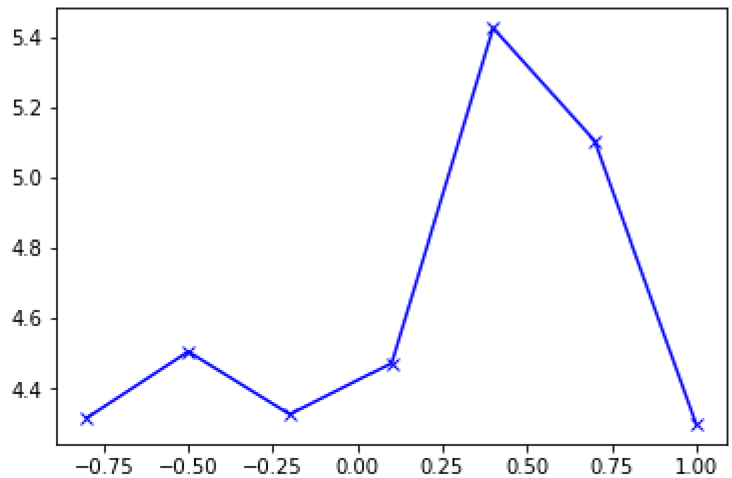

6.8
6.6
6.4
6.2
6.0
5.8
5.6
5.4
5.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

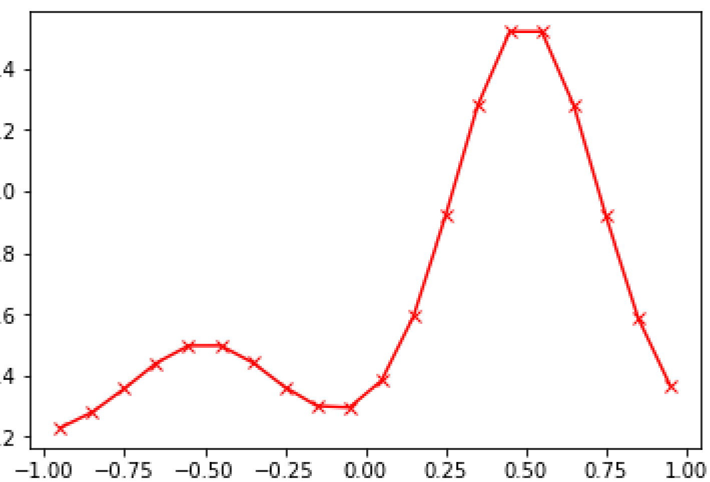

10
9
8
7
6
5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

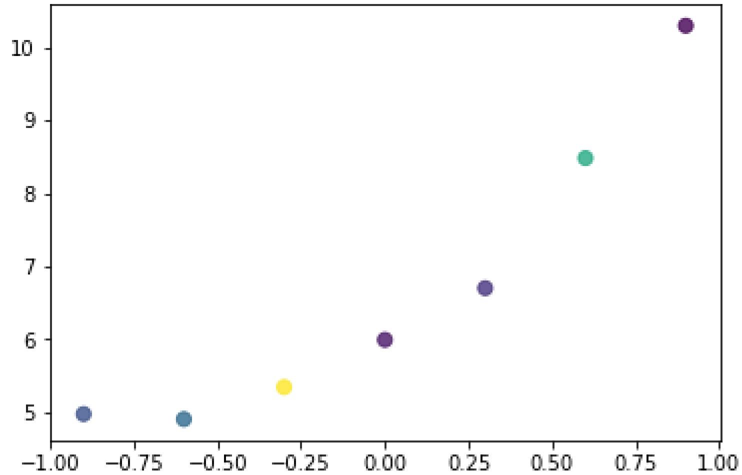

5.2
5.0
4.8
4.6
4.4
4.2
4.0
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75

