Maximum Likelihood Estimation (MLE) and Maximum a Posteriori (MAP)

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
import sys
sys.path.append('/content/drive/MyDrive/smm/homeworks/')
import numpy as np
from utils.optimization import gd, sgd
import matplotlib.pyplot as plt
import scipy

np.random.seed(42)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

Dataset

In [ ]:
```
def Phi(X, K):
    return np.array([X**j for j in range(K)]).T # vandermonde matrix to generate the pol

def createDataset(N, a, b, K, theta_true, variance, random_seed=32):
    X_full = np.linspace(a, b, N)
    Y_full = Phi(X_full, K) @ theta_true + np.random.default_rng(random_seed).normal(0,
    return X_full, Y_full

def train_test_split(X, Y, train_size, random_seed=42):
    idxs = np.arange(0, X.shape[0])
    np.random.default_rng(random_seed).shuffle(idxs)
    X_train = X[idxs[:train_size]]
    Y_train = Y[idxs[:train_size]]
    X_test = X[idxs[train_size:]]
    Y_test = Y[idxs[train_size:]]

    return X_train, Y_train, X_test, Y_test
```

In [ ]:
```
N = 200 # number of samples
K = 3 #degree of the polynomial
a, b = 0, 1
variance = 0.1  #0.1
theta_true = np.ones(K)

X_full, Y_full = createDataset(N, a, b, K, theta_true, variance)
X_train, Y_train, X_test, Y_test = train_test_split(X_full, Y_full, int(0.75*X_full.shap

print(f"Train shape: {X_train.shape}")
print(f"Test shape: {X_test.shape}")
```

```
Train shape: (150,)
Test shape: (50,)
```

Maximum Likelihood Estimation

In [ ]:
```
def loss_mle(theta, Phi_X, Y):
    return (1/2) * np.linalg.norm(Y - (theta @ Phi_X))**2

def grad_loss_mle(theta, Phi_X, Y):
    return -((Y.T @ Phi_X.T) - (theta.T @ Phi_X @ Phi_X.T))
```

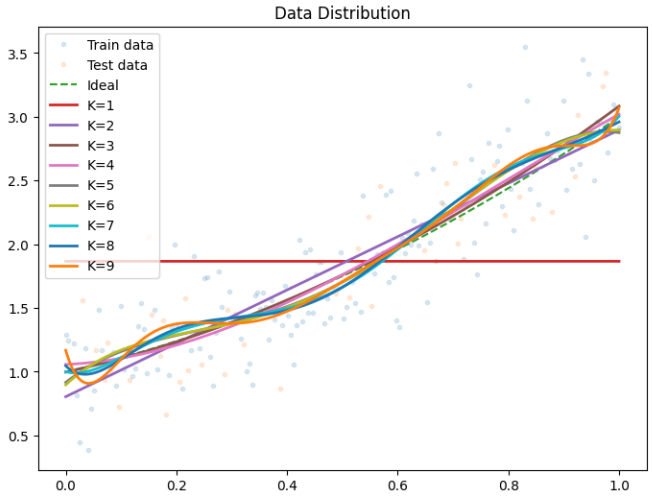
```
def fit_sgd_mle(X, Y, K, batch_size=64, epochs=100, lr=3e-3):
    history_w, history_loss, history_grad, history_err = sgd(loss=loss_mle,grad_loss=gra
        batch_size=batch_size,n_epochs=epochs,lr=lr)
    theta = history_w[-1]
    return theta

def fit_gd_mle(X, Y, K, epochs=100, tol_loss=1e-6, tol_theta=1e-6):
    history_w, _, history_loss, history_grad, history_err = gd(loss=loss_mle,grad_loss=g
        k_max=epochs,tol_loss=tol_loss,tol_w=tol_theta,alpha=None)
    theta = history_w[-1]
    return theta

def fit_neq_mle(X, Y, K):
    Phi_X = Phi(X, K)
    theta = scipy.linalg.cho_solve(scipy.linalg.cho_factor(Phi_X.T @ Phi_X), (Phi_X.T @
    return theta

def predict(X, theta, K):
    return Phi(X, K) @ theta
```

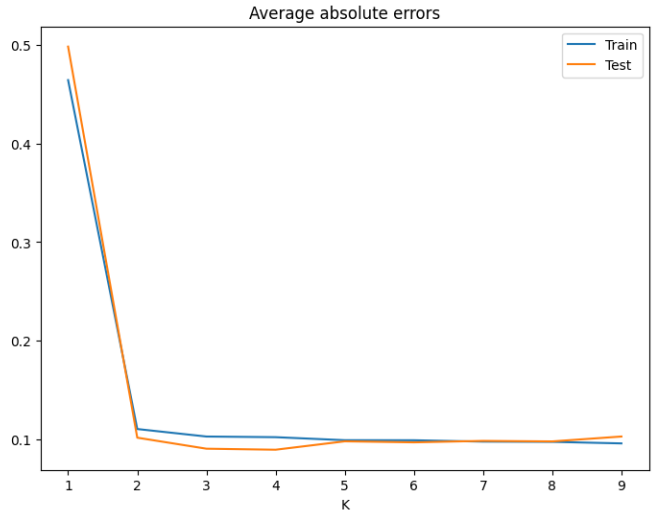
In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

def avgAbsoluteError(X, Y, theta, K):
    Y_pred = predict(X, theta, K)
    #return np.mean(np.abs(Y - Y_pred))
    return (1/Y.shape[0]) * (np.linalg.norm(Y_pred - Y, 2)**2)

def plotDataForVaryingK(fit_function, X_train, Y_train, X_test, Y_test, real_K, K_variat
    to_test_K = [*range(max(1, real_K-K_variation), real_K)] + [real_K] + [*range(real_K
    train_errors = []
    test_errors = []
    plot_X = np.linspace(X_train.min(), X_train.max(), 10000)

    plt.figure(figsize=fig_size)

    plt.subplot(1, 2, 1)
    plt.title("Data Distribution")
    plt.plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
    plt.plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
    if theta_true is not None:
        plt.plot(plot_X, Phi(plot_X, real_K) @ theta_true, "--", label="Ideal")

    for K in to_test_K:
        if lamb is not None: # for MAP
            theta = fit_function(X_train, Y_train, K, lamb)
        else:
            # MLE
            theta = fit_function(X_train, Y_train, K)

        train_errors.append(avgAbsoluteError(X_train, Y_train, theta, K))
        test_errors.append(avgAbsoluteError(X_test, Y_test, theta, K))
        print(f"K={K}: Average absolute error: [test] = {test_errors[-1]:<20}, [train] =
        plt.plot(plot_X, predict(plot_X, theta, K), "-", label=f"K={K}", linewidth=2)

    plt.legend()
    plt.subplot(1, 2, 2)
    plt.title("Average absolute errors")
    plt.plot(to_test_K, train_errors, label="Train")
    plt.plot(to_test_K, test_errors, label="Test")
    plt.xlabel("K")
    plt.xticks(to_test_K)
    plt.legend()
    plt.show()
```

In [ ]:
```
plotDataForVaryingK(fit_function=fit_neq_mle, X_train=X_train, Y_train=Y_train, X_test=X
                    real_K=3, K_variation=6, theta_true=theta_true)
```

```
K=1: Average absolute error: [test] = 0.49851805037227254 , [train] = 0.4645274673506662
  , average value of the parameters  = 1.8665193014009778
K=2: Average absolute error: [test] = 0.10137056473607894 , [train] = 0.1101598202222929
8 , average value of the parameters  = 1.448528887479862
K=3: Average absolute error: [test] = 0.09018194965405466 , [train] = 0.1025409285413384
8 , average value of the parameters  = 1.0278844352264274
K=4: Average absolute error: [test] = 0.08915740412479177 , [train] = 0.1018830211473290
8 , average value of the parameters  = 0.7552314378655105
K=5: Average absolute error: [test] = 0.09767220470039455 , [train] = 0.0988269157156878
7 , average value of the parameters  = 0.5751322757777466
K=6: Average absolute error: [test] = 0.09664638479928986 , [train] = 0.0987724139423146
3 , average value of the parameters  = 0.48273278747669685
K=7: Average absolute error: [test] = 0.09811612871225817 , [train] = 0.0973835708959833
3 , average value of the parameters  = 0.42921242521431474
K=8: Average absolute error: [test] = 0.09759133059816638 , [train] = 0.0971428773121814
1 , average value of the parameters  = 0.37000247201720526
K=9: Average absolute error: [test] = 0.10255457615090098 , [train] = 0.0955711401771038
6 , average value of the parameters  = 0.34129575921901456
```





Maximum a Posteriori

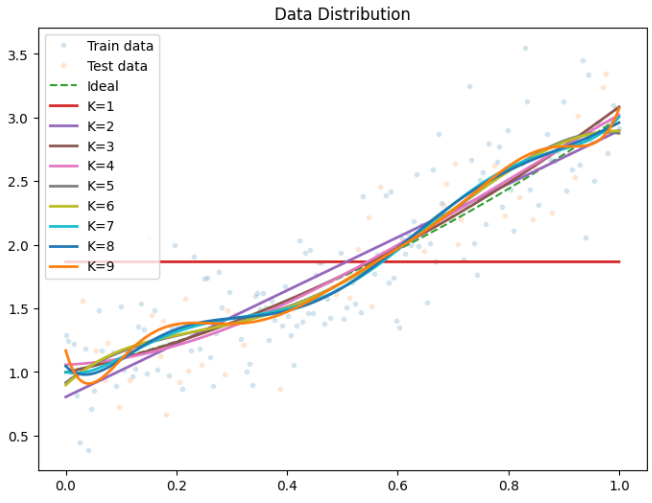
In [ ]:
```
import numpy as np
import scipy.linalg

