A Neural Network Based Automatic Generation Controller Design through Reinforcement Learning

Imthias Ahamed T.P.*  Nagendra Rao P.S.†
Sastry P.S.‡

*T. K. M. College of Engineering, imthiasa@gmail.com
†Indian Institute of Science, Bangalore, nagendra@ee.iisc.ernet.in
‡Indian Institute of Science, Bangalore, sastry@ee.iisc.ernet.in

Copyright ©2006 The Berkeley Electronic Press. All rights reserved.
Abstract

This paper presents the design and implementation of a learning controller for the Automatic Generation Control (AGC) in power systems based on a reinforcement learning (RL) framework. In contrast to the recent RL scheme for AGC proposed by us, the present method permits handling of power system variables such as Area Control Error (ACE) and deviations from scheduled frequency and tie-line flows as continuous variables. (In the earlier scheme, these variables have to be quantized into finitely many levels). The optimal control law is arrived at in the RL framework by making use of Q-learning strategy. Since the state variables are continuous, we propose the use of Radial Basis Function (RBF) neural networks to compute the Q-values for a given input state. Since, in this application we cannot provide training data appropriate for the standard supervised learning framework, a reinforcement learning algorithm is employed to train the RBF network. We also employ a novel exploration strategy, based on a Learning Automata algorithm, for generating training samples during Q-learning. The proposed scheme, in addition to being simple to implement, inherits all the attractive features of an RL scheme such as model independent design, flexibility in control objective specification, robustness etc. Two implementations of the proposed approach are presented. Through simulation studies the attractiveness of this approach is demonstrated.

KEYWORDS: power system control, automatic generation control, neural networks, reinforcement learning, radial basis function networks

*First author acknowledges the support given by All India Council for Technical Education under TAPTEC scheme (Ref. no: 8021/RID/NPROJ/TAP-139/2002-03).
Introduction

Interconnected power systems are made up of several areas. The Automatic Generation Controller (AGC) in each area monitors the system frequency and the inter-area tie line flows, computes the net change in the area generation required (generally referred to as Area Control Error, ACE) and changes the set points of the generators within the area so as to keep the time average of the ACE at a low value. AGC function can be viewed as a supervisory control function which attempts to match the generation trend within an area, to the trend of the randomly changing load of the area, so as to keep the system frequency and tie line power flow close to scheduled values.

Most of the current implementations of AGC choose the control objective as that of keeping the time average of the ACE at a low value. Many control schemes have been proposed for AGC (see, e.g., [1, 2, 3]). In the recent past, application of Neural Network based control [4, 5], Fuzzy Control [6] and Genetic Algorithm [7] have also been proposed. In this paper we consider application of a neural network controller trained through reinforcement learning for design of AGC. We first explain our motivation for adopting the reinforcement learning approach and then briefly explain the contributions of this paper.

There are several issues which are important in a practical implementation of AGC. As is well recognized in industry circles [8], AGC acts at a slower time scale (as compared to rest of the power system) and thus considering AGC and rest of the power system as a single feedback control system is not very appropriate. Almost all AGC algorithms available in the literature employ simplified linear models and use the standard turbine generator models to represent area dynamics. However, it is desirable to have an AGC design methodology that does not need any system model and which is capable of taking care of physical limitations of the subsystems such as the upper bound on the amount of increase or decrease in generated power which can be achieved in a given small time interval. Another important issue is the overall objective of AGC. Most of the currently available techniques in the literature choose objectives such as quickly achieving zero ACE. However, in a practical system the overall objective of an AGC has a significant qualitative part in it [8] and hence, it is desirable to have a design methodology where such objectives can be easily incorporated.

Recently we have proposed [9, 10, 11] that the AGC problem can be viewed as a stochastic multistage decision making problem and have pre-
Presented algorithms for designing AGC based on the Reinforcement Learning (RL) approach. It has been shown that the RL method is attractive because it gives sufficient scope for addressing all the issues mentioned above (see the discussion in [9]).

A major strength of RL approach is the fact that it is model independent. What the learning algorithm require is only a "black box model" which will compute the output for any given input. Therefore, incorporating the nonlinearities such as Governor dead-band, generation rate constraint (GRC) are straightforward. We have demonstrated such a controller for the discrete state space case in [9] considering a 2-area system. We have shown the effectiveness of this approach for systems with reheat turbines [10, 11] and for a 4-area system having both hydro and thermal units [12].

The RL-AGC design essentially consists of learning the so called optimal Q-values or Q-function [13] using a series of examples generated either online or through a simulation model. The training examples are traces of closed-loop system trajectories in response to different kinds of load disturbances. The optimal Q-values are represented by \( Q^*(x, a) \), where \( x \) is the state information input to the AGC and \( a \) is the control action (or the output of the AGC). Essentially, \( Q^*(x, a) \) indicates the goodness of the control action \( a \) in response to the observed state \( x \). (See [9] for a detailed discussion).

If we know the Q-function, then the best control strategy is to choose, in response to the state \( x \), the action which maximizes \( Q^*(x, a) \) over all \( a \). The Q-values depend on the dynamics of the system and hence, in the absence of detailed knowledge of system dynamics, they are unknown. Thus the crux of this RL based approach to AGC design is that of learning the (optimal) Q-values.

In the RL-AGC algorithms presented earlier [9, 10], it is assumed that the state information input to AGC as well as the output of AGC can take only finitely many values. While the output of the AGC (which represents change in set point of the governor) can be assumed discrete, assuming that the input to AGC is discrete is unrealistic because most system variables are continuous. However, such an assumption considerably simplifies the problem of learning the Q-values. With this assumption, the Q-values, can be stored as a finite table and the algorithms essentially learn a table-look-up controller.

For implementing such a controller, we need to quantize the observed values of state variables into finitely many levels. However, if the dynamic range of the state variables is large, or if we want to have a fine quantization, or if there are many state variables, the finite state space could become

---

10.2202/1553-779X.1141
Ahamed T.P. et al.: A Neural Network Based AGC Design through Reinforcement Learning

In order to extend our RL-AGC to continuous state space case, we need to learn Q-function (or Q-values) over the continuous state space. However, standard RL approach is best suited for finite state space. There have been broadly two approaches to tackle this issue. One approach is to use a fuzzy inference system as an interpolator of the discrete RL controller [14, 15]. The fuzzy approach is often more attractive when we want, in addition to a continuous state space, the actions or outputs of the controller also to be continuous. In using a fuzzy inference system, the interpolation at the input end (that is, in the state space) is essentially designed by the choice of membership functions etc., and reinforcement learning is used to tune the conclusions part of the rule base [15]. Such a fuzzy inference system for extending our RL approach to AGC has been tried and is found to be quite effective [16]. A second approach is to use an appropriate parametrized function class (e.g., neural networks), and learn the best function (parameter vector) in this class to approximate the optimal Q-function [13, 17]. This is the method we pursue in this paper. Here, while we take the state space (or the input space of the controller) to be continuous, the actions or outputs of the controller are discrete. The main problem here is to learn the Q-function over the continuous state space ensuring proper generalization from the experience gained during training. In the fuzzy inference system approach, the interpolation of the Q-function to the continuous state space is somewhat arbitrary in the sense that it is very much influenced by the choice of fuzzy membership function by the designer. In contrast, in the approach followed here, the proper interpolation is learnt through the training phase. This function approximator approach is interesting in itself because here we need to learn the function using only reinforcement feedback.

