Investigaton of computational load and parallel computing of Volterra series method for frequency analysis of nonlinear systems

S. KAÇAR[a,*], İ. ÇANKAYA[b], A. F. BOZ[c]
[a,c]*Department of Electrical and Electronic Engineering, Technology Faculty, Sakarya University, Sakarya, Turkey*
[b]*Department of Electronic and Communication Engineering, Yıldırım Beyazıt University, Ankara, Turkey*

In this study, one of the prominent and the basic analytical methods for the frequency analysis of non-linear systems, Volterra Series method is discussed. Then the computational load of the method and the symmetrization process is investigated. For this, a third-order non-linear system model has been used and the frequency response of the system has been obtained. The investigation is performed by detecting the periods of every step of the method and the symmetrization process. The obtained results are presented in tables and graphs. The factors which affect the performance of the method are identified. Finally, parallel computing of the method has been realized on 8-thread computer by using MATLAB® parallel processing toolbox and the results (speedup and efficiency) have been interpreted.

(Received November 11, 2013; accepted May 15, 2014)

Keywords: Non-linear systems, Volterra series, Frequency response functions, Symmetrization process, Computational load, Parallel computing

1. Introduction

The most realistic way to get information about systems is to implement the system (physically), to interpret obtained results. However, because of some reasons such as inappropriate environmental conditions, dangerous processes, high cost and time loss, systems cannot be implemented physically and experimental studies cannot be performed. Therefore, the most rational and easy way to get information about systems is modelling the relationship between the input and output of the systems by mathematical expressions, analyzing systems with the appropriate methods and so that the desired results can be obtained. Mathematical models of the systems depend on the characteristics of elements composing the system and environmental conditions. For that reason, there are a wide variety of mathematical models. According to mathematical models, systems can be classified as shown in Fig. 1. In Fig. 1, system classes which are coloured grey are attended in this work. In the figure, dashed lines mean that the classification on the branch is same as other branch's classification.

Real systems are modelled completely with a non-linear structure. Non-linear components are capable of accelerating, attenuating, reinforcing or retarding in the non-linear systems. The non-linear terms in the non-linear system models can be exponential, radical, denominator, multiplied with each other or absolute expressions of variables. Unlike linear systems, various behaviours such as jump phenomena, bifurcation, and chaos can be seen in non-linear systems.

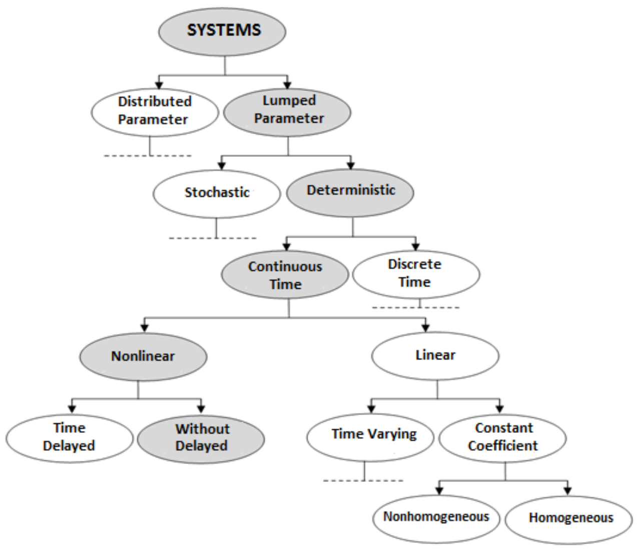

Fig. 1. Classification of Systems [1].

There are several methods for modelling and analysis of the non-linear systems. Approaches as Volterra, bilinear, perturbation methods can be used for modelling of the systems. There are many methods used in time and frequency domains for the analysis [2]. Methods such as perturbation method, averaging method, multiple-time scale method can be referred as the methods applied in the time domain. Methods such as Volterra Series Method, describing functions method, generalized harmonic balance method can be examplified as methods applied in the frequency domain.

It is thought that using the time domain methods is easier for system analysis. However, system behaviours cannot be fully analysed in time domain. Therefore, for the system analysis, especially for non-linear system analysis, it is more useful to use the frequency domain methods.

The Volterra Series Method (frequency response function method), the generalized harmonic balance method, the describing functions method, which are most commonly used the frequency domain methods of non-linear system analysis, are based on the Volterra theory. The most basic and general method of these methods is The Volterra Series Method.

The Volterra Series Method is first studied by Vito Volterra. He conducted the first study on Volterra Series [3]. Volterra showed that a non-linear system can be defined and output of a single input analytical system can be explained with infinite Volterra Series which is named with his name. After Volterra, Brilliant showed that non-linear memoryless systems can be modelled and analyzed by the Volterra Series Method [4]. In Bedrosian and Rice's study, The Volterra Series were moved to the frequency domain by using the Fourier transform and the harmonic probing algorithm was designed so that exponential input method was produced and communication systems were analyzed [5]. Billing and Tsang also adapted this algorithm to discrete. This topic was covered in three separate studies [6, 7, 8]. By Billings and Peyton Jones, for the direct production of the generalized frequency response functions obtained by using Volterra Series from non-linear difference equations and non-linear integro-differential equations, recursive algorithms were developed [9,10]. In another study a new error-free algorithm for the determination of Volterra kernels of discrete non-linear systems was presented [11]. For the non-linear system analysis which uses Volterra Series theory, some truncations should be done. Billings and Lang handled this context and developed an efficient algorithm for determining the truncations and useful terms for Volterra Series analysis [12]. A new procedure, which processes Volterra kernels by recursive algorithms, was presented for determining the parameters by Chatterjee and Vyas [13]. In addition, the books written by Schetzen [14] and Rugh [15] may be referred as key resources related to The Volterra Series method. In the study of Peyton Jones, which is the main reference of this present work, a simplified algorithm for the computation of the frequency response functions was produced [16]. The goal of the algorithm is to obtain *n*th order frequency response functions (FRF) without using recursive functions of the traditional method. In the new method, only lower order FRFs which can contribute to *n*th order FRF are produced. The algorithm of the study has been used in a .NET based web interface for non-linear system analysis by Kaçar and Çankaya [17]. For non-linear Volterra systems including a non-linear state equation and a non-linear output function, frequency response functions and characteristics were developed and discussed by Jing *et al.* [18]. Jing and Lang developed a new function based on parametric characteristics of generalized FRFs for Volterra systems modelled by Narx (non-linear auto regressive with exogenous input) model [19]. In another study, an approach for deriving the Volterra Series expansion for the multi-linear discrete system involving input - output coupling terms was developed [20].

This paper is organized as follows. Volterra model structure, time domain and frequency domain representations of non-linear systems are explained in the next section. In the third section, old and new approaches of harmonic probing method developed by Peyton Jones are described. Then the computational load of the new approach which expressed as simplified method in [16] is examined in terms in the fourth section. In the fifth section, parallel computing of the method is performed by using MATLAB® parallel processing toolbox and the results have been presented as graphics. The final section includes conclusions and suggestions.

2. Identification of non-linear systems in time and frequency domains with volterra series

In the time domain, the relationship between inputs and outputs of the non-linear systems can be defined as follows [3].

