A Learning Algorithm for Neural Networks and General Dynamic Systems

Feng Lin
Department of ECE
Wayne State University

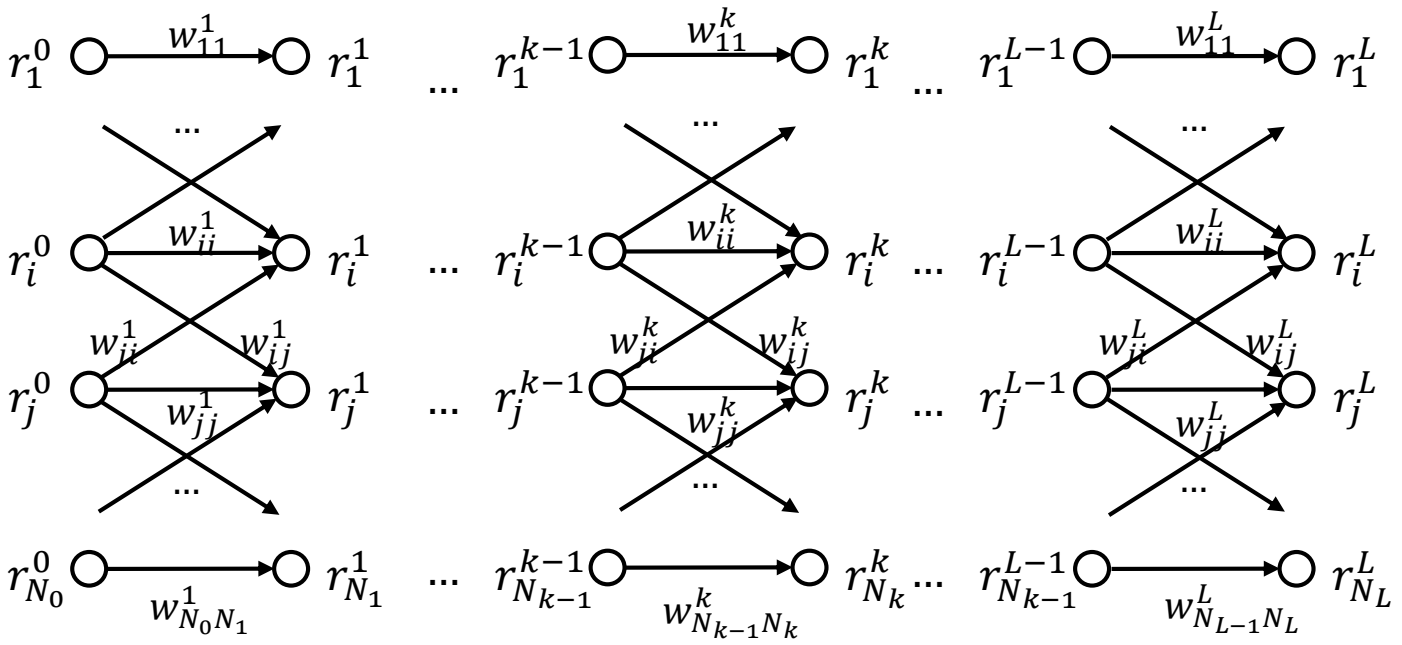
ICINCO 2021
July 6, 2021

Outline

- Introduction
- Algorithm and Implementation without Feedback Network
- Simulation Results
- Generalized Signal-Flow Graphs (GSFG)
- On-line Learning of GSFG
- Adaptive PID Control
- Simulation Results
- Conclusion

I. Introduction

A neural network consists of L layers.



Layer k consists of N_k neurons, $k = 1, 2, ..., L$.

r_i^k - the firing rates of the neurons in Layer k.

r_i^0 - the inputs to the neural network.

r_i^L - the output from the neural network

L is the last or output layer.

w_{ij}^k - the weight between the i-th neuron in Layer $k-1$ and the j-th neuron in Layer k.

The outputs of of neurons in Layer k can be calculated as

$$
\begin{aligned}
p_j^k &= \sum_{m=1}^{N_{k-1}} w_{mj}^k r_m^{k-1} \\
r_j^k &= \sigma(p_j^k),
\end{aligned}
\tag{1}
$$

where p_j^k are the membrane potentials of the neurons and $\sigma(x) = 1/(1+e^{-x})$ is the sigmoidal function.

The goal is to learn/adapt the weights w_{ij}^k so that the least square error is minimized.

$$E = \frac{1}{2} \sum_{n=1}^{N_L} (r_n^L - \bar{r}_n^L)^2, \tag{2}$$

where $\bar{r}_n^L$ is the desired/target output of the n-th neuron in the last (output) layer.

The adaptation is achieved by letting

$$\dot{w}_{ij}^{k} = -\alpha \frac{dE}{dw_{ij}^{k}},$$

where α is the adaptation parameter that can be adjusted during the adaptation.

The conventional back-propagation algorithm (C-B-algorithm) can be derived as

$$\dot{w}_{ij}^{k} = -\alpha \frac{dE}{dw_{ij}^{k}} = -\alpha \phi_{j}^{k} r_{i}^{k-1}, \tag{3}$$

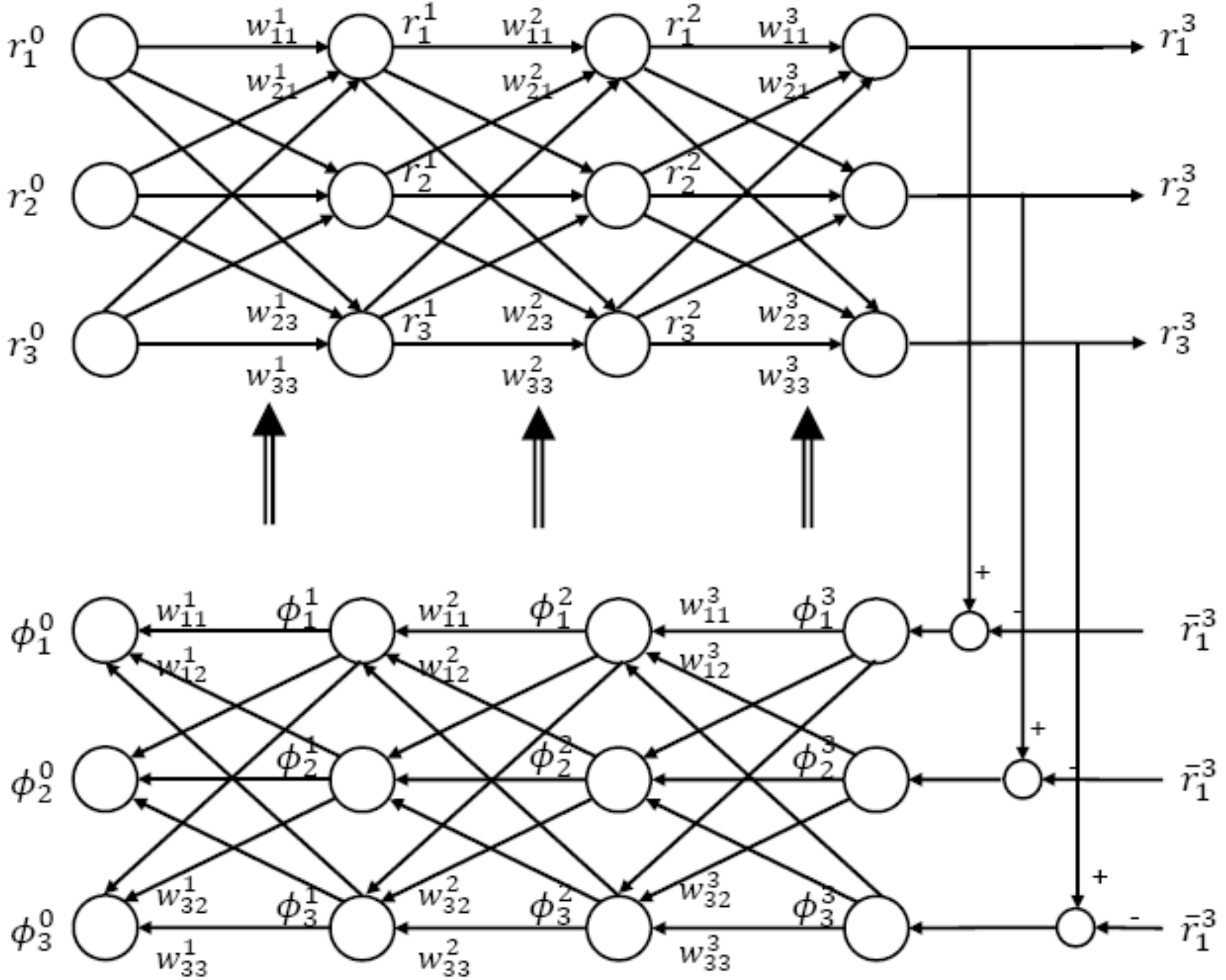
where

$$\phi_{j}^{L} = r_{j}^{L}(1 - r_{j}^{L})(r_{j}^{L} - \bar{r}_{j}^{L}), \ L \text{ is the last layer}$$

$$\phi_{j}^{k} = r_{j}^{k}(1 - r_{j}^{k}) \sum_{n=1}^{N_{k+1}} \phi_{n}^{k+1} w_{jn}^{k+1}, k = L-1, ..., 1. \tag{4}$$

$$\phi_j^k = r_j^k(1 - r_j^k) \sum_{n=1}^{N_{k+1}} \phi_n^{k+1} w_{jn}^{k+1}, k = L - 1, ..., 1. \tag{4}$$

Note that a feedback network is needed to back-propagate the error signal from ϕ_n^{k+1} to ϕ_j^k. The weights of the feedback network must be kept the same as the feed-forward network, because w_{ij}^k appears in both Equation (1) and Equation (4).


r_1^0
r_2^0
r_3^0
w_{11}^1
w_{21}^1
w_{23}^1
w_{33}^1
r_1^1
r_2^1
r_3^1
w_{11}^2
w_{21}^2
w_{23}^2
w_{33}^2
r_1^2
r_2^2
r_3^2
w_{11}^3
w_{21}^3
w_{23}^3
w_{33}^3
r_1^3
r_2^3
r_3^3
ϕ_1^0
ϕ_2^0
ϕ_3^0
w_{12}^1
w_{32}^1
ϕ_1^1
ϕ_2^1
ϕ_3^1
w_{12}^2
w_{32}^2
ϕ_1^2
ϕ_2^2
ϕ_3^2
w_{12}^3
w_{32}^3
ϕ_1^3
ϕ_2^3
ϕ_3^3
+
-
$\bar{r}_1^3$
$\bar{r}_1^3$
$\bar{r}_1^3$

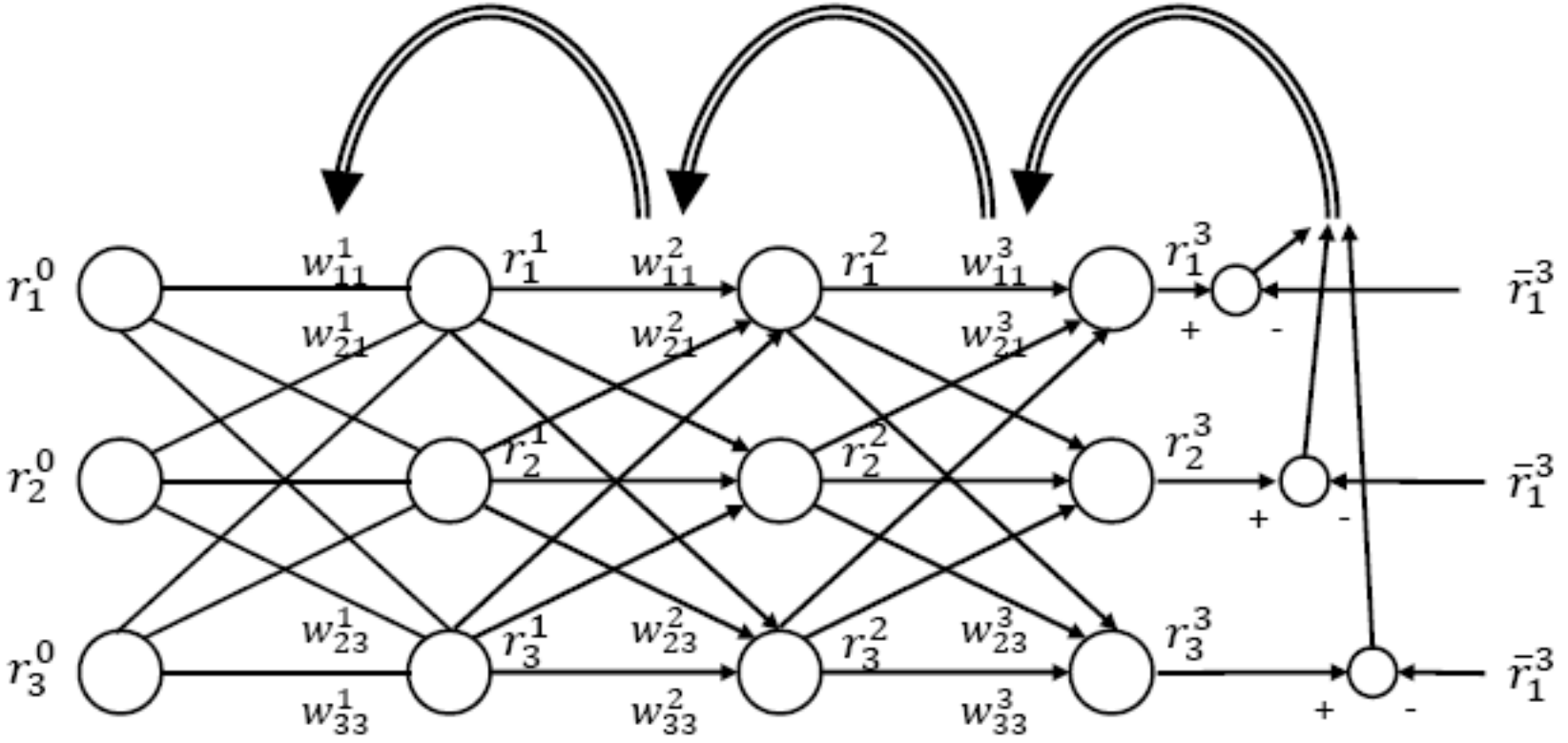
II. Algorithm and Implementation without Feedback Network

An equivalent learning algorithm (B-L-algorithm), first published in 1996 by Brandt and Lin.

