The Matlab codes for this problem are as follows:

```matlab
%% load and split data
data = load('u_data.txt');
data = data(:,1:3); % remove timestamp
[M_train, M_test] = split_data(data, 0.2); % randomly split data
E_train = spones(M_train);
E_test = spones(M_test);

%% simple average
rmse1 = zeros(2,1);
for mode = 1 : 2 % movie mean or user mean
    pred = simple_avg(M_train, mode);
    rmse1(mode) = norm((pred - M_test).*E_test, 'fro') / sqrt(nnz(M_test));
end

function [M_train, M_test] = split_data(data, alpha)
% This function randomly splits the input data to the training set and
% test set, according to the test data ratio alpha.
% Input:
%%% data: the n*3 raw data (row, column, value)
%%% alpha: the proportion of test data (=0.2 typically)
% Output:
%%% M_train: the sparse matrix containing training data
%%% M_test: the sparse matrix containing test data

    test_idx = (rand(length(data),1) < alpha);
    train_idx = ~test_idx;
    M_train = sparse(data(train_idx,1),data(train_idx,2),data(train_idx,3),...  
                 max(data(:,1)),max(data(:,2)));
    M_test = sparse(data(test_idx,1),data(test_idx,2),data(test_idx,3),... 
                   max(data(:,1)),max(data(:,2)));

function pred = simple_avg(M_train, mode)
% Prediction using simple average (over movies/users)
% Input:
%%% M_train: training data, a sparse matrix
%%% mode: 1 = movie mean, 2 = user mean
% Output:
%%% pred: the resulting full matrix completion

    [len, wid] = size(M_train);
    E_train = spones(M_train); % indicator matrix of nonzero positions
    M_train_full = full(M_train);
    E_train_full = full(E_train);
    avg_all = sum(sum(M_train)) / nnz(M_train);
    avg_mode = sum(M_train_full, mode) ./ sum(E_train_full, mode);
    avg_mode(isnan(avg_mode)) = avg_all; % in case some movie/user has no ranking
```
if mode == 1
    pred = ones(len,1) * avg_mode;
else
    pred = avg_mode * ones(1,wid);
end

Based on the code, the RMSE for the movie mean is 1.0258, while that for the user mean is 1.0437. This result conforms to the usual study that the movie (item) mean is usually better than the user mean.
Solution to Part (b).

The Matlab code for this part is as follows:

```matlab
%% svd projection
r_set = [5,10,15];
alpha_set = 0:0.02:0.8;
rmse2 = zeros(length(r_set),length(alpha_set));

M_avg = simple_avg(M_train(:,1)); % movie mean
M_train_new = (M_train - M_avg).*E_train;
for iter_r = 1 : length(r_set)
    for iter_alpha = 1 : length(alpha_set)
        pred = svd_projection(M_train_new,r_set(iter_r));
        pred = M_avg + pred / alpha_set(iter_alpha);
        pred(pred>5) = 5; pred(pred<1) = 1; % truncation
        rmse2(iter_r,iter_alpha) = norm( (pred - M_test).*E_test, 'fro' ) ...
                                      / sqrt(nnz(M_test));
    end
end

function pred = svd_projection(M_train, r)
% Prediction using projection to rank-r matrices
% Input:
% M_train: the sparse training matrix
% r: rank
% alpha: regularizer
% Output:
% pred: the resulting full matrix completion
[X,S,Y] = svds(M_train,r); % rank-r sparse svd
pred = X*S*Y'; % the predictor
```

Specifically, we choose the rank $r \in \{5, 10, 15\}$ and $\alpha \in \{0.02, 0.04, \cdots, 0.8\}$, and the RMSEs are shown in the following figure.

![Figure 1. RMSEs based on simple SVD for different rank r and scaling factor alpha](image-url)
We can observe from the figure that the best choices are \( r = 10 \) and \( \alpha = 0.5 \). In theory, we know that as \( r \) increases, the bias decreases while the variance increases, and empirically we conclude that \( r = 10 \) may be a good choice. As for \( \alpha \), in theory the unbiasedness requirement yields to

\[
\alpha = \frac{|\Omega|}{mn} = \frac{80,000}{943 \times 1,682} \approx 0.05
\]

which is considerably smaller than the empirical one \( \alpha = 0.5 \). It means that for \( \alpha = 0.05 \), the variance is still too large and we need to increase \( \alpha \) considerably to overcome it.