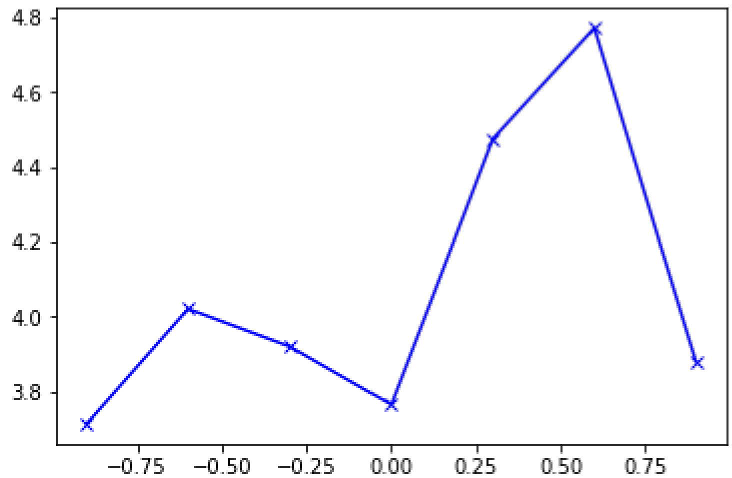

5.2
5.0
4.8
4.6
4.4
4.2
4.0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

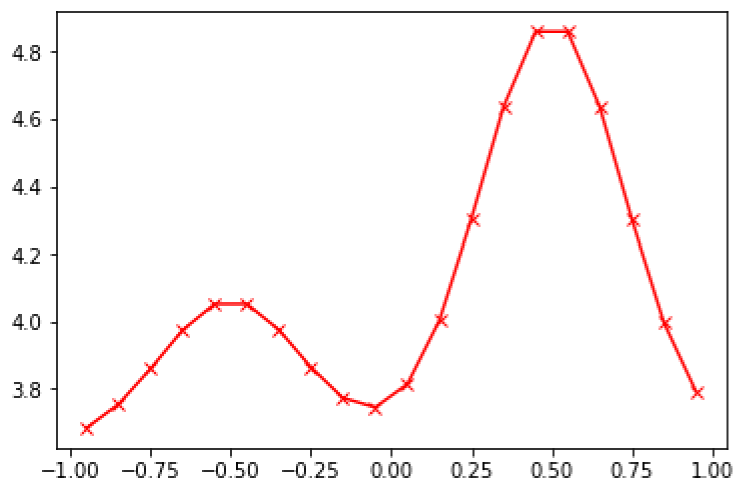

11
10
9
8
7
6
5
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

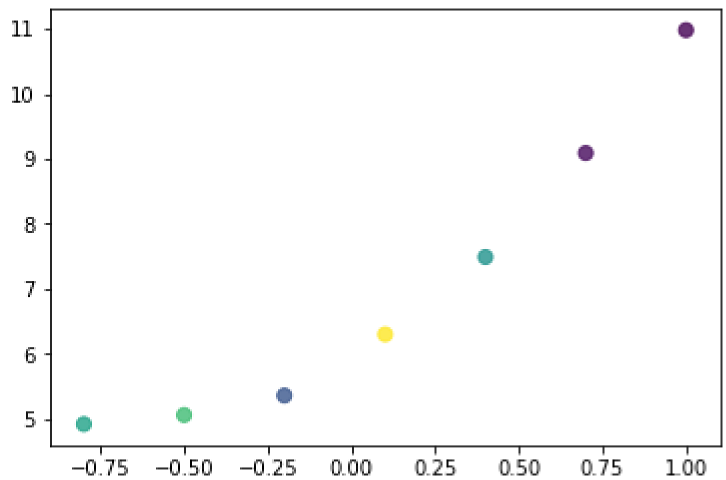

5.4
5.2
5.0
4.8
4.6
4.4
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

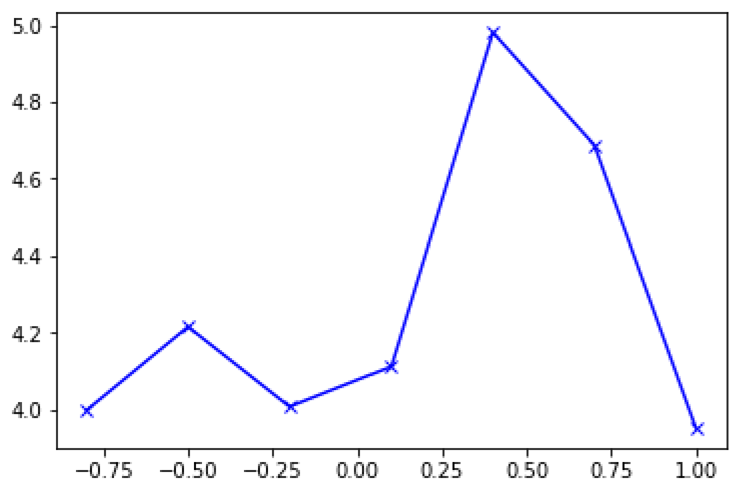

5.4
5.2
5.0
4.8
4.6
4.4
4.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

