```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
import os


# pdf of exponential distribution
def exp_dist_PDF(x, l):
    # lambda e^(-lambnda x)
    return l*np.exp(-l*x)


def plot_mixture_pdf(pi_1, pi_2, l_1, l_2, name, ylab, iters=False):

    # generate samples
    num_samples = 10000
    exp_mixture = np.zeros(shape=(num_samples, 2))

    for i in range(num_samples):
        # rescale i
        x_input = i / 1000
        # mixture PDF
        mix_y = pi_1 * exp_dist_PDF(x_input, l_1) + pi_2 * exp_dist_PDF(x_input, l_2)
        exp_mixture[i] = [x_input, mix_y]

    # save PDF data
    plt.plot(exp_mixture[:, 0], exp_mixture[:, 1])
    plt.title(name)
    plt.xlabel(r'$x$')
    plt.ylabel(ylab)
    if iters:
        plt.savefig("plots/iters/" + name + ".png")
    else:
        plt.savefig("plots/" +name+".png")
    plt.clf()
    plt.close()

def plot_log_likelihood_history(LL_hist):
    x_axis = np.arange(LL_hist.shape[0])
    plt.plot(x_axis, LL_hist)
    plt.xlabel("Iterations")
    plt.ylabel(r'$\log f(X ; \hat{\theta})$')
    plt.title("Log Likelihood")
    plt.savefig("plots/LogLikelihood.png")
    plt.clf()
    plt.close()


def E_step(data, pi_1, pi_2, l1, l2):

    # compute conditional expectation for each mixture
    Q_theta = np.zeros(shape=(data.shape[0], 2))

    for i in range(data.shape[0]):
        # get data point
        xi = data[i]
        # compute each expectation
        qi1_num = pi_1*exp_dist_PDF(xi, l1)
        qi2_num = pi_2*exp_dist_PDF(xi, l2)
        qi_denom= qi1_num + qi2_num
        # add it to expectation buffer
        Q_theta[i][0] = qi1_num / qi_denom
        Q_theta[i][1] = qi2_num / qi_denom

    # return the conditional expectation
    return Q_theta
```

```python
def M_step(data, Q_theta):

    # update parameters
    l1_update_num = 0
    l1_update_denom = 0

    l2_update_num = 0
    l2_update_denom = 0

    pi_1_update_num = 0
    pi_1_update_denom = 0

    pi_2_update_num = 0
    pi_2_update_denom = 0

    # for each data point
    for i in range(data.shape[0]):
        # compute parameter update for each data point
        pi_1_update_num += Q_theta[i][0]
        pi_2_update_num += Q_theta[i][1]

        pi_1_update_denom += Q_theta[i][0] + Q_theta[i][1]
        pi_2_update_denom += Q_theta[i][0] + Q_theta[i][1]

        l1_update_num += Q_theta[i][0]
        l2_update_num += Q_theta[i][1]

        l1_update_denom += Q_theta[i][0]*data[i]
        l2_update_denom += Q_theta[i][1]*data[i]

    # combine all update rules to get the parameter update
    pi_1_update = pi_1_update_num / pi_1_update_denom
    pi_2_update = pi_2_update_num / pi_2_update_denom

    l1_update = l1_update_num / l1_update_denom
    l2_update = l2_update_num / l2_update_denom
    # return all parameters
    return pi_1_update, pi_2_update, l1_update, l2_update



def check_early_termination(theta_hist, idx):
    # make sure we're not checking this at the first iteration
    # otherwise we get an out of bounds error
    if idx > 0:
        # if every element in the previous iteration is exactly the same
        # end the EM algorithm at it reached a local or global minima
        if np.array_equal(theta_hist[idx], theta_hist[idx-1]):
            return True
        # otherwise don't terminate and keep going
        else:
            return False


def current_log_likelihood(data, pi_1, pi_2, l1, l2):
    # compute the log likelihood of the current estimate
    LL = 0
    # for every datapoint
    for d in data:
        # sum the log likelihood at this datapoint
        LL += np.log(pi_1*exp_dist_PDF(d, l1) + pi_2*exp_dist_PDF(d, l2))
    return LL
```

```python
def EM(data):

    # randomly initialize guesses
    l1 = np.random.rand()*5.0
    l2 = np.random.rand()*5.0

    pi_1 = np.random.rand()
    pi_2 = 1 - pi_1

    # fixed number of steps if our estimate doesn't converge
    num_steps = 100000
    # history of estimates
