Intelligent Feedback Scheduling of Control Tasks

Fatin I. Telchy
Computer Engineering Branch
Control and System Engineering Dep., University of Technology
Baghdad, Iraq
fatin.telchy@yahoo.com

***Abstract**, an efficient feedback scheduling scheme based on the proposed Feed Forward Neural Network (FFNN) scheme is employed to improve the overall control performance while minimizing the overhead of feedback scheduling which exposed using the optimal solutions obtained offline by mathematical optimization methods. The previously described FFNN is employed to adapt online the sampling periods of concurrent control tasks with respect to changes in computing resource availability. The proposed intelligent scheduler will be examined with different optimization algorithms. An inverted pendulum cost function is used in these experiments. Then, simulation of three inverted pendulums as intelligent Real Time System (RTS) is described in details.*

Numerical simulation results demonstrates that the proposed scheme can reduce the computational overhead significantly while delivering almost the same overall control performance as compared to optimal feedback scheduling

***Index Terms*—Feedback Scheduling, Real Time Control Task, Feed Forward Neural Network (*FFNN*), Inverted Pendulum, Optimization Algorithms, Optimal Sampling Period**

I. INTRODUCTION

The feedback scheduler attempts to control the *CPU* utilization to less than or equal to $U_R=\sum C_i/h_i$ (where U_R is *CPU* utilization reference, C_i is the processing time, h_i is the sampling periods) by rescaling the sampling periods of the controllers. The feedback scheduler is implemented as a high-priority task that regularly collects execution-time measurements from the control tasks and estimates the *CPU* load. Based on the load estimate, new sampling periods are communicated to the control tasks.

Generally, online scheduling using optimal feedback scheduling schemes is too computationally demanding in terms of cost and time. An efficient feedback scheduling scheme using Feed Forward Neural Networks will be proposed in this paper, It should be mentioned here the applied assumptions in this research where as follows:

- Hard real time tasks scheduling.
- All hard tasks are periodic, with known periods.
- All release times are zero
- Jobs are ready to run at their release times.
- Deadlines are equal to periods.
- Jobs do not suspend themselves.
- Jobs are independent (i.e., they do not communicate or share other resources than the CPU).
- All systems context switching time and so on are ignored in *JSSP* and *CS*.
- Non preemption concept has been limited the scheduling of the jobs

Several attempts have been dome in this field and below are some of them: Scto et. al. (1996) [1] considered periodic tasks for scheduling of optimized performance of real time control system. Varying sampling period with defined rages of the lower and upper bounds to optimize the system control performance subjected to RMS constraints; Eker et. al. (2000) [2] used feedback scheduler to distribute computing resources over a set of real-time control loops in order to optimize the total control performance. Two issues had been treated; first, how the control performance depends on the sampling interval. Second, how a recursive resource allocation optimization routine can be designed; Cervin et. al. (2002) [3] used a scheduler architecture which combines feedback and feedforward action in order to optimize control performance while maintaining high resource utilization. The feedback performed via the

execution time measurements and feedforward via the workload changes to adjust the sampling periods of the control tasks so that the combined performance of the controllers is optimized. An inverted pendulum example is presented with using cost function describe the performance of each IP's controller. The application for periodic tasks with Earliest Deadline First (*EDF*) scheduling under overload conditions makes it possible to, in certain situations, to interpret a plain EDF dispatcher as a feedback scheduler for control tasks; and Feng Xia et. al. (2007) [4] used feedback scheduling scheme based on *FFNN*. The optimal solutions had been obtained offline by mathematical optimization methods, a Back-Propagation Neural Network (*BPNN*) had been designed to adapt online the sampling periods of concurrent control tasks with respect to changes in computing resource availability. Numerical simulation results show that the proposed scheme can reduce the computational overhead significantly while delivering almost the same overall control performance as compared to optimal feedback scheduling.

II. FEEDBACK SCHEDULING CONTROL SYSTEM

The problem of feedback scheduling can be expressed as a constrained optimization problem, which is usually referred to as Optimal Feedback Scheduling [5].

Feedback Scheduling has been employed in real time systems to overcome the uncertainty in the availability of the resources. The feedback scheduler should control the workload of the processor by adjusting the sampling periods of the controllers. At the same time, it should optimize the overall control performance. This is stated as the following optimization problem [6]. In feedback scheduling, the scheduler get feedback information about the actual execution time of a task, and also feedforward information when the control tasks are about to switch mode. It tries to keep the *CPU* utilization at set-point by manipulating the sampling periods of each task [7].

Optimal feedback scheduling schemes are usually too computationally expensive to be used online, though they are in principle capable of maximizing the overall Quality of Control (*QoC*) of real-time control systems. With the goal of optimizing the overall *QoC* of multitasking embedded control systems, the problem of optimal feedback scheduling is explicitly formulated, and relevant mathematical solutions are discussed [8].

To overcome the disadvantage of overly large computational overheads associated with mathematical optimization routines, a fast feedback scheduling scheme exploiting *FFNN* is used in this chapter. It aims at reducing the feedback scheduling overhead while delivering almost optimal overall *QoC*. Besides, the use of neural networks potentially enhances the adaptability, robustness, and fault-tolerance of the feedback scheduler [8].

Since optimal feedback scheduling schemes are generally too computationally expensive to be used online, schemes with much less computational complexity are desirable. The typical scheme of Intelligent Feedback Scheduler is shown in Fig. 1.

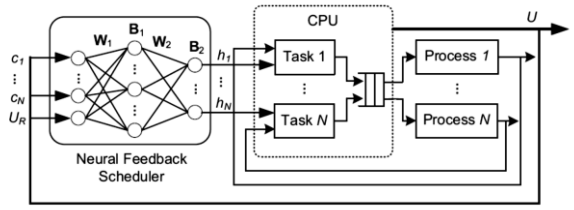


Fig. 1: Architecture of Neural Network Scheduling [9]

Some reasons why the neural networks technology is introduced into the framework of feedback scheduling are described below [8]:

(1) From the overhead perspective, feedforward neural networks with simple structures can yield smaller feedback scheduling overheads than mathematical optimization methods with less complex computations. In addition, information processing inside neural networks is highly parallel, which makes it possible to achieve higher processing speed by means of suitable hardware implementation.

(2) With regard to the accuracy of solutions, mathematical optimization methods can generate accurate optimal solutions offline, which should be exploited in the design of online feedback schedulers. On the other hand, neural networks are powerful in learning and adapting, and enable of approximating arbitrarily complex nonlinear functions with arbitrary precision. Once well trained by the accurate optimal solutions at design time, neural networks will be able to deliver online almost optimal feedback scheduling performance.

(3) The generalization capability of neural networks is also very good, in that neural

networks can easily handle things like untrained input data, noise, and incomplete data. This helps to improve the performance of the feedback scheduler with better fault tolerance.

In this work, different optimization methods had been used to figure out the effectivity of the developed NN with different optimizations methods. Then, as an application, inverted pendulum control system case is presented with *SQP* as optimization method for the optimal feedback scheduling scheme.

A. Neural Feedback Schedule Strategy

The goal of the Neural Feedback Schedule "*NFS*" is to optimally allocate available *CPU* resources among control loops, and ultimately to optimize the overall quality of the control of the system, by exploiting feedback scheduling. In the feedback scheduling loop, the *CPU* utilization is chosen as the controlled variable. It is the responsibility of the feedback scheduler to adapt sampling periods (i.e. task periods) so that the *CPU* utilization is maintained at a desired level. For simplicity, assume that all task execution times and the *CPU* workload are known at runtime.

B. Optimization Complexity

There are many well-known methods for solving the constrained optimization problem formulated by the cost function and the condition. The necessary and sufficient condition for the optimal solutions is given by Kuhn-Tucker condition [10].

Given that the cost function $J_i(f_i)$ is convex. When $J_i(f_i)$ is not convex, the Kuhn-Tucker condition becomes a necessary condition. Since $J_i(f_i)$ is convex for most control systems [2], the Kuhn-Tucker condition can be regarded as a general tool for obtaining the optimal sampling frequencies/periods. Sequential quadratic programming (*SQP*) has been recognized as one of the most efficient methods for solving constrained optimization problems [11].

The optimization role is to minimize the total control cost (*J*) by optimizing scheduling parameters of control tasks under the constraint of system schedulability. As feedback schedulers are usually executed at runtime, it is of paramount importance to take into account the computational overhead of the feedback scheduling algorithm to be employed. If the feedback scheduler consumes too much computing resources, the execution of control tasks definitely will be impacted in the presence of resource constraint. This may then cause significant degradation of the overall performance. Optimization solutions typically involve complex computations, which induce large feedback scheduling overheads. Therefore, they are not suitable for online use in most cases [4].

Solving the optimization problem exactly can be very time-consuming. Evaluating a cost function for a single sampling frequency involves a large amount of computations. If the resource allocation problem is to be solved by an on-line optimizer, the cost functions for the plants must be computed off-line and then approximated by simpler functions. A quadratic approximation was suggested in [2]. The solution to the approximated problem can in both cases be interpreted as a simple linear rescaling of the nominal sampling periods.

