Table of Contents


```
% Filtering and Identification - final assignment
% Section 1 - Random Walk Model

close all;

load systemMatrices.mat;
load turbulenceData.mat; % load the data provided
```

Question 1

```
% We see that we have been provided data corresponding to s0(k) and G,
 and
% we have to determine the value of phi(k) such that the error, e(k),
 is
% minimum, given no prior information. This is clearly a linear least
% squares problem.

rank_G = rank(G);

% We get the rank of G as 47, whereas the number of columns in G is 49
 and
% the number of rows is 72. Hence, G is not full rank, we conclude.

% Getting the SVD of G:
[U, S, V] = svd(G, 0);
```

Question 5

```
% In Question 5, we need to know the rank of the system matrix H so as
 to
% determine a solution for the linear least squares problem minimizing
 the
% 2-norm of epsilon (k + 1|k)
rank_H = rank(H);

% We get the rank of H as 49, and hence we conclude that H is a full
 rank,
% invertible matrix.
```

Question 6

```
% The function AOloopRW is present in the same directory as the
 current
% one.

% Considering the first cell in the given cell array
phiSim_1 = phiSim{1,1};

% Calculating the dimensions of each phiSim cell
[phi_len, n] = size(phiSim_1);

% Covariance matrix is square
covariance_phi = zeros(phi_len, phi_len);

% We are provided with a cell array of 20 datasets for phiSim
num_Datasets = length(phiSim);

variance_closed_loop = 0;
variance_no_control = 0;

for cellIndex = 1:num_Datasets
    phi_currentCell = phiSim{1,cellIndex};
    for k = 1:n
        covariance_phi = covariance_phi +
 (phi_currentCell(:,k)*phi_currentCell(:,k)');
    end

    covariance_phi = covariance_phi/n;

    variance_closed_loop = variance_closed_loop + AOloopRW(G,H,
 covariance_phi, sigmae, phi_currentCell);
    variance_no_control = variance_no_control +
 AOloop_nocontrol(phi_currentCell,sigmae,H,G);
end

% Taking the average of the values obtained from all of the provided
% datasets
variance_closed_loop = variance_closed_loop/num_Datasets;
variance_no_control = variance_no_control/num_Datasets;
```

Question 7

```
VAF_cumulative = 0;

for cellIndex = 1:num_Datasets
    phi_currentCell = phiSim{1,cellIndex};
    covariance_phi = zeros(phi_len, phi_len);
    for k = 1:n
        covariance_phi = covariance_phi +
 (phi_currentCell(:,k)*phi_currentCell(:,k)');
    end
```

```
    covariance_phi = covariance_phi/n;

    VAF_cumulative = VAF_cumulative + VAF_RW(G,H, covariance_phi,
 sigmae, phi_currentCell);
end

VAF = VAF_cumulative/num_Datasets;
```