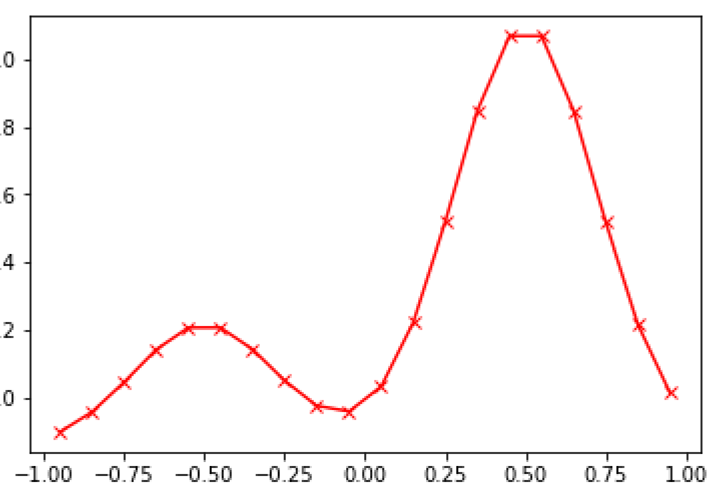

10
9
8
7
6
5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

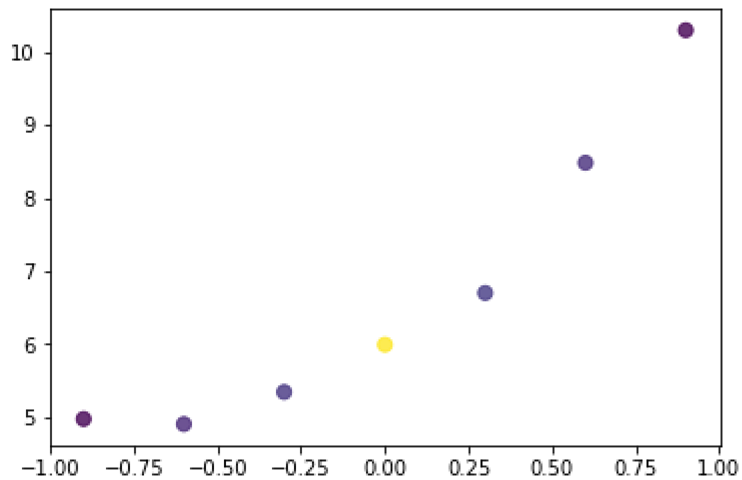

4.8
4.6
4.4
4.2
4.0
3.8
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

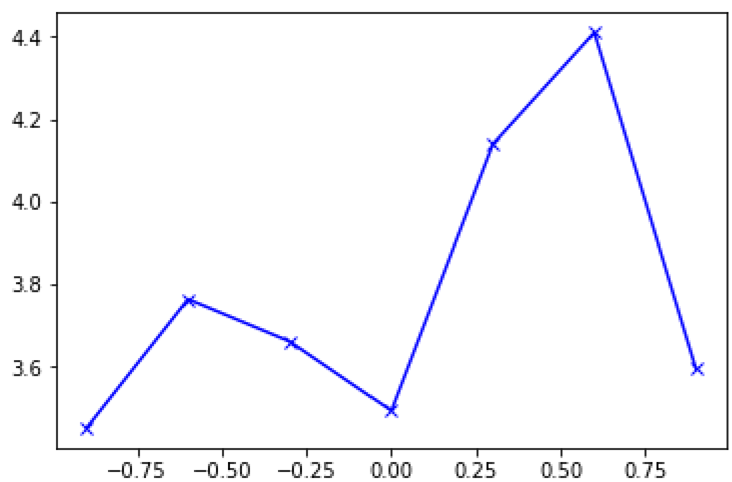

4.8
4.6
4.4
4.2
4.0
3.8
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

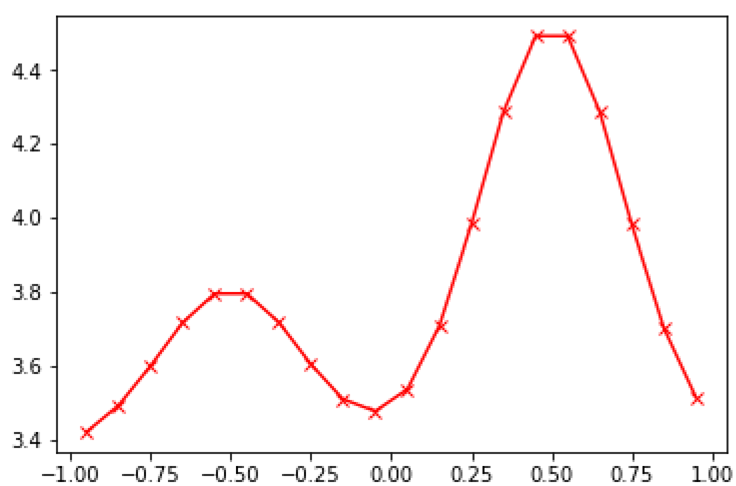

11
10
9
8
7
6
5
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

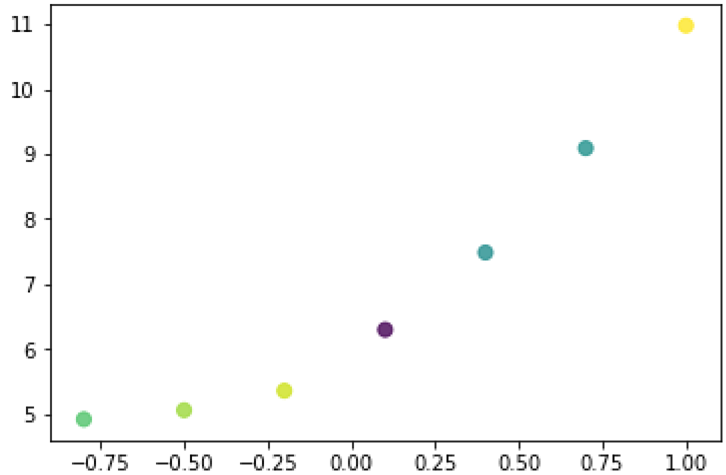