The reasons that the computational complexity of the optimization algorithms to be considerably high; In the *SQP* method, for instance, one or two quadratic programming sub-problems must be solved in each iteration and thus take a great deal of time to complete. When applied to feedback scheduling, this algorithm may introduce significant overheads. Practically most of existing optimal feedback scheduling algorithms suffer from the problem of too large computational overheads, which impair their application [4].

The minimization process is carried out using *MATLAB 2012* function *fmincon*, which finds the constrained minimum of a nonlinear multivariable scalar function starting at an initial estimate. This is generally referred to as constrained nonlinear optimization or nonlinear programming.

1) Cost Functions (J)

As a Quality-of-Service (*QoS*) measure, each controller is associated with a cost function *J(h)*, which measures the performance of the controller as a function of the sampling period *h* [2]. And smaller sampling periods yield better control performance, because the performance of sampled-data control with smaller sampling periods approaches more closely to that of

continuous-time control. However, the decrease in sampling period will result in an increase in the requested *CPU* utilization of the relevant control task. In extreme cases the schedulability of the system may be violated, and hence the control performance will deteriorate due to deadline misses. In order to optimize the overall performance, the sampling periods should be adjusted under the constraint of system schedulability [12].

C. General NN Design methodology.

To overcome the problem associated with the large computational overheads of optimal feedback scheduling algorithms, a Neural Feedback Scheduling (*NFS*) scheme has been proposed. The goal is to optimize the overall performance of multitasking control systems through feedback scheduling while minimizing the feedback scheduling overhead. A Feedforward Back-Propagation Neural Network (*FFBPNN*) with a simple structure is adopted to build the feedback scheduler. Training and testing data can be easily created offline by applying the Sequential Quadratic Programming (*SQP*). This scheme has advantages such as low feedback scheduling overhead, wide applicability, and intelligent computation. It is also capable of delivering almost optimal performance.

In this work, the *NN* has been placed to do the optimization role (function) to overcome the weak points which has been mentioned before. The first step starts with determine the cost function of the system and the scheduling method to specify the scheduling condition in the optimization procedure according to the system type. Ranges of data should be specified to be optimized to get the required training data for the pre-specified training data. After train the *NN*, no need to go back to the optimization any more as the *NN* will be placed instead of it to proceed the required data will minimize the overhead and better performance.

The design flow of the neural feedback scheduler is as follows:

Firstly, formulate the problem in the form of constrained optimization equation. Determine the form of the cost functions based on control systems analysis, and initialize related parameters.

Secondly, analyze the characteristics of the execution times of the control tasks to obtain the ranges of their values. Within the ranges of C_i as well as U_R, select a number of data pairs, and for each pair, use the *SQP* method to solve the optimal feedback scheduling problem offline, producing sufficient sample data sets.

Thirdly, determine the number of hidden neurons according to the number of control loops, and initialize the *NN*.

Finally, train and test the neural network using pre-processed sample data sets. Once the *BP* network passes the test, it can thereafter be used online as the neural feedback scheduler.

1) NN Design methodology.

The main role of the *NFS* is to approximate the optimal solutions, which are obtained using mathematical optimization methods. Based on this, training and testing data will not be a problem since it can be easily created offline by applying any of the following optimization algorithm:

- Sequential Quadratic Programming (*SQP*).
- Active Set.
- Interior Point.

Then a suitable *NN* will be developed with each algorithm to overcome the optimization algorithms' complexity and overhead.

As shown in Fig. 1, there is only one hidden layer apart from the input and output layers in the *BP* network used as the neural feedback scheduler. Since feedforward neural networks with only one hidden layer are able to approximate arbitrary functions with arbitrary precision that are continuous on closed intervals, one hidden layer is sufficient for guaranteeing solution accuracy [4].

A three-layer feedforward back-propagation (*BP*) network has been used to build the feedback scheduler as shown in Fig. 2.

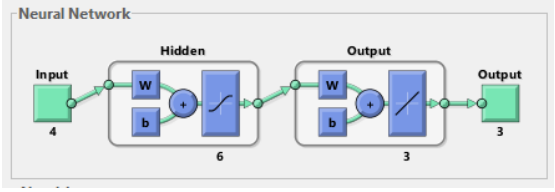


Fig. 2: A Three-Layer Feedforward Back-Propagation (BP) Network

With given cost functions, the values of sampling frequencies (f_i) will depend on the execution time (C_i) of each control task and the desired *CPU* utilization U_R. The consequence of this work, (N+1) inputs, i.e.,(C_1, …, C_N, and U_R), are set for the neural feedback scheduler inputs. Since the role of the feedback scheduler is to determine sampling periods of all loops, the sampling periods (h_1, …, h_N) or frequencies (f_1, …, f_N) are natural outputs of the neural feedback scheduler.

Both inputs and outputs of the feedback scheduler are related to resource utilization based on real-time scheduling viewpoint. While sampling periods/frequencies are important design parameters of the control loops based on control perspective. For that reason, the neural feedback scheduler establishes a mapping from temporal parameters (for real-time scheduling) to controller parameters (for real-time control) [5].

2) Neural Feedback Scheduler Design

In order to determine the number of hidden neurons, i.e. the value of M, neural networks of different sizes are compared, gives the training errors of neural networks with M = 4, 6, 8, 10, 16, and 20, respectively. Given that the performance is comparable, a smaller M value should be chosen in order to reduce the feedback scheduling overhead. From this insist, it is set that M = 6 because of the good performance of corresponding neural network and the fast reaching to the required results.

III. SIMULATION AND RESULTS

In this work, the developed feedback scheduling scheme using *FFNN* will be presented in addition to presenting the effectivity of NN to solve different optimization complexity and act as optimization technique. Simulation carried out by using *Matlab 2012.*

The runtime overhead of the neural feedback scheduler is examined in comparison with the optimal feedback scheduler. Both of the feedback schedulers are implemented in the same environment using *Matlab 2012*. The hardware platform is the same PC with an Intel® *Core (TM) i5-3337U @ 1.8 GHz, RAM 8.00 GB*. The operating system is Windows® 8.1.

A. NNO for different Optimization Algorithms

1) General Overview on Optimization Methods

Determine the "best" solutions to certain mathematically defined problems that are under constrained:

- determine optimality criteria.
- determine the convergence of the solution.

The advent of computer had great impact on the development of optimization methods:

a. Linear Programming: is the problem of finding a vector x that minimizes a linear function $f^T x$ subject to linear constraints:

$$\min_x f^T x \quad (1)$$

Such that one or more of the following hold: A·x ≤ b, Aeq·x = beq, l ≤ x ≤ u

- Large Scale Linear Programming
- Active-Set Medium-Scale linprog Algorithm
- Medium-Scale linprog Simplex Algorithm

b. Quadratic programming: is the problem of finding a vector x that minimizes a quadratic function, possibly subject to linear constraints

$$\min_x \frac{1}{2} x^T H x + c^T x \quad (2)$$

Such that A·x ≤ b, A_{eq}·x = b_{eq}, l ≤ x ≤ u

- Interior-Point-Convex quadprog Algorithm
- Trust-Region-Reflective quadprog Algorithm
- Active-Set quadprog Algorithm

c. Nonlinear Optimization Algorithms: constrained minimization is the problem of finding a vector x that is a local minimum to a scalar function f(x) subject to constraints on the allowable x

$$\min_x f(x) \quad (3)$$

such that one or more of the following holds: c(x) ≤ 0, c_{eq}(x) = 0, A·x ≤ b, A_{eq}·x = b_{eq}, l ≤ x ≤ u.

- Trust Region Reflective Algorithm

- Active Set Algorithm
- *SQP* Algorithm
- Interior Point Algorithm

2) Complexity analysis.

The application of neural feedback schedulers contains not only online computations but also offline computations, e.g., mathematical optimization, network training and test, etc., only the complexity of online computations is of concern. The feedback scheduling overhead is determine the computational operations at runtime.

With a given time interval, the amount of computing resource consumed by the feedback scheduler depends directly on the *CPU* time needed for each run.

Below methods has been chosen to compare results with adapted neural network.

- Interior-point methods: in mathematical programming have been the largest and most dramatic area of research in optimization since the development of the simplex method [13].

Interior-point methods have permanently changed the landscape of mathematical programming theory, practice and computation (also referred to as barrier methods) which deals with a certain class of algorithms to solve linear and nonlinear convex optimization problems.

- Active Set: Is a procedure for determining inequality constraints which will be "Active" at each iteration. An essential part of many iterative methods for linearly constrained [13].
- Sequential Quadratic Programming (*SQP*): is one of the most successful methods for the numerical solution of constrained nonlinear optimization problems. It relies on a profound theoretical foundation and provides powerful algorithmic tools for the solution of large-scale technologically relevant problems [14].

Tables 1, 2, and 3 have been created based on inverted pendulum cost function to optimize the sampling periods with *RMS* constraints. The execution time Ci ranges started from 0.001 to 0.009 seconds.

