

ELEC 574 Project

Jacob Morrison

April 24th, 2019

General Problem

Start by reading the recently published paper "Adaptive Air-Fuel Ratio Regulation for Port-Injected Spark-Ignited Engines Based on a Generalized Predictive Control Method" by L. Meng et al.

This project is somewhat inspired by that paper. Here we will only consider the design of an adaptive controller for Φ_{exh} , the engine fuel/air equivalent ratio using the fuel injection mass flowrate as the manipulated variable. We will assume that the system can be described by a CARIMA model:

$$A(q^{-1})y(k) = B(q^{-1})u(k-1) + C(q^{-1})e(k)/\Delta$$

where y is Φ_{exh} , $u \geq 0$ is the fuel injection flow rate, and e is zero-mean white noise with variance $\sigma_e^2 = 0.02^*$, $\Delta = 1 - q^{-1}$ and

$$\begin{aligned} A(q^{-1}) &= 1 - (e^{-T/\tau_{exh}} + e^{-T/\tau_f})q^{-1} + e^{-(T/\tau_{exh} + T/\tau_f)}q^{-2} \\ B(q^{-1}) &= q^{-c} \left(\frac{1-X}{\dot{m}_{ap}} + \frac{X - e^{-T/\tau_f}}{\dot{m}_{ap}} q^{-1} \right) \\ C(q^{-1}) &= 1 \end{aligned}$$

where

\dot{m}_{ap} = air mass flow into cylinder, kg/s

X = fraction of fuel flow into film

τ_f = fuel film evaporation time constant

τ_{exh} = time constant of gas exhaust

T = sampling time = $0.05s$

$c = \tau_d/T$ where τ_d is the AFR time delay with $3 \leq c \leq 5$

The control objective is to maintain the engine equivalent ratio $\Phi = 1$ during simulations of 1000 samples.

***Note:** Based on my interpretation of the results of the paper "Adaptive Air-Fuel Ratio Regulation for Port-Injected Spark-Ignited Engines Based on a Generalized Predictive Control Method" as well as my own testing, I believe the paper misreported the standard deviation of the noise used as the variance. A variance of $\sigma_e^2 = 0.02$ means a standard deviation of $\sigma_e \approx 0.14$ which is very large given the situation. With that in mind, I use a standard deviation of $\sigma_e = 0.02$ when answering the questions involved in this project, as I believe that is the noise level that was actually used in the paper, and a standard deviation of $\sigma_e \approx 0.14$ is too large to see any meaningful results.

1 Subproblem One

We will assume that the following parameters are constant: $\dot{m}_{ap} = 15g/s$, $\tau_{exh} = 0.15s$, $\tau_f = 2s$, $X = 0.7$, $c = 4$. Design and simulate an adaptive generalized predictive controller that works well for this nominal set of parameters.

1.1 Problem Set Up

First, the actual values for $A(q^{-1})$ and $B(q^{-1})$ can be solved by plugging in the assumed constants, leading to a time invariant sytem.

$$\begin{aligned}
 A(q^{-1}) &= 1 - (e^{-T/\tau_{exh}} + e^{-T/\tau_f})q^{-1} + e^{-(T/\tau_{exh}+T/\tau_f)}q^{-2} \\
 &= 1 - (e^{-0.05/0.15} + e^{-0.05/2})q^{-1} + e^{-(0.05/0.15+0.05/2)}q^{-2} \\
 &\approx 1 - 1.692q^{-1} + 0.699q^{-2} \\
 B(q^{-1}) &= q^{-c} \left(\frac{1-X}{\dot{m}_{ap}} + \frac{X - e^{-T/\tau_f}}{\dot{m}_{ap}} q^{-1} \right) \\
 &= q^{-4} \left(\frac{1-0.7}{15} + \frac{0.7 - e^{-0.05/2}}{15} q^{-1} \right) \\
 &\approx q^{-4}(0.02 - 0.0184q^{-1})
 \end{aligned}$$

Note that approximate values for the coefficients are shown above, but exact values will be used in all simulations. Further, when it comes to designing a generalized predictive controller (GPC) it is necessary to select values for N_1 , N_2 , and N_u , which are the minimum prediction horizon, maximum prediction horizon, and control horizon respectively. In this case the minimum prediction horizon value will be chosen as $N_1 = 4$ to match the time delay, while N_2 and N_u will be left as tuning parameters. Also note that the controller design here will be similiar to the controller design seen in the 3rd course assignment as both cases required the design of a GPC controller.

1.2 Controller Design

In general a GPC is developed by minimizing the following cost function:

$$J(N_1, N_2, N_u) = E \left\{ \sum_{j=N_1}^{N_2} (y(k+j) - w(k+j))^2 + \sum_{j=1}^{N_u} \rho \Delta u(k+j-1)^2 \right\} \quad (1)$$

where w is the reference signal and ρ is a tuning parameter that regulates large inputs. First though, in order to design a GPC the Diophantine equation must be introduced. In this case it takes the following form:

$$1 = A\Delta F_j(q^{-1}) + q^{-j}G_j(q^{-1}) \quad (2)$$

where j is the number of time steps ahead used for prediction, $\Delta = 1 - q^{-1}$, $\deg(F_j) = j - 1$, and $\deg(G_j) = n - 1$. This equation can be solved recursively using:

$$\begin{aligned}
 f_j &= g_0^j \\
 g_i^{j+1} &= g_{i+1}^j - \tilde{a}_{i+1} f_j \\
 F_{j+1}(q^{-1}) &= F_j(q^{-1}) + f_j q^{-j} \\
 F_1 &= 1 \\
 G_1 &= q(1 - \tilde{A})
 \end{aligned} \quad (3)$$

where f_j is the j^{th} term of F_j , g_i^j is the i^{th} term of G_j , $\tilde{A} = A\Delta$, and \tilde{a}_i is the i^{th} coefficient of \tilde{A} . Now, with the ability to solve F_j and G_j , an equation for the response estimate at time $t+j$ can be developed by combining the given CARIMA model with equation (2) to give:

$$\hat{y}(k+j | k) = G_j y(k) + R_j \Delta u(k+j-d) \quad (4)$$

where $R_j = BF_j$ and the coefficients of R_j are denoted r_i . Further, the prediction for $y(t+j)$ under the assumption that future control signals are 0 is defined as:

$$\bar{y}(k+j) = G_j y(k) \quad (5)$$

This allows the cost function to be written as:

