SERBIAN JOURNAL OF ELEKCTRICAL ENGINEERING
Vol. 7, No. 1, May 2010, 95-119

UDK: 623.74:681.516.3

Using of Particle Swarm for Control of Helicopter

Alireza Rezaee[1]

Abstract: The CE150 Helicopter is one of the ranges offered by HUMUSOFT for teaching systems dynamics and control engineering principles. It has two degrees of freedom and is a MIMO system. We consider only Azimuth system for identification and control. There are many approaches to system identification. We choose NN structure and train it by Back-propagation and then use GA and PSO for optimizing the NN's training. At last we design PSO controller of Azimuth system.

Keywords: CE150 Helicopter Model, System identification, State feedback control, Particle Swarm, Neural Networks, Genetic Algorithm, Matlab.

1 Introduction

The CE150 Helicopter Model is one of the unique ranges of products designed for the theoretical study and practical investigation of basic and advanced control engineering principles.

This includes system dynamics modeling, identification, analysis and various controllers design by classical and modern methods. The model consists of a body carrying two DC motors. These motors drive the propellers. The body has two degrees of freedom. The axes of the body rotation are perpendicular as well as the axes of the motors. Both body position angles, i.e. azimuth angle in horizontal (ϕ) and elevation angle in vertical plane (ψ) are influenced by the rotating propellers simultaneously. But torque that is produced by main propeller is more effective on (ψ) and the torque of side propeller is effective on (ϕ) rather than the other. The helicopter model is a multivariable dynamical system with up to three manipulated inputs $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, $\boldsymbol{u}_3$ ($\boldsymbol{u}_3$ is the model of disturbance) and two measured outputs ϕ, ψ. All inputs and outputs are coupled. The system is essentially nonlinear and at least of the sixth order, depending on the modeling precision.

The mathematical Model can be linearized around the operating point [1, 2].

Schematics diagram of helicopter model is shown in Fig. 1.

[1]Islamic Azad University Abhar Branch; E-mail: arrezaee@yahoo.com

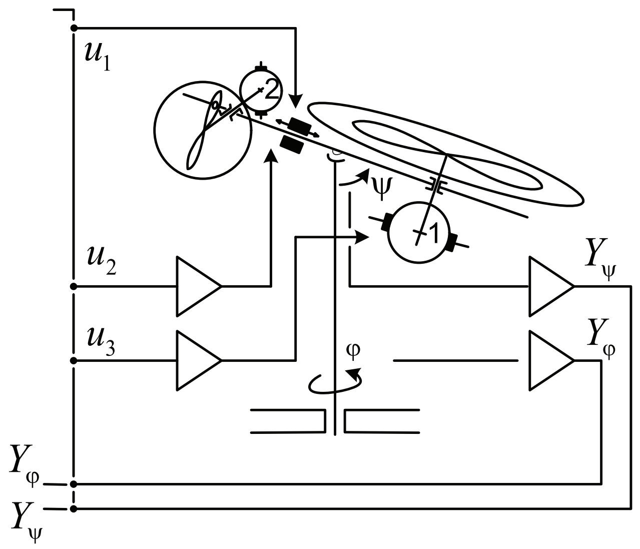

Fig. 1 – *Helicopter model.*

In this project, the main goal is to identify and control of SISO model, so we can eliminate the coupling between azimuth and elevation by fastening the special screw that is on the main body of helicopter and considering the side motor as an actuator and azimuth angle as measured output regardless the influence of the cross couplings between the elevation and azimuth dynamics. Then SISO subsystem of Azimuth can be like Fig. 2 [3].

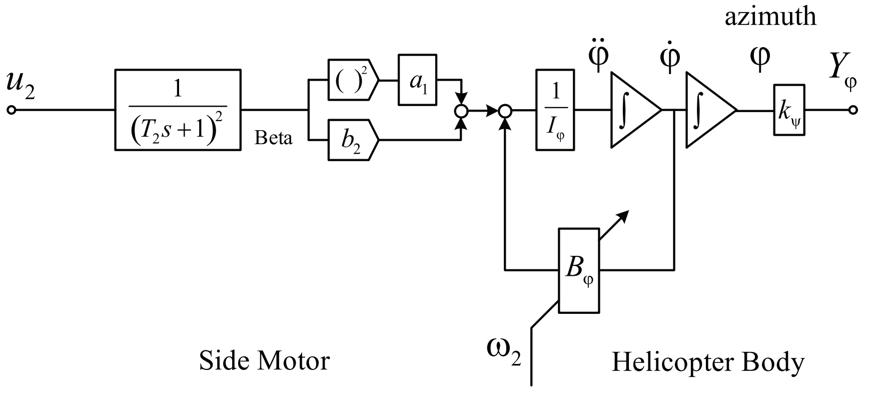

Fig. 2 – *Azimuth model.*

Nonlinear model by using physical and electrical rules is obtained like the following (1) for Azimuth rotation.

$$\boldsymbol{f}_{\phi}(\boldsymbol{z}(t),\boldsymbol{u}(t),t)=\begin{bmatrix} z_2 \\ \frac{1}{I_{\phi}}(-B_{\phi}z_2+a_2z_3^2+b_2z_3) \\ z_4 \\ \frac{1}{T_2^2}(u_2-z_3+2T_2z_4) \end{bmatrix},$$

$$\boldsymbol{g}_{\phi}(\boldsymbol{z}(t),\boldsymbol{u}_2(t),t)=\begin{bmatrix} z_1 \end{bmatrix}, \tag{1}$$

$$\boldsymbol{z}(t)=\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}=\begin{bmatrix} \phi \\ \dot{\phi} \\ \beta \\ \dot{\beta} \end{bmatrix}.$$

The operating point of system is $\hat{\boldsymbol{z}}=\begin{bmatrix} \hat{z}_1 \\ \hat{z}_2 \\ \hat{z}_3 \\ \hat{z}_4 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$.

Around the operating point, we linearize model [4] and consider this linear model for identification, control and show how it is helpful and correct for nonlinear system.

Equations (2) shows the linear model:

$$\boldsymbol{A}_{\phi}=\boldsymbol{J}_z[\hat{\boldsymbol{z}}(t),\hat{\boldsymbol{u}}_2,t]=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.2791 & 6.8372 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -16 & -8 \end{bmatrix},$$

$$\boldsymbol{B}_{\phi}=\boldsymbol{J}_u[\hat{\boldsymbol{z}}(t),\hat{\boldsymbol{u}}_2,t]=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 16 \end{bmatrix}, \tag{2}$$

$$\boldsymbol{C}_{\phi}=\boldsymbol{H}_x[\hat{\boldsymbol{z}}(t),\hat{\boldsymbol{u}}_2(t),t]=\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix},$$

$$\boldsymbol{D}_{\phi}=\boldsymbol{H}_u[\hat{\boldsymbol{z}}(t),\hat{\boldsymbol{u}}_2(t),t]=\begin{bmatrix} 0 \end{bmatrix}.$$

System poles are at 0, –0.2791, –4 and –4. The repeated poles are related to dynamics of side motor. The 0 and –0.2791 are dominating poles and –4, –4 are insignificant poles (the step response and root-locus plot and Nyquist). Although none of poles are unstable but with a very low gain it passes $j\omega$ axis and easily becomes unstable.