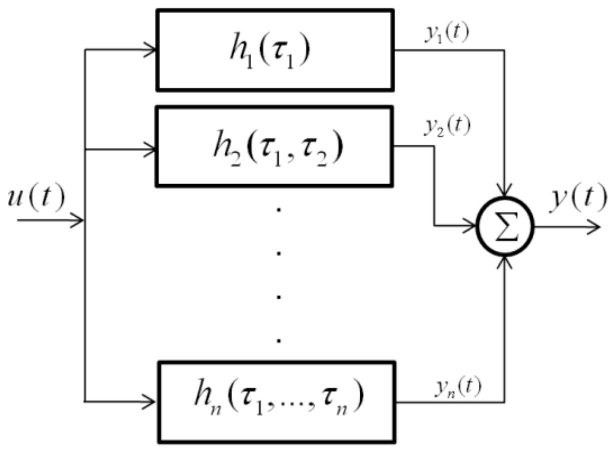

Fig. 2. Volterra model structure.

It is seen from Fig. 2 that the non-linear system is composed of *N* parallel subsystems from $h_1(\tau_1)$ to $h_n(\tau_1,...,\tau_n)$. The input signal, *u*(*t*), is applied to each subsystem and the output signal, $y_n(t)$, is obtained from each subsystem. At the end, outputs of subsystems are added and the output of entire system, *y*(*t*), is generated. This structure can be defined as a mathematical expression in Equation (1).

$$y(t)=\sum_{n=1}^{N} y_n(t) \tag{1}$$

Each subsystem output is defined in the time domain as Equation (2).

$$y_n(t) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} h_n(\tau_1, \dots, \tau_n) \prod_{i=1}^{n} u(t - \tau_i) d\tau_i \quad , n > 0 \tag{2}$$

where $h_n(\tau_1, \dots, \tau_n)$ defines *n*th order impulse response function of *n*th order subsystem. Because of multi-dimensional structure of right side of Equation (2), the multi-dimensional Fourier transformation should be applied to Equation (2) for transformation to frequency domain. To do so, $y_n(t)$ is formed as Equation (3) and the constraint in Equation (4) must be assured.

$$y_n(t) = y_n(t_1, \dots, t_n)|_{t_1 = \dots = t_n = t} \tag{3}$$

$$\omega = \sum_{i=1}^{n} \omega_i \tag{4}$$

where ω_i defines of input harmoic frequencies. If the multi-dimensional Fourier transformation is applied to Equation (2) under the constraints of Equation (3) and (4), $y_n(t)$ function is obtained in the frequency-domain as $Y_n(j\omega)$, in Equation (5).

$$Y_n(j\omega) = \frac{1/\sqrt{n}}{(2\pi)^{n-1}} \int_{\omega = \sum_{i=1}^{n} \omega_i} H_n(j\omega_1, \dots, j\omega_n) \prod_{i=1}^{n} U(j\omega_i) d\omega_1, \dots, d\omega_{n-1} \tag{5}$$

In Equation (5), $U(j\omega)$ is termed as Fourier transform of the input and $H_n(j\omega_1, \dots, j\omega_n)$ is termed as *n*th order Frequency Response Function (FRF). As a result, the output of the system in the frequency-domain can be written as the sum of all output components as in Equation (6).

$$Y(j\omega) = \sum_{n=1}^{N} A^n Y_n(j\omega) \tag{6}$$

3. Computation of higher order frequency response functions with harmonic probing method

The most common methods of mathematical modelling of systems are methods which use parametric approaches such as differential or difference equations. One of them which is used to represent continuous-time non-linear systems (without delayed) is NDE (Non-linear Differential Equations) model.

$$\sum_{m=1}^{M} \sum_{p=0}^{m} \sum_{l_1, l_{p+q}=0}^{L} c_{p,q}(l_1, \dots, l_{p+q}) \prod_{i=1}^{p} \frac{d^{(l_i)} y(t)}{dt^{(l_i)}} \prod_{i=p+1}^{p+q} \frac{d^{(l_i)} u(t)}{dt^{(l_i)}} = 0 \tag{7}$$

Where l_i denotes the order of the derivative, $c_{p,q}(l_1, \dots, l_{p+q})$ terms coefficients of the model, *p* number of the output components, *q* number of the input components of the terms, *M* is maximum order of nonlinearity and *m* is maximum order of input nonlinearity. As an example, a mechanical system modelled with a differential equation is given in Fig. 3 and Equation (8) [21].

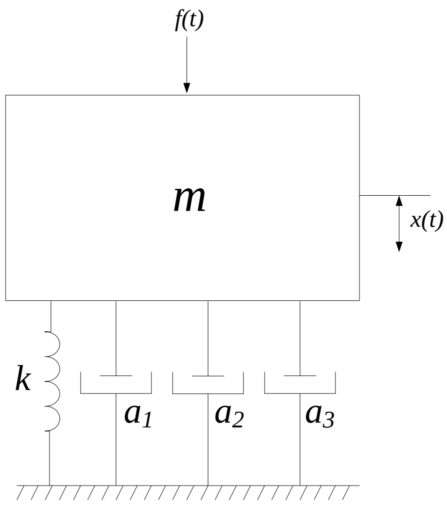

Fig. 3. Mechanical system model.

$$m\frac{d^2x(t)}{dt^2} + kx(t) + a_1\frac{dx(t)}{dt} + a_2\left(\frac{dx(t)}{dt}\right)^2 + a_3\left(\frac{dx(t)}{dt}\right)^3 - f(t) = 0 \tag{8}$$

The terms coefficients of this differential equation are determined according to the NDE model as Equation (9).

$$c_{1,0}(2) = m, c_{1,0}(1) = a_1, c_{1,0}(0) = k, c_{0,1}(0) = -1,$$
$$c_{2,0}(1,1) = a_2, c_{3,0}(1,1,1) = a_3 \text{ others } c_{p,q} = 0 \tag{9}$$

In this presented study, the harmonic probing algorithm, which is based on The Volterra Series, has been used for the frequency analysis of the non-linear system in Equation (8). The harmonic probing algorithm was developed by Billings and Peyton Jones in works realized in 1989 and 1990 [9, 10]. With this method, the system FRFs are obtained using the system's parameters. Thus, the frequency-domain behaviours of the systems can be analyzed.

Terms of the systems and their contributions to *n*th order FRF are examined into three groups in the harmonic probing method and the exponential input method: non-linear terms containing only input component ($H_{n_u}(.)$), non-linear terms containing only the output ($H_{n_y}(.)$) and non-linear terms containing input and output components together ($H_{n_{uy}}(.)$) [10]. This method is used for expressing *n*th order FRF an asymmetric structure as follows.

$$H_n^{asym}(j\omega_1,\ldots,j\omega_n)=$$
$$\frac{-\left[H_{n_u}(j\omega_1,\ldots,j\omega_n)+H_{n_{uy}}(j\omega_1,\ldots,j\omega_n)+H_{n_y}(j\omega_1,\ldots,j\omega_n)\right]}{\sum_{l_1=0}^{L} c_{1,0}(l_1)(j\omega_1+\ldots+j\omega_n)^{l_1}} \tag{10}$$

Contributions of $H_{n_u}(.)$, $H_{n_y}(.)$ and $H_{n_{uy}}(.)$ functions to nth order FRF are defined with following formulas.