$$\begin{aligned} J(N_1, N_2, N_u) &= E \{ (\mathbf{y} - \mathbf{w})^\top (\mathbf{y} - \mathbf{w}) + \rho \Delta \mathbf{u}^\top \Delta \mathbf{u} \} \\ &= (\mathbf{R} \Delta \mathbf{u} + \bar{\mathbf{y}} - \mathbf{w})^\top (\mathbf{R} \Delta \mathbf{u} + \bar{\mathbf{y}} - \mathbf{w}) + \rho \Delta \mathbf{u}^\top \Delta \mathbf{u} \end{aligned} \quad (6)$$

where

$$\begin{aligned} \mathbf{y} &= [\hat{y}(k+N_1 | k), \dots, \hat{y}(k+N_2 | k)]^\top \\ \Delta \mathbf{u} &= [\Delta u(k+N_1-d), \dots, \hat{y}(k+N_2-d)]^\top \\ \bar{\mathbf{y}} &= [\bar{y}(k+N_1), \dots, \bar{y}(k+N_2)]^\top \\ \mathbf{w} &= [w(k+N_1), \dots, w(k+N_2)]^\top \end{aligned}$$

and \mathbf{R} is the dynamic matrix of the system which consists of the coefficients of R_j , has dimensions $(N_2 - N_1) \times N_u$, and is defined in this case as:

$$\mathbf{R} = \begin{bmatrix} r_{N_1-1} & r_{N_1-2} & \dots & r_0 & 0 & \dots & \dots & 0 \\ r_{N_1} & r_{N_1-1} & \dots & \dots & r_0 & 0 & \dots & 0 \\ r_{N_1+1} & \vdots & \dots & \dots & \dots & r_0 & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & r_0 \\ r_{N_2-1} & r_{N_2-2} & \dots & \dots & \dots & \dots & \dots & r_{N_2-N_u} \end{bmatrix}$$

Finally, by minimizing equation (6) with respect to $\Delta \mathbf{u}$ the optimal change in control input is determined to be:

$$\Delta \mathbf{u} = [\mathbf{R}^\top \mathbf{R} + \rho]^{-1} \mathbf{R}^\top (\mathbf{w} - \bar{\mathbf{y}}) \quad (7)$$

Which leads to the following control update:

$$u(k) = u(k-1) + \Delta u(k+N_1-d) \quad (8)$$

Note that the system input cannot be negative ($u \geq 0$) so any values calculated as negative will simply be set to 0.

1.3 Recursive Parameter Estimation

In order to make the GPC adaptive, it is necessary to estimate the model parameters at each time step. Since the system is time invariant and $C(q^{-1}) = 1$, this will be done using a standard recursive least squares method. The estimated parameters will then be used at each time step to specify the control input, while the given model parameters will only be used to simulate the plant.

1.4 Simulation Results

The performance of the controller was explored by putting it through the simulation outlined in the problem. Numerous variations of tuning parameters N_u , N_2 , and ρ were examined, with the best result displayed below. This result featured the values $N_u = 3$, $N_2 = 8$, and $\rho = 0.1$. Note that while this case led to the best control, it is less computationally efficient than the case when $N_u = 1$. Overall, the controller is able to maintain the system output (Φ_{exh}) at approximately the reference value of 1 quite well. Further, parameter estimates converged to approximately the actual parameter values after about 100 samples. The Matlab code for this simulation is displayed in section 5.1.

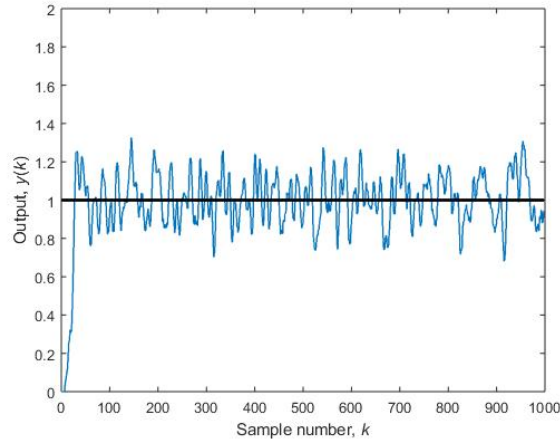


Figure 1: System output over time compared to the reference signal. The output is the engine fuel/air equivalent ratio Φ of the system

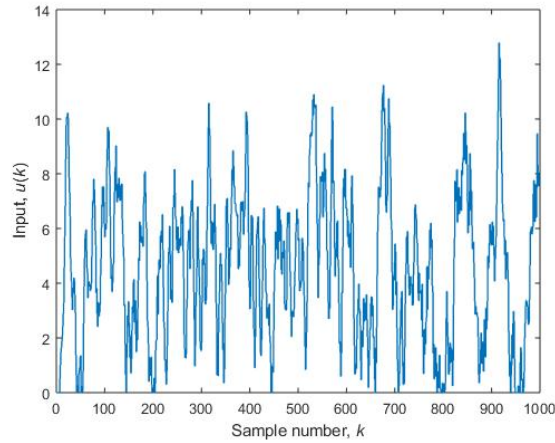


Figure 2: Input to the sytem over time. The input is the fuel injection mass flowrate in g/s

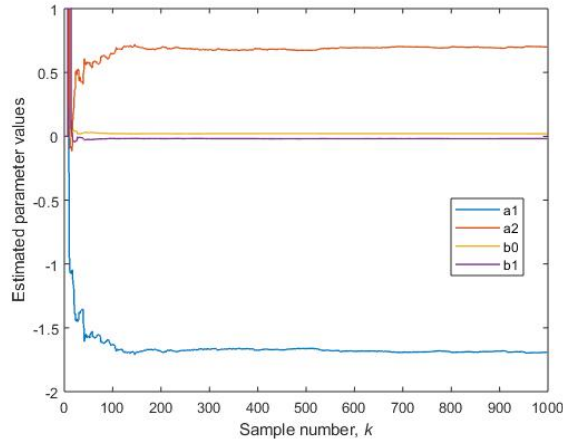


Figure 3: Model parameter convergence over time

2 Subproblem Two

Simulate the nominal adaptive controller designed above for the situations $c = 3$ and $c = 5$. Make any change in the design parameters that you deem necessary to ensure robustness.

2.1 Problem Set Up

The problem set up and controller design in this case are essentially the same as in the previous subproblem. There are only 2 slight changes that will be made in order to ensure robustness. Firstly, N_1 will be set to 3 instead of 4 to account for the minimum possible input delay. Secondly, the order of the recursive estimate for $B(q^{-1})$ will be increased from 2 to 4 so as to include all possible time delay values in the system ($3 \leq c \leq 5$), which is a common practice when the delay is unknown.