One of the requirements in system identification is the collection of 'information rich' input/output data. The Azimuth angle does not give us enough information about system. The system becomes unstable quickly. In order to adequate model it is necessary to stabilize it using a feedback controller. By using a feedback controller, the output data will contain more information describing the process. A full state feedback controller is developed to stabilize and control the linear Azimuth system. Because of controllability and observability of system, the full state feedback controller can stabilize the system by positioning the closed loop poles in the stable region according to **Table 1** [5]. We use integrator for eliminating the steady state error. The proper closed loop poles to reach desired performance can specify by using SISO Tools of Matlab.

Table 1
Closed loop poles.

Closed loop poles	–0.4+0.48j	–0.4–j0.48	–4	–5	–6

In this way, system becomes stable and has desired response as like Fig. 3.

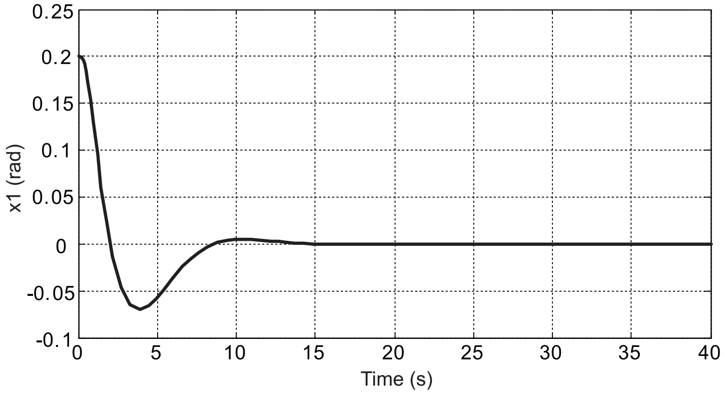

Fig. 3 – *Controlled system response.*

Developing a controller for the non-linear system is more difficult. Linear control techniques such as full-state feedback were tested and it is almost successful in controlling the non-linear system, too. But we must always consider that all the following design and discussion are true only around operating point of system and their realization for nonlinear system must be checked [2].

2 Identification

In control engineering, system identification is employed to determine a model of the system (plant) subject to control. In this context, system models describe the behavior of the plant over time as it is exposed to control and influence from external factors. System identification consists of two subtasks:

(i) Identification parts by direct measurement of physically accessible parameters and identification of model subsystems;

(ii) Processing the input and output signals, considering the system as a black box (data driven method).

The first method is time consuming but it gives good understanding of the system but in many systems it's so hard and almost impossible. The second approach is general, elegant. But it must be used around the chosen set point in nonlinear system. We follow second approach to identify Azimuth system.

A system identification problem can be formulated as an optimization task where the objective is to find a model and a set of parameters that minimize the prediction error between system output $y(t)$, i.e., the measured data, and model output $\hat{y}(t,\hat{\theta})$ at each time-step t (Fig. 4).

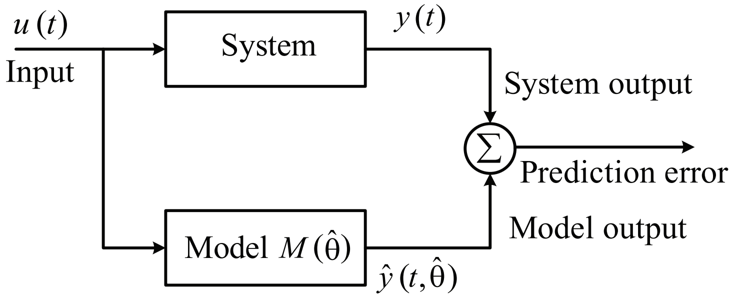

Fig. 4 – *Data driven identification.*

For system identification, we use Artificial Neural Network. In this way, we have a parametric structure that model system. First neural network is trained by back propagation and then we optimize it by **GA** (Genetic Algorithm) and **PSO** (Particle Swarm Optimization). The stages of developing identification are discussed in details in the following section.

2.1 Multi Layer Perceptron (MLP)

Artificial Neural Networks (ANN) provides a general method for learning arbitrary mapping between two data sets. A typical ANN contains a number of adjustable parameters called weights. In particular, supervised *learning* involves finding a set of weights that minimizes the mapping error. In our case, mapping error is defined the difference between observed output and NN's output. Fig. 5 shows a typical MLP ANN [4, 6].

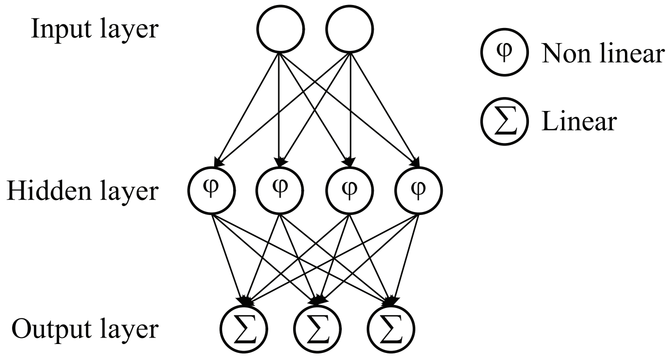

Fig. 5 – *Typical* MLP NN.

ANN has some noticeable advantages:

(i) The main advantage of neural networks is possibility of training it to perform a particular function by adjusting the values of connections (weights) between elements and their structures.

(ii) Neural networks are composed of elements operating in parallel. Parallel processing increases speed of calculation compared to slower sequential processing.

(iii) It has memory.

The number of hidden layers and neuron in each layer is problem depended (Problem complexity defines NN's complexity). The number of weights determines the learning ability. So it is very important to choose correct number that can train network correctly and not entrapped into over fitness for limited data. So we choose a topology that balances generalization and specialization.

Choosing correct activation function is so important too [4].

We designed ANN with following features:

Supervised learning, 3 layers, sigmoid activation function for hidden layer and pure linear function for output layer, 4 neuron for hidden layer (therefore 8 weights must be adjusted), training constant $\eta = 0.8$, train data size =100.

Result:

To reach reliable identification we repeated experiment 10 times and then calculated average. Fig. 6 shows the best result.

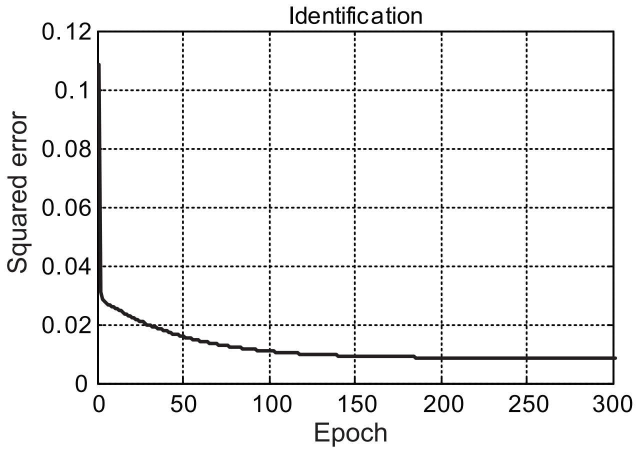

Fig. 6 – *Identification Error.*

Using low training constant make NN entraps into local minimum and with high training constant NN might pass global minimum (Fig. 23 shows this effect). The number of training data is effective on speed convergence and reliability (Fig. 24 shows this effect).

2.2 Evolutionary NN training

It has been shown that Multilayer Perceptron (MLP) networks can be used with great success to solve function approximation problems. The main difficulty in using this type of network is the training phase (which can be error prone and slow) due to its nonlinear nature. Many powerful optimization algorithms have been devised.

Evolutionary computation methodologies have been applied to three main attributes of neural networks: network connection weights, network architecture (network topology, transfer function), and network learning algorithms.

