PID Tuning with Neural Networks

Chapter · March 2019

DOI: 10.1007/978-3-030-14799-0_41

Antonio Marino
Università degli Studi di Genova
1 PUBLICATION **4** CITATIONS

PID Tuning with Neural Networks

Antonio Marino and Filippo Neri(✉)

Department of Electrical Engineering and Information Technologies, University of Naples, Naples, Italy
marinoantonio.96@gmail.com, filippo.neri.email@gmail.com

Abstract. In this work we will report our initial investigation of how a neural network architecture could become an efficient tool to model Proportional-Integral-Derivative controller (PID controller). It is well known that neural networks are excellent function approximators, we will then be investigating if a recursive neural networks could be suitable to model and tune PID controllers thus could assist in determining the controller's proportional, integral, and the derivative gains. A preliminary evaluation is reported.

Keywords: PID tuning and approximation · Machine learning · Neural networks

1 Introduction

Industrial plants are characterized by a variety of processes [1] that require the design of a controller able to complete the control task and respecting the required performances. In this research activity, we will study if some neural network architectures might be an efficient tool to model Proportional- Integral-Derivative controller (PID controller) [2]. It is well known that neural networks are excellent function approximators we are then interested in studying if some of their topologies could be suitable to model PID controllers. A variety of approaches to model or tune PID controllers exist: some works explore architectures for neural networks, other uses fuzzy logic, other evolutionary computation [3–10]. This paper reports our initial investigation in this area. To begin with we will empirically investigate recursive multi layer perceptrons in order to synthesize a PID controller and its gains by determining the proportional, integral, and derivative actions. In the paper we will report the performances of a recursive neural network for tuning PID controller on testcase system developed in Symulink and we will compare the architecture's performances with the methods by Zigler and Nichols (ZN) [2] and Extremum Seeking (ES) [11]. The paper reports a preliminary empirical investigation of the novel methodology. Given the great variety of Artificial Intelligence techniques that have been applied in several context, including financial time series [12,13], one has to wonder if such techniques could be leveraged to also model PID controllers.

N. T. Nguyen et al. (Eds.): ACIIDS 2019, LNAI 11431, pp. 476–487, 2019.
https://doi.org/10.1007/978-3-030-14799-0_41

2 The PID Controller

The PID controller is usually the best choice when designing a control system because of its simple implementation and design. Indeed its standard architecture is usually the best choice because it consists of a linear combination of three components: the proportional, the integral and the derivative actions as in

$$u = K_p e_p + K_i e_i + K_d e_d \tag{1}$$

where u is the control output and e_{index} are the error components (actions) which are defined as follows:

$$e_p = e, e_i = \int_0^t e dt, e_d = \frac{de}{dt} \tag{2}$$

where e is the input error.

The great diffusion of the PID controller is due to its low cost (with respect to the cost of a microprocessor) and also is due to the possibility of utilization without the exact knowledge of the industrial process to be controlled. This last characteristic is not to be undervalued considering the great variety of industrial processes. Possibly controllers designed ad hoc for a specific industrial process, maybe even including a model for it or even its simulation, could result in a better controller than a PID but would require a significant increase in the design and realization costs.

The components of Eq. (1) are the essential actions for a linear time invariant controller which allows for the proportional action, an appropriate gain of reference signal, for the integral action, an overtime error converging to zero, and for the derivative action, a correction speed which adjust over time. It is not necessary for all the actions to occur at the same time. In fact by setting to zero a K coefficient a specific action gain can be canceled. In reality, it is in fact common to use just P and PI controllers.

The techniques by Ziegler e Nichols (ZN) [2] are some of the classic techniques for tuning PID controllers. Of heuristic nature, they are based on a preliminary evaluation of the plant for determining the initial values of the controller. The first technique uses an approximation of the plant to a system of the first order with an apparent delay to calculate the controller parameters. The second technique is concerned with different plant typologies of higher order on which a closed loop test with a purely proportional gain is executed. This leads to extablish the proportional gain that brings the system on the bound of its stability. Then this initial value will help to obtain the optimized target gains.