The goal of creating Tables 1, 2, and 3 is to compare the mentioned three optimization algorithms results with the developed *FFNN* in order to show the effectivity of *NN* in solving optimizations issues.

Table 1: Interior Point Optimization Data Sample

C_1	C_2	C_3	U	h_1	h_2	h_3	J_{min}
0.001	0.001	0.001	0.7	0.005335	0.004274	0.003589	0.856772
0.001	0.001	0.002	0.26	0.016732	0.013404	0.01592	3.129933
0.001	0.001	0.004	0.28	0.018646	0.014938	0.02509	4.186178
0.001	0.001	0.006	0.3	0.01963	0.015726	0.03235	4.971029
0.001	0.001	0.008	0.32	0.020164	0.016153	0.038369	5.594411
0.001	0.002	0.001	0.34	0.012505	0.014167	0.008413	2.286172
0.001	0.002	0.003	0.36	0.014833	0.016805	0.017284	3.405738
0.001	0.002	0.005	0.38	0.016023	0.018154	0.024105	4.195334
0.001	0.002	0.007	0.4	0.016745	0.018971	0.029806	4.822609
0.001	0.002	0.009	0.42	0.017201	0.019488	0.034717	5.343487
0.001	0.003	0.002	0.44	0.011964	0.016601	0.011383	2.708084
0.001	0.003	0.004	0.46	0.013336	0.018505	0.017945	3.518109
0.001	0.003	0.006	0.48	0.014173	0.019666	0.023356	4.145868
0.001	0.003	0.008	0.5	0.014732	0.020442	0.028034	4.666357
0.001	0.004	0.001	0.52	0.009582	0.015353	0.006447	2.053177
0.001	0.004	0.003	0.54	0.011243	0.018013	0.013101	2.934895
0.001	0.004	0.005	0.56	0.012179	0.019513	0.018322	3.571633
0.001	0.004	0.007	0.58	0.012809	0.020523	0.0228	4.091764
0.001	0.004	0.009	0.6	0.013259	0.021245	0.026762	4.535926
0.001	0.005	0.002	0.62	0.009505	0.017027	0.009044	2.408705
0.001	0.005	0.004	0.64	0.010569	0.018932	0.014221	3.073915
0.001	0.005	0.006	0.66	0.011261	0.020172	0.018557	3.598654
0.001	0.005	0.008	0.68	0.011758	0.021062	0.022374	4.042283
0.001	0.006	0.001	0.7	0.00792	0.015542	0.005328	1.888039
0.001	0.006	0.002	0.26	0.023691	0.046489	0.022541	6.274874
0.001	0.006	0.004	0.28	0.025108	0.049271	0.033785	7.590355
0.001	0.006	0.006	0.3	0.025661	0.050356	0.042289	8.494793
0.001	0.006	0.008	0.32	0.025818	0.050663	0.049129	9.171821
0.001	0.007	0.001	0.34	0.017027	0.036088	0.011455	4.238305
0.001	0.007	0.003	0.36	0.019103	0.04049	0.02226	5.648975
0.001	0.007	0.005	0.38	0.020069	0.042537	0.030191	6.581145
0.001	0.007	0.007	0.4	0.020588	0.043637	0.036647	7.290391
0.001	0.007	0.009	0.42	0.020861	0.044217	0.042105	7.859498
0.001	0.008	0.002	0.44	0.015074	0.034157	0.014342	4.299216
0.001	0.008	0.004	0.46	0.016312	0.036961	0.021948	5.262839
0.001	0.008	0.006	0.48	0.017024	0.038574	0.028055	5.981715
0.001	0.008	0.008	0.5	0.017469	0.039584	0.033243	6.56136

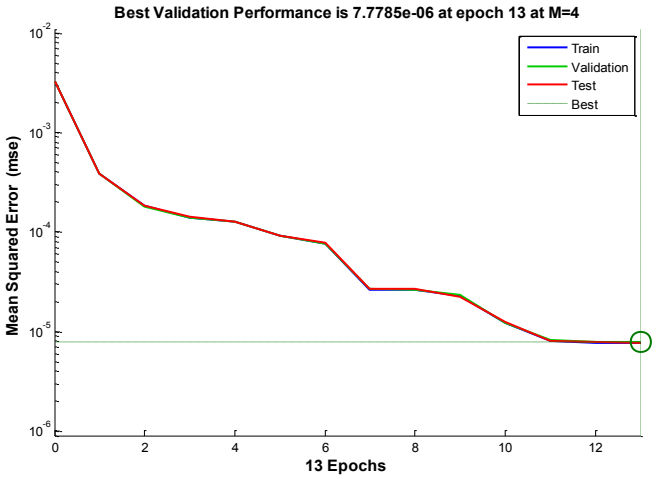
Table 2 :Active Set Optimization Sample Data

C_1	C_2	C_3	U	h_1	h_2	h_3	J_{min}
0.001	0.001	0.001	0.7	0.005335193	0.004274056	0.003589432	0.856771075
0.001	0.001	0.002	0.26	0.016730129	0.013403069	0.015921271	3.129921914
0.001	0.001	0.004	0.28	0.01864702	0.014938149	0.02508946	4.186176493
0.001	0.001	0.006	0.3	0.019631421	0.015726379	0.032349571	4.971027696
0.001	0.001	0.008	0.32	0.020163258	0.016153717	0.03836938	5.594410211
0.001	0.002	0.001	0.34	0.012507925	0.014166614	0.008412282	2.286170665
0.001	0.002	0.003	0.36	0.014834498	0.01680423	0.017283901	3.405737467
0.001	0.002	0.005	0.38	0.016023595	0.018153875	0.02410536	4.195333434
0.001	0.002	0.007	0.4	0.016744687	0.018970928	0.029805613	4.822606909
0.001	0.002	0.009	0.42	0.017202523	0.019489173	0.034715729	5.343477418
0.001	0.003	0.002	0.44	0.011963733	0.016600641	0.011383125	2.708080312
0.001	0.003	0.004	0.46	0.013336234	0.018505308	0.017945213	3.518108945
0.001	0.003	0.006	0.48	0.014168518	0.019677233	0.023349874	4.145858904
0.001	0.003	0.008	0.5	0.014732295	0.020442044	0.028034199	4.66635459
0.001	0.004	0.001	0.52	0.009583103	0.015353068	0.006446807	2.053175691
0.001	0.004	0.003	0.54	0.011244536	0.018012625	0.013100352	2.934894367
0.001	0.004	0.005	0.56	0.012178283	0.019512143	0.018322643	3.571630847
0.001	0.004	0.007	0.58	0.012809305	0.020522577	0.02279947	4.091762445
0.001	0.004	0.009	0.6	0.0132602	0.021244393	0.026761696	4.535924044
0.001	0.005	0.002	0.62	0.009505178	0.017027205	0.00904378	2.408704497
0.001	0.005	0.004	0.64	0.010569199	0.018933375	0.014219607	3.073874337
0.001	0.005	0.006	0.66	0.011259323	0.020174224	0.018556104	3.598653829
0.001	0.005	0.008	0.68	0.011757904	0.021062329	0.022373856	4.042282229
0.001	0.006	0.001	0.7	0.007919821	0.015541538	0.005328435	1.888036689
0.001	0.006	0.002	0.26	0.023690459	0.046491382	0.022539587	6.274873122
0.001	0.006	0.004	0.28	0.025108393	0.049270774	0.033784739	7.590352951
0.001	0.006	0.006	0.3	0.025662114	0.050353861	0.042290637	8.49479014
0.001	0.006	0.008	0.32	0.025816638	0.050663856	0.049128652	9.171815716
0.001	0.007	0.001	0.34	0.017026107	0.03608891	0.011454587	4.238265352
0.001	0.007	0.003	0.36	0.019103005	0.040489827	0.022260262	5.648972532
0.001	0.007	0.005	0.38	0.020068964	0.042537352	0.03019131	6.581142508
0.001	0.007	0.007	0.4	0.020587886	0.043637158	0.036646528	7.290388873
0.001	0.007	0.009	0.42	0.020861183	0.044216134	0.0421051	7.859496341
0.001	0.008	0.002	0.44	0.015074251	0.034156807	0.014342285	4.299215955
0.001	0.008	0.004	0.46	0.016311515	0.036959712	0.021948817	5.262833494
0.001	0.008	0.006	0.48	0.017024219	0.038574504	0.028054497	5.981710375
0.001	0.008	0.008	0.5	0.017471684	0.03958263	0.033242513	6.56135729