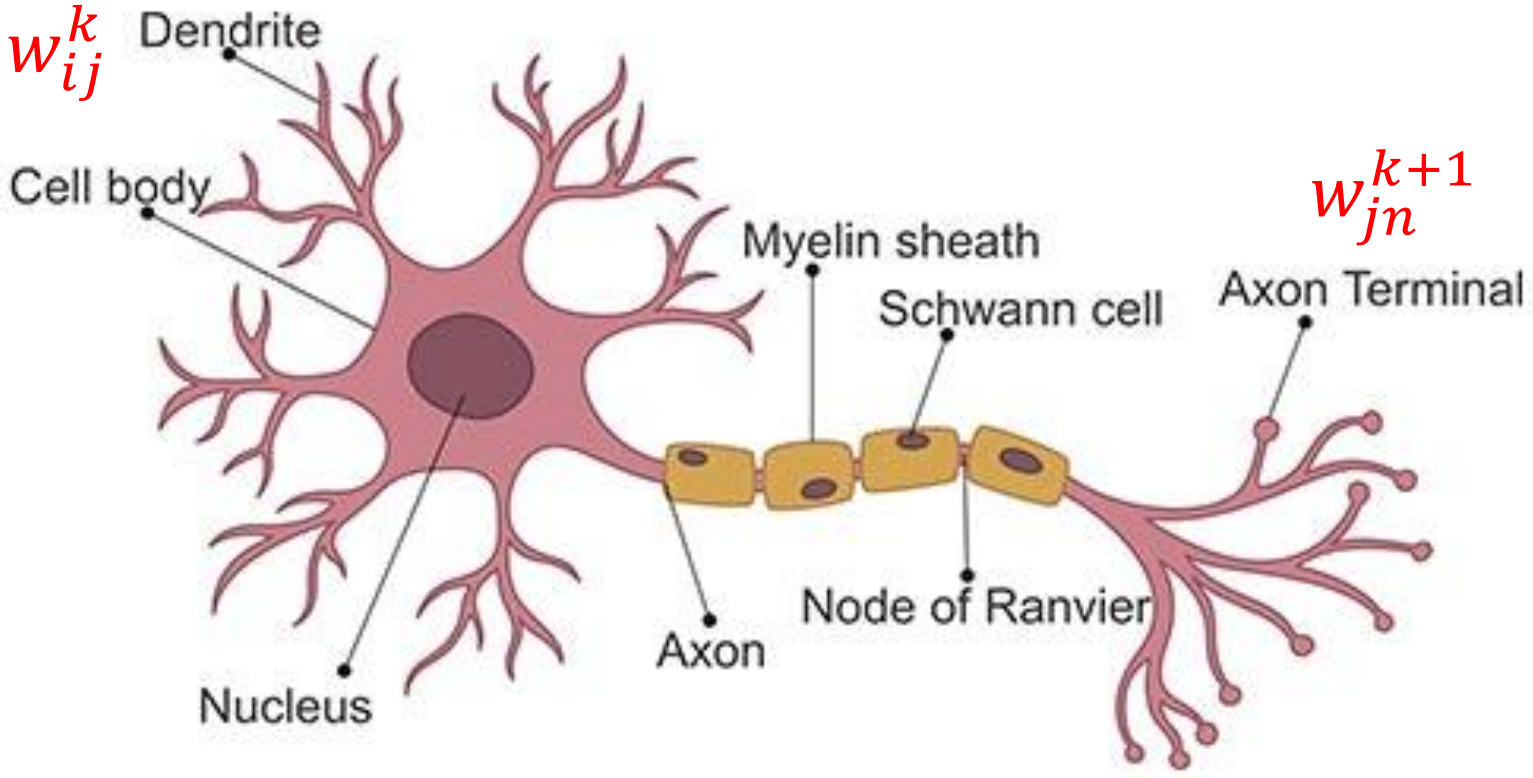
$$\dot{w}_{ij}^{L} = -\alpha r_i^{L-1} r_j^{L} (1 - r_j^{L})(r_j^{L} - \bar{r}_j^{L}),$$

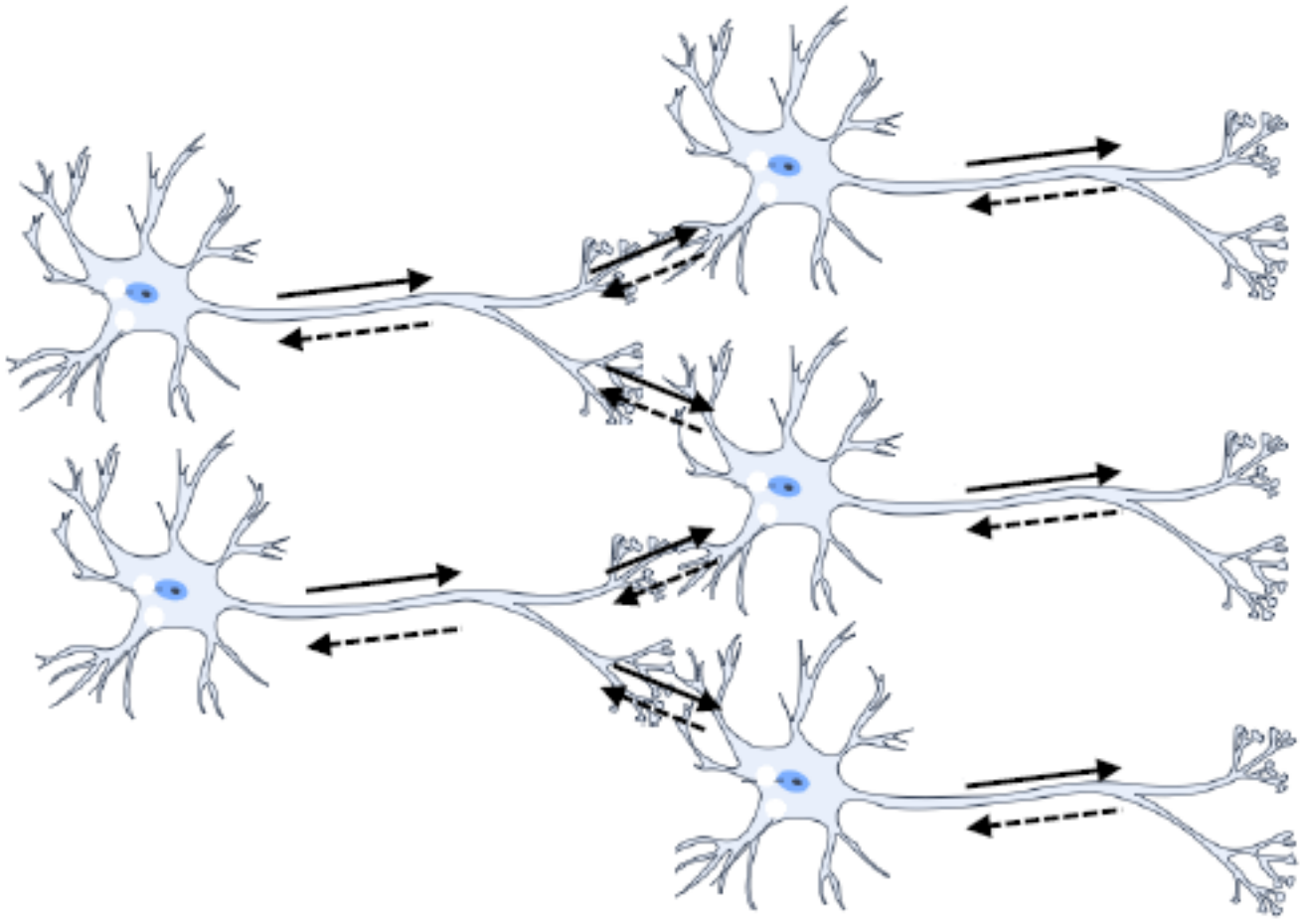
$$\dot{w}_{ij}^{k} = r_i^{k-1}(1 - r_j^{k}) \sum_{n=1}^{N_{k+1}} \dot{w}_{jn}^{k+1} w_{jn}^{k+1}. \qquad (5)$$

$$k = L - 1, ..., 1.$$


r_1^0
r_2^0
r_3^0
w_{11}^1
w_{21}^1
w_{23}^1
w_{33}^1
r_1^1
r_2^1
r_3^1
w_{11}^2
w_{21}^2
w_{23}^2
w_{33}^2
r_1^2
r_2^2
r_3^2
w_{11}^3
w_{21}^3
w_{23}^3
w_{33}^3
r_1^3
r_2^3
r_3^3
+
-
+
-
+
-
$\bar{r}_1^3$
$\bar{r}_1^3$
$\bar{r}_1^3$

In a neuron, the weights w_{ij}^{k} of dendritic synapses are adapted based in the information on the weights w_{jn}^{k+1} of its axonic synapses




forward information flow
backward information flow

References

- R. D. Brandt and F. Lin, 1996, “Supervised learning in neural networks without feedback network,” Proceeding of the 1996 IEEE International Symposium on Intelligent Control, pp. 86-90.
- F. Lin, “Supervised learning in neural networks: Feedback-network-free implementation and biological plausibility,” IEEE Transactions on Neural Networks and Learning Systems, 2021.

Supervised Learning in Neural Networks: Feedback-Network-Free Implementation and Biological Plausibility

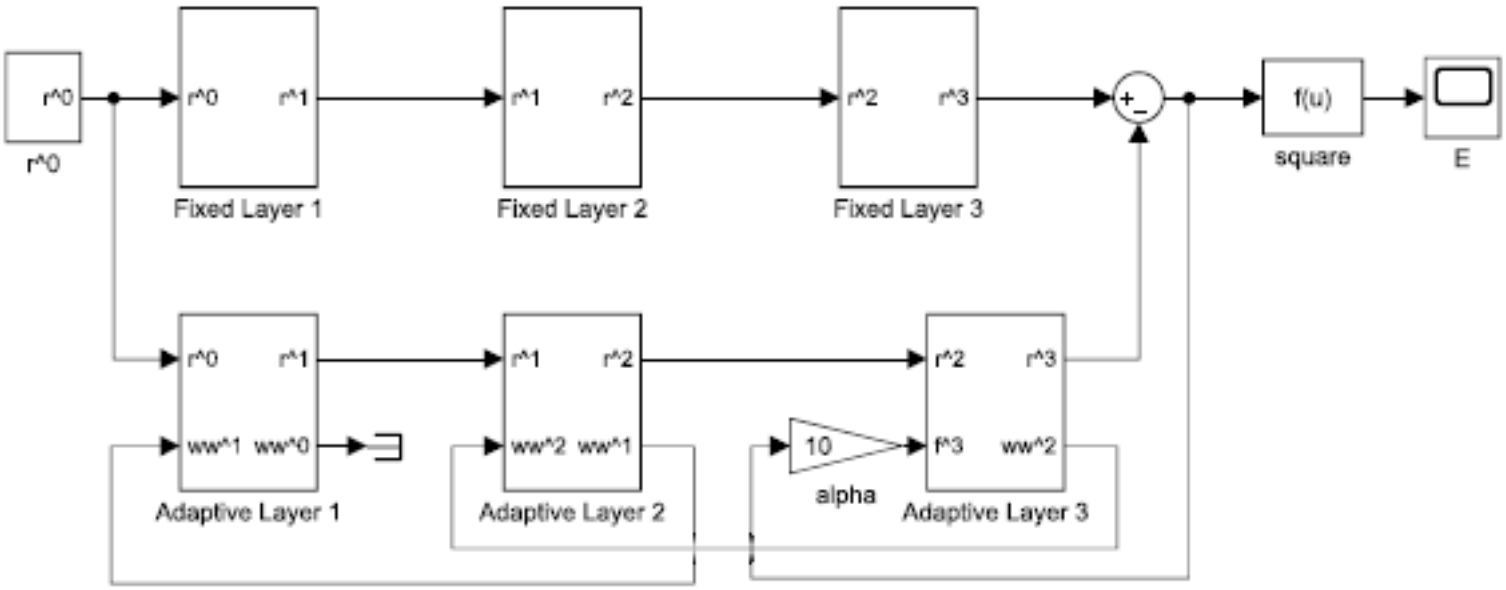
Feng Lin, *Fellow, IEEE*

Abstract— **The well-known backpropagation learning algorithm is probably the most popular learning algorithm in artificial neural networks. It has been widely used in various applications of deep learning. The backpropagation algorithm requires a separate feedback network to back propagate errors. This feedback network must have the same topology and connection strengths (weights) as the feed-forward network. In this article, we propose a new learning algorithm that is mathematically equivalent to the backpropagation algorithm but does not require a feedback network. The elimination of the feedback network makes the implementation of the new algorithm much simpler. The elimination of the feedback network also significantly increases biological plausibility for biological neural networks to learn using the new algorithm by means of some retrograde regulatory mechanisms that may exist in neurons. This new algorithm also eliminates the need for two-phase adaptation (feed-forward phase and feedback phase). Hence, neurons can adapt asynchronously and concurrently in a way analogous to that of biological neurons.**

well-known and widely used [17]–[19]. It allows errors to be backpropagated via a feedback network so that the strengths/weights of each synapse/connection can be learned/adapted to reduce the errors.

Researchers have investigated whether an analogous adaptation mechanism might occur in biological neural systems [20] since the introduction of the backpropagation algorithm. The consensus among neuroscientists is that the backpropagation is not likely to occur in biological neural systems [21]. The main reason for this consensus is that the backpropagation algorithm requires a dedicated feedback network to backpropagate errors (see, for example, [17]). Such a separate feedback network does not exist as a biological neural system. It is unreasonable to require that a biological neural system has a one-to-one correspondence between synapses in the feed-forward network and feedback network. It is even more unreasonable to require

Simulink implementation: The neural network with fixed layers are used to generate the desired/target outputs. The neural network with adaptive layers uses the B-L-algorithm.



