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

PLOT_COLORS = ['red', 'green', 'blue', 'orange']  # Colors for your plots
K = 4           # Number of Gaussians in the mixture model
NUM_TRIALS = 3  # Number of trials to run (can be adjusted for debugging)
UNLABELED = -1  # Cluster label for unlabeled data points (do not change)


def main(is_semi_supervised, trial_num):
    """Problem 3: EM for Gaussian Mixture Models (unsupervised and semi-supervised)"""
    print('Running {} EM algorithm...'
          .format('semi-supervised' if is_semi_supervised else 'unsupervised'))

    # Load dataset
    train_path = os.path.join('.', 'train.csv')
    x_all, z_all = load_gmm_dataset(train_path)

    # Split into labeled and unlabeled examples
    labeled_idxs = (z_all != UNLABELED).squeeze()
    x_tilde = x_all[labeled_idxs, :]   # Labeled examples
    z_tilde = z_all[labeled_idxs, :]   # Corresponding labels
    x = x_all[~labeled_idxs, :]        # Unlabeled examples

    # *** START CODE HERE ***
    # (1) Initialize mu and sigma by splitting the n_examples data points uniformly at random
    # into K groups, then calculating the sample mean and covariance for each group
    n, d = x.shape
    group = np.random.choice(K, n)
    mu = [np.mean(x[group == g, :], axis=0) for g in range(K)]
    sigma = [np.cov(x[group == g, :].T) for g in range(K)]

    # (2) Initialize phi to place equal probability on each Gaussian
    # phi should be a numpy array of shape (K,)
    phi = np.full((K,), fill_value=(1. / K), dtype=np.float32)

    # (3) Initialize the w values to place equal probability on each Gaussian
    # w should be a numpy array of shape (m, K)
    w = np.full((n, K), fill_value=(1. / K), dtype=np.float32)
    # *** END CODE HERE ***

    if is_semi_supervised:
        w = run_semi_supervised_em(x, x_tilde, z_tilde, w, phi, mu, sigma)
    else:
        w = run_em(x, w, phi, mu, sigma)

    # Plot your predictions
    z_pred = np.zeros(n)
    if w is not None:  # Just a placeholder for the starter code
        for i in range(n):
            z_pred[i] = np.argmax(w[i])

    plot_gmm_preds(x, z_pred, is_semi_supervised, plot_id=trial_num)


def run_em(x, w, phi, mu, sigma):
    """Problem 3(d): EM Algorithm (unsupervised).

    See inline comments for instructions.

    Args:
        x: Design matrix of shape (n_examples, dim).
        w: Initial weight matrix of shape (n_examples, k).
        phi: Initial mixture prior, of shape (k,).
        mu: Initial cluster means, list of k arrays of shape (dim,).
        sigma: Initial cluster covariances, list of k arrays of shape (dim, dim).

    Returns:
        Updated weight matrix of shape (n_examples, k) resulting from EM algorithm.
        More specifically, w[i, j] should contain the probability of
        example x^(i) belonging to the j-th Gaussian in the mixture.
```

```
    """
    # No need to change any of these parameters
    eps = 1e-3  # Convergence threshold
    max_iter = 1000

    # Stop when the absolute change in log-likelihood is < eps
    # See below for explanation of the convergence criterion
    it = 0
    ll = prev_ll = None
    while it < max_iter and (prev_ll is None or np.abs(ll - prev_ll) >= eps):
        pass  # Just a placeholder for the starter code
        # *** START CODE HERE
        # (1) E-step: Update your estimates in w
        w = e_step(x, w, phi, mu, sigma)

        # (2) M-step: Update the model parameters phi, mu, and sigma
        phi, mu, sigma = m_step(x, w, mu, sigma)

        # (3) Compute the log-likelihood of the data to check for convergence.
        # By log-likelihood, we mean `ll = sum_x[log(sum_z[p(x|z) * p(z)])]`.
        # We define convergence by the first iteration where abs(ll - prev_ll) < eps.
        # Hint: For debugging, recall part (a). We showed that ll should be monotonically increasing.
        prev_ll = ll
        ll = log_likelihood(x, phi, mu, sigma)
        it += 1
        print('[iter: {:03d}, log-likelihood: {:.4f}]'.format(it, ll))
        # *** END CODE HERE ***

    return w


def run_semi_supervised_em(x, x_tilde, z_tilde, w, phi, mu, sigma):
    """Problem 3(e): Semi-Supervised EM Algorithm.

