Industrial Instrumentation & Control

Assignment Report

Submitted by –

Yash Sarawgi 2016A8PS0198P

Kaustubh Namjoshi 2016A8PS0406P

Paper title - Design of PID controller with incomplete derivation based on differential evolution algorithm.

PID controller with incomplete derivation - In general, derivation control can improve the dynamic behaviour of a control system, but a pure derivative cannot and should not be implemented, because it will give a very large amplification of measurement noise. To overcome this drawback, a low-pass filter is often added to derivation term. The derivation control with a low-pass filter is called the incomplete derivation control. PID controller with incomplete

derivation has better control performances compared with general PID controller.

$$U(s) = \left(K_p + \frac{K_p}{T_i s} + \frac{K_p T_d s}{1 + T_f s}\right) E(s)$$

Performance criteria of PID controller –

Minimum phase systems –

$$W(K) = (1 - e^{-\beta}) \cdot (M_p + E_{ss}) + e^{-\beta} \cdot (t_s + t_r)$$

where K is [Kp, Ki, Kd], β ∈ [0.8, 1.5] is the weighting factor, Mp is the overshoot, ts is the setting time, tr is the rise time, and the Ess is the steady-state error.

Non minimum phase systems –

$$W(K) = (1 - e^{-\beta}) \cdot (M_p + E_{ss} + |M_u|) + e^{-\beta} \cdot (t_s + t_r)$$

where Mu is peak undershoot.

Differential Evolution –

DE has three operations: mutation, crossover, and selection. The crucial idea behind DE is a scheme for generating trial vectors. Mutation and crossover are used to generate trial vectors, and selection then determines which of the vectors will survive into the next generation.

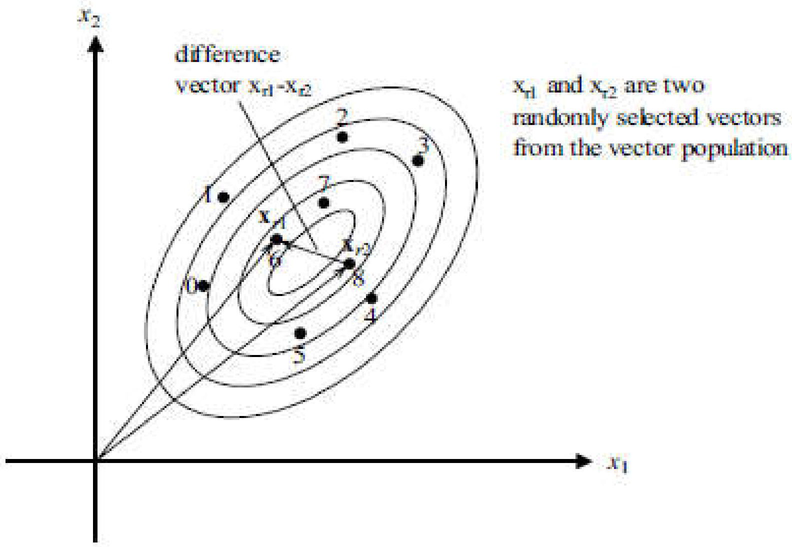
Code –

```
function Xres = DE_PID(X,F,CR,Np,caseType,n)
    G = 400;
    cost = zeros(G,1);
    for i=1:G
        v = Mutation(X,F,Np);
        u = Crossover(X,v,CR,Np);
        X = nextGeneration(X,Np,caseType,u,n);
        ind = findFinal(X,caseType,Np,n);
        cost(i,1) = computeCost3(X(:,ind),n);
    end
    plot(cost);
    Xres = X;
end
```

Mutation

For each target vector xGij , a mutant vector v is generated according to

$$v_i^{G+1} = x_{r1}^G + F \cdot (x_{r2}^G - x_{r3}^G)$$



Generating the perturbation: $\mathbf{x}_{r1} - \mathbf{x}_{r2}$

Code –

```
function v = Mutation(X,F,Np)
    v= zeros(3,Np);
    for i=0:Np-1
        while(1)
            r1 = round(rand*(Np-1));
            r2 = round(rand*(Np-1));
            r3 = round(rand*(Np-1));
            if(r1~=i && r2~=i && r3~=i && r1~=r2 && r2~=r3 && r1~=r3)
                break;
            end
        end

        v(:,i+1) = X(:,r1+1)+(F*(X(:,r2+1)-X(:,r3+1)));
    end

end
```

Crossover

- Generate a 1×N random number vector and compare it with constant Crossover Rate (CR), which results in a logical constant given as follows:

  $K_1^{(i)}$ = rand(1×N) ≤ CR

  For example: $K_1^{(i)}$ = [0.9 0.3 0.8 0.5 0.2] ≤ 0.8 = [0 1 1 1 1]