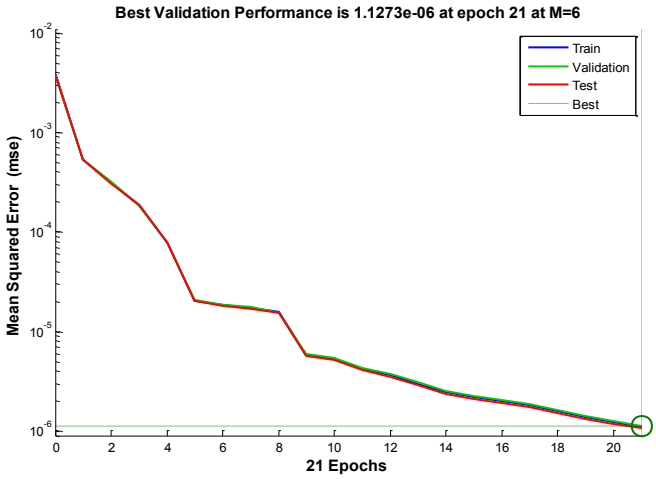
Table 3: SQP Optimization Data Sample

C_1	C_2	C_3	U	h_1	h_2	h_3	J_{min}
0.001	0.001	0.001	0.7	0.005335185	0.004274113	0.003589396	0.856771149
0.001	0.001	0.002	0.26	0.016732035	0.013404303	0.015919574	3.129925393
0.001	0.001	0.004	0.28	0.018646413	0.014938	0.02508985	4.18617755
0.001	0.001	0.006	0.3	0.019629785	0.015725861	0.032350684	4.971028433
0.001	0.001	0.008	0.32	0.020163581	0.01615343	0.038369438	5.594410346
0.001	0.002	0.001	0.34	0.012504844	0.014167418	0.008413118	2.286171439
0.001	0.002	0.003	0.36	0.014832687	0.016804753	0.017284354	3.405737562
0.001	0.002	0.005	0.38	0.016023493	0.018153861	0.02410542	4.19533376
0.001	0.002	0.007	0.4	0.01674467	0.018970925	0.029805622	4.82260691
0.001	0.002	0.009	0.42	0.017201062	0.019487923	0.034717348	5.343484559
0.001	0.003	0.002	0.44	0.011963847	0.016600741	0.011383037	2.708083671
0.001	0.003	0.004	0.46	0.013336487	0.018505419	0.017945021	3.518108973
0.001	0.003	0.006	0.48	0.014172697	0.019665724	0.023356108	4.145859751
0.001	0.003	0.008	0.5	0.014732278	0.020442164	0.028034141	4.666356314
0.001	0.004	0.001	0.52	0.009582415	0.015353358	0.006446919	2.05317615
0.001	0.004	0.003	0.54	0.011242561	0.018013223	0.013100826	2.934894505
0.001	0.004	0.005	0.56	0.01217886	0.019513326	0.018321569	3.571632891
0.001	0.004	0.007	0.58	0.012808756	0.020522671	0.022799669	4.091764012
0.001	0.004	0.009	0.6	0.013259381	0.02124468	0.026761878	4.535925362
0.001	0.005	0.002	0.62	0.009505179	0.017027241	0.009043754	2.408704516
0.001	0.005	0.004	0.64	0.010568644	0.018932086	0.014220777	3.073875298
0.001	0.005	0.006	0.66	0.011260663	0.020171844	0.018557178	3.59865391
0.001	0.005	0.008	0.68	0.011757758	0.021062333	0.02237392	4.042282261
0.001	0.006	0.001	0.7	0.007919902	0.015541543	0.005328396	1.888036726
0.001	0.006	0.002	0.26	0.023690881	0.046489437	0.022540776	6.274873876
0.001	0.006	0.004	0.28	0.025108324	0.04927087	0.033784708	7.590353435
0.001	0.006	0.006	0.3	0.025661484	0.050356306	0.042289205	8.494790857
0.001	0.006	0.008	0.32	0.025817749	0.050663026	0.049128765	9.17181859
0.001	0.007	0.001	0.34	0.017026116	0.036087547	0.011455545	4.238265422
0.001	0.007	0.003	0.36	0.019102899	0.040489763	0.022260358	5.64897283
0.001	0.007	0.005	0.38	0.020068935	0.042537355	0.030191322	6.58114257
0.001	0.007	0.007	0.4	0.020587855	0.043637214	0.036646503	7.290388874
0.001	0.007	0.009	0.42	0.020861156	0.044216499	0.042104855	7.859496366
0.001	0.008	0.002	0.44	0.015074179	0.034156706	0.01434239	4.299216047
0.001	0.008	0.004	0.46	0.016311869	0.036960352	0.02194826	5.262838669
0.001	0.008	0.006	0.48	0.017023854	0.03857445	0.028054723	5.981712554
0.001	0.008	0.008	0.5	0.017469381	0.03958398	0.033242605	6.561357533

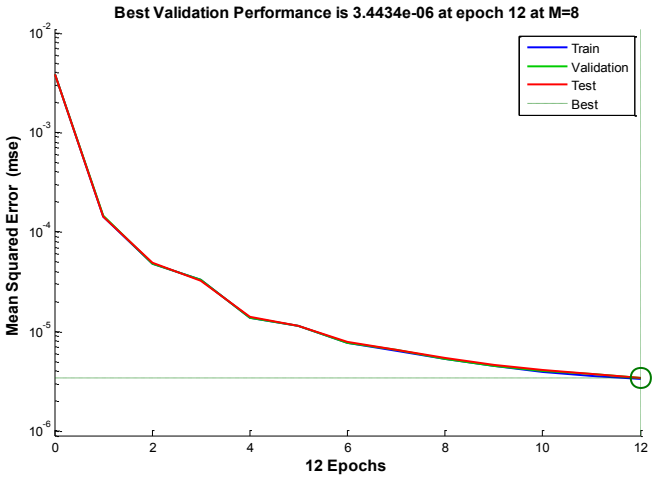
In order to determine the number of hidden neurons, i.e. the value of *M*, neural networks of different sizes (4,6,8,10,16, and 20) has been compared and the most applicable one was with *M*=20, as the performance indices equals to 1.8079×10^{-6}, 2.7711×10^{-7}, and 7.301×10^{-8} for Interior-Point, Active-Set, and SQP optimization algorithms' respectively as shown in Fig. 4, 5, and 6. Given that the performance is comparable. From this perspective, it is set that $M = 20$ because of the good performance of corresponding *NN* and the fast reaching to the required results.



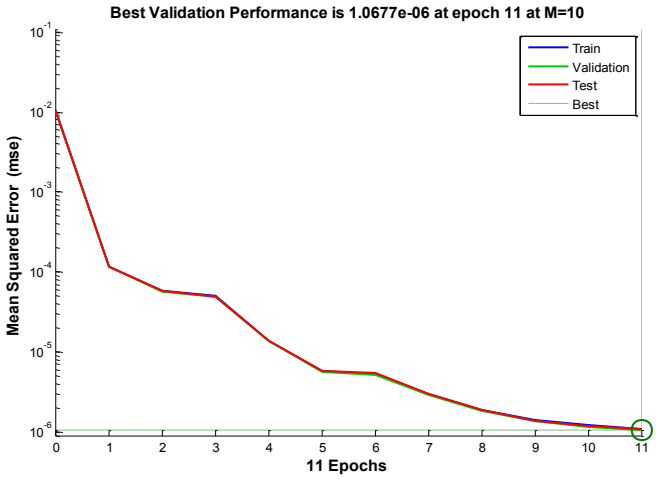
(a)



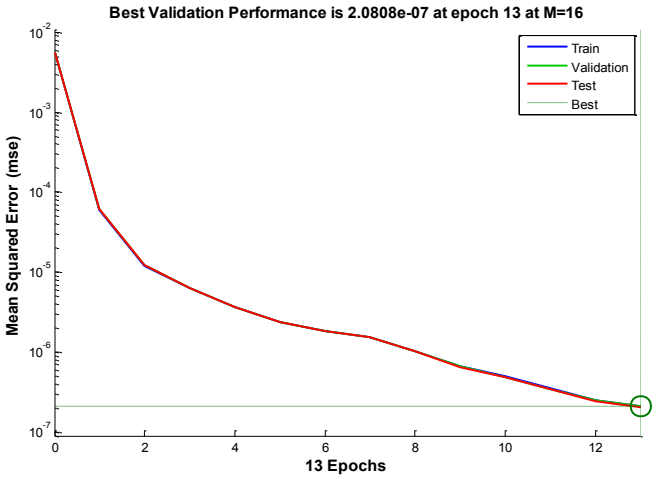
(b)



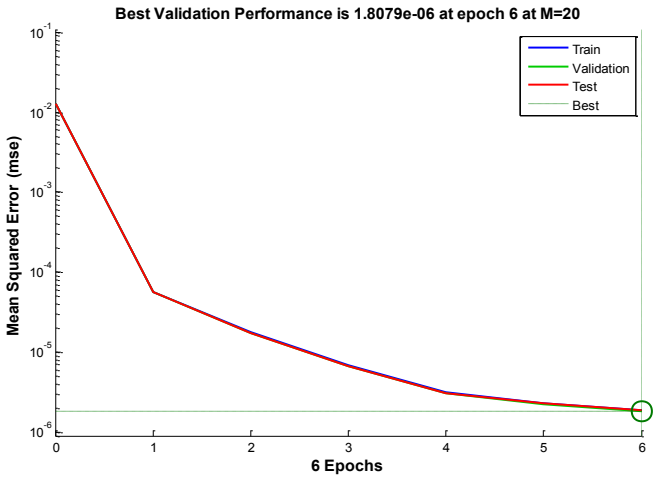
(c)



