SPECIALIZED SYMBOLIC COMPUTATION FOR STEADY STATE PROBLEMS

Marcin Sowa
Silesian University of Technology, Faculty of Electrical Engineering

Abstract. *An implementation of symbolic computation for steady state problems is proposed in the paper. A mathematical basis is derived in order to specify the quantities that the implementation will concern. An analysis is performed so that an optimal algorithm can be chosen in terms of the two chosen criteria – the operation time and memory needed to store symbolic expressions. The implementation scheme of the specialized class for symbolic computation is presented with the use of a general figure and by an example. The implementation is made in C++ but the presented idea can also be applied in other programming languages that share similar properties. A program using the proposed algorithm was studied for its efficiency in terms of calculation time and memory used by symbolic expressions. This is made by comparing the calculations made by the author's program with those made by a script written in Mathematica.*

Keywords: symbolic computation, steady state, C++ implementation

SPECJALISTYCZNE OBLICZENIA SYMBOLICZNE DLA PROBLEMÓW W STANACH USTALONYCH

Streszczenie. *W artykule zaproponowano implementację obliczeń symbolicznych dla problemów w stanach ustalonych. Wyprowadzono podstawę matematyczną aby sprecyzować wielkości, dla których dokonana jest implementacja. Przeprowadzono analizę dzięki której dobrano optymalny algorytm pod względem wybranych kryteriów użyteczności tj. czasu wykonywania operacji oraz pamięci wymaganej do zapisu wyrażeń symbolicznych. Schemat implementacji specjalistycznej klasy do obliczeń symbolicznych przedstawiono za pomocą ogólnego schematu oraz z wykorzystaniem przykładu. Implementacji dokonano w języku C++ lecz ogólna idea przedstawiona jest w ten sposób, aby można ją było wykorzystać również w innych językach programowania o podobnych cechach. Wydajność programu wykorzystującego proponowany algorytm sprawdzono pod względem czasu wykonywania obliczeń i zajmowanej pamięci przez wyrażenia symboliczne. Dokonano tego poprzez porównanie obliczeń z autorskiego programu z wykonanymi przez skrypt napisany w programie Mathematica.*

Słowa kluczowe: obliczenia symboliczne, stany ustalone, implementacja w C++

Introduction

Symbolic computation is applicable in analyses that require the presentation of relationships between parameters or certain quantities and quantities that are observed. Among programs for mathematical analysis with a featured ability of symbolic computation, one can distinguish popular software such as *Mathematica* [1] or *Maple* [2]. These include a wide range of methods when it comes to operations on (or between) symbolic expressions. Symbolic computation can also be performed in *Matlab* [3], where the *Symbolic Math Toolbox* [4] was created for these purposes. The mentioned commercial software mostly has symbolic computation implemented in such ways that it is possible to present symbolic expressions in various forms i.e. they can be expanded, reduced etc.

A versatile approach such as the one proposed by the mentioned software has many benefits. However, in many cases the calculations require a long time to be completed. They also require much computer memory for complicated and large expressions to be stored.

An implementation of a specialized case of symbolic computation is considered i.e. dealing only with specific problems. Here the implementation is proposed for certain steady state problems.

The paper addresses a case where in an engineering problem, the symbolic expression form is known i.e. all symbolic expressions can be presented in the one desired form. This also means that any operations performed on the symbolic expressions will allow to obtain an expression following the same general rules. This allows to reduce the overall description of possible symbolic expressions.

The paper aims at proving that an implementation of a specialized symbolic computation algorithm can be useful in steady state problems. Results of two chosen criteria are of interest. The first is an observation of the amount of memory used in order to store the symbolic expressions. Secondly, the presented implementation is examined in terms of the calculation time required to perform an operation between symbolic expressions. Observations are made with reference to results obtained by a chosen commercial program (in this case *Mathematica*). The paper also presents the simplicity of the chosen implementation of symbolic computation.

Symbolic computation in steady states can be useful in many technical genres that contain complicated mathematical calculations e.g. the author has shown that the presented implementation can be used for certain nonlinear electromagnetic field problems [5, 6, 7].