In this method we need to choose an appropriate parameterized function class and a learning algorithm for obtaining a good approximation to the optimal Q-function over the continuous state space. In this paper, we propose the use of Radial Basis Function neural network for this purpose. We denote the RL-AGC proposed here, which uses Radial Basis Function (RBF) network to represent Q-values, as RL-RBF-AGC. Typically all RBF neural network models are trained in the supervised mode. However, in our case the examples are only system traces, and we have no knowledge of
the true Q-values. Thus supervised training of the RBF neural network is infeasible. In this paper, we propose a reinforcement learning algorithm for training our RBF network.

For learning Q-values we need to generate a training set of examples (from a simulation model) using an exploration policy. In this paper, we introduce a novel exploration policy using ideas from the Learning Automata Theory [18]. A special feature of this approach is that the exploration policy is also updated after each example. In itself, this would constitute a novel exploration policy for RL schemes with continuous state spaces.

We demonstrate the effectiveness of the new approach by presenting two different AGC implementations. In the first implementation, we consider ACE as the only state variable. This is a novel RL-AGC design because it does not use any information regarding rate of change of ACE. The second AGC implementation is based on two state variables. The two quantities chosen as the state variables in this case are the two measured quantities in the system: $\Delta f$ (frequency deviation) and $\Delta P_{tie}$ (tie-line flow deviation). The special merit of this choice (as elaborated in [16]) is that we no longer need to assume knowledge of $\beta$, the natural response of the power system areas. This quantity can often change with the system state and hence the assumption of constant (and known) $\beta$ is a rather restrictive assumption which is employed in all other AGC designs.

The rest of this paper is organized as follows. In section 2 we give a brief overview of RBF neural networks mainly to introduce our notation. Section 3 describes our first RBF network based AGC. We present detailed discussion on the architecture of the network, the learning algorithm and the overall design methodology. Section 4 then describes the second RBF-AGC that uses observed deviations in frequency and tie-line flows as the state variables. Section 5 presents simulation results to illustrate the performance of the design and Section 6 concludes the paper.

2 Radial Basis Function Networks

Neural networks constitute a useful class of parameterized models for learning a function from examples provided [19]. For our purpose, the Radial Basis Function (RBF) neural network is attractive because here the target function is represented as a sum of many ‘local’ functions.

Figure 1 shows a typical RBF network consisting of three layers: the input layer with $n$ input nodes, the hidden layer with $m$ hidden nodes, and
the output layer with one output node. The output of the network for an input $x \in \mathbb{R}^n$ is given by

$$y = \sum_{i=1}^{m} W_i \phi_i(x), \ x \in \mathbb{R}^n \tag{1}$$

where $\{W_i : i = 1, \ldots, m\}$ are the weights of the connections into the (linear) output node and $\{\phi_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m\}$ is a set of $m$ radial basis functions. (These $m$ radial basis functions (rbf’s) constitute the $m$ hidden nodes of the RBF network). A popular rbf is the Gaussian function. The output of a Gaussian RBF network with $m$ hidden nodes is given by

$$y = \sum_{i=1}^{m} W_i \exp \left( -\frac{\sum_{j=1}^{n} (x_j - c_i^j)^2}{2(\sigma_i^2)} \right). \tag{2}$$

where $c_i^j = [c_1^i, \ldots, c_n^i]^t$, $i = 1, \ldots, m$ are called the centers of the RBF\’s and $\sigma_i^j$, $i = 1, \ldots, m$ are called their widths. (Here, $x = [x_1, \ldots, x_n]^t$ is the $n$-dimensional input vector to the RBF net). This RBF network is completely
specified by the parameter vector \( \theta = [c, \sigma, W] \) where \( c = \{c^1, \ldots, c^m\} \), \( \sigma = \{\sigma^1, \ldots, \sigma^m\} \), and \( W = [W_1, \ldots, W_m]^t \). In many of the applications of RBF networks, the parameters \( c \) and \( \sigma \) are fixed heuristically and then the weights into the output layer are learnt to minimize the error.

Extension of the RBF network given in Fig. 1 to represent a map \( F : X \rightarrow Y; X \subset \mathbb{R}^n, Y \subset \mathbb{R}^p \) is straightforward [19]. The only difference is that the RBF network will have \( p \) output nodes. Associated with each output node there is a set of weights. We denote the weights of the \( l^{th} \) output node as \( W[l][1], W[l][2], \ldots, W[l][m] \). With this notation, the \( l^{th} \) output is

\[
y_l = \sum_{i=1}^{m} W[l][i] \exp \left( -\frac{\sum_{j=1}^{n} (x_j - c^j_i)^2}{2(\sigma^i)^2} \right)
\]

It may be noted that \( W \) in this case is a \( p \times m \) matrix of real numbers.

Normally, RBF networks are trained in a supervised learning framework. In our case, we want the network to learn (to represent) the optimal Q-values. To use the standard learning techniques (e.g., [19]), we need data in the form of \( Q^*(x_i, a_i) \) for various \( x_i, a_i \). However, the Q-values are not known since we assume no knowledge of the Power System model. Hence we employ a reinforcement learning algorithm for training the RBF network as explained in the next section.

3 RL-RBF-AGC with ACE as the State Variable: RLRB1

In this section we explain our first implementation of RBF-based AGC which uses only one state variable input to AGC, namely the area control error (ACE). (We denote this input to AGC by \( x \)). We refer to this AGC as RLRB1.

