37. Convert the AR model into state space form:

$$\begin{bmatrix} y_{k-1} \\ y_k \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix} \begin{bmatrix} y_{k-2} \\ y_{k-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w_k$$

Then, we compute the eigenvalues of the matrix $\begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix}$, then we solve the equation

$$det \begin{bmatrix} -\lambda & 1 \\ -a_2 & -a_1 - \lambda \end{bmatrix} = 0.$$

The eigenvalues of the matrix are $\lambda_{1,2} = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2}}{2}$.

To make sure the system is asymptotically stationary, we have $|\lambda_{1,2}| < 1$.

If $a_1^2 - 4a_2 < 0$, meaning that $\lambda_{1,2}$ are complex conjugate, then we have

$$\left| \frac{-a_1 + \sqrt{a_1^2 - 4a_2}}{2} \right|^2 = \left| \frac{-a_1 - \sqrt{a_1^2 - 4a_2}}{2} \right|^2 = \frac{a_1^2 + 4a_2 - a_1^2}{4} < 1$$

Then we have

$$a_2 < 1.$$

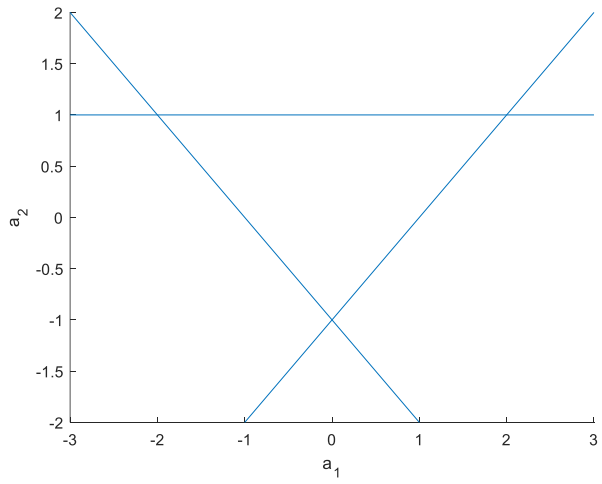
If $a_1^2 - 4a_2 \geq 0$, meaning that $\lambda_{1,2}$ are real.

If $a_1 \geq 0$, then we have $\left| -a_1 - \sqrt{a_1^2 - 4a_2} \right| = a_1 + \sqrt{a_1^2 - 4a_2} < 2$, then we have

$$a_1 - a_2 < 1$$

If $a_1 < 0$, then we have $\left| -a_1 + \sqrt{a_1^2 - 4a_2} \right| = -a_1 + \sqrt{a_1^2 - 4a_2} < 2$, then we have