Finally, we remark that a common approach in reducing the variance in spectral initialization is to use trimming, i.e., remove dense rows and columns to alleviate their influence on SVD. However, in this experiment we found that trimming contributes to little performance improvement (also in Part (c) and Part (d)), so throughout we do not use it.
Solution to Part (c).

The Matlab code is as follows.

```matlab
clear; clc; close all;

m = 943; n = 1682; r = 20;
niter = 20; % number of MC iterations
delta_set = 0.05:0.05:0.95;
rmse = zeros(niter, length(delta_set));
for iter = 1 : niter
    U = 2 * floor(2*rand(m,r)) - 1;
    V = 2 * floor(2*rand(n,r)) - 1;
    X = U * V' / 2;
    for delta_iter = 1 : length(delta_set)
        E = 1 - ceil(rand(m,n) - delta_set(delta_iter));
        Y = sparse(X.*E) / delta_set(delta_iter); % partial revealed matrix
        pred = svd_projection(Y,r);
        rmse(iter, delta_iter) = norm(pred - X, 'fro') / sqrt(m*n);
    end
end
```

The RMSE as a function of the coverage $\delta$ is shown as follows.

![Image](link_to_the_image)

**Figure 2.** RMSEs based on simple SVD with different coverages $\delta$ on synthetic data

Compared with the real data, we see that the RMSE of simple SVD on synthetic data is typically much smaller (already < 1 for $\delta \geq 0.2$), and tends to zero as $\delta$ becomes large. This is because that the synthetic data is exactly low-rank and noise-free, while the real data may be not and quite noisy. Moreover, the plot differs from the theoretical upper bound that when $\delta \gtrsim \frac{\log(m\vee n)}{m\vee n} \sim 0.1$, the upper bound on the RMSE should decay like $c\delta^{-\frac{1}{2}}$ in theory, while it is more or less like $c\delta^{-1}$ in the plot.
Solution to Part (d).

To minimize the target function, we use alternate minimization. Firstly, we initialize $X = X_0 \in \mathbb{R}^{m \times r}$ to be the matrix consisting of left singular vectors in the rank-$r$ SVD of $M$. Now given $X_0$, we seek

$$Y_0 = \arg \min_Y L(X_0, Y) = \arg \min_Y \|P_\Omega(M - X_0 Y^T)\|_F^2 + \lambda \|X_0\|_2^2 + \lambda \|Y\|_F^2.$$ 

For each $j = 1, \ldots, n$, let $\Omega_j \triangleq \{ i \in [m] : (i, j) \in \Omega \}$, and let $X_{0,\Omega_j} \in \mathbb{R}^{r \times |\Omega_j|}$ is the submatrix of $X_0$ consisting of rows in $\Omega_j$ and all columns. Moreover, let $m_j \in \mathbb{R}^{|\Omega_j|}$ be the vector consisting of all nonzero elements of $M$ in column $j$, and $y_1^T, \ldots, y_n^T$ be all row vectors of $Y$. Then it’s easy to obtain

$$L(X_0, Y) = \lambda \|X_0\|_2^2 + \sum_{j=1}^n (\|m_j - X_{0,\Omega_j} y_j\|_F^2 + \lambda \|y_j\|_F^2)$$

i.e., different $y_j$’s are decoupled. Now it is immediate to conclude that the minimizing $Y_0$ is obtained by

$$y_{0,j} = (X_{0,\Omega_j}^T X_{0,\Omega_j} + \lambda I_r)^{-1} X_{0,\Omega_j}^T m_j, \quad j = 1, \ldots, n.$$ 

