Table of Contents

AE740 HW1 akshatdy

```
clc;
clear;
close all;
```

1. Proof

$J(x) = x^T Qx + c^T x$

Derivative of a scalar with respect to a vector

$\nabla_x c^T x = c$

Product Rule

$\nabla_x x^T Qx = (Q + Q^T)x$

since we know that Q is symmetric

$\nabla_x x^T Qx = 2Qx$

therefore,

$\nabla_x J(x) = 2Qx + c$

1.a Second term

$c^T x = \sum_{i=1}^{n} c_i x_i$

partial derivatives with respect to x_i

$\frac{\partial}{\partial x_i} c^T x = c_i$

therefore,

$\nabla_x c^T x = c$

1.b First term

$$x^TQx = \sum_{i=1}^{n}\sum_{j=1}^{n} x_iQ_{ij}x_j$$

partial derivatives with respect to x_i

$$\frac{\partial}{\partial x_i}x^TQx = \sum_{j=1}^{n} Q_{ij}x_j + \sum_{j=1}^{n} Q_{ji}x_j$$

since Q is symmetric, $Q_{ij} = Q_{ji}$

$$\frac{\partial}{\partial x_i}x^TQx = \sum_{j=1}^{n} Q_{ij}x_j + \sum_{j=1}^{n} Q_{ij}x_j$$

$$\frac{\partial}{\partial x_i}x^TQx = 2\sum_{j=1}^{n} Q_{ij}x_j$$

therefore,

$$\nabla_x x^TQx = 2Qx$$

for example, when n = 2

$$x^TQx = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x^TQx = x_1Q_{11}x_1 + x_1Q_{12}x_2 + x_2Q_{21}x_1 + x_2Q_{22}x_2$$

since Q is symmetric, $Q_{12} = Q_{21}$

$$x^TQx = x_1Q_{11}x_1 + 2x_1Q_{12}x_2 + x_2Q_{22}x_2$$

partial derivatives with respect to x_1

$$\frac{\partial}{\partial x_1}x^TQx = 2x_1Q_{11} + 2x_2Q_{12}$$

partial derivatives with respect to x_2

$$\frac{\partial}{\partial x_2}x^TQx = 2x_1Q_{12} + 2x_2Q_{22}$$

therefore,

$$\nabla_x x^TQx = \begin{bmatrix} 2Q_{11}x_1 + 2Q_{12}x_2 \\ 2Q_{12}x_1 + 2Q_{22}x_2 \end{bmatrix}$$

and

$$2Qx = 2\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2Q_{11}x_1 + 2Q_{12}x_2 \\ 2Q_{21}x_1 + 2Q_{22}x_2 \end{bmatrix}$$

therefore,

$$\nabla_x x^T Qx = 2Qx$$

1.c Hessian

$$\nabla_x J(x) = 2Qx + c$$

using the same rule as we used for the second part of the derivative

$$\nabla_x^2 J(x) = 2Q$$

Published with MATLAB® R2023b

Table of Contents


```
clc;
clear;
close all;
```

2

```
% setup matrices for part 2
A = [
    4/3 -2/3;
    1 0;
];
B = [
    1;
    0;
];
C = [-2/3 1];

% weighing matrices
Q = C' * C;
R = 0.001;
P = 0;
```

2.a

```
disp("2.a");
if all(abs(eig(A)) < 1)
    disp('all eigenvalues of A are inside the unit circle');
else
    disp('all eigenvalues of A are not inside the unit circle');
end
```

```
2.a
all eigenvalues of A are inside the unit circle
```

A is a Schur matrix if all eigenvalues are inside the unit circle, so A is a Schur matrix

```
if rank(ctrb(A, B)) == size(A, 1)
    disp('(A, B) has full rank');
else
    disp('(A, B) does not have full rank');
end
```

```
(A, B) has full rank
```

(A, B) controllability matrix has full rank, so there are no uncontrollable modes, so it is stabilizable

(A, B) is controllable if the controllability matrix has full rank, so it is controllable

```
if rank(obsv(A, C)) == size(A, 1)
    disp('(A, C) has full rank');
else
    disp('(A, C) does not have full rank');
end
```

```
(A, C) has full rank
```

(C, A) observability matrix has full rank, so there are no unobservable modes, so it is detectable

(C, A) is observable if the observability matrix has full rank, so it is observable

2.b Transfer function symbolically

```
disp("2.b");
z = sym('z');
disp("Symbolic transfer function");
disp(simplify(C * inv(z * eye(size(A)) - A) * B + 0));
disp("Eigenvalues of A");
disp(eig(A));
disp("Magnitude of eigenvalues of A");
disp(abs(eig(A)));
```

```
2.b
Symbolic transfer function
-(2*z - 3)/(3*z^2 - 4*z + 2)

Eigenvalues of A
   0.6667 + 0.4714i
   0.6667 - 0.4714i

Magnitude of eigenvalues of A
    0.8165
    0.8165
```

Transfer function is

$$-\frac{2z-3}{3z^2-4z+2}$$

Zeros(zeros of numerator)

$$2z-3=0$$

$$z=\frac{3}{2}$$

$$z=1.5$$

Zeroes are out of the unit disk

Poles(zeros of denominator)

$3z^2 - 4z + 2 = 0$

$z = \frac{2}{3} \pm \frac{\sqrt{2}}{3}$

$z = 0.6667 \pm 0.4714i$

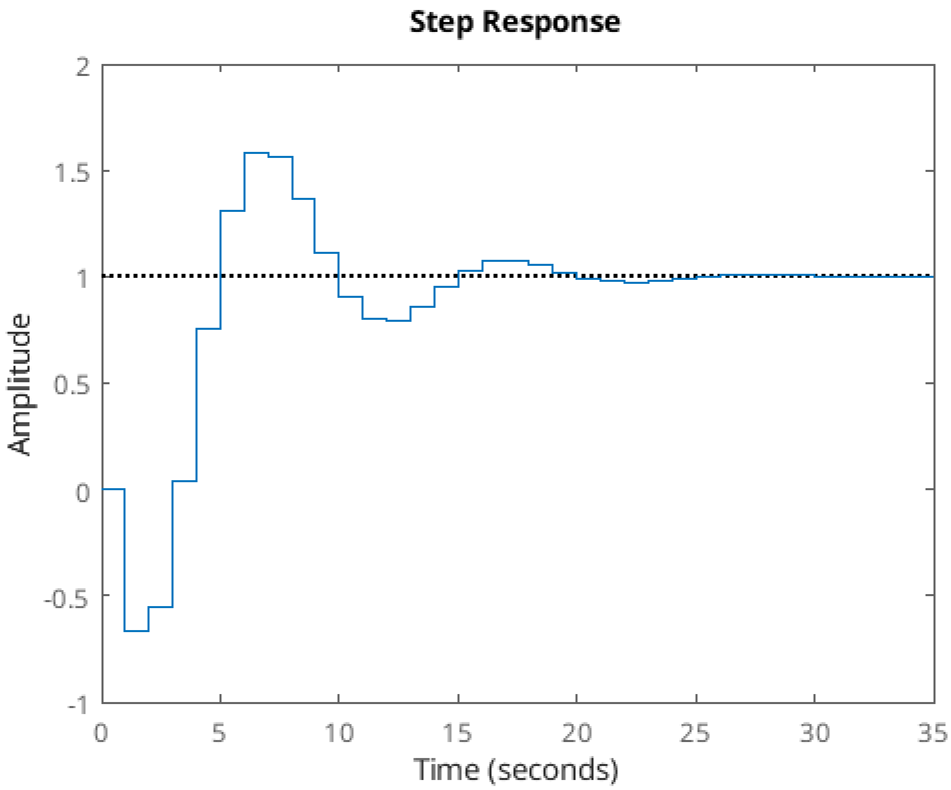
Poles are inside the unit disk

Poles are the same as eigenvalues of A

2.c Step response

```
disp("2.c");
dstep(A, B, C, 0);
```

2.c



Published with MATLAB® R2023b

Table of Contents


```
clc;
clear;
close all;
```

3

```
% setup matrices from part 2
A = [
    4/3 -2/3;
    1 0;
];
B = [
    1;
    0;
];
C = [-2/3 1];
Q = C' * C;
R = 0.001;
P = 0;
```