$$H_{n_u}(j\omega_1,\ldots,j\omega_n)=\sum_{l_1,l_n=0}^{L} c_{0,n}(l_1,\ldots,l_n)\prod_{i=1}^{n}(j\omega_i)^{l_i} \tag{11}$$

$$H_{n_y}(j\omega_1,\ldots,j\omega_n)=\sum_{p=2}^{n}\sum_{l_1,l_n=0}^{L} c_{p,0}(l_1,\ldots,l_p)H_{n,p}(j\omega_1,\ldots,j\omega_n) \tag{12}$$

$$H_{n_{uy}}(j\omega_1,\ldots,j\omega_n)=\sum_{q=1}^{n-1}\sum_{p=1}^{n-q}\sum_{l_1,l_n=0}^{L} c_{p,q}(l_1,\ldots,l_{p+q})$$

$$H_{n-q,p}(j\omega_1,\ldots,j\omega_{n-q})\prod_{i=n-q+1}^{p+q}(j\omega_i)^{l_i} \tag{13}$$

If the above equations are examined carefully, it can be said that the terms which exist only the nth order non-linear input component can contribute to only the nth order FRF. Contributions of the terms containing output component to *n*th order FRF are determined by the function expressed with $H_{n,p}(\cdot)$. This function can be produced with two different algorithms [16, 17]. The first of these is an recursive algorithm [10].

$$H_{n,p}^{asym}(j\omega_1,\ldots,j\omega_n)=$$
$$\sum_{i=1}^{n-p+1} H_i(j\omega_1,\ldots,j\omega_i)H_{n-i,p-1}(j\omega_{i+1},\ldots,j\omega_n)(j\omega_1+\ldots+j\omega_i)^{l_p} \tag{14}$$

$$H_{n,1}^{asym}(j\omega_1,\ldots,j\omega_n)=H_n(j\omega_1,\ldots,j\omega_n)(j\omega_1+\ldots+j\omega_i)^{l_p} \tag{15}$$

Because the algorithm given in Equation (14) and (15) is recursive, the algorithm has computationally intensive for production of higher order FRFs [16, 17]. The new algorithm of PEYTON JONES in [16] was developed in a more simple structure to eliminate the disadvantage of the old algorithm. In the new algorithm, features of the exponential input method were used as in the old algorithm and $H_{n,p}(\cdot)$ function was produced for each term of the system model.

In spite of the old algorithm, the lower order FRFs which can be used for computing the nth order FRF, are determined with lower process load in the simplified algorithm. Then, $H_{n,p}(\cdot)$ function is produced and it is substituted into Equation (12) or (13). The flowchart in Figure 4 explains how the simplified algorithm works. The process in the first step of the flowchart determines combinations of the lower order FRFs which can be contribute to *n*th order FRF and whose orders summation is equal to *n* ($\sum_{i=1}^{p}\gamma_i=n$). γ_i expresses the orders of FRFs which can be contribute to *n*th order FRF. In the second step, input harmonics in set $\{\omega_1,\ldots,\omega_n\}$ are grouped according to produced combinations. In the third step, $f_y(\mathbf{w}_{\gamma_1},\ldots,\mathbf{w}_{\gamma_p})$ defines terms which must be in $H_{n,p}^{asym}(\cdot)$ function except the lower order FRFs. $\mathbf{w}_{\gamma_i}$ defines a group of the input harmonics which is determined with $\{\gamma_1,\ldots,\gamma_p\}$ combination and has γ_i frequency components. In this step, $f_y(\mathbf{w}_{\gamma_1},\ldots,\mathbf{w}_{\gamma_p})$ is computed for all permutations of each combinations produced in former steps. In the last step, desired $H_{n,p}^{asym}(\cdot)$ function is computed with obtained terms and the simplified algorithm is finalized.

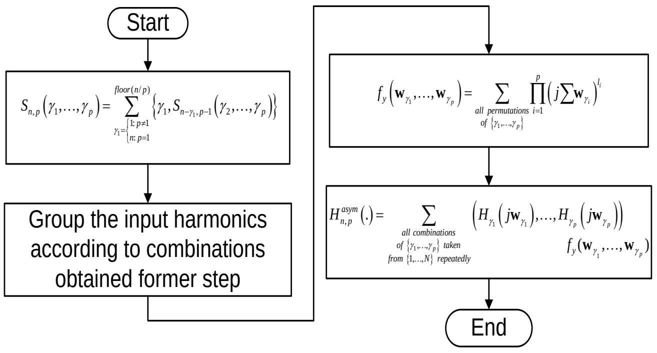
Start
$S_{n,p}(\gamma_1,\ldots,\gamma_p)=\sum_{\gamma_1=\begin{cases}1: p\neq 1\\ n: p=1\end{cases}}^{floor(n/p)}\{\gamma_1,S_{n-\gamma_1,p-1}(\gamma_2,\ldots,\gamma_p)\}$
Group the input harmonics according to combinations obtained former step
$f_y(\mathbf{w}_{\gamma_1},\ldots,\mathbf{w}_{\gamma_p})=\sum_{\substack{all\ permutations\\ of\ \{\gamma_1,\ldots,\gamma_p\}}}\prod_{i=1}^{p}(j\sum\mathbf{w}_{\gamma_i})^{l_i}$
$H_{n,p}^{asym}(.)=\sum_{\substack{all\ combinations\\ of\ \{\gamma_1,\ldots,\gamma_p\}\ taken\\ from\ \{1,\ldots,N\}\ repeatedly}}(H_{\gamma_1}(j\mathbf{w}_{\gamma_1}),\ldots,H_{\gamma_p}(j\mathbf{w}_{\gamma_p}))\ f_y(\mathbf{w}_{\gamma_1},\ldots,\mathbf{w}_{\gamma_p})$
End

Fig. 4. Simplified Volterra Series method.

After computation of $H_{n,p}(\cdot)$ functions, the method is continued to apply. The desired (*n*th order) asymmetric FRF ($H_n^{asym}(\cdot)$) is computed by using Equation (10), (11), (12) and (13). In this step, there is an important problem because of $H_n(j\omega_1,\ldots,j\omega_n)$ function. In FRFs, if order of the input harmonics changes, the FRF changes. But this change may not effect the output function ($y_n(t)$) [14]. For eliminating this problem, the symmetric FRF ($H_n^{sym}(.)$) is used and a better analysis can be performed. The result of $H_n^{sym}(.)$ function is independent from the order of variables. The symmetric function is formulated as Equation (16) [16, 17].

$$H_n^{sym}(j\omega_1,\ldots,j\omega_n)=\frac{1}{n!}\sum_{\substack{all\ permutations\\ of\ \{\omega_1,\ldots,\omega_n\}\ set}} H_n^{asym}(j\omega_1,\ldots,j\omega_n) \tag{16}$$

For example, in the symmetrization process of 5th order asymmetric FRF, $n!=120$ permutations of $n=5$ input harmonics are generated firstly as Table 1.

Table 1. Generation of permutations for a 5th order FRF.