- Generate another logical constant $K_2^{(i)}$ which is complement of $K_1^{(i)}$ as follows:

  $K_2^{(i)} = K_1^{(i)} < 0.5$

For example: $K_2^{(i)}$ = [0 1 1 1 1] < 0.5 = [1 0 0 0 0]

- Generate trial vector as a result of crossover operation between mutation and population vectors as given below:

  $\boldsymbol{u}^{(i)} = K_1^{(i)} \boldsymbol{v}^{(i)} + K_2^{(i)} \boldsymbol{x}^{(i)}$

- Above equation means that if random values in crossover operation are

  - less than or equal to *CR*, take element resulting from mutation operation as trial population element, greater than *CR*, take current population element as trial population element.

Code –

```
function u = Crossover(X,v,CR,Np)
    u = zeros(3,Np);
    for i=0:Np-1
        K2 = rand(3,1)>CR;
        K1 = K2<0.5;
        u(:,i+1) =  (K1.*v(:,i+1))+(K2.*X(:,i+1));
    end
end
```

Selection

- Evaluate fitness function $F(\boldsymbol{u})$ for the trial vector $\boldsymbol{u}$.

- Compare $F(\boldsymbol{u}^{(i)})$ with $F(\boldsymbol{x}^{(i)})$ such that if

  - $F(\boldsymbol{u}^{(i)}) \leq F(\boldsymbol{x}^{(i)})$, then $\boldsymbol{x}^{(i)} := \boldsymbol{u}^{(i)}$ i.e. trial vector replaces current population vector for next generation DE optimization.

  - $F(\boldsymbol{u}^{(i)}) > F(\boldsymbol{x}^{(i)})$, then $\boldsymbol{x}^{(i)} := \boldsymbol{x}^{(i)}$ i.e. keep current population vector for next generation DE optimization.

```
function Xres = nextGeneration(X,Np,CaseType,u,n)
       Xres = zeros(3,Np);
       for i=0:Np-1
           if(CaseType==1)
               XCost = computeCost1(X(:,i+1),n);
               uCost = computeCost1(u(:,i+1),n);
           elseif(CaseType==2)
               XCost = computeCost2(X(:,i+1),n);
               uCost = computeCost2(u(:,i+1),n);
           elseif(CaseType==3)
               XCost = computeCost3(X(:,i+1),n);
               uCost = computeCost3(u(:,i+1),n);
           else
               disp("Invalid Case");
               return;
           end

           if(uCost<XCost)
               Xres(:,i+1) = u(:,i+1);
```

```
        else
            Xres(:,i+1) = X(:,i+1);
        end


    end
end
```

Summarized Steps:

Step 1 Specify the number of population NP, difference vector scale factor F, crossover probability constant CR, and the maximum number of generations. Initialize randomly the individuals of the population and the trial vector in the given searching space.

Step 2 Use each individual as the PID controller parameters and calculate the values of the four performance criteria of the system unit step response in the time domain, namely Mp, Ess, tr and ts.

Step 3 Calculate the fitness value of each individual in the population using the performance criterion function

Step 4 Compare the fitness value of each individual and get the best fitness and best individual.

Step 5 Generate a mutant vector for each individual.

Step 6 Do the crossover operation and yield a trial vector.

Step 7 Do the selection operation in terms of (15) and generate a new population.

Step 8 G = G+1, return to Step 2 until to the maximum number of generations.

Simulation:-

Case 1 (Three-order system)