3.b Compute feedback gain using uncMPC

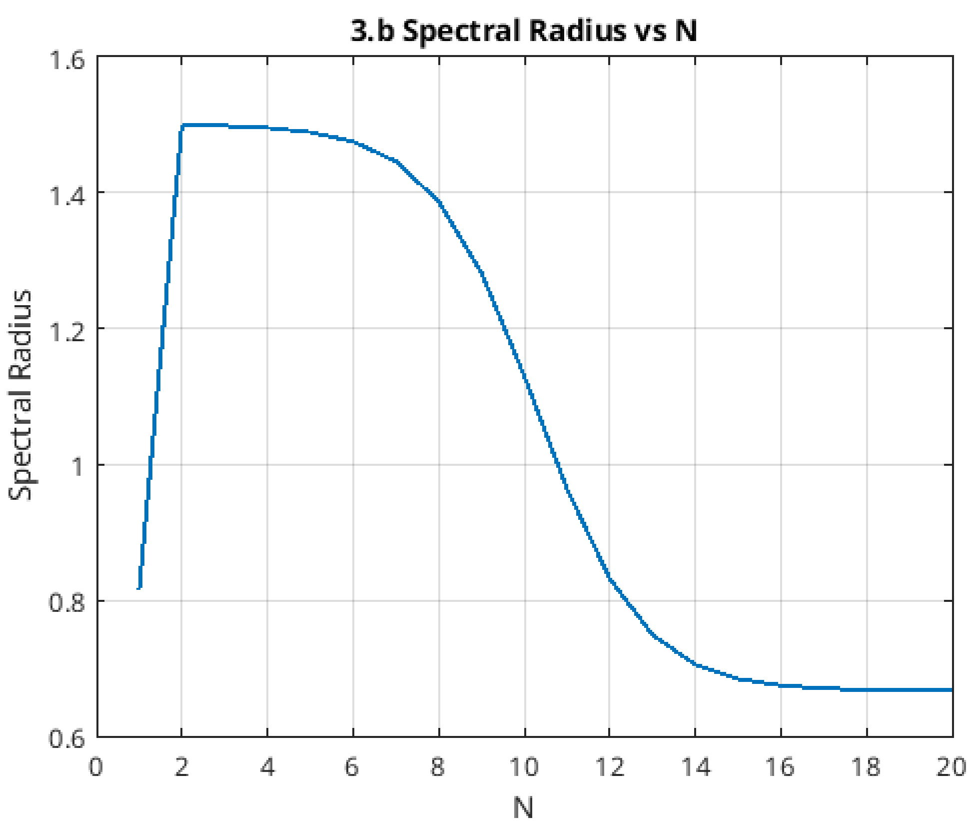
```
disp("3.b")
NHoriz = 20;
specRadList = zeros(20, 1);
eigvalList = zeros(NHoriz*2, 1);
K20 = []; % for use in 3.d
KList = zeros(NHoriz, 2); % for use in 3.e
for N = 1:NHoriz
    [S, M, Qbar, Rbar, K0N] = uncMPC(N, A, B, Q, R, P);
    KList(N, :) = K0N;
    eigval = eig(A + B * K0N);
    eigvalRow = (N-1) * 2 + 1;
    eigvalList(eigvalRow:eigvalRow+1, :) = eigval;
    specRad = max(abs(eigval));
    specRadList(N, 1) = specRad;
    % check if the max of abs of eigenvalues of A + BK are less than 1
    if specRad < 1
        disp("System is closed-loop stable at N = " + num2str(N));
```

```
    end
    if N == NHoriz
        K20 = K0N;
    end
end
```

3.b

The closed loop system is stable at N = 1 and N >= 11

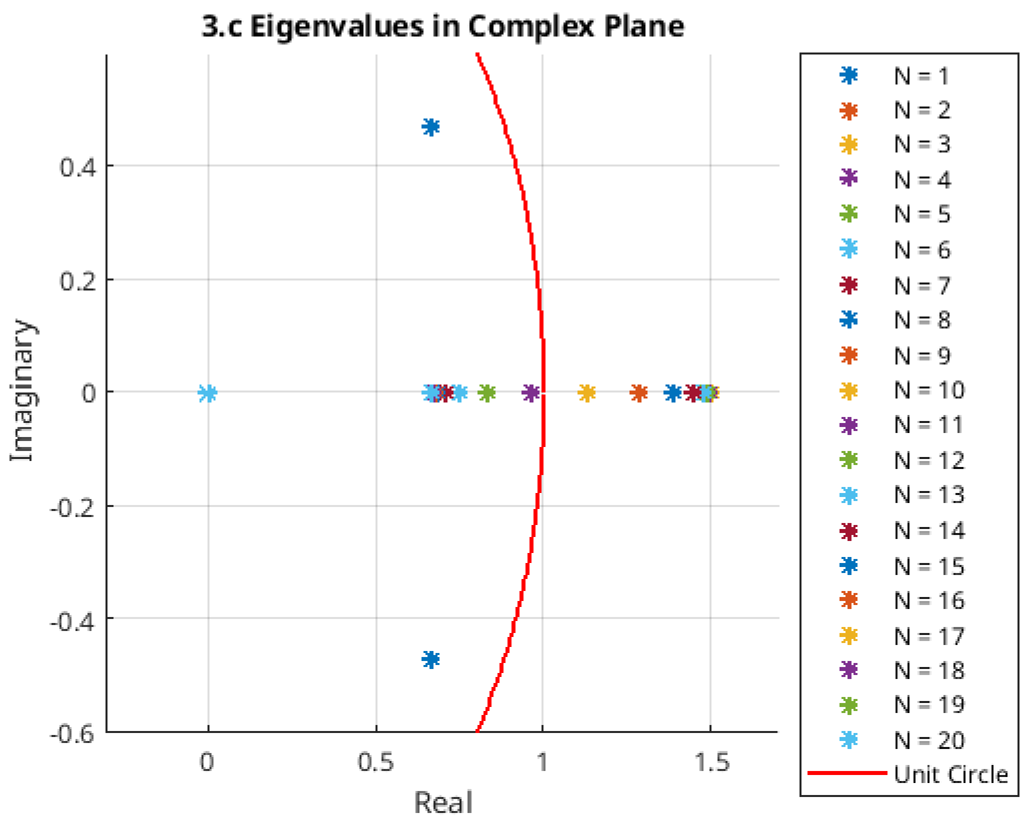
```
% plot the spectral radius vs N
figure(1);
plot(1:NHoriz, specRadList);
grid on;
title('3.b Spectral Radius vs N');
xlabel('N');
ylabel('Spectral Radius');
ylim([0.6, 1.6]);
```



3.c Plot eigenvalues in complex plane for N[1, 20]

```
figure(2);
hold on;
for N = 1:NHoriz
    eigvalRow = (N-1) * 2 + 1;
```

```
    plot(real(eigvalList(eigvalRow:eigvalRow+1, :)),
imag(eigvalList(eigvalRow:eigvalRow+1, :)), "*");
end
grid on;
title('3.c Eigenvalues in Complex Plane');
xlabel('Real');
xlim([-0.3, 1.7]);
ylabel('Imaginary');
ylim([-0.6, 0.6]);
% plot unit circle
th = 0:0.01:2*pi;
x = cos(th);
y = sin(th);
plot(x, y, 'r');
legend("N = 1", "N = 2", "N = 3", "N = 4", "N = 5", "N = 6", "N = 7", "N
= 8", "N = 9", "N = 10", "N = 11", "N = 12", "N = 13", "N = 14", "N = 15",
"N = 16", "N = 17", "N = 18", "N = 19", "N = 20", "Unit Circle", "Location",
"eastoutside");
hold off;
```