Most of the works involving the evolution of ANN has focused on the network weights and topological structure. Usually the weights and/or topological structure are encoded as a chromosome in GA or as particles in PSO. The fitness function is defined sum square error between system and model output [7].

The advantage of the EC is that it can be used in cases with non-differentiable PE transfer functions and no gradient information available. Another advantage is searching whole solution space parallel.

The disadvantages are the performance is not competitive in some problems and representation of the weights is difficult and the genetic operators have to be carefully selected or developed.

Genetic Algorithm approach

The GA, originally described by Holland is an *Evolutionary Algorithm* [8].

In this method we consider a population of individuals that each of them represents potential solution and a function that determines the quality of a solution, called the fitness function. Then use evolutionary operator (such as mutation, cross over, reinforcement…) to produce off springs. And according fitness new population is selected. Fig. 7 shows brief and simple GA algorithm [9, 10, 11].

```
$t \leftarrow 0$
$P(t) \leftarrow \text{initialise}(\mu)$
$F(t) \leftarrow \text{evaluate}(P(t), \mu)$
repeat:
    $P'(t) \leftarrow \text{recombine}(P(t), \theta_r)$
    $P''(t) \leftarrow \text{mutate}(P''(t), \theta_m)$
    $F(t) \leftarrow \text{evaluate}(P''(t), \lambda)$
    $P(t+1) \leftarrow \text{select}(P''(t), F(t), \mu, \theta_s)$
    $t \leftarrow t + 1$
until stopping criterion is met
```

Fig. 7 – *Pseudo* GA *code.*

Probability of mutation, crossover, kind of doing them, population size and selection kind is so effective on algorithm efficiency.

Usually mutation probability is assumed high at firs generation to search extended space and then decreasing it. But the exact value is problem depended. When algorithm entraps into local minimum, increasing mutation probability seems a good solution.

By increasing population size, we search more extended space but it's sometimes needs more time.

We can use coded or real value chromosomes that we preferred to use real value population.

There are many crossover and mutation methods. We used *Arithmetic* crossover and *nonuniform* mutation [9].

All NN's weights are considered as genes (we have 8 genes); fitness function is defined sum square error between model and system output (so we have minimizing problem). We assume $p_m = 0.1$, $p_c = 0.7$ and population size = 10.

In our case decreasing p_m didn't cause better result.

Result:

Our algorithm is robust to initial condition. With any initial condition, after transient we reach to same result. Fig. 8 shows average result.

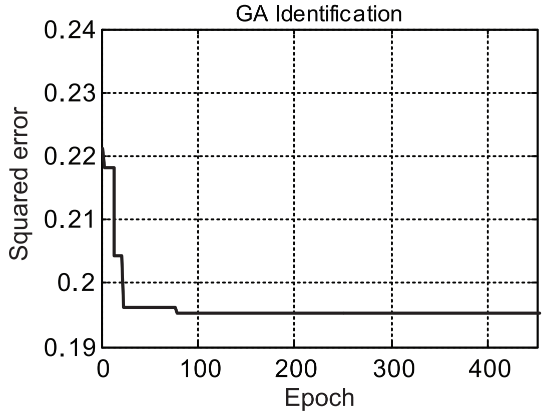

Fig. 8 – *Error identification.*

We changed and tested almost all variables but algorithm didn't reach lower error than 0.19 and couldn't come out of this local minimum. For better result we can use gradient method to send it out of local minimum. We tried this method, too; but about our system it didn't make better result than pure GA.

Particle Swarm Optimization Method

Particle swarm optimization was introduced in 1995 by Kennedy and Eberhart [12].

Some of the attractive features of the PSO include the ease of implementation and the fact that no gradient information is required. It can be used to solve a wide range of optimization problems, including most of the problems can be solved using Genetic Algorithms; some example applications include neural networks training [13]. It is like the other Evolutionary Algorithm, a *stochastic* algorithm and sociologically inspired method.

The following is a brief introduction to the operation of the particle swarm algorithm. Consider a flock or swarm of *p* particles, where each particle's position representing a possible solution point in the design problem space *D*. For each particle *i*, Kennedy and Eberhart proposed that the position x^i be updated in the following manner:

$$x_{k+1}^i = x_k^i + v_{k+1}^i \,. \tag{3}$$

The velocity vector keeps track of the speed and direction the particle is currently traveling.

With a pseudo-velocity v_{k+1}^i calculated as follows:

$$v_{k+1}^i = w_k v_k^i + c_1 r_1 (p_k^i - x_k^i) + c_2 r_2 (p_k^g - x_k^i) \,. \tag{4}$$

Here, subscript *k* indicates a (unit) pseudo-time increment, p_k^i represents the best ever position of particle *i* at time *k* (yielding the highest fitness) and p_k^g represents the global best position in the swarm at time *k* (social contribution). r_1 and r_2 represent uniform random numbers between 0 and 1, they are used to effect the stochastic nature of the algorithm. To allow the product $c_1 r_1$ or $c_2 r_2$ to have a mean of 1, that c_1 is cognitive and c_2 is social scaling parameters and can be selected such that $c_1 = c_2 = 2$ .They influence the maximum size of step that a particle can take in a single iteration. It's better to choose c_1, c_2 in the way that $c_1 + c_2 \le 4$ [7]. If $c_1 + c_2 \ge 4$, Velocities and positions tend to explode toward infinity. The result of using these proposed values is that the particles overshoot the target half the time, thereby maintaining separation within the group and allowing for a greater area to be searched. The addition of the variable w_k (Inertia weight), is a modification to the original PSO algorithm [7]. This allows a more refined search as the optimization progresses by reducing its value linearly or dynamically [7].

This algorithm includes several stages:

1. Initialization

(a) Set constants c_1, c_2, w_0;

(b) Randomly initialize particle positions from the uniform random distribution on the interval $[x_{\min}, x_{\max}]$;

(c) Randomly initialize particle velocities $[0, v_{\max}]$;

(d) Randomly initialize p^g and p^i .

2. Optimization

(a) Evaluate function value f_k^i using design space coordinates x_i^k;

(b) If $f_k^i \le f_{best}^i$ then $f_{best}^i = f_k^i$, $p^i = x_k^i$;

(c) If $f_k^i \le f_{gbest}^i$ then $f_{gbest}^i = f_k^i$, $p^g = x_k^i$;

(d) If stopping condition is satisfied then go to 3;

(e) Update particle velocity vector v_{k+1}^i using (4);

(f) Update particle position vector x_{k+1}^i using (3);

(g) Go to 2 (a).

3. Report result

4. Terminate

PSO is a promising method to train ANN. It is faster and gets better results in most cases. It also avoids some of the problems GA met.

This type of behavior seems to be ideal when exploring large error surfaces, especially with a relatively large maximum velocity parameter. Some particles can explore far beyond the current minimum (while the population still remembers the global best solution) This solves one of the problems of gradient based optimization techniques, namely their poor performance in regions with very flat gradients. Should the random initialization cause the starting position to be in such a region, the particle swarm optimizer can quickly move closer towards a minimum, where the gradient will typically be much steeper.

Result:

For a neural network implementation, the fitness function is sum square error like GA, and the position vector corresponds to the weight vector of the network. At the end of the algorithm, the global best particle's position serves as the answer.

For NN training, all weights are considered as particles, so we have 8 dimensions space. At first we assume population size=10, $c_1 = 1$, $c_2 = 2$ and $w_k = 1$ (one can find other details in main algorithm).

Fig. 9 shows the result.