1. Mathematical background

This section provides the mathematical basis of the selected symbolic expressions and the operations that they undergo. First, it must be mentioned that the further presented expressions concern only steady state problems. The numerical presentation of a single quantity dependent on time can be (among other forms) made with the use of a cosine Fourier series:

$$x(t) = \sum_{h=0}^{h_{\max}} B_h \cos(h\omega_0 t + \xi_h). \tag{1}$$

If a presentation using symbols must be made, the above can be written with taking into account that $B_h = B_h(\mathbf{s})$ and $\xi_h = \xi_h(\mathbf{s})$, where $\mathbf{s}$ is introduced as a vector of base symbolic expressions.

If the implementation is to be made so that every result of operations performed on the time dependent quantities of the form (1) is to yield a result following the same general rules, an assumption must be made that between two expressions of the form (1) only mathematical operations of either addition, subtraction or multiplication are performed.

Because a presentation through cosine Fourier series was chosen, it is assumed that the base symbolic expressions are those determining amplitudes and phase shifts of the Fourier series. Because various dependencies on the base expressions can be present at each harmonic, a description can be introduced, which depicts each harmonic through sub-terms. These are described by a unique j index each and together form subsequent time harmonic functions:

$$x(t) = \sum_{h=0}^{h_{\max}} \left(\sum_{j=1}^{M_h} \left(A_{h,j} a_{h,j}(\mathbf{c}) \cos(h\omega_0 t + \theta_{h,j} + f_{h,j}(\boldsymbol{\alpha})) \right) \right), \tag{2}$$

where the base symbolic expressions determining the amplitudes are:

$$\mathbf{c} = [c_1 \quad c_2 \quad \ldots \quad c_n], \tag{3}$$

and phase shifts are determined by the following vector of base symbolic expressions:

$$\boldsymbol{\alpha} = [\alpha_1 \quad \alpha_2 \quad \ldots \quad \alpha_m]. \tag{4}$$

As mentioned, the discussed symbolic expressions are either added, subtracted or multiplied hence a general form of the time function can be derived:

$$x(t)=\sum_{h=0}^{h_{\max}}\left(\sum_{j=1}^{M_h}\left(A_{h,j}\left(\prod_{i=1}^{n}c_i^{\,p_{h,j,i}}\right)\cdot\right.\right.$$
$$\left.\left.\cdot\cos\left(h\omega_0 t+\theta_{h,j}+\sum_{k=1}^{m}g_{h,j,k}\alpha_k\right)\right)\right) \tag{5}$$

where it can be observed that $p \in \mathbb{N}_0$, $g \in \mathbb{Z}$. One can notice that the form (5) is derived in such a way that when the mentioned mathematical operations (addition, subtraction or multiplication) are made on the given expressions then the result will also be obtained in the above form.

One can notice, that the number of sub-terms (given by M_h for each harmonic) represents the number of dependencies on the base symbolic expressions $\mathbf{c}$ and $\boldsymbol{\alpha}$. The smaller the M_h value, the less the amount of memory the algorithm will use to store the symbolic expressions. This also affects the time that will be used for the operations on the symbolic expressions. A minimal amount of sub-terms for a given time harmonic can be provided by such an implementation that will ensure that the product:

$$a_{h,j}(\mathbf{c})=\prod_{i=1}^{n}c_i^{\,p_{h,j,i}}, \tag{6}$$

and sum:

$$f_{h,j}(\boldsymbol{\alpha})=\sum_{k=1}^{m}g_{h,j,k}\alpha_k, \tag{7}$$

are not repeated in a single time harmonic.

A simplification is made in the implementation so that it deals with objects of inputs given as Fourier series of unknown amplitudes and phase shifts. The output is given also by Fourier series (Fig. 1). Further simplifications can be made when the specifications of the object S are known. In the current considerations it is assumed that during the calculation of the output vector $\mathbf{y}(t)$, the Fourier series with symbolic expressions undergo only the three mentioned basic mathematical operations.