$$a_1 + a_2 > -1$$

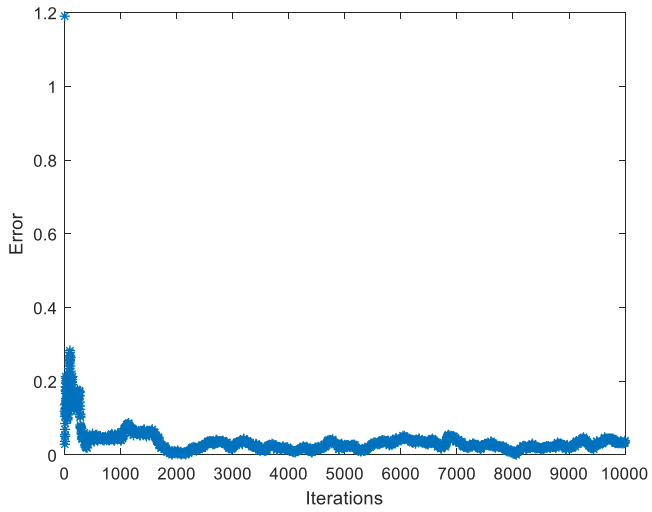


38. Choose $a_1 = -0.5, a_2 = 0.5, \rho = 0.999, c = 2$

The MATLAB code is as follows

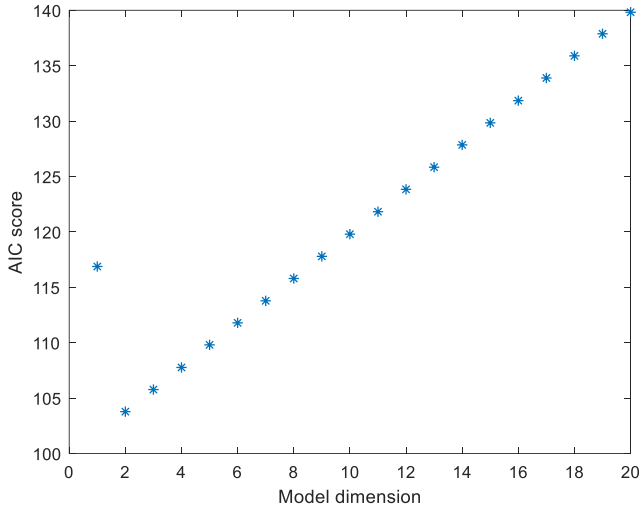
```
clc,clear,close all
a1 = -0.5;
a2 = 0.5;
N = 10000;
y = [];
y0 = 1;
y1 = 4;
ys = [y0,y1];
for i = 3:N
    y = -a1*ys(i-1)-a2*ys(i-2) + randn(1);
    ys = [ys,y];
end

theta0 = [0.8,0.8]';
rho = 0.999;
c = 2;
theta = theta0;
P = eye(2);
errs = [];
for i = 3:N
    phi = [ys(i-1);ys(i-2)];
    theta = theta + P*phi/(rho/c + phi'*P*phi)*(ys(i) - phi'*theta);
    P = 1/rho*(P - P*(phi*phi')*P/(rho/c + phi'*P*phi));
    err = norm([theta(1)-a1,theta(2)-a2]);
    errs = [errs,err];
end
plot(errs,'*')
```

The error of the estimated θ and the true parameters are show below:



The algorithm seems to be very robust to different parameter initializations. Very different parameters end up with very close parameter estimation.

39. Using the Recursive Least Square estimator developed from problem 38. We iteration the model dimension from 1 to 20, then the corresponding AIC score of the dimension is shown as the following plot:

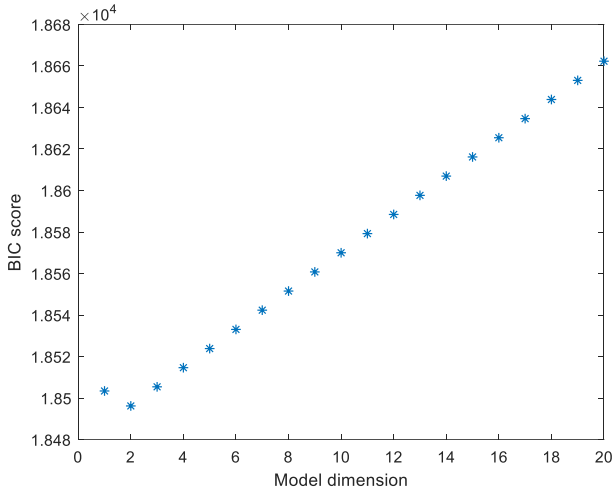


Consider that $BIC = \ln(n)\, d - 2\ln(\hat{L})$,where n is the number of data points, $\hat{L}$ is likelihood of all data points and d is the dimension of the model.

$$\ln(\hat{L}) = \ln\left(\frac{1}{\sqrt[n]{2\pi}}\prod_{i=1}^{n}\exp\left(-\frac{(y_i-\hat{y}_i)^2}{2}\right)\right) = -\frac{n}{2}\ln(2\pi) - \sum_{i=1}^{n}\frac{(y_i-\hat{y}_i)^2}{2}$$

$$BIC = \ln(n)\, d + \frac{n}{2}\ln(2\pi) + s\sum_{i=1}^{n}\frac{(y_i-\hat{y}_i)^2}{2}$$

The BIC score versus the model dimension is shown as the following plot:



Both AIC and BIC scores tell that the model dimension estimate is 2.s

The MATLAB code is shown as below:

```
clc,clear,close all
a1 = 0.5;
a2 = -0.5;
N = 10000;
y = [];
y0 = 1;
y1 = 4;
ys = [y0,y1];
for i = 3:N
    y = a1*ys(i-1)+a2*ys(i-2) + randn(1);
    ys = [ys,y];
end

dims = 1:20;
aics = [];
for d = dims
    aic = computeAIC(ys,d,N);
    aics = [aics,aic];
end
figure(1)
plot(aics,'*')
xlabel('Model dimension')
ylabel('AIC score')
bics = [];
for d = dims
    bic = computeBIC(ys,d,N);
    bics = [bics,bic];
end
figure
plot(bics,'*')
xlabel('Model dimension')
ylabel('BIC score')
function aic = computeAIC(ys,d,N)
theta0 = rand(1,d)';
rho = 0.999;
c = 2;
theta = theta0;
P = eye(d);
for i = (d+1):N
    phi = [];
    for j = 1:d
        phi = [phi;ys(i-j)];
    end
    theta = theta + P*phi/(rho/c + phi'*P*phi)*(ys(i) - phi'*theta);
    P = 1/rho*(P - P*(phi*phi')*P/(rho/c + phi'*P*phi));
end
ysim = ys(1:d);
for i = (d+1):N
    phi = [];
    for j = 1:d
        phi = [phi;ys(i-j)];
    end
    y = theta'*phi;
    ysim = [ysim,y];
end
```

```
aic = norm(ys-ysim) + 2*d;
end


function bic = computeBIC(ys,d,N)
theta0 = rand(1,d)';
rho = 0.999;
c = 2;
theta = theta0;
P = eye(d);
for i = (d+1):N
    phi = [];
    for j = 1:d
        phi = [phi;ys(i-j)];
    end
    theta = theta + P*phi/(rho/c + phi'*P*phi)*(ys(i) - phi'*theta);
    P = 1/rho*(P - P*(phi*phi')*P/(rho/c + phi'*P*phi));
end
ysim = ys(1:d);
for i = (d+1):N
    phi = [];
    for j = 1:d
        phi = [phi;ys(i-j)];
    end
    y = theta'*phi;
    ysim = [ysim,y];
end
bic = log(N)*d + 2*(N/2*log(2*pi)+1/2*norm(ys-ysim));
end
```