The ZN's techniques are usually suitable for many kind of systems, producing however a relatively ready response with overshooting and sustained oscillations since they have been designed to obtain relatively low dampings.

Another classic technique for PID tuning is Extremum Seeking (ES) [14] which is a widely used technique used for real time tuning. It does not need any knowledge about the system or about the error function. However it assumes that the error function respects the preconditions of the gradient theorem since ES uses the minimization of the gradient to compute the three PID gains. The ES can be realized in hardware with a sequence of filter, modulator and integrator pointing out the trend of the error and an estimation of the parameters. Both approaches, ZN and ES, are largely overtaken by more sophisticated techniques based on system identification or by optimization techniques, still they remain important methods to benchmark the performance of new PID tuning methods.

3 Neural Networks

Under the supervised learning paradigm, an artificial agent received a set of labeled instances formed by couples <inputs, output> where output is the classification label to be predicted. The output values are produced by an unknown control function f. The agent has as objective the learning of the function f which, according to the machine learning terminology, is called hypothesis by searching an hypothesis space for a function f' where at least a good approximation of f is supposed to exist. The quality or performance of the discovered f' is then evalutated on a separate set of instances called test set not occurring in the set of instances used for finding f'. As stated above, the selection of the hypothesis space determines the possibility to find a solution f' which is consistent (explains) the training set. An acceptable solution f' may not be perfect as f but will explain the majority of the learning instances and perform well on the test set.

A Neural Network (NN) [15] is a non linear mathematichal structure made of a set of interconnected neurons or nodes. The properties of the NN are determined by its topology and by the features of the nodes. Feed forward NNs can be represented as a set of node levels where we can identify the input level, the subset of nodes receiving the inputs, the output level, the set of nodes producing the NN's outputs, and the intermediary levels, all the remaining nodes. In a feed forward network the NN is traversed only in one direction from the input layer to the output layer, it contains no cycles and thus no internal memory.

Other types of NNs exist: for instance in a recursive NN the outputs are fed again to the input nodes thus determining a memory effect where previous inputs/outputs can also affect the present outputs together with the current inputs. Because, in the scope of this research, we are interested in modeling a PID controller, which is a dynamic model, we will consider recursive NNs.

4 PID Neural Network

In this section, we describe the PID Neural Network (PIDNN) used in our experimentation. The PIDNN uses three special types of neurons: P-neuron, I-neuron and D-neuron which realize the three fundamental controlling actions. The network is built with three level. The input level is made up by two neurons which takes as input the output of the plant and the reference input. The second level is made up by the P-,I-,D-neurons. And the third level combines the three controlling action components in the controlling action to be applied to the plant (Fig. 1).

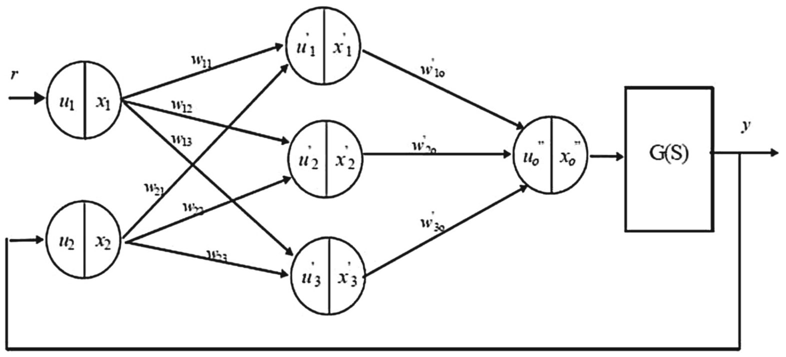

Fig. 1. PIDNN where G(s) is the system to be controlled.

The functions of the neurons are defined as follows:

$$P - neuron\ \ output = \sum_i input_i \tag{3}$$

$$I - neuron\ \ output = \int_0^t \sum_i input_i \tag{4}$$

$$D - neuron\ \ output = \frac{d}{dt} \sum_i input_i \tag{5}$$

The integration and derivative functions can be implemented in different way: we have chosen a numerical derivation (adoptable even in the simplest embedded system) whereas the integration function consists in the integration done till the current time.