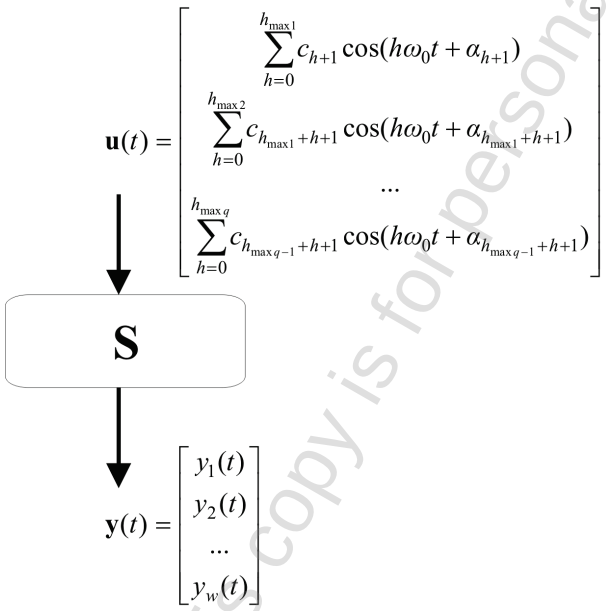

Fig. 1. Exemplary system where the presented implementation may be used

If the base symbolic expressions are amplitudes and phase shifts of certain initial time functions then it can be assumed that:

$$n=m. \tag{8}$$

The implementation (as will be shown in the next paragraph) is made so that if the above equality is not fulfilled then the implementation will also work properly. Modifications can also be made to the algorithm for similar dependencies if needed.

It can be noticed, that although the implementation has been brought forth for systems of a certain type (Figure 1) one can also use the proposed symbolic computation for parametric analysis where additional base expressions in the $\mathbf{c}$ or $\boldsymbol{\alpha}$ vectors can represent the properties of the object S. These would respectively affect the amplitudes or phase shifts of certain functions.

2. Specialized symbolic class implementation

The previous section presented a detailed description of the mathematical basis of the discussed implementation. This paragraph explains how the mathematical basis is used for the implementation. Possibilities of various ideas are mentioned in order to find an optimal solution satisfying the criteria of memory and operation time.

The form (5) allows the first partition to be made so that it does not concern symbolic expressions directly i.e. the partition into harmonic terms. For this purpose, a separate class may be implemented. In order to present each harmonic separately, it is useful to apply complex numbers. At the initial ramification, the form (5) can be presented through the vector:

$$\mathbf{x}=[\underline{x}_0 \quad \underline{x}_1 \quad \cdots \quad \underline{x}_{h\max}], \tag{9}$$

where the element of index h, responsible for the respective harmonic can be expressed as follows:

$$\underline{x}_h=\sum_{j=1}^{M_h}A_{h,j}\left(\prod_{i=1}^{n}c_i^{\,p_{h,j,i}}\right)\exp\left(\mathrm{i}(\theta_{h,j}+\sum_{k=1}^{m}g_{h,j,k}\alpha_k)\right). \tag{10}$$

At the stage of the symbolic expressions (10), vectors can be used to represent consecutive terms with a varying j index. However, if taking into account the aforementioned property that p and g are both integers, then the indexing according to j can be replaced by an indexing by the integers p and g. The latter idea has a specific advantage that when the indexing is used, the existing dependencies between the complete symbolic expressions and individual base symbolic expressions can be directly retrieved.

The simplest manner in which the above idea can be accomplished in C++ is the presentation through a $2n$-dimensional dynamic array. The exponentiations p and the multipliers g would (along with certain auxiliary variables) allow to point out the complex value $A_{h,j}\exp(\mathrm{i}\theta_{h,j})$. The indexing allows to assure that the (6) and (7) expressions are not repeated in a single time harmonic (what was mentioned in the previous section). This could aid both in memory and calculation time efficiency (the search for the same existing dependence on $\mathbf{c}$ and $\boldsymbol{\alpha}$ would require additional computation time).

To leave the idea in the form of a $2n$-dimensional array would however leave a serious flaw. This is because if symbolic expressions are large then they would be presented as data of $\prod_{N=1}^{2n}i_N$ elements (including an $i_1\times i_2\times\ldots\times i_{2n}$ array). In the case of rare occurrences of certain dependencies (exponentiations p and multipliers g) this would lead to a large amount of zero elements (which would mainly lead to unneeded memory being occupied and, what follows, the operation time would be increased). In order to reduce the said drawbacks, an implementation has been applied that uses consecutive object links. The object sequences express dependencies on the base symbolic expressions. The last objects in the sequences then link to appropriate $A_{h,j}\exp(\mathrm{i}\theta_{h,j})$ complex values.

If additionally the assumption (8) is made, then the dependencies on c_i and α_i can be represented through a single object in each sequence. It can be then assumed that the consecutive objects express the relationship with base complex symbolic expressions $\underline{c}$ understood as:

$$\underline{c}_i=c_i\mathrm{e}^{\mathrm{i}\alpha_i}. \tag{11}$$

The general idea of how the proposed implementation deals with storing a symbolic expression is depicted in Figure 2.