def loss_map(theta, Phi_X, Y, lamb):
    return (1/2)*(np.linalg.norm(Y - (theta @ Phi_X))**2) + (1/2)*(lamb * np.linalg.norm

def grad_loss_map(theta, Phi_X, Y, lamb):
    return -((Y.T @ Phi_X.T) - (theta.T @ Phi_X @ Phi_X.T)) + (lamb * theta)

def fit_sgd_map(X, Y, K, lamb, batch_size=64, epochs=100, lr=3e-3):
    loss_with_lamb = lambda theta, Phi_X, Y: loss_map(theta, Phi_X, Y, lamb) #wrap -> cr
    grad_loss_with_lamb = lambda theta, Phi_X, Y: grad_loss_map(theta, Phi_X, Y, lamb)
    history_w, history_loss, history_grad, history_err = sgd(loss=loss_with_lamb,grad_lo
        data=(Phi(X, K).T, Y),batch_size=batch_size,n_epochs=epochs,lr=lr)
    theta = history_w[-1]
    return theta

def fit_gd_map(X, Y, K, lamb, epochs=100, tol_loss=1e-6, tol_theta=1e-6):
    loss_with_lamb = lambda theta, Phi_X, Y: loss_map(theta, Phi_X, Y, lamb)
    grad_loss_with_lamb = lambda theta, Phi_X, Y: grad_loss_map(theta, Phi_X, Y, lamb)
    history_w, _, history_loss, history_grad, history_err = gd(loss=loss_with_lamb,grad_
        k_max=epochs,tol_loss=tol_loss,tol_w=tol_theta,alpha=None)
    theta = history_w[-1]
```

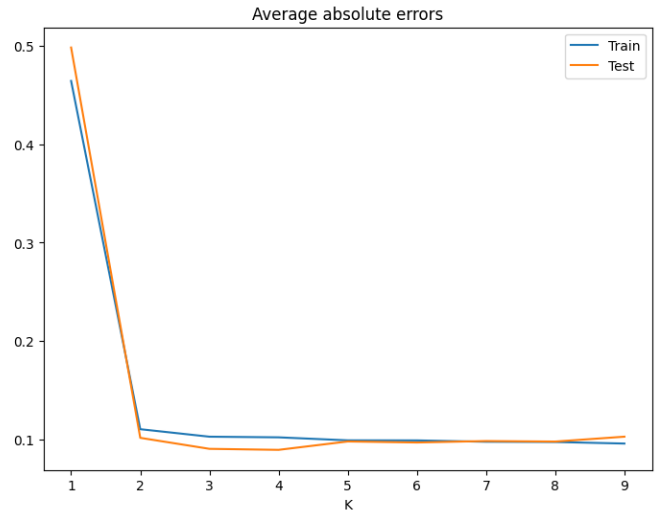
```
    return theta

def fit_neq_map(X, Y, K, lamb):
    Phi_X = Phi(X, K)
    theta = scipy.linalg.cho_solve(
        scipy.linalg.cho_factor((Phi_X.T @ Phi_X) + (lamb * np.identity(Phi_X.shape[1]))
        (Phi_X.T @ Y))
    return theta
```

Fixed λ, varying K

In [ ]:
```
for to_try_lambda in np.linspace(0, 1, 5):
    print(f">>>>> lambda = {to_try_lambda} <<<<<")
    plotDataForVaryingK(fit_function=fit_neq_map, X_train=X_train, Y_train=Y_train, X_te
                        real_K=3, K_variation=6, theta_true=theta_true, lamb = to_try_lambda
```

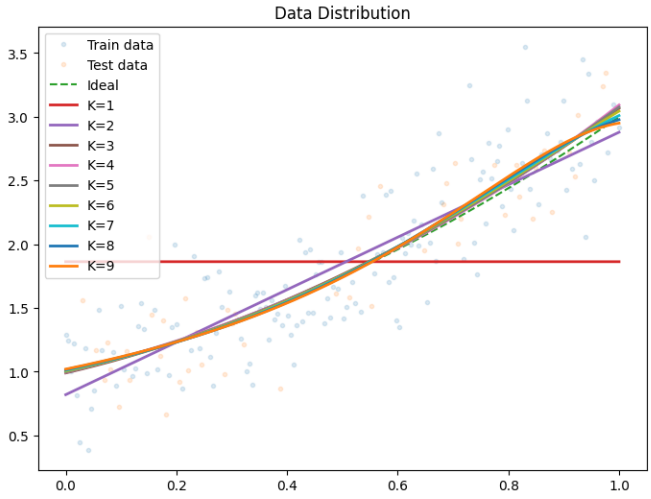
```
>>>>> lambda = 0.0 <<<<<
K=1: Average absolute error: [test] = 0.49851805037227254 , [train] = 0.4645274673506662
  , average value of the parameters  = 1.8665193014009778
K=2: Average absolute error: [test] = 0.10137056473607894 , [train] = 0.1101598202222929
8 , average value of the parameters  = 1.448528887479862
K=3: Average absolute error: [test] = 0.09018194965405466 , [train] = 0.1025409285413384
8 , average value of the parameters  = 1.0278844352264274
K=4: Average absolute error: [test] = 0.08915740412479177 , [train] = 0.1018830211473290
8 , average value of the parameters  = 0.7552314378655105
K=5: Average absolute error: [test] = 0.09767220470039455 , [train] = 0.0988269157156878
7 , average value of the parameters  = 0.5751322757777466
K=6: Average absolute error: [test] = 0.09664638479928986 , [train] = 0.0987724139423146
3 , average value of the parameters  = 0.48273278747669685
K=7: Average absolute error: [test] = 0.09811612871225817 , [train] = 0.0973835708959833
3 , average value of the parameters  = 0.42921242521431474
K=8: Average absolute error: [test] = 0.09759133059816638 , [train] = 0.0971428773121814
1 , average value of the parameters  = 0.37000247201720526
K=9: Average absolute error: [test] = 0.10255457615090098 , [train] = 0.0955711401771038
6 , average value of the parameters  = 0.34129575921901456
```

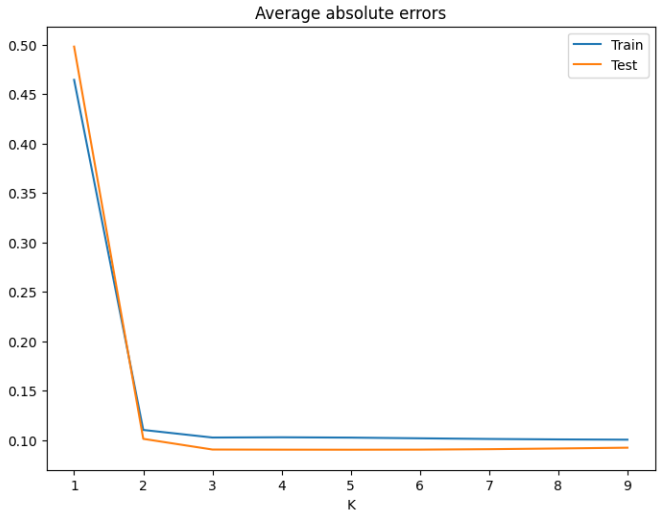




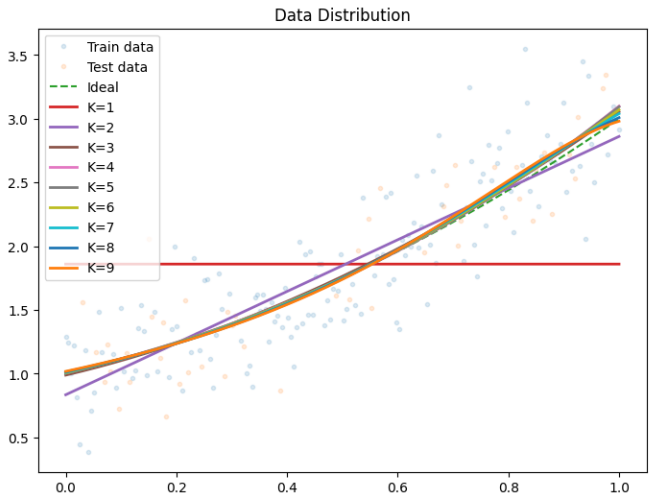

```
>>>>> lambda = 0.25 <<<<<
K=1: Average absolute error: [test] = 0.49815771359432853 , [train] = 0.4645371126570262
4 , average value of the parameters  = 1.8634136120475648
K=2: Average absolute error: [test] = 0.10124023944682176 , [train] = 0.1102543258202042
  , average value of the parameters  = 1.4395144111333493
K=3: Average absolute error: [test] = 0.09042908317792782 , [train] = 0.1025735448084980
2 , average value of the parameters  = 1.0223466555187801
K=4: Average absolute error: [test] = 0.09030161102679106 , [train] = 0.1028259299187502
  , average value of the parameters  = 0.7732374060180482
K=5: Average absolute error: [test] = 0.09024649303026522 , [train] = 0.1024899084233987
8 , average value of the parameters  = 0.6153280279885378
K=6: Average absolute error: [test] = 0.09033682593039039 , [train] = 0.1017906530737209
5 , average value of the parameters  = 0.5074706650782551
```

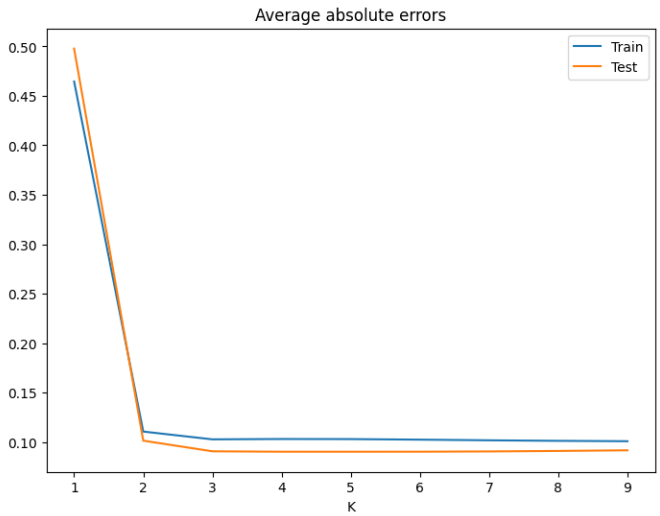
```
K=7: Average absolute error: [test] = 0.09078480943712526 , [train] = 0.1011254507970116
2 , average value of the parameters  = 0.4299711913371497
K=8: Average absolute error: [test] = 0.0914974931992259  , [train] = 0.1006582381125171
3 , average value of the parameters  = 0.37218560118514926
K=9: Average absolute error: [test] = 0.09229019226811198 , [train] = 0.1003924388258496
1 , average value of the parameters  = 0.3278008398696274
```





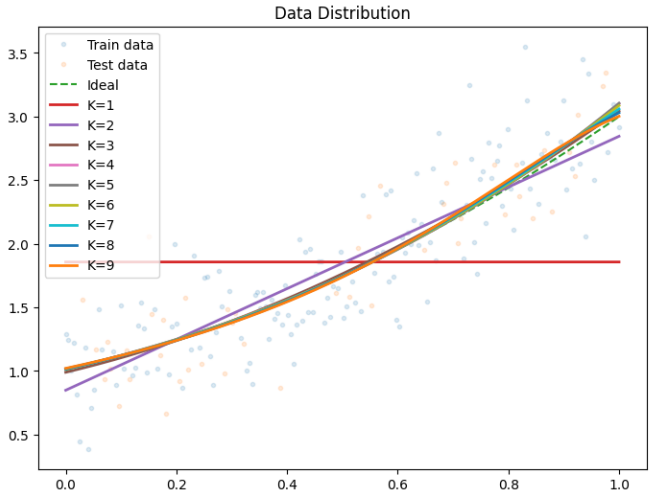

```
>>>>> lambda = 0.5 <<<<<
K=1: Average absolute error: [test] = 0.4978177685350947  , [train] = 0.4645659205057364
  , average value of the parameters  = 1.8603182405989809
K=2: Average absolute error: [test] = 0.10133157827945739  , [train] = 0.1105203900591205
1 , average value of the parameters  = 1.430878499200452
K=3: Average absolute error: [test] = 0.09056991871237015 , [train] = 0.1026392006326679
9 , average value of the parameters  = 1.0182230364826068
K=4: Average absolute error: [test] = 0.09015976125340586 , [train] = 0.1030024743320924
9 , average value of the parameters  = 0.7748080654104131
K=5: Average absolute error: [test] = 0.09015113018719699 , [train] = 0.1029038122967807
2 , average value of the parameters  = 0.6188919362376865
K=6: Average absolute error: [test] = 0.09017588646925143 , [train] = 0.1023487221878850
9 , average value of the parameters  = 0.5117677493233416
K=7: Average absolute error: [test] = 0.09043150154630285 , [train] = 0.1016933868021277
6 , average value of the parameters  = 0.434300568356818
K=8: Average absolute error: [test] = 0.09093185701993627 , [train] = 0.1011460049405708
3 , average value of the parameters  = 0.37614877663518315
K=9: Average absolute error: [test] = 0.09157594126956704 , [train] = 0.1007700749720478
  , average value of the parameters  = 0.33121444638920106
```

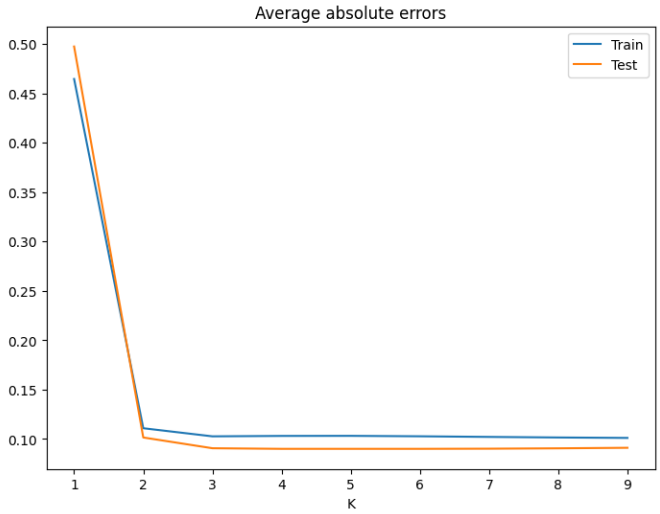




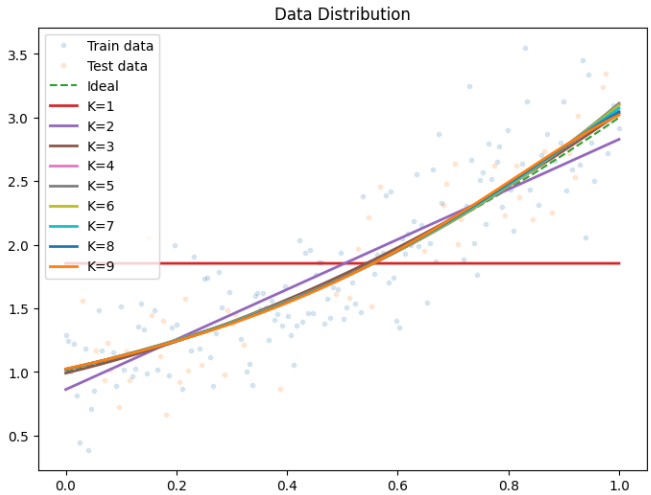