    See inline comments for instructions.

    Args:
        x: Design matrix of unlabeled examples of shape (n_examples_unobs, dim).
        x_tilde: Design matrix of labeled examples of shape (n_examples_obs, dim).
        z_tilde: Array of labels of shape (n_examples_obs, 1).
        w: Initial weight matrix of shape (n_examples, k).
        phi: Initial mixture prior, of shape (k,).
        mu: Initial cluster means, list of k arrays of shape (dim,).
        sigma: Initial cluster covariances, list of k arrays of shape (dim, dim).

    Returns:
        Updated weight matrix of shape (n_examples, k) resulting from semi-supervised EM algorithm.
        More specifically, w[i, j] should contain the probability of
        example x^(i) belonging to the j-th Gaussian in the mixture.
    """
    # No need to change any of these parameters
    alpha = 20.  # Weight for the labeled examples
    eps = 1e-3   # Convergence threshold
    max_iter = 1000

    # Stop when the absolute change in log-likelihood is < eps
    # See below for explanation of the convergence criterion
    it = 0
    ll = prev_ll = None
    while it < max_iter and (prev_ll is None or np.abs(ll - prev_ll) >= eps):
        pass  # Just a placeholder for the starter code
        # *** START CODE HERE ***
        # (1) E-step: Update your estimates in w
        w = e_step(x, w, phi, mu, sigma)

        # (2) M-step: Update the model parameters phi, mu, and sigma
        phi, mu, sigma = m_step_ss(x, x_tilde, z_tilde, w, phi, mu, sigma, alpha)

        # (3) Compute the log-likelihood of the data to check for convergence.
        # Hint: Make sure to include alpha in your calculation of ll.
        # Hint: For debugging, recall part (a). We showed that ll should be monotonically increasing.
        prev_ll = ll
```

```python
        ll = log_likelihood(x, phi, mu, sigma)
        ll += alpha * log_likelihood(x_tilde, phi, mu, sigma, z_tilde)
        it += 1
        print('[iter: {:03d}, log-likelihood: {:.4f}]'.format(it, ll))
        # *** END CODE HERE ***

    return w


# *** START CODE HERE ***
# Helper functions

def e_step(x, w, phi, mu, sigma):
    """E-step for both unsupervised and semi-supervised EM."""
    n, d = x.shape
    k = len(mu)

    for i in range(n):
        for j in range(k):
            w[i, j] = p_x_given_z(x[i], mu[j], sigma[j]) * phi[j]

    w /= np.sum(w, axis=1, keepdims=True)

    return w


def m_step(x, w, mu, sigma):
    """M-step for unsupervised EM."""
    n, d = x.shape
    k = len(mu)

    phi = np.mean(w, axis=0)

    for j in range(k):
        w_j = w[:, j:j + 1]
        mu[j] = np.sum(w_j * x, axis=0) / np.sum(w_j)

        sigma[j] = np.zeros_like(sigma[j])
        for i in range(n):
            x_minus_mu = x[i] - mu[j]
            sigma[j] += w[i, j] * np.outer(x_minus_mu, x_minus_mu)
        sigma[j] /= np.sum(w_j)

    return phi, mu, sigma


def m_step_ss(x, x_tilde, z_tilde, w, phi, mu, sigma, alpha):
    """M-step for semi-supervised EM."""
    n, _ = x.shape
    n_tilde, _ = x_tilde.shape
    k = len(mu)

    w_colsums = np.sum(w, axis=0)
    k_counts = [np.sum(z_tilde == j) for j in range(k)]
    for j in range(k):
        phi[j] = (w_colsums[j] + alpha * k_counts[j]) / (n + alpha * n_tilde)

        w_j = w[:, j:j + 1]
        mu[j] = ((np.sum(w_j * x, axis=0)
                     + alpha * np.sum(x_tilde[(z_tilde == j).squeeze(), :], axis=0))
                 / (np.sum(w_j) + alpha * k_counts[j]))

        sigma[j] = np.zeros_like(sigma[j])
        for i in range(n):
            x_minus_mu = x[i] - mu[j]
            sigma[j] += w[i, j] * np.outer(x_minus_mu, x_minus_mu)
        for i in range(n_tilde):
            if z_tilde[i] == j:
                x_minus_mu = x_tilde[i] - mu[j]
                sigma[j] += alpha * np.outer(x_minus_mu, x_minus_mu)
        sigma[j] /= (np.sum(w_j) + alpha * k_counts[j])
```