If, like mentioned in paragraph 1, one would want $n \neq m$ to be assumed, the class can be modified so that instead of 2D dynamic arrays, one can construct 1D arrays in objects for each base symbolic expression alone. The most important part is that the code must know how to handle each variant.

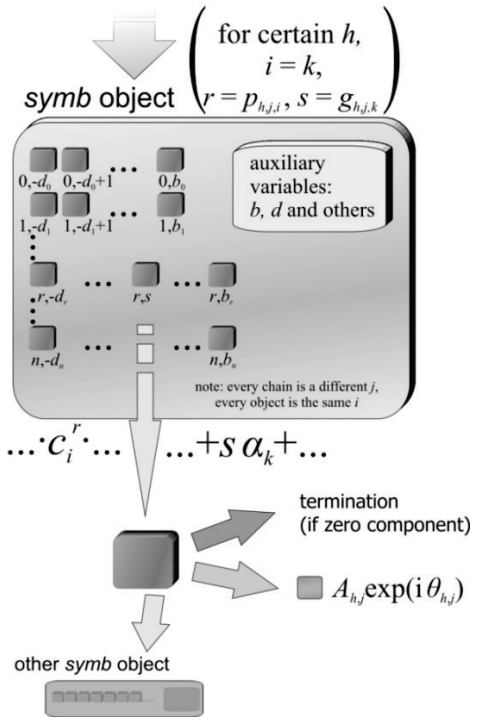

Fig. 2. Sketch of the idea of symbolic computation objects

An exemplary configuration of symbolic object chains is presented in Figure 3 in order to clarify the idea even further.

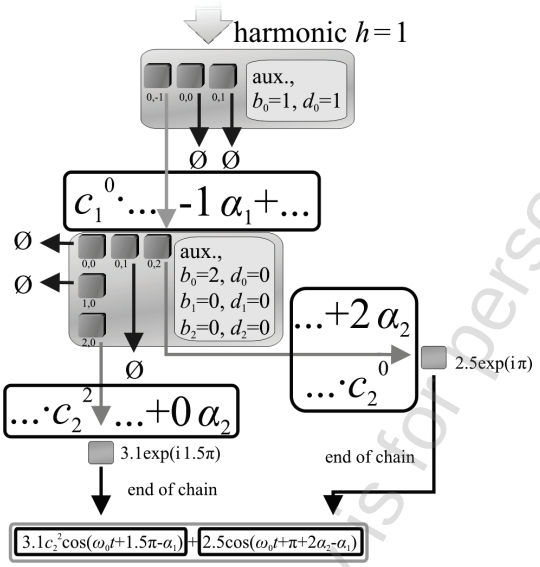

Fig. 3. Example of how a set of symbolic object chains formulates a symbolic expression

3. Test problems

A comparative analysis is performed which aims at verifying whether the proposed implementation is made properly i.e. if, in accordance with the intent, the description's simplifications and implementation idea create an efficient algorithm with respect to the chosen criteria of memory and computation time reduction.

Mathematica is an efficient environment for the analysis of various mathematical problems and features a wide variety of operations on expressions in symbolic form. Therefore, it has been chosen as a referential program for the comparative analysis in order to verify the correctness of the proposed method. The results of a few calculations have been compared to ascertain whether a specially written C++ program and *Mathematica* yield equivalent expressions. Furthermore, the comparison is made with respect to the mentioned efficiency criteria.

The implementation has been made in C++ but it can also be applied in other languages that allow for such linking of data types as discussed here.

The implementation of the specialized symbolic computation uses a harmonic balance vector (9) that expresses the Fourier series terms by complex numbers. This representation has also been adapted in exemplary *Mathematica* scripts which will be used in the comparative analysis. Thus, from the point of view of operations on time harmonics, the C++ implementation will generally not differ from the *Mathematica* scripts. It is the symbolic expressions that will be dealt with much differently. *Mathematica* includes an extensive assortment of tools, which allow operations on symbolic expressions, but it is not guaranteed that an optimal operation will be performed with respect to memory and computation time efficiency. Several drawbacks of a versatile implementation are worth mentioning:

- because one does not specify the intermