Model Predictive Control (MPC) for Process Systems

This document explains step by step the pure Octave implementation of MPC for a first-order plus dead-time (FOPDT) process system.

Step 1: Process Parameters

octave

```
K = 2;      % Process gain
T = 10;     % Time constant
L = 3;      % Dead time (in samples)
Ts = 1;     % Sampling time
N = 100;    % Simulation length
```

K (gain): How strongly the system output reacts to input changes.

T (time constant): How quickly the system responds (large T = slower response).

L (dead time): Delay before the input affects the output (common in chemical processes).

Ts: Sampling time for discrete simulation.

N: Number of simulation steps.

We're modeling a first-order plus dead-time (FOPDT) system, which is widely used in process control.

Step 2: Discrete-Time Approximation

octave

```
a = exp(-Ts/T);          % decay factor
b = K*(1-a);             % input gain
```

Converts the continuous system into a difference equation.

a determines how much of the previous output carries over.

b determines how much the input contributes each step.

Equation:

$y(k)=a*y(k-1)+b*u(k-L)$

Step 3: MPC Parameters

octave

```
PredictionHorizon = 20;
ControlHorizon = 5;

lambda = 0.1;   % weight on input moves
Q = 1;          % weight on output error

u_min = 0; u_max = 100;   % input constraints
```

Prediction horizon: How far ahead MPC predicts the system output.

Control horizon: How many future control moves it optimizes.

Q: Weight on tracking error (output vs setpoint).

lambda: Weight on input changes (penalizes aggressive moves).

Constraints: Input limited between 0 and 100.

This balances performance vs smoothness.

Step 4: Initialization

octave

```
y = zeros(N,1);       % process output
u = zeros(N,1);       % control input
r = ones(N,1)*50;     % setpoint = 50
disturbance = zeros(N,1);
disturbance(40:60) = 10; % synthetic disturbance

u(1) = 10;                    % initial kick
u_buffer = ones(L,1)*u(1);     % initialize buffer with non-zero input
```

y: Output history.

u: Input history.

r: Desired setpoint (constant 50).

disturbance: External disturbance added between steps 40-60.

Initial kick: Ensures the system doesn't stay at zero.

Dead-time buffer: Stores past inputs to simulate delay.

Step 5: Simulation Loop

octave

```
for k = 2:N
    % Process update
    u_delayed = u_buffer(end);
    u_buffer = [u(k-1); u_buffer(1:end-1)];
    y(k) = a*y(k-1) + b*u_delayed + disturbance(k);

    % MPC optimization
    candidates = linspace(u_min,u_max,20);
```

```
    bestJ = Inf; bestU = u(k-1);

    for cand = candidates
        y_temp = y(k);
        u_buf_temp = [cand; u_buffer(1:end-1)];
        u_temp = u_buf_temp(end);
        y_pred_test = zeros(PredictionHorizon,1);

        for j = 1:PredictionHorizon
            y_temp = a*y_temp + b*u_temp;
            y_pred_test(j) = y_temp;
        end

        ref_segment = r(k:min(N,k+PredictionHorizon-1));
        len = length(ref_segment);
        J = sum(Q*(ref_segment - y_pred_test(1:len)).^2) ...
            + lambda*(cand - u(k-1))^2;

        if J < bestJ
            bestJ = J;
            bestU = cand;
        end
    end

    u(k) = bestU;
end
```

Updates the process output using the difference equation.

Tests candidate inputs between min and max.

Predicts future outputs for each candidate.

Evaluates cost function (tracking error + input move penalty).

Selects the input with the lowest cost.

This is a simplified MPC using greedy search instead of quadratic programming.

Step 6: Plot Results

octave

```
figure;
subplot(2,1,1);
plot(1:N,y,'b','LineWidth',1.5); hold on;
plot(1:N,r,'r--','LineWidth',1.5);
xlabel('Time step'); ylabel('Output');
legend('Output','Setpoint');
title('MPC Control of Process System (Pure Octave)');
grid on;

subplot(2,1,2);
stairs(1:N,u,'k','LineWidth',1.5);
xlabel('Time step'); ylabel('Control Input');
title('MPC Control Input');
grid on;
```

Top plot: Output vs setpoint shows tracking performance.

Bottom plot: Control input shows how MPC adjusts inputs.

Summary

This implementation demonstrates MPC principles without external packages. It predicts future outputs, evaluates costs, and chooses the best input to control a process system.