$$G_1(s) = \frac{6\ 068}{s(s^2 + 110s + 6\ 068)}$$

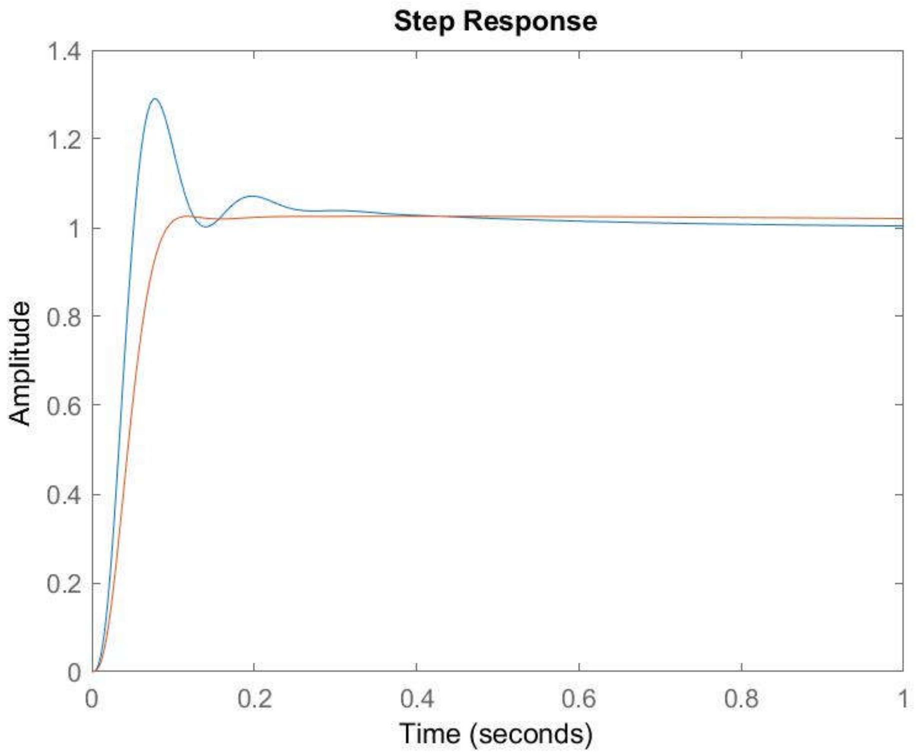
Case 2 (Time-delay system)

$$G_2(s) = \frac{2e^{-s}}{1 + 5s}$$

Case 3 (Non-minimum phase system)

$$G_3(s) = \frac{1 - 0.5s}{(1 + s)^3}$$

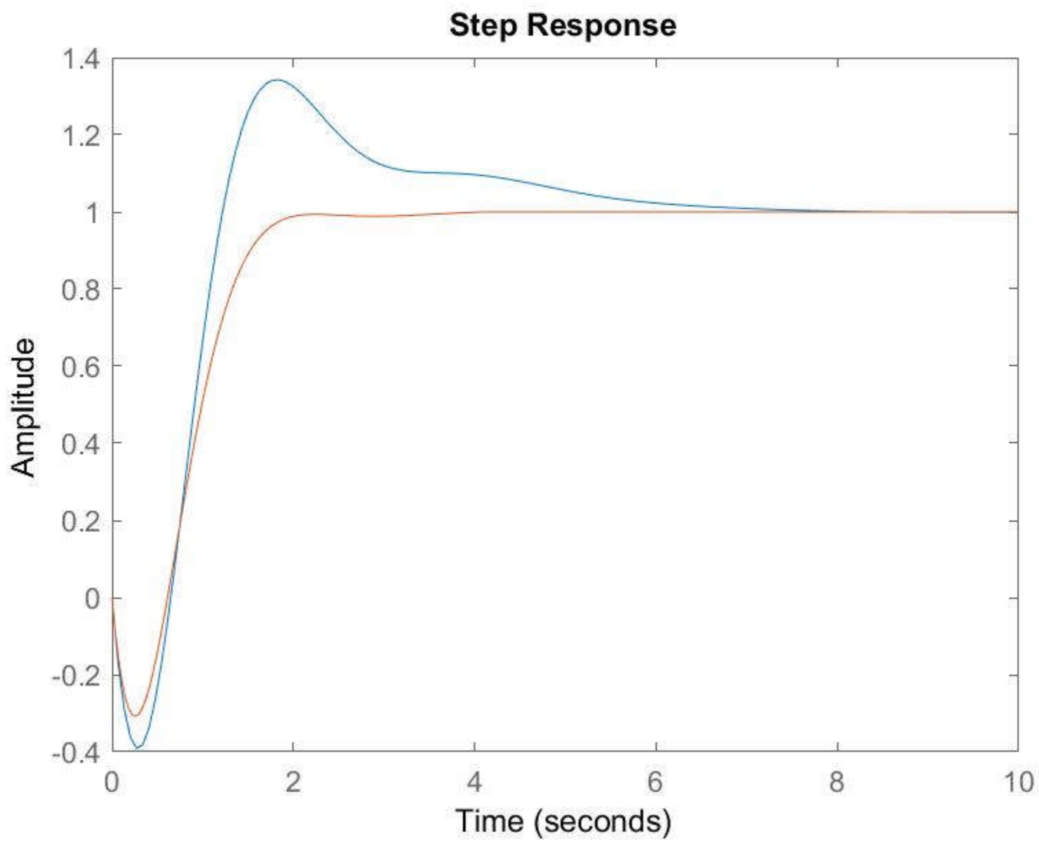
Case1:




```
>> main(1)
   33.1040

    0.3018

    0.0724


ans =

  struct with fields:

        RiseTime: 0.0310
    SettlingTime: 0.5060
     SettlingMin: 0.9029
     SettlingMax: 1.2905
       Overshoot: 29.0537
      Undershoot: 0
            Peak: 1.2905
        PeakTime: 0.0780
```

```
   18.5594

    1.0935

    0.0728


ans =

  struct with fields:

        RiseTime: 0.0536
    SettlingTime: 1.0563
     SettlingMin: 0.9101
     SettlingMax: 1.0257
       Overshoot: 2.5722
      Undershoot: 0
            Peak: 1.0257
        PeakTime: 0.3936
```