These choices has been guided by the results obtained during preliminary experimentation. We notice that the implementation choices impact on the efficiency of the PIDNN because they require different sampling methods of the input signals.

The weights are defined as follows:

$$w_{i,j} = +1, w_{2,j} = -1, w_{1o} = K_P, w_{2o} = K_I, w_{3o} = K_D \tag{6}$$

Going across the network from the input layer to the output layer:

$$x'_1 = u'_1 = w_{11}x_1 + w_{21}x_2 = r - y = e \tag{7}$$

$$x'_2 = \int_0^t u_2 d\tau = \int_0^t e d\tau; with\ \ e = w_{12}x_1 + w_{22}x_2 = r - y \tag{8}$$

$$x'_3 = \frac{du_3}{dt} = \frac{de}{dt}; with\ e = w_{12}x_1 + w_{22}x_2 = r - y \tag{9}$$

$$x''_o = u''_o = \sum_{j=1}^{3} w'_{jo}x_j = w'_{1o}x'_1 + w'_{2o}x'_2 + w'_{3o}x'_3 \tag{10}$$

$$= K_p e + K_i \int_0^t e d\tau + K_d \frac{de}{dt} \tag{11}$$

5 Experimental Setting and Results

The plant model chosen for the experimental evaluation is:

$$P(s) = \frac{(1 - 5s)}{(1 + 10s)(1 + 20s)}$$

This system is a non-minimum phase one with the zero with a positive real part. Moreover it has two small poles, although asymptotically stable, it is really slow. We choose a Linear Time Invariant system to begin with our study although our neural network based methodology should also allow to synthesize controller for non-linear system.

An example of the development workspace in Simulink used to run the experiments is shown in Fig. 2.

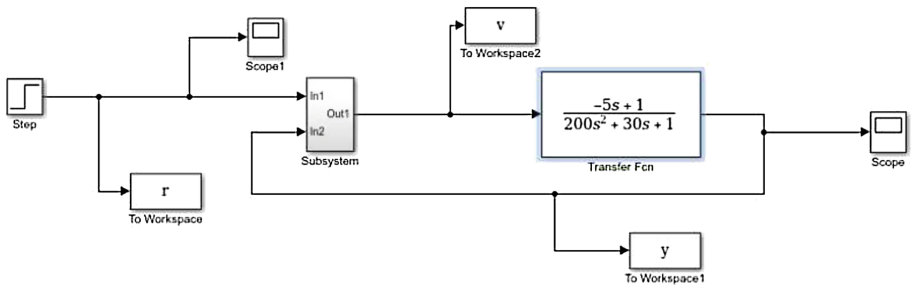

Fig. 2. Simulation workspace

This is a method for off-line auto-tuning the controller based on repeated trials to improve the controller's performances. During the trials the convergence of the controller is not guaranteed hence it would be impossible to perform the trials on a real plant. In Fig. 3, the architecture of the NN based PID used in our experiments is reported. It will be included in the architecture of Fig. 2 in the subsystem box.

In the first set of experiments, the error back-propagation algorithm, written in Matlab, changes the weights of the links initialized with fixed values. The initial weight values come out directly from the second techniques of Ziegler-Nichols which has provides as initial values: $Kp = 3.53$, $Ki = 0.2101$, $Kd =$