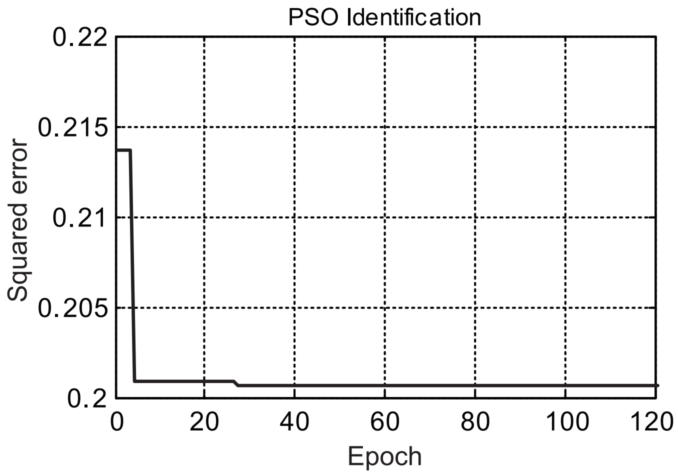

Fig. 9 – *Identification Error.*

3 Control

We usually have some specific goals of controlling a system:

(i) stabilizing system;

(ii) closed-loop system tracks set point with desired manner;

(iii) reducing the effect of noise and disturbance on system performance;

(iv) decreasing system sensitivity.

Table 2
Zeros boundaries.

	Real	Imaginary
First pair	[–0.5,4]	[0.5,5]
Second pair	[4,10]	[0.5,5]

So we consider these goals as first condition for our controller.

We design our controller with an evolutionary Algorithm PSO directly.

PSO Controller

We design our controller with root-locus method.

The root-locus approach to design controller powerful when then the specification are given in terms of time-domain quantities like, damping ration, maximum overshoot, rise time, or settling time.

From the performance specification, the desired locations for dominant closed-loop poles are determined, and then we try to reach these closed loop poles by designing suitable compensators.

We design controller with aim that the step response curve will exhibit maximum overshoot of 25% or less and settling time of 15s or less.

We make use of SISO Tools of Matlab to find right controller structure.

System root-locus clears that system is unstable even for a very low gain (one can find root-locus plot in Fig. 19).

Because of it, we must add some zeros (the number of added zeros must be defined) at proper place that pull the root locus to left and tend to make the system more stable and can speed up the settling of the response. To have proper controller we need poles that are insignificant in comparison with system's poles.

At last we design 4th order controller that stabilize system and satisfy performance constraints.

It seems that each pair zeros must be at specific boundary that our goals satisfy. Limitation is defined like table (2). We put all controller poles at -10 so they don't effect on desired performance.

At this stage we consider PSO for finding exact places of controller zeros.

We define search space as limited parameters space. Our PSO Controller has 5 dimension (controller zeros and gain) and population size=50 and *gbest* algorithm. And after that we change parameters to effect of them on system performance. The fitness function is defined based on max overshoot and settling time so it is trade-off between minimizing max overshoot and settling time. We define a factor to adjust the weight of each factor to reach better response. Cost function is defined like:

$$\eta(\text{settling time}) + (1-\eta)(\text{max} - \text{overshoot})\,. \qquad (5)$$

Result:

Fig. 10 shows the closed loop system step response ($0.4u(t)$, as much as possible near linear condition of system).

We consider average result that is more reliable, as shown in Fig. 11.

If we decrease population size, algorithm becomes slower and need more time to execute (one can see Fig. 25).

We use inertia weight and decrease linearly from 1.2 to 0.8. When it is more than 1, particles accelerate up to the maximum velocity and when it is less than 1 will cause the particle to slowly decelerate until its velocity reaches zero and algorithm may entrap in local minimum. But we don't get better result. If executing iteration isn't limited maybe it works (one can see related Figs. 25 and 26).

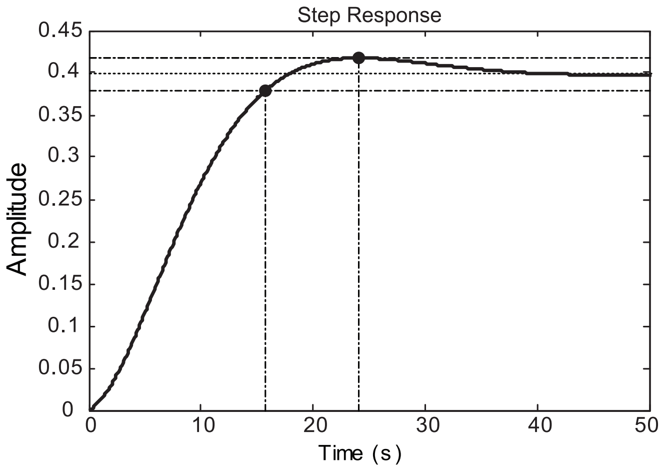

Fig. 10 – *Step response* (max overshoot = 4.56 %, settling time = 15.6 s).

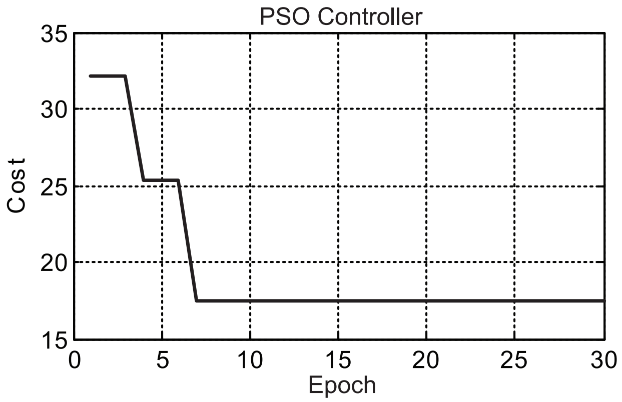

Fig. 11 – *Cost function*.

Then we used *breeding* method. In this method, the effect of reproduction and recombination on PSO is investigated [7] when our algorithm entraps into local minimum this method can send out of it (Fig. 27).

In other experiment we decreased parameter c_1 linearly from 1.6 to 0.8. The result is more robust to initial condition (Fig. 28). We changed c_2, too. But result didn't change noticeably.

4 Discussion and Conclusion

1. Identification: Neural network is a good structure for system identification.

It has been proven that this architecture can approximate any continuous function to any degree of accuracy of a compact set. When we can use gradient information and in small structure, back propagation seems optimum solution. But it is time consuming procedure.

Compared with genetic algorithms (GAs), PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. The information sharing mechanism in PSO is significantly different. In GA, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area.

In our case we reached better result in training mode with back propagation.

But in test mode PSO and GA lead to lower error. May be the NN that is trained with back-propagation is overfit for train data (over-fitting occurs when the error on the training set is driven to a very small value but when new data is presented to the network the error is large. The network has memorized the training examples but has learned not to generalize to new situations).

Fig. 12 shows comparison between these algorithms in test mode.

We examined our NN identification for nonlinear model the result (Fig. 13) is acceptable. So linear assumption around operating point didn't limit our identification.