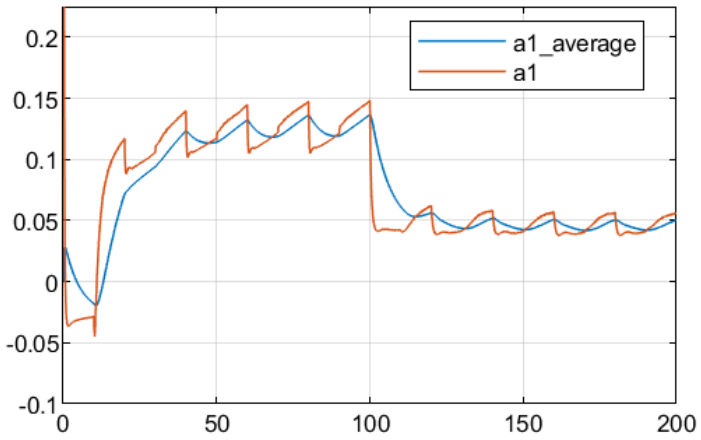
43. According to the example, the engine is modeled as follows:

$$y_n = a_1 u_{n-1} + a_2 u_{n-1}^2 + a_3 y_{n-1}$$

Then the three parameters are estimated as $a_1 = 0.05562, a_2 = -0.03716, a_3 = 0.9986.$

The inertial is not explicitly included in the three parameters but we can observe the change of inertial by inspecting the change parameters.

We can observe the following the plot that a_1 is changing for the first 100 seconds and becomes a constant afterwards. This observation indicates that the inertia changes for the first 100 seconds, which sis as expected.



46. $E(\hat{\sigma}^2) = E\left\{\frac{1}{N-n}(Y-\Psi\theta_*)^T(Y-\Psi\theta_*)\right\} = E\left\{\frac{1}{N-n}(Y^TY - \theta_*^T\Psi^TY - Y^T\Psi\theta_* + \theta_*^T\Psi^T\Psi\theta_*\right\}$

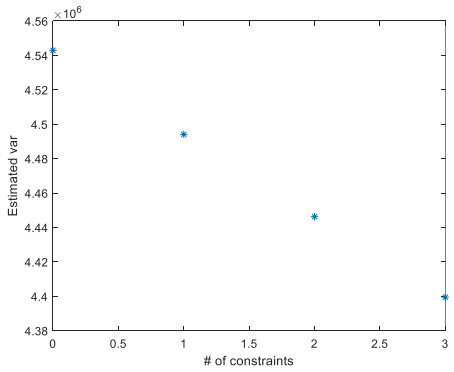
Plug in $\theta_* = (\Psi^T\Psi)^{-1}\Psi^TY$, then we have

$$E(\hat{\sigma}^2) = E\left\{\frac{1}{N-n}(Y^TY - \theta_*^T\Psi^TY - Y^T\Psi\theta_* + \theta_*^T\Psi^T\Psi(\Psi^T\Psi)^{-1}\Psi^TY\right\}$$
$$= E\left\{\frac{1}{N-n}(Y^TY - \theta_*^T\Psi^TY - Y^T\Psi\theta_* + \theta_*^T\Psi^TY\right\}$$
$$= E\left\{\frac{1}{N-n}(Y^TY - Y^T\Psi(\Psi^T\Psi)^{-1}\Psi^TY\right\}$$