2.2 Simulation Results

The robustness of the adaptive controller was explored by putting it through two simulations relating to the situations $c = 3$ and $c = 5$. Here the values for N_u , N_2 , and ρ were simply taken from the previous subproblem, with the results displayed below. As a whole, the controller appears quite robust to the various input delays and is able to maintain the system output (Φ_{exh}) at approximately the reference value of 1, both when $c = 3$ and $c = 5$. It does perform better when $c = 3$, but this makes sense as lesser time delays are easier to control. Additionally, the parameter estimates for the coefficients of $A(q^{-1})$ converged to approximately their actual values rather quickly in both cases, while the estimates for the coefficients of $B(q^{-1})$ converged quite fast but are different than the actual values due to the increased order of $B(q^{-1})$ in the recursive estimation. The Matlab code for these simulations is identical to the code displayed in section 5.1 except $N_1 = 3$, the order of the estimate for $B(q^{-1})$ was set to 4 ($nb = 4$), c was set to 3 for one simulation and 5 for the other, and line 107 was commented out while lines 98-104 were uncommented.

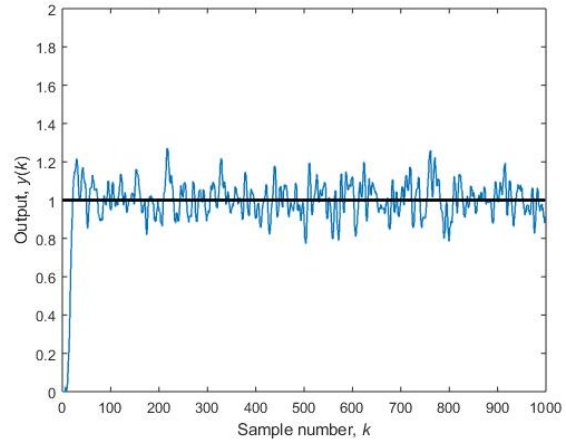


Figure 4: System output over time compared to the reference signal when $c = 3$. The output is the engine fuel/air equivalent ratio Φ of the system

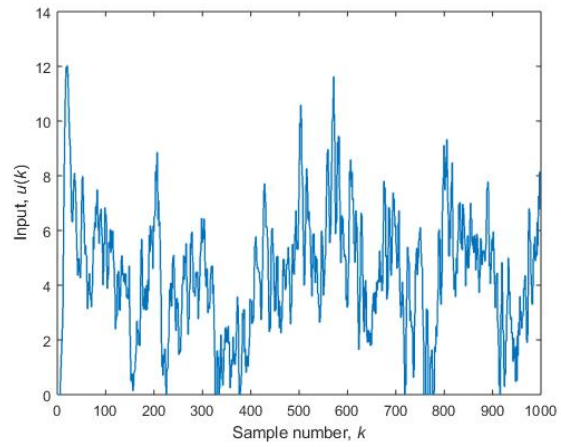


Figure 5: Input to the sytem over time when $c = 3$. The input is the fuel injection mass flowrate in g/s

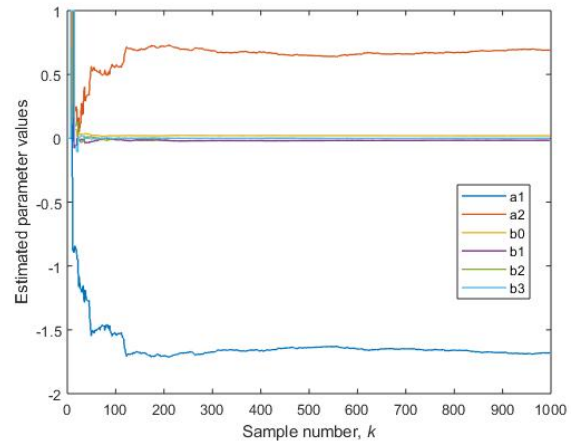


Figure 6: Model parameter estimates over time when $c = 3$

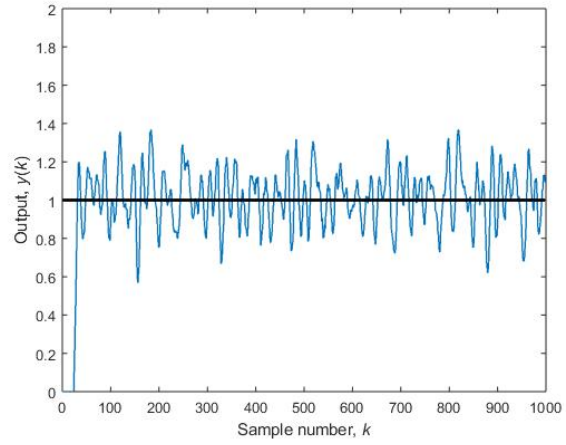


Figure 7: System output over time compared to the reference signal when $c = 5$. The output is the engine fuel/air equivalent ratio Φ of the system

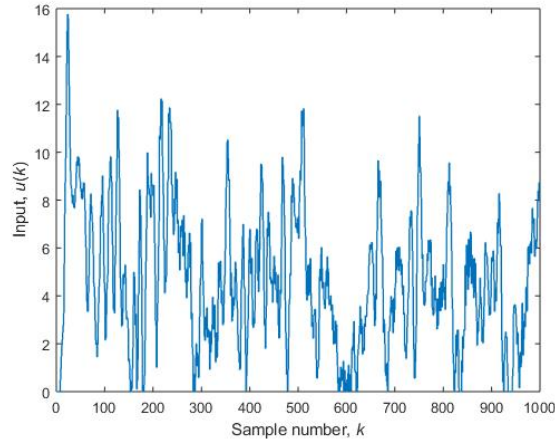


Figure 8: Input to the system over time when $c = 5$. The input is the fuel injection mass flowrate in g/s

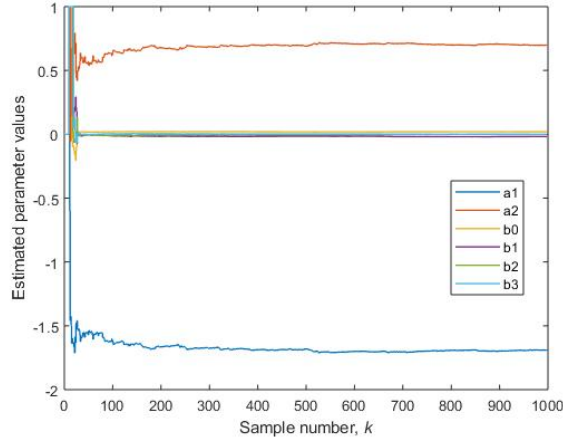


Figure 9: Model parameter estimates over time when $c = 5$

3 Subproblem Three

We will now assume that X is time-varying and can be described as a sine wave of peak-to-peak amplitude of 0.2 and period of 200 samples around its nominal value of 0.7. We will also assume that $m_{ap} = 15g/s$ except for: samples 200-220: $m_{ap} = 20g/s$; samples 400-420: $m_{ap} = 30g/s$; samples 600-620: $m_{ap} = 40g/s$; Simulate your adaptive controller for this situation when $c = 4$.