5.0
4.8
4.6
4.4
4.2
4.0
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

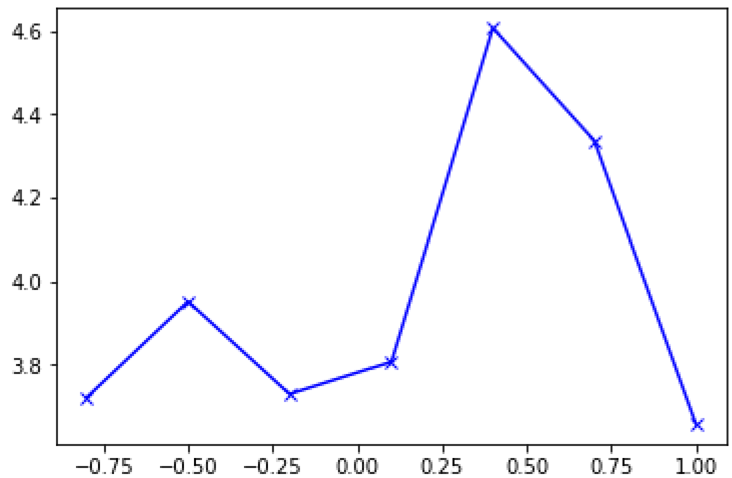

5.0
4.8
4.6
4.4
4.2
4.0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

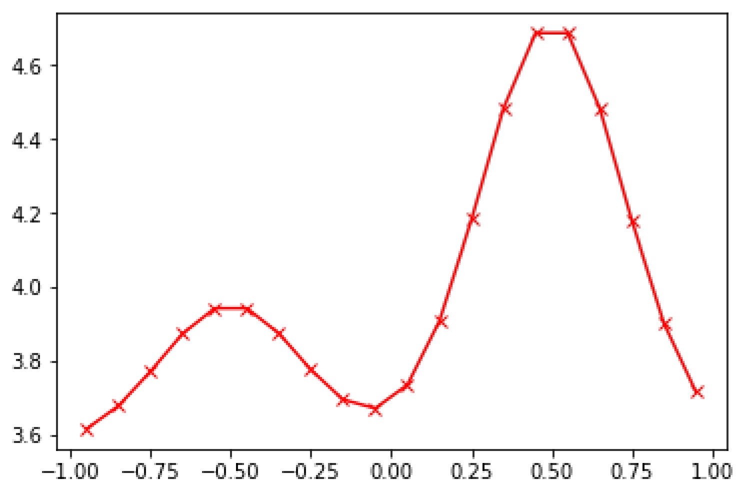

10
9
8
7
6
5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

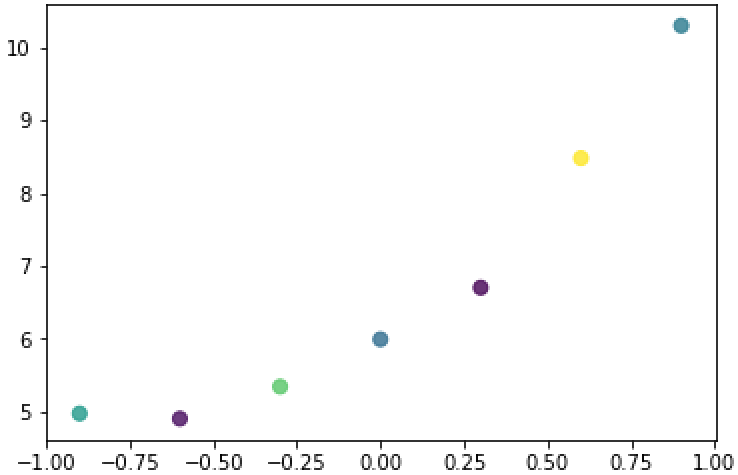

4.4
4.2
4.0
3.8
3.6
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75

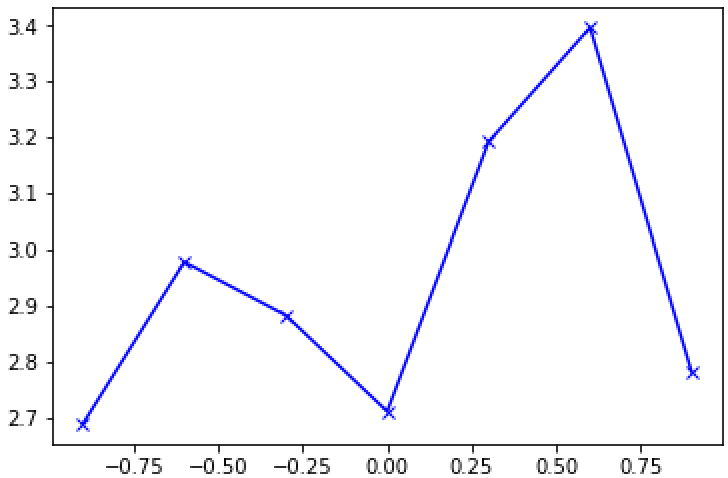

4.4
4.2
4.0
3.8
3.6
3.4