```
>>>>> lambda = 0.75 <<<<<
K=1: Average absolute error: [test] = 0.49749801840614605 , [train] = 0.4646137002236777
  , average value of the parameters  = 1.8572331357223655
K=2: Average absolute error: [test] = 0.10156916441527744  , [train] = 0.1109346570769019
4 , average value of the parameters  = 1.4225937733719696
K=3: Average absolute error: [test] = 0.09068565027328397 , [train] = 0.1027289247733510
9 , average value of the parameters  = 1.0146523486234138
```

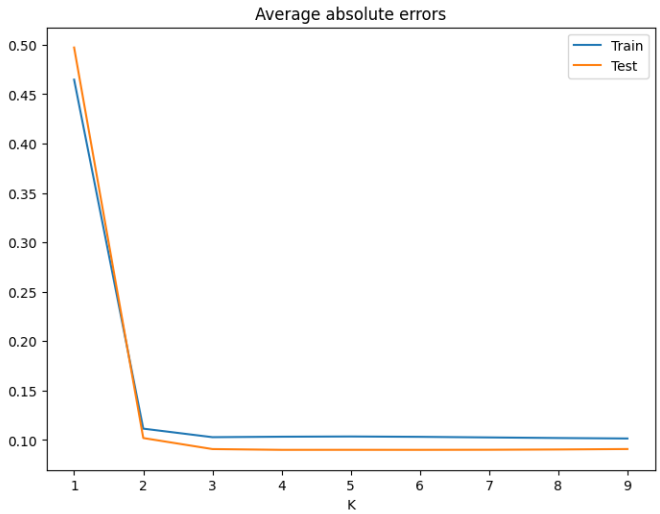
K=4: Average absolute error: [test] = 0.09006960460159738 , [train] = 0.1031606075044467
1 , average value of the parameters = 0.7753819500519665
K=5: Average absolute error: [test] = 0.09007539348014985 , [train] = 0.1032359469530823
2 , average value of the parameters = 0.6210079889519762
K=6: Average absolute error: [test] = 0.09007005685678646 , [train] = 0.1027845521126252
  , average value of the parameters = 0.5144445397710808
K=7: Average absolute error: [test] = 0.09022814767538595 , [train] = 0.1021566392217722
9 , average value of the parameters = 0.4370942519378633
K=8: Average absolute error: [test] = 0.09060372116963902 , [train] = 0.1015779908187840
9 , average value of the parameters = 0.3788166429138548
K=9: Average absolute error: [test] = 0.09113893365163028 , [train] = 0.1011400333684181
2 , average value of the parameters = 0.3336257857908883





>>>>> lambda = 1.0 <<<<<
K=1: Average absolute error: [test] = 0.49719826835261755 , [train] = 0.4646802630307901
3 , average value of the parameters = 1.854158246424812
K=2: Average absolute error: [test] = 0.10197629310501594 , [train] = 0.1114770636538352
5 , average value of the parameters = 1.4146354568499708
K=3: Average absolute error: [test] = 0.09080233088007647 , [train] = 0.1028422177655987
2 , average value of the parameters = 1.0113718201321118
K=4: Average absolute error: [test] = 0.09002415185067461 , [train] = 0.1033139155985105
  , average value of the parameters = 0.7754432852512121
K=5: Average absolute error: [test] = 0.09004404669164233 , [train] = 0.1035472615033905
8 , average value of the parameters = 0.6224475068901338
K=6: Average absolute error: [test] = 0.09002235347768238 , [train] = 0.1031899472608911
6 , average value of the parameters = 0.5164082624526461
K=7: Average absolute error: [test] = 0.09011391729662917 , [train] = 0.1025927357445154
3 , average value of the parameters = 0.4392051658864543
K=8: Average absolute error: [test] = 0.09039809534960698 , [train] = 0.1019985475371953
1 , average value of the parameters = 0.38088203204655324
K=9: Average absolute error: [test] = 0.09084543907543373 , [train] = 0.1015182554230396
6 , average value of the parameters = 0.3355417803052814



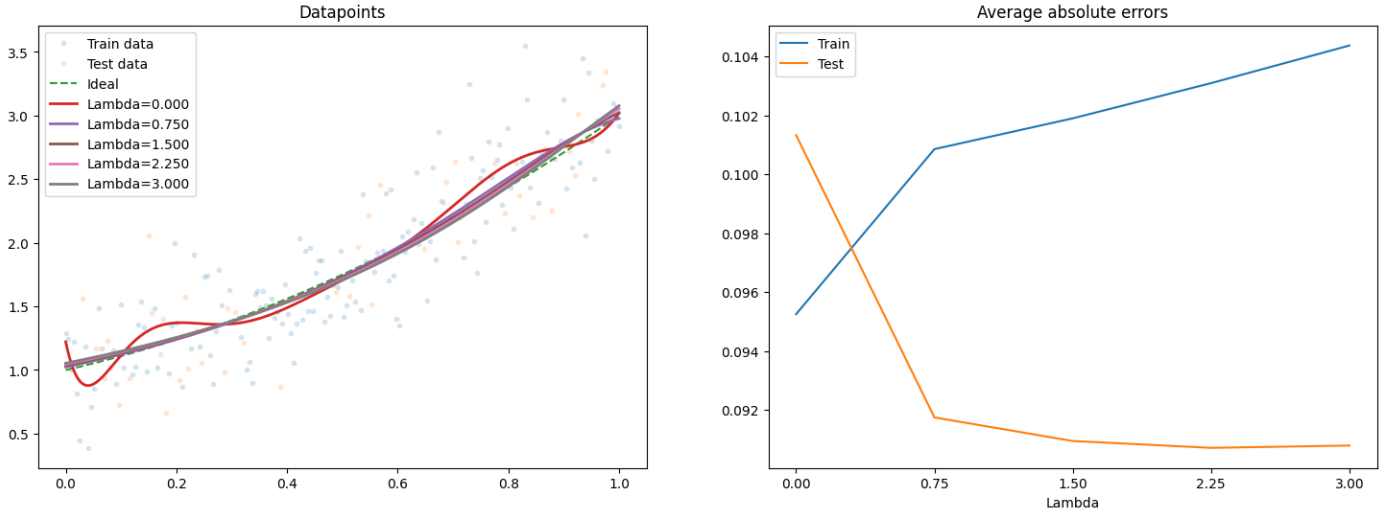


Fixed K, varying λ

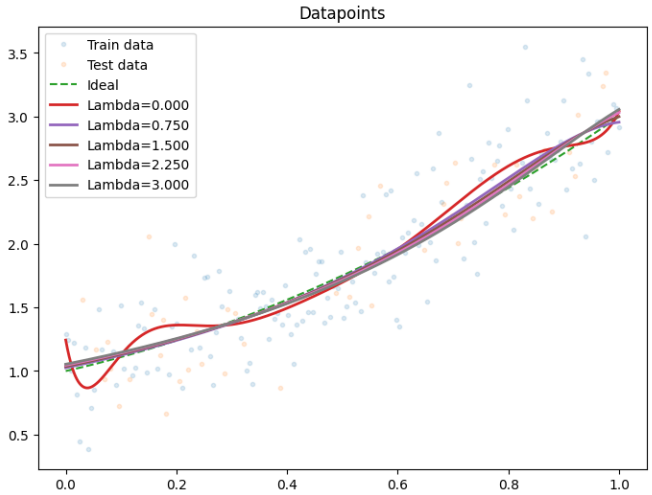
In [ ]:
```
def plotDataForVaryingLambda(fit_function, K, X_train, Y_train, X_test, Y_test, to_try_l
    train_errors = []
    test_errors = []
    plot_X = np.linspace(X_train.min(), X_train.max(), 10000)

    plt.figure(figsize=fig_size)

    plt.subplot(1, 2, 1)
    plt.title("Datapoints")
    plt.plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
    plt.plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
    plt.plot(plot_X, Phi(plot_X, real_K) @ theta_true, "--", label="Ideal")

    for lamb in to_try_lambda:
        theta = fit_function(X_train, Y_train, K, lamb)
        train_errors.append(avgAbsoluteError(X_train, Y_train, theta, K))
        test_errors.append(avgAbsoluteError(X_test, Y_test, theta, K))

        print(f"Lambda={lamb:.3f} | Avg absolute error: [test] {test_errors[-1]:<20} | [
        plt.plot(plot_X, predict(plot_X, theta, K), "-", label=f"Lambda={lamb:.3f}", lin

    plt.legend()
    plt.subplot(1, 2, 2)
    plt.title("Average absolute errors")
    plt.plot(to_try_lambda, train_errors, label="Train")
    plt.plot(to_try_lambda, test_errors, label="Test")
    plt.xlabel("Lambda")
    plt.xticks(to_try_lambda)
    plt.legend()
    plt.show()
```

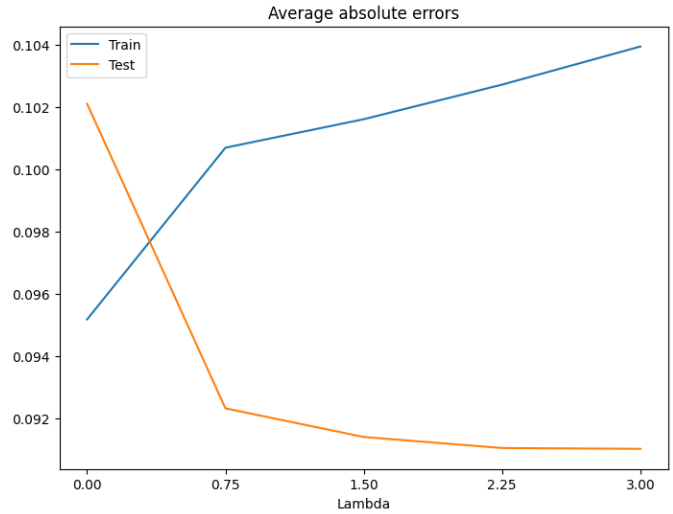
In [ ]:
```
for try_K in [10,11,12]:
    print(f">>>>> K = {try_K} <<<<<")
    plotDataForVaryingLambda(fit_neq_map, try_K, X_train, Y_train, X_test, Y_test, to_tr
```