We run several simulations to test the effectiveness of the B-L-algorithm. A typical simulation is as follows. Let

$$L = 3,$$

$$N_1 = N_2 = N_3 = 3.$$

We select $\bar{w}_{ij}^k$ with $k = 1, 2, 3$ and $i, j = 1, 2, 3$ randomly as

$$\begin{bmatrix} \bar{w}_{11}^1 & \bar{w}_{12}^1 & \bar{w}_{13}^1 \\ \bar{w}_{21}^1 & \bar{w}_{22}^1 & \bar{w}_{23}^1 \\ \bar{w}_{31}^1 & \bar{w}_{32}^1 & \bar{w}_{33}^1 \end{bmatrix} = \begin{bmatrix} -1.9365 & 3.1763 & -1.2139 \\ 0.0851 & 2.9483 & 3.1158 \\ 0.1077 & 1.4432 & 0.3283 \end{bmatrix}$$

$$\begin{bmatrix} \bar{w}_{11}^2 & \bar{w}_{12}^2 & \bar{w}_{13}^2 \\ \bar{w}_{21}^2 & \bar{w}_{22}^2 & \bar{w}_{23}^2 \\ \bar{w}_{31}^2 & \bar{w}_{32}^2 & \bar{w}_{33}^2 \end{bmatrix} = \begin{bmatrix} -1.4927 & 0.5016 & -2.9226 \\ 4.3900 & 1.2248 & -1.9875 \\ 3.7594 & 0.8704 & -0.2908 \end{bmatrix}$$

$$\begin{bmatrix} \bar{w}_{11}^3 & \bar{w}_{12}^3 & \bar{w}_{13}^3 \\ \bar{w}_{21}^3 & \bar{w}_{22}^3 & \bar{w}_{23}^3 \\ \bar{w}_{31}^3 & \bar{w}_{32}^3 & \bar{w}_{33}^3 \end{bmatrix} = \begin{bmatrix} 1.2206 & -0.9819 & -3.7668 \\ -1.4905 & -4.2403 & -3.1609 \\ 0.1325 & -2.6008 & -2.6005 \end{bmatrix}$$

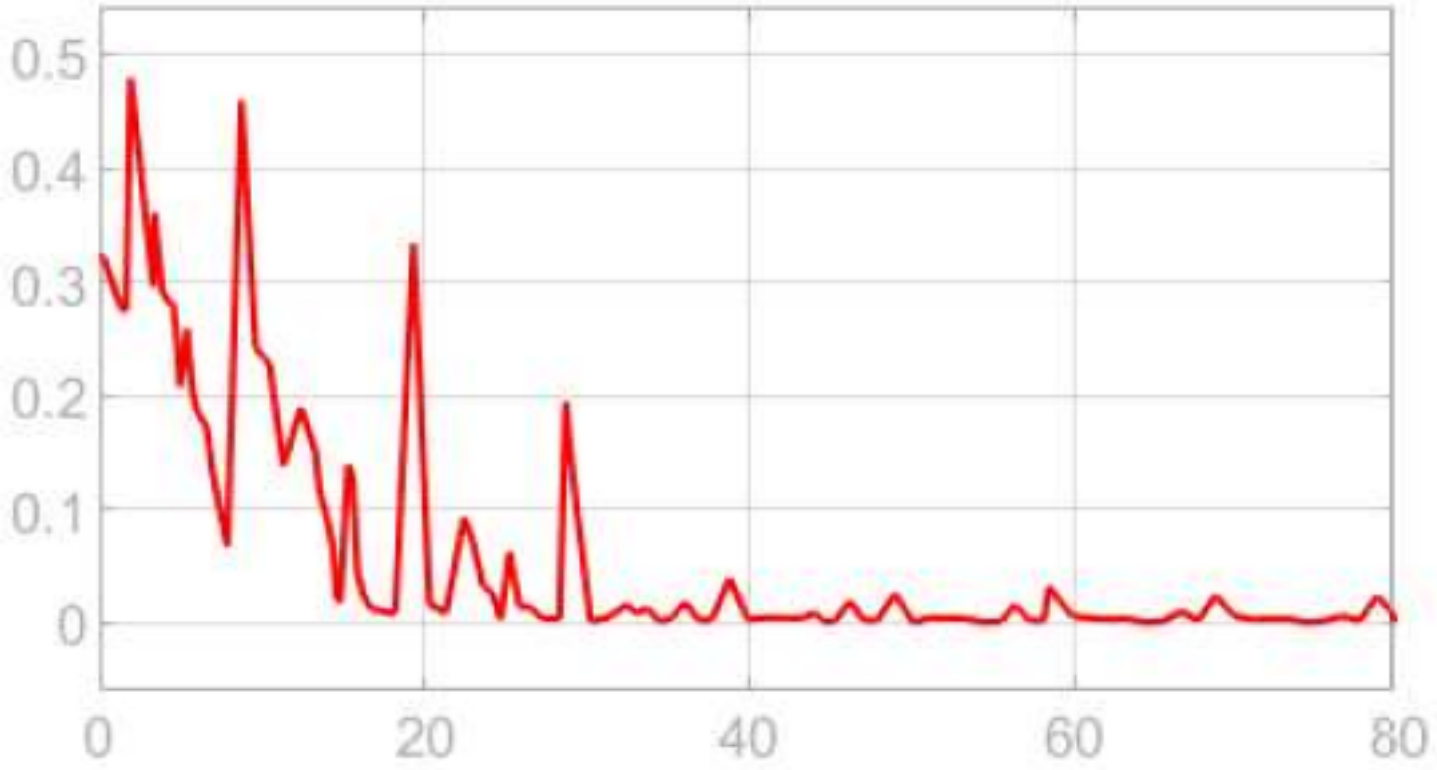
The initial weights (before learning) for w_{ij}^{k} with $k = 1, 2, 3$ and $i, j = 1, 2, 3$ are also selected randomly as

$$\begin{bmatrix} w_{11}^{1} & w_{12}^{1} & w_{13}^{1} \\ w_{21}^{1} & w_{22}^{1} & w_{23}^{1} \\ w_{31}^{1} & w_{32}^{1} & w_{33}^{1} \end{bmatrix} = \begin{bmatrix} 2.3172 & 1.4775 & -0.4908 \\ 0.4701 & -2.0368 & 2.4469 \\ -3.1104 & 1.8678 & -3.1649 \end{bmatrix}$$

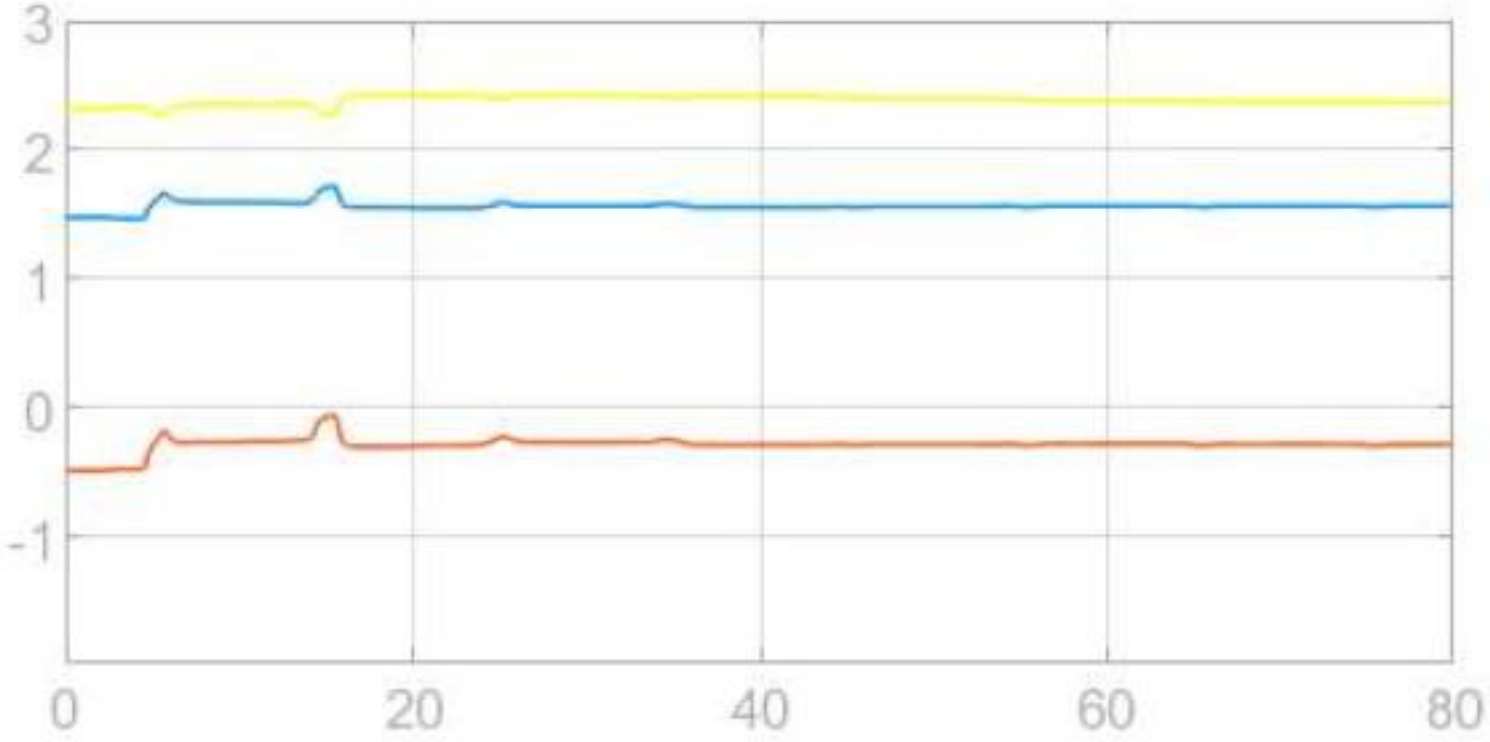
$$\begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{bmatrix} = \begin{bmatrix} 0.7521 & -4.4022 & -2.6522 \\ -1.4684 & 3.2119 & -4.8460 \\ -4.5698 & -3.3101 & 1.4912 \end{bmatrix}$$

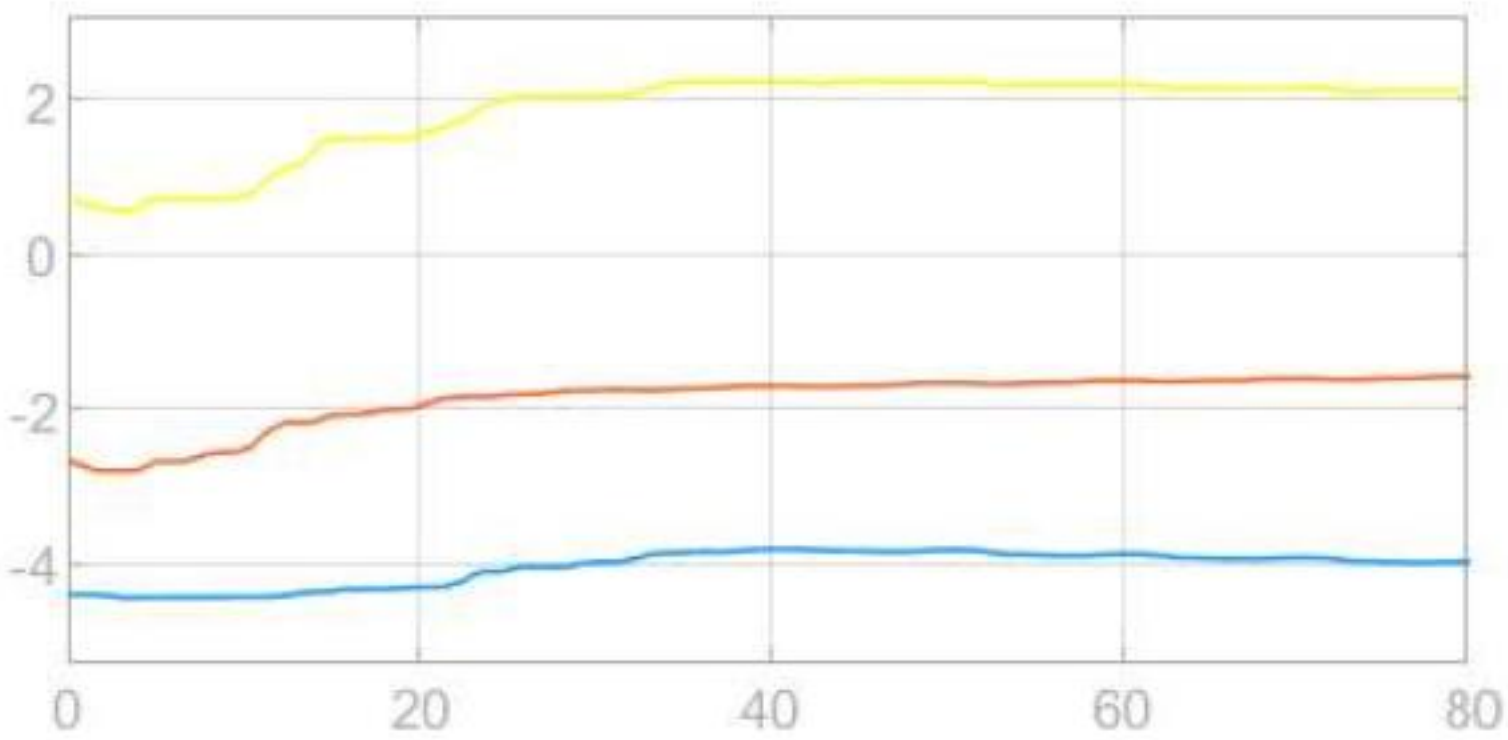
$$\begin{bmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 \\ w_{21}^3 & w_{22}^3 & w_{23}^3 \\ w_{31}^3 & w_{32}^3 & w_{33}^3 \end{bmatrix} = \begin{bmatrix} -0.4115 & 4.6309 & 0.4681 \\ 0.2114 & -2.6841 & -0.1110 \\ 1.2406 & 1.7914 & -1.0448 \end{bmatrix}$$

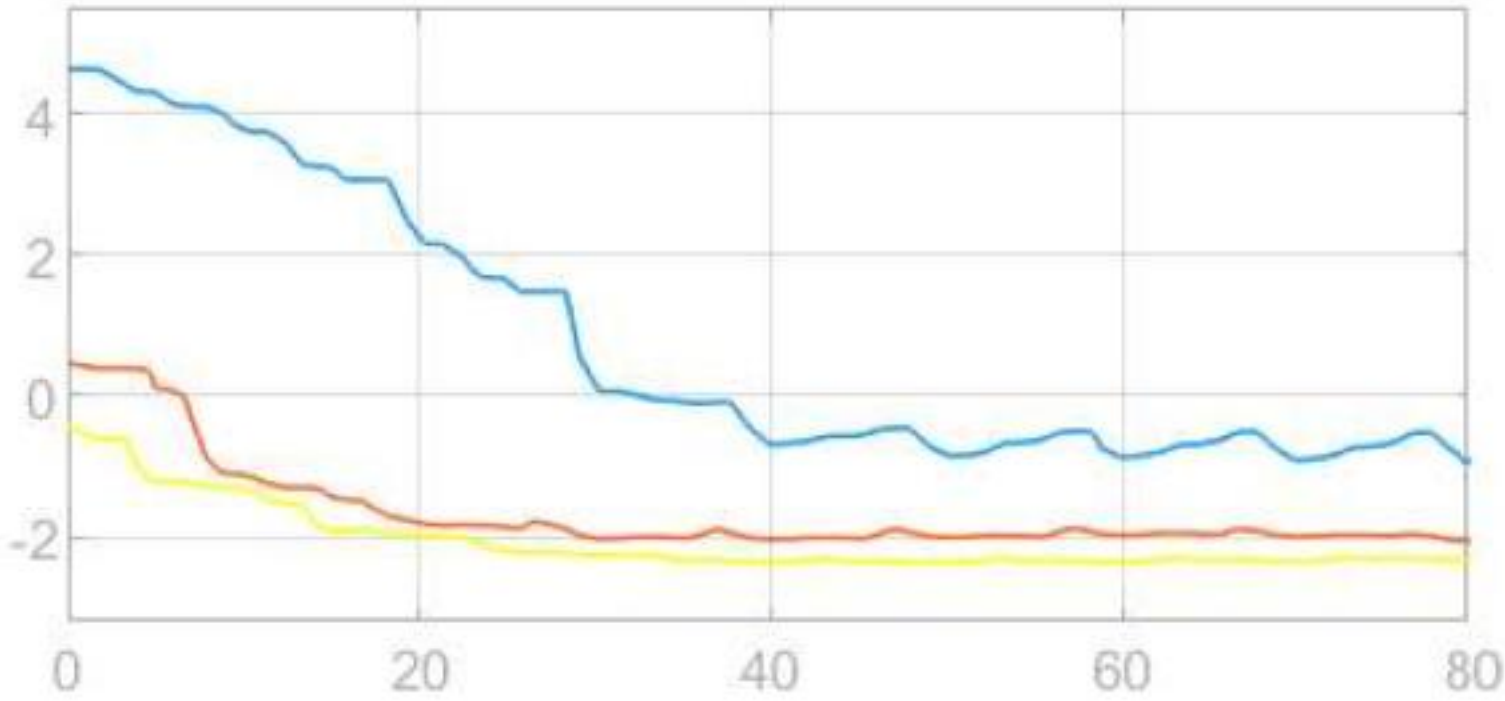
Simulation shows that the error reduces to 0.