$H_n^{asym}(\cdot)$		Harmonics				
		1.	2.	3.	4.	5.
Permutations	1.	$j\omega_1$	$j\omega_2$	$j\omega_3$	$j\omega_4$	$j\omega_5$
	2.	$j\omega_2$	$j\omega_1$	$j\omega_3$	$j\omega_4$	$j\omega_5$
	3.	$j\omega_3$	$j\omega_2$	$j\omega_1$	$j\omega_4$	$j\omega_5$
	⋮	⋮	⋮	⋮	⋮	⋮
	118.	$j\omega_5$	$j\omega_4$	$j\omega_2$	$j\omega_3$	$j\omega_1$
	119.	$j\omega_5$	$j\omega_4$	$j\omega_3$	$j\omega_1$	$j\omega_2$
	120.	$j\omega_5$	$j\omega_4$	$j\omega_3$	$j\omega_2$	$j\omega_1$

After obtaining the permutations, they are applied to the FRF one by one and all results are added. Finally the summation is divided by the number of permutations and the symmetric function is obtained as Equation (17).

$$H_5^{sym}(j\omega_1,\cdots,j\omega_5)=\frac{1}{5!}\sum_{\substack{all\ permutations\ of\\ \{\omega_1,\omega_2,\omega_3,\omega_4,\omega_5\}\ set}} H_5^{asym}(j\omega_1,j\omega_2,j\omega_3,j\omega_4,j\omega_5)$$

$$=\frac{1}{120}\times\begin{pmatrix} H_5^{asym}(j\omega_1,j\omega_2,j\omega_3,j\omega_4,j\omega_5)+H_5^{asym}(j\omega_2,j\omega_1,j\omega_3,j\omega_4,j\omega_5) \\ +H_5^{asym}(j\omega_3,j\omega_2,j\omega_1,j\omega_4,j\omega_5)+\cdots\cdots\cdots\cdots \\ +\cdots\cdots\cdots\cdots+\cdots\cdots\cdots\cdots \\ +\cdots\cdots\cdots\cdots+H_5^{asym}(j\omega_5,j\omega_4,j\omega_1,j\omega_3,j\omega_2) \\ +H_5^{asym}(j\omega_5,j\omega_4,j\omega_3,j\omega_1,j\omega_2)+H_5^{asym}(j\omega_5,j\omega_4,j\omega_3,j\omega_2,j\omega_1) \end{pmatrix} \tag{17}$$

As an example, the parameters of the system model in Equation (8) are determined as in Equation (18) and the 5th symmetric FRF can be used for getting results.

$$m=240 \text{ kg},\ k=16000 \text{ N/m},\ a_1=296 \text{ Ns/m},\ a_2=2000 \text{ Ns/m},\ a_3=800 \text{ Ns/m} \tag{18}$$

The graphics in Figs. 5 and 6 are the magnitude and the phase response of the 5th symmetric FRF of the system for the parameters in Equation (18). Since the FRF is 5th, there are five input harmonics according to the method. For that reason the graphics are 3-D. Because of dimensions limitation, some of the harmonics are chosen at same frequency value for visualisation the results. In the graphics, the x axis is defined as $\omega_1=\omega_2=\omega_3=\omega_4$ and the y axis is defined as ω_5. The values of x and y axis are between 0 and 20 rad/sec. The z axis contains the results. Figs. 7 and 8 are the contour plots of Figs. 5 and 6 that they can be used for more detailed examination.

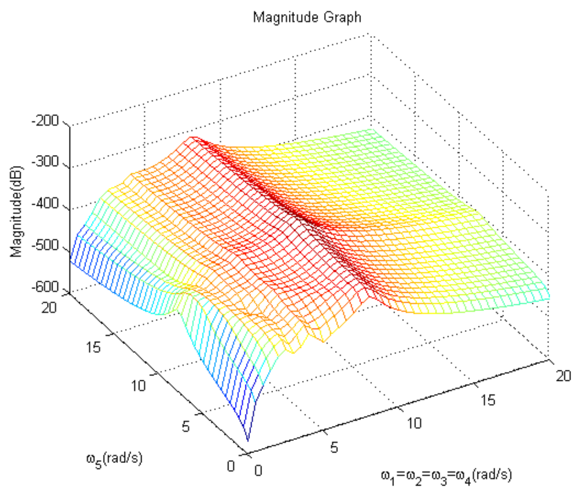

Fig. 5. Magnitude graphic of 5th order FRF.

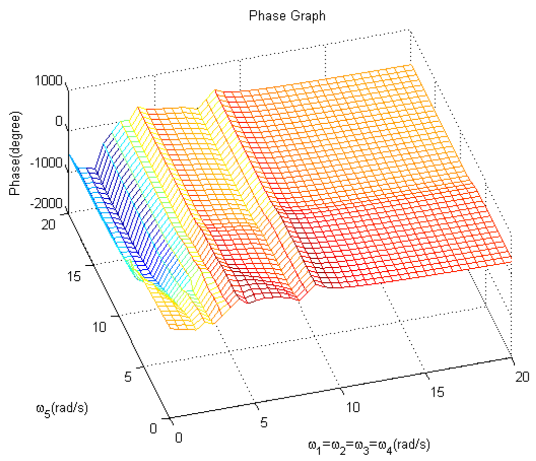

Fig. 6. Phase graphic of 5th order FRF.

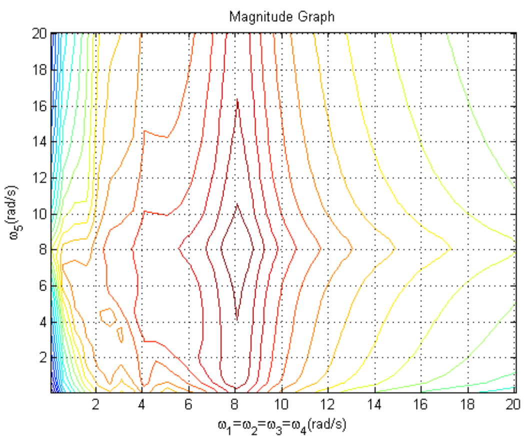

Fig. 7. Contour plot of magnitude graphic of 5th order FRF.

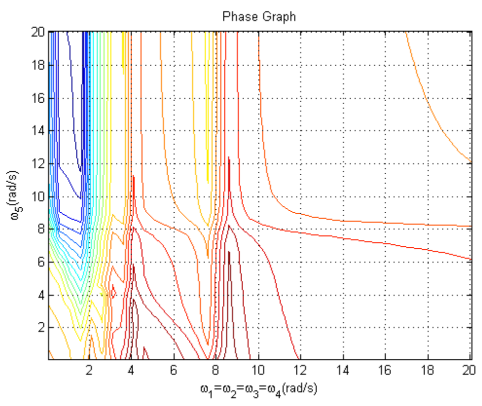

Fig. 8. Contour plot of phase graphic of 5th order FRF.

Sample applications of the old and new algorithms presented in [16] can be seen broadly in the study of KAÇAR [17].

4. Investigation of computational load

In the Volterra Series method, the most important factors, which effect computational load and realization time of analysis, are the order of desired FRF and the number of output components in analyzed term. If the order of terms and FRFs increases, computational load and realization time of analysis increases too and this situation is normal. Spectacular feature of the situation is the logarithmic increment of the number of combinations and permutations, as in Table 2. Entire numbers of combinations and permutations, which can contribute *n*th order FRF, are given in Table 2 for all $H_{n,p}^{asym}(\cdot)$ functions which can consist up to *n*=10. The numbers of combinations and permutations are related to the desired FRF order (*n*) and the number of output components of analyzed term (*p*). The numbers in Table 2 are obtained as results of former three steps of the simplified algorithm.