(d)



(e)



(f)

Fig. 3: NNO Performance for Interior-Point Dataset

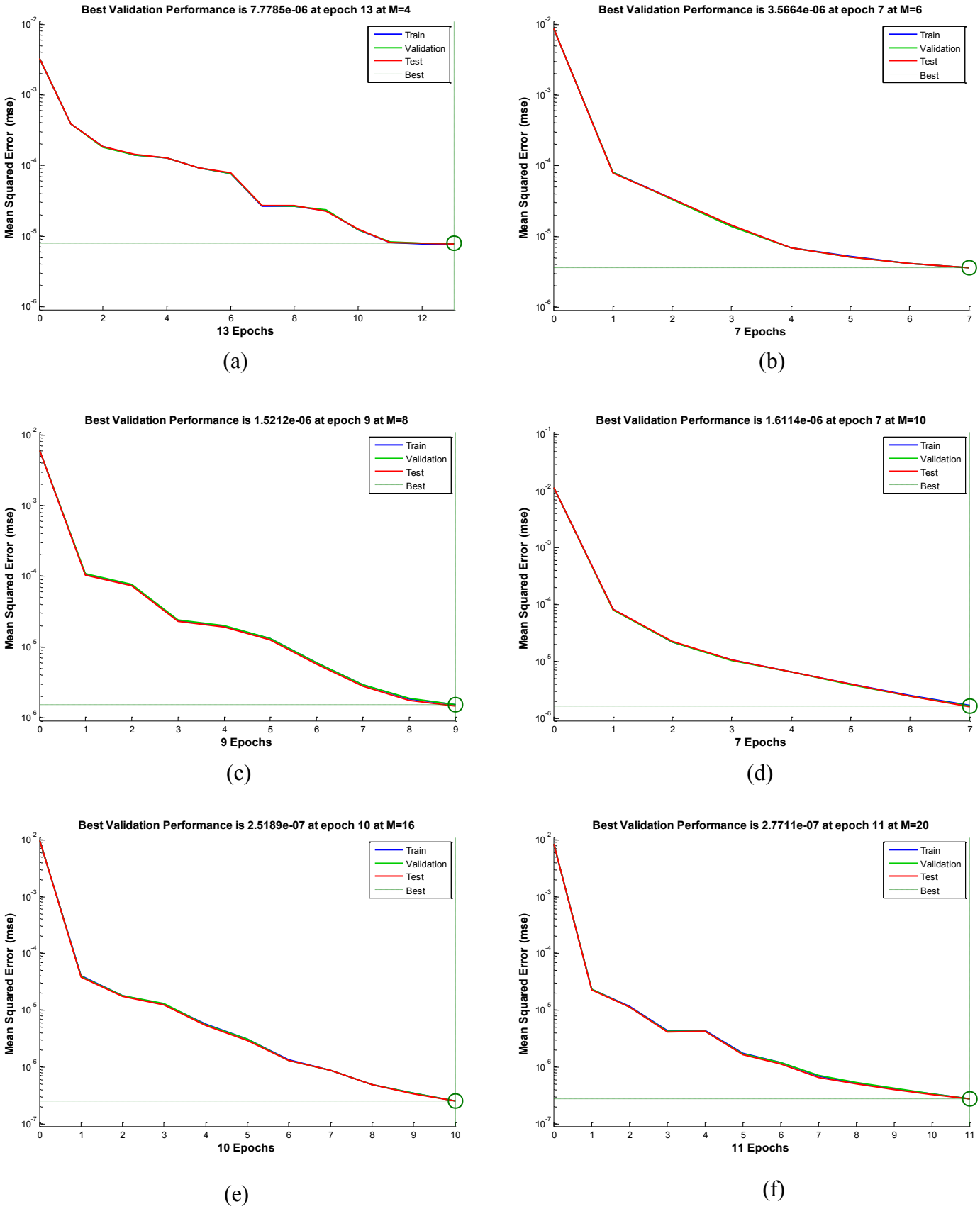


Fig. 4: NNO Performance for Active Set Dataset

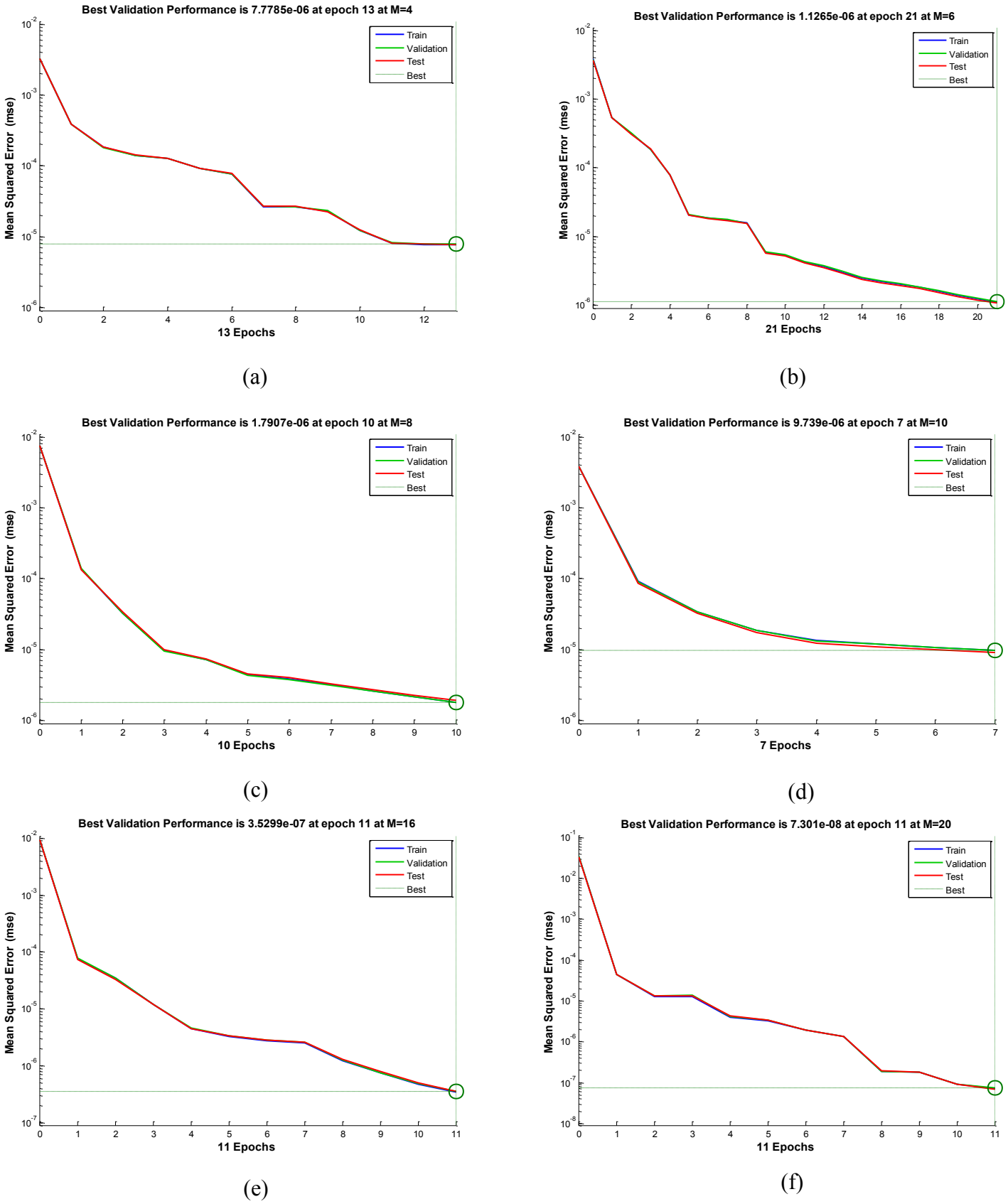


Fig. 5: NNO Performance for SQP Dataset

3) NNO and Optimization Algorithm Comparison

By choosing random values for C_1, C_2, C_3, and U to evaluate the *NNO* performance comparing with 3 different optimization algorithms, let $C_1=0.001$, $C_2=0.004$, $C_3=0.009$; and $U=0.6$, the optimization and *NNO* results are shown in Table 4, Table 5, and Table 6.

As shown in Table 4, the *NNO* enhanced the overhead by 82.5% in Interior Point optimization algorithm type with the same minimum cost function value *J*.

Table 4: Comparison Between Interior Point and NNO Performance

Method	h_1 (s)	h_2 (s)	h_3 (s)	Min J	U< 0.7798	Over-head (s)
Interior-Point	0.013259	0.021245	0.026762	4.535926	0.6	0.6210
NNO	0.0133	0.0212	0.0268	4.5359	0.6	0.108410

As shown in Table 5, the *NNO* enhanced the overhead by 91.88% than Active Set optimization algorithm type with the same minimum cost function value *J*.