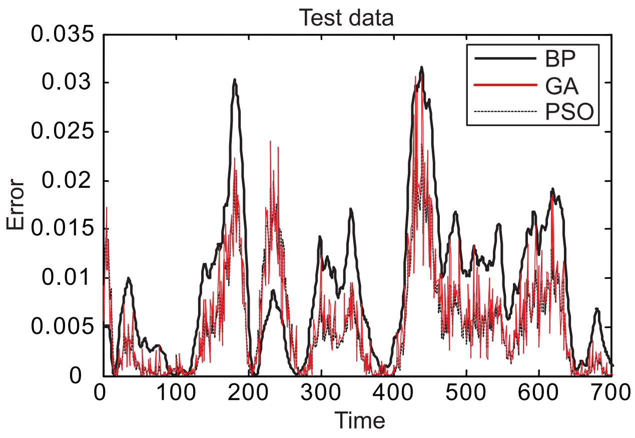

Fig. 12 – *Test mode.*

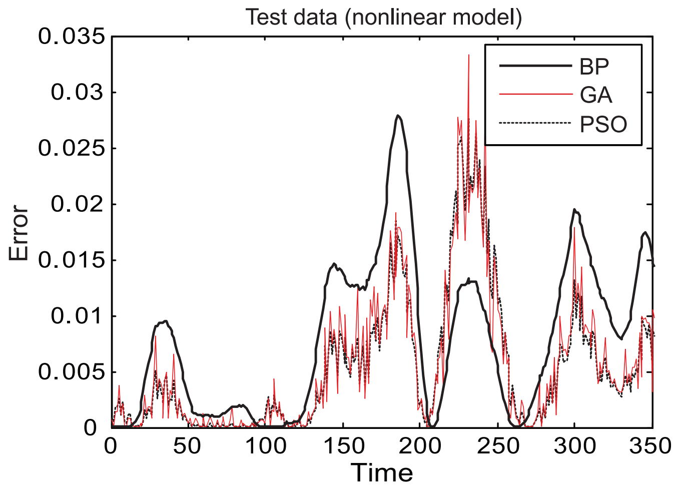

Fig. 13 – *Test mode for nonlinear system.*

2. Control: Control problem can be considered such as an optimization problem. By defining proper cost function, we can reach the desired response. Cost function can include many factors like desired transient qualities, steady state response, etc. Although we defined cost function of max overshoot and settling time and didn't consider control effort in it, signal control behavior is almost acceptable.

We tested different PSO modifications and you can result in Fig. 14. PSO with 10 population has better result than other methods. In breeding the result has steep slope. But all of them are successful in stabilizing and controlling system over almost with every initial condition.

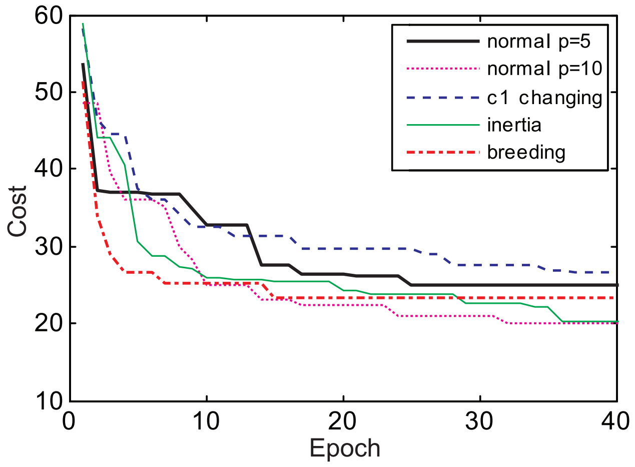

Fig. 14 – PSO *cost function.*

5 Appendix

5.1 Appendix 1

This appendix contains following parts:

(i) Nonlinear model;
(ii) Azimuth response;
(iii) Testing state feedback controller for nonlinear system;
(iv) System step response and root-locus and niquist plots.