```
>>>>> K = 10 <<<<<
Lambda=0.000 | Avg absolute error: [test] 0.10132666809449735  | [train] 0.0952491842033
4041 , average value of the parameters  = 0.30197342331334764
Lambda=0.750 | Avg absolute error: [test] 0.09174272250402181  | [train] 0.1008532682236
9893 , average value of the parameters  = 0.2977578656937399
Lambda=1.500 | Avg absolute error: [test] 0.09094231008238156  | [train] 0.1018975846474
8336 , average value of the parameters  = 0.3022281351500296
Lambda=2.250 | Avg absolute error: [test] 0.09071003854001986  | [train] 0.1030937045350
409  , average value of the parameters  = 0.30535749995442385
Lambda=3.000 | Avg absolute error: [test] 0.09078841288187879  | [train] 0.1043711914404
3762 , average value of the parameters  = 0.3077362463028416
```

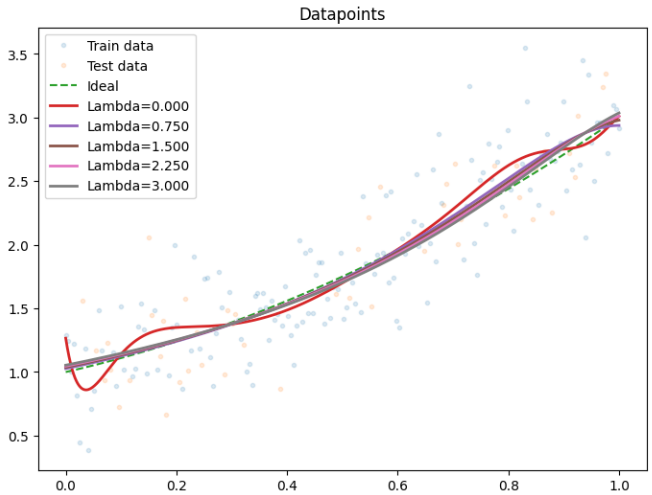


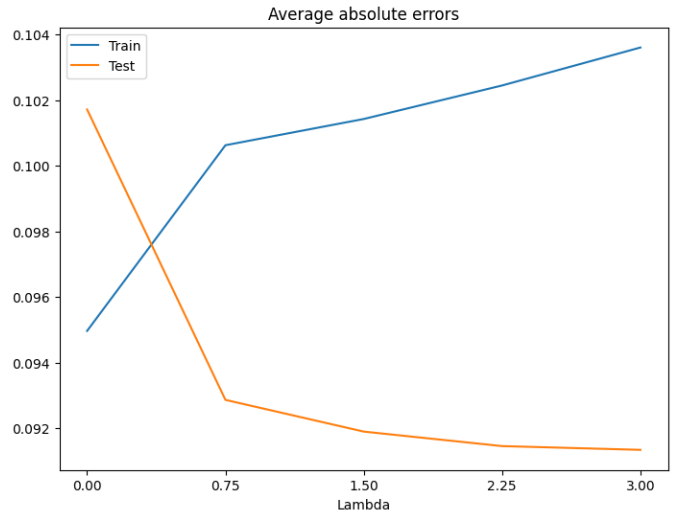
```
>>>>> K = 11 <<<<<
Lambda=0.000 | Avg absolute error: [test] 0.10210019685271444  | [train] 0.0951882361707
9344 , average value of the parameters  = 0.2764368375318637
Lambda=0.750 | Avg absolute error: [test] 0.09233560851760927  | [train] 0.1006922518357
7545 , average value of the parameters  = 0.2687302501987967
Lambda=1.500 | Avg absolute error: [test] 0.09141737732284057  | [train] 0.1016059481089
6072 , average value of the parameters  = 0.2726925059710036
Lambda=2.250 | Avg absolute error: [test] 0.091063425058491    | [train] 0.1027148221423
809  , average value of the parameters  = 0.27559363643609486
Lambda=3.000 | Avg absolute error: [test] 0.09103722599572706  | [train] 0.1039370053102
48   , average value of the parameters  = 0.27786848546073784
```






```
>>>>> K = 12 <<<<<
Lambda=0.000 | Avg absolute error: [test] 0.10171128182463442  | [train] 0.0949654481964
5915 , average value of the parameters  = 0.25091392795244855
Lambda=0.750 | Avg absolute error: [test] 0.09286439323065186  | [train] 0.1006217877301
3752 , average value of the parameters  = 0.24484384685830332
Lambda=1.500 | Avg absolute error: [test] 0.09189796568410531  | [train] 0.1014214841561
043  , average value of the parameters  = 0.24828125332361003
Lambda=2.250 | Avg absolute error: [test] 0.09145621823666647  | [train] 0.1024419961568
5987 , average value of the parameters  = 0.25091987023012957
Lambda=3.000 | Avg absolute error: [test] 0.09134205893344192  | [train] 0.1035988558812
376  , average value of the parameters  = 0.2530499795729761
```



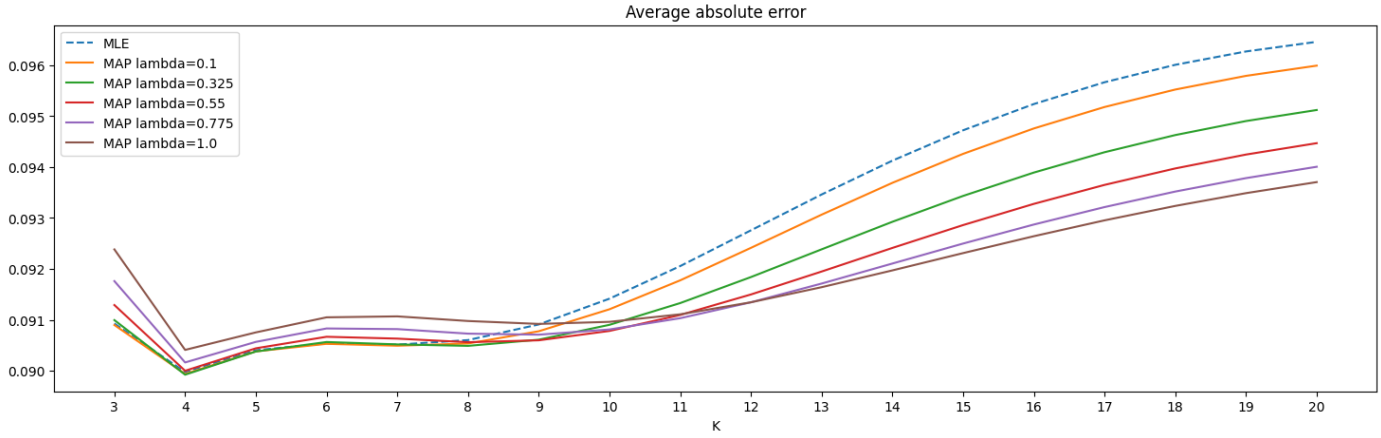


MLE vs MAP

Average absolute error for greater K

In [ ]:
```
to_try_K = [K+i for i in range(0, 18)]
to_try_lambda = np.linspace(0.1, 1, 5)
```

```
mle_abs_errors = []
map_abs_errors = { l: [] for l in to_try_lambda }

for try_K in to_try_K:
    theta_mle = fit_sgd_mle(X_train, Y_train, try_K, batch_size=64, epochs=100, lr=3e-3)
    mle_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_mle, try_K))

    for i, lamb in enumerate(to_try_lambda):
        theta_map = fit_sgd_map(X_train, Y_train, try_K, batch_size=64, epochs=100, lr=3
        map_abs_errors[lamb].append(avgAbsoluteError(X_test, Y_test, theta_map, try_K))