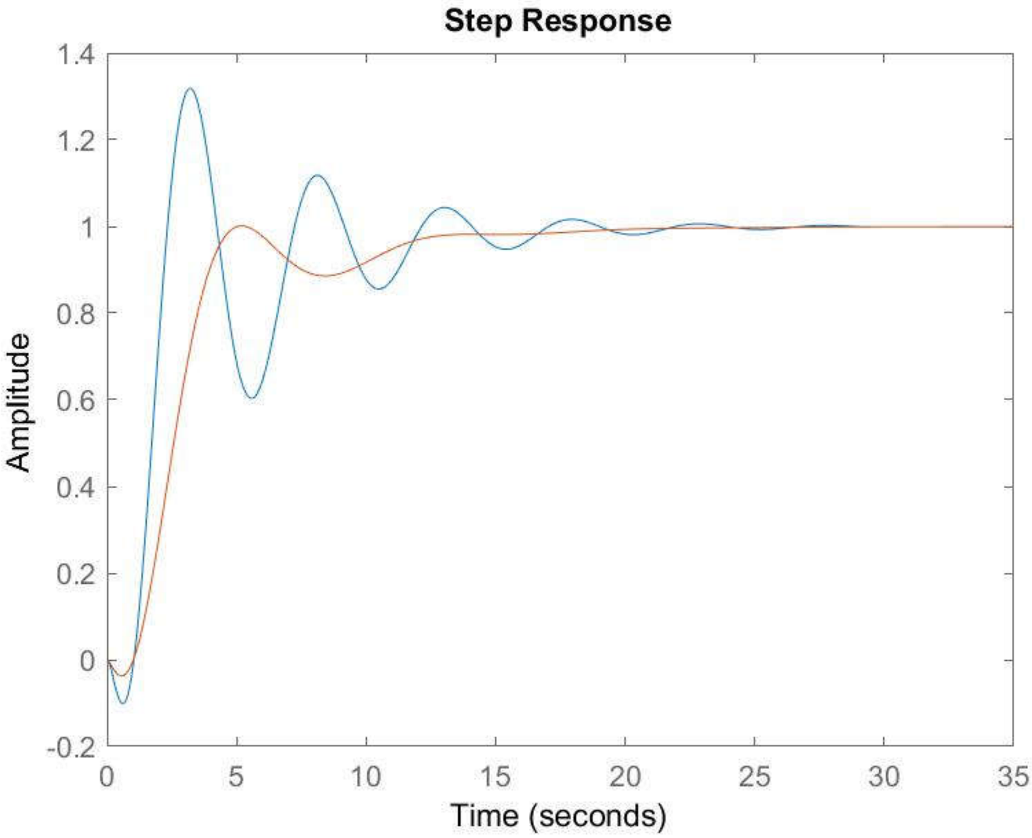
Case 2:




```
>> main(2)
    5.5000

    2.3900

    0.3586


ans =

  struct with fields:

        RiseTime: 0.4442
    SettlingTime: 5.7818
     SettlingMin: 0.9167
     SettlingMax: 1.3426
       Overshoot: 34.2580
      Undershoot: 39.1231
            Peak: 1.3426
        PeakTime: 1.8376
```

```
    4.4642

    5.2192

    0.3620


ans =

  struct with fields:

        RiseTime: 0.8392
    SettlingTime: 1.8311
     SettlingMin: 0.9076
     SettlingMax: 1.0000
       Overshoot: 0.0045
      Undershoot: 30.7654
            Peak: 1.0000
        PeakTime: 4.1067
```

Case 3:




```
    1.9200

    4.4200

    0.6637


ans =

  struct with fields:

       RiseTime: 0.9813
   SettlingTime: 16.5067
    SettlingMin: 0.6033
    SettlingMax: 1.3186
      Overshoot: 31.8617
     Undershoot: 10.1286
           Peak: 1.3186
       PeakTime: 3.2154
```

```
    1.0572

    3.5001

    0.1883


ans =

  struct with fields:

       RiseTime: 2.4950
   SettlingTime: 12.9679
    SettlingMin: 0.8857
    SettlingMax: 1.0011
      Overshoot: 0.1136
     Undershoot: 3.7363
           Peak: 1.0011
       PeakTime: 5.2021
```