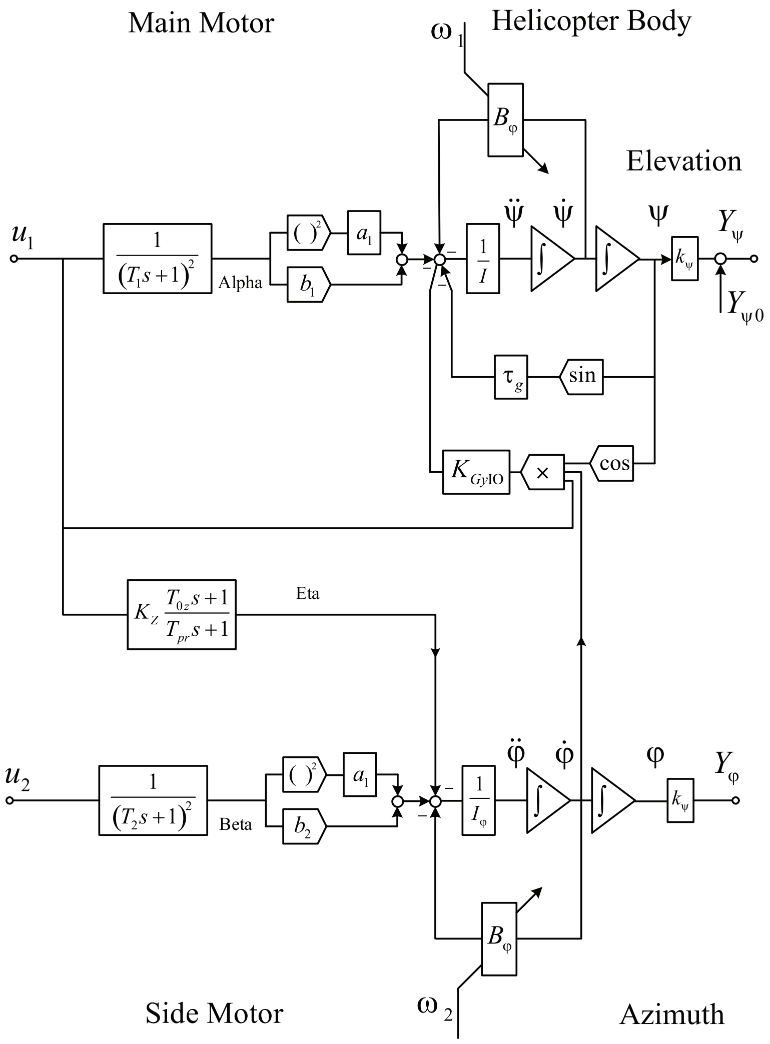

Fig. 15 – *Nonlinear system.*

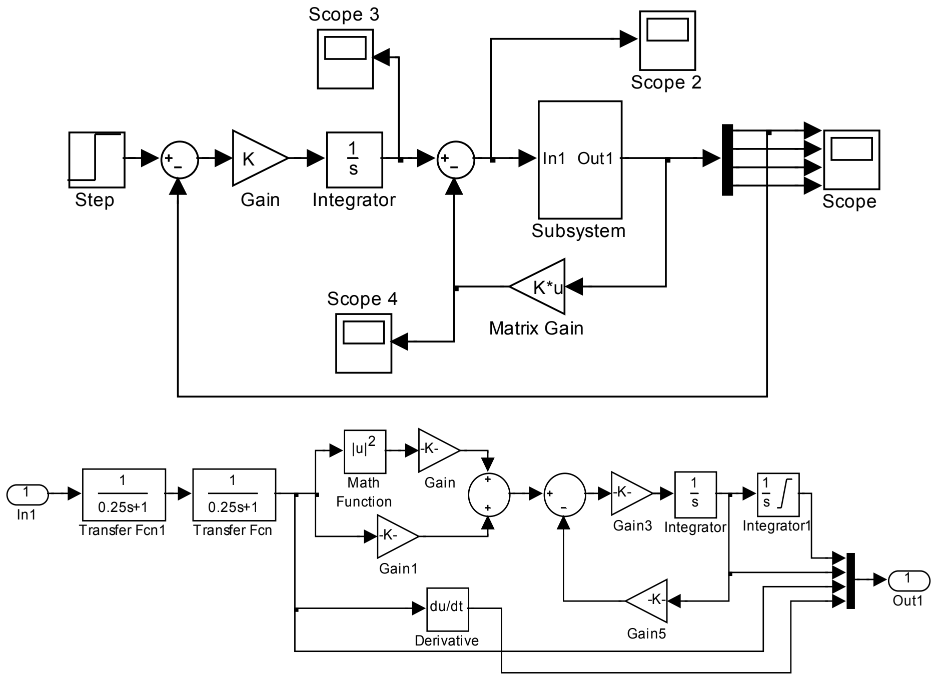

Fig. 16 – *Nonlinear azimuth system.*

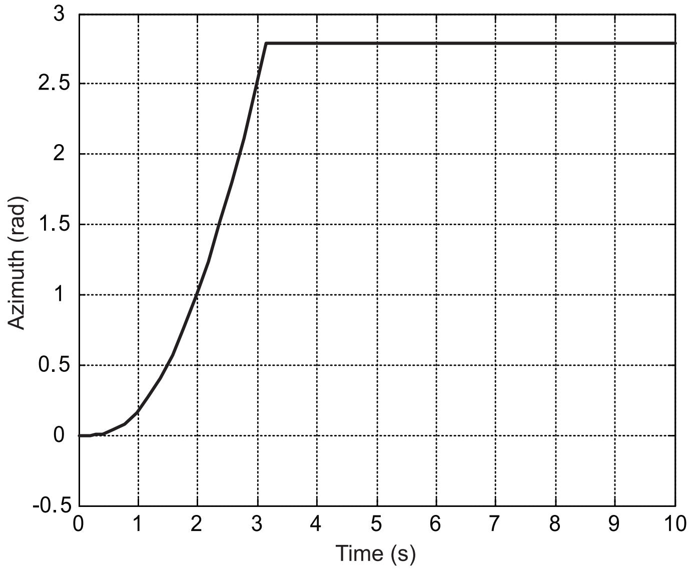

Fig. 17 – *Azimuth response without any control.*

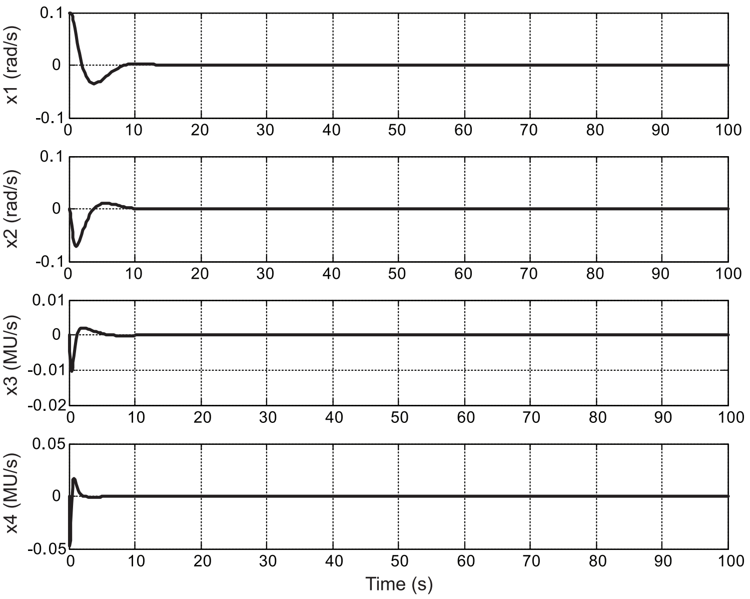

Fig. 18 – *Nonlinear system with state feedback controller.*

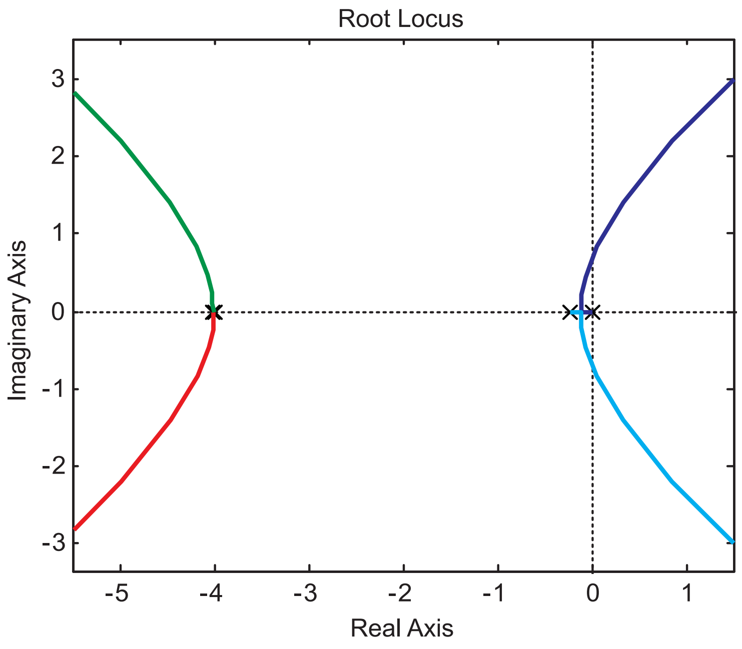

Fig. 19 – *System root-locus plot.*

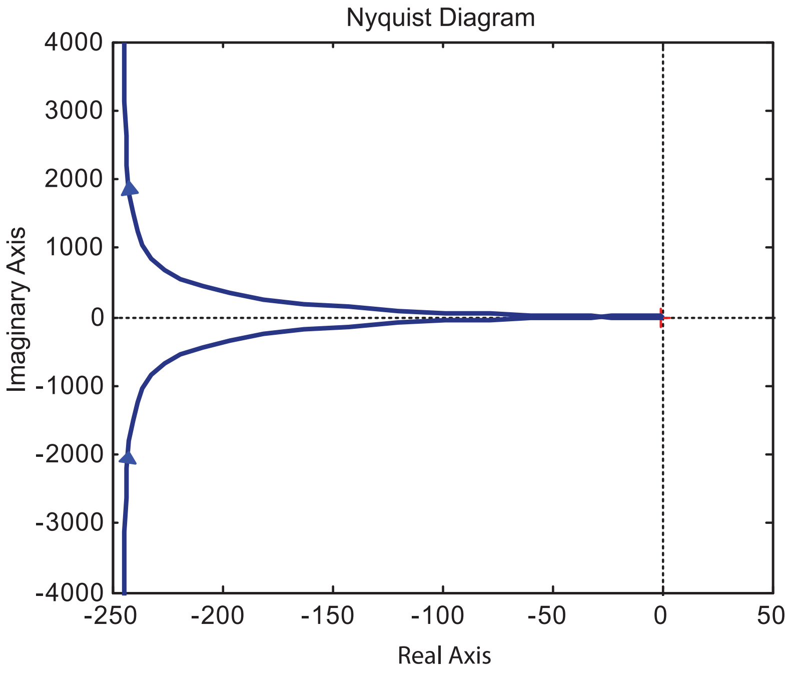

Fig. 20 – *System Niquist plot.*

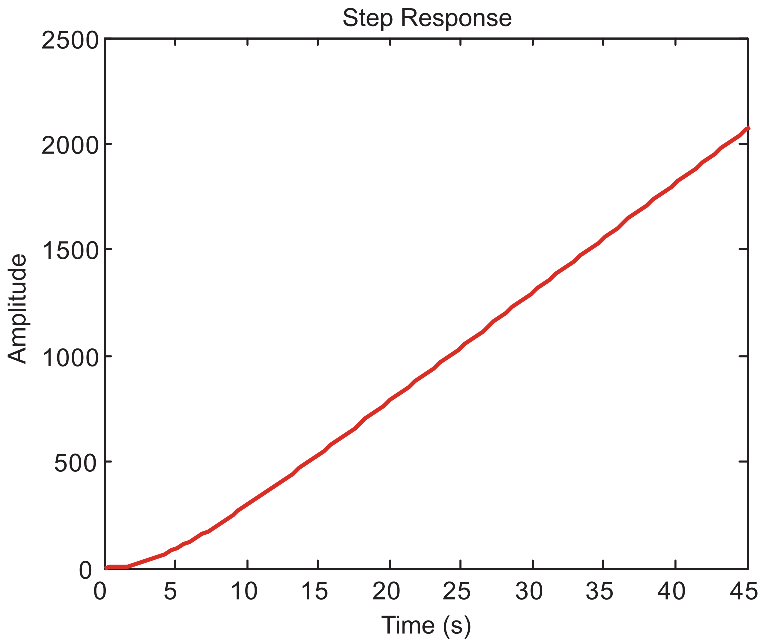

Fig. 21 – *System step response.*

5.2 Appendix 2

```
%*************** Artificial Nueral Network *******************
clear
clear state
clc
%*************************loading Data***************************
load input
load output
TM=u(1:NP);
d=y(1:NP);
%************ Determining Neural Network Characteristics ********
% Number of Layers = 3 : input layer, one hidden layer, output layer
% NNH = Number of Neurons in the Hidden Layer