3.1 Architecture of the RBF network for representing Q-values

We need to represent (and learn) the Q-value for every state-action pair, \((x, a)\). Here \( x \) is a continuous variable while \( a \) can assume only finitely many values. (Recall that each action represents a specific change to the amount of generation). Let \( p \) denote the number of possible actions and let the set of actions be \( A = \{a_1, \ldots, a_p\} \). A straightforward way to represent Q-values using an RBF network is to use a network with two inputs, namely
the state-action pair \((x, a)\) and one output which gives (an approximation for) \(Q^*(x, a)\). However, for computational efficiency, it turns out to be better to have a network with one input (that is the state \(x\)) and \(p\) outputs. Each of the outputs would be the (approximate) Q-value of the given state (input) and one of the actions. The architecture of this RBF network is given in Fig. 2. This RBF network has one input, \(m\) hidden units (radial basis functions), and \(p\) outputs.

The input to the RBF network, \(x\), is obtained as follows. First the average value of the ACE in the last AGC decision cycle is calculated. If the magnitude of the average value of ACE is less than (a predetermined bound) \(L_{ACE}\), the input of the RBF network is obtained by dividing the average value of ACE by \(L_{ACE}\). Otherwise, \(x\) is set to +1 or −1 depending on whether the average value of ACE is greater than \(+L_{ACE}\) or less than \(−L_{ACE}\) respectively.

Each hidden unit is a Gaussian function with center \(c^i\) and width \(\sigma^i\). The \(p\) outputs of the RBF network are denoted by \(y_1, y_2, \ldots, y_p\), each of which is a linear combination of the outputs of the hidden units. The \(j^{th}\)
output is given by
\[ y_j = \sum_{i=1}^{m} W_Q[j][i] \exp \left( \frac{-(x - c_i^j)^2}{2(\sigma_i^j)^2} \right), \quad j = 1, \ldots, p \] (3)
where, \( W_Q \) is \( p \times m \) matrix of reals, and \( W_Q[j][1], \ldots, W_Q[j][m] \) are the weights of the \( j \)th output unit.

3.2 The Learning Algorithm for the RBF network

After the architecture is fixed, the next step is to get the ‘optimal’ parameters \( \{c_i^*, i = 1, \ldots, m; \sigma_i^*, i = 1, \ldots, m; W_Q^*[j][i], j = 1, \ldots, p, i = 1, \ldots, m\} \).

Since the ACE is normalized so that \( x \in [-1, 1] \), we choose the centers on a uniform grid with width chosen as half the distance between the centers. That is, the \( m \) centers are \((-1, -1 + \frac{2}{m-1}, \ldots, +1)\) and \( \sigma_i^* = \frac{1}{m-1}, \forall i \).

Next, we need to learn ‘optimal’ values of \( W_Q^*[j][i] \) such that \( y_j \) given by (3) would be a good approximation of \( Q^*(x, a_j), \forall x \in \mathcal{X} \). If we know the \( Q^* \)-values for a large number of state-action pairs, \( W_Q^* \) can be obtained through a standard supervised learning procedure. However, we do not know the true \( Q \)-values.

The weights are to be learnt using the training examples. Each example is of the form \((x, a_l, x', g(x, a_l, x'))\) and it represents the information that, in a specific instance, on taking action \( a_l \) in the (current) state \( x \), the next (random) state was \( x' \) and the so called immediate reinforcement for this control action is given by \( g(x, a_l, x') \). The immediate reinforcement function, \( g \), is part of the controller design and it essentially encodes the control objective. When the state-action pair in the current example is \((x, a_l)\), this example can give information only about \( Q^*(x, a_l) \) and hence our weight update should be such that we are essentially updating the estimate for this \( Q \)-value only, namely, \( y_l \). Thus, after this example, our weight update should be of the form
\[ W_{Q}^{k+1}[l][i] = W_{Q}^{k}[l][i] + \Delta W_{Q}^{k}[l][i], \quad i = 1, \ldots, m \] (4)
\[ W_{Q}^{k+1}[j][i] = W_{Q}^{k}[j][i], \quad j = 1, \ldots, p, \quad j \neq l, \quad i = 1, \ldots, m \] (5)
Since we want the \( l \)th output of the RBF network, \( y_l = Q(x, a_l, W_Q) \), to be a good approximation of \( Q^*(x, a_l) = y_l^* \), say, we have to minimize the error, \((y_l^* - y_l)^2\). Hence, we can have
\[ \Delta W_{Q}^{k}[l][i] = -\alpha \frac{\partial}{\partial W_{Q}^{k}[l][i]} \frac{1}{2}(y_l^* - y_l)^2 = \alpha (y_l^* - y_l) \phi_l(x) \] (6)
where $\alpha$ is the step size for the gradient descent optimization.

Now, to complete the update algorithm for $W_Q[l][i]$, we need an expression for $(y^*_l - y_l)$. Though we do not know $y^*_l$, we can get a quantity that is a good estimate of $(y^*_l - y_l)$.

In the case of finite state space, by virtue of the dynamic programming equation satisfied by optimal Q-values, the following iterative algorithm learns optimal Q-values [13].

$$Q^{k+1}(x, a_l) = Q^k(x, a_l) + \alpha(g(x, a_l, x') + \gamma \max_{a \in A} Q^k(x', a) - Q^k(x, a_l))$$

$$Q^{k+1}(x, a) = Q^k(x, a) \quad \forall (x, a) \neq (x, a_l),$$

(7)

We can view this algorithm as an error correcting algorithm. We can think of $Q^k(x, a_l)$ as a direct estimate of Q-values (at $k^{th}$ iteration), while $(g(x, a_l, x') + \gamma \max_{a \in A} Q^k(x', a))$ can be thought of as a one step look ahead estimate of the same Q-value based on the current observation.

In our case with continuous state space, the estimate of optimal Q-value at $k^{th}$ iteration for any state action pair, $(x, a_j)$ is given by the $j^{th}$ output of our RBF network with input $x$ and weights $W^k_Q$. (We denote the $j^{th}$ output by $y_j = Q(x, a_j, W^k_Q))$. Thus, we can think of the expression $g(x, a_l, x') + \gamma \max_{a \in A} Q(x', a, W^k_Q) - Q(x, a_l, W^k_Q)$ as a good (current) estimate of the error $(y^*_l - y_l)$. Thus our final algorithm for updating $W_Q[j][i]$ (after seeing the example $(x, x', a_l, g(x, x', a_l))$) is

$$W^{k+1}_Q[l][i] = W^k_Q[l][i]$$

$$+ \alpha(g(x, a_l, x') + \gamma \max_{a \in A} Q(x', a, W^k_Q) - Q(x, a_l, W^k_Q))\phi_i(x)$$

$$i \in \{1, \ldots, m\}$$

$$W^{k+1}_Q[j][i] = W^k_Q[j][i] \quad j \in \{1, \ldots, p\}, \ j \neq l, \ i \in \{1, \ldots, m\}$$

(8)