3.d Generate LQR gain using dlqr

```
disp("3.d")
[Kinf, P, e] = dlqr(A, B, Q, R, P);
display("Kinf = " + mat2str(-Kinf));
```

```
display("K20 = " + mat2str(K20));
display("norm(Kinf - K20) = " + num2str(norm(-Kinf - K20)));
```

3.d

```
    "Kinf = [-0.665912358378151 0.666001020246729]"

    "K20 = [-0.665600689796951 0.66600070894093]"

    "norm(Kinf - K20) = 0.00031167"
```

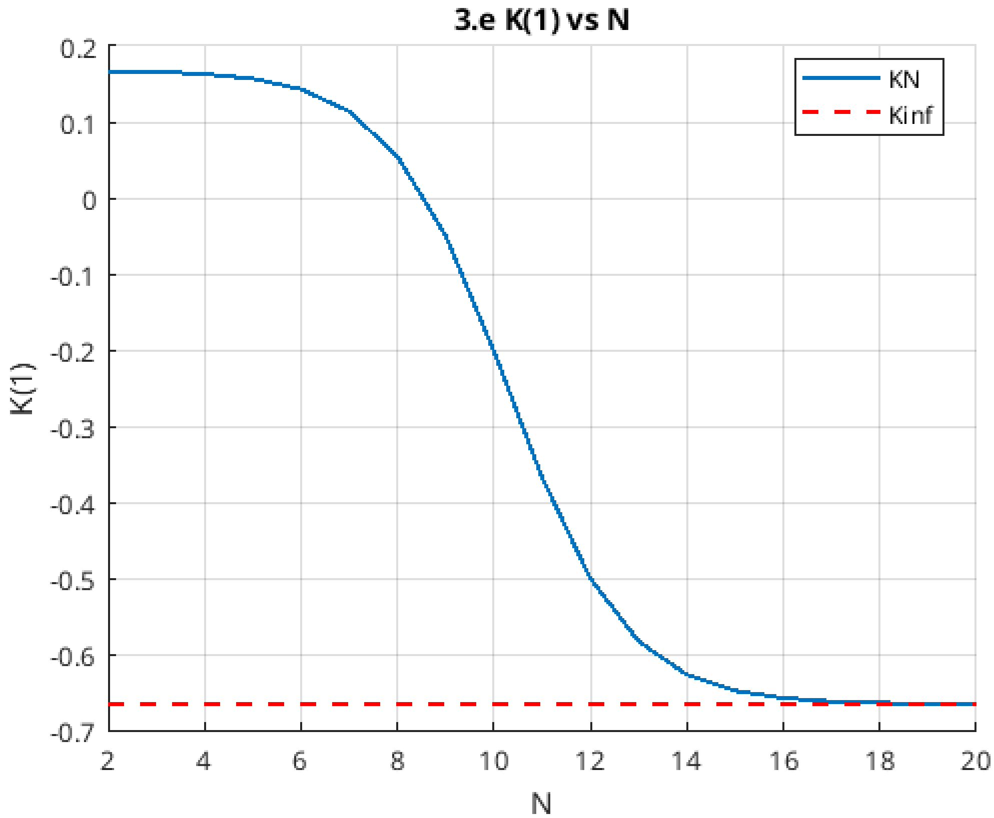
Yes, $K_{0,N}$ is close to K_∞ (norm of just 0.00031167) and $K_{0,N} \to K_\infty$ as $N \to \infty$

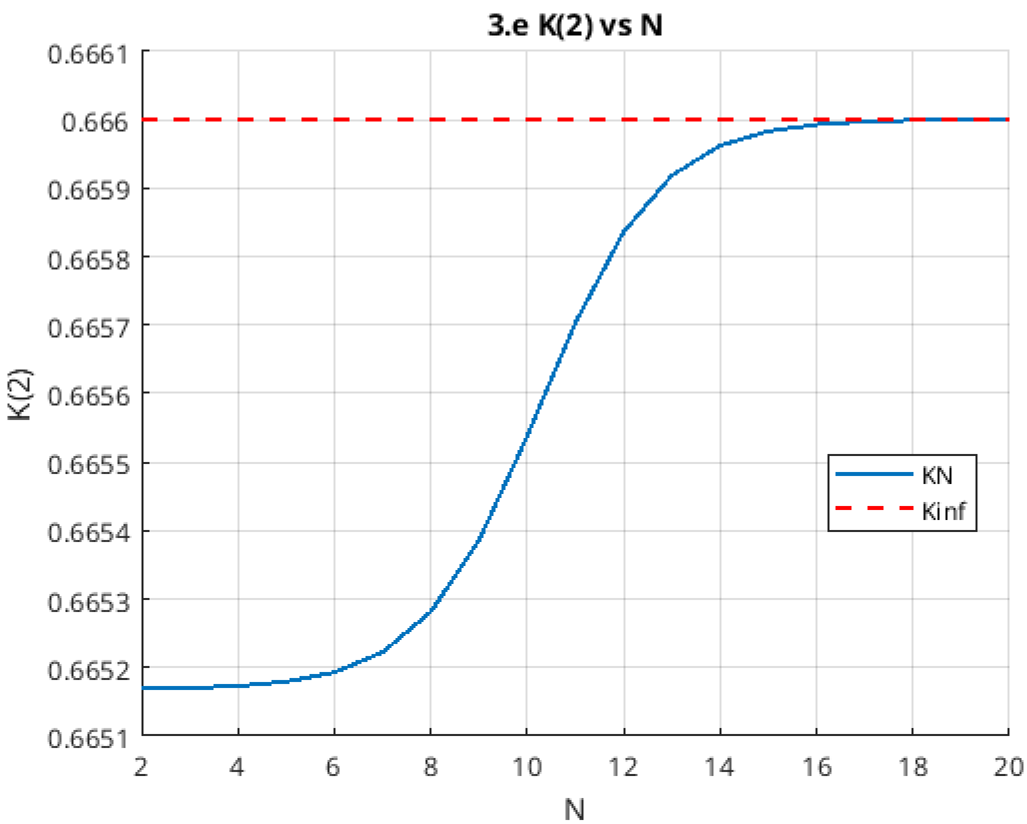
In addition, since (A, B) is stabilizing and (C, A) is detectable, the LQR gain is stabilizing.

3.e Plot the components of K

```
figure(3);
hold on;
grid on;
title('3.e K(1) vs N');
xlabel('N');
ylabel('K(1)');
plot(2:NHoriz, KList(2:end, 1));
% plot the limits as the Kinf values
plot([2, NHoriz], [-Kinf(1), -Kinf(1)], '--r');
legend("KN", "Kinf", "Location", "best")
hold off;

figure(4);
hold on;
grid on;
title('3.e K(2) vs N');
xlabel('N');
ylabel('K(2)');
plot(2:NHoriz, KList(2:end, 2));
% plot the limits as the Kinf values
plot([2, NHoriz], [-Kinf(2), -Kinf(2)], '--r');
legend("KN", "Kinf", "Location", "best")
```


3.e K(1) vs N
KN
Kinf
K(1)
N
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
2
4
6
8
10
12
14
16
18
20


3.e K(2) vs N
KN
Kinf
K(2)
N
0.6661
0.666
0.6659
0.6658
0.6657
0.6656
0.6655
0.6654
0.6653
0.6652
0.6651
2
4
6
8
10
12
14
16
18
20