```python
    return phi, mu, sigma


def log_likelihood(x, phi, mu, sigma, z=None):
    """Get log-likelihood of the data `x` given model parameters
    `phi`, `mu`, and `sigma`.
    """
    n, d = x.shape
    k = len(phi)
    ll = 0.
    for i in range(n):
        if z is None:  # Unsupervised case
            p_x = 0.
            for j in range(k):
                p_x += p_x_given_z(x[i], mu[j], sigma[j]) * phi[j]
        else:  # Supervised case
            j = int(z[i])
            p_x = p_x_given_z(x[i], mu[j], sigma[j]) * phi[j]
        ll += np.log(p_x)

    return ll


def p_x_given_z(x, mu, sigma):
    """Get probability of a single example `x` given model parameters
    `mu` and `sigma` (corresponding to cluster z = j).
    """
    d = len(x)
    assert d == len(mu) and sigma.shape == (d, d), 'Shape mismatch.'

    c = 1. / ((2. * np.pi) ** (d / 2) * np.sqrt(np.linalg.det(sigma)))
    x_minus_mu = x - mu
    sigma_inv = np.linalg.inv(sigma)
    p_val = c * np.exp(-.5 * x_minus_mu.dot(sigma_inv).dot(x_minus_mu.T))

    return p_val
# *** END CODE HERE ***


def plot_gmm_preds(x, z, with_supervision, plot_id):
    """Plot GMM predictions on a 2D dataset `x` with labels `z`.

    Write to the output directory, including `plot_id`
    in the name, and appending 'ss' if the GMM had supervision.

    NOTE: You do not need to edit this function.
    """
    plt.figure(figsize=(12, 8))
    plt.title('{} GMM Predictions'.format('Semi-supervised' if with_supervision else 'Unsupervised'))
    plt.xlabel('x_1')
    plt.ylabel('x_2')

    for x_1, x_2, z_ in zip(x[:, 0], x[:, 1], z):
        color = 'gray' if z_ < 0 else PLOT_COLORS[int(z_)]
        alpha = 0.25 if z_ < 0 else 0.75
        plt.scatter(x_1, x_2, marker='.', c=color, alpha=alpha)

    file_name = 'pred{}_{}.pdf'.format('_ss' if with_supervision else '', plot_id)
    save_path = os.path.join('.', file_name)
    plt.savefig(save_path)


def load_gmm_dataset(csv_path):
    """Load dataset for Gaussian Mixture Model.

    Args:
         csv_path: Path to CSV file containing dataset.

    Returns:
        x: NumPy array shape (n_examples, dim)
        z: NumPy array shape (n_exampls, 1)

```

```
    NOTE: You do not need to edit this function.
    """

    # Load headers
    with open(csv_path, 'r') as csv_fh:
        headers = csv_fh.readline().strip().split(',')

    # Load features and labels
    x_cols = [i for i in range(len(headers)) if headers[i].startswith('x')]
    z_cols = [i for i in range(len(headers)) if headers[i] == 'z']

    x = np.loadtxt(csv_path, delimiter=',', skiprows=1, usecols=x_cols, dtype=float)
    z = np.loadtxt(csv_path, delimiter=',', skiprows=1, usecols=z_cols, dtype=float)

    if z.ndim == 1:
        z = np.expand_dims(z, axis=-1)

    return x, z


if __name__ == '__main__':
    np.random.seed(229)
    # Run NUM_TRIALS trials to see how different initializations
    # affect the final predictions with and without supervision
    for t in range(NUM_TRIALS):
        main(is_semi_supervised=False, trial_num=t)

        # *** START CODE HERE ***
        # Once you've implemented the semi-supervised version,
        # uncomment the following line.
        # You do not need to add any other lines in this code block.

        # main(is_semi_supervised=True, trial_num=t)

        # *** END CODE HERE ***
```