This is the algorithm we use for training our RBF network using the sequence of examples. (It may be noted that in (8), $Q(x, a_l, W^k_Q)$ refers to the $l^{th}$ output of the RBF network with input $x$ and weights $W^k_Q$). The idea of using the difference between the current estimate of Q-value and the one step look ahead value computed using the immediate reinforcement, as an error signal, has a long history in Reinforcement Learning [13], and such an idea was first used in the mid fifties in the famous program for learning to play checkers developed by Samuel [20].
3.3 The Adaptive Exploration policy

To train the RBF network using (8), we need a sequence of training samples. We use a stochastic exploration policy for this and our exploration policy is also updated after each example so that as learning proceeds, the exploration policy becomes the optimal policy (that is, one based on optimal Q-values). The exploration policy is to be specified as a probability distribution over $A$ for each $x$. Consider a set of parametrized functions $P(x, a, W_P)$ such that

$$
P(x, a_i, W_P) \geq 0, \ \forall x \in X, \forall W_P, \ i = 1, \ldots, p;
$$

$$
\sum_{i=1}^{p} P(x, a_i, W_P) = 1, \ \forall x \in X, \forall W_P, \ i = 1, \ldots, p,
$$

(9)

where $W_P$ denotes the parameter vector of appropriate dimension. Now, for any value of the parameter vector $W_P$, we can use $P(x, a, W_P)$ as the probability with which action $a$ is chosen in state $x$ under the exploration policy. In deciding on the form for the stochastic exploration policy and for the algorithm to update this exploration policy, the main motivation is that we make use of our current (learnt) knowledge of the system as represented by our current estimate of Q-values and, at the same time, keep exploring the state space sufficiently well so as to aid the learning of optimal control law. Let $Q^k(x, a, W_Q^k)$ denote the current estimated Q-values (which would be the outputs of our RBF network with current weight values). Let

$$
Q^k(x, a^k_{gr(x)}, W_Q^k) = \max_a Q^k(x, a, W_Q^k).
$$

That is, action $a^k_{gr(x)}$ is the best action in state $x$ based on our estimated Q-values at time $k$. Then we call $a^k_{gr(x)}$ the greedy action in state $x$ at $k$. We want to update the exploration policy (after each example) in such a way that the probability of choosing the greedy action in state $x$ is increased (by a small amount) at $k$ if $x$ is the state at $k$. As learning proceeds, our estimates of Q-values converge and hence the optimal action in any state also becomes the greedy action. Thus, the exploration policy would finally evolve towards the optimal policy. There is one other point to be noted while deciding on the updating strategy for exploration policy. Since the state space is continuous, the updating of $P(x, a, W_P)$ should be such that similar changes in probabilities of different actions is effected for all states that are sufficiently close to the current state while probabilities of actions in states far away from the current state are not affected. This way, we can ensure proper learning of correct control actions even for states which are not experienced during training.
With the above considerations, we use the following exploration policy. The parameterized class \( P(\cdot, \cdot, W_P) \) of probability distributions is defined as

\[
P(x, a_i, W_P) = F(S(x, a_i, W_P)) \quad i = 1, \ldots, p,
\]

where \( F \) and \( S \) are functions as defined below. (Recall that \( p \) is the cardinality of the action set).

\[
S(x, a_i, W_P) = \sum_{j=1}^{m} W_P[i][j] \exp \left( -\frac{||x - c_j||^2}{2(\sigma_j)^2} \right), \quad i = 1, \ldots, p,
\]

where \( c_j, \sigma_j \), \( j = 1, \ldots, m \) are some parameters and

\[
F(S_i) = \frac{1/(1 + \exp(-S_i))}{\sum_{j=1}^{p} 1/(1 + \exp(-S_j))}, \quad i = 1, \ldots, p,
\]

where \( S_i = S(x, a_i, W_P) \). It is easy to see that \( P(x, a_i, W_P) = F(S(x, a_i, W_P)) \) will satisfy (9).

It may be noted that \( S(\cdot, \cdot, W_P) \) is a real valued function defined over state-action pairs. Thus, for any \( x \), \( (S(x, a_1, W_P), \ldots, S(x, a_p, W_P)) \) is a vector from \( \mathbb{R}^p \) and any such vector is converted by the function \( F \) into a probability vector representing a probability mass function over the finite set \( \mathcal{A} \). Hence, we can view \( S(x, a, W_P) \) as another representation for a probability distribution over the action set. (Such functions are called probability generating function in Learning Automata literature [18]). From (11), the functional form of \( S \) is that of a radial basis function neural network with \( m \) hidden nodes. This means we are essentially using a RBF network for representing our exploration policy also. Hence we take \( c^i \) and \( \sigma^i \) and \( m \) to be same as the corresponding quantities in the RBF that represents our Q-values. (That is the reason we have used the same symbols for them).

Given the above, we can tune the exploration policy, as learning of Q-values proceeds, by updating the parameters (or weights) \( W_P \) at each iteration. This updating is done as follows. Let \( W_P^k \) denote the parameter values at iteration \( k \). Let \( x_k \) be state at \( k \) and let \( a_{gr(x)} \) be the greedy action in the current state. We want to increase \( P(x(k), a_{gr(x)}, W_P) \) and decrease the probabilities of other actions. This is done if we increase \( W_P[gr(x)][i] \), and decrease \( W_P[i][j] \); \( i \neq gr(x) \). Since we want similar changes in probabilities around the current state \( x_k \), we modulate the updating of weights \( W_P[i][j] \) by \( \phi_j(x) = \exp\left(-\frac{||x - c_j||^2}{(2\sigma_j)^2}\right) \). We use the following algorithm to update the weights \( W_P[i][j], i = 1, \ldots, p. \)
The above algorithm always projects $W[k+1][i][j]$ into the interval $[-M, M]$ (for suitable large constant $M$). This is to ensure that the values of the function $S$ are in reasonable range so as to avoid numerical problems while calculating the probabilities of actions using (12). The form of this updating is derived from generalized learning automata algorithms [18]. The above algorithm can thus be viewed as a generalization of our earlier pursuit algorithm based exploration policy [9] to the case of continuous state spaces by utilizing the formalism of generalized learning automata [18]. This novel exploration policy should prove useful in many other continuous reinforcement learning problems. We would be exploring such issues including theoretical properties of the algorithm in our future work.

### 3.4 Summary of RLRB1

We can now summarize RLRB1, which uses the continuous value of ACE as its only state variable.

