```python
import os
import scipy.io
import numpy as np
import scipy.linalg
import matplotlib.pyplot as plt

# Load training data from MAT file
R = scipy.io.loadmat('movie_data/movie_train.mat')['train']

# Load validation data from CSV
val_data = np.loadtxt('movie_data/movie_validate.txt', dtype=int, delimiter=',')

# Helper method to get training accuracy
def get_train_acc(R, user_vecs, movie_vecs):
    num_correct, total = 0, 0
    for i in range(R.shape[0]):
        for j in range(R.shape[1]):
            if not np.isnan(R[i, j]):
                total += 1
                if np.dot(user_vecs[i], movie_vecs[j])*R[i, j] > 0:
                    num_correct += 1
    return num_correct/total

# Helper method to get validation accuracy
def get_val_acc(val_data, user_vecs, movie_vecs):
    num_correct = 0
    for val_pt in val_data:
        user_vec = user_vecs[val_pt[0]-1]
        movie_vec = movie_vecs[val_pt[1]-1]
        est_rating = np.dot(user_vec, movie_vec)
        if est_rating*val_pt[2] > 0:
            num_correct += 1
    return num_correct/val_data.shape[0]

# Helper method to get indices of all rated movies for each user,
# and indices of all users who have rated that title for each movie
def get_rated_idxs(R):
    user_rated_idxs, movie_rated_idxs = [], []
    for i in range(R.shape[0]):
        user_rated_idxs.append(np.argwhere(~np.isnan(R[i, :])).reshape(-1))
    for j in range(R.shape[1]):
        movie_rated_idxs.append(np.argwhere(~np.isnan(R[:, j])).reshape(-1))
    return np.array(user_rated_idxs), np.array(movie_rated_idxs)

# Part (c): SVD to learn low-dimensional vector representations
def svd_lfm(R):

    r = np.copy(R)

    # Fill in the missing values in R
    r[np.isnan(r)] = 0

    # Compute the SVD of R
    U, s, Vh = scipy.linalg.svd(r, full_matrices = False)

    # Construct user and movie representations
    user = U
    movie = np.diag(s) @ Vh


    user_vecs = user
    movie_vecs = movie.T
    return user_vecs, movie_vecs

# Part (d): Compute the training MSE loss of a given vectorization
# def get_train_mse_old(R, user_vecs, movie_vecs):
```

```
#     # Compute the training MSE loss
#     mse_loss = 0
#     count = 0
#     for i in range(user_vecs.shape[0]):
#         for j in range(movie_vecs.shape[0]):
#             if not np.isnan(R[i, j]):
#                 dots = user_vecs[i].dot(movie_vecs[j])
#                 mse_loss += (dots - R[i, j]) ** 2
#                 count += 1

#     return mse_loss / count

#Part (d): Compute the training MSE loss of a given vectorization
def get_train_mse(R, user_vecs, movie_vecs):

    UDVT = user_vecs @ movie_vecs.T

    r = np.copy(R)
    k = np.isnan(r)
    r[k] = 0
    UDVT[k] = 0
    mse_loss = np.linalg.norm(r - UDVT) ** 2

    return mse_loss

# Part (e): Compute training MSE and val acc of SVD LFM for various d
d_values = [2, 5, 10, 20]
train_mses, train_accs, val_accs = [], [], []
user_vecs, movie_vecs = svd_lfm(np.copy(R))
for d in d_values:
    train_mses.append(get_train_mse(np.copy(R), user_vecs[:, :d], movie_vecs[:, :d]))
    train_accs.append(get_train_acc(np.copy(R), user_vecs[:, :d], movie_vecs[:, :d]))
    val_accs.append(get_val_acc(val_data, user_vecs[:, :d], movie_vecs[:, :d]))
plt.clf()
plt.plot([str(d) for d in d_values], train_mses, 'o-')
plt.title('Train MSE of SVD-LFM with Varying Dimensionality')
plt.xlabel('d')
plt.ylabel('Train MSE')
plt.savefig(fname='train_mses.png', dpi=600, bbox_inches='tight')