Plotting without N = 1 to display the graph better

Yes, $K_{0,N}$ converges to K_∞ as $N \to \infty$

3.a Implement unconstrained LQ-MPC

```
function [S, M, Qbar, Rbar, K0N] = uncMPC(N, A, B, Q, R, P)
        % Compute the matrices S, M, Qbar, Rbar, and K0N
        % for the unconstrained LQ-MPC problem
        %
        % Inputs:
        %   N: Prediction horizon
        %   A: State matrix
        %   B: Input matrix
        %   Q: State cost matrix
        %   R: Input cost matrix
        %   P: Terminal state cost matrix

        nx = size(A, 1);
        nu = size(B, 2);

        % Initialize matrices
        S = zeros(N*nx, N*nu);
        M = zeros(N*nx, nx);
        Qbar = zeros(N*nx, N*nx);
        Rbar = zeros(N*nu, N*nu);

        % Compute the first column of S
        for i = 1:N
            rowStart = (i - 1) * nx + 1;
            rowEnd = i * nx;
            S(rowStart:rowEnd, 1:nu) = A^(i-1)*B;
        end
        % Pad the first column and set it to other columns of S
        for i = 2:N
            colStart = (i - 1) * nu + 1;
            colEnd = i * nu;
            zeroRows = (i - 1) * nx;
            zeroCols = nu;
            S(:, colStart:colEnd) = [zeros(zeroRows, zeroCols); S(1:end -
zeroRows, 1:nu)];
        end

        % Compute first row of M
        M(1:nx, :) = A;
        % Compute the rest of M
        for i = 2:N
            rowStart = (i - 1) * nx + 1;
            rowEnd = i * nx;
            % just multiply the previous rows by A to get higher powers
            M(rowStart:rowEnd, :) = A * M(rowStart - nx:rowEnd - nx, :);
        end
```

```
        % Compute Qbar except for the last row
        for i = 1:N
            % Q is square so we can reuse indices
            rowStart = (i - 1) * nx + 1;
            rowEnd = i * nx;
            temp = Q;
            if i == N
                temp = P;
            end
            Qbar(rowStart:rowEnd, rowStart:rowEnd) = temp;
        end

        % Compute Rbar
        for i = 1:N
            % R is square so we can reuse indices
            rowStart = (i - 1) * nu + 1;
            rowEnd = i * nu;
            Rbar(rowStart:rowEnd, rowStart:rowEnd) = R;
        end

        % Compute K0N
        K0N = -[eye(nu), zeros(nu, nu * (N - 1))] * inv(S'*Qbar*S + Rbar) *
S'*Qbar*M;
end
```

```
System is closed-loop stable at N = 1
System is closed-loop stable at N = 11
System is closed-loop stable at N = 12
System is closed-loop stable at N = 13
System is closed-loop stable at N = 14
System is closed-loop stable at N = 15
System is closed-loop stable at N = 16
System is closed-loop stable at N = 17
System is closed-loop stable at N = 18
System is closed-loop stable at N = 19
System is closed-loop stable at N = 20
```

Published with MATLAB® R2023b

Table of Contents


```
clc;
clear;
close all;
```

4

```
% setup matrices from part 2
A = [
    4/3 -2/3;
    1 0;
];
B = [
    1;
    0;
];
C = [-2/3 1];
Q = C' * C;
R = 0.001;
P = 0;
```

4.b Compare using problem 2

```
disp("4.b")
[Kinf, Pinf, e] = dlqr(A, B, Q, R, P);
disp("Using dlqr: Pinf=");disp(Pinf);
% for n=[2, 11, 20]
%     [S, M, Qbar, Rbar, K0N, P0N] = uncMPCric(n, A, B, Q, R, P);
%     disp("Using uncMPCric: N= " + num2str(n) + " P= " + mat2str(P0N) + "
norm(Pinf-P0N)= " + num2str(norm(Pinf-P0N)));
% end
% cant do loops cos then matlab wont publish the disp
disp("Using uncMPCric:");
n=2;
[S, M, Qbar, Rbar, K02, P0N] = uncMPCric(n, A, B, Q, R, P);
disp("N=" + num2str(n));disp("P=");disp(P0N);disp("norm(Pinf-
P0N)=");disp(norm(Pinf-P0N));
n=11;
[S, M, Qbar, Rbar, K011, P0N] = uncMPCric(n, A, B, Q, R, P);
disp("N=" + num2str(n));disp("P=");disp(P0N);disp("norm(Pinf-
```

```
P0N)=");disp(norm(Pinf-P0N));
n=20;
[S, M, Qbar, Rbar, K020, P0N] = uncMPCric(n, A, B, Q, R, P);
disp("N=" + num2str(n));disp("P=");disp(P0N);disp("norm(Pinf-
P0N)=");disp(norm(Pinf-P0N));
```

```
4.b
Using dlqr: Pinf=
    1.0005   -0.6671
   -0.6671    1.0004

Using uncMPCric:
```

The difference in the norm between P from dlqr and uncMPCric decreases as N increases and is 0.00020802 at N=20, which means that the two methods are converging.

4.c Compare using P = Pinf

```
disp("4.c")
disp("Using dlqr: Kinf=");disp(Kinf);disp("Pinf=");disp(Pinf);
% for n=[2, 11, 20]
%     [S, M, Qbar, Rbar, K0N, P0N] = uncMPCric(n, A, B, Q, R, Pinf);
%     disp("Using uncMPCric: N= " + num2str(n) + " K0N= " + mat2str(K0N) +
" norm(Kinf-K0N)= " + num2str(norm(-Kinf-K0N)) + " P= " + mat2str(P0N) + "
norm(Pinf-P0N)= " + num2str(norm(Pinf-P0N)));
% end
% cant do loops cos then matlab wont publish the disp
disp("Using uncMPCric:");
n=2;
[S, M, Qbar, Rbar, K0N, P0N] = uncMPCric(n, A, B, Q, R, Pinf);
disp("N=" + num2str(n));disp("K0N=");disp(K0N);disp("norm(Kinf-
K0N)= ");disp(norm(-Kinf-K0N));disp("P=");disp(P0N);disp("norm(Pinf-
P0N)=");disp(norm(Pinf-P0N));
n=11;
[S, M, Qbar, Rbar, K0N, P0N] = uncMPCric(n, A, B, Q, R, Pinf);
disp("N=" + num2str(n));disp("K0N=");disp(K0N);disp("norm(Kinf-
K0N)= ");disp(norm(-Kinf-K0N));disp("P=");disp(P0N);disp("norm(Pinf-
P0N)=");disp(norm(Pinf-P0N));
n=20;
[S, M, Qbar, Rbar, K0N, P0N] = uncMPCric(n, A, B, Q, R, Pinf);
disp("N=" + num2str(n));disp("K0N=");disp(K0N);disp("norm(Kinf-
K0N)= ");disp(norm(-Kinf-K0N));disp("P=");disp(P0N);disp("norm(Pinf-
P0N)=");disp(norm(Pinf-P0N));
```

```
4.c
Using dlqr: Kinf=
    0.6659   -0.6660

Pinf=
    1.0005   -0.6671
   -0.6671    1.0004

Using uncMPCric:
N=2
```

```
K0N=
   -0.6659    0.6660

norm(Kinf-K0N)=
   3.1264e-15

P=
    1.0005   -0.6671
   -0.6671    1.0004

norm(Pinf-P0N)=
   1.7764e-15

N=11
K0N=
   -0.6659    0.6660

norm(Kinf-K0N)=
   3.7894e-15

P=
    1.0005   -0.6671
   -0.6671    1.0004

norm(Pinf-P0N)=
   1.8966e-15

N=20
K0N=
   -0.6659    0.6660

norm(Kinf-K0N)=
   3.7894e-15

P=
    1.0005   -0.6671
   -0.6671    1.0004

norm(Pinf-P0N)=
   1.8966e-15
```

The difference in the norm between K from dlqr and uncMPCric is extremely small regardless of N, in the order of 1e-15.

The same is true is for P as well, so we can conclude that using P = Pinf in uncMPCric makes both P and K converge to the values from dlqr regardless of N.

By setting P=Pinf, the gains K for any prediction horizon N are are stabilizing.