Table 5: Comparison Between Active Set and NNO Performance

Method	h1 (s)	h2 (s)	h3 (s)	Min J	U< 0.7798	Over-head (s)
Active Set	0.013260	0.021244393	0.026761696	4.535924044	0.6	1.3606
NNO	0.0137	0.0212	0.0267	4.5359	0.6	0.110410

As shown in Table 6, the *NNO* enhanced the overhead by 79.75% than *SQP* optimization algorithm type with the same minimum cost function value *J*.

Table 6: Comparison Between SQP and NNO Performance

Method	h1 (s)	h2 (s)	h3 (s)	Min J	U< 0.7798	Over-head (s)
SQP	0.013259381	0.02124468	0.026761878	4.535925362	0.6	0.6339
NNO	0.0133	0.0214	0.0264	4.5359	0.6	0.128397

B. Intelligent Feedback Scheduling for Inverted Pendulum Control System

In this case, multitasking embedded processor that is responsible for controlling three inverted pendulums concurrently has been considered, as shown in Fig. 7.

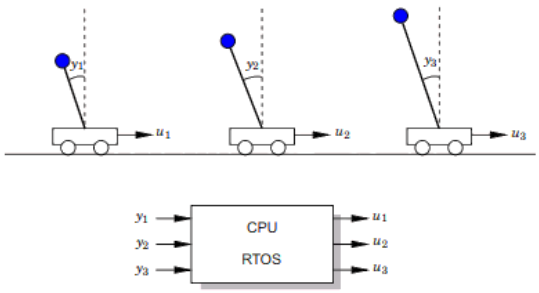


Fig 6: Real Time Operating System (RTOS)

System where three independent control tasks running and controlled by a one processor (resource) with limited processing capability. This control problem has been depicted in Fig7. A linear digital controller is designed for each pendulum. The pendulum different lengths motivate different sampling intervals for the different controllers: h_1, h_2, h_3 = 58.8, 71.4, and 83.3 ms [15]. Besides these three control tasks, there is an additional periodic non-control task running on the same processor. The execution time of this task is variable, causing UR to vary over time. The execution times of control tasks and the requested *CPU* utilization of non-control tasks may change over time. The desired total *CPU* utilization of all tasks is set to $U = 0.75 < 4(21/4 - 1) = 0.76$. The feedback scheduler adapts the sampling periods (h_i) of the control tasks to workload variations so that the CPU utilization is maintained at a desired level $U_R=U-C_4/h_4$ [5]. According to Liu and Layland [12], the system schedulability under the *RMS* algorithm is guaranteed by *U*. The execution time of the non-control task is C_4, and its period h_4 = 10 ms. Therefore, $U_R=U-C_4/h_4$, implying that UR will change with *C4*. All task execution times (C_i) and the *CPU* workload has been assumed available at run-time and by using *NNO*, sampling periods can be produced to fit control stability criteria and *RMS* constraints. The scheduling part is done by *NNS* which built based on *RMS* criteria.

Let h_i and C_i denote the sampling period and the execution time of control task *i*, respectively. Optimal feedback scheduling can be formulated as a constrained optimization problem [8]:

$$\underbrace{min}_{h_1,\ldots,h_N} \quad J = \sum_{i=1}^{N} J_i(h_i) \tag{4}$$

Subjected to $$\sum_{i=1}^{N} \frac{C_i}{h_i} \le U_R \tag{5}$$

where $J_i(h_i)$ is the control cost function of loop *i*, as a function of the sampling period; U_R is the maximum allowable utilization reference of all control tasks and is related to the underlying scheduling policy employed and the requested utilization of disturbing task.

Each pendulum is controlled independently by *PID* Controller, whose objective is to

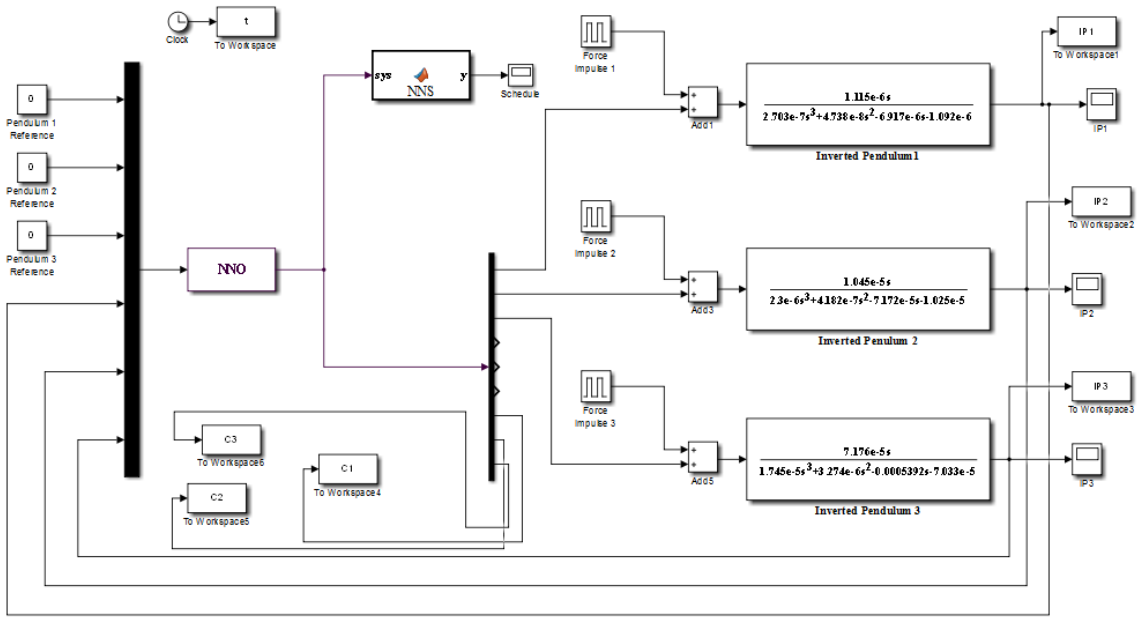


Fig. 7: Inverted Pendulums Simulation Scheme

minimize the below cost function. The costs functions for the three pendulums as functions of the sampling period are computed according to cost function.

$$J_i(h) = \alpha_i + \gamma_i h_i \quad (6)$$

For simplicity the constant α equal to zeroes for all values and the estimated slopes of the cost functions are γ=43, 67, 95, and the pendulum frequencies is estimated as h_i = 1/58.8, 1/71.4, 1/83.3 in *ms*, and for non-control task $h_4$will be 10 ms [5].

After subscription in above Equation 5.4 and Equation 5.6, the cost function will be represented as:

$$\underbrace{min}_{h_1,h_2,h_3} \; J = 43h_1 + 67h_2 + 95h_3 \quad (7)$$

Subjected to $\frac{c_1}{h_1} + \frac{c_2}{h_2} + \frac{c_3}{h_3} \leq 0.75 - \frac{c_4}{0.01}$ (8)

For the purpose of creating sample data, the ranges of C_1, C_2 , C_3 are taken as [0.1, 0.9], [0.1, 0.9], and [0.1, 0.9],respectively, with increments of 0.1. C_4 takes on values ranging from 0.5 to 5 with increments of 0.05. The unit of these parameters is *ms*. For all possible values of these parameters, applying *SQP* offline resulted totally 33534 sets of sample data.

The applied parameters for inverted pendulums are listed in
Table 7.

Table 7: Inverted Pendulums Parameters

For all Inverted Pendulums		
M	mass of the cart	0.5 kg
b	coefficient of friction for cart	0.1 N/m/sec
I	mass moment of inertia of the pendulum	0.006 kg.m2
Inverted Pendulum 1		
m	mass of the pendulum	0.133 kg
l	length to pendulum center of mass	0.2 m
Inverted Pendulum 2		
m	mass of the pendulum	0.2 kg
l	length to pendulum center of mass	0.3 m
Inverted Pendulum 3		
m	mass of the pendulum	0.267 kg
l	length to pendulum center of mass	0.4 m