plt.figure(figsize=(18, 5))
plt.plot(to_try_K, mle_abs_errors, "--", label="MLE")
for lamb in to_try_lambda:
    plt.plot(to_try_K, map_abs_errors[lamb], label=f"MAP lambda={lamb}")
plt.title("Average absolute error")
plt.xlabel("K")
plt.xticks(to_try_K)
plt.legend()
plt.show()
```

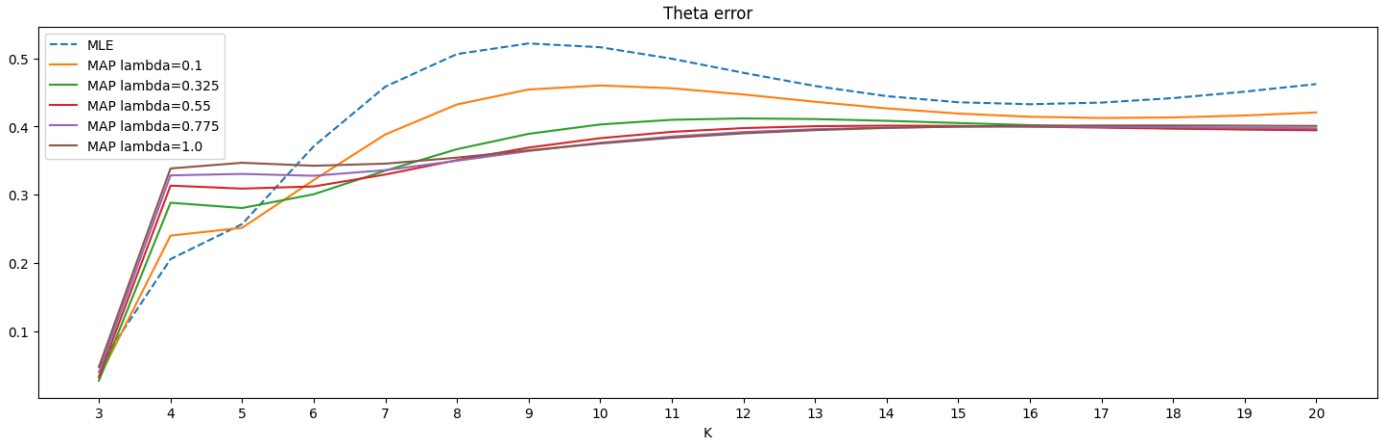


Theta error for greater K

In [ ]:

```
def thetaError(theta, theta_true):
    padded_theta = np.zeros(theta.shape)
    min_len = min(len(theta), len(theta_true))
    padded_theta[:min_len] = theta_true[:min_len]
    return np.linalg.norm(theta - padded_theta, 2) / np.linalg.norm(padded_theta, 2)

#to_try_K = [K+i for i in range(1,12)]
to_try_K = [K+i for i in range(0,18)]
mle_theta_errors = []
to_try_lambda = np.linspace(0.1, 1, 5)
map_theta_errors = { l: [] for l in to_try_lambda }

for try_K in to_try_K:
    theta_mle = fit_sgd_mle(X_train, Y_train, try_K, batch_size=64, epochs=1000, lr=3e-3
    mle_theta_errors.append(thetaError(theta_mle, theta_true))

    for i, lamb in enumerate(to_try_lambda):
        theta_map = fit_sgd_map(X_train, Y_train, try_K, batch_size=64, epochs=1000, lr=
        map_theta_errors[lamb].append(thetaError(theta_map, theta_true))

plt.figure(figsize=(18, 5))
plt.plot(to_try_K, mle_theta_errors, "--", label="MLE")
for lamb in to_try_lambda:
    plt.plot(to_try_K, map_theta_errors[lamb], label=f"MAP lambda={lamb}")
plt.title("Theta error")
plt.xlabel("K")
```

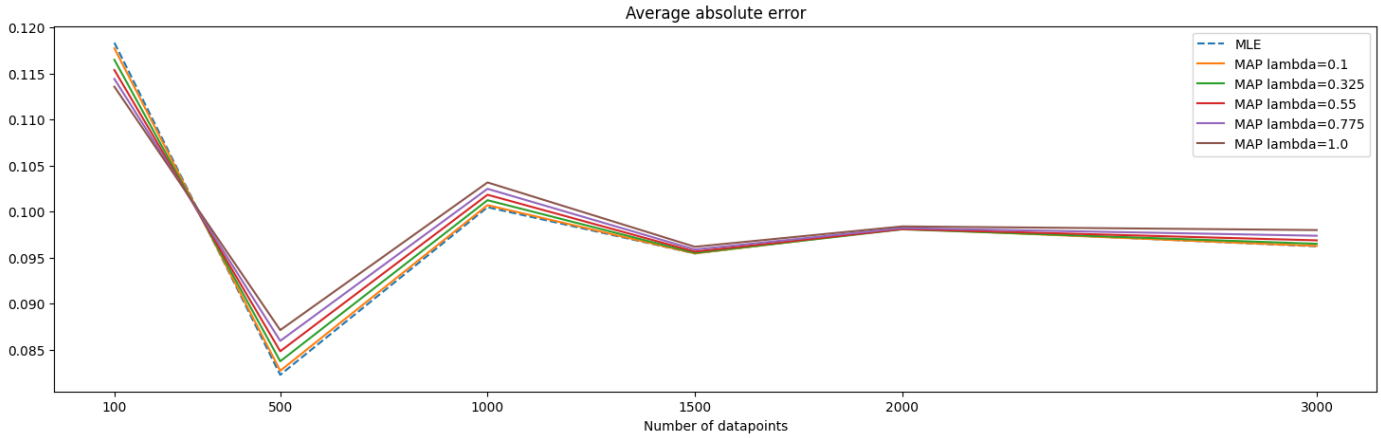
```
plt.xticks(to_try_K)
plt.legend()
plt.show()
```

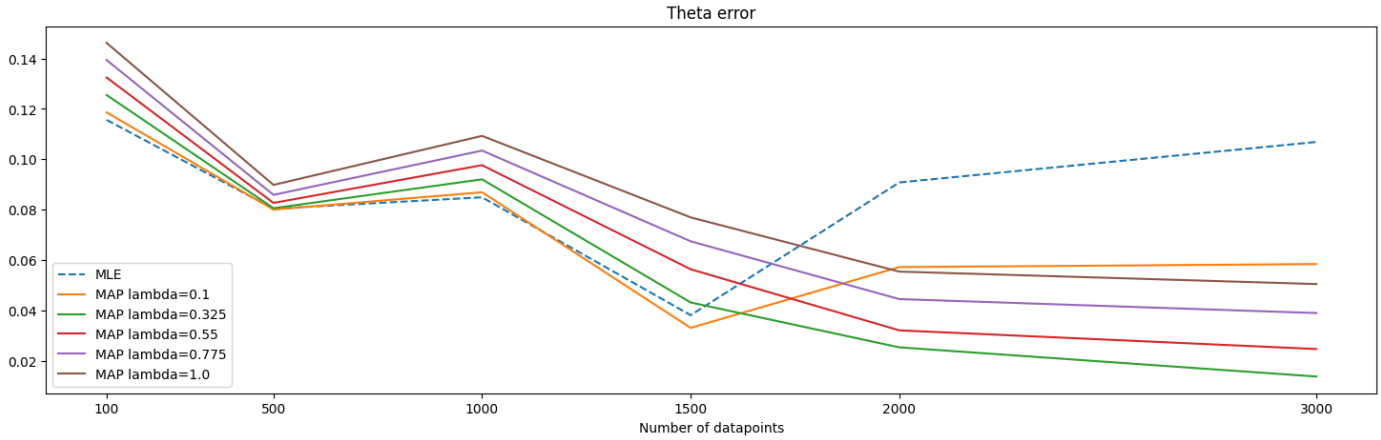


Different number of datapoints with correct K

In [ ]:

```
to_try_N = [100, 500, 1000, 1500, 2000, 3000]
mle_abs_errors = []
mle_theta_errors = []
to_try_lambda = np.linspace(0.1, 1, 5)
map_abs_errors = { l: [] for l in to_try_lambda }
map_theta_errors = { l: [] for l in to_try_lambda }

for try_N in to_try_N:
    X_full, Y_full = createDataset(try_N, a, b, K, theta_true, variance)
    X_train, Y_train, X_test, Y_test = train_test_split(X_full, Y_full, int(0.75*X_full.
    theta_mle = fit_sgd_mle(X_train, Y_train, K, batch_size=64, epochs=100, lr=3e-3)
    mle_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_mle, K))
    mle_theta_errors.append( thetaError(theta_mle, theta_true) )

    for lamb in to_try_lambda:
        theta_map = fit_sgd_map(X_train, Y_train, K, batch_size=64, epochs=100, lr=3e-3,
        map_abs_errors[lamb].append( avgAbsoluteError(X_test, Y_test,theta_map,K) )
        map_theta_errors[lamb].append( thetaError(theta_map, theta_true) )


plt.figure(figsize=(18, 5))
plt.title("Average absolute error")
plt.plot(to_try_N, mle_abs_errors, "--", label="MLE")
for lamb in to_try_lambda:
    plt.plot(to_try_N, map_abs_errors[lamb], label=f"MAP lambda={lamb}")
plt.xlabel("Number of datapoints")
plt.xticks(to_try_N)
plt.legend()
plt.show()