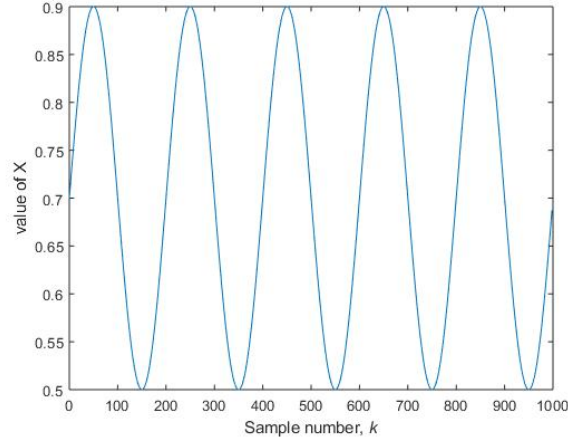


Figure 10: variation in X value over time

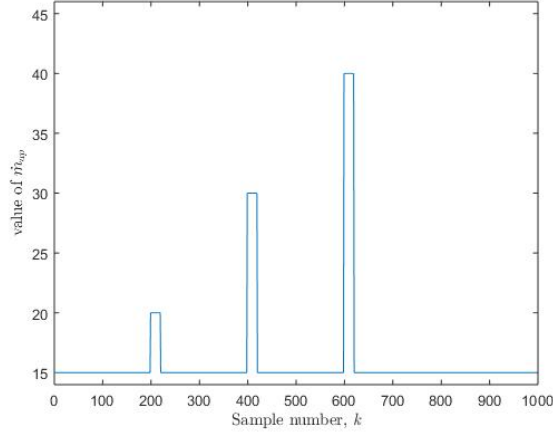


Figure 11: variation in \dot{m}_{ap} value over time

3.1 Problem Set Up

The controller design in this case is the same as in subproblem one. The set up of $B(Q^{-1})$ changes slightly here though, as $B(Q^{-1})$ is a function of X and \dot{m}_{ap} , which now both change over time. This means that the system is now time varying, so in order to make the adaptive GPC effective a new recursive estimation technique will be required.

3.2 Recursive Parameter Estimation

Due to the time varying nature of the system, the recursive parameter estimation will be accomplished using the Exponential Forgetting and Resetting Algorithm (EFRA). The EFRA allows the estimation to focus on recent data, without considering older data that is no longer representative of the process. There is a version of RLS that includes a forgetting factor that also focuses on recent data, though EFRA is more effective as the covariance matrix P will never grow exponentially as it is bounded on both sides at all times. The estimate update process with EFRA works similarly to that of RLS, except with a few extra tuning parameters that are detailed below.

$$\begin{aligned}
K(k+1) &= \frac{P(k)x(k+1)}{\lambda + x^\top(k+1)P(k)x(k+1)} \\
P(k+1) &= \frac{1}{\lambda} \left(P(k) - \frac{P(k)x(k+1)x^\top(k+1)P(k)}{\lambda + x^\top(k+1)P(k)x(k+1)} \right) + \beta I - \gamma P(k)^2 \\
\hat{\theta}(k+1) &= \hat{\theta}(k) + \alpha K(k+1) \left(y(k+1) - x^\top(k+1)\hat{\theta}(k) \right)
\end{aligned}$$

Here x is the same as in RLS, λ is the forgetting factor (also featured in the forgetting factor version of RLS), and α , β , and γ are additional parameters that can be tuned. As before, the estimated parameters will be used at each time step to specify the control input, while the given model parameters will only be used to simulate the plant.

3.3 Simulation Results

The performance of the controller with the updated parameter estimation was explored by putting it through the same simulation as in subproblem one, except this time with the time varying values for X and \dot{m}_{ap} . Numerous variations of tuning parameters N_u , N_2 , and ρ were again examined, with the best result displayed below. In this case the best result featured the values $N_u = 3$, $N_2 = 12$, and $\rho = 0.15$. Additionally, values of $\lambda = 0.95$, $\alpha = 0.9$, and $\beta = \gamma = 0.001$ were selected through trial and error for the recursive parameter estimation. The controller seems to perform reasonably well, though not quite as well as the controller in subproblem one. This makes sense seeing as time varying systems are typically harder to control. Further, parameter estimates vary a lot more than in the previous subproblems due to the EFRA estimation technique. It is interesting to note that in reality only $B(q^{-1})$ is time varying, but the EFRA leads to increased variation in the estimates for the coefficients of $A(q^{-1})$ as well. Nonetheless, this does not seem to hinder controller performance. The Matlab code for this simulation is displayed in section 5.2.

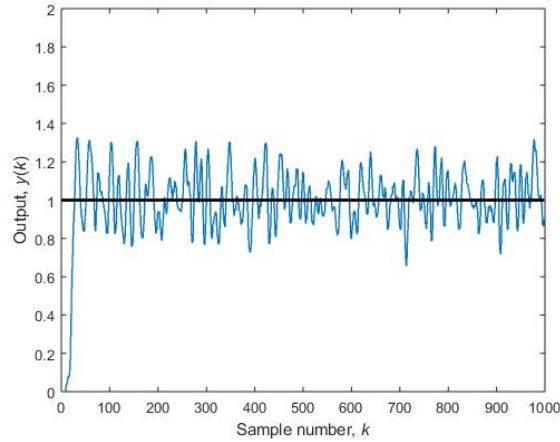


Figure 12: System output over time compared to the reference signal. The output is the engine fuel/air equivalent ratio Φ of the system

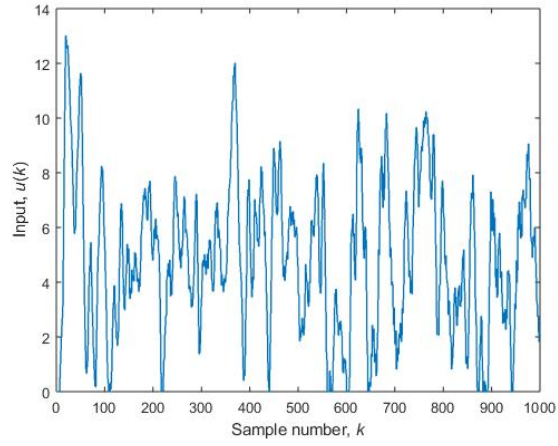


Figure 13: Input to the sytem over time. The input is the fuel injection mass flowrate in g/s

