```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#### DATA PREPARATION

data_file = 'data_ex1_wt.csv' # absolute path respect to when the script is run
df = pd.read_csv(data_file, header=None, names=['time', 'metric'])

x = df['time'].values
y = df['metric'].values

#### REMOVING TREND FROM THE DATA
m = 5 #best degree

def least_squares_methods(x, y, degree):

    #compute Vandermonde matrix
    A = []

    for i in range(len(x)):
        A.append([])

    for j in range(len(x)):
        for i in range(degree+1):
            A[j].append(x[j]**i)

    #transpose of A
    A_transpose = np.transpose(A)

    # Compute A' A
    A_transpose_A = np.dot(A_transpose, A)
    # Compute (A' A)^-1
    A_transpose_A_inv = np.linalg.inv(A_transpose_A)
    # Compute (A' A)^-1 A'
    A_transpose_A_inv_transpose_A = np.dot(A_transpose_A_inv, A_transpose)
    # Compute (A' A)^-1 A' y (= b)
    b = np.dot(A_transpose_A_inv_transpose_A, y)

    return b

p = least_squares_methods(x, y, m)
yy = np.zeros(len(x))
for j in range(len(p)):
    yy += p[j] * (x**j)

residuals = y - yy
y = residuals

# pdf of a gaussian
def f(x, mu, sigma):
    return (1/(sigma * np.sqrt(2 * np.pi))) * np.exp( - (x - mu)**2 / (2 * sigma**2) )
#

########################################
####### IMPLEMENTING EXPECTATION MAXIMIZATION ALGORITHM

##EM algorithm to fit a mixture of gaussians to the data
## Parameters:
# data: the data to fit
# gaussians: the number of gaussians to fit
# num_epochs: the number of epochs to run the algorithm
# print_every: the number of epochs to print the parameters
# Returns:
# parameters: the parameters of the gaussians parameters[i] = [mu, sigma]
# assignments: the assignments of the data points to the gaussians
# priors: the priors of the gaussians
def em(data, gaussians, num_epochs=90, print_every=15):

    #Auxiliary functions
    ##################################

    #random value in an internval [low, high] - we initialize in this way to speed up the convergence
    def get_random(low=-np.std(data), high=np.std(data)):
        return np.random.rand() * (high-low) + low

    #pdf of a gaussian
    def f(x, mu, sigma):
        return (1/(sigma * np.sqrt(2 * np.pi))) * np.exp( - (x - mu)**2 / (2 * sigma**2) )

    #internal function to print parameters
    def print_parameters():
        for k in range(gaussians):
            print("\tmu", k, "sigma", k, parameters[k][0], parameters[k][1])
        print("\tPriors: ", priors)
        print("\n")

    #internal function to compute the likelihood of datapoint coming from gaussian i
    def compute_likelihood(mu, sigma, datapoint, i):
        numerator = f(datapoint, mu, sigma) * priors[i]
        denominator = 0
        for k in range(gaussians):
            denominator += f(datapoint, parameters[k][0], parameters[k][1]) * priors[k]
        return numerator / denominator

    #internal function to reset the parameters
    def reset():
        for k in range(gaussians):
            parameters[k] = [0, 0]

    ##################################

    #Parameters initialization
    start1 = np.mean(data)
    start2 = np.std(data)
```

```python
    parameters = []
    priors = []


    #initialize the parameters of the gaussians
    for _ in range(gaussians):
        parameters.append([start1 + get_random(), start2])

    #initialize the priors of the gaussians
    for _ in range(gaussians):
        priors.append(1/gaussians)

    N = len(data)

    for epoch in range(num_epochs):

        if print_every != 0 and epoch % print_every == 0:
            print("Epoch: ", epoch)
            print_parameters()

        assignments = []

        #E-step
        for i in range(N):
            assignments.append([])
            #for each gaussian, compute the likelihood of the datapoint coming from that gaussian
            for k in range(gaussians):
                assignments[i].append(compute_likelihood(parameters[k][0], parameters[k][1], data[i], k))


        #M-step
        reset()
        counts = [0] * gaussians

        #update the parameters of the gaussians with various iterations (to compute sum of the likelihoods abd mean)
        for i in range(N):
            for k in range(gaussians):
                counts[k] += assignments[i][k]
                parameters[k][0] += assignments[i][k] * data[i]

        for k in range(gaussians):
            parameters[k][0] /= counts[k]

        #std dev computation
        for i in range(N):
            for k in range(gaussians):
                parameters[k][1] += assignments[i][k] * ((data[i] - parameters[k][0])**2)

        for k in range(gaussians):
            parameters[k][1] /= counts[k]
            parameters[k][1] = np.sqrt(parameters[k][1])

        #update the priors of the gaussians
        for k in range(gaussians):
            priors[k] = counts[k] / N

    return parameters, assignments, priors

parameters, assignments, priors = em(y, 3, num_epochs=150, print_every=30)

real_mu1, real_sigma1 = -5, np.sqrt(3)
real_mu2, real_sigma2 = 0, np.sqrt(6)
real_mu3, real_sigma3 = 4, np.sqrt(1)
print("Final mu1, sigma1", parameters[0][0], parameters[0][1])
print("Final mu2, sigma2", parameters[1][0], parameters[1][1])
print("Final mu3, sigma3", parameters[2][0], parameters[2][1])
print("Actual mu1, sigma1", real_mu1, real_sigma1)
print("Actual mu2, sigma2", real_mu2, real_sigma2)
print("Actual mu3, sigma3", real_mu3, real_sigma3)

###PLOTTING THE RESULTS
fig = plt.figure()

BINS=200
plt.hist(residuals, bins=BINS, color='red', alpha=0.7, density=True)
plt.xlabel('Metric values')
plt.ylabel('Density')
plt.title('Histogram of data WITHOUT trend')

points = np.linspace(-11, 11, 1000)
plt.plot(points, list(map(lambda x: f(x, parameters[0][0], parameters[0][1]), points)), color='green', linestyle='dashed')
plt.plot(points, list(map(lambda x: f(x, parameters[1][0], parameters[1][1]), points)), color='orange', linestyle='dashed')
plt.plot(points, list(map(lambda x: f(x, parameters[2][0], parameters[2][1]), points)), color='purple', linestyle='dashed')
plt.plot(points, list(map(lambda x: f(x, real_mu1, real_sigma1), points)), color='black')
plt.plot(points, list(map(lambda x: f(x, real_mu2, real_sigma2), points)), color='brown')
plt.plot(points, list(map(lambda x: f(x, real_mu3, real_sigma3), points)), color='gray')

plt.tight_layout()
plt.show()

#Few discrepancies betwen the estimated and real distributions used in generating the data
#Increasing number of epochs we could achieve a better fit
#Additionally, it is important to note that it's very difficult to assign every point correctly to the right gaussian (according to the generation process)
#This is because maybe in the generation process it belongs to the tail of one distribution, while maybe here is estiamted to belong to another one
#Anyway, reasoning in terms of "classification", this can be considered a good and desirable result, since maybe the point was an outlier in the generation pro

#At the end, with enough number of epochs, the algorithm converges always to the solution that better fits the data
```