Simulation shows the convergence of weights.




2
0
-2
-4
0
20
40
60
80


4
2
0
-2
0
20
40
60
80

Select a different $\bar{w}_{ij}^{k}$, the results are similar. .

$$\begin{bmatrix} \bar{w}_{11}^{1} & \bar{w}_{12}^{1} & \bar{w}_{13}^{1} \\ \bar{w}_{21}^{1} & \bar{w}_{22}^{1} & \bar{w}_{23}^{1} \\ \bar{w}_{31}^{1} & \bar{w}_{32}^{1} & \bar{w}_{33}^{1} \end{bmatrix} = \begin{bmatrix} -3.8880 & -2.5831 & -3.6803 \\ 2.8025 & -0.9609 & 4.4205 \\ -1.1026 & -4.0355 & 4.5613 \end{bmatrix}$$

$$\begin{bmatrix} \bar{w}_{11}^{2} & \bar{w}_{12}^{2} & \bar{w}_{13}^{2} \\ \bar{w}_{21}^{2} & \bar{w}_{22}^{2} & \bar{w}_{23}^{2} \\ \bar{w}_{31}^{2} & \bar{w}_{32}^{2} & \bar{w}_{33}^{2} \end{bmatrix} = \begin{bmatrix} -0.8273 & 4.4479 & -1.6228 \\ -4.5035 & -0.0914 & 4.0005 \\ 4.0272 & -0.1075 & -1.3075 \end{bmatrix}$$

$$\begin{bmatrix} \bar{w}_{11}^{3} & \bar{w}_{12}^{3} & \bar{w}_{13}^{3} \\ \bar{w}_{21}^{3} & \bar{w}_{22}^{3} & \bar{w}_{23}^{3} \\ \bar{w}_{31}^{3} & \bar{w}_{32}^{3} & \bar{w}_{33}^{3} \end{bmatrix} = \begin{bmatrix} 1.2206 & -0.9819 & -3.7668 \\ -1.4905 & -4.2403 & -3.1609 \\ 0.1325 & -2.6008 & -2.6005 \end{bmatrix}$$

The B-L-algorithm has the following properties.

1) Two algorithms are mathematically equivalent. In other words, the B-L-algorithm can be used wherever the C-B-algorithm can be used. In any applications, the performance of the B-L-algorithm will be as good/bad as the C-B-algorithm.

2) The implementation of the B-L-algorithm does not require a feedback network whose topology and weights must be kept the same as the feed-forward network.

3) Because of the removal of this requirement, adaptation according to the B-L-algorithm is much more plausible in biological neural systems. This is because it is unlikely that the requirement of feedback network can be met in biological neural systems.

4) Comparing Equation (3) with Equation (5), we observe that the relevant error feedback is implicit in the weights (w_{ij}^k) and their rates of change ($\dot{w}_{ij}^k$). More precisely, the appropriate error feedback for any neuron is proportional to the derivative of the sum of the squares of the strengths of the axonic connections, because $(\sum_{n=1}^{N_{k+1}} (w_{jn}^{k+1})^2)' = 2 \sum_{n=1}^{N_{k+1}} \dot{w}_{jn}^{k+1} w_{jn}^{k+1}$.

5) Since the B-L-algorithm does not require a separate feedback network, it eliminates the needs for having a feed-forward phase and a feedback phase. Adaptation can be performed in a phaseless fashion by processing information asynchronously and concurrently.

6) The adaptation parameter α appears only at the last layer. That means α can be easily adjusted during the adaptation.

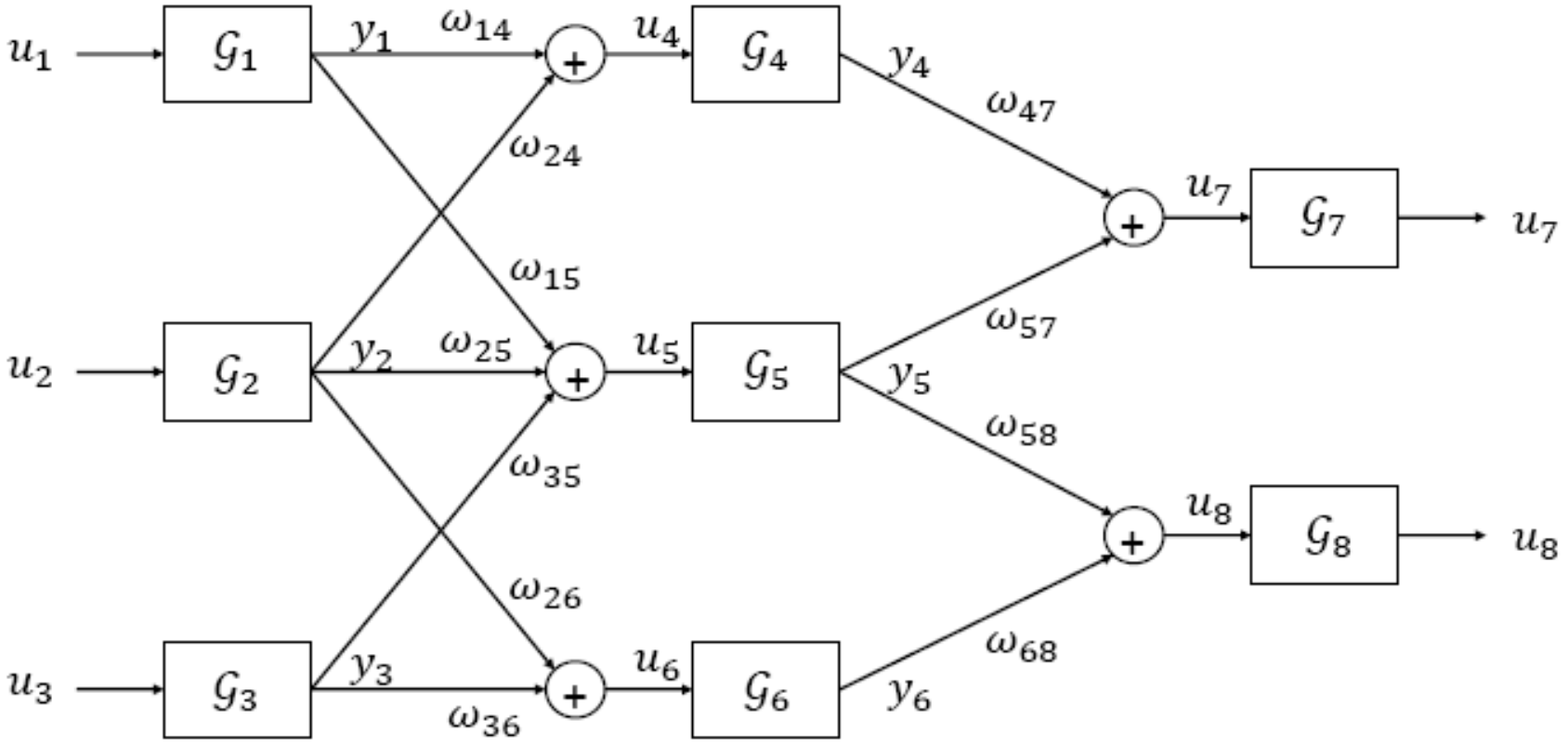
7) All layers of the neural network have the same structure, except the last layer, which is slightly different. Hence, only two types of implementation blocks need to be built: one for the last layer and one for the other layers. These blocks can then be easily connected in Simulink and other implementation plan-forms.

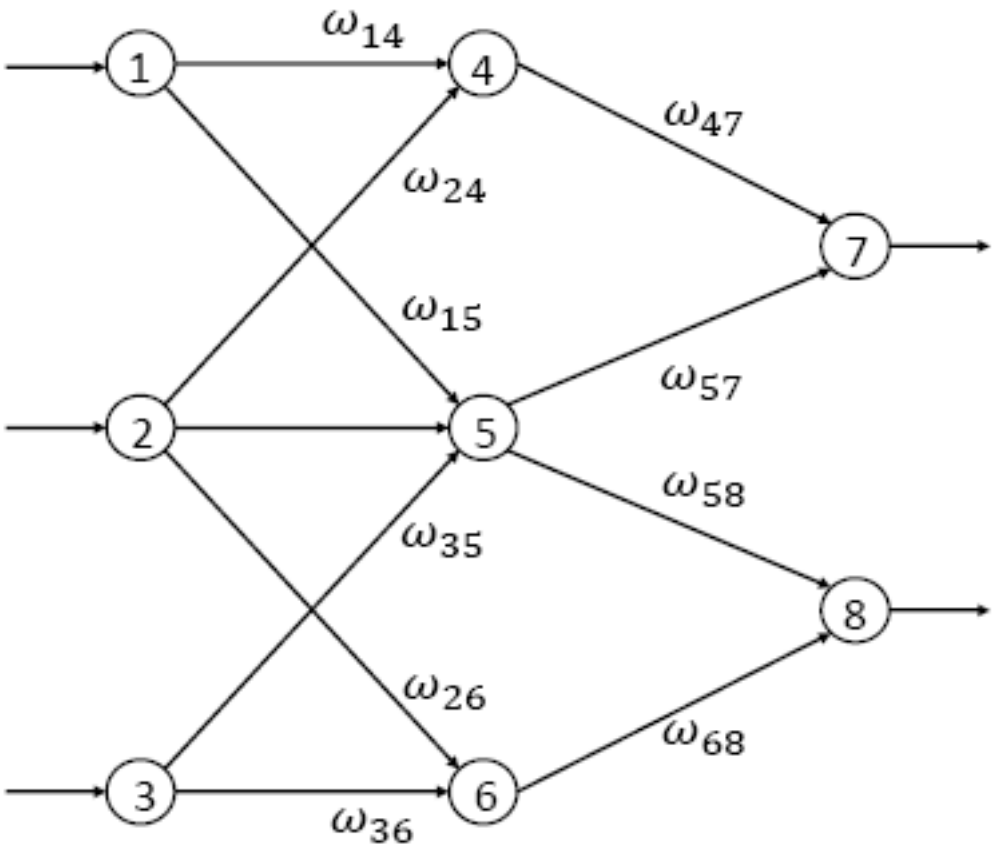
8) If there is a need to implement neurons on silicon, the B-L-algorithm provides a much simpler implementation. The elimination of feedback network significantly reduces wiring layout complexity among neurons, as there is no need to implement a separate feedback network and then connect neurons in the feed-back network with the corresponding neurons in the feed-forward network.

9) Using the B-L-algorithm, an adaptive neuron can be designed as an identical and standard unit without considering network topology or adaptation parameter α. These adaptive neurons can then be interconnected arbitrarily. Hence, the B-L-algorithm can be used to design neural networks with dynamically reconfigurable topologies.

10) Since all feedback and connections are local, implementations using the B-L-algorithm are more fault-tolerant in the sense that failures of some neurons will not cause the entire neural network to become nonfunctional.

III. Generalized Signal-Flow Graphs




1
2
3
4
5
6
7
8
ω_{14}
ω_{24}
ω_{15}
ω_{35}
ω_{26}
ω_{36}
ω_{47}
ω_{57}
ω_{58}
ω_{68}

An GSFG is similar to a CSFG, but some nodes in GSFG are super nodes.

Assume that there are N nodes. Denote a node by

$$n \in \mathcal{N} = \{1, 2, ..., N\}.$$

Denote the branch and its gain from node i to node j by ω_{ij}. The set of branches/gains is defined as

$$\Omega = \{\omega_{ij} : i, j \in \mathcal{N} \wedge i \text{ is connected to } j\}.$$

The set Ω is partitioned into two sets:

$$\Omega = \Omega_a \cup \Omega_{na},$$

where Ω_a is the set of adaptable branches/gains and Ω_{na} is the set of non-adaptable branches/gains. Non-adaptable branches have constant gains $\bar{\omega}_{ij}$.

Some nodes in $\mathcal{N}$ are super nodes. A super node consists of a pair of input and output, denoted by

$$(u_n, y_n).$$

The relationship between u_n and y_n is described by a functional

$$\mathcal{G}_n : \mathcal{R} \to \mathcal{R}.$$

For a linear system, $\mathcal{G}_n$ is given by transfer function $G_n(s)$.

$$Y_n(s) = G_n(s)U_n(s).$$

For a nonlinear system,

$$y_n(t) = \mathcal{G}_n[u_n(t)].$$

We assume that the Fréchet derivative of $\mathcal{G}_n$ exists. If a node $n \in \mathcal{N}$ is not a super node, then $y_n = u_n$.