    theta_history = np.zeros(shape=(num_steps+1, 4))
    log_likelihood_history = []
    # save first guess in history buffer
    theta_history[0] = [l1, l2, pi_1, pi_2]
    for i in range(num_steps):

        # Compute log likelihood for current estimate
        log_likelihood_history.append(current_log_likelihood(data, pi_1, pi_2, l1, l2))
        # E step
        Q_theta = E_step(data, pi_1, pi_2, l1, l2)
        # M Step
        pi_1, pi_2, l1, l2 = M_step(data, Q_theta)

        # append data to our estimation history
        theta_history[i+1] = [l1, l2, pi_1, pi_2]
        # check for early termination
        if check_early_termination(theta_history, i):
            # early termination, slice history array and return
            theta_history = theta_history[:i+2]
            break


    return theta_history, np.asarray(log_likelihood_history)



# rejection sampling from mixture PDF
def rejection_sampling(l_1, l_2, pi_1, pi_2, num_samples = 20):
    # count number of accepted samples
    num_accepted = 0
    # max of our pdf
    max_pdf = pi_1 * exp_dist_PDF(0, l_1) + pi_2 * exp_dist_PDF(0, l_2)
    # store all samples here
    samples = []
    # keep iterating until we get the desired number of samples
    while num_accepted != num_samples:
        # uniform sample from a support
        x_i = np.random.rand()*10
        # get the pdf of that sample
        pdf_y = pi_1 * exp_dist_PDF(x_i, l_1) + pi_2 * exp_dist_PDF(x_i, l_2)
        # uniform sample from our support of PDFs
        u_i = np.random.rand()*max_pdf
        # if our random sample is less than the sampled pdf
        # accept the sample
        if u_i < pdf_y:
            samples.append(x_i)
            num_accepted += 1

    return np.asarray(samples)



if __name__ == "__main__":

    # Set to true for plot of each iteration
    DETAILED_OUTPUT = False
```

```python
# create directories for saving figures
if not os.path.exists("plots"):
    os.makedirs("plots")

if DETAILED_OUTPUT:
    if not os.path.exists("plots/iters"):
        os.makedirs("plots/iters")


# fix seed for reproducibility
np.random.seed(999)
# plotting params
rcParams.update({'figure.autolayout': True})

# true parameters
true_l1   = 1
true_l2   = 3
true_pi_1 = 0.25
true_pi_2 = 1.0 - true_pi_1

# plot the mixture pdf
plot_mixture_pdf(l_1=true_l1, l_2=true_l2, pi_1=true_pi_1, pi_2=true_pi_2
                 , name="ExponentialMixture", ylab=r'$f(x; \theta)$')
# generate n observations
data = rejection_sampling(l_1=true_l1, l_2=true_l2, pi_1=true_pi_1, pi_2=true_pi_2)
# run EM
theta_history, LL_history = EM(data)

# grab the last estimate of our parameters
theta_history_last = theta_history[-1]

# grab each parameters separately to plot pdf
estimated_l1   = theta_history_last[0]
estimated_l2   = theta_history_last[1]
estimated_pi_1 = theta_history_last[2]
estimated_pi_2 = theta_history_last[3]

# plot final estimated parameters pdf
plot_mixture_pdf(l_1= estimated_l1, l_2= estimated_l2, pi_1= estimated_pi_1, pi_2=estimated_pi_2
                 , name="EstimatedExponentialMixture", ylab=r'$f(x; \hat{\theta})$')

if DETAILED_OUTPUT:
    # plot estimated parameter pdf for some iterations
    # save every x steps
    save_every = 1
    for i in range(theta_history.shape[0]):
        estimate_at_i = theta_history[i]
        if (i % save_every) == 0:
            estimated_l1 = estimate_at_i[0]
            estimated_l2 = estimate_at_i[1]
            estimated_pi_1 = estimate_at_i[2]
            estimated_pi_2 = estimate_at_i[3]
            # plot estimated parameters pdf for current step
            plot_mixture_pdf(l_1=estimated_l1, l_2=estimated_l2, pi_1=estimated_pi_1, pi_2=estimated_pi_2
                             , name="EstimatedExponentialMixture_Iteration"+str(i), ylab=r'$f(x; \hat{\theta})$',
                             iters=True)



plot_log_likelihood_history(LL_history)
np.savetxt("plots/theta_history.txt", theta_history)
```