Published with MATLAB® R2018a

AOloopRW function

```
function [var_eps] = AOloopRW(G,H, covariance_phi, sigma_e, phi_sim)
% Variance calculation of an AO system in the closed-loop
 configuration
% with the Random Walk model used for calculations
% IN
% G     : measurement matrix
% H     : influence matrix mapping the wavefront on the mirror
% covariance_phi  : covariance matrix of the turbulence wavefront
% sigma_e : measurement noise parameter for determining its covariance
% phi_sim : simulation data for the wavefront
% OUT
% var_eps : variance of the residual wavefront after taking N_t points
% within the closed-loop operation

% dimension lifted wavefront
n_H = size(H,1);

% dimension lifted sensor slopes
n_G = size(G,1);

% Number of sample points for phi_sim
T = length(phi_sim);

u = zeros(n_H,T);

% This term is multiplied with the slope vector to obtain the
 eps_hat(k|k)
% value
eps_pred_multiplier_matrix = (H'*H)\H'*(covariance_phi*G'/
(G*covariance_phi*G' + (sigma_e^2)*eye(n_G)));

% epsilon matrix
eps_k = zeros(size(phi_sim,1),T);

% epsilon matrix with mean removed:
eps_mean_removed_k = zeros(size(phi_sim,1),T);

% Constructing a vector of all the variance values for eps
var_eps = zeros(T,1);

% An assumption is that H is full rank.

% We have the data for phi_k, and we know the matrix H.
% Since we don't have the values for the slopes s(k), we compute the
 vector as:
% s(k) = G*phi_k - G*H*u(k-1) + sigma_e*randn(n_G,1)
% We use this s(k) in the expression for the optimal increment u(k) -
% u(k-1), and calculate the values of the u(k) vector recursively,
 since we
% know u(0) = 0.
```

```
% Based on the calculated u(k), we calculate the value of
% epsilon and then accordingly its variance.

% u(0) is 0 since we don't apply any control input before the first
 wavefront datum,
% and hence we calculate u(1) taking u(0) = 0
u(:,1) = eps_pred_multiplier_matrix*(G*phi_sim(:,1) +
 sigma_e*randn(n_G,1));
eps_k(:,1) = phi_sim(:,1);

eps_mean_removed_k(:,1) = eps_k(:,1) - mean(eps_k(:,1));

var_eps(1) = var(eps_mean_removed_k(:,1));

for k = 2:T
    eps_k(:,k) = phi_sim(:,k) - H*u(:,k-1);
    u(:,k) = eps_pred_multiplier_matrix*(G*eps_k(:,k) +
 sigma_e*randn(n_G,1)) + u(:,k-1);
    eps_mean_removed_k(:,k) = eps_k(:,k) - mean(eps_k(:,k));
    var_eps(k) = var(eps_mean_removed_k(:,k));
end

var_eps = mean(var_eps);

end
```

Published with MATLAB® R2018a

VAF calculation function for the random walk model

```
function [VAF] = VAF_RW(G,H, covariance_phi, sigma_e, phi_sim)
% VAF calculation
% IN
% G     : measurement matrix
% H     : influence matrix mapping the wavefront on the mirror
% covariance_phi  : covariance matrix of the turbulence wavefront
% sigma_e : measurement noise parameter for determining its covariance
% phi_sim : simulation data for the wavefront
% OUT
% VAF : VAF value

% dimension lifted wavefront
n_H = size(H,1);

% dimension lifted sensor slopes
n_G = size(G,1);

% Number of sample points for phi_sim
T = length(phi_sim);

u = zeros(n_H,T);

% The following term is multiplied with the slope vector to obtain the
 eps_hat(k|k)
% value
eps_pred_multiplier_matrix = H\(covariance_phi*G'/(G*covariance_phi*G'
 + (sigma_e^2)*eye(n_G)));

% epsilon matrix
eps_k = zeros(size(phi_sim,1),T);

% epsilon matrix with mean removed:
eps_mean_removed_k = zeros(size(phi_sim,1),T);

% An assumption is that H is full rank.

% We have the data for phi_k, and we know the matrix H.
% Since we don't have the values for the slopes s(k), we compute the
 vector as:
% s(k) = G*phi_k - G*H*u(k-1) + sigma_e*randn(n_G,1)
% We use this s(k) in the expression for the optimal increment u(k) -
% u(k-1), and calculate the values of the u(k) vector recursively,
 since we
% know u(0) = 0.
% Based on the calculated u(k), we calculate the value of
% epsilon and then accordingly its variance.
```

```
% u(0) is 0 since we don't apply any control input before the first
 wavefront datum,
% and hence we calculate u(1) taking u(0) = 0
u(:,1) = eps_pred_multiplier_matrix*(G*phi_sim(:,1) +
 sigma_e*randn(n_G,1));
eps_k(:,1) = phi_sim(:,1);

eps_mean_removed_k(:,1) = eps_k(:,1) - mean(eps_k(:,1));

% Calculation of variance accounted for:

deviation_norm_sum = 0;
actual_phi_norm_sum = 0;

for k = 2:T-1
    eps_k(:,k) = phi_sim(:,k) - H*u(:,k-1);
    slope_k = G*eps_k(:,k) + sigma_e*randn(n_G,1);
    u(:,k) = eps_pred_multiplier_matrix*(slope_k) + u(:,k-1);
    %eps_mean_removed_k(:,k) = eps_k(:,k) - mean(eps_k(:,k));

    predicted_phi = H*eps_pred_multiplier_matrix*slope_k + H*u(:,k-1);
    predicted_phi = predicted_phi - mean(predicted_phi);
    phi_sim_mean_removed = phi_sim(:,k+1) - mean(phi_sim(:,k+1));
    current_norm = (norm(phi_sim_mean_removed - predicted_phi))^2;
    deviation_norm_sum = deviation_norm_sum + current_norm;
    actual_phi_norm_sum = actual_phi_norm_sum +
 (norm(phi_sim_mean_removed))^2;
end

mean_deviation_norm = deviation_norm_sum/(T-2);
mean_actual_norm = actual_phi_norm_sum/(T-2);
VAF = max(0,100*(1 - mean_deviation_norm/mean_actual_norm));

end
```