The input signal of node n is the sum of all signals flowing to n:

$$u_n = \sum_{m=1}^{N} \omega_{mn}\ y_m. \tag{6}$$

The output signal of node n is then given by

$$y_n = \mathcal{G}_n[u_n]. \tag{7}$$

IV. On-line Learning of GSFG

Our goal is to use on-line learning to learn/adapt the gains $\omega_{ij} \in \Omega_a$ so that the least square error is minimized.

$$E = \frac{1}{2} \sum_{m \in \mathcal{O}} (y_m - \tilde{y}_m)^2,$$

where $\tilde{y}_m$ is the desired output of node $m \in \mathcal{O}$.

Use gradient decent for $\omega_{ij} \in \Omega_a$

$$\dot{\omega}_{ij} = -\alpha \frac{dE}{d\omega_{ij}}.$$

Learning algorithm for GSFG

$$\dot{\omega}_{ij} = \mathcal{G}'_j[u_j]\frac{y_i}{y_j}(-\alpha y_j(y_j - \tilde{y}_j) + \sum_{\omega_{jm}\in\Omega_a} \omega_{jm}\,\dot{\omega}_{jm} + \sum_{\omega_{jm}\in\Omega_{na}} \bar{\omega}_{jm}\,\dot{\omega}_{jm}). \tag{8}$$

Two special cases.

Case 1: Node j is an output node. In this case, Equation (8) reduces to

$$\dot{\omega}_{ij} = -\alpha y_i \mathcal{G}'_j[u_j](y_j - \tilde{y}_j). \tag{9}$$

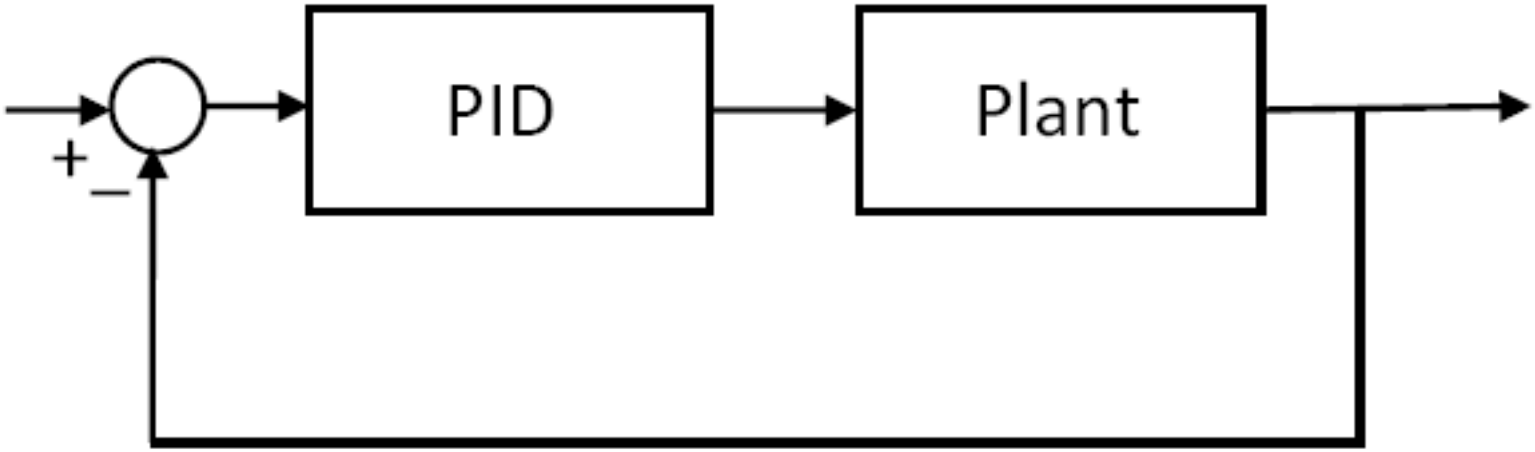
Case 2: Node j is not an output node. In this case, Equation (8) reduces to

$$\dot{\omega}_{ij} = \mathcal{G}'_j[u_j]\frac{y_i}{y_j}\Big(\sum_{\omega_{jm}\in\Omega_a} \omega_{jm}\ \dot{\omega}_{jm} + \sum_{\omega_{jm}\in\Omega_{na}} \bar{\omega}_{jm}\ \dot{\omega}_{jm}\Big). \tag{10}$$

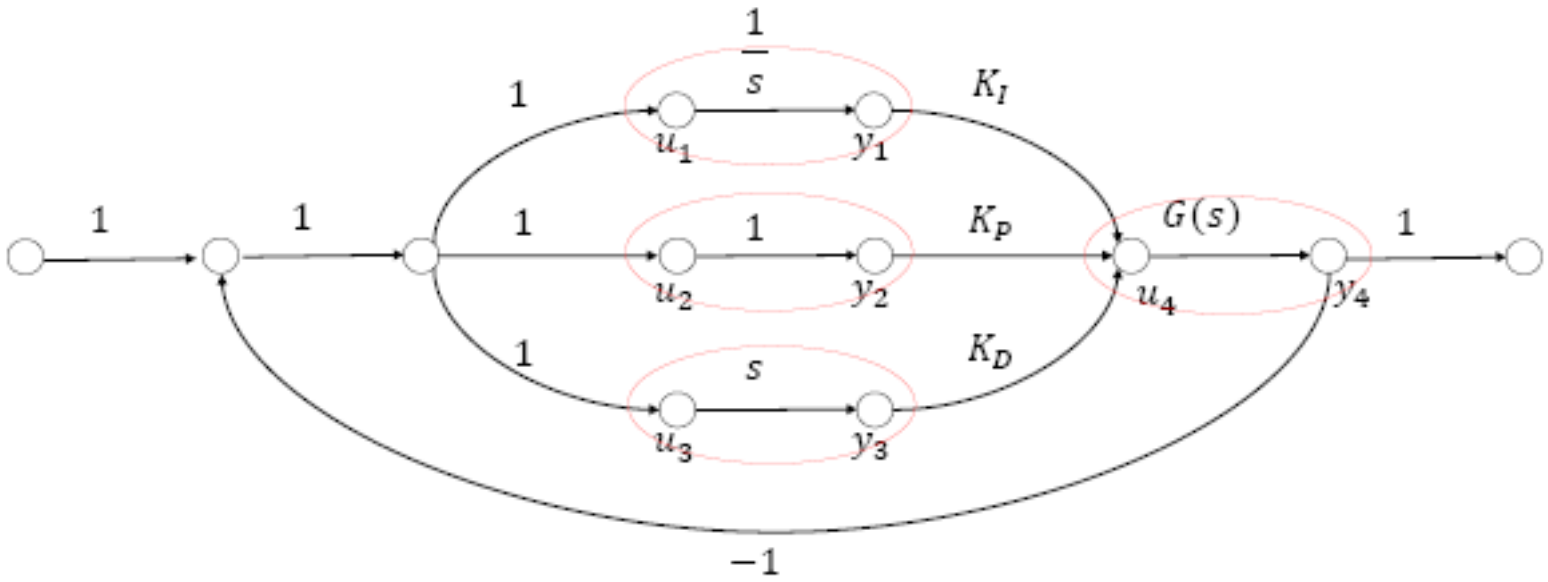
Note that the adaptation parameter α does not appear in the above equation.

V. Adaptive PID Control

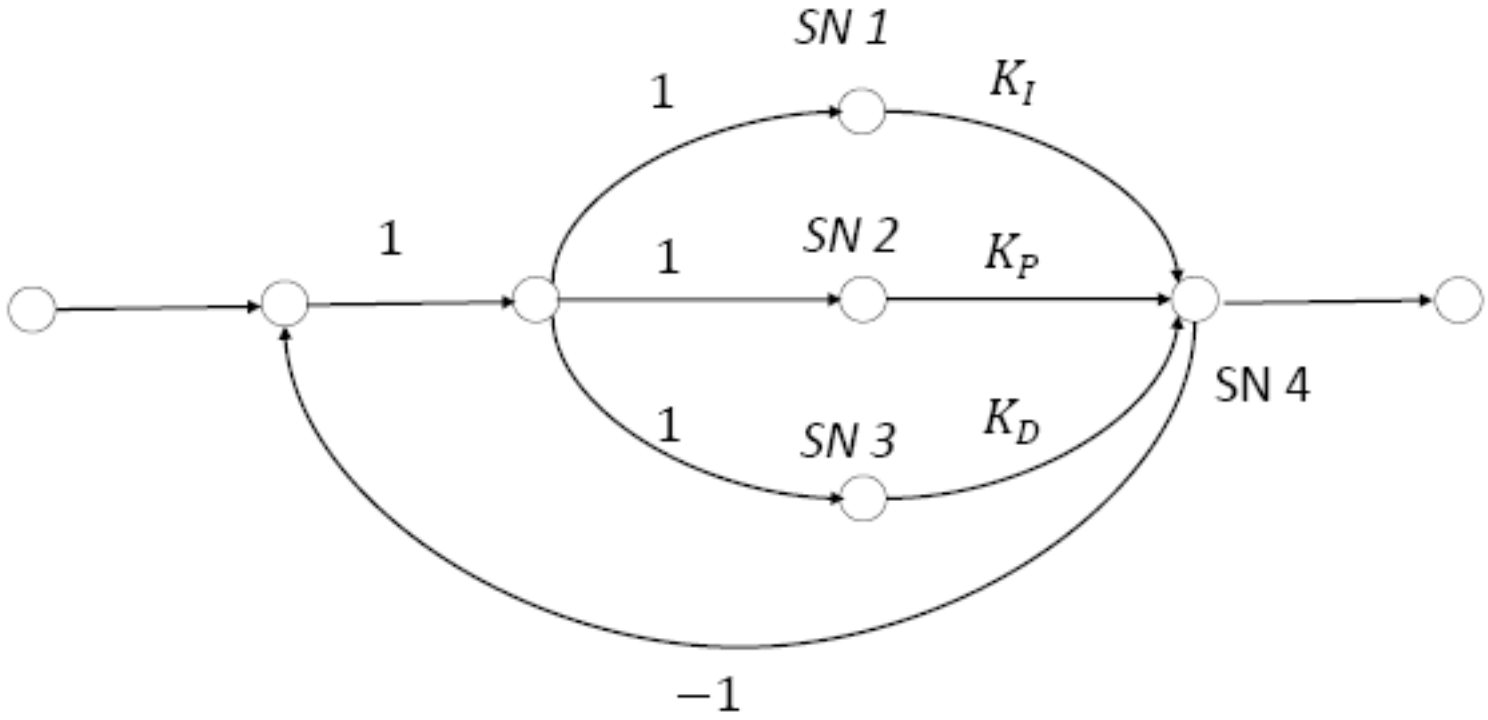
A feedback control system with a PID controller.

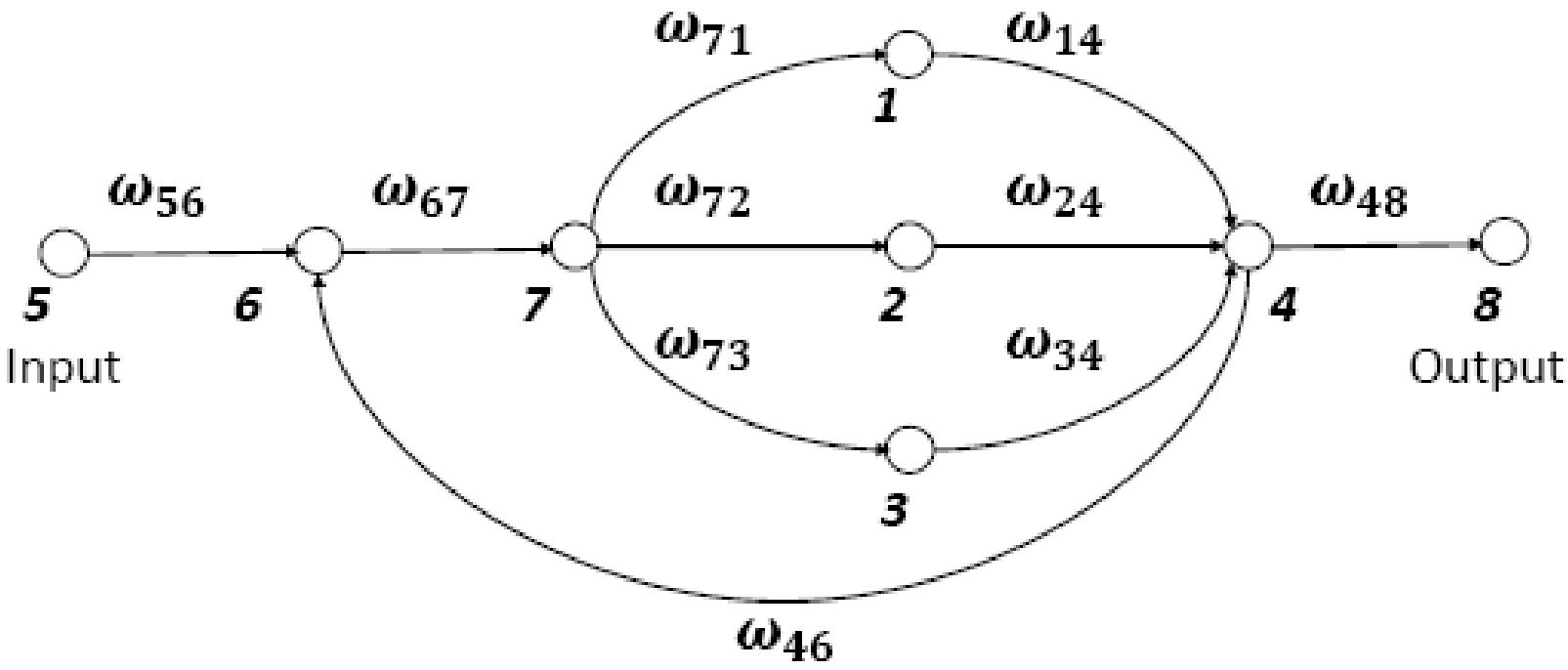


Conventional signal-flow graph (CSFG)



Generalized signal-flow graph (CSFG)




ω_{71}
ω_{14}
1
ω_{56}
ω_{67}
ω_{72}
ω_{24}
ω_{48}
5
6
7
2
4
8
Input
ω_{73}
ω_{34}
Output
3
ω_{46}

The PID control system consists of 8 nodes, including 4 super nodes (SN).

SN 1 is the integral part $G_1(s) = 1/s$.

SN 2 is the proportional part, $G_2(s) = 1$.

SN 3 is the derivative part, $G_3(s) = s$.