4.d Proof

$$J = \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k$$

(Convert u to x) We have $u_k = K_k x_k$, substituting into J:

$$J = \sum_{k=0}^{\infty} x_k^T Q x_k + K^T x_k^T R K x_k$$

$$J = \sum_{k=0}^{\infty} x_k^T (Q + K^T R K) x_k$$

(Convert x_k to x_0) For closed-loop system, given control gain K:

$$x_{k+1} = Ax_k + Bu_k$$

$$x_{k+1} = Ax_k + BKx_k$$

$$x_{k+1} = (A + BK)x_k$$

$$x_{k+1} = (A + BK)x_k \to x_{k+1} = (A + BK)^{k+1} x_0 \to x_k = (A + BK)^k x_0$$

substituting into J:

$$J = \sum_{k=0}^{\infty} x_0^T ((A + BK)^k)^T (Q + K^T R K)(A + BK)^k x_0$$

$$J = x_0^T \left(\sum_{k=0}^{\infty} ((A + BK)^k)^T (Q + K^T R K)(A + BK)^k \right) x_0$$

Let $P_k = \sum_{k=0}^{\infty} ((A + BK)^k)^T (Q + K^T R K)(A + BK)^k$, then:

Then $J = x_0^T P_k x_0$

$$P_k = \sum_{k=0}^{\infty} ((A + BK)^k)^T (Q + K^T R K)(A + BK)^k$$

Multiply by $(A + BK)^T$ on the left and $(A + BK)$ on the right:

$$(A + BK)^T P_k (A + BK) = (A + BK)^T \left(\sum_{k=0}^{\infty} (A + BK)^k)^T (Q + K^T R K)(A + BK)^k \right) (A + BK)$$

$$(A + BK)^T P_k (A + BK) = \sum_{k=0}^{\infty} ((A + BK)^{k+1})^T (Q + K^T R K)(A + BK)^{k+1}$$

Subtract P_k from both sides:

$$(A+BK)^T P_k (A+BK) - P_k = \sum_{k=0}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1} - P_k$$

$$= \sum_{k=0}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1} - \sum_{k=0}^{\infty}((A+BK)^{k})^T(Q+K^TRK)(A+BK)^{k}$$

For $k=0$, $(A+BK)^T P_k (A+BK) - P_k$:

$$= ((A+BK)^1)^T(Q+K^TRK)(A+BK)^1 - ((A+BK)^0)^T(Q+K^TRK)(A+BK)^0$$

$$= (A+BK)^T(Q+K^TRK)(A+BK) - (Q+K^TRK)$$

For $k=[1,\infty]$, $(A+BK)^T P_k (A+BK) - P_k$:

$$= \sum_{k=1}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1} - \sum_{k=1}^{\infty}((A+BK)^{k})^T(Q+K^TRK)(A+BK)^{k}$$

Adding the two gives us: $(A+BK)^T P_k (A+BK) - P_k$:

$$= (A+BK)^T(Q+K^TRK)(A+BK) - (Q+K^TRK)$$

$$+ \sum_{k=1}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1} - \sum_{k=1}^{\infty}((A+BK)^{k})^T(Q+K^TRK)(A+BK)^{k}$$

$$= -(Q+K^TRK)$$

$$+(A+BK)^T(Q+K^TRK)(A+BK) + \sum_{k=1}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1}$$

$$- \sum_{k=1}^{\infty}((A+BK)^{k})^T(Q+K^TRK)(A+BK)^{k}$$

We know that if $k=0$, the first term in the summation:

$$\sum_{k=0}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1} = (A+BK)^T(Q+K^TRK)(A+BK)$$

So:

$$(A+BK)^T(Q+K^TRK)(A+BK) + \sum_{k=1}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1}$$

$$= \sum_{k=0}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1}$$

additionally,

$$\sum_{k=0}^{\infty}((A+BK)^{k+1})^T(Q+K^TRK)(A+BK)^{k+1} = \sum_{k=1}^{\infty}((A+BK)^k)^T(Q+K^TRK)(A+BK)^k$$

so, $(A+BK)^TP_k(A+BK)-P_k$:

$$= -(Q+K^TRK)$$

$$+\sum_{k=1}^{\infty}((A+BK)^k)^T(Q+K^TRK)(A+BK)^k$$

$$-\sum_{k=1}^{\infty}((A+BK)^k)^T(Q+K^TRK)(A+BK)^k$$

$$= -(Q+K^TRK)$$

Hence:

$$(A+BK)^TP_k(A+BK)-P_k = -(Q+K^TRK)$$

the Lyapunov equation is:

$$(A+BK)^TP_k(A+BK)-P_k+(Q+K^TRK)=0$$

Rewriting the equation:

$$(A+BK)^TP_k(A+BK)-P_k = -(Q+K^TRK)$$

Which is the same result we got earlier, hence, P_k is the solution to the Lyapunov equation.

4.e Compute finite horizon performance

```
disp("4.e")
x0 = [1;-0.5];
% For N=inf
Kinf = -Kinf;
ABK = A + B*Kinf;
QKRK = Q + Kinf'*R*Kinf;
Pk = dlyap(ABK', QKRK);
Jinf = x0'*Pk*x0;
disp("Cost under N=inf: " + num2str(Jinf));
% For N=11
ABK = A + B*K011;
QKRK = Q + K011'*R*K011;
Pk = dlyap(ABK', QKRK);
J11 = x0'*Pk*x0;
disp("Cost under N=11: " + num2str(J11));
```

```
4.e
Cost under N=inf: 1.9178
Cost under N=11: 3.1952
```

The cost under N=inf is a lot lower than N=11, which is expected since the cost is minimized over a much longer horizon.

4.a Riccati recursion

```
function [S, M, Qbar, Rbar, K0N, P0N] = uncMPCric(N, A, B, Q, R, P)
    % Compute the matrices S, M, Qbar, Rbar, and K0N
    % for the unconstrained LQ-MPC problem
    %
    % Inputs:
    %   N: Prediction horizon
    %   A: State matrix
    %   B: Input matrix
    %   Q: State cost matrix
    %   R: Input cost matrix
    %   P: Terminal state cost matrix

    nx = size(A, 1);
    nu = size(B, 2);

    % Initialize matrices
    S = zeros(N*nx, N*nu);
    M = zeros(N*nx, nx);
    Qbar = zeros(N*nx, N*nx);
    Rbar = zeros(N*nu, N*nu);

    Pk = P;

    for k=1:N
        Kkm1 = -inv(R + B'*Pk*B)*B'*Pk*A;
        Pkm1 = Q + A'*Pk*A + A'*Pk*B*Kkm1;

        Pk = Pkm1;
    end
    K0N = Kkm1;
    P0N = Pkm1;
end
```

```
N=2
P=
    0.4445   -0.6666
   -0.6666    1.0004

norm(Pinf-P0N)=
    0.5561

N=11
P=
    0.8016   -0.6669
   -0.6669    1.0004
```

```
norm(Pinf-P0N)=
    0.1990

N=20
P=
    1.0003   -0.6671
   -0.6671    1.0004

norm(Pinf-P0N)=
   2.0802e-04
```

Published with MATLAB® R2023b

Table of Contents


```
clc;
clear;
close all;
```

5

5.c Continuous time linearized model

```
disp('5.c');
x0 = [0; 0; 0; 0; 0; 0; 0; 0; 0];
u0 = [0; 0; 0];
syms J1 J2 J3 phi theta psi omega1 omega2 omega3 r1 r2 r3 M1 M2 M3;
x = [phi; theta; psi; omega1; omega2; omega3; r1; r2; r3];
u = [M1; M2; M3];
J = [J1; J2; J3];
f = dxdtwJ(x, u, J);
A = jacobian(f, x);
B = jacobian(f, u);
Ac = subs(A, [x; u], [x0; u0]);
Bc = subs(B, [x; u], [x0; u0]);

disp('Ac = ');
disp(Ac);
disp('Bc = ');
disp(Bc);
```

```
5.c
Ac =
[0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
Bc =
[   0,    0,    0]
[   0,    0,    0]
[   0,    0,    0]
[1/J1,    0,    0]
[   0, 1/J2,    0]
[   0,    0, 1/J3]
[   0,    0,    0]
[   0,    0,    0]
[   0,    0,    0]
```