In RLRB1, we represent $Q$-values in a Gaussian RBF network whose structure is given in Fig. 2. In this RBF, the centers ($c^j$) and widths ($\sigma^j$) of the hidden nodes are fixed as explained in Section 3.2. The weights connecting hidden nodes to output nodes are learnt using the reinforcement learning algorithm given by (8). For this, we need a sequence of examples in the form $(x_k, a_k, x_{k+1}, g(x_k, x_{k+1}, a_k))$. These examples are generated using a simulation model of the system. For this, we start the system in some quiescent state, disturb it using some random load functions. The AGC acts at discrete time instants. So, using the current AGC action we integrate the system model for the next time interval corresponding to AGC cycle time and then obtain the next state (which is the averaged value of ACE over this time interval). During the training process, we use our exploration policy to choose actions. We also update the exploration policy at each iteration.
Ahamed T.P. et al.: A Neural Network Based AGC Design through Reinforcement Learning
as explained in Section 3.3. This full procedure is presented as an algorithm in Table 1.

### Learning Algorithm for RLRB1

Initialize elements of $W_Q$ and $W_P$ to random values.
Initialize $k = 0$.
Initialize $T_{AGC} = AGC$ decision time.

**Repeat**

**Begin**

Start the system in some quiescent state, set $x_0$ to this state.

**while** (states are within safe limits)

**Do**

Choose action $a_k$ based on the current distribution $P(x_k,.,W_k^P)$.

Apply $a_k$ to the plant. Read the current load disturbance from the load model and integrate the plant model for the next $T_{AGC}$ seconds.

From the observed values of ACE during these $T_{AGC}$ seconds obtain the average ACE. Calculate $x_{k+1}$ by normalizing the averaged value of ACE.

Calculate $g(x_k, x_{k+1}, a_k)$.

Update $W_Q^k$ to $W_Q^{k+1}$ using equation (8).

Obtain $a_{gr}(x)$ (greedy action) w.r.t $Q(x, ., W_Q^{k+1})$.

Update $W_P^k$ to $W_P^{k+1}$ using equation (13).

$k = k + 1$.

**end Do**

**end**

---

**Table 1: Learning Algorithm for RLRB1**

To complete the specification of the algorithm we need to specify the
The immediate reinforcement should essentially capture the design objective for the controller. Thus, the quantity, \( g(x_k, a_k, x_{k+1}) \) should give some idea of how good is the action \( a_k \) in state \( x_k \). The attractiveness of RL approach is that it is enough if we can provide some rough assessment of the action taken. Hence we use a very simple \( g \) function as given below.

\[
g(x_k, a_k, x_{k+1}) = \begin{cases} 0 & \text{if } |x_{k+1}| \leq \epsilon_{ACE} \\ -1 & \text{otherwise} \end{cases} \quad (14)
\]

In RLRB1, the state at \( k \), \( x_k \) is (averaged) value of ACE. Our \( g \) function has value 0 if ACE is close enough to zero and is \(-1\) otherwise. The threshold \( \epsilon_{ACE} \) is another parameter of the algorithm.

Finally, we have to decide on a stopping criterion for the algorithm. Firstly, we note that theoretically, there is no need for a stopping criterion. As the algorithm progresses, the estimated Q-values as given by the RBF network outputs would converge to their true values.\(^1\) If the estimated Q-values are close to their true values, then the greedy action in each state would be same as the optimal action and hence, due to the learning algorithm for tuning \( W_P \), our exploration policy would closely approximate the optimal policy. Thus we can keep using only the exploration policy, and as learning progresses this becomes the optimal policy.

In practice, we can stop learning after some time and from then onwards we can use the optimal policy obtained from our estimated Q-values. We can ascertain convergence through two different methods. If the changes made to \( W_Q^k \) are very small over many iterations then we can say the Q-value estimates have converged. Or we can run the algorithm till, under the exploration policy, for each state there is one action whose probability is close to one. We can stop learning by either of these criteria and thus obtain the final AGC.

\(^1\)For this we need to ensure that all relevant state-action pairs are encountered sufficiently often during training. This would be so if during training we use good representative load disturbances for generating training examples. This ensures that all relevant regions of the state space are encountered. In the exploration policy, initially we start with a uniform distribution and use a small value for the learning parameter \( \lambda \) while updating \( W_P \). This would ensure that different actions are chosen enough number of times in each state.
In the previous section, we developed the RL-RBF-AGC with ACE as the single state variable (RLRB1). In this section, we present an RL-RBF-AGC with two state variables.

As in the case of RLRB1, we store the Q-values in an RBF network. The state variables of RLRB2 are normalized values of average frequency deviation, denoted by $\Delta f$ and the normalized value of average deviation in tie line flow, denoted by $\Delta P_{tie}$. An interesting feature of this choice of state variables is that now we do not need to assume the knowledge of $\beta_{area}$, the natural response of the area. The input to AGC is now a vector $x = (x^1, x^2) = (\Delta f, \Delta P_{tie})$. The RBF network used here has two input units, $m$ hidden units and $p$ output units. The architecture of this RBF network is same as that given in Fig. 2 except that there will be two input nodes instead of one. The $j^{th}$ output of this RBF network is given by

$$y_j = Q(x, a_j, W_Q) = \sum_{i=1}^{m} W_Q[j][i] \exp\left(\frac{-2}{\sigma_i^2} \sum_{l=1}^{2} (x^l - c_i^l)^2 \right)$$

$$j = 1, \ldots, p. \quad (15)$$

It may be noted that in cases where one needs more than 2 state variables, we can easily extend the RBF network defined by the above equation. The only change in the equation for such cases will be the upper limit of the summing index $l$.

As in the case of RLRB1, here also we fix the centers and widths of the hidden nodes in the RBF heuristically. We normalize $\Delta f$ and $\Delta P_{tie}$ such that $x \in [-1, 1] \times [-1, 1]$. This is achieved by using bounds $L_f$ and $L_t$ on $\Delta f$ and $\Delta P_{tie}$, in the same way that we use the bound $L_{ACE}$ in normalizing ACE for RLRB1. Since $x \in [-1, 1] \times [-1, 1]$, we place the centers on a uniform grid with $m_f$ columns and $m_t$ rows within the square $[-1, 1] \times [-1, 1]$. This also fixes $\sigma_1 = \frac{1}{m_f-1}, \sigma_2 = \frac{1}{m_t-1}, \forall i$.

Next step is to obtain $W_Q^{*}$, the optimal values of weights in the RBF network. The procedure for finding $W_Q^{*}$ is essentially the same as the procedure explained in section 3.2. Similar to RLRB1, the Gaussian RBF network of RLRB2 has $m$ hidden units and $p$ output units. We use (8) to update $W_Q$’s.