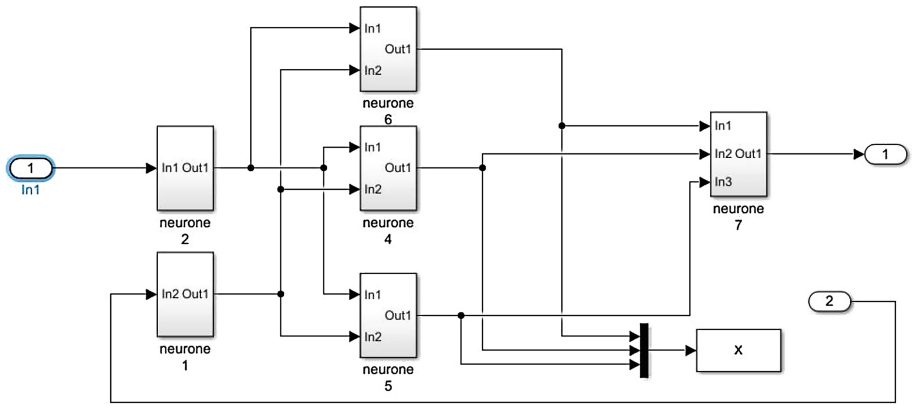

Fig. 3. PIDNN implemented in Simulink. PIDNN is implemented in the subsystem box of Fig. 2.

14.8260 for the weights on the input edges of the output neurons (5, 6, 7) while leaving the initial values of 1 and –1 for the others. The algorithm follows.

```
// Global variables
// m is the current sample of the signals
// $n_{i1}$ learning rate for the first layer
// $n_i$ learning rate for the second layer

//Definitions of the input parameters
//r, the reference input of the plant
//y, the real output of the closed loop system
//v, the control input of the plant
//x, the output vector of the PID neurons


$BackPropagation(r, y, v, x, K)$
    $J = 0$    // vector of upgrades set initially to zero
    $y_1 = y(m+1)$    // the following output
    $y_2 = y(m)$      // the current output
    $v_1 = v(m-1)$    // the previous control input
    $v_2 = v(m)$      // the current control input
    $x1 = x(m)$       // the vector of the output signal at current time of the
                         three neurons
    $x2 = x(m+1)$     // the vector of the output signal at next time of the three
                         neurons
    $\Delta = 2 * (r - y_1) * (y_2 - y_1)/(v_1 - v_2)$
    $\Delta 1 = \Delta * k_p * (x2(1) - x(1))/((r - y_2) - (r - y_1))$
    $\Delta 2 = \Delta * k_i * (x2(2) - x(2))/((r - y_2) - (r - y_1))$
    $\Delta 3 = \Delta * k_d * (x2(3) - x(3))/((r - y_2) - (r - y_1))$
```

```
//Computation of the error vector
For Each i From 0 To m
Do
    $J(1) = J(1) - (\Delta * x1(1,1)/m)$
    $J(2) = J(2) - (\Delta * x1(1,2)/m)$
    $J(3) = J(3) - (\Delta * x1(1,3)/m)$
    $J(4) = J(4) - (\Delta1 * r/m) - (\Delta2 * r/m) - (\Delta3 * r/m)$
    $J(5) = J(5) - (\Delta1 * y_1/m) - (\Delta2 * y_1/m) - (\Delta3 * y_1/m)$
End Do

//update of the gains for the controller
$k_p = k_p - ni * J(1)$
$k_i = k_i - ni * J(2)$
$k_d = k_d - ni * J(3)$
$k_j1(1) = k_j1(1) - ni1 * J(4)$
$k_j2(1) = k_j2(1) - ni1 * J(5)$

$K = [k_p, k_i, k_d, k_j1(1), k_j2(1)]$
$m = m + 1$
$return(K)$
```

The proposed algorithm monitors the output of the system, the output of the controller, and of the individual neurons comparing the result with the reference input. Then the algorithm learns by sampling these signals whose number depends on the sampling rate. The learning rate ni and ni1 are at the core of this procedure because they establish the learning rate of the PIDNN separately for the input and output neurons.

The effect of choosing the values for the learning rates will be explored in a future work, yet we already have observed that a suitable choice can produce a robust controller, or a performant one, and also that out of some ranges the PIDNN would not converge thus producing a controller causing unstability in the system.

During the preliminary experimentation, we also learned that one factor affecting the instability in the system/controller is due to an high sampling rate. It appears in fact that an high sampling rate would produce a signal sampling whose beginning portion bears no revelance with the end of it.

For the experimentation performed, we selected a sampling rate of 0.1 seconds and we utilized the above algorithm in parallel on different portions of the signal curve and we averaged the gains obtained to produce the final ones. The following code show how we proceeded.

```
initialization of the PID gain
$K = [K_p, K_i, K_d, 1, -1]$

acquiring signal from the simulation

m = 0;
for each j in range of 1 and $NUMBER\_OF\_SCAN$
do
    for each i in range of 1 and $NUMBER\_OF\_ITERATIONS/5$
    do

        $m = i$
        $K1 = BackPropagation(r, y, v, x, K(1), K(2), K(3), K(4), K(5))$

        $m = NUMBER\_OF\_ITERATIONS/5 + i$
        $K2 = BackPropagation(r, y, v, x, K(1), K(2), K(3), K(4), K(5))$

        $m = 2 * NUMBER\_OF\_ITERATIONS/5 + i$
        $K3 = BackPropagation(r, y, v, x, K(1), K(2), K(3), K(4), K(5))$

        $m = 3 * NUMBER\_OF\_ITERATIONS/5 + i$
        $K4 = BackPropagation(r, y, v, x, K(1), K(2), K(3), K(4), K(5))$

        $m = 4 * NUMBER\_OF\_ITERATIONS/5 + i$
        $K5 = BackPropagation(r, y, v, x, K(1), K(2), K(3), K(4), K(5))$

        $K(1) = (K1(1) + K2(1) + K3(1) + K4(1) + K5(1))/5$
        $K(2) = (K1(2) + K2(2) + K3(2) + K4(2) + K5(2))/5$
        $K(3) = (K1(3) + K2(3) + K3(3) + K4(3) + K5(3))/5$

        $K(4) = (K1(4) + K2(4) + K3(4) + K4(4) + K5(4))/5$
        $K(5) = (K1(5) + K2(5) + K3(5) + K4(5) + K5(5))/5$

        $k_1(i) = K(1)$
        $k_2(i) = K(2)$
        $k_3(i) = K(3)$
        $k_4(i) = K(4)$
        $k_5(i) = K(5)$

        // starting simulation and acquisition of new singals
    end
    $f = f + 1$
end
```

The repetition of the algorithm on a singular sampling at time proved inefficient, therefore it has been thought to parallelize the action of the algorithm to five samplings of the signal. In that way each piece of the signal gives a greater contribution to the definition of the final gains.

The final result shows significant improvements with respect to the performances observed using the ZN and ES methods with a settling time reduced by 60 and 20 s respectively, Fig. 4.