% NI = Number of Inputs
% NO = Number of Outputs,equals number of inputs.
% LC = Leanrning Coeffincient
% NP = Number of Patterns
NNH=4;
NI=1;
NO=1;
eta=0.8;
NP=100;
Emax=.01;
E=0.15;
k=1;
epoch=0;
SS=0;
tic;
%*************************Initializing Weights******************
V=rand(NNH,NI);
W=rand(NO,NNH);
y=zeros(NNH,1);
deltao=zeros(NO,1);
deltay=zeros(NNH,1);
%************************* Training Phase **********************
while epoch<1000
    epoch=epoch+1;
    E=0;
    for p=1:NP
        nety=V*TM(p);
        y=logsig(nety);
        o=W*y;
        deltao=(d(p)-o);
        for j=1:NNH
            deltay(j)=.5*(1-y(j).^2)*((deltao)*W(j));
        end
```

Fig. 22 – *Neural Network Algorithm.*

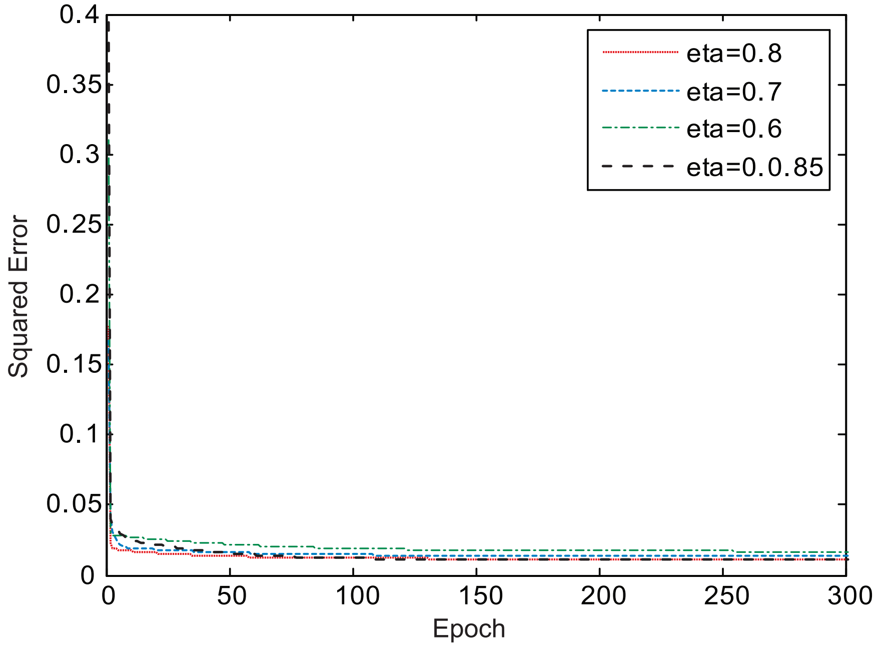

Fig. 23 – *Effect of training constant.*

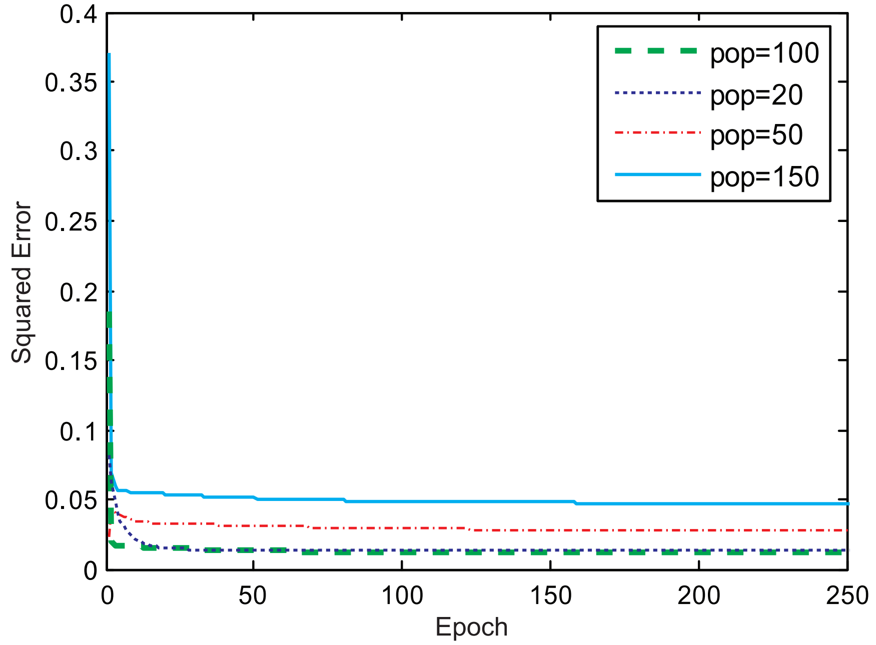

Fig. 24 – *Effect of population size.*

PSO Controller result

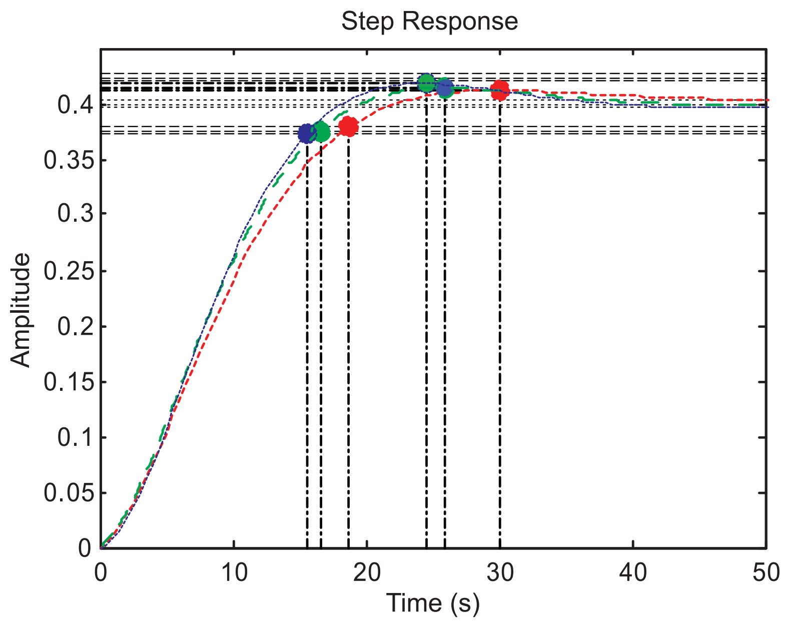

Fig. 25 – `pop_size = 10`.