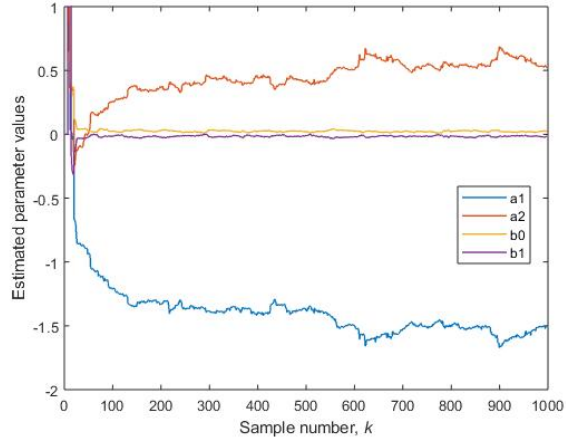


Figure 14: Model parameter convergence over time

4 Subproblem Four

Feeling brave, we will now simulate the above situation but adding variations for the delay c : samples 1-200: $c = 4$; samples 200-500: $c = 3$; samples 500-800: $c = 5$; samples 800-1000: $c = 4$;

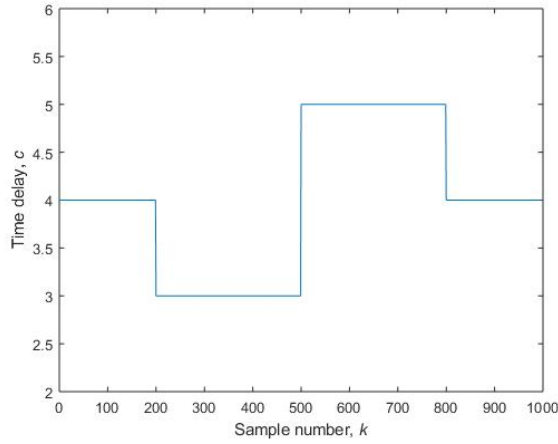


Figure 15: variation in time delay c over time

4.1 Problem Set Up

The problem set up and controller design in this case are very similar to the previous subproblem. The differences here are the same as the differences between subproblem one and two. N_1 will again be set as 3 to account for the minimum possible input delay while the order of the recursive estimate for $B(q^{-1})$ will be set to 4 so as to include all possible time delay values in the system ($3 \leq c \leq 5$).

4.2 Simulation Results

The performance of the previously designed GPC was examined by carrying out the simulation described in this subproblem. Here the values for N_u , N_2 , and ρ were again taken from the previous subproblem, with the results displayed below. As a whole, the controller appears quite robust to the variations in c , never allowing the system output to stray too far from the reference. This leads to the positive conclusion that adaptive GPC with EFRA parameter estimation represents a solid control option for time varying single-input single-output systems, even when the time delay is both unknown and variable. The Matlab code for this simulation is displayed in section 5.3.

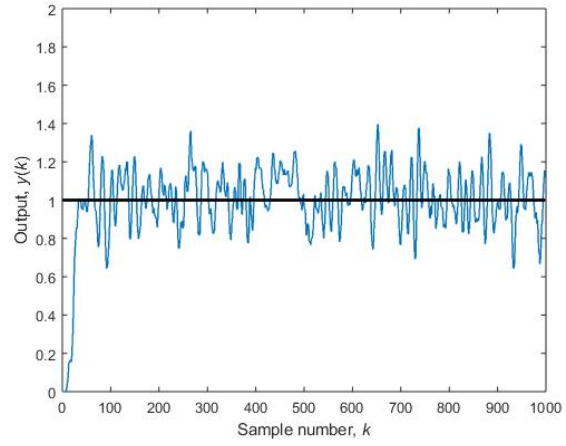


Figure 16: System output over time compared to the reference signal when the time delay is varying. The output is the engine fuel/air equivalent ratio Φ of the system

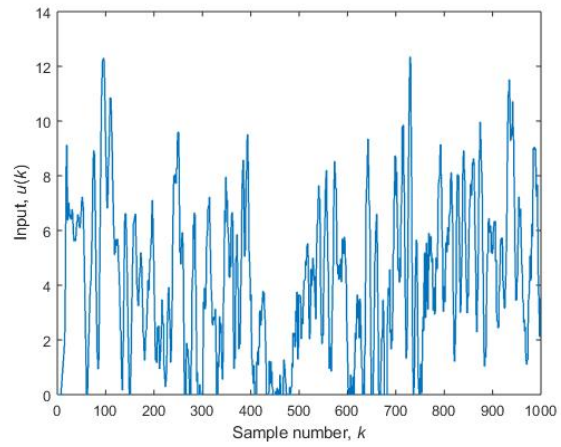


Figure 17: Input to the sytem over time when the time delay is varying. The input is the fuel injection mass flowrate in g/s

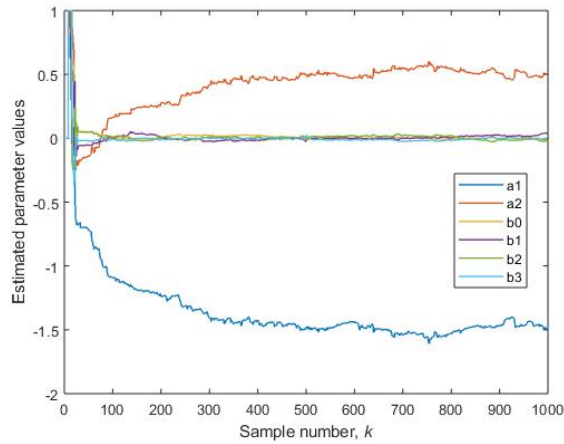


Figure 18: Model parameter estimates over time when the time delay is varying

5 Matlab Code

5.1 Code for Subproblem One and Two

The code displayed below was used for subproblem one. Note that the code for subproblem two is identical, save for the minor differences outlined in section 2.2.

```

1  clc;
2  clear;
3  close all;
4
5  %set up vars
6  map = 15; %g/s
7  texh = 0.15; %s
8  tf = 2; %s
9  X = 0.7;
10 T = 0.05; %sampling time in seconds
11
12 %set up reference
13 t= (0:1200)';
14 N = length(t);
15 yr = zeros(N,1);
16 yr(1:N) = 1;
17
18 %set up actual model, estimated model, control params
19 %actual model
20 A = [ (exp( T/texh)+exp( T/tf)), exp( (T/texh+T/tf)) ]';
21 B = [(1-X)/(map), (X exp( T/tf))/map]';
22 c = 4;
23 e = random('norm', 0, 0.02, 1 , N);
24
25 %initialize estimated model
26 na = 2;
27 nb = 2;