5.d Discrete time linearized model

```
disp('5.d');
Ts = 2;
Ac = double(subs(A, [x; u; J], [x0; u0; [120; 100; 80]]));
Bc = double(subs(B, [x; u; J], [x0; u0; [120; 100; 80]]));
Cc = zeros(size(Ac, 1), size(Ac, 2));
Dc = zeros(size(Cc, 1), size(Bc, 2));
[Ad, Bd, Cd, Dd] = c2dm(Ac, Bc, Cc, Dc, Ts, 'zoh');
% check if Ad is schur
disp('Eigenvalues of Ad = ');
disp(eig(Ad));
if all(abs(eig(Ad)) < 1)
    disp('Ad is schur');
else
    disp('Ad is not schur');
end
% check if Ad, Bd is controllable
% we dont care about controlling the r1, r2, r3 states, because they are
reference
% hence ignore the last 3 states
if rank(ctrb(Ad, Bd)) == size(Ad, 1)-3
    disp('(Ad, Bd) is controllable');
else
    disp('(Ad, Bd) is not controllable');
end
```

```
5.d
Eigenvalues of Ad =
     1
     1
     1
     1
     1
     1
     1
     1
     1
```

```
Ad is not schur
(Ad, Bd) is controllable
```

Ad is not a schur matrix since the eigenvalues are not inside the unit circle

Ad, Bd is controllable since the rank of the controllability matrix is equal to the number of states that we want to control (6)

5.e Implement terminal cost function

```
disp('5.e');
Q = diag([1, 1, 1, 0.01, 0.01, 0.01]);
R = diag([1,1,1])*0.01;
Ad = Ad(1:6, 1:6);
Bd = Bd(1:6, :);
global Pinf;
[Kinf, Pinf, e] = dlqr(Ad, Bd, Q, R);
disp('Pinf = ');
disp(Pinf);
```

```
5.e
Pinf =
    3.0005    0.0000   -0.0000    6.0002   -0.0000   -0.0000
    0.0000    2.7919    0.0000    0.0000    5.0002    0.0000
   -0.0000    0.0000    2.5622    0.0000   -0.0000    4.0003
    6.0002    0.0000    0.0000   30.0125    0.0000    0.0000
   -0.0000    5.0002   -0.0000    0.0000   22.9250   -0.0000
   -0.0000    0.0000    4.0003    0.0000   -0.0000   16.5042
```

5.f Simulate the system

```
disp('5.f');
import casadi.*
mpc = import_mpctools();
Nx = 9;
Nu = 3;
Nt = 30;
Delta = 2;
Nsim = 100;
N = struct('x', Nx, 'u', Nu, 't', Nt);
x = NaN(Nx, Nsim+1);
x(:, 1) = [-0.4; -0.8; 1.2; -0.02; -0.02; 0.02; 0; 0; 0];
u = NaN(Nu, Nsim);
for i=0:1
    for t = 0:Nsim
        if i==0
            odeFun = @(x,u) ode(x,u); % nonlinear
        else
            odeFun = @(x,u) (Ac*x + Bc*u); % linear
        end
        f = mpc.getCasadiFunc(odeFun, [Nx, Nu], {'x', 'u'}, 'rk4', true,
'Delta', Delta, 'M', 1);
```

```
        l = mpc.getCasadiFunc(@stagecost, [Nx, Nu], {'x', 'u'}, {'l'});
        Vf = mpc.getCasadiFunc(@termcost, [Nx], {'x'}, {'Vf'});
        lbx = -inf*ones(Nx, Nt+1);
        lbx(4:6, :) = -0.03;
        ubx = inf*ones(Nx, Nt+1);
        ubx(4:6, :) = 0.03;
        lbu = -0.1*ones(Nu, Nt);
        ubu = 0.1*ones(Nu, Nt);
        commonargs = struct('l', l, 'Vf', Vf, 'lb', struct('u', lbu, 'x',
lbx), 'ub', struct('u', ubu, 'x', ubx));
        solvers = mpc.nmpc('f', f, 'N', N, 'Delta', Delta, '**', commonargs);
        if t < Nsim/2
            r = [0; 0; 0];
        else
            r = [0.5; 0; -0.5];
        end
        x(7:9,t+1) = r;
        solvers.fixvar('x', 1, x(:, t+1));
        solvers.solve();
        % fprintf('%d: %s\n', t, solvers.status);
        if ~isequal(solvers.status, 'Solve_Succeeded')
            warning('%s failed at time %d', solvers.name, t);
        end
        solvers.saveguess();
        u(:, t+1) = solvers.var.u(:, 1);
        x(:, t+2) = solvers.var.x(:, 2);
    end

    % plot the figures
    set(0, 'DefaultLineLineWidth', 1.5);

    Time=0:Delta:(Nsim+1)*Delta;

    figure(i*3 +1);
    if i==0
        sgtitle("Euler Angles Nonlinear");
    else
        sgtitle("Euler Angles Linear");
    end
    subplot(3,1,1)
    plot(Time,x(1,:))
    grid on;
    hold on
    plot(Time,x(7,:),'-.')
    set(gca,'xticklabel',[])
    ylabel('phi')
    xlim([Time(1),Time(end)]);
    legend('MPC','setpoint','Fontsize',12)


    subplot(3,1,2)
    plot(Time,x(2,:))
    grid on;
    hold on
```

```
plot(Time,x(8,:),'-.')
set(gca,'xticklabel',[])
ylabel('theta')
xlim([Time(1),Time(end)]);


subplot(3,1,3)
plot(Time,x(3,:))
grid on;
hold on
plot(Time,x(9,:),'-.')
set(gca,'xticklabel',[])
ylabel('psi')
xlim([Time(1),Time(end)]);
snapnow;

figure(i*3 + 2);
if i==0
sgtitle("Angular velocities Nonlinear")
else
sgtitle("Angular velocities Linear")
end
subplot(3,1,1)
plot(Time,x(4,:))
grid on;
hold on
set(gca,'xticklabel',[])
ylabel('w1')
xlim([Time(1),Time(end)]);
yline(0.03,'--','LineWidth',1);
yline(-0.03,'--','LineWidth',1);
legend('MPC', 'Upper bound', "Lower bound", 'Fontsize',12);

subplot(3,1,2)
plot(Time,x(5,:))
grid on;
hold on
set(gca,'xticklabel',[])
ylabel('w2')
xlim([Time(1),Time(end)]);
yline(0.03,'--','LineWidth',1);
yline(-0.03,'--','LineWidth',1)

subplot(3,1,3)
plot(Time,x(6,:))
grid on;
hold on
set(gca,'xticklabel',[])
ylabel('w3')
xlim([Time(1),Time(end)]);
yline(0.03,'--','LineWidth',1);
yline(-0.03,'--','LineWidth',1)
snapnow;
```

```
    figure(i*3 + 3);
    if i==0
    sgtitle("Inputs Nonlinear")
    else
    sgtitle("Inputs Linear")
    end
    subplot(3,1,1)
    plot(Time(2:end),u(1,:));
    grid on;
    hold on
    set(gca,'xticklabel',[])
    ylabel('M1')
    xlim([Time(1),Time(end)]);
    yline(0.1,'--','LineWidth',1);
    yline(-0.1,'--','LineWidth',1)
    legend('MPC', 'Upper bound', "Lower bound", 'Fontsize',12)

    subplot(3,1,2)
    plot(Time(2:end),u(2,:))
    grid on;
    hold on;
    set(gca,'xticklabel',[]);
    ylabel('w2');
    xlim([Time(1),Time(end)]);
    yline(0.1,'--','LineWidth',1);
    yline(-0.1,'--','LineWidth',1)

    subplot(3,1,3)
    plot(Time(2:end),u(3,:))
    grid on;
    hold on
    set(gca,'xticklabel',[])
    ylabel('w3')
    xlim([Time(1),Time(end)]);
    yline(0.1,'--','LineWidth',1);
    yline(-0.1,'--','LineWidth',1)
    snapnow;
end
```

5.f

5.a Implement continuous time system function

```
function dxdt = ode(x, u)
    J = [120; 100; 80];
    dxdt = dxdtwJ(x, u, J);
end

function dxdtwJ = dxdtwJ(x, u, J)
    J1 = J(1);
```