Inverted Pendulum Transfer Function is:

$$\emptyset = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \quad (9)$$

where $q = [(M+m)(I+ml^2) - (ml)^2]$

1) Simulation under SQP Optimization Results

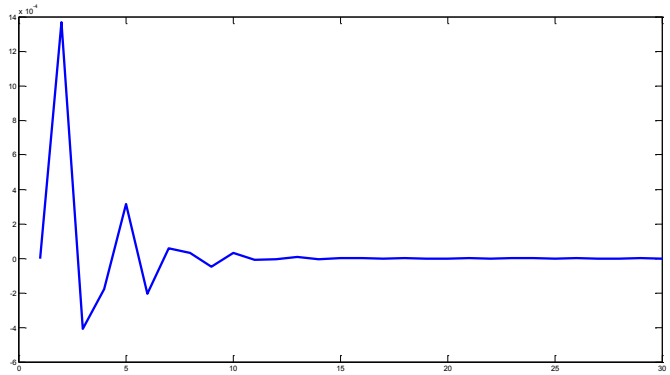


Fig. 8: Inverted Pendulum 2 Response after applying Traditional SQP Algorithm

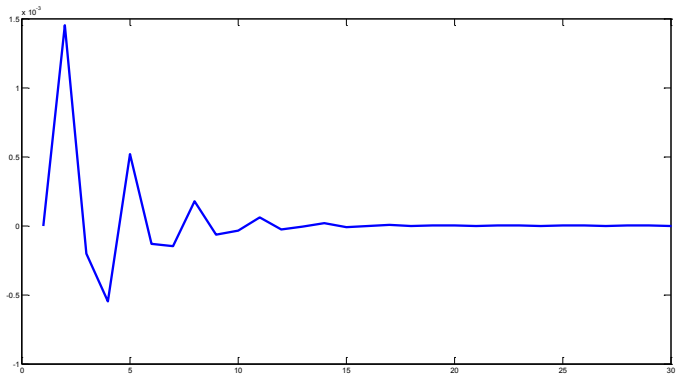


Fig. 9: Inverted Pendulum 2 Response after applying NNO

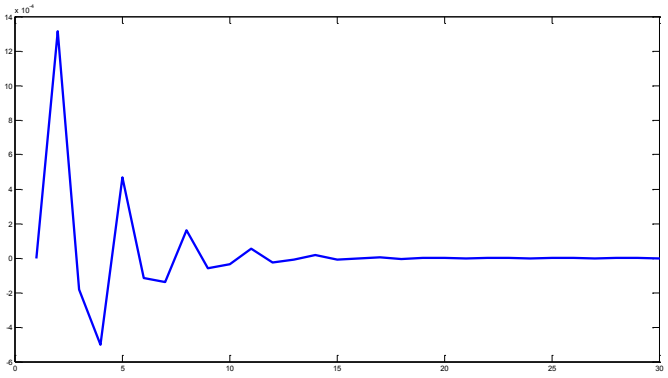


Fig. 10: Inverted Pendulum 3 Response after applying Traditional SQP Algorithm

2) Simulation under Proposed NNO Results

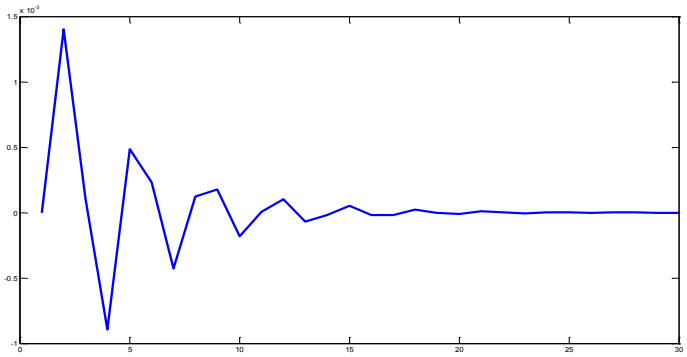


Fig. 11: Inverted Pendulum 1 Response after applying NNO

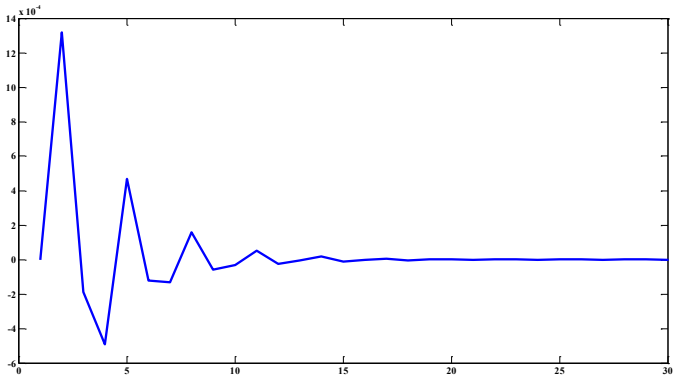


Fig. 12: Inverted Pendulum 1 Response after applying Traditional SQP Algorithm

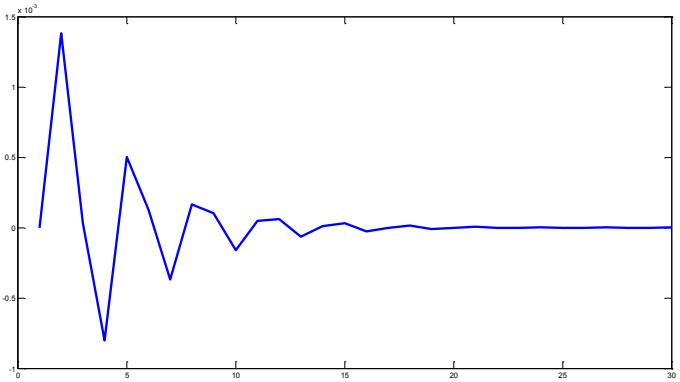


Fig. 13: Inverted Pendulum 3 Response after applying NNO

3) Scheduling Results by Proposed NNS

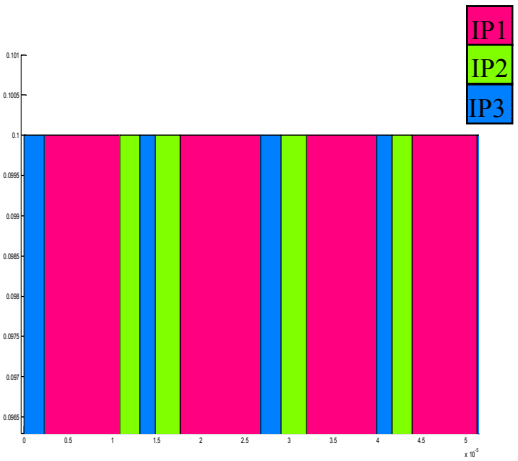


Fig. 14: RMS Scheduling of Inverted Pendulums

From Control perspective the results comparison of *SQP* optimization and the proposed *NNO* show that all pendulums demonstrated same response and speed to stable state as shown in Figures 10 to 15.

From scheduling perspective, Fig. 14 shows the scheduling results of the control tasks based on *RMS* algorithm which depended on the sampling periods.

The comparison of different feedback schedulers in terms of time overhead can be clarified as; for the optimal feedback scheduler and the neural feedback scheduler, the *CPU* time they actually expend for 1000 consecutive runs is recorded, respectively. In each run, task execution time is randomly drawn from the sets given in Fig. 15. The average execution time of the optimal feedback scheduler based on the SQP method falls between (C_1 = *1.77*$\times 10^{-05}$ *and 0.0002024)s, (*C_2 = *6.27*$\times 10^{-06}$ *and 2.74*$\times 10^{-05}$ *)s, and (*C_3 = *6.27*$\times 10^{-06}$ *and 4.96*$\times 10^{-05}$) in most cases, with an average of C_1=*2.43E-05,* C_2=*8.73*$\times 10^{-06}$*, and* C_3=*8.99*$\times 10^{-06}$ sec. And the execution time of the neural feedback scheduler falls between (C_1 =*1.65E-05 and 9.81*$\times 10^{-05}$ *)s, (*C_2 = *5.13*$\times 10^{-06}$ *and 7.01*$\times 10^{-05}$)s, and (C_3 = *4.56*$\times 10^{-06}$ *and 2.57*$\times 10^{-05}$)s in most cases, with an average of C_1=*1.99*$\times 10^{-05}$, C_2=*6.40*$\times 10^{-06}$*, and* C_3=*5.51*$\times 10^{-06}$ *sec*. The ratio of the time overhead of NFS is only 18%, 26%, and 39% that of *SQP* for C_1, C_2, *and* C_3.

IV. DISCUSSION

As a fast and intelligent feedback scheduling scheme, neural feedback scheduling has been proposed in this chapter for real-time control tasks. It fully exploits the offline solutions for the optimal feedback scheduling problem, which are offered by mathematical optimization algorithms. With the proposed approach, almost optimal *QoC* can be achieved. Meanwhile, compared to optimal feedback scheduling, it can significantly reduce the runtime overhead, which is particularly beneficial to embedded control systems that operate in resource-constrained and dynamic environments.

Simulation results argue that neural feedback scheduling can dramatically reduce the feedback scheduling overhead, while yielding overall *QoC* almost identical with optimal feedback scheduling.

It is clear from the results that the neural feedback scheduler induces significantly less computational overhead than the optimal feedback scheduler. The proposed approach does not rely on any specific forms of the control cost functions, making it widely applicable. In addition, the use of neural networks potentially enhances the adaptability, robustness, and fault-tolerance of the feedback schedule.