```

```

28 nk = c; %input delay
29 nparams = na+nb; %6 parameters to be estimated (a1,a2,a3,b0,b1,b2)
30
31 ts = max([na nb+nk])+1; %sample number to start iterations at
32
33 u = zeros(N,1);
34 y = zeros(N,1);
35
36 %set up RLS components
37 theta = [ ones(na,1); ones(nb,1) ];
38 thetavec = zeros(N,nparams);
39 alpha = 1000;
40 P = alpha*eye(nparams);
41 x = zeros(nparams,1);
42
43 %control params
44 N1 = 4;
45 N2 = 8;
46 Nu = 3;
47 rho = 0.1;
48 %alpha2 = 0;
49 te = N N2; %sample number to stop iterations at
50 ybar = zeros(N2 N1+1,1);
51 Adelta = zeros(na+1,1);
52 %w = zeros(N2 N1+1,1);
53 G = zeros(N2 N1+1,na+1);
54 f = zeros(N2,1);
55 r = zeros(length(f)+1,1);
56 R_tmp = zeros(length(r)+Nu-1,Nu);
57 R = zeros(N2 N1+1,Nu);
58
59 %simulation and recursive estimation
60 %comment out the following line to leave noise in the system
61 %e = zeros(1,length(e));
62
63 for i = ts:te
64     %test first without estimating params
65     %Aest = A;
66     %Best = B;
67     %get estimated parameters from theta:
68     Aest = theta(1:na);
69     Best = theta(na+1:nparams);
70
71     %get Adelta from Aest
72     for j = 1:na+1
73         if j == 1
74             Adelta(j) = Aest(j) - 1;
75         elseif j == na+1
76             Adelta(j) = Aest(j-1);
77         else
78             Adelta(j) = Aest(j) - Aest(j-1);

```

```

79         end
80     end
81
82     %solve diophantine equation
83     G(1,:) = Adelta';
84     f(1) = G(1,1);
85     r(1) = Best(1);
86     r(2) = Best(2)+Best(1)*f(1);
87     for j = 1:N2-1
88         for k = 1:na+1
89             if k == na+1
90                 G(j+1,k) = Adelta(k)*f(j);
91             else
92                 G(j+1,k) = G(j,k+1) Adelta(k)*f(j);
93             end
94         end
95         f(j+1) = G(j+1,1);
96         %set up R matrix coefficients
97         %if order(B) = 4
98         %     if j == 1
99         %         r(j+2) = Best(3)+Best(2)*f(j)+Best(1)*f(j+1);
100        %     elseif j==2
101        %         r(j+2) = Best(4)+Best(3)*f(j-1)+Best(2)*f(j)+Best(1)*f(j+1);
102        %     else
103        %         r(j+2) = Best(4)*f(j-2)+Best(3)*f(j-1)+Best(2)*f(j)+Best(1)*f(j
104        %         +1);
105        %     end
106
107        %if order(B) = 2
108        r(j+2) = Best(2)*f(j)+Best(1)*f(j+1);
109    end
110    F = [1; f];
111
112    %set up dynamic matrix R
113    for j = 1:Nu
114        R_tmp(j:j+length(r)-1,j) = r;
115    end
116    R = R_tmp(N1:N2,:);
117
118    %simulate plant output
119    y(i) = A(1)*y(i-1) A(2)*y(i-2) + B(1)*u(i-c-1) ...
120        +B(2)*u(i-c-2)+e(i);
121
122    %investigate altered reference values from paper
123    %for j = N1:N2
124    %w(j-N1+1) = alpha2*y(i)+(1-alpha2)*yr(i);
125    %end
126    %get simple future reference values
127    w = yr((i+N1):(i+N2));
128
129    %predict future outputs

```



```

129     for j = N1:N2
130         ybar(j, N1+1) = G(j, :) * y(i: 1:i 2);
131     end
132
133     %update control action
134     Rmat = R'*R+rho*eye(min(size(R)));
135     delta_u = Rmat \ (R') * (w ybar);
136     u(i) = u(i 1) + delta_u(1);
137     if u(i) < 0
138         u(i) = 0;
139     end
140
141     %update parameter estimates
142     for j = 1:na
143         x(j) = y(i j);
144     end
145     for j = 1:nb
146         x(na+j) = u(i nk j);
147     end
148     K = (P*x)/(1+x'*P*x);
149     P = P / (P*(x*x')*P)/(1+x'*P*x);
150     theta = theta + K*(y(i) - x'*theta);
151     thetavec(i, :) = [Aest' Best'];
152 end
153
154 %Plot results
155 val=999;
156 figure(2)
157 stairs(t(1:val), u(1:val), 'LineWidth', 1);
158 ylabel({'Input', '\itu\rm(\itk\rm)'});
159 xlabel({'Sample number', '\itk'});
160 figure(3);
161 stairs1 = stairs(t(1:val), thetavec(1:val, :), 'LineWidth', 1);
162 set(stairs1(1), 'DisplayName', 'a1');
163 set(stairs1(2), 'DisplayName', 'a2');
164 set(stairs1(3), 'DisplayName', 'b0');
165 set(stairs1(4), 'DisplayName', 'b1');
166 %set(stairs1(5), 'DisplayName', 'b2');
167 %set(stairs1(6), 'DisplayName', 'b3');
168 ylabel({'Estimated parameter values'});
169 xlabel({'Sample number', '\itk'});
170 legend('show');
171 figure(1);
172 stairs2 = stairs(t(1:val), [y(1:val), yr(1:val)]);
173 set(stairs2(1), 'LineWidth', 1);
174 set(stairs2(2), 'LineWidth', 2, 'Color', [0 0 0]);
175 ylabel({'Output', '\ity\rm(\itk\rm)'});
176 xlabel({'Sample number', '\itk'});
177 ylim([0 2]);

```

5.2 Code for Subproblem Three

```

1  clc;
2  clear;
3  close all;
4
5  %set up reference
6  t= (0:1200)';
7  N = length(t);
8  yr = zeros(N,1);
9  yr(1:N) = 1;
10
11 %set up vars
12 map = 15*ones(N,1); %g/s
13 map(200:220) = 20*ones(21,1);
14 map(400:420) = 30*ones(21,1);
15 map(600:620) = 40*ones(21,1);
16 texh = 0.15; %s
17 tf = 2; %s
18 X = 0.7+0.2*sin(t*pi/100);
19 T = 0.05; %sampling time in seconds
20
21 %set up actual model, estimated model, control params
22 %actual model
23 A = [ (exp(-T/texh)+exp(-T/tf)), exp(-(T/texh+T/tf))]';
24 B = [(1-X)./(map), (X*exp(-T/tf))./map]';
25 c = 4;
26 e = random('norm', 0, 0.02, 1, N);
27
28 %initialize estimated model
29 na = 2;
30 nb = 2;
31 nk = c; %input delay
32 nparams = na+nb; %6 parameters to be estimated (a1,a2,a3,b0,b1,b2)
33
34 %set up parameter estimation constants
35 lambda = 0.95; %forgetting factor
36 alpha3 = 0.9;
37 beta = 0.001;
38 gamma = 0.001;
39
40 ts = max([na nb+nk])+2; %sample number to start iterations at
41
42 u = zeros(N,1);
43 y = zeros(N,1);
44
45 %set up RLS components
46 theta = [ones(na,1);ones(nb,1)];
47 thetavec = zeros(N,nparams);
48 alpha = 1000;
49 P = alpha*eye(nparams);
50 x = zeros(nparams,1);
51