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

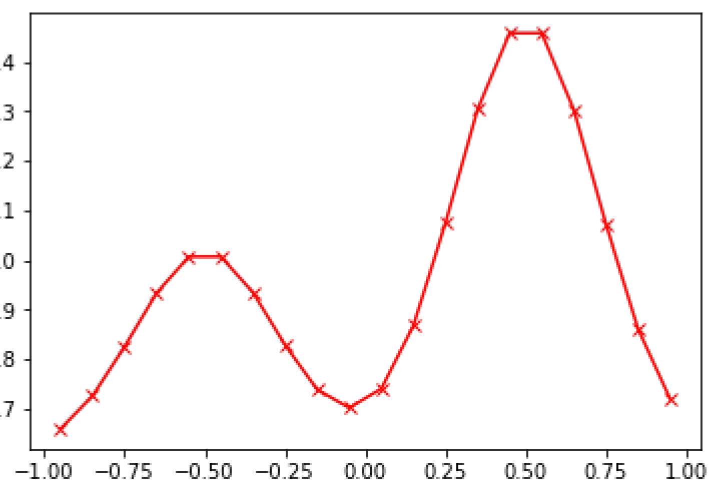

11
10
9
8
7
6
5
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

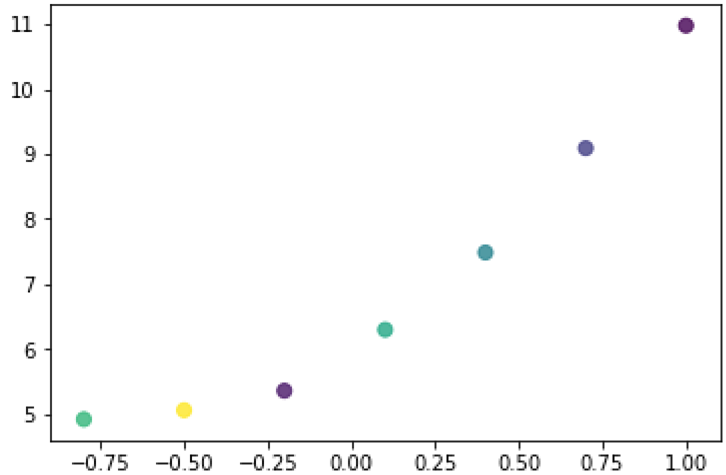

4.6
4.4
4.2
4.0
3.8
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

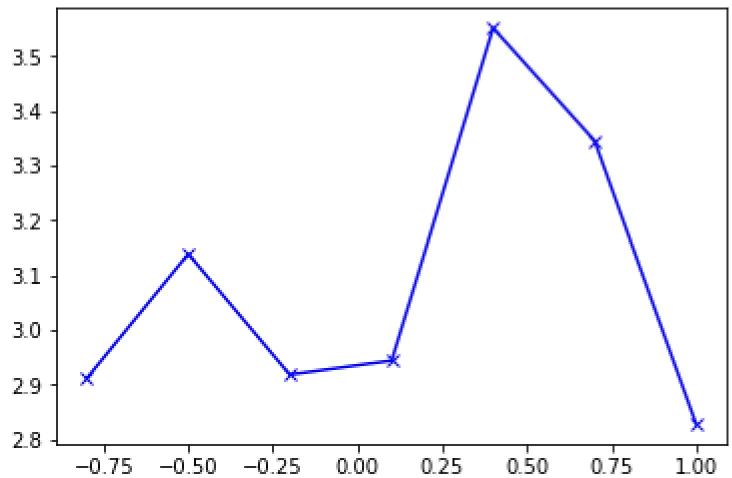

4.6
4.4
4.2
4.0
3.8
3.6
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

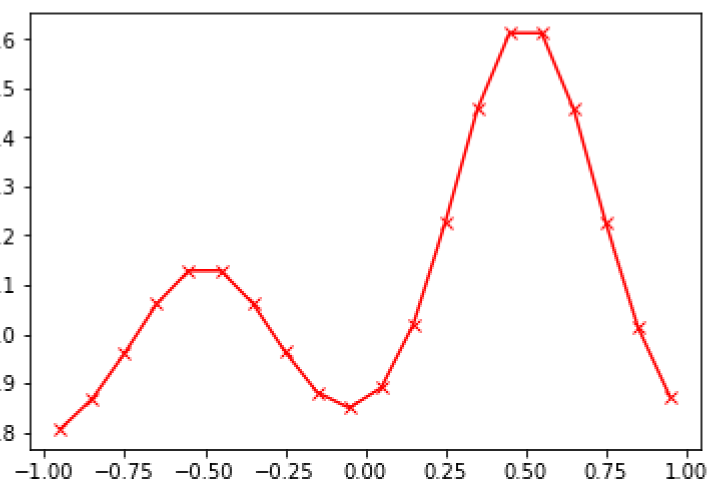

10
9
8
7
6
5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