plt.clf()
plt.plot([str(d) for d in d_values], train_accs, 'o-')
plt.plot([str(d) for d in d_values], val_accs, 'o-')
plt.title('Train/Val Accuracy of SVD-LFM with Varying Dimensionality')
plt.xlabel('d')
plt.ylabel('Train/Val Accuracy')
plt.legend(['Train Accuracy', 'Validation Accuracy'])
plt.savefig(fname='trval_accs.png', dpi=600, bbox_inches='tight')

print(train_mses)
print(train_accs)
print(val_accs)


# Part (f): Learn better user/movie vector representations by minimizing loss
best_d = 10 #d_values[np.argmax(val_accs)] #TODO(f): Use best from part (e)
np.random.seed(20)
user_vecs = np.random.random((R.shape[0], best_d))
movie_vecs = np.random.random((R.shape[1], best_d))
user_rated_idxs, movie_rated_idxs = get_rated_idxs(np.copy(R))

def get_A(user_vecs, movie_vecs, R):

    UDVT = user_vecs @ movie_vecs.T

    r = np.copy(R)
    k = np.isnan(r)
    r[k] = 0
```

```
    UDVT[k] = 0

    A = (r - UDVT)
    print(A.shape)
    return A


# Part (f): Function to update user vectors
# def update_user_vecs(user_vecs, movie_vecs, R, user_rated_idxs):

#     U = user_vecs
#     V = movie_vecs

#     R[np.isnan(R)] = 0
#     user_vecs_new = R @ V @ np.linalg.inv((V.T @ V) + np.eye(V.shape[1]))
#     print(np.mean(user_vecs_new), np.std(user_vecs_new))
#     return user_vecs_new

# # Part (f): Function to update user vectors
# def update_movie_vecs(user_vecs, movie_vecs, R, movie_rated_idxs):

#     # Update movie_vecs to the loss-minimizing value
#     U = user_vecs
#     V = movie_vecs

#     R[np.isnan(R)] = 0
#     movie_vecs_new = R.T @ U / 2

#     return movie_vecs_new

def update_user_vecs(user_vecs, movie_vecs, R, user_rated_idxs):
    user_vecs_new = np.zeros(user_vecs.shape)
    y = movie_vecs
    x = user_vecs


    for i in range(x.shape[0]):
        l = y.shape[1]
        youter = np.eye(l)
        Ry = np.zeros(y.shape[1])


        for j in user_rated_idxs[i]:
            youter += np.outer(y[j], y[j])
            Ry += R[i, j] * y[j].T

        user_vecs_new[i] = np.linalg.inv(youter) @ (Ry)


    return user_vecs_new

def update_movie_vecs(user_vecs, movie_vecs, R, movie_rated_idxs):

    movie_vecs_new = np.zeros(movie_vecs.shape)
    y = movie_vecs
    x = user_vecs


    for j in range(y.shape[0]):
        l = x.shape[1]
        xouter = np.eye(l)
        Rx = np.zeros(x.shape[1])


        for i in movie_rated_idxs[j]:
            xouter += np.outer(x[i], x[i])
            Rx += R[i, j] * x[i].T
```

```
        movie_vecs_new[j] = np.linalg.inv(xouter) @ (Rx)



    return movie_vecs_new

# Part (f): Perform loss optimization using alternating updates
train_mse = get_train_mse(np.copy(R), user_vecs, movie_vecs)
train_acc = get_train_acc(np.copy(R), user_vecs, movie_vecs)
val_acc = get_val_acc(val_data, user_vecs, movie_vecs)
print(f'Start optim, train MSE: {train_mse:.2f}, train accuracy: {train_acc:.4f}, val accuracy: {val_acc:.4f}')
for opt_iter in range(20):
    user_vecs = update_user_vecs(user_vecs, movie_vecs, np.copy(R), user_rated_idxs)
    movie_vecs = update_movie_vecs(user_vecs, movie_vecs, np.copy(R), movie_rated_idxs)
    train_mse = get_train_mse(np.copy(R), user_vecs, movie_vecs)
    train_acc = get_train_acc(np.copy(R), user_vecs, movie_vecs)
    val_acc = get_val_acc(val_data, user_vecs, movie_vecs)
    print(f'Iteration {opt_iter+1}, train MSE: {train_mse:.2f}, train accuracy: {train_acc:.4f}, val accuracy: {val_acc:.4f}')
```