```

```

52 %control params
53 %3, 8, 2, 0.1
54 %4, 12, 3, 0.4, 0.2
55 N1 = 4;
56 N2 = 12;
57 Nu = 3;
58 rho = 0.15;
59 %alpha2 = 0;
60 te = N2; %sample number to stop iterations at
61 ybar = zeros(N2 N1+1,1);
62 Adelta = zeros(na+1,1);
63 %w = zeros(N2 N1+1,1);
64 G = zeros(N2 N1+1,na+1);
65 f = zeros(N2,1);
66 r = zeros(length(f)+1,1);
67 R_tmp = zeros(length(r)+Nu-1,Nu);
68 R = zeros(N2 N1+1,Nu);
69
70 %simulation and recursive estimation
71 %comment out the following line to leave noise in the system
72 %e = zeros(1,length(e));
73
74 for i = ts:te
75     %test first without estimating params
76     %Aest = A;
77     %Best = B(:,i);
78     %get estimated parameters from theta:
79     Aest = theta(1:na);
80     Best = theta(na+1:nparams);
81
82     %get Adelta from Aest
83     for j = 1:na+1
84         if j == 1
85             Adelta(j) = Aest(j)-1;
86         elseif j == na+1
87             Adelta(j) = Aest(j)-1;
88         else
89             Adelta(j) = Aest(j)-Aest(j-1);
90         end
91     end
92
93     %solve diophantine equation
94     G(1,:) = Adelta';
95     f(1) = G(1,1);
96     r(1) = Best(1);
97     r(2) = Best(2)+Best(1)*f(1);
98     for j = 1:N2-1
99         for k = 1:na+1
100             if k == na+1
101                 G(j+1,k) = Adelta(k)*f(j);
102             else

```

```

103         G(j+1,k) = G(j,k+1) - Adelta(k)*f(j);
104     end
105 end
106     f(j+1) = G(j+1,1);
107     %set up R matrix coefficients
108     r(j+2) = Best(2)*f(j)+Best(1)*f(j+1);
109 end
110 F = [1; f];
111
112 %set up dynamic matrix R
113 for j = 1:Nu
114     R_tmp(j:j+length(r)-1,j) = r;
115 end
116 R = R_tmp(N1:N2,:);
117
118 %simulate plant output
119 y(i) = A(1)*y(i-1) - A(2)*y(i-2) + B(1)*u(i-1) ...
120     +B(2)*u(i-2)+e(i);
121
122 %investigate altered reference values from paper
123 %for j = N1:N2
124     %w(j-N1+1) = alpha2*y(i)+(1-alpha2)*yr(i);
125 %end
126 %get simple future reference values
127 w = yr((i+N1):(i+N2));
128
129 %predict future outputs
130 for j = N1:N2
131     ybar(j-N1+1) = G(j,:)*y(i-1:i-2);
132 end
133
134 %update control action
135 Rmat = R'*R+rho*eye(min(size(R)));
136 delta_u = Rmat\((R')*(w-ybar);
137 u(i) = u(i-1)+delta_u(1);
138 if u(i)<0
139     u(i)=0;
140 end
141
142 %update parameter estimates
143 for j = 1:na
144     x(j) = y(i-j);
145 end
146 for j = 1:nb
147     x(na+j) = u(i-nk-j);
148 end
149 %RLS
150 % K = (P*x)/(lambda+x'*P*x);
151 % P = (P - (P*(x*x')*P)/(lambda+x'*P*x))*(1/lambda);
152 % theta = theta+K*(y(i)-x'*theta);
153 % thetavec(i,:) = [Aest' Best'];

```

```

154
155 %EFRA
156 K = (P*x)/(lambda+x'*P*x);
157 P = (P ((P*(x*x')*P)/(lambda+x'*P*x)))*(1/lambda)+...
158     beta*eye(nparams) gamma*P*P;
159 theta = theta+alpha3*K*(y(i) x'*theta);
160 thetavec(i,:) = [Aest' Best'];
161 end
162
163 %Plot results
164 val=999;
165 figure(2)
166 stairs(t(1:val),u(1:val),'LineWidth',1);
167 ylabel({'Input, \itu\rm(\itk\rm)'});
168 xlabel({'Sample number, \itk'});
169 figure(3);
170 stairs1 = stairs(t(1:val),thetavec(1:val,:), 'LineWidth',1);
171 set(stairs1(1),'DisplayName','a1');
172 set(stairs1(2),'DisplayName','a2');
173 set(stairs1(3),'DisplayName','b0');
174 set(stairs1(4),'DisplayName','b1');
175 ylabel({'Estimated parameter values'});
176 xlabel({'Sample number, \itk'});
177 legend('show');
178 figure(1);
179 stairs2 = stairs(t(1:val),[y(1:val),yr(1:val)]);
180 set(stairs2(1),'LineWidth',1);
181 set(stairs2(2),'LineWidth',2,'Color',[0 0 0]);
182 ylabel({'Output, \ity\rm(\itk\rm)'});
183 xlabel({'Sample number, \itk'});
184 ylim([0 2]);

```

5.3 Code for Subproblem Four

```

1 clc;
2 clear;
3 close all;
4
5 %set up reference
6 t= (0:1200)';
7 N = length(t);
8 yr = zeros(N,1);
9 yr(1:N) = 1;
10
11 %set up vars
12 map = 15*ones(N,1); %g/s
13 map(200:220) = 20*ones(21,1);
14 map(400:420) = 30*ones(21,1);
15 map(600:620) = 40*ones(21,1);
16 texh = 0.15; %s
17 tf = 2; %s
18 X = 0.7+0.2*sin(t*pi/100);