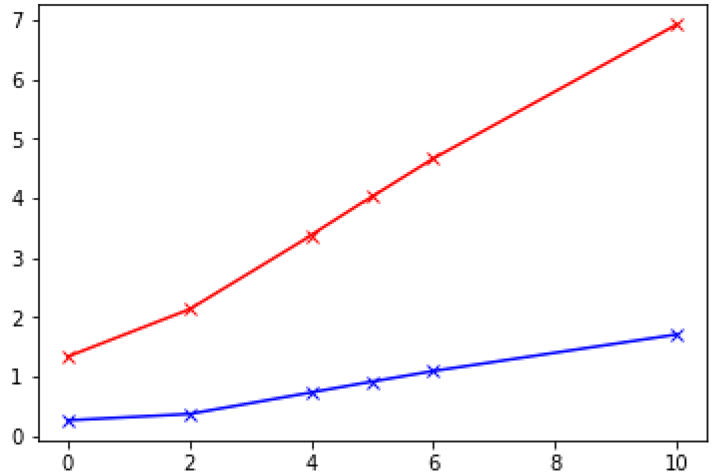

3.4
3.3
3.2
3.1
3.0
2.9
2.8
2.7
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75


3.4
3.3
3.2
3.1
3.0
2.9
2.8
2.7
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


11
10
9
8
7
6
5
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


3.5
3.4
3.3
3.2
3.1
3.0
2.9
2.8
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


3.6
3.5
3.4
3.3
3.2
3.1
3.0
2.9
2.8
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