Similarly, we can obtain $X_1, Y_1, X_2, \ldots$ sequentially, and arrive at the following Matlab program:

```matlab
function [pred, rmse] = alternate_minimization(M_train, r, lambda, M_avg, M_test)
% Prediction using alternate minimization and RMSE computation
% Input:
%%% M_train: the sparse training matrix (with movie mean subtracted)
%%% r: the rank used
%%% lambda: regularizer
%%% M_avg: the movie mean (only for RMSE evaluation purpose)
%%% M_test: the test matrix (only for RMSE evaluation purpose)
% Output:
%%% pred: our predictor
%%% RMSE: the RMSEs after 100 iterations

%% svd initialization
[X, ~] = svds(M_train, r); % rank-r sparse svd

%% alternate minimization
rmse = zeros(100,1); % default: 100 iterations
for iter = 1 : 100
    Y = least_squares_Y(M_train, X, lambda);
    X = least_squares_X(M_train, Y, lambda);
    pred = X * Y';
    rmse(iter) = evaluate_rmse(pred, M_avg, M_test);
    fprintf(1,'\%d\t\%f\n',iter,rmse(iter));
end

end
```

```matlab
%% alternating minimization
lambda_set = 5:5:30;
rmse3 = zeros(100, length(lambda_set));
for lambda_iter = 1 : length(lambda_set)
    [~, rmse3(:,lambda_iter)] = alternate_minimization(M_train_new,10,...
        lambda_set(lambda_iter),M_avg,M_test);
end
```

```matlab
function rmse = evaluate_rmse(pred, M_avg, M_test)
% Evaluation of RMSE
% Input:
%%% pred: the predictor to be evaluated
```
E_test = spones(M_test);
pred = M_avg + pred;  % add movie mean back
pred(pred>=5) = 5; pred(pred<=1) = 1;  % truncation
rmse = norm((pred - M_test).*E_test, 'fro')/ sqrt(nnz(M_test));

end

function Y = least_squares_Y(M, X, lambda)
% Minimizer of Y to the quadratic program
% Input:
% %% M: the partially observed sparse matrix
% %% X: the given X in the last iteration
% %% lambda: regularizer
% Output:
% %% Y: the minimizer to the program
M_full = full(M);
[wid, r] = size(M);
Y = zeros(wid,r);
for col = 1 : wid  % solve each row vector of Y
    pos = (M_full(:,col) ~= 0);
    M_new = M_full(pos,col);
    X_new = X(pos,:);
    if ~any(pos)
        Y(col,:) = zeros(1,r);
    else
        Y(col,:) = ((lambda*eye(r)+X_new'*X_new)\(X_new'*M_new))';
    end
end

end

function X = least_squares_X(M, Y, lambda)
% Minimizer of X to the quadratic program
% Input:
% %% M: the partially observed sparse matrix
% %% Y: the given Y in the last iteration
% %% lambda: regularizer
% Output:
% %% X: the minimizer to the program
M_full = full(M);
[len, r] = size(M);
[Y, r] = size(Y);
X = zeros(len,r);
for row = 1 : len  % solve each row vector of X
    pos = (M_full(row,:) ~= 0);
    M_new = M_full(row,pos)';
    Y_new = Y(pos,:);
    if ~any(pos)
        X(row,:) = zeros(1,r);
    else
        X(row,:) = {{lambda*eye(r)+Y_new'*Y_new}\(Y_new'*M_new)}';
    end
end
The prediction RMSE after $t \in \{1, \cdots, 100\}$ steps for $\lambda \in \{5, 10, \cdots, 30\}$ is as follows:

![Figure 3. Prediction RMSE for alternate minimization with different regularization $\lambda$](image)

We can observe from the figure that for small $\lambda$, the error first decreases and then increases. However, as $\lambda$ increases, the error curve tends to be flat. This is because large $\lambda$ enforces stronger regularization, and thus makes it hard to overfit the training data. Moreover, compared with the simple SVD, this alternate minimization approach enjoys a much smaller error just after a few iterations. We can also see that $\lambda = 10$ results in the optimal test error, which attains an RMSE 0.9397 after 4 iterations.

A drawback of this approach is that when there is no regularization, $\mathcal{L}(X, Y)$ in fact only depends on the subspace spanned by $X, Y$ and invariant with suitable right multiplications of them: $\mathcal{L}(X, Y) = \mathcal{L}(XS, YS^{-1})$ for any $S \in GL_r(\mathbb{R})$. As a result, $\mathcal{L}(X, Y)$ can in fact be defined on the Grassmann manifold $G(m, r) \times G(n, r)$ (e.g., $\mathcal{L}(X, Y) \triangleq \min_{S \in \mathbb{R}^{r \times r}} \|P_{\Omega}(M - XSY)\|_F^2$), and the corresponding optimization may be more efficient. Moreover, the explicit regularization can also be replaced by some implicit regularization with a careful initialization (e.g., with trimming). Based on this idea and the Matlab code I implemented for a previous course, we can actually achieve a better RMSE $< 0.92$ on this dataset.