When the analysis is performed by using the simplified algorithm, the process load (number of combinations and permutations) may be very low or high according to values of *n* and *p*, as seen in Table 2. Especially, for higher order FRFs, there are too many permutations remarkably for some *p* numbers. In Table 2, all numbers of combinations and permutations are 1, if *p* is equal to *n* or 1 for all *n* values. When *n* increases, number of combinations and permutations increases also. The most remarkable point in Table 2 is that number of permutations is maximum value, when *p* is equal to *n/2* for even *n* numbers and *p* is equal to *up*(*n/2*) for odd *n* numbers. For other *p* values, numbers of combination and permutations are lower. For example, the highest combination and permutation values (7 and 126) consist for *n*=10 and *p*=5. Whereas numbers of permutation is equal to 9 for *p*=9 or *p*=2. As result, the process load relates *n* and *p* values and it also relates the difference between *n* and *p*. Process time of each step and total process time of the simplified algorithm is shown in Table 3. The time periods in Table 3 are obtained by using a computer which has a configuration as seen in Table 4.

Table 2. The numbers of combinations and permutations for all $H_{n,p}^{asym}(\cdot)$ functions up to n=10.

n,p	The numbers of combinations	The numbers of permutations	n,p	The numbers of combinations	The numbers of permutations
1,1	1	1	8,1	1	1
2,1	1	1	8,2	4	7
2,2	1	1	8,3	5	21
3,1	1	1	8,4	5	35
3,2	1	2	8,5	3	35
3,3	1	1	8,6	2	21
4,1	1	1	8,7	1	7
4,2	2	3	8,8	1	1
4,3	1	3	9,1	1	1
4,4	1	1	9,2	4	8
5,1	1	1	9,3	7	28
5,2	2	4	9,4	6	56
5,3	2	6	9,5	5	70
5,4	1	4	9,6	3	56
5,5	1	1	9,7	2	28
6,1	1	1	9,8	1	8
6,2	3	5	9,9	1	1
6,3	3	10	10,1	1	1
6,4	2	10	10,2	5	9
6,5	1	5	10,3	8	36
6,6	1	1	10,4	9	84
7,1	1	1	10,5	7	126
7,2	3	6	10,6	5	126
7,3	4	15	10,7	3	84
7,4	3	20	10,8	2	36
7,5	2	15	10,9	1	9
7,6	1	6	10,10	1	1
7,7	1	1			

Table 3. Obtained periods of all $H^{asym}_{n,p}(\cdot)$ *functions up to n=10.*

N,p	Comb. Time (sec.)	Group Time (sec.)	*Fy* Func. Time (sec.)	Total Time (sec.)	n,p	Comb. Time (sec.)	Group Time (sec.)	*Fy* Func. Time (sec.)	Total Time (sec.)
1,1	0.00006	0.00034	0.00032	0,00072	8,1	0.00008	0.00095	0.0010	0,00203
2,1	0.00009	0.00057	0.00049	0,00115	8,2	0.00051	0.0041	0.0117	0,01631
2,2	0.00036	0.00060	0.00059	0,00155	8,3	0.00082	0.0127	0.0301	0,04362
3,1	0.00009	0.00063	0.00067	0,00139	8,4	0.0013	0.0061	0.0576	0,065
3,2	0.00021	0.00069	0.0013	0,0022	8,5	0.0009	0.0049	0.0708	0,0766
3,3	0.00048	0.0019	0.00098	0,00336	8,6	0.00087	0.0051	0.0393	0,09117
4,1	0.00007	0.00073	0.00067	0,00147	8,7	0.00081	0.0017	0.0144	0,01691
4,2	0.00042	0.0013	0.0022	0,00392	8,8	0.0008	0.0017	0.0022	0,0047
4,3	0.00037	0.00096	0.0028	0,00413	9,1	0.00009	0.0024	0.0012	0,00369
4,4	0.00052	0.0010	0.0013	0,00282	9,2	0.00052	0.0101	0.0126	0,02322
5,1	0.00009	0.00083	0.00088	0,0018	9,3	0.0011	0.0137	0.0459	0,0607
5,2	0.0004	0.0016	0.0035	0,0055	9,4	0.0013	0.0072	0.0864	0,0949
5,3	0.00056	0.0019	0.0084	0,01086	9,5	0.0013	0.0070	0.1228	0,1311
5,4	0.00050	0.0012	0.0062	0,0079	9,6	0.0011	0.0046	0.1288	0,1345
5,5	0.00055	0.0012	0.0014	0,00315	9,7	0.00095	0.0034	0.0533	0,05765
6,1	0.00009	0.00092	0.00098	0,00199	9,8	0.00087	0.0020	0.0283	0,03117
6,2	0.00051	0.0028	0.0052	0,00851	9,9	0.0010	0.0021	0.0028	0,0059
6,3	0.00066	0.0029	0.0105	0,01406	10,1	0.00009	0.0011	0.0012	0,00239
6,4	0.00064	0.0026	0.0124	0,01564	10,2	0.0016	0.0068	0.0187	0,0271
6,5	0.00056	0.0012	0.0078	0,00956	10,3	0.0012	0.0125	0.0487	0,0624
6,6	0.0007	0.0013	0.0018	0,0038	10,4	0.0016	0.0115	0.1203	0,1334
7,1	0.00005	0.00088	0.00099	0,00192	10,5	0.0018	0.0101	0.2336	0,2455
7,2	0.00039	0.0064	0.0057	0,01249	10,6	0.0015	0.0076	0.3160	0,3251
7,3	0.0008	0.0050	0.0298	0,0356	10,7	0.0011	0.0069	0.2559	0,2639
7,4	0.00075	0.0042	0.0435	0,04845	10,8	0.0010	0.0036	0.0773	0,0819
7,5	0.00065	0.0072	0.0300	0,03785	10,9	0.00097	0.0022	0.0231	0,02627
7,6	0.00076	0.0014	0.0107	0,01286	10,10	0.0011	0.0023	0.0071	0,0105
7,7	0.00075	0.0015	0.0020	0,00425					

Table 4. Configuration of used computer.

Processor Model	Intel(R) Core(TM) i7 CPU 950
Num. of Threads	8 threads
Processor Freq.	3.07 GHz
Ram Capacity	4.00 GB
Operating System	Win 7 64 Bit

The results in Table 3 support the explanations of above related to the numbers of combinations and permutations according to *n* and *p* values and their process load. In order to understand better the results are presented graphically in Fig. 9.

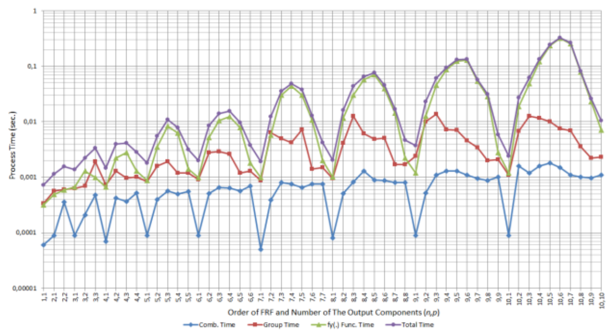

Fig. 9. Obtained periods of all $H^{asym}_{n,p}(\cdot)$ *functions up to n=10.*

As seen in Fig. 9, while the simplified algorithm is running, major amount of the time is spent in step of *fy*(.) function which is the third step of the algorithm. The most important and significant point of the graphic is that the spent time is significantly related to difference between FRF order and the number of output components more than FRF order. As mentioned above, the maximum time for each value of *n* is composed around *p* = *n* / 2.