plt.figure(figsize=(18, 5))
plt.title("Theta error")
plt.plot(to_try_N, mle_theta_errors, "--", label="MLE")
for lamb in to_try_lambda:
    plt.plot(to_try_N, map_theta_errors[lamb], label=f"MAP lambda={lamb}")
plt.xlabel("Number of datapoints")
plt.xticks(to_try_N)
plt.legend()
plt.show()
```





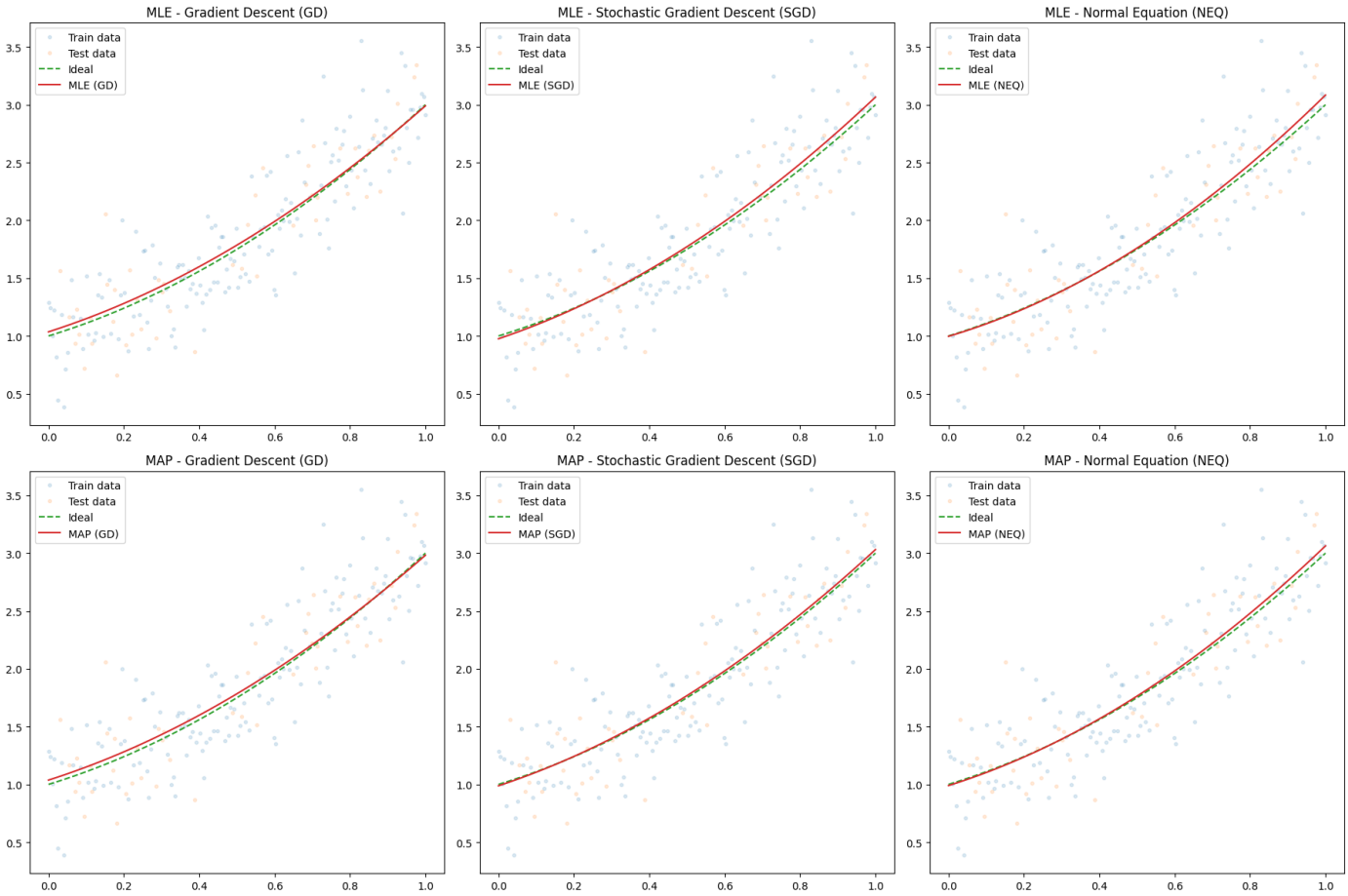
Different optimizers

In [ ]:
```
X_full, Y_full = createDataset(N, a, b, K, theta_true, variance)
X_train, Y_train, X_test, Y_test = train_test_split(X_full, Y_full, int(0.75*X_full.shap
```

In [ ]:
```
theta_mle_sgd = fit_sgd_mle(X_train, Y_train, K, batch_size=64, epochs=100, lr=3e-3)
theta_mle_gd = fit_gd_mle(X_train, Y_train, K, epochs=100)
theta_mle_neq = fit_neq_mle(X_train, Y_train, K)

lamb = 0.325
theta_map_sgd = fit_sgd_map(X_train, Y_train,K, batch_size=64, epochs=100, lr=3e-3, lamb
theta_map_gd = fit_gd_map(X_train, Y_train, K, lamb = lamb, epochs=100)
theta_map_neq = fit_neq_map(X_train, Y_train, K, lamb = lamb)


print(f"          | {'Theta error':^25} | {'Avg absolute error':^25} | {'Parameters':^25
print(f"MLE (SGD) | {thetaError(theta_mle_sgd, theta_true):^25.10f} | {avgAbsoluteError(
print(f"MLE (GD)  | {thetaError(theta_mle_gd, theta_true):^25.10f} | {avgAbsoluteError(X
print(f"MLE (NEQ) | {thetaError(theta_mle_neq, theta_true):^25.10f} | {avgAbsoluteError(
print(f"MAP (SGD) | {thetaError(theta_map_sgd, theta_true):^25.10f} | {avgAbsoluteError(
print(f"MAP (GD)  | {thetaError(theta_map_gd, theta_true):^25.10f} | {avgAbsoluteError(X
print(f"MAP (NEQ) | {thetaError(theta_map_neq, theta_true):^25.10f} | {avgAbsoluteError(

import numpy as np
import matplotlib.pyplot as plt


plot_X = np.linspace(X_train.min(), X_train.max(), 100)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))


axes[0, 0].plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
axes[0, 0].plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
axes[0, 0].plot(plot_X, predict(plot_X, theta_true, K), "--", label="Ideal")
```

```
axes[0, 0].plot(plot_X, predict(plot_X, theta_mle_gd, K), label="MLE (GD)")
axes[0, 0].set_title('MLE - Gradient Descent (GD)')
axes[0, 0].legend()

axes[0, 1].plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
axes[0, 1].plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
axes[0, 1].plot(plot_X, predict(plot_X, theta_true, K), "--", label="Ideal")
axes[0, 1].plot(plot_X, predict(plot_X, theta_mle_sgd, K), label="MLE (SGD)")
axes[0, 1].set_title('MLE - Stochastic Gradient Descent (SGD)')
axes[0, 1].legend()

axes[0, 2].plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
axes[0, 2].plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
axes[0, 2].plot(plot_X, predict(plot_X, theta_true, K), "--", label="Ideal")
axes[0, 2].plot(plot_X, predict(plot_X, theta_mle_neq, K), label="MLE (NEQ)")
axes[0, 2].set_title('MLE - Normal Equation (NEQ)')
axes[0, 2].legend()


axes[1, 0].plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
axes[1, 0].plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
axes[1, 0].plot(plot_X, predict(plot_X, theta_true, K), "--", label="Ideal")
axes[1, 0].plot(plot_X, predict(plot_X, theta_map_gd, K), label="MAP (GD)")
axes[1, 0].set_title('MAP - Gradient Descent (GD)')
axes[1, 0].legend()

axes[1, 1].plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
axes[1, 1].plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
axes[1, 1].plot(plot_X, predict(plot_X, theta_true, K), "--", label="Ideal")
axes[1, 1].plot(plot_X, predict(plot_X, theta_map_sgd, K), label="MAP (SGD)")
axes[1, 1].set_title('MAP - Stochastic Gradient Descent (SGD)')
axes[1, 1].legend()