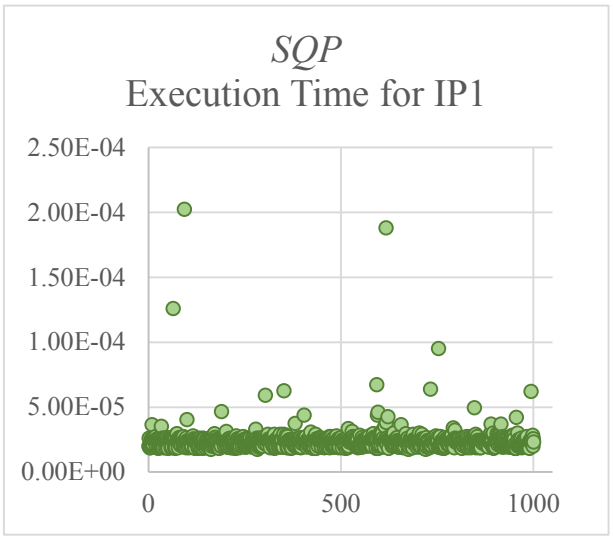
V. REFERENCES

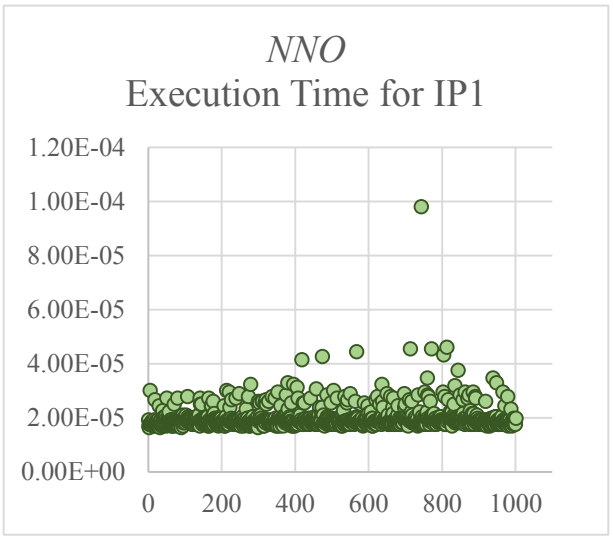
[1] Danbing Seto, J. P. Lehoczky, Lui Sha, and Kang Shin, "On Task Schedulability in Real-Time Control Systems," in *In Proceedings of the 17th IEEE Real-Time Systems Symposium*, Los Alamitos, CA, 1996, pp. 13 - 21.

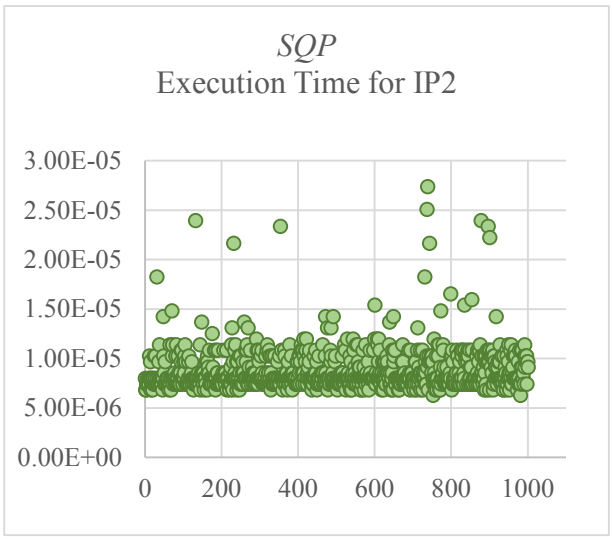
[2] Johan Eker, Per Hagander, and Karl-Erik Årzén, "A Feedback Scheduler For Real-Time Controller Tasks," *Control Engineering Practice*, vol. 8, no. 12, pp. 1369-1378, 2000.

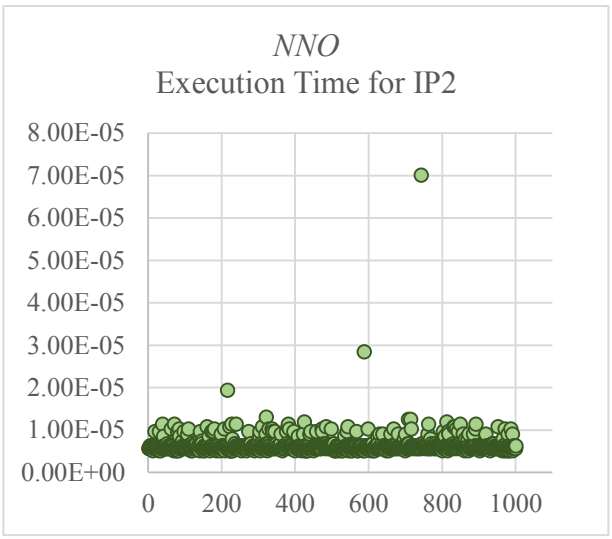
[3] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén, "Feedback-Feedforward Scheduling of Control Tasks," *Real-Time Systems*, vol. 23, no. 1-2, pp. 25-53, July 2002.

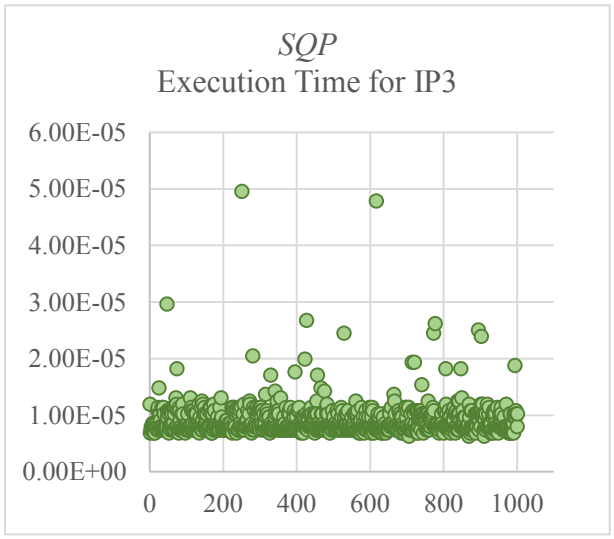
[4] Feng Xia, Yu-Chu Tian, Youxian Sun, and Jinxiang Dong, "Neural Feedback Scheduling Of Real-Time Control Tasks," *International Journal of Innovative Computing, Information and Control, ICIC*, vol. 4, no. 11, pp. 2965–2975, December 2007.











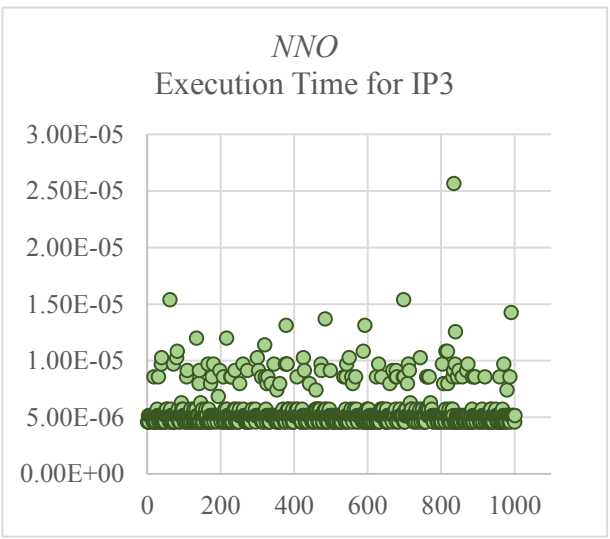


Fig. 15: Execution Time for Inverted Pendulums

[5] Feng Xia, Yu-Chu Tian, Youxian Sun, and Jinxiang Dong, "Neural Network Scheduling of Real Time Control Tasks," *International Journal of Innovative*, vol. 4, no. 11, pp. 2965 - 2975, November 2008.

[6] Danbing Seto, Lehoczky J.P. , Lui Sha, and Shin K.G., "On Task Schedulability in Real-Time Control Systems.," in *Real-Time Systems Symposium, 17th IEEE*, Los Alamitos, CA , 1996, pp. 13 - 21.

[7] Wei Zhang, Stability Analysis of Networked Control Systems, Augest 2001.

[8] Feng Xia and Youxian Sun, *Control and Scheduling Codesign Flexible Resource Management in Real-Time Control Systems*.: Co-published by Zhejiang University Press, Han^hou and Springer -Verlag GmbH Berlin Heidelberg, 2008.

[9] Feng Xia and Youxian Sun, "Control-Scheduling Codesign: A Perspective on Integrating Control and Computing," *Published i Dynamics of Continuous, Discreate and Impulsive Systems - Series B*, vol. 13, no. S1, pp. 1352-1358, 2006.

[10] Boyd, S. and L. Vandenberghe, *Convex Optimization*. United Kingdom: Cambridge University Press, 2004.

[11] Zhu, Z.B., "A simple feasible SQP algorithm for inequality constrained optimization," *Applied Mathematics and Computation*, vol. 182, pp. 987-998, 2006.

[12] C.L. Liu and J.W. Layland, "Scheduling algorithm for multiprogramming in a hard-real-time environment," *J.ACM*, vol. 20(1), pp. 46-61, January 1973.

[13] Sven Leyffer, "The Return of the Active Set Method," ARGONNE NATIONAL LABORATORY, Argonne, 2005.

[14] Dr. Ronald H.W. Hoppe, Optimization Theory. http://www.math.uh.edu/~rohop/fall_06/. 2006.

[15] Anton Cervin, *Integrated Control and Real-Time Scheduling*. Lund, Sweden: Bloms i Lund Tryckeri AB, 2003.