Accordingly, while the systems are being analyzed by this method, it can be said that numbers of FRF orders should be chosen equal or close to numbers of output components for lower spent time periods.

The simplified algorithm examined above forms a part of the Volterra method. After *n*th order asymmetric FRF is obtained by the Volterra method, it must be symmetrized and then true results can be obtained. So that the Volterra method can be separated in two parts. The first part is obtaining the *n*th order asymmetric FRF, the second part is symmetrization of the FRF and obtaining the results. When higher order FRFs are considered, the first part spends much less time than the second part. This seems clearly from the results in Table 5 which were obtained from the system model in Equation (8) for m=240 kg, k=16000 N/m, a_1=296 Ns/m, a_2=296 Ns/m, a_3=296 Ns/m (parameter values from [22]), 8 frequency sets up to n=8.

Table 5. Obtained time periods of FRF and the results up to n=8 for the system model in Equation (8).

n	FRF time (sec)	Symmetrization and result time (sec)	*n*	FRF time (sec)	Symmetrization and result time (sec)
1	0.00167668	0.0018734	5	0.015279	0.84472
2	0.0036378	0.0085667	6	0.033202	6.7618
3	0.0073273	0.032149	7	0.049726	60.6733
4	0.010497	0.14523	8	0.062537	606.8232

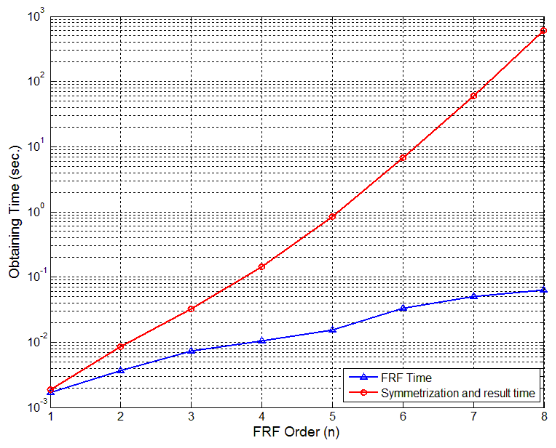

Fig. 10. Obtained time periods of FRF and the results up to n=8 for the system model in Equation(8).

It is clear from Fig. 10 that if order (n) of the desired FRF increases, process time also increases. However, the difference between obtained time periods of the FRFs and time spent for symmetrization and the results increases logarithmically with increasing of n. For lower-order FRFs, symmetrization and obtaining the results take less time. When the order increases, they begin to take a lot of time for higher order FRFs. The reason of this is logarithmical increasing of permutations which is a result of increasing of the order of FRF in symmetrization process given in Equation (10). Therefore, the process load and time spent for obtaining the results increase logarithmically. For this reason, while an analysis of a non-linear system is performed by The Volterra Series method with the simplified algorithm, choosing the order of desired FRF equal to order of the highest order term is more useful for obtaining the results. On the other hand, decreasing the symmetrization and result time is more efficient way to decrease the total time of the method.Using parallel computing methods is an appropriate way for providing time saving.

5. Parallel computing of the method

Today, majority of the computers have multiple processors, multi-thread CPUs or GPUs. Traditional programming approaches are not suitable for using the hardware. For this reason, parallel computing methods are used to benefit the capacity of the computers fully and to perform processes faster. There are numerous studies about parallel computing in the field of nonlinear systems analysis and simulation [23 - 27].

One of the main platforms of parallel programming is the multi-thread computers. Today, a PC which has an average configuration, has a multi-thread (2, 4, 6, 8, etc. threads) CPU. For this reason, a complex algorithm required high processing capacity and speed as Volterra Series Method, especially its symmetrization process, should be programmed in parallel to use all threads in CPU. The architecture for used computer with a multi-thread (N threads) CPU can be visualized generally as in Fig. 11. If the method is programmed in parallel and run in a computer with a multi-thread CPU which has N threads, the method is decomposed in N threads. Thus, the computer uses the CPU full capacity and the process is finished in shorter time.

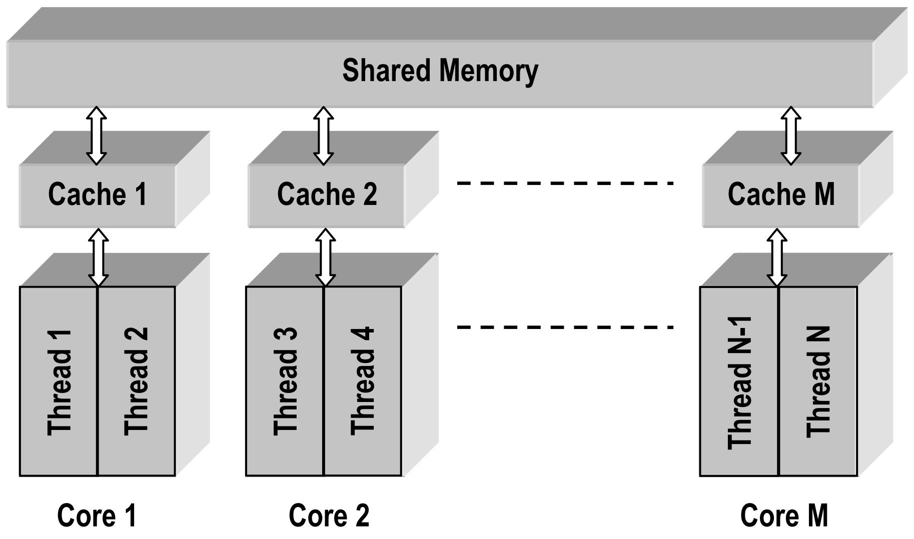

Fig. 11. The architecture of the used computer with a multi-thread CPU.