Published with MATLAB® R2018a

```
% Filtering and Identification - final assignment
% Section 2 - VAR model

close all;

load systemMatrices.mat;
load turbulenceData.mat; % load the data provided

phi_currentCell = phiSim{1,1};
[phi_len, n] = size(phi_currentCell);

% Total number of given datasets with simulated phi values
num_Datasets = length(phiSim);
```

Question 7

```
var_KalmanAR = 0;
VAF_Kalman = 0;

for cellIndex = 1:num_Datasets
    phi_currentCell = phiSim{1,cellIndex};

    % Given definition for C_phi(0)
    covariance_phi_0 = zeros(phi_len, phi_len);
    for k = 1:n
        covariance_phi_0 = covariance_phi_0 +
 (phi_currentCell(:,k)*phi_currentCell(:,k)');
    end

    covariance_phi_0 = covariance_phi_0/n;

    covariance_phi_1 = zeros(phi_len, phi_len);

    % Given definition for C_phi(1)
    for k = 2:n
        covariance_phi_1 = covariance_phi_1 +
 (phi_currentCell(:,k)*phi_currentCell(:,k-1)');
    end

    covariance_phi_1 = covariance_phi_1/(n-1);

    % Computing the values for the Kalman filter
    [A, Cw, K] = computeKalmanAR(covariance_phi_0, covariance_phi_1,
 G, sigmae);
    % Computing the variance for the current dataset
    var_KalmanAR = var_KalmanAR + AOloopAR(G,H, covariance_phi_0,
 sigmae, A, Cw, K, phi_currentCell);

    %Computing the VAF values
    VAF_Kalman = VAF_Kalman + VAF_VAR(G,H, sigmae, A, K,
 phi_currentCell);
end
```

```
% Taking the average values over 20 datasets
var_KalmanAR = var_KalmanAR/num_Datasets;
VAF_Kalman = VAF_Kalman/num_Datasets;
```

Question 8

```
VAF_Kalman
```

Published with MATLAB® R2018a

computeKalmanAR function

```
function [A, Cw, K] = computeKalmanAR(C_phi_0, C_phi_1, G, sigma_e)

%COMPUTEKALMANAR Computes parameters for the Kalman filter in the VAR
 model
%   IN
% C_phi_0 : The auto-covariance matrix for the given phi dataset
% C_phi_1 : The C_phi(1) matrix for the given phi dataset
% G : The G matrix relating epsilon(k) to s(k)
% sigma_e : SD of the noise vector

% OUT
% A : State matrix defining the relation between the current and next
% states
% Cw : Covariance matrix for process noise
% K : Stationary Kalman filter gain

phi_len = size(C_phi_0, 1);
A = C_phi_1/C_phi_0;

Cw = (eye(size(A)) - A*A)*C_phi_0;

% For the DARE, we need the Q matrix to be symmetric. So we take the
% average of the Cw matrix and its transpose
Cw = (Cw + Cw')/2;

R = (sigma_e^2)*eye(size(G,1));

E = eye(phi_len);
S = zeros(size(G'));

% Making use of the DARE provided by MATLAB to compute the covariance
% matrix for epsilon
P = dare(A',G',Cw,R,S,E);

% Kalman gain
K = A*P*G'/(G*P*G' + R);

end
```