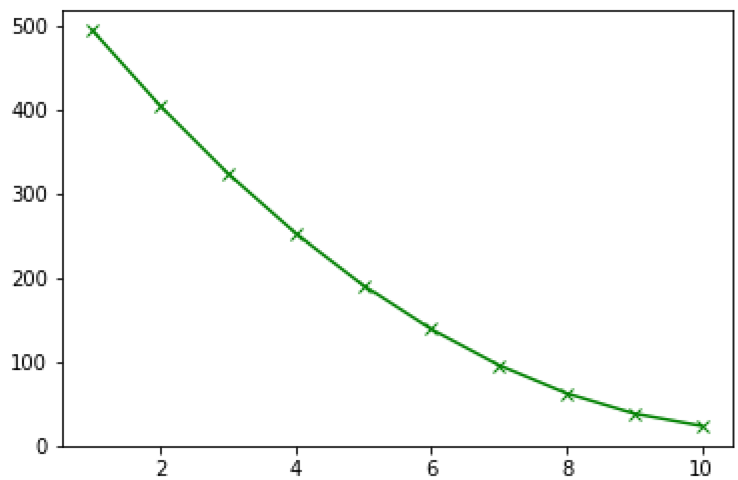


In [15]:
```
# PART-2
```

In [16]:
```
def error(i, w, basis):
    value = t[i] - basis[0]*w[0] - basis[1]*w[1] - basis[2]*w[2]
    return value

def update(i, learning_rate, w):

    basis = np.array((1, basis_2i(i), basis_3i(i)))
    learning_const = error(i, w, basis) * learning_rate
    result = np.add(w, np.multiply(learning_rate,basis))
    return result

def sse_error(y,t):
    return (t-y)**2

def basis_2i(i):
    return np.exp(-((x[i]-0.5)**2/0.1))

def basis_3i(i):
    return np.exp(-((x[i]+0.5)**2/0.1))
```

In [17]:
```
def epoch(learning_rate, w):

    for i in range(15):
        w = update(i, learning_rate, w)
    return w

def find_sse(w):
    basis = np.array((basis_1(7), basis_2(x[14:21]), basis_3(x[14:21
])))
    y = np.matmul(w,basis)
    return np.sum([sse_error(y[i], t[i+14]) for i in range(7)])
```

In [18]:
```
# 1. Initialize w
w = np.zeros(3)
sse = np.array([])
epoch_x = np.array([])
learning_rate = 0.05
for i in range(10):
    w = epoch(learning_rate, w)
    sse = np.append(sse, find_sse(w))
    epoch_x = np.append(epoch_x,i + 1)

plt.plot(epoch_x, sse , 'xg-')
plt.show()
print(w)
```




```
[7.5        1.1512478  2.78407634]
```