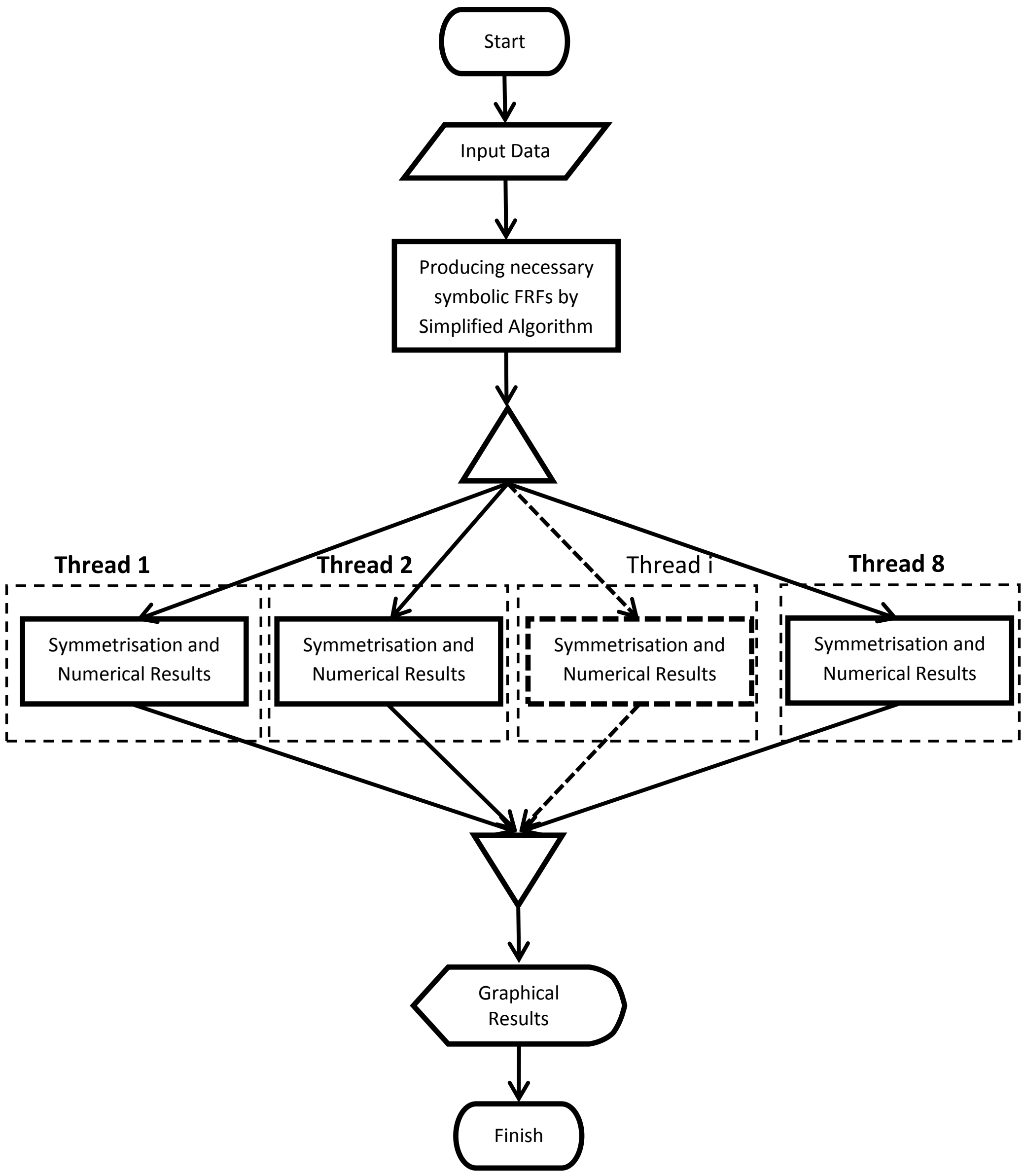
Start
Input Data
Producing necessary symbolic FRFs by Simplified Algorithm
Thread 1
Thread 2
Thread i
Thread 8
Symmetrisation and Numerical Results
Symmetrisation and Numerical Results
Symmetrisation and Numerical Results
Symmetrisation and Numerical Results
Graphical Results
Finish

Fig. 12. The flowchart of the parallelized method.

The method has been coded in data-parallel by using MATLAB program and its parallel processing toolbox. This toolbox provides programming in all parallel approaches (task-parallel, data-parallel, distributed computation, CUDA and multi-thread programming) [28]. In this study, for data-parallel programming, parallel "*for*" (*parfor*) loops provided by MATLAB parallel processing toolbox have been used with the computer introduced in Table 4 for computing numerical results of the system given in Equation (8) . The parallelization of the method is shown in Fig. 12. As seen in Fig. 12, symmetrisation and computing numerical results step is parallelized and same tasks for different data are processed concurrently by 8 threads in CPU.

Table 6. Process times obtained after parallel computing (in seconds).

		Number of threads							
		sequental	2	3	4	5	6	7	8
Order of FRF	1	0.0018734	0.10931	0.12956	0.13632	0.16119	0.20276	0.22	0.23824
	2	0.0085667	0.16807	0.21187	0.21576	0.23543	0.27783	0.31698	0.34043
	3	0.032149	0.17643	0.21395	0.22001	0.24021	0.2789	0.31784	0.38403
	4	0.14523	0.23629	0.2598	0.24593	0.28585	0.31233	0.34678	0.38868
	5	0.84472	0.56451	0.47831	0.43685	0.42302	0.49162	0.57606	0.54123
	6	6.7618	3.4361	2.6603	1.9168	2.1936	2.3569	2.3526	1.8041
	7	60.6733	29.4879	22.1357	15.3056	16.412	18.3519	19.0018	13.2724
	8	606.8232	296.2393	228.5412	153.9728	168.0649	185.03	199.2235	130.1962

Process times obtained after parallel computing of symmetrisation and numerical results step are given in Table 6. The time results have been obtained for the FRFs up to 8th order by using 8 frequency sets (one set for each thread). These results should be used for determining the performance of the parallel computing. The performance can be assessed with criterias such as speedup and efficiency. The speedup and the efficiency are computed by using Equation 19 and 20 [29].

$$S_p(n) = T^*(n) / T_p(n) \qquad (19)$$

$$E_p(n) = S_p(n) / p \qquad (20)$$

where p is the number of threads, n is order of FRF as identifier of computation load of the process, $S_p(n)$ is speedup, $T^*(n)$ is the process time of nth order FRF as sequential algorithm performed with single thread in Table 6, $T_p(n)$ is the process time of nth order FRF processed with p threads and $E_p(n)$ is the efficiency. Fig. 13 and 14 show the speedup and the efficiency of parallel computing.

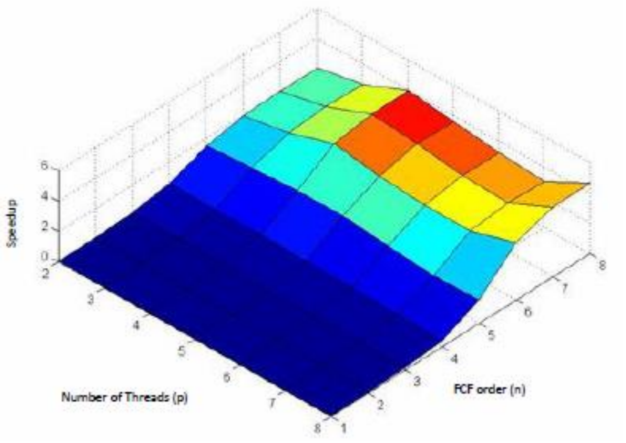

Fig. 13. Speedup of parallel computing.

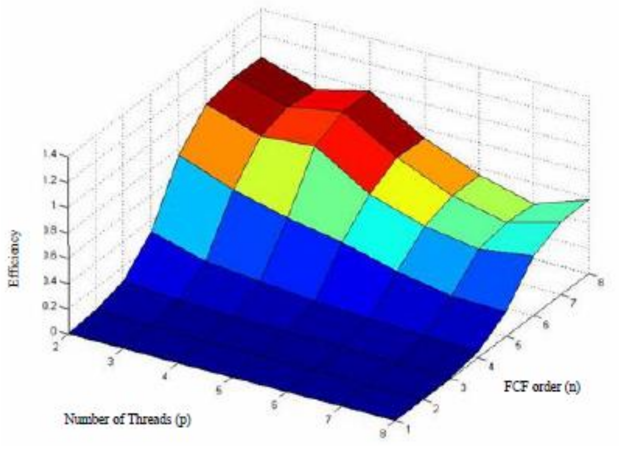

Fig. 14. Efficiency of parallel computing.