The exploration policy used in RLRB2 is also same as that in RLRB1. Since there are two components to the state vector, the function $S$ would now use a two dimensional Gaussian function. Once again the centers and
To complete the description of RLRB2, we have to specify the $g$ function to be used. Once again we use a simple binary function as given below.

$$g(x_k, a_k, x_{k+1}) = 0 \text{ if } |x_{k+1}^1| \leq \epsilon_f \text{ and } |x_{k+1}^2| \leq \epsilon_{\text{tie}}$$
$$= -1 \text{ otherwise} \quad (16)$$

The thresholds, $\epsilon_f$ and $\epsilon_{\text{tie}}$ are parameters to be chosen by the designer.

## 5 Simulation Results

The steps in the design of RLRB1 and RLRB2 are illustrated by considering a two area test system. Performance of RLRB1 and RLRB2 are provided by considering different load disturbances. The performance of the new controllers are compared with that of a conventional integral type controller for a step disturbance. We also highlight the special features of the proposed RLRB type controller with the RL controller proposed in [9].

### 5.1 Test System

<table>
<thead>
<tr>
<th>$T_g$</th>
<th>$R$</th>
<th>$T_l$</th>
<th>$K_p$</th>
<th>$T_p$</th>
<th>$T_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.08s</td>
<td>2.4 Hz/pu</td>
<td>.3s</td>
<td>120 Hz/pu</td>
<td>20s</td>
<td>0.545</td>
</tr>
</tbody>
</table>

Table 2: Parameters of the two area Power System model

The test system used for studying the performance of RLRB1 and RLRB2 is the widely used model of a two area system. This model represents the relevant dynamics of a system having simple drum type of thermal turbine/generator units for AGC studies. A block schematic diagram of the model used for our simulation studies is given in Figure 3. The parameters of this model given in Table 2 are taken from [21].
5.2 RL-RB1

5.2.1 Controller Design

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\alpha$</th>
<th>$\lambda$</th>
<th>$\gamma$</th>
<th>$L_{ACE}$</th>
<th>$\epsilon_{ACE}$</th>
<th>$m$</th>
<th>$U_{MAX}$</th>
<th>$P_{min}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.01</td>
<td>0.01</td>
<td>0.9</td>
<td>0.02</td>
<td>0.002</td>
<td>11</td>
<td>0.02</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Table 3: Parameters of RLRB1

Controller design involves several design choices. The choice of values for $L_{ACE}$, $\epsilon_{ACE}$, $U_{MAX}$, and $P_{min}$ is dictated by the system constraints, whereas the choice of the values for $m$, $\alpha$, $\lambda$, and $\gamma$ are dictated by the considerations of the learning algorithm.

This controller uses only one variable to represent the state of the system, namely, the ACE. While implementing the controller one must decide the scheme of obtaining this quantity. One possible choice in an actual system is to take the value of ACE as the average of the measured values of the
ACE over the AGC decision cycle. (The number of available measurements depends on the SCADA). In our simulations here an AGC cycle time of 2 seconds has been adopted. The system variables are tracked by integrating the model using a time step of 0.05 seconds. Therefore the ACE value over an AGC period is obtained as the average of the 40 values of ACE computed over 2 seconds. For computing the value of the state \(x\), the input to the RBF network at each AGC decision cycle, the average value of ACE is normalized using \(L_{ACE}\), which is the largest value of ACE for which the controller must respond. Here, \(L_{ACE}\) is chosen as 0.02\(^2\).

\(\epsilon_{ACE}\) is the band around zero within which the controller tries to keep the ACE. A non-zero value is chosen for \(\epsilon_{ACE}\) since AGC is not intended to track small random changes in ACE. Here, \(\epsilon_{ACE}\) is chosen as 0.002.

The parameters, \(U_{MAX}\) and \(P_{min}\), the maximum and minimum control effort possible within an AGC decision cycle, are dictated by the system constraints. Here we chose 0.02 and 0.002 for these values respectively and adopt uniform quantization to get an action set \(A\) with 21 actions.

In order to learn the RBF controller, we first have to decide the structure of the RBF network. Since the input and output have been chosen the only other parameter required to decide the structure is \(m\), the number of hidden nodes. A large \(m\) could slow down the learning while a small value of \(m\) may degrade the performance. Hence some judgment is called for in the choice of \(m\). Here we choose a value of 11 for \(m\). Since we have decided to fix the centers of radial basis functions on a uniform grid, fixing \(m\) fixes all hidden nodes. We note here that it is not necessary (in general) to prefix the (number and) centers of the radial basis functions. Methods such as [22] could be adopted to simultaneously learn the centers along with the weights. These have not been explored here because the performance of the controller designed using our simple strategy of fixing the centers, is very much satisfactory.

In order to complete the controller design we must learn the weights of the RBF network. The proposed learning algorithm requires choices for \(\alpha\) (step-size), \(\gamma\) (discount factor) used in the learning algorithm for updating the weights \(W_Q\) (cf. eqn. (8)) and \(\lambda\) (step-size) used in the learning algorithm for updating \(W_P\) (cf. eqn. (13)). In general we need small values for the learning step-sizes, \(\alpha\) and \(\lambda\). We normally choose \(\gamma\), the discount factor, close to unity because the effect of a control action is felt in the system for

\(^2\)all quantities representing ACE, frequency deviation, \(P_{tie}\), and \(\Delta P\) are in pu and are not explicitly indicated
considerable duration.

The values for all these parameters which we have used in the implementation of RLRB1 discussed here are tabulated in Table 3. (It may be noted that all power system related quantities are expressed in per-unit).

5.2.2 Controller Performance

![Figure 4: Step response of the system with RLRB1](image)

We illustrate the performance of the system by showing the response of the controller and the system to various kinds of load disturbances. All these results are obtained by using the learnt AGC in the power system simulation model. As the two areas are identical, in both areas we used the same AGC. In all the graphs shown here we follow the following notation. The load disturbance applied in area-A is denoted by $P_{lA}$ while $ACE_A$, $u_A$ and $P_{cA}$ denote the ACE as measured in area-A, the output or action of the AGC
Figure 5: Response of the system with RLRB1 for a load disturbance in area A

in area-A and the set point of governor in area-A, respectively. Similarly, \( P_{lB}, ACE_{B}, u_B, P_{cB} \) denote the corresponding quantities for area-B.

We first present the performance of the controller for a step change in load in only area A of the study system. Figure 4 gives the plots of system variables of interest consequent to this disturbance. From this figure, we see that in response to a step load change (\( P_{lA} \)) of 0.02 in area A, the controller changes the set point of the governor (\( P_{cA} \)) in the next AGC cycle, selecting the maximum possible action of 0.02. The mechanical power output (\( P_{mA} \)) also increases by a value of 0.02 in about 6 seconds. However, the variation in \( PmA \) is not instantaneous, and is dictated by the dynamics of the system. Moreover, because of the frequency variation (see plot of \( \Delta f_A \)), oscillations

\[^3\text{It may be noted that the current value of } P_{cA} \text{ is the previous value of } P_{cA} + \text{ the current action of the RLRB1 (uA).}\]
Figure 6: Response of the system with RLRB1 for a load disturbance in both area A and area B

are also seen in $PmA$. As $PmA$ increases, the deviation in frequency ($\Delta fA$) and deviation in tie line flow ($PtieA$) also tend to zero. The plot of $PcB$ is a flat line, there by indicating that the controller in area B does not unnecessarily respond for this disturbance in area A. Figure 4 also gives the plots of the $ACE_A$ as obtained through the integration of the system model and its averaged value ($ACE_{avg}$), which is used by the controller.

The response of the system under a sequence of step load disturbances in area A and without any load disturbance in area B is given in figure 5. Fig. 5 a shows the plots of the load disturbance in area A, $PlA$, the variation of the set point to the governor in areas A and B ($PcA$ and $PcB$). Fig. 5 b shows the output or action of the AGC in area-A ($uA$) and the action of the AGC in area-B ($uB$). Fig. 5 c show the plots of averaged value of ACE ($ACE_{avg}$) in each AGC cycle.
From Fig 5 the following features are evident. The RL controller tracks (plot of $PcA$ in Fig 5a) the load change in the area properly. The controller in the area not subjected to disturbances does not respond to the disturbance (Plot of $PcB$ in Fig. 5a is a flat line). From subplot b, we can see that RL controller takes appropriate control action following almost all the disturbances. For example, following a load increase of 0.01 at $t=300s$, the controller takes an action of 0.01 and following a load decrease of 0.02 at $t=500s$, the controller takes an action of -0.02. Some minor deviations from the desired response are also seen. For example, a small over-correction is seen at 100s. For a decrease in load of 0.01, the controller decreases the set point by 0.012. This over correction is recognized and corrected in the next AGC cycle. Similarly, from $t = 200s$ and $t = 300s$ there seems to be a small steady state error. This is because the controller is designed not to respond if $ACE_{avgA} < \epsilon_{ACE}$. Further, since the action set is discretized, when the
load disturbance is not exactly equal to one of the actions (or the sum of a number of actions), a small difference, which is less than $\epsilon_{ACE}$, can be expected between the disturbance and the controller response.

Simulation results under a sequence of load disturbances consisting of ramps and steps acting in area A and another sequence of step load changes in area B are given in Fig. 6. From Fig. 6a, we can see that $PcA$ is following $PlA$ perfectly. For example, $PcA$ follows the decreasing ramp at $t = 100s$ in steps of 0.002 (the value of $Pmin$), and when a sudden load change occurs at $t = 150s$, $PcA$ also changes by the same amount in the next AGC decision cycle. From Fig. 6b, we can see that $PcB$ is following its load disturbance $PlA$ almost perfectly but for a minor over correction at $t = 500s$. We have conducted numerous experiments with different types of load disturbances in area A and area B. Performance is found to be very good in all the simulation experiments.
5.2.3 Discussions

In this section we presented results to illustrate the performance of RL based AGC controller under a variety of load disturbances. From the results presented it is clear that the RBF network based RL controller, RLRB1, does a very good job of tracking the load change correctly. For the sake of comparison, we have also obtained the AGC response of the system with an integral type of controller for a step load change in area A. This response is given in Fig. 7. The controller gain is chosen to be 0.2. For this system this value is known[5] to provide a good compromise between speed and overshoot. From Fig. 7 it is seen that the performance of the integral controller is also reasonably good. For this disturbance, the settling time is slightly larger than the RLRB1 controller. This is evident from the the two responses in Fig. 4 and Fig. 7. What is more significant is the fact that the RLRB1 is designed without using any information of the system dynamics (system model is totally unknown), where as the integral controller design requires the complete knowledge about the system model.

As discussed in section 1, the main motivation for using a neural network (trained through reinforcement learning) for representing the Q-values is that we need not worry about proper quantization of state space. We have earlier proposed an RL based AGC that operates under the assumption of
Ahamed T.P. et al.: A Neural Network Based AGC Design through Reinforcement Learning

We present below a comparison of RLC1 with RLRB1.

Design of RLC1 requires the quantization of state space. If the quantization is appropriate then the performance of RLC1 will be as good as that of RLRB1. However, it is not easy to come up with a proper quantization of state space in a systematic way. In order to highlight this aspect we provide in Fig. 8 comparison of the response of RLRB1 and RLC1 to a sequence of load disturbances. For this simulation, we designed RLC1 following the procedure given in [9] with state space uniformly quantized to 11 states and action space being the same as that of RLRB1. A comparison of the response of RLRB1 (Fig. 8a) and response of RLC1 (Fig. 8c) clearly shows that the performance of the RLC1 is not as good as that of RLRB1. In order to understand the reason for this difference in performance of the controllers, we compare the control policies learnt by these two controllers. Since both controllers have only one input and one output the control policies can be represented as an action vs state plot. The learnt policies of RLRB1 and RLC1 for this system are shown in Fig. 9. Both policies have to choose from the same action set $\mathcal{A}$.

Since $\mathcal{A}$ is finite, whether or not the state space is finite, the control policy could be a stair case like function. That is even when we are using continuous state space as in RLRB1, the learnt control policy effectively quantizes the state space. In the case of RLRB1, the quantization is a learnt one. On the other hand for RLC1, the designer has to make a choice for the quantization of the state space. Once this choice is made, only one action is possible within each chosen quantized level. For RLRB1, the designer has to chose the centers for the RBF network. However, the choice of centers does does not directly dictate the final effective quantization of the learnt policy. For the results shown in Fig. 8, the 11 quantization levels chosen in RLC1 are the same as the centers chosen for the RBF network. As can be see from Fig. 9, the policy learnt by RLC1 uses only 9 actions though the action set $\mathcal{A}$ has 21 discrete actions. On the other hand, the policy learnt by RLRB1 is made of 20 actions. It may be possible to learn a policy for RLC1 that uses more discrete actions by finer quantization of the state space. This could increase the computational cost of learning. More importantly, even with a finer quantization, the policy learnt by RLC1 may not replicate that of RLRB1 because for RLC1 we have to specify the state quantization a priori. As can be seen from Fig. 9a, even when the RBF centers are uniformly spaced, the effective quantization of state space as reflected in the learnt policy is non-uniform.
Thus, using RBF neural network for representation of Q-values results in a learnt controller being able to automatically decide on the different points in the state space where the same action is appropriate. Another point that we want to emphasize here is that the choice of RBF network centers is not very crucial.