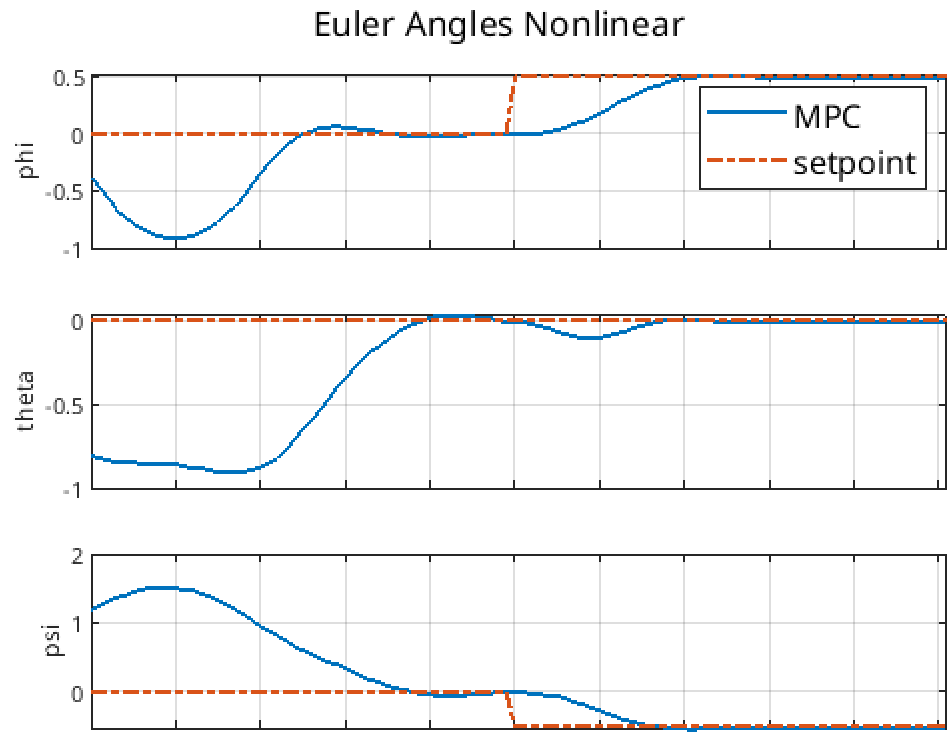
```
    J2 = J(2);
    J3 = J(3);
    phi = x(1);
    theta = x(2);
    % psi = x(3);
    omega1 = x(4);
    omega2 = x(5);
    omega3 = x(6);
    % r1 = x(7);
    % r2 = x(8);
    % r3 = x(9);
    M1 = u(1);
    M2 = u(2);
    M3 = u(3);
    dstatedt = 1/cos(theta) * ...
        [
            cos(theta) sin(phi)*sin(theta) cos(phi)*sin(theta);
            0 cos(phi)*cos(theta) -sin(phi)*cos(theta);
            0 sin(phi) cos(theta);
        ] * ...
        [omega1; omega2; omega3];
    dinputdt = [
        ((J2 - J3) * omega2 * omega3)/J1 + M1/J1;
        ((J3 - J1) * omega3 * omega1)/J2 + M2/J2;
        ((J1 - J2) * omega1 * omega2)/J3 + M3/J3;
    ];
    dxdtwJ = [
            dstatedt;
            dinputdt;
            [0; 0; 0];
        ];
end
```

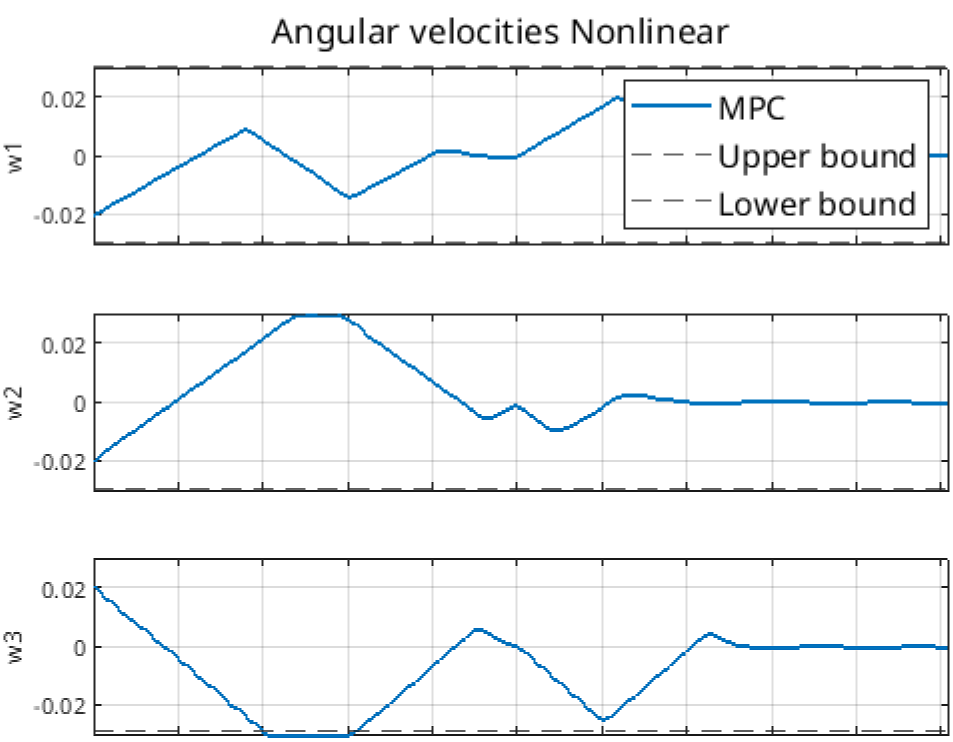
5.b Implement stage cost function

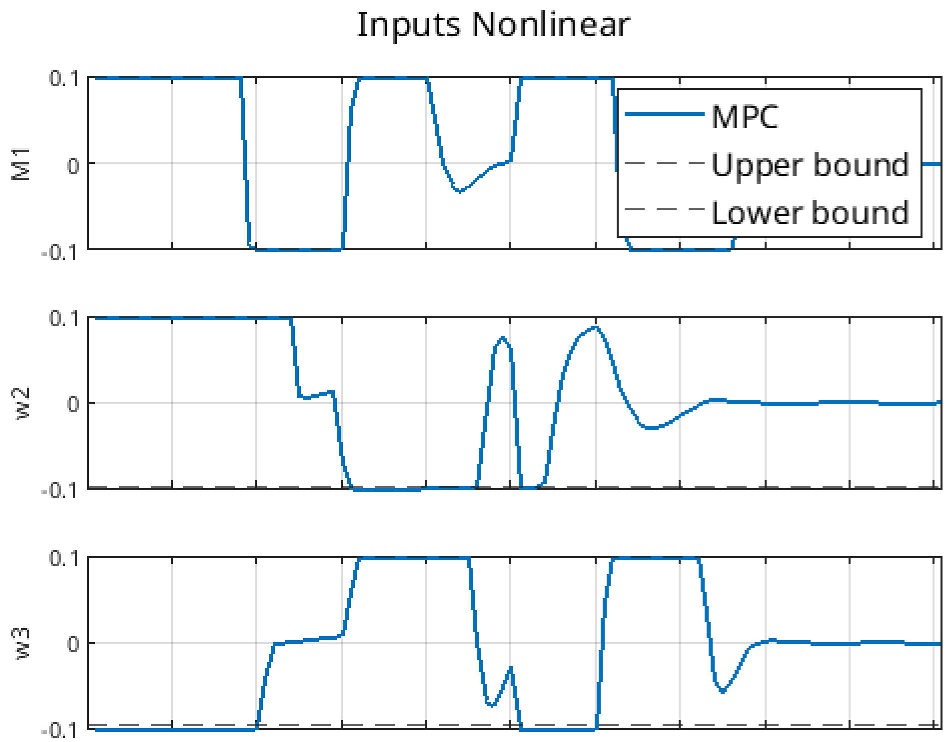
```
function l = stagecost(x, u)
    r = x(7:9);
    Q = diag([1, 1, 1, 0.01, 0.01, 0.01]);
    R = diag([1,1,1])*0.01;
    e = [x(1:3)-r(:);x(4:6)];
    l = e'*Q*e + u'*R*u;
end
```