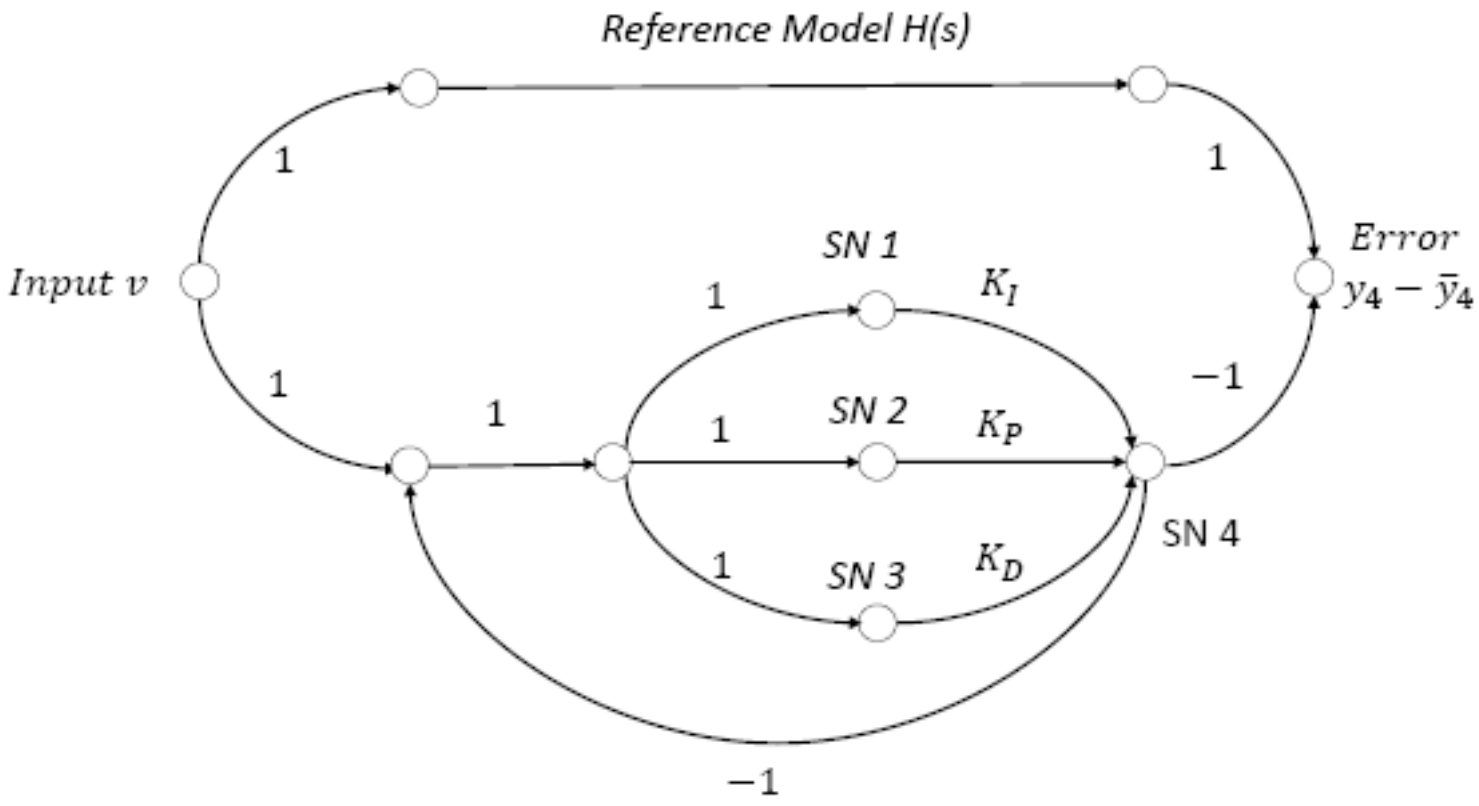
SN 4 is the plant, given by either $G_4(s)$ (linear) or $\mathcal{G}_4$ (nonlinear).

The PID gains are represented by branch gains as follows.

$$\begin{aligned} &\text{Integral:} && K_I = \omega_{14} \\ &\text{Proportional:} && K_P = \omega_{24} \\ &\text{Derivative:} && K_D = \omega_{34} \end{aligned}$$

All other branch gains are 1, except $\omega_{46} = -1$.

The model reference adaptive PID control



The objective is to adapt the gains K_I, K_P, K_D so that the output of the controlled system follows the output of the reference model.

The reference model: transfer function $H(s)$ or impulse response $h(t)$:

$$\tilde{y}_4(t) = h(t) * v(t).$$

Hence,

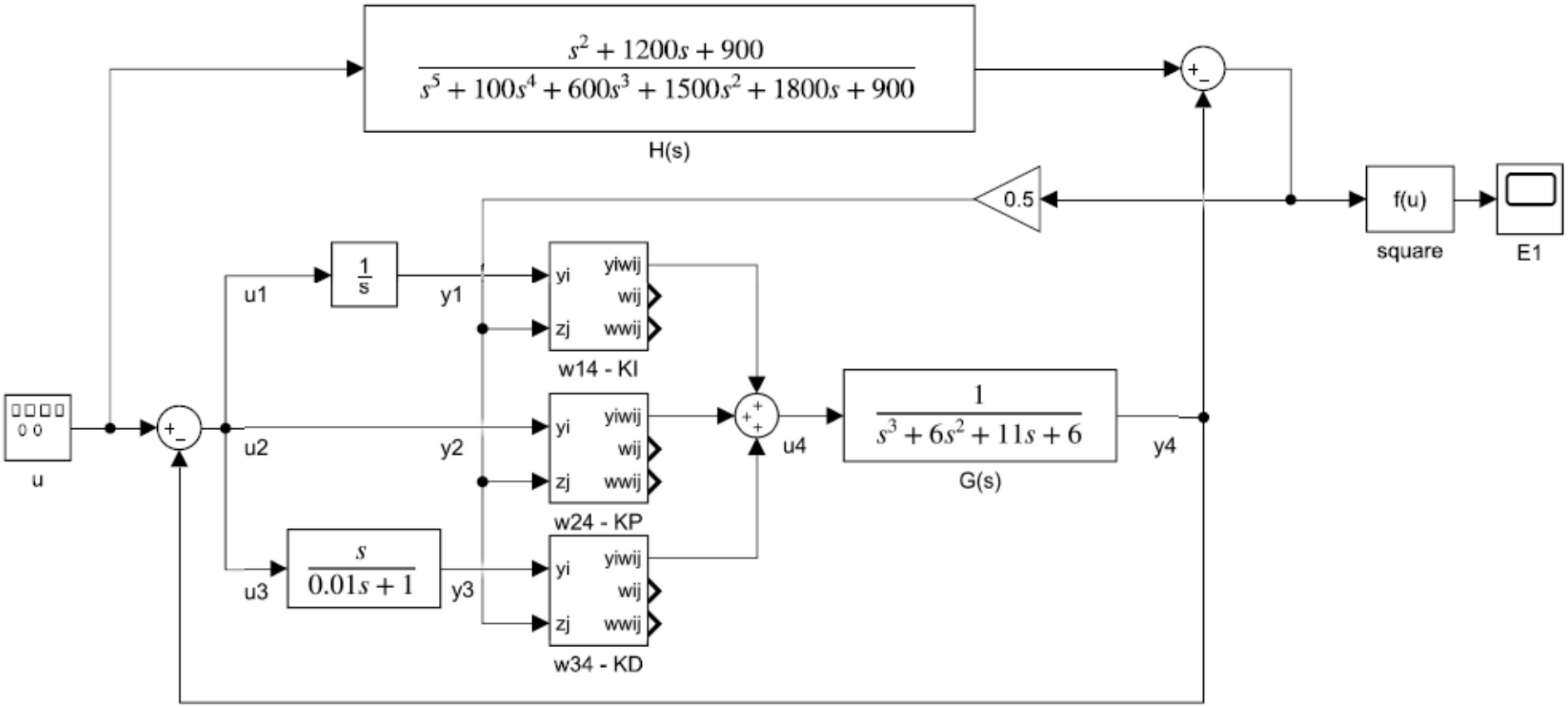
$$E = \frac{1}{2}(y_4 - \tilde{y}_4)^2.$$

SN 4 is an output node:

$$\dot{\omega}_{i4} = -\gamma y_i \mathcal{G}_4'[u_4] \frac{\partial E}{\partial y_4} = -\gamma y_i \mathcal{G}_4'[u_4](y_4 - \tilde{y}_4).$$

$$\begin{aligned} \dot{K}_I &= -\gamma y_1 \mathcal{G}_4'[u_4](y_4 - \tilde{y}_4) \\ \dot{K}_P &= -\gamma y_2 \mathcal{G}_4'[u_4](y_4 - \tilde{y}_4) \\ \dot{K}_D &= -\gamma y_3 \mathcal{G}_4'[u_4](y_4 - \tilde{y}_4). \end{aligned} \tag{11}$$

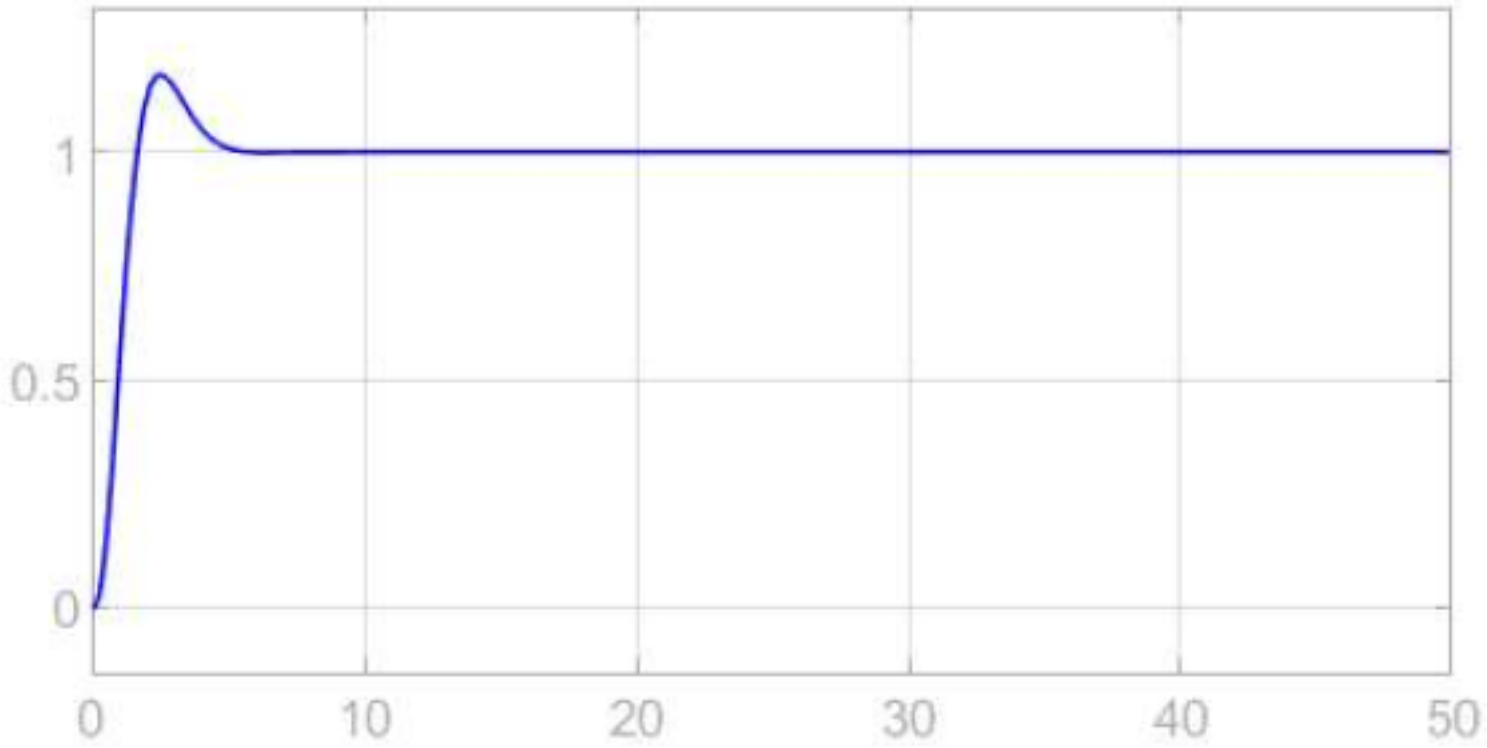
VI. Simulation Results



Reference model

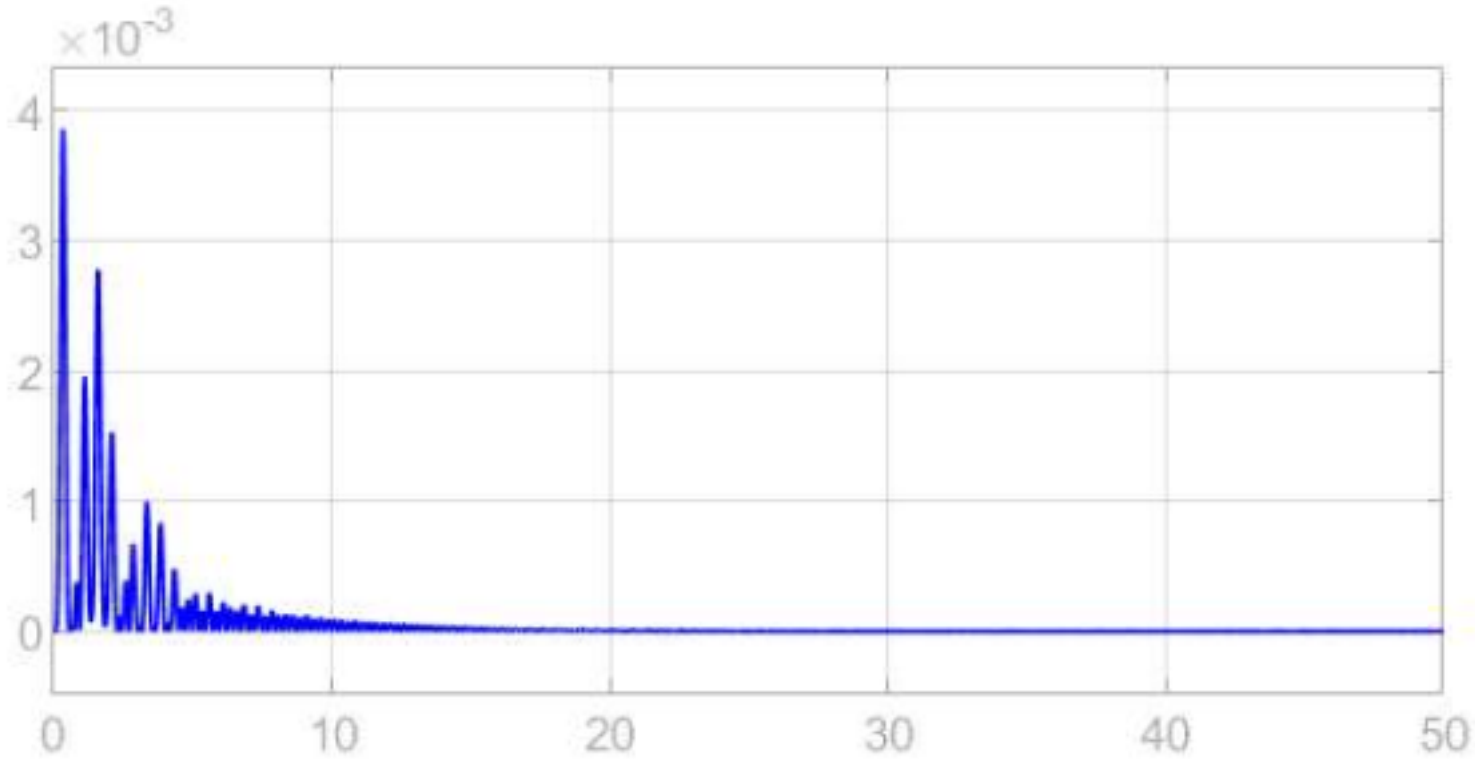
$$H(s) = \frac{s^2 + 1200s + 900}{s^6 + 100s^4 + 600s^3 + 1500s^2 + 1800s + 900}.$$

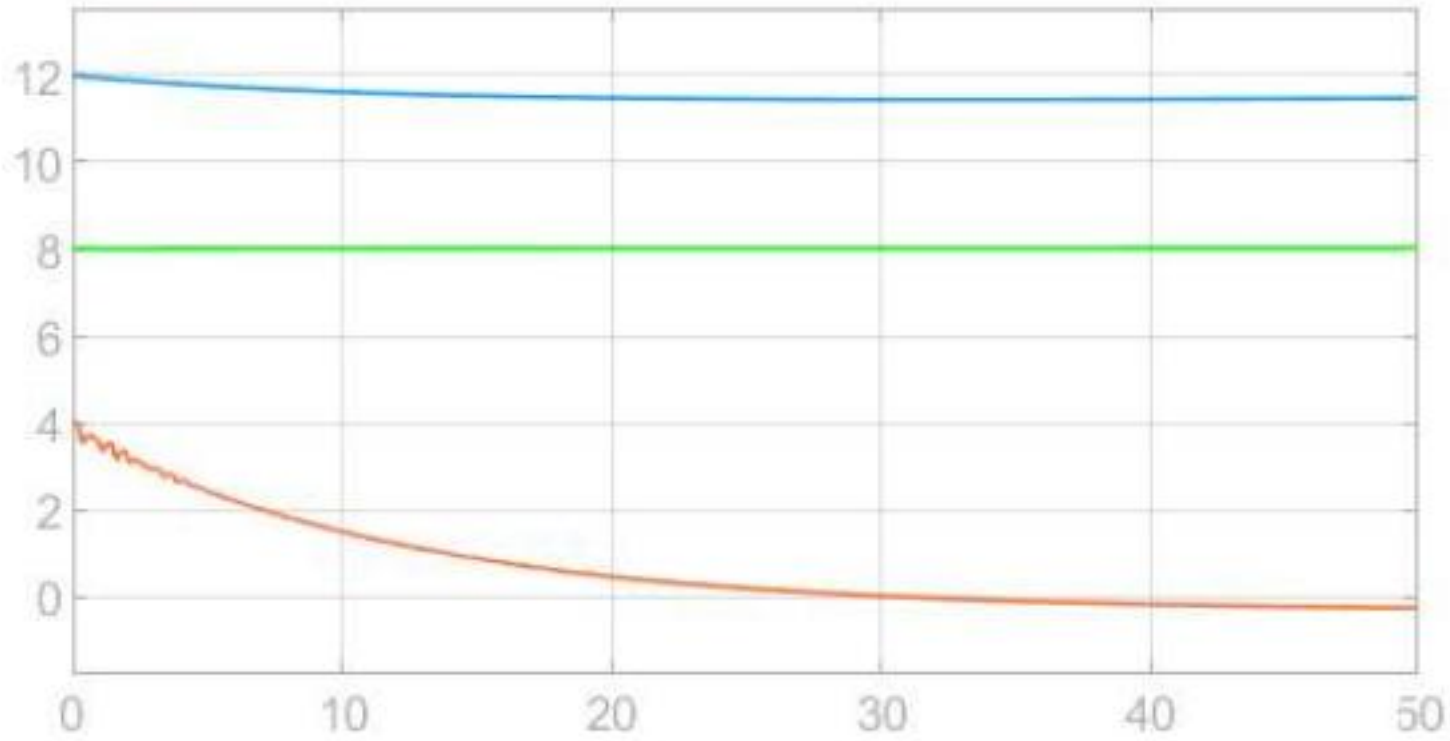
Step response of the reference model $H(s)$



Stable Plant

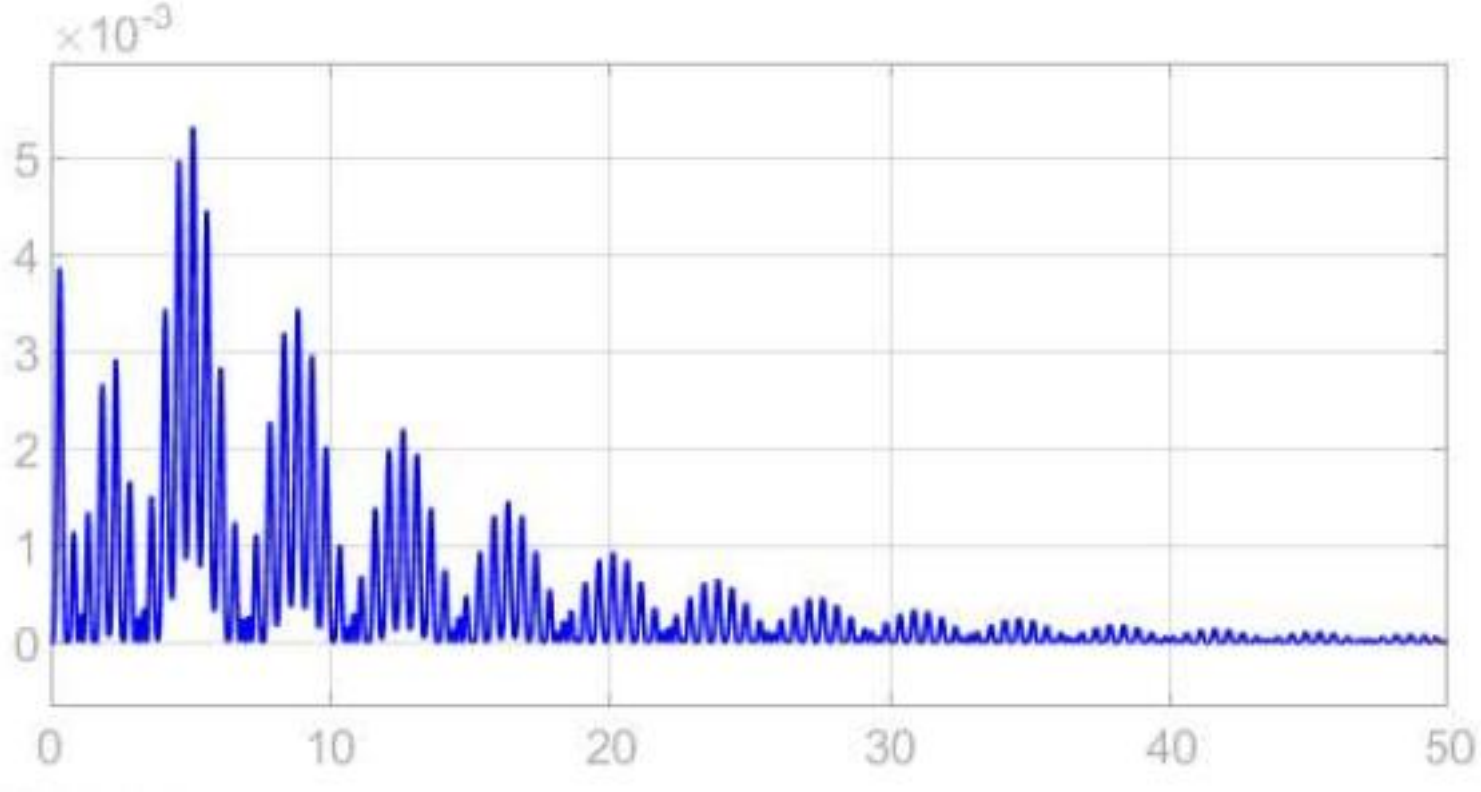
$$G_1(s) = \frac{1}{s^3 + 6s^2 + 11s + 6}.$$