axes[1, 2].plot(X_train, Y_train, ".", alpha=0.15, label="Train data")
axes[1, 2].plot(X_test, Y_test, ".", alpha=0.15, label="Test data")
axes[1, 2].plot(plot_X, predict(plot_X, theta_true, K), "--", label="Ideal")
axes[1, 2].plot(plot_X, predict(plot_X, theta_map_neq, K), label="MAP (NEQ)")
axes[1, 2].set_title('MAP - Normal Equation (NEQ)')
axes[1, 2].legend()
plt.tight_layout()
plt.show()
```

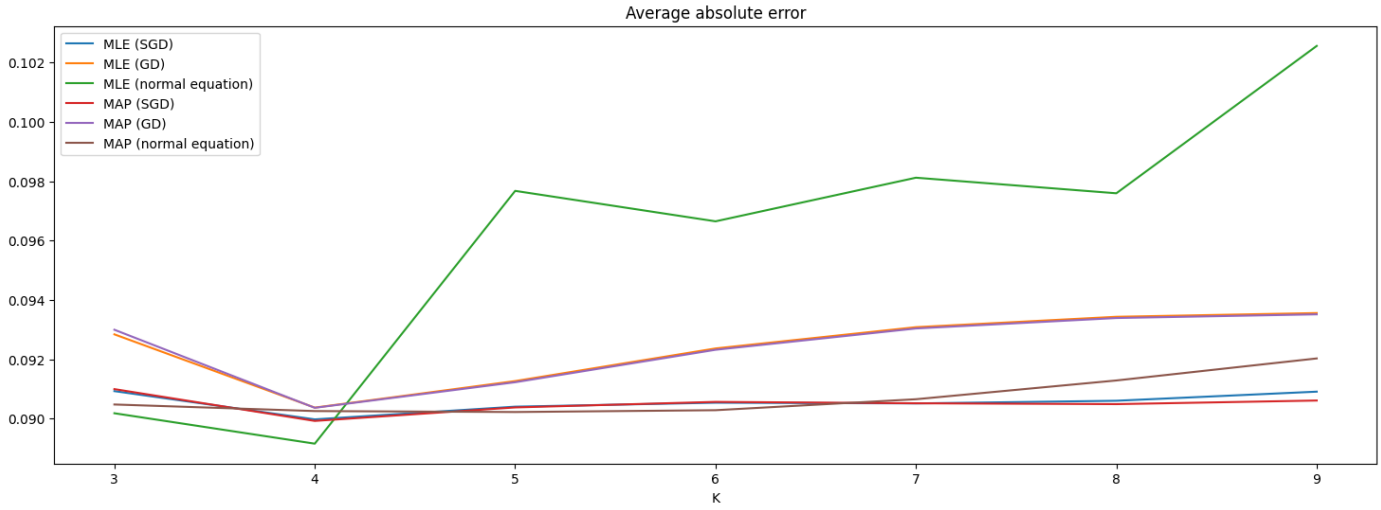
```
          |        Theta error        |    Avg absolute error     |        Parameters

MLE (SGD) |       0.0571361213        |       0.0909271447        | [0.97583546 1.095784
86 0.99408855]
MLE (GD)  |       0.0630910127        |       0.0928394935        | [1.03565914 1.046556
76 0.90779196]
MLE (NEQ) |       0.0729879546        |       0.0901819497        | [0.9969243  0.965225
83 1.12150318]
MAP (SGD) |       0.0485095814        |       0.0909947723        | [0.98797857 1.076246
53 0.96681129]
MAP (GD)  |       0.0648740103        |       0.0929944843        | [1.03714365 1.042133
59 0.90268083]
MAP (NEQ) |       0.0328824394        |       0.0904768697        | [0.98985963 1.021415
72 1.05179097]
```

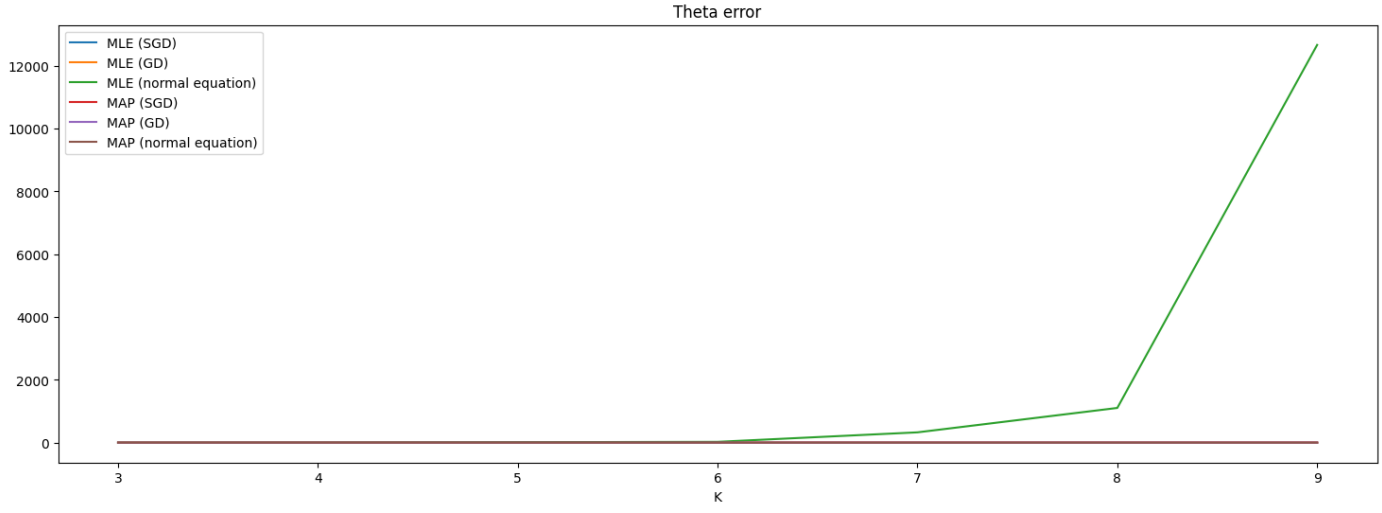


we can observe that in MLE (SGD) parameters are trained in order to adapt the curve to the noise defined by the gaussian distribution, while MAP impose that the value of the parameters has to be as low as possible, so we can observe that the MAP's shape is closer to the ideal one

In [ ]:
```
to_try_K = range(3,10)
mle_sgd_abs_errors = []
mle_gd_abs_errors = []
mle_neq_abs_errors = []
map_sgd_abs_errors = []
map_gd_abs_errors = []
map_neq_abs_errors = []
mle_sgd_theta_errors = []
mle_gd_theta_errors = []
mle_neq_theta_errors = []
map_sgd_theta_errors = []
map_gd_theta_errors = []
map_neq_theta_errors = []

for K_try in to_try_K:
    theta_mle_sgd = fit_sgd_mle(X_train, Y_train, K_try,epochs=100)
    theta_mle_gd = fit_gd_mle(X_train, Y_train, K_try,epochs=100)
    theta_mle_neq = fit_neq_mle(X_train, Y_train, K_try)
    theta_map_sgd = fit_sgd_map(X_train, Y_train, K_try, lamb = 0.325,epochs=100)
    theta_map_gd = fit_gd_map(X_train, Y_train, K_try, lamb = 0.325,epochs=100)
    theta_map_neq = fit_neq_map(X_train, Y_train, K_try, lamb = 0.325)
    mle_sgd_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_mle_sgd, K_try))
    mle_gd_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_mle_gd, K_try))
    mle_neq_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_mle_neq, K_try))
    map_sgd_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_map_sgd, K_try))
    map_gd_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_map_gd, K_try))
    map_neq_abs_errors.append(avgAbsoluteError(X_test, Y_test, theta_map_neq, K_try))
    mle_sgd_theta_errors.append(thetaError(theta_mle_sgd, theta_true))
    mle_gd_theta_errors.append(thetaError(theta_mle_gd, theta_true))
    mle_neq_theta_errors.append(thetaError(theta_mle_neq, theta_true))
```

```
    map_sgd_theta_errors.append(thetaError(theta_map_sgd, theta_true))
    map_gd_theta_errors.append(thetaError(theta_map_gd, theta_true))
    map_neq_theta_errors.append(thetaError(theta_map_neq, theta_true))

plt.figure(figsize=(18, 6))
plt.plot(to_try_K, mle_sgd_abs_errors, label="MLE (SGD)")
plt.plot(to_try_K, mle_gd_abs_errors, label="MLE (GD)")
plt.plot(to_try_K, mle_neq_abs_errors, label="MLE (normal equation)")
plt.plot(to_try_K, map_sgd_abs_errors, label="MAP (SGD)")
plt.plot(to_try_K, map_gd_abs_errors, label="MAP (GD)")
plt.plot(to_try_K, map_neq_abs_errors, label="MAP (normal equation)")
plt.title("Average absolute error")
plt.xlabel("K")
plt.xticks(to_try_K)
plt.legend()
plt.show()

plt.figure(figsize=(18, 6))
plt.plot(to_try_K, mle_sgd_theta_errors, label="MLE (SGD)")
plt.plot(to_try_K, mle_gd_theta_errors, label="MLE (GD)")
plt.plot(to_try_K, mle_neq_theta_errors, label="MLE (normal equation)")
plt.plot(to_try_K, map_sgd_theta_errors, label="MAP (SGD)")
plt.plot(to_try_K, map_gd_theta_errors, label="MAP (GD)")
plt.plot(to_try_K, map_neq_theta_errors, label="MAP (normal equation)")
plt.title("Theta error")
plt.xlabel("K")
plt.xticks(to_try_K)
plt.legend()
plt.show()
```





here we can understand how much is important the MAP's regularizer factor if we observe the solution computed using NE in case of MLE and MAP