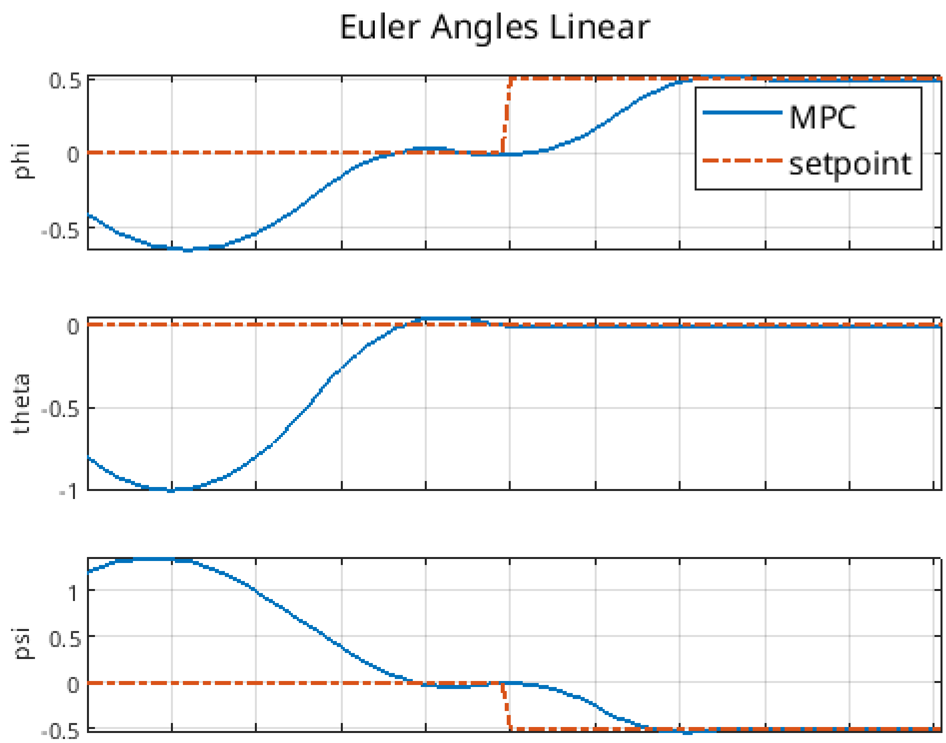
5.e Implement terminal cost function

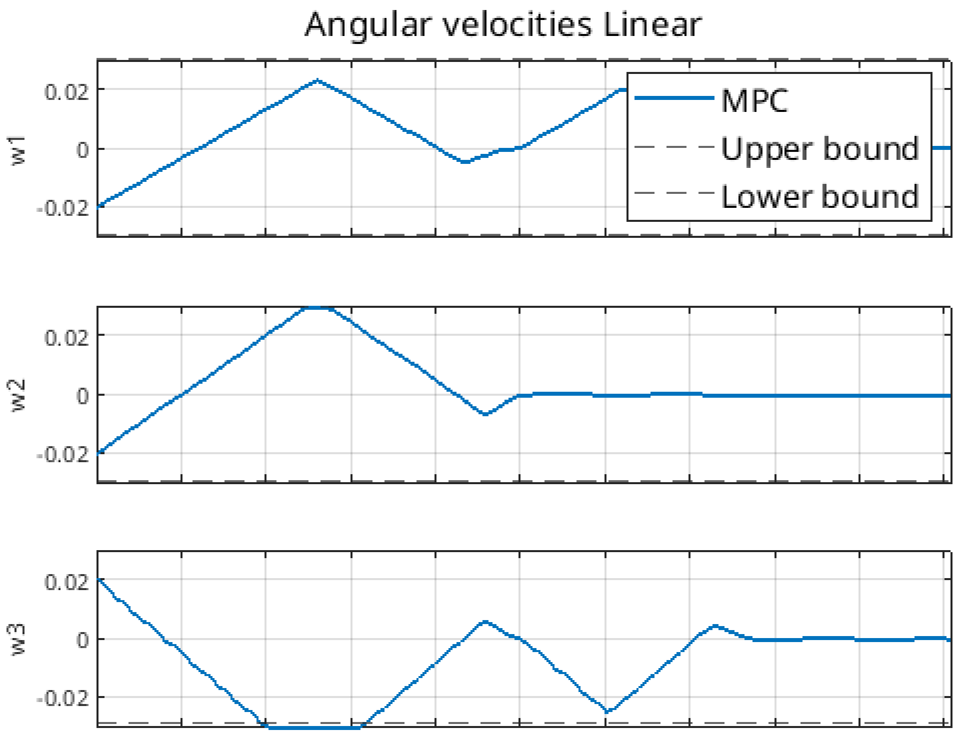
```
function Vf = termcost(x)
    r = x(7:9);
    e = [x(1:3)-r(:);x(4:6)];
    global Pinf;
    Vf = e'*Pinf*e;
end
```

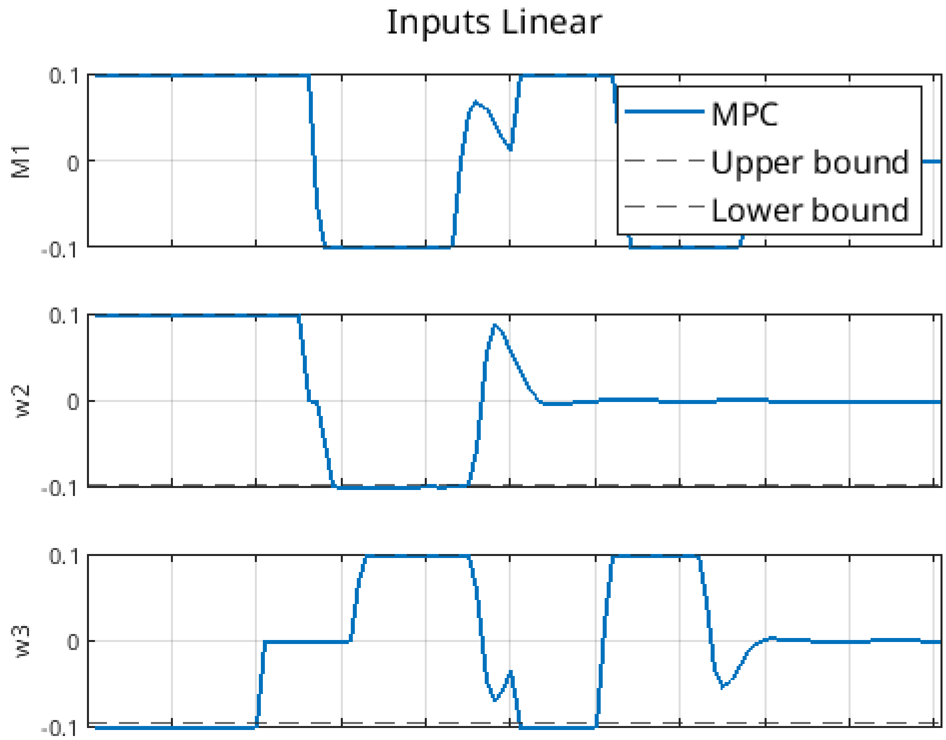

Euler Angles Nonlinear
phi
theta
psi
MPC
setpoint
0.5
0
-0.5
-1
-1
2
1


Angular velocities Nonlinear
w1
w2
w3
MPC
Upper bound
Lower bound
0.02
0
-0.02


Inputs Nonlinear
M1
w2
w3
0.1
0
-0.1
MPC
Upper bound
Lower bound


Euler Angles Linear
phi
theta
psi
0.5
0
-0.5
-1
1
MPC
setpoint


Angular velocities Linear
w1
w2
w3
0.02
0
-0.02
MPC
Upper bound
Lower bound


Inputs Linear
M1
w2
w3
0.1
0
-0.1
MPC
Upper bound
Lower bound

Published with MATLAB® R2023b