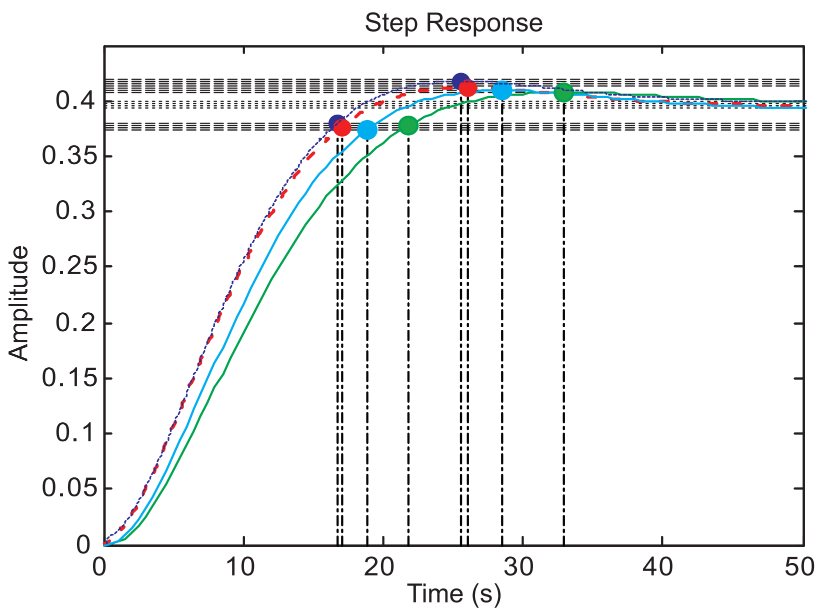

Fig. 26 – `pop_size=5`.

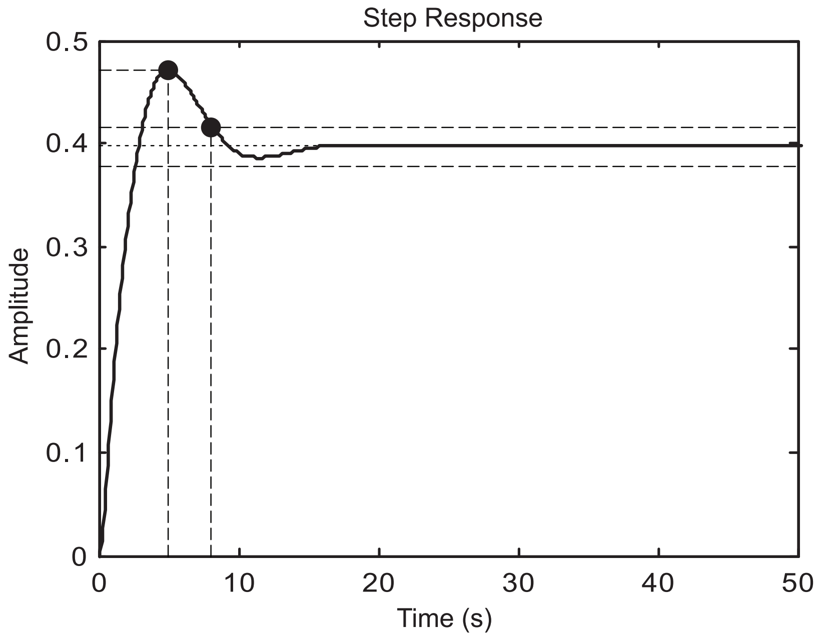

Fig. 27 – *Breeding method.*

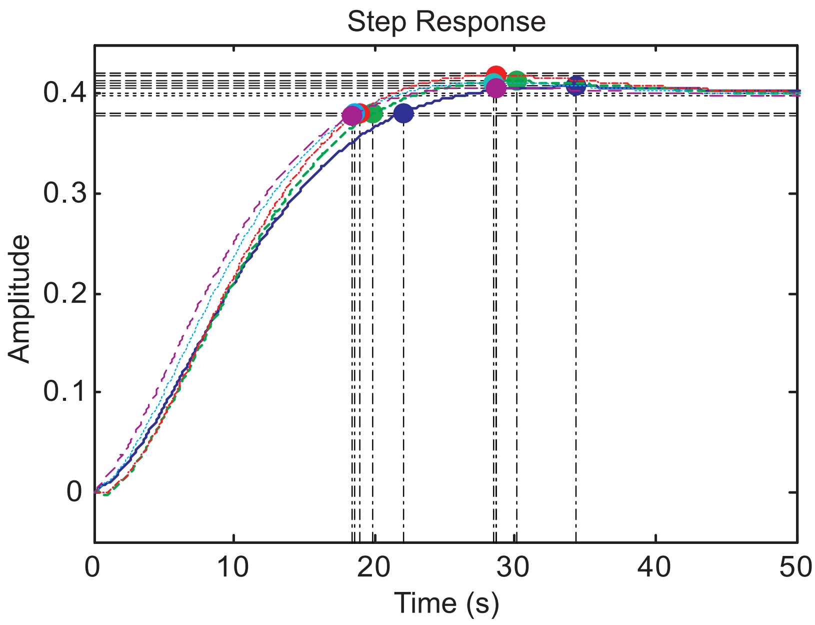

Fig. 28 – c_1 *changing.*

6 References

[1] J. Wang, B.T. Brackett, R.G. Harley: Particle Swarm-assisted State Feedback Control: From Pole Selection to State Estimation**,** American Control Conference, St. Louis, MO, USA, June 2009, pp. 1493 – 1498.

[2] P. Nourian: Designing and Implementation SISO Controllers of CE150 Helicopter Model, Khaje Nasir Univ, 2004.

[3] CE150 Helicopter Model User's Manual, Humusoft.

[4] J.M. Zurada: Introduction to Artificial Neural Systems, PWS Publishing Company, 1992.

[5] H.D. Taghirad: An Introduction to modern control, Khaje Nasir Toosi University of Technology, Tehran, Iran, 2004.

[6] C.M. Bishop: Neural Networks for Pattern Recognition, Oxford University Press, New York, 1995.

[7] F.V.D. Bergh: An Analysis of Particle Swarm Optimizers, PhD Thesis, University of Pretoria, 2001.

[8] J.H. Holland: Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, USA, 1992.

[9] M. Gen, R. Cheng: Genetic Algorithms and Engineering Design*,* John Wiley & Sons, 1997.

[10] D. Whitley: Genetic Algorithms and Neural Networks*,* John Wiley & Sons, 1995.

[11] D. Whitley: A Genetic Algorithm Tutorial, Statistics and Computing, Vol. 4, No. 2, June 1994, pp. 65 – 85.

[12] J. Kennedy: The Behavior of Particles, V.W. Porto, N Saravanan , D. Waagen (eds), Proc. Of the 7th Int.Conf. On Evolutionary Programming, 1998, pp 581 – 589.

[13] K.E. Parsopoulos, M.N. Vrahatis: Recent Approaches to Global Optimization Problems through Particle Swarm Optimization, Natural Computing, Vol. 1, No. 2-3, June 2002, pp. 235 – 306.