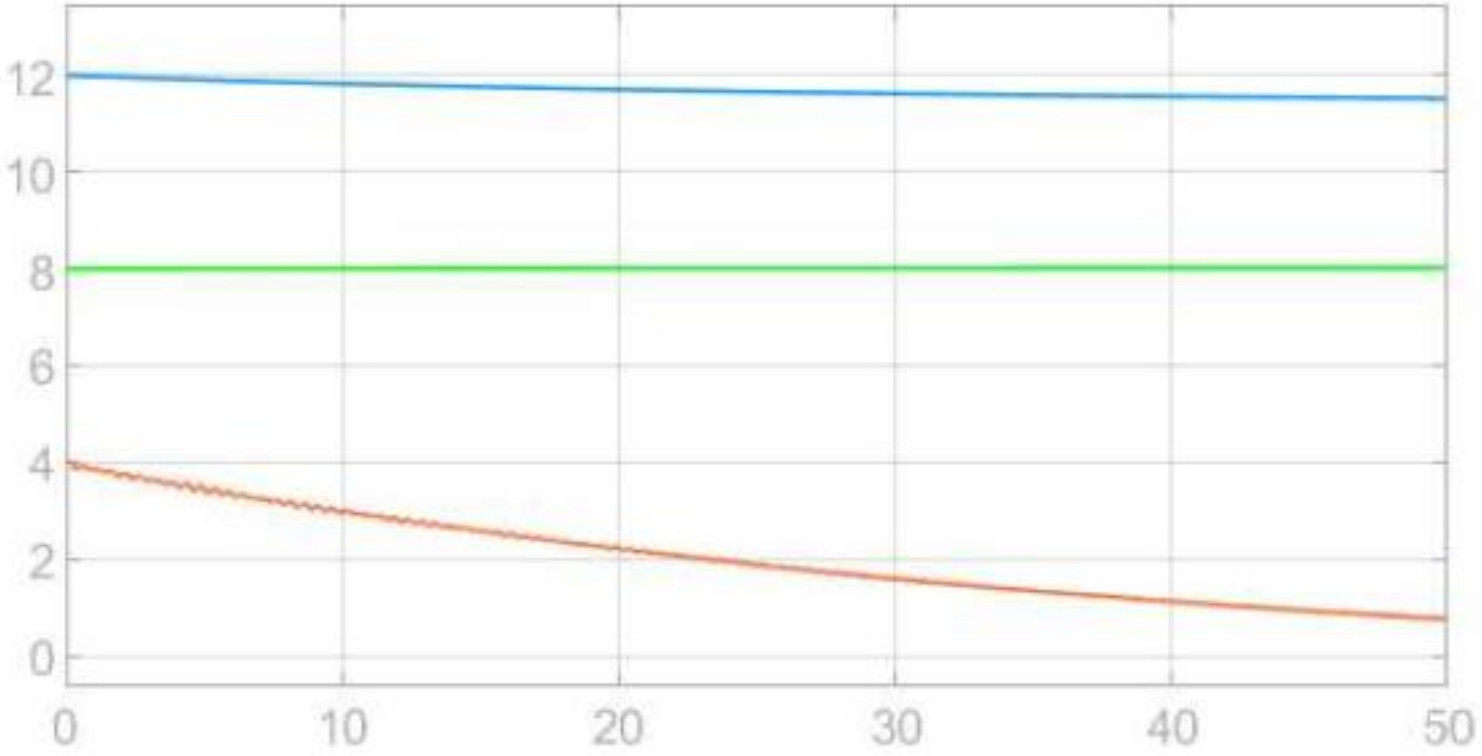



12
10
8
6
4
2
0
0
10
20
30
40
50

Unstable Plant

$$G_2(s) = \frac{1}{s^3 + 6s^2 + 11s - 6}.$$

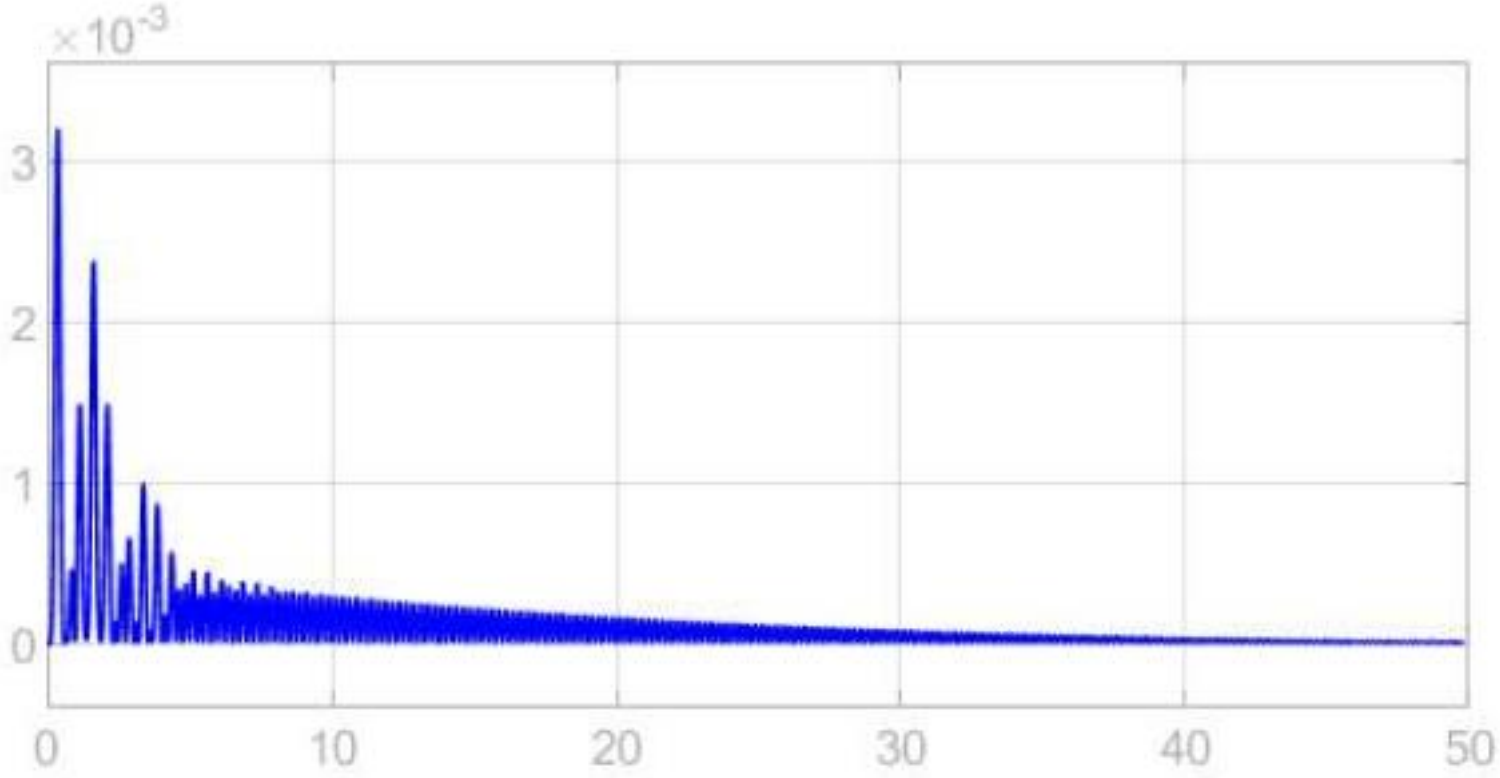


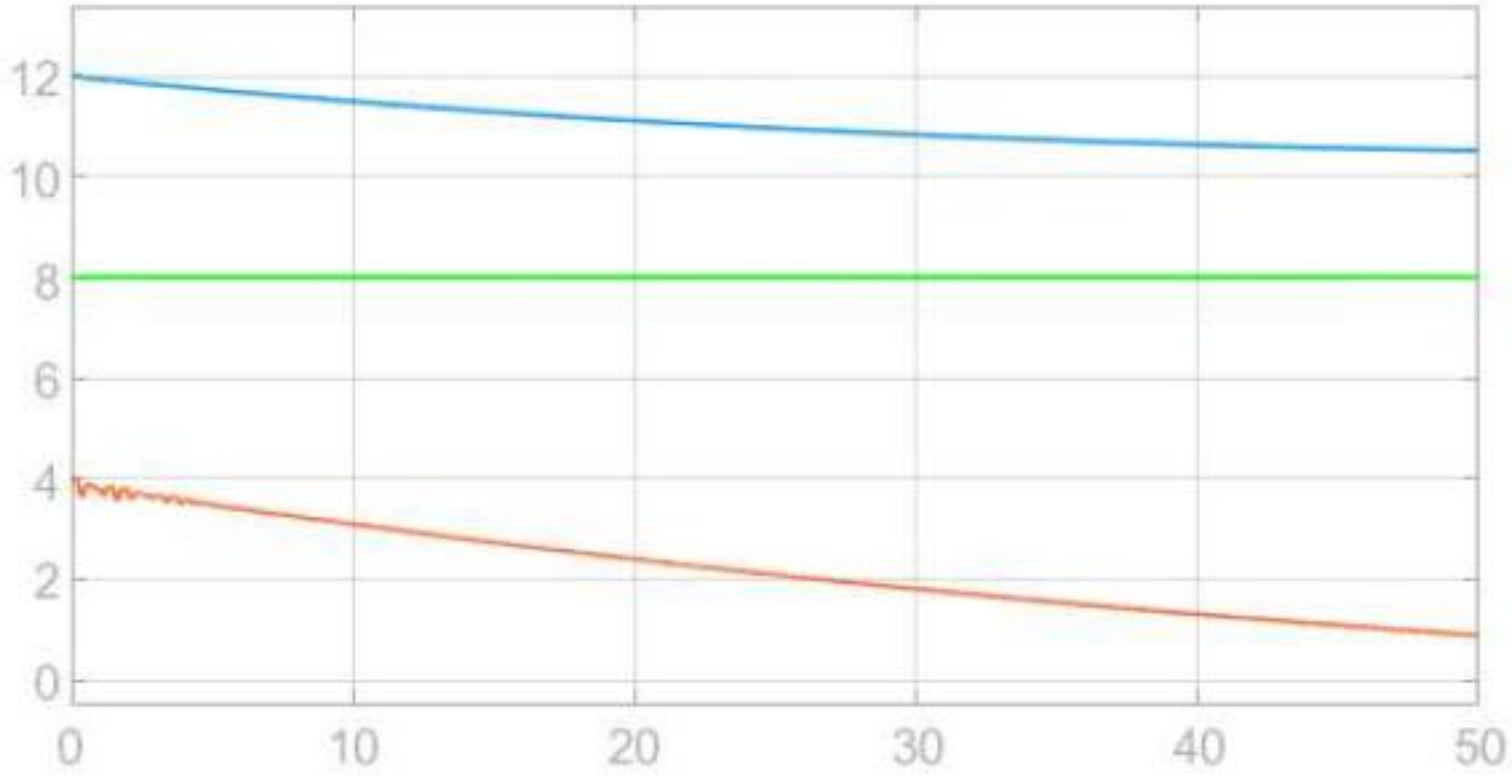

12
10
8
6
4
2
0
0
10
20
30
40
50

Systems with Time Delay

Add a delay of 0.03 second before

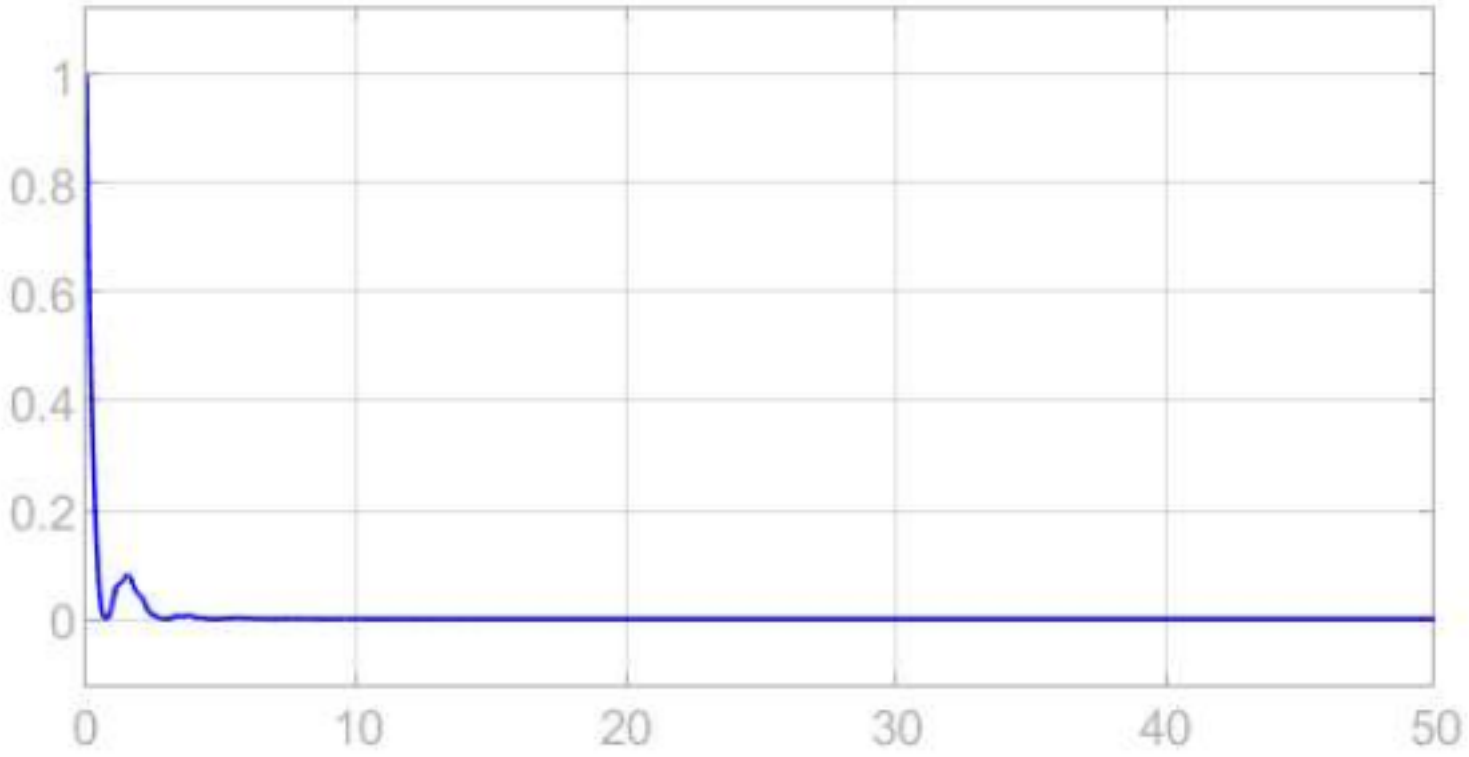
$$G_1(s) = \frac{1}{s^3 + 6s^2 + 11s + 6}.$$

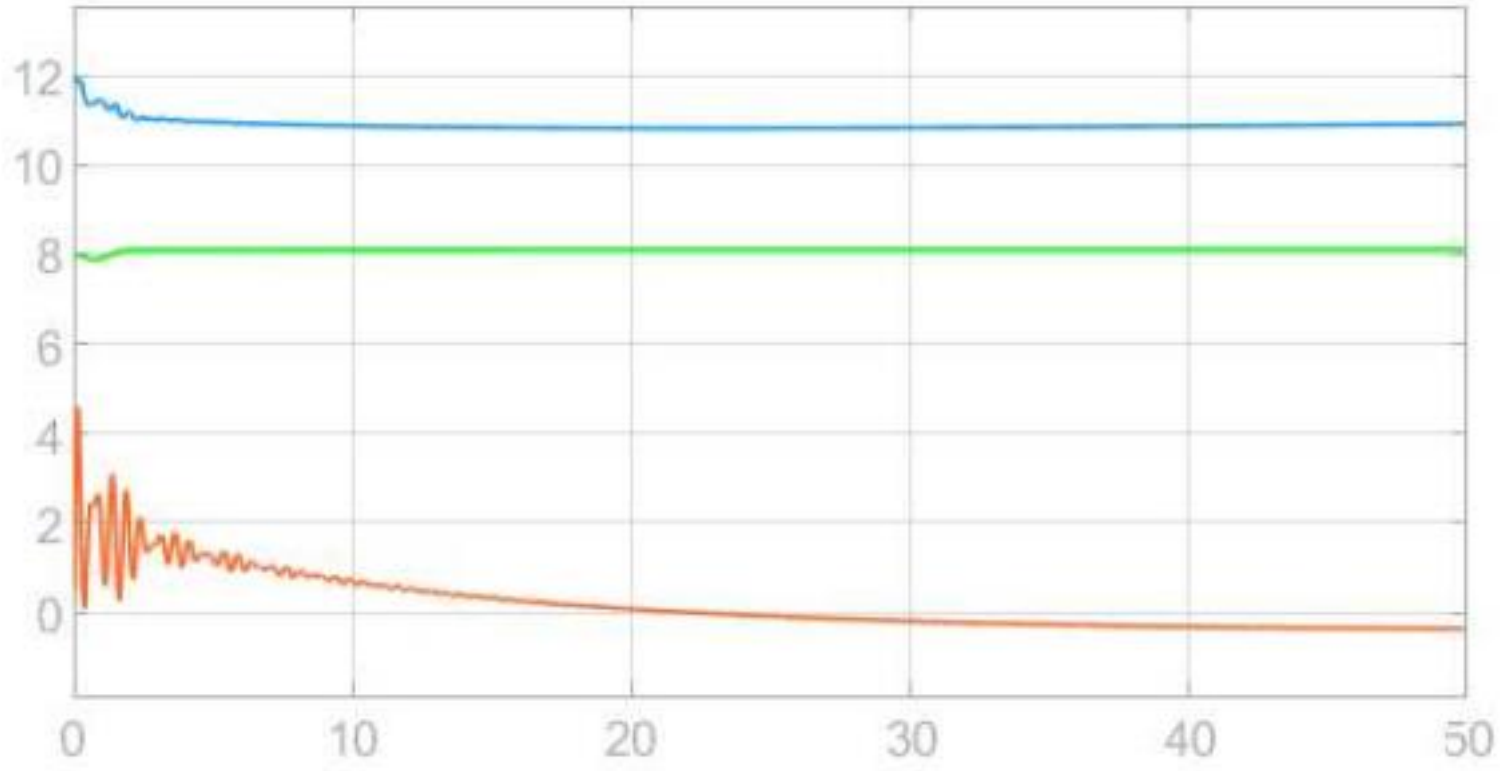



12
10
8
6
4
2
0
0
10
20
30
40
50

Nonlinear Systems

$$\dot{x}_1 = -x_1 + 0.5\sin(x_1) + u_4$$
$$\dot{x}_2 = -2x_2 - x_2^3 + x_1$$
$$\dot{x}_3 = -3x_3 - 0.2\tan(x_3) + x_2$$
$$y_4 = x_3.$$


1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50


12
10
8
6
4
2
0
0
10
20
30
40
50

Conclusions

- The B-L-algorithm is mathematically equivalent to the C-B-algorithm, but has many advantages in terms of implementation and biological plausibility.
- The B-L-algorithm can be generalized from neural networks to general systems, making them capable of learning just like neural networks.
- One practical application is to model reference adaptive PID control.

References

- F. Lin, “Supervised learning in neural networks: Feedback-network-free implementation and biological plausibility,” IEEE Transactions on Neural Networks and Learning Systems, 2021.
- F. Lin, R.D. Brandt and G. Saikalis, 2000, “Self-tuning of PID controllers by adaptive interaction,” Proceedings of the 2000 American Control Conference, pp. 3676-3682.
- R. D. Brandt and F. Lin, 1999, “Adaptive interaction and its application to neural networks,” Information Sciences 121, pp. 201-215.
- R. D. Brandt and F. Lin, 1996, “Supervised learning in neural networks without feedback network,” Proceeding of the 1996 IEEE International Symposium on Intelligent Control, pp. 86-90.
- R. D. Brandt and F. Lin, 1994, “Supervised learning in neural networks without explicit error back-propagation,” Proceeding of the 32nd Annual Allerton Conference on Communication, Control, and Computing, pp. 294-303.

THANK YOU!

flin@wayne.edu