Figure 10: Control laws of RLRB1 with 13 centers

To illustrate this point, we show in Fig. 10 the control law learnt for the same system with 13 centers rather than 11. Comparing the policy depicted in Fig. 10 with that of the RLRB1 policy in Fig. 9, we see that they are almost the same. An exact match cannot be expected since these policies are learnt through a stochastic learning algorithm. Hence, even for the same set of design parameters, the policies learnt using different sets of training sample sequence would be different.

5.3 RLRB2

In this section we briefly present the performance of RLRB2, an RL-RBF-AGC with two state variables. These results are intended to show that having more than one state variable does not complicate the design process. Depending on the problem and availability of measurements, one can design RL-RBF-AGC with any number of state variables.

5.3.1 Controller Design

The states chosen for RLRB2 are based on the two measurements that are normally available at the control center, and the control objective is so defined so as to avoid calculation of the ACE which requires the value of $\beta_{\text{area}}$, a system specific information that is not easy to get. The control objective for this controller also is to keep $\Delta f$ and $\Delta P_{\text{tie}}$ within acceptable
Figure 11: Response of the system with RLRB2 with LD2 in area A and LD1 in area B.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$L_f$</th>
<th>$\epsilon_f$</th>
<th>$L_t$</th>
<th>$\epsilon_t$</th>
<th>$m_f$</th>
<th>$m_t$</th>
<th>$U_{MAX}$</th>
<th>$P_{min}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.02</td>
<td>0.002</td>
<td>0.01</td>
<td>0.001</td>
<td>11</td>
<td>11</td>
<td>0.02</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Table 4: Parameters of RLRB2

bounds without calculating the ACE based on $\beta_{area}$. The state variables $(x^1, x^2)$ used are derived from $\Delta f$ and $\Delta P_{tie}$. They are obtained here as follows.

At the beginning of the $k^{th}$ AGC decision cycle we have the previous values (over the last 2 seconds) of $\Delta f$. From these values the average value at the $k^{th}$ decision cycle, $\Delta f_{avg_k}$, is calculated. For computing the value of $x^1$, $\Delta f_{avg_k}$, is normalized with respect to $L_f$. $L_f$ is chosen as the largest
value of $\Delta f$ for which the AGC must respond. Here value of $L_f$ is chosen as 0.02. Similarly $x^2$ is obtained by normalizing $\Delta P_{tie}$ with respect to $L_t$. Here $L_t$ is chosen as 0.01.

$\epsilon_f$ is the band around zero within which the controller seeks to keep the value of $\Delta f$, and $\epsilon_t$ is a similar band within which the controller seeks to keep the value of $\Delta P_{tie}$. In this implementation we have chosen $\epsilon_f = 0.002$ and $\epsilon_t = 0.001$. The action set $\mathcal{A}$ of RLRB2 is chosen to be the same as that of RLRB1.

As mention in section 4, centers of RLRB2 are placed on a uniform grid with $m_f$ columns and $m_t$ rows within the square $[-1,1] \times [-1,1]$. Some judgment is called for in the choice of $m_f$ and $m_t$. As a thumb rule, one may use the closest integer to $L_f/\epsilon_f$ and $L_t/\epsilon_t$ for $m_f$ and $m_t$ respectively. We choose $m_f = m_t = 11$.

We use the same values for the learning parameters $\alpha, \beta,$ and $\gamma$ as in RLRB1. The value of all these parameters are tabulated in Table 4.

### 5.3.2 Controller Performance

The response with different sequence of disturbance acting in area A and B respectively is given in Fig. 11 with a RLRB2 type controller in each area. From the figure we can see that the controllers in areas A and B track the load disturbance in their own area satisfactorily. However, there are few overshoots. For example, there are overshoots at $t=150s$ and $t=550s$ in the plot of $P_{cA}$ in the subplot-a of the figure. From the subplot-b we can see that area B responds at $t=150s$, even though there is no load change at $t=150s$. However, $P_{cB}$ corrects immediately. From subplots c and d, we can see that there is small steady state error in $f$ and $P_{tie}$. This is because the objective is only to bring back $f$ within $\epsilon_f$ and $P_{tie}$ within $\epsilon_t$.

We have conducted numerous simulation experiments with different load disturbances in both areas. The performance was good in all the cases.

### 6 Conclusions

In this paper we have considered application of reinforcement learning techniques to an important control problem in Power Systems, namely the AGC problem. From the application point of view, the RL approach is attractive because it does not need any knowledge of the system model. Moreover, we can easily incorporate constraints such as the bound on the maximum change of generation possible in an AGC decision cycle, and also allow a
Ahamed T.P. et al.: A Neural Network Based AGC Design through Reinforcement Learning

A lot of flexibility in specifying control objective through the immediate reinforcement function. The proposed method essentially learns the optimal control law through a sequence of training examples under the Q-learning framework. We have earlier demonstrated the feasibility of this approach by quantizing any power system variable that has to be input to AGC. In this paper we have considered generalizing this by putting together an RL method that works with continuous state spaces. We have shown how RBF networks can be used to learn and store Q-values for such applications. Since supervised training of RBF networks is infeasible for such applications, we have proposed a reinforcement learning algorithm to train the RBF network. Another significant contribution of this approach is a novel exploration policy which is also adapted as the learning of Q-values proceeds. This novel exploration policy could be useful, in general, for Reinforcement Learning of controllers with continuous state spaces. This algorithm is motivated by the schemes used in learning automata literature [18]. It is seen to be very effective in the application considered here. We plan to investigate the theoretical properties of this algorithm in our future work.

The RL based design procedure for AGC has been illustrated by presenting two controllers: One which uses ACE as the only input, and another which uses $\Delta f$ and $\Delta P_{tie}$ as the inputs. The later is attractive from the application point of view because it does not need $\beta_{area}$ used in the computation of ACE. Simulation results of the performance of the controllers have been presented. The results show that the controller perform very well and compare well with the performance of a integral controller designed with full knowledge of the system model. We have also compared the performance of the controller presented here with our earlier controller that needs proper quantization of the state space [9]. It is shown that the continuous controller performs better than the discrete controller. In our future work we will investigate the effect of system non-linearities like GRC and dead-band on the performance of RL-RBF-AGC and also explore RL-RBF-AGC for a multi-area power system.

References


Ahamed T.P. et al.: A Neural Network Based AGC Design through Reinforcement Learning