```

```

19 T = 0.05; %sampling time in seconds
20
21 %set up actual model, estimated model, control params
22 %actual model
23 A = [ (exp( T/texh)+exp( T/tf)), exp( (T/texh+T/tf)) ]';
24 B = [(1 X)./(map), (X exp( T/tf))./map]';
25 c = 4*ones(N,1);
26 c(201:500) = 3*ones(300,1);
27 c(501:800) = 5*ones(300,1);
28 e = random('norm', 0, 0.02, 1 , N);
29
30 %initialize estimated model
31 na = 2;
32 nb = 4;
33 nk = 3; %input delay
34 nparams = na+nb; %6 parameters to be estimated (a1,a2,a3,b0,b1,b2)
35
36 %set up parameter estimation constants
37 lambda = 0.95; %forgetting factor
38 alpha3 = 0.9;
39 beta = 0.001;
40 gamma = 0.001;
41
42 ts = max([na nb+nk])+2; %sample number to start iterations at
43
44 u = zeros(N,1);
45 y = zeros(N,1);
46
47 %set up RLS components
48 theta = [ ones(na,1); ones(nb,1) ];
49 thetavec = zeros(N,nparams);
50 alpha = 1000;
51 P = alpha*eye(nparams);
52 x = zeros(nparams,1);
53
54 %control params
55 %3, 8, 2, 0.1
56 %4, 12, 3, 0.4, 0.2
57 N1 = 4;
58 N2 = 12;
59 Nu = 3;
60 rho = 0.15;
61 %alpha2 = 0;
62 te = N N2; %sample number to stop iterations at
63 ybar = zeros(N2 N1+1,1);
64 Adelta = zeros(na+1,1);
65 %w = zeros(N2 N1+1,1);
66 G = zeros(N2 N1+1,na+1);
67 f = zeros(N2,1);
68 r = zeros(length(f)+1,1);
69 R_tmp = zeros(length(r)+Nu 1 ,Nu);

```

```

70 R = zeros(N2 N1+1,Nu);
71
72 %simulation and recursive estimation
73 %comment out the following line to leave noise in the system
74 %e = zeros(1,length(e));
75
76 for i = ts:te
77     %test first without estimating params
78     %Aest = A;
79     %Best = B(:,i);
80     %get estimated parameters from theta:
81     Aest = theta(1:na);
82     Best = theta(na+1:nparams);
83
84     %get Adelta from Aest
85     for j = 1:na+1
86         if j == 1
87             Adelta(j) = Aest(j) 1;
88         elseif j == na+1
89             Adelta(j) = Aest(j 1);
90         else
91             Adelta(j) = Aest(j) Aest(j 1);
92         end
93     end
94
95     %solve diophantine equation
96     G(1,:) = Adelta';
97     f(1) = G(1,1);
98     r(1) = Best(1);
99     r(2) = Best(2)+Best(1)*f(1);
100     for j = 1:N2 1
101         for k = 1:na+1
102             if k == na+1
103                 G(j+1,k) = Adelta(k)*f(j);
104             else
105                 G(j+1,k) = G(j,k+1) Adelta(k)*f(j);
106             end
107         end
108         f(j+1) = G(j+1,1);
109         %set up R matrix coefficients
110         if j == 1
111             r(j+2) = Best(3)+Best(2)*f(j)+Best(1)*f(j+1);
112         elseif j==2
113             r(j+2) = Best(4)+Best(3)*f(j 1)+Best(2)*f(j)+Best(1)*f(j+1);
114         else
115             r(j+2) = Best(4)*f(j 2)+Best(3)*f(j 1)+Best(2)*f(j)+Best(1)*f(j+1)
116                 ;
117         end
118     end
119     F = [1; f];

```

```

120 %set up dynamic matrix R
121 for j = 1:Nu
122     R_tmp(j:j+length(r)-1,j) = r;
123 end
124 R = R_tmp(N1:N2,:);
125
126 %simulate plant output
127 y(i) = A(1)*y(i-1) + A(2)*y(i-2) + B(1)*u(i-c(i)-1) ...
128         +B(2)*u(i-c(i)-2)+e(i);
129
130 %investigate altered reference values from paper
131 %for j = N1:N2
132     %w(j-N1+1) = alpha2*y(i)+(1-alpha2)*yr(i);
133 %end
134 %get simple future reference values
135 w = yr((i+N1):(i+N2));
136
137 %predict future outputs
138 for j = N1:N2
139     ybar(j-N1+1) = G(j,:)*y(i-1:i-2);
140 end
141
142 %update control action
143 Rmat = R'*R+rho*eye(min(size(R)));
144 delta_u = Rmat\((R')*(w-ybar));
145 u(i) = u(i-1)+delta_u(1);
146 if u(i)<0
147     u(i)=0;
148 end
149
150 %update parameter estimates
151 for j = 1:na
152     x(j) = y(i-j);
153 end
154 for j = 1:nb
155     x(na+j) = u(i-nk-j);
156 end
157 %RLS with forgetting factor
158 % K = (P*x)/(lambda+x'*P*x);
159 % P = (P - (P*(x*x')*P)/(lambda+x'*P*x))*(1/lambda);
160 % theta = theta+K*(y(i)-x'*theta);
161 % thetavec(i,:) = [Aest' Best'];
162
163 %EFRA
164 K = (P*x)/(lambda+x'*P*x);
165 P = (P - (P*(x*x')*P)/(lambda+x'*P*x))*(1/lambda) + ...
166     beta*eye(nparams) - gamma*P*P;
167 theta = theta+alpha3*K*(y(i)-x'*theta);
168 thetavec(i,:) = [Aest' Best'];
169 end
170

```



```

171 %Plot results
172 val=999;
173 figure(2)
174 stairs(t(1:val),u(1:val),'LineWidth',1);
175 ylabel({'Input, \itu\rm(\itk\rm)'});
176 xlabel({'Sample number, \itk'});
177 figure(3);
178 stairs1 = stairs(t(1:val),thetavec(1:val,:), 'LineWidth',1);
179 set(stairs1(1), 'DisplayName', 'a1');
180 set(stairs1(2), 'DisplayName', 'a2');
181 set(stairs1(3), 'DisplayName', 'b0');
182 set(stairs1(4), 'DisplayName', 'b1');
183 set(stairs1(5), 'DisplayName', 'b2');
184 set(stairs1(6), 'DisplayName', 'b3');
185 ylabel({'Estimated parameter values'});
186 xlabel({'Sample number, \itk'});
187 legend('show');
188 figure(1);
189 stairs2 = stairs(t(1:val),[y(1:val),yr(1:val)]);
190 set(stairs2(1), 'LineWidth',1);
191 set(stairs2(2), 'LineWidth',2, 'Color',[0 0 0]);
192 ylabel({'Output, \ity\rm(\itk\rm)'});
193 xlabel({'Sample number, \itk'});
194 ylim([0 2]);

```