Plug in $Y = \Psi\theta + \epsilon$

$$E(\hat{\sigma}^2) = E\left\{\frac{1}{N-n}(Y^TY - Y^T\Psi(\Psi^T\Psi)^{-1}\Psi^TY\right\}$$
$$= E\left\{\frac{1}{N-n}((\Psi\theta+\epsilon)^T(\Psi\theta+\epsilon) - (\Psi\theta+\epsilon)^T\Psi(\Psi^T\Psi)^{-1}\Psi^T(\Psi\theta+\epsilon))\right\}$$
$$= E\{\frac{1}{N-n}(\theta^T\Psi^T\Psi\theta + \epsilon^T\epsilon + \epsilon^T\Psi\theta + \theta^T\Psi^T\epsilon - \Big(\theta^T\Psi^T(\Psi\theta+\epsilon) + \epsilon^T\Psi(\Psi^T\Psi)^{-1}\Psi^T(\Psi\theta+\epsilon)\Big))\}$$
$$= E\{\frac{1}{N-n}(\epsilon^T\epsilon - \epsilon^T\Psi(\Psi^T\Psi)^{-1}\Psi^T\epsilon)\}$$
$$= E\{\frac{1}{N-n}(\epsilon^T\epsilon - \epsilon^T\Psi(\Psi^T\Psi)^{-1}\Psi^T\epsilon)\}$$
$$= \frac{1}{N-n}\{E\{\epsilon^T\epsilon\} - E\{\mathrm{tr}(\epsilon^T\Psi(\Psi^T\Psi)^{-1}\Psi^T\epsilon)\}\}$$
$$= \frac{1}{N-n}\{E\{\epsilon^T\epsilon\} - E\{\mathrm{tr}((\Psi^T\Psi)^{-1}\Psi^T\epsilon\epsilon^T\Psi)\}\}$$
$$= \frac{1}{N-n}\{E\{\epsilon^T\epsilon\} - \mathrm{tr}((\Psi^T\Psi)^{-1}\Psi^T I_{n\times n}\Psi)\sigma^2\}$$
$$= \frac{1}{N-n}\{N\sigma^2 - n\sigma^2\}$$
$$= \sigma^2$$

I used a dataset of which Ψ has 4 columns and computed the variance estimate with number of constraints from 3 to 0. The results are shown in the following plot:



We can observe that as the number of constraints increases, the estimated variance of the model decreases.

47. The procedure is as follows:

Step 1 : Suppose $A = [a_1, a_2, a_3 \dots a_K]$ is a matrix of K vectors of face images

Step 2: Compute $A^T A$ and AA^T use PCA to compute matrices V, U such that $VA^T AV^T$ and $UAA^T U^T$ are diagonal.

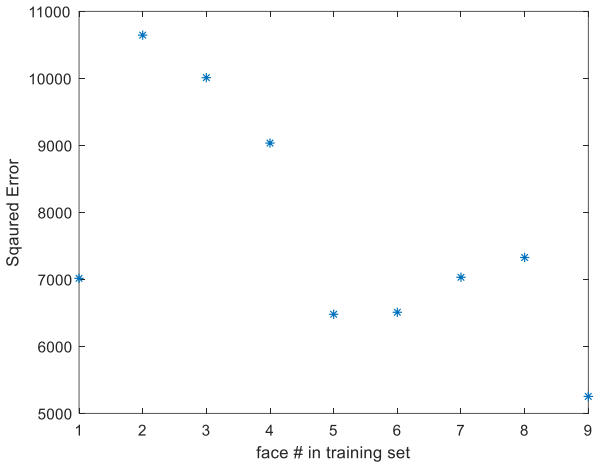
Step3: $Ur = U(:, 1:K)$ is the first most significant eigenfaces. Then the features of the K training faces are $F = Ur'A = [f_1, f_2, \dots, f_K]$.

Step 4: Compute the feature of test image $f_{test} = Ur'sa_{test}$

Step 5: $face\ class = arg \min_{i \in \{1,2,\dots,K\}} \|f_i - f_{test}\|^2$

I used 9 faces of different people as training set and another face image of the 9th person as the test case. Follow the described procedure above, the MATLAB code and final results are shown as below:

```
clc,clear,close all
addpath('faces/')
faces_matrix = [];
for i = 1:9
    im = imread(strcat('train',int2str(i),'.jpg'));
    im = rgb2gray(im);
    im = imresize(im,[45,45]);
    im_vec = double(im(:));
    faces_matrix = [faces_matrix,im_vec];
end
test_im = rgb2gray(imread('test9.jpg'));
test_im = imresize(test_im,[45,45]);
faces_matrix = faces_matrix- mean(faces_matrix,2);
test_vec = double(test_im(:)) - mean(faces_matrix,2);
[U,D,V] = svd(faces_matrix);
Ur = U(:,1:9);
features = D*V';
features = features(1:9,1:9);
new_feature = (test_vec'*Ur)';
errs = [];
for i = 1:9
    face_feature = features(:,i);
    error = norm(face_feature - new_feature);
    errs = [errs,error];
end
plot(errs,'*');
xlabel('face # in training set')
ylabel('Sqaured Error');
```



The 9th sample has the least error, which is consistent with the expectation.s