Published with MATLAB® R2018a

AOLoopAR function

```
function [var_eps] = AOloopAR(G,H, covariance_phi, sigma_e, A, Cw, K,
 phi_sim)
% Variance calculation of an AO system in the closed-loop
 configuration
% with a Kalman filter involved
% IN
% G     : measurement matrix
% H     : influence matrix mapping the wavefront on the mirror
% covariance_phi  : covariance matrix of the turbulence wavefront
% sigma_e : measurement noise parameter for determining its covariance
% A : A matrix from the state-space model
% Cw : Covariance matrix of the process noise
% K : Stationary Kalman filter gain
% phi_sim : simulation data for the wavefront
% OUT
% var_eps : variance of the residual wavefront after taking N_t points
% within the closed-loop operation

% dimension lifted wavefront
n_H = size(H,1);

% dimension lifted sensor slopes
n_G = size(G,1);

% Number of sample points for phi_sim
T = length(phi_sim);

u = zeros(n_H,T);

% epsilon matrix
eps_k = zeros(size(phi_sim,1),T);

eps_kplus1k = zeros(size(phi_sim,1),T);

% epsilon matrix with mean removed:
eps_mean_removed_k = zeros(size(phi_sim,1),T);

% Constructing a vector of all the variance values for eps
var_eps = zeros(T,1);

% An assumption is that H is full rank.

% s(k) = G*eps(k) + e(k)
% Hence s(k) = G*phi(k) - G*H*u(k-1) + e(k)

% u(k) = 0 for k < 1

% eps(k+1|k) = Ks(k) + A*H*u(k-1) - H*u(k) + (A - KG)eps(k|k-1)
```

```
% eps(k+1|k) = K*G*phi(k) - K*G*H*u(k-1) + K*e(k) + A*H*u(k-1) -
 H*u(k) + (A - KG)eps(k|k-1)

% First slope value, no input applied before
s_k = G*phi_sim(:,1) + sigma_e*randn(n_G,1);

% Initial input
u(:,1) = H\(K*s_k);

% eps(1|0) = 0;
%eps_kplus1k = eps(k+1|k)
eps_kplus1k(:,1) = K*G*phi_sim(:,1) + K*sigma_e*randn(n_G,1) -
 H*u(:,1);

% Initial eps(k)
eps_k(:,1) = phi_sim(:,1);
eps_mean_removed_k(:,1) = eps_k(:,1) - mean(eps_k(:,1));

var_eps(1) = var(eps_mean_removed_k(:,1));

for k = 2:T
    % Slope value
    s_k = G*phi_sim(:,k) - G*H*u(:,k-1) + sigma_e*randn(n_G,1);

    % Optimum input value
    u(:,k) = H\(K*s_k + (A - K*G)*eps_kplus1k(:,k-1) + A*H*u(:,k-1));

    % Next eps(k+1|k) value
    eps_kplus1k(:,k) = K*s_k + A*H*u(:,k-1) - H*u(:,k) + (A -
 K*G)*eps_kplus1k(:,k-1);

    % eps(k) value = phi(k) - Hu(k-1)
    eps_k(:,k) = phi_sim(:,k) - H*u(:,k-1);
    eps_mean_removed_k(:,k) = eps_k(:,k) - mean(eps_k(:,k));
    var_eps(k) = var(eps_mean_removed_k(:,k));
end

var_eps = mean(var_eps);

end
```