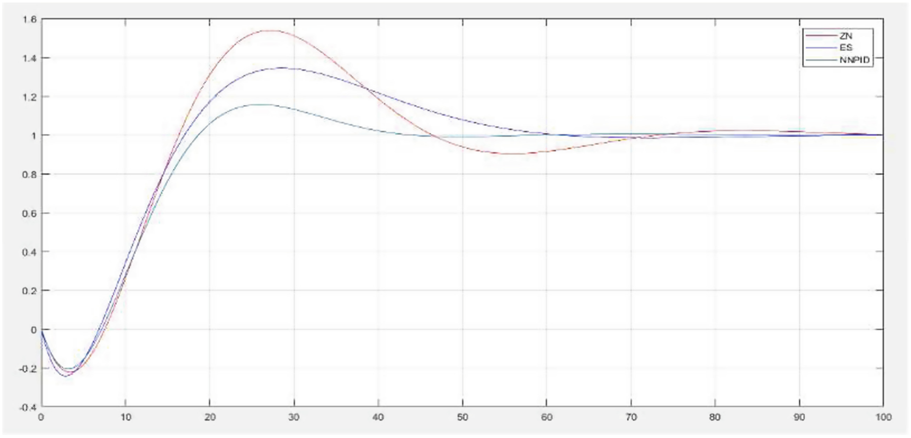

Fig. 4. Compared response for the PIDNN, ZN, and ES techniques

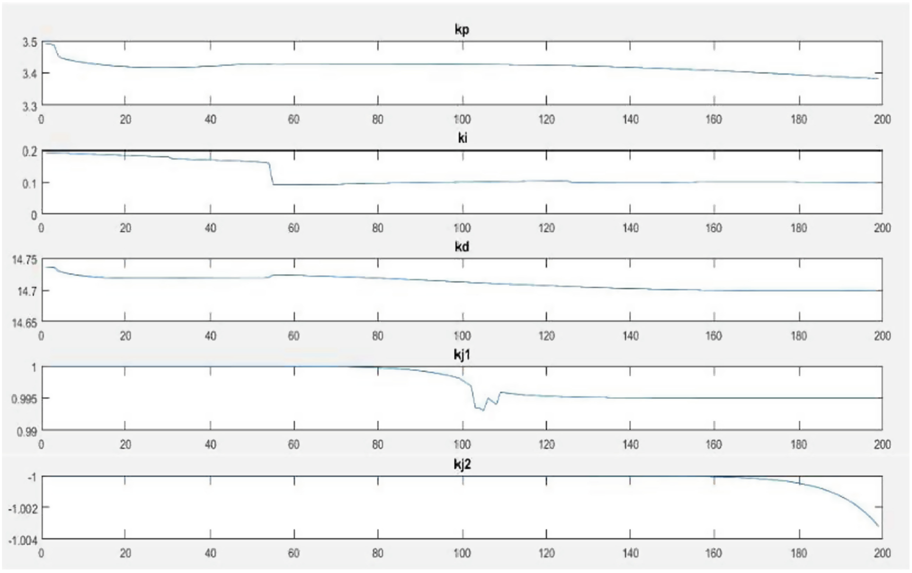

Fig. 5. Learning rate of our PIDNN network

As it's can be seen form the outputs reported, the solution found also improves the undershoot of a few percentage points and, above all, the overshoot that records 30% and 10% less than the first two methods. In order to train the network, 1000 as NUMBER_OF_ITERATIONS was used, after which a widespread overfitting phenomenon was appearing, so it was useless to continue. The learning curves used to evaluate the velocity of convergence for the learning rates are reported in Fig. 5.

Where kp, ki, and kd are the gains for the controller while kj1 and kj2 are the weights of the forward and feeback edges respectively.

6 Conclusions

The goal of this work is to explore a new method PIDNN for the tuning of PID controllers based on neural networks. We investigate both an architectural choice for the NN-based controller and we study its performances in learning the controller's gains in a given setting. The empirical results shows that the PIDNN method is very promising in term of a faster learning of the controller gains with respect to more traditional methods such as ZN and ES. We report in details the algorithms we used as well as we show the behavior of the controller's gains during the learning of PIDNN. As future works we will consider to expand the set of cases where PIDNN could be applied such as the investigation of the modeling of non-linear and Multiple Input - Multiple Output systems. Also we will consider how different levels of noise and disturbances will affect the stability and the convergence of PIDNN in a given amount of time. Another research direction that we will pursue is to understand how alternative artificial intelligence techniques may be used, like agent based modeling, in modeling a plant controller. Agent based modeling has in fact been successfully used to model time series [13,16–22] and then might be used to model the behavior of a PI controller over time. Also in terms of optimization of the PI controller, one may explore how meta-learning and hyper-heuristics [23,24] could be used to improve the neural network learning in PIDNN.

References

1. Huailin Shu, Y.P.: Decoupled temperature control system based on PID neural network. In: ACSE 05 Conference, CICC, Cairo, Egypt, 19–21 December 2005 (2005)
2. Ziegler, J.G., Nichols, N.B.: Optimum settings for automatic controllers. Trans. ASME **64**, 759–768 (1942)
3. Boubertakh, H., Tadjine, M., Glorennec, P.Y., Labiod, S.: Tuning fuzzy PD and PI controllers using reinforcement learning. ISA Trans. **49**, 543–551 (2010)
4. Carlucho, I., Paula, M.D., Villar, S.A., Acosta, G.G.: Incremental Q-learning strategy for adaptive pid control of mobile robots. Expert Syst. Appl. **80**, 183–199 (2017)