For the sample model in Equation 8, Fig. 13 and 14 shows that the multi-thread acceleration is not effective for lower order FRFs, but it is very useful for higher order FRFs bigger than 4th order. Because the parallel overheads of the parallel computing are higer than the process times of lower order FRFs. So, it can be said that the parallel computing should be applied for computing FRFs which have high complexity and computational load.

The other factor affecting the performance of the parallel computing is the number for threads. It is expected that the speedup increases proportionally with number of threads, but the parallel overhead and the unbalanced load block the proportional increase. The speedup graphic in Fig. 13 shows that the speedup increases between 2 and 4 threads and then it decreases between 4 and 7 threads. For 8 threads, the speedup is at the top point of the graphic. Although the speedup is the highest value for 8 threads, it can not be said the same for the efficiency. For 8th order FRF, the efficiency is the highest value for 2 threads and the lowest value for 7 threads and for 8 threads the efficiency is nearly 0.6. Because of the appropriate parallel overhead and computational load, processes for 2 and 4 threads are the most efficient. At the same time, the efficiency is higer than 1 for the process of 2 threads. This is called superlinearity in parallel computing [30].

All of these put forth the speedup and the efficiency of parallel computing of Volterra Series Method are releated with the complexity of the model, load balance of the threads and parallel overhead of the process. While selecting of order of FRFs and number of threads, these factors should be considered.

6. Conclusions

In this work, the simplified algorithm which was presented by PEYTON JONES was coded in MATLAB® platform and the spent time periods for each step of the algorithm and symmetrization process were determined for a third order non-linear system. So that, for the computer, the process load of the algorithm steps and symmetrization process was designated. Obtained time periods were presented in tabular form and were visualized graphically. After that, parallel computing of the method performed with multi-thread (8 threads) computer.

It is seen that major part of spent time for implementation of the method, especially for higher order FRFs, is used for symmetrization process, when obtained results are analyzed. Creation process of $f_y(.)$ function is determined as the longest process in the simplified algorithm. The common point of symmetrization and creation of $fy(.)$ function is that these processes contain a large number of permutations related to desired FRF order. According to this situation, it can be said that the process load of the new Volterra Series Method presented by PEYTON JONES consists mainly because of permutation operations. Besides, since number of permutations increases because of increasing of the FRF orders, the time periods of processes also increase. This increment, especially for symmetrization process, is logarithmic. This significantly reduces effective availability of the method for higher order systems and FRFs.

One of the ways to use the method more effectively is parallel programming and multi-thread programming is the most general and simple type of the parallel programming. In this study, Volterra Series Method is coded in parallel by using parallel "*for*" loops provided by MATLAB® Parallel Processing Toolbox for parallel computing. As a result of this, it is seen that the parallel computing is more efficient for higher order and complex FRFs. The parallel overhead and the load balancing are two factors which affect the speedup and the efficiency. So, the number of threads and the order of FRFs should be selected appropriately for a good speedup and efficiency. To get better results, Volterra Series Method can be modified as a parallel algorithm for parallel programming.

Acknowledgements

The authors gratefully acknowledge that this work was supported as a project (project no: 2013-50-02-003) by Sakarya University Scientific Research Projects Coordinatorship.

References

[1] W. L. Brogan, Modern Control Theory, Prentice-Hall, New Jersey, (1991).
[2] G. Kerschen, K. Worden, A. F. Vakakis, J. C. Golinval, Mech. Syst. Signal Pr., **20**, 505 (2006).
[3] V. Volterra, Blackie and Son Limited, London, (1930).
[4] M. B. Brilliant, Theory of the analysis of non-linear systems, Technical Report: MIT Research Lab of Electronics, Cambridge, Massachusetts, (1958).
[5] E. Bedrosian, S. O. Rice, Proc. IEE, **59**, 1688 (1971).
[6] S. A. Billings, K. M. Tsang, Mech. Syst. Signal Pr., **3**(4), 319 (1989).
[7] S. A. Billings, K. M. Tsang, Mech. Syst. Signal Pr., **3**(4), 341 (1989).
[8] S. A. Billings, K. M. Tsang, G. R. Tomlinson, Mech. Syst. Signal Pr., **4**(1), 3 (1990).
[9] J. C. Peyton Jones, S. A. Billings, Int. J. Control, **50**(5), 1925 (1989).
[10] S. A. Billings, J. C. Peyton Jones, Int. J. Control, **52**(4), 863 (1990).
[11] M. Morhac, Appl. Math. Comput., **38**(2), 87 (1990).
[12] S. A. Billings, Z. Q. Lang, Int. J. Control, **68**(5), 1019 (1997).
[13] A. Chatterjee, N. S. Vyas, J. Sound Vib., **268,** 657 (2003).
[14] M. Schetzen, The Volterra and Wiener Theories of Non-linear Systems, John Wiley and Sons, New York, (1980).
[15] W. J. Rugh, Non-linear System Theory: The Volterra/Wiener Approach, John Hopkins University Pres, Baltimore, Maryland, USA, (1981).
[16] J. C. Peyton Jones, Mech. Syst. Signal Pr., **21**, 1452 (2007).

[17] S. Kaçar, İ. Çankaya, J. Fac. Eng. Arch. Gazi Univ., **27**(4), 797 (2012).
[18] X. J. Jing, Z. Q. Lang, S. A. Billings, Int. J. Control, **81**(2), 235 (2008).
[19] X. J. Jing, Z. Q. Lang, J. Dyn. Syst. – T. Asme, **131**, 061002-1/8 (2009).
[20] S. N. Sharma, Appl. Math. Comput., **216**, 1918 (2010).
[21] S. Caffery, J. Giacomin, K. Worden, IUTAM Symposium on Identification of Mechanical Systems, Wuppertal, Germany, 1993.
[22] Z. Q. Lang, S. A. Billings, R. Yue, J. Li, Automatica, **43**, 805 (2007).
[23] L. Zeng, J. Xiang-long, C. Xiang-dong, J. Comput. Nonlinear Dynam., **2**, 366 (2007).
[24] I. Akhtar, O. A. Marzouk, A. H. Nayfeh, J. Comput. Nonlinear Dynam., **4**, 041006-1/9, (2009).
[25] Y. Özyörük, E. Alpman, V. Ahuja, L. N. Long, J. Sound Vib., **270**, 933 (2004).
[26] C. Richter, J. A. Hay, L. Panek, N. Schönwald, S. Busse, F. Thiele, J. Sound Vib., **330**, 3859 (2011).
[27] J. I. Aliaga, P. Bientinesi, D. Davidovic, E. D. Napoli, F. D. Igual, E. S. Quintana-Orti, Appl. Math. Comput., **218**, 11279 (2012).
[28] MATLAB Parallel Processing ToolboxTM 5 User's Guide, The MathWorks Inc., (2010).
[29] T. Rauber, G. Rünger, Parallel Programming For Multicore and Cluster Systems, Springer, Verlag Berlin Heidelberg, (2010).
[30] A. Grama, A. Gupta, G. Karypis, V. Kumar, "Introduction to Parallel Computing", Second Edition, Pearson Education Limited, USA, (2003).

*Corresponding author: skacar@sakarya.edu.tr