Published with MATLAB® R2018a

VAF calculation function for the VAR model

```
function [VAF] = VAF_VAR(G,H, sigma_e, A, K, phi_sim)
% VAF calculation
% IN
% G     : measurement matrix
% H     : influence matrix mapping the wavefront on the mirror
% sigma_e : measurement noise parameter for determining its covariance
% A : A matrix from the state space model
% K : Kalman gain value
% phi_sim : simulation data for the wavefront
% OUT
% VAF : VAF value

% dimension lifted wavefront
n_H = size(H,1);

% dimension lifted sensor slopes
n_G = size(G,1);

% Number of sample points for phi_sim
T = length(phi_sim);

u = zeros(n_H,T);

% epsilon matrix
eps_k = zeros(size(phi_sim,1),T);

% An assumption is that H is full rank.

% s(k) = G*eps(k) + e(k)
% Hence s(k) = G*phi(k) - G*H*u(k-1) + e(k)

% u(k) = 0 for k < 1

% eps(k+1|k) = Ks(k) + A*H*u(k-1) - H*u(k) + (A - KG)eps(k|k-1)

% eps(k+1|k) = K*G*phi(k) - K*G*H*u(k-1) + K*e(k) + A*H*u(k-1) -
 H*u(k) + (A - KG)eps(k|k-1)

% First slope value, no input applied before
s_k = G*phi_sim(:,1) + sigma_e*randn(n_G,1);

% Initial input
u(:,1) = H\(K*s_k);

% e(1|0) = 0;
eps_kplus1k(:,1) = K*G*phi_sim(:,1) + K*sigma_e*randn(n_G,1) -
 H*u(:,1);

deviation_phi_norm_sum = 0;
actual_phi_norm_sum = 0;
```

```
for k = 2:T-1
    % Slope value
    s_k = G*phi_sim(:,k) - G*H*u(:,k-1) + sigma_e*randn(n_G,1);

    % Optimum input value
    u(:,k) = H\(K*s_k + (A - K*G)*eps_kplus1k(:,k-1) + A*H*u(:,k-1));

    % Next eps(k+1|k) value
    eps_kplus1k(:,k) = K*s_k + A*H*u(:,k-1) - H*u(:,k) + (A -
 K*G)*eps_kplus1k(:,k-1);

    predicted_phi = eps_kplus1k(:,k) + H*u(:,k);
    predicted_phi = predicted_phi - mean(predicted_phi);
    phi_sim_mean_removed = phi_sim(:,k+1) - mean(phi_sim(:,k+1));
    current_norm = (norm(phi_sim_mean_removed - predicted_phi))^2;
    deviation_phi_norm_sum = deviation_phi_norm_sum + current_norm;
    actual_phi_norm_sum = actual_phi_norm_sum +
 (norm(phi_sim_mean_removed))^2;

end

mean_deviation_norm = deviation_phi_norm_sum/(T-2);
mean_actual_norm = actual_phi_norm_sum/(T-2);
VAF = max(0,100*(1 - mean_deviation_norm/mean_actual_norm));

end
```

Published with MATLAB® R2018a

Table of Contents


```
% Filtering and Identification - final assignment
% Section 3 - SI State Space Model
%

close all;

load systemMatrices.mat;
load turbulenceData.mat; % load the data provided
```

Question 5: Sub Identification

```
% The functions SubId and AOloopSID is present in the same directory
 as the current
% one.

% Considering the first cell in the given cell array
phiIdent_1 = phiIdent{1,20}; % Trials with different identification
 datasets

% Calculating the dimensions of each phiSim cell
[phi_len, n] = size(phiIdent_1);

% Size of G matrix
n_G = size(G,1);

% Number of sample points for phi_sim
T = length(phiIdent_1);

% Get sid matrix
sid = G*phiIdent_1 + sigmae*randn(n_G,T);

% Choose s and n paramters
% Chosen using by analysing the SVD using: semilogy(diag(Sigma),'xr');
s = 6;
n = 60;

[As, Cs, Ks] = SubId(sid, s,n); % (Trials with different
 identification datasets)
```

Question 5: Validation

```
phiSim_2 = phiSim{1,6}; % Trials with different simulation datasets
```

```
[var_eps, VAF] = AOloopSID(G,H,As,Cs,Ks,sigmae,phiSim_2);
```

Question 5: Validation over entire dataset

```
% We are provided with a cell array of 20 datasets for phiSim
num_Datasets = length(phiSim);

var_eps_total = 0;
VAF_cumulative = 0;

for cellIndex = 1:num_Datasets
    phi_currentCell = phiSim{1,cellIndex};
    [var_currentCell, VAF_currentCell] =
 AOloopSID(G,H,As,Cs,Ks,sigmae,phi_currentCell);
    var_eps_total = var_eps_total + var_currentCell;
    VAF_cumulative = VAF_cumulative + VAF_currentCell;
end

% Taking the average of the values obtained from all of the provided
% datasets
fprintf("Results:\n");
Variance_avg = var_eps_total/num_Datasets
VAF_avg = VAF_cumulative/num_Datasets
```

Published with MATLAB® R2018a

SubId function

```
function [As, Cs, Ks] = SubId(sid,s,n)
% SubID calculations
% IN
% sid : Slopes of Identification data
% s   : Parameter 's'. Rows of Henkel matrix.
% n   : Order of states 'n'
% OUT
% As : State Space matrix
% Cs : State Space matrix
% Ks : Kalman Gain in Observer form

% Number of sample points for Subspace Identification
N = size(sid,2);

% Dimension of sid
M = size(sid,1);

% Construct Hankel Matrix using sid
Y = [];
sid_stacked = sid(:);

for i=1:N-s+1
    Y = [Y sid_stacked(M*(i-1)+1:M*(s+i-1))];
end

Ypast = Y(:,1:N-2*s+1);
Yfuture = Y(:,s+1:N-s+1);

R = triu(qr([Ypast; Yfuture]'))';
R32 = R(M*s+1:M*2*s,1:M*s);
R22 = R(1:M*s,1:M*s);

[U,Sigma,V] = svd(R32*inv(R22)*Ypast);

% Assigning Sigma for semilogy for choosing n
assignin('base','Sigma',Sigma);

% Reduced System
U = U(:,1:n);
Sigma = Sigma(1:n,1:n);
V = V(:,1:n);

X = sqrt(Sigma)*V';

% Solve least squares problem for X
Xpast = X(:,1:end-1);
Xfuture = X(:,2:end);
Ylsq = sid(:,s+1:end-s);

F = Xpast;
```

```
Sys = (F*(F'))\(F*([Xfuture; Ylsq]'));
Sys = Sys';

As = Sys(1:n,:);
Cs = Sys(n+1:end,:);

% Find K using residuals
Res = [Xfuture; Ylsq] - [As; Cs]*Xpast;
Nt = size(Res,2);
Covar = (Res*Res')/Nt;
Q = Covar(1:n,1:n);
S = Covar(1:n,n+1:end);
R = Covar(n+1:end,n+1:end);

% Choosing P from the solution of the DARE, we can define a Kalman
 gain Ks
% that is asymptotically stable.
% Solution of DARE:
E = eye(n,n);
[P,~,~] = dare(As',Cs',Q,R,S,E);

% Kalman Gain:
Ks = (S + As*P*(Cs'))*inv(R + (Cs*P*(Cs')));

end
```

Published with MATLAB® R2018a

AOloopSID function

```
function [var_eps, VAF] = AOloopSID(G,H,As,Cs,Ks,sigma_e,phi_sim)
% Variance calculation of an AO system in the closed-loop
 configuration,
% using the Subspace Identification model
% IN
% G     : measurement matrix
% H     : influence matrix mapping the wavefront on the mirror
% As    : State matrix for turbulence model
% Cs    : State matrix for turbulence model
% Ks    : Kalman Gain for turbulence model
% sigma_e : measurement noise parameter
% phi_sim : simulation data for the wavefront
% OUT
% var_eps : variance of the residual wavefront after taking N_t points
% within the closed-loop operation
% VAF: Variance accounted for by the turbulence model

% State Dimension
n = size(As,1);

% dimension lifted wavefront
n_H = size(H,1);

% dimension lifted sensor slopes
n_G = size(G,1);

% Number of sample points for phi_sim
T = length(phi_sim);

% Control Input u
u = zeros(n_H,T);

% epsilon matrix
eps_k = zeros(size(phi_sim,1),T);

% epsilon matrix with mean removed:
eps_mean_removed_k = zeros(size(phi_sim,1),T);

% Constructing a vector of all the variance values for eps
var_eps = zeros(T,1);

% VAF
VAF = zeros(T,1);

% An assumption is that H is full rank.

% We use s(k) in the equation to find optimal prediction x(k+1|k).
% the optimal prediction for s(k+1|k) is hence found as Cs*x(k+1|k)
% We can now find prediction for phi_k using the expression derived in
 Part
```

```
% 1.1.
% Hence, a minimizing u(k) can be found by solving a least squares
 problem to
% minimize the predicted residual (epsilon(k+1|k)).
% This u(k) will be used to calculate the actual residual epsilon(k)
 and
% then accordingly its variance.

% u(0) is 0 since we don't apply any control input before the first
% wavefront datum.

eps_k(:,1) = phi_sim(:,1);

% We have the data for phi_sim
% We compute the measurements for s as:
s_sim = G*phi_sim + sigma_e*randn(n_G,T);
%s_current = G*eps_k(:,1) + sigma_e*randn(n_G,1);

% (Taking initial x as zero)
x_current = zeros(n,1);
x_pred = (As-Ks*Cs)*x_current + Ks*(s_sim(:,1));
s_pred = Cs*x_pred;
% We use the same method as 1.1 to get a pseudo inverse of G. This is
% equivalent to pinv()
Ginv = pinv(G);
phi_pred = Ginv*s_pred;

% Finally, calculating input u:
Hterm = inv((H')*H)*(H');
u(:,1) = Hterm*phi_pred;

% Mean removal of epsilon
eps_mean_removed_k(:,1) = eps_k(:,1) - mean(eps_k(:,1));
var_eps(1) = var(eps_mean_removed_k(:,1));

actual_phi_norm_sum = 0; % For VAF calculation
deviation_norm_sum = 0; % For VAF calculation

for k = 2:T-1
    % Calculate actual residual epsilon
    eps_k(:,k) = phi_sim(:,k) - H*u(:,k-1);
    %s_current = G*eps_k(:,k) + sigma_e*randn(n_G,1);

    % Calculate next u based on predicted epsilon
    x_current = x_pred;
    x_pred = (As-Ks*Cs)*x_current + Ks*(s_sim(:,k));
    s_pred = Cs*x_pred;
    phi_pred = Ginv*s_pred;

    u(:,k) = Hterm*phi_pred;

    % Mean removal and Variance
    eps_mean_removed_k(:,k) = eps_k(:,k) - mean(eps_k(:,k));
    var_eps(k) = var(eps_mean_removed_k(:,k));
```

```
    % VAF calculation
    phi_pred_mean_removed = phi_pred - mean(phi_pred);
    phi_sim_mean_removed = phi_sim(:,k+1) - mean(phi_sim(:,k+1));
    current_norm = (norm(phi_sim_mean_removed -
 phi_pred_mean_removed))^2;
    deviation_norm_sum = deviation_norm_sum + current_norm;
    actual_phi_norm_sum = actual_phi_norm_sum +
 (norm(phi_sim_mean_removed))^2;
end

var_eps = mean(var_eps);

mean_deviation_norm = deviation_norm_sum/(T-2);
mean_actual_norm = actual_phi_norm_sum/(T-2);
VAF = max(0,100*(1 - mean_deviation_norm/mean_actual_norm));

end
```

Published with MATLAB® R2018a

Table of Contents


```
% Filtering and Identification - final assignment
% Section 4 - Critical thinking

close all;

load systemMatrices.mat;
load turbulenceData.mat; % load the data provided

% Considering the first dataset for this section
phi_sim = phiSim{1,1};

phi_size = size(phi_sim,1);
```

Question 4.1

```
% Rank of G*H
rank_GH = rank(G*H);

% G*H is not full-rank; rank = 47
[U, S, V] = svd(G*H);
U1 = U(:,1:47);
V1 = V(:,1:47);
Sigma_SVD = S(1:47,1:47);

% dimension lifted wavefront
n_H = size(H,1);

% dimension lifted sensor slopes
n_G = size(G,1);

covariance_phi = zeros(phi_size, phi_size);

% Number of sample points for phi_sim
T = length(phi_sim);

u = zeros(n_H,T);
for k = 1:T
    covariance_phi = covariance_phi + (phi_sim(:,k)*phi_sim(:,k)');
end

covariance_phi = covariance_phi/T;

s_k = zeros(n_G,length(phi_sim));
```

```
eps_k = zeros(n_H,length(phi_sim));

s_k(:,1) = G*phi_sim(:,1) + sigmae*randn(n_G,1);

eps_pred_multiplier_matrix = (covariance_phi*G'/(G*covariance_phi*G' +
 (sigmae^2)*eye(n_G)));

% Linear least-squares solution
u(:,1) = (V1/Sigma_SVD)*U1'*(G*eps_pred_multiplier_matrix*s_k(:,1));

var_s = zeros(T,1);

for k = 2:T
    s_k(:,k) = G*(phi_sim(:,k) - H*u(:,k-1)) + sigmae*randn(n_G,1);
    Y_k = G*(eps_pred_multiplier_matrix*s_k(:,k) + H*u(:,k-1));
    u(:,k) = (V1/Sigma_SVD)*U1'*Y_k;
    eps_k(:,k) = phi_sim(:,k) - H*u(:,k-1);

    var_s(k) = var(eps_k(:,k) - mean(eps_k(:,k)));

end

var_s = mean(var_s);
```

Question 4.2

```
% VAF calculation

deviation_norm_sum = 0;
actual_phi_norm_sum = 0;

for k = 2:T-1
    % predicted_phi = eps_predicted + Hu(k)
    predicted_phi = eps_pred_multiplier_matrix*s_k(:,k) + H*u(:,k-1);
    predicted_phi = predicted_phi - mean(predicted_phi);
    phi_sim_mean_removed = phi_sim(:,k+1) - mean(phi_sim(:,k+1));
    current_norm = (norm(phi_sim_mean_removed - predicted_phi))^2;
    deviation_norm_sum = deviation_norm_sum + current_norm;
    actual_phi_norm_sum = actual_phi_norm_sum +
 (norm(phi_sim_mean_removed))^2;
end

mean_deviation_norm = deviation_norm_sum/(T-2);
mean_actual_norm = actual_phi_norm_sum/(T-2);
VAF = max(0,100*(1 - mean_deviation_norm/mean_actual_norm));
```

Question 4.3

```
phi_vec = null(G);
%phi_vec = phi_sim(:,1);
phi_matrix_1 = zeros(7,7);
phi_matrix_2 = zeros(7,7);
for i = 1:7
```

```
    phi_matrix_1(:,i) = phi_vec(7*(i-1) + 1:7*i,1);
    phi_matrix_2(:,i) = phi_vec(7*(i-1) + 1:7*i,2);
end

% To see how the turbulent wavefront needs to be to lie in the
 nullspace of
% G
figure(1);
imagesc(phi_matrix_1);

figure(2);
imagesc(phi_matrix_2);
```

Published with MATLAB® R2018a