5. Zhang, J., Wang, N., Wang, S.: A developed method of tuning PID controllers with fuzzy rules for integrating processes. In: Proceeding of the 2004 American Control Conference Boston, Massachusetts, 30 June -2 July 2 (2004)
6. Kim, J.S., Kim, J.H., Park, J.M., Park, S.M., Choe, W.Y., Heo, H.: Auto tuning PID controller based on improved genetic algorithm for reverse osmosis plant. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng. **2**11 (2008)
7. Salem, A., Hassan, M.A.M., Ammar, M.E.: Tuning PID controllers using artificial intelligence techniques applied to DC-motor and AVR system. Asian J. Eng. Technol. **2**2 (2014). ISSN 2321–2462
8. Muderrisoglu, K., Arisoy, D.O., Ahan, A.O., Akdogan, E.: PID parameters prediction using neural network for a linear quarter car suspension control. Int. J. Intell. Syst. Appl. Eng. (2014)
9. Scott, G.M., Shavlik, J.W., Ray, W.H.: Refining PID controllers using neural networks. National Science Foundation Graduate Fellowship, pp. 555–562 (1994)
10. Shen, J.C.: Fuzzy neural networks for tuning PID controller for plants with underdamped responses. IEEE Trans. Fuzzy Syst. **9**(2), 333–342 (2001)
11. Killingsworth, N., Krstic, M.: PID tuning using extremum seeking: online, model-free performance optimization. IEEE Control Syst. **26**, 70–79 (2006)
12. Papoutsidakis, M., Piromalis, D., Neri, F., Camilleri, M.: Intelligent algorithms based on data processing for modular robotic vehicles control. WSEAS Trans. Syst. **13**, 242–251 (2014)
13. Neri, F.: PIRR: a methodology for distributed network management in mobile networks. WSEAS Trans. Inf. Sci. Appl. **5**, 306–311 (2008)
14. Draper, C., Li, Y.: Principles of optimalizing control systems and an application to the internal combustion engine. Optimal and Selfoptimizing Control (1951)
15. Rumelhart, D.E., Widrow, B., Lehr, M.A.: The basic ideas in neural networks. Commun. ACM **37**, 87–92 (1994)
16. Neri, F.: Learning and predicting financial time series by combining natural computation and agent simulation. In: Di Chio, C., et al. (eds.) EvoApplications 2011. LNCS, vol. 6625, pp. 111–119. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20520-0_12
17. Neri, F.: A comparative study of a financial agent based simulator across learning scenarios. In: Cao, L., Bazzan, A.L.C., Symeonidis, A.L., Gorodetsky, V.I., Weiss, G., Yu, P.S. (eds.) ADMI 2011. LNCS (LNAI), vol. 7103, pp. 86–97. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27609-5_7
18. Staines, A., Neri, F.: A matrix transition oriented net for modeling distributed complex computer and communication systems. WSEAS Trans. Syst. **13**, 12–22 (2014)
19. Neri, F.: Agent-based modeling under partial and full knowledge learning settings to simulate financial markets. AI Commun. **25**, 295–304 (2012)
20. Neri, F.: Case study on modeling the silver and nasdaq financial time series with simulated annealing. In: Rocha, Á., Adeli, H., Reis, L.P., Costanzo, S. (eds.) WorldCIST 2018. AISC, vol. 746, pp. 755–763. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77712-2_71
21. Neri, F.: Combining machine learning and agent based modeling for gold price prediction. In Cagnoni, S. (ed.) WIVACE 2018, Workshop on Artificial Life and Evolutionary Computation, vol. tbd. Springer (2018, in press)
22. Neri, F.: Can agent based models capture the complexity of financial market behavior. In: 42nd Annual Meeting of the AMASES Association for Mathematics Applied to Social and Economic Sciences, Napoli. University of Naples and Parthenope University Press (2018, in press)

23. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.: A classification of hyper-heuristics approaches. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research and Management Science, vol. 146, pp. 449–468. Springer, Heidelberg (2009). https://doi.org/10.1007/978-1-4419-1665-5_15. In press
24. Camilleri, M., Neri, F., Papoutsidakis, M.: An algorithmic approach to parameter selection in machine learning using meta-optimization techniques. WSEAS Trans. Syst. **13**, 203–212 (2014)