In [19]:
```
# PART-3 Kernel trick
```

In [20]:
```
X1 = np.array([ -1, -0.7, -0.4, -0.1, 0.2, 0.5, 0.8 ])
V1 = np.array([ -0.9, -0.6, -0.3, 0, 0.3, 0.6, 0.9 ])
T1 = np.array([ -0.8, -0.5, -0.2, 0.1, 0.4, 0.7, 1])
X0 = np.array([ 5.12, 4.83, 5.29, 5.76, 6.66, 7.92, 9.70 ])
V0 = np.array([ 4.97, 4.90, 5.34, 5.99, 6.70, 8.48, 10.30])
T0 = np.array([ 4.92, 5.06, 5.36, 6.30, 7.49, 9.09, 10.98 ])

def kernel(x1, x2, sig):
    return np.exp(-((x1-x2)**2)/(2*sig))
sig = 0.1
sse = np.array([])

while sig != 0.6:
    V0_est = np.array([])

    for j in range(7):

        M = np.array([kernel(X1[i], V1[j], sig) for i in range(7)])
        print(M)

        sum_M = np.sum(M)

        Norm_M = (1/sum_M) * M

        V0_est = np.append(V0_est, np.matmul(Norm_M.transpose(), X0))

    V0_diff = np.subtract(V0, V0_est)
    sse = np.append(sse, np.matmul(V0_diff.transpose(), V0_diff))

    sig += 0.1
```

```
[9.51229425e-01 8.18730753e-01 2.86504797e-01 4.07622040e-02
 2.35786201e-03 5.54515994e-05 5.30206120e-07]
[4.49328964e-01 9.51229425e-01 8.18730753e-01 2.86504797e-01
 4.07622040e-02 2.35786201e-03 5.54515994e-05]
[0.08629359 0.44932896 0.95122942 0.81873075 0.2865048  0.0407622
 0.00235786]
[0.00673795 0.08629359 0.44932896 0.95122942 0.81873075 0.2865048
 0.0407622 ]
[2.13900415e-04 6.73794700e-03 8.62935865e-02 4.49328964e-01
 9.51229425e-01 8.18730753e-01 2.86504797e-01]
[2.76077257e-06 2.13900415e-04 6.73794700e-03 8.62935865e-02
 4.49328964e-01 9.51229425e-01 8.18730753e-01]
[1.44872049e-08 2.76077257e-06 2.13900415e-04 6.73794700e-03
 8.62935865e-02 4.49328964e-01 9.51229425e-01]
[9.75309912e-01 9.04837418e-01 5.35261429e-01 2.01896518e-01
 4.85578213e-02 7.44658307e-03 7.28152539e-04]
[0.67032005 0.97530991 0.90483742 0.53526143 0.20189652 0.04855782
 0.00744658]
[0.2937577  0.67032005 0.97530991 0.90483742 0.53526143 0.20189652
 0.04855782]
[0.082085   0.2937577  0.67032005 0.97530991 0.90483742 0.53526143
 0.20189652]
[0.01462533 0.082085   0.2937577  0.67032005 0.97530991 0.90483742
 0.53526143]
[0.00166156 0.01462533 0.082085   0.2937577  0.67032005 0.97530991
 0.90483742]
[1.20362805e-04 1.66155727e-03 1.46253347e-02 8.20849986e-02
 2.93757700e-01 6.70320046e-01 9.75309912e-01]
[0.98347145 0.93550699 0.65924063 0.34415379 0.13309839 0.03813333
 0.00809372]
[0.76592834 0.98347145 0.93550699 0.65924063 0.34415379 0.13309839
 0.03813333]
[0.44190221 0.76592834 0.98347145 0.93550699 0.65924063 0.34415379
 0.13309839]
[0.1888756  0.44190221 0.76592834 0.98347145 0.93550699 0.65924063
 0.34415379]
[0.05980496 0.1888756  0.44190221 0.76592834 0.98347145 0.93550699
 0.65924063]
[0.01402847 0.05980496 0.1888756  0.44190221 0.76592834 0.98347145
 0.93550699]
[0.00243778 0.01402847 0.05980496 0.1888756  0.44190221 0.76592834
 0.98347145]
[0.9875778  0.95122942 0.73161563 0.44932896 0.22035839 0.08629359
 0.0269843 ]
[0.81873075 0.9875778  0.95122942 0.73161563 0.44932896 0.22035839
 0.08629359]
[0.54199419 0.81873075 0.9875778  0.95122942 0.73161563 0.44932896
 0.22035839]
[0.2865048  0.54199419 0.81873075 0.9875778  0.95122942 0.73161563
 0.44932896]
[0.12093525 0.2865048  0.54199419 0.81873075 0.9875778  0.95122942
 0.73161563]
[0.0407622  0.12093525 0.2865048  0.54199419 0.81873075 0.9875778
 0.95122942]
[0.010971   0.0407622  0.12093525 0.2865048  0.54199419 0.81873075
 0.9875778 ]
[0.99004983 0.96078944 0.77880078 0.52729242 0.29819728 0.14085842
```

```
 0.05557621]
[0.85214379 0.99004983 0.96078944 0.77880078 0.52729242 0.29819728
 0.14085842]
[0.61262639 0.85214379 0.99004983 0.96078944 0.77880078 0.52729242
 0.29819728]
[0.36787944 0.61262639 0.85214379 0.99004983 0.96078944 0.77880078
 0.52729242]
[0.18451952 0.36787944 0.61262639 0.85214379 0.99004983 0.96078944
 0.77880078]
[0.07730474 0.18451952 0.36787944 0.61262639 0.85214379 0.99004983
 0.96078944]
[0.02705185 0.07730474 0.18451952 0.36787944 0.61262639 0.85214379
 0.99004983]
```

In [21]:

```
min_sse = min(sse)
min_sig = np.where(sse == min_sse)
print(min_sse)
```

```
2.245319628735499
```

In [22]:

```
T0_est = np.array([])

for j in range(7):
    M = np.array([kernel(X1[i], V1[j], 0.1 + min_sig[0]*0.1) for i in range(7)])
    sum_M = np.sum(M)
    Norm_M = (1/sum_M) * M
    T0_est = np.append(T0_est, np.matmul(Norm_M.transpose(), T1))

print(T0)
print(T0_est)
```

```
[ 4.92  5.06  5.36  6.3   7.49  9.09 10.98]
[-0.58228581 -0.37354811 -0.0973657   0.19909468  0.48642671  0.72279721
  0.87086692]
```

In [23]:

```
T0_diff = np.subtract(T0, T0_est)
test_sse = np.matmul(T0_diff.transpose(), T0_diff)
print(test_sse)
```

```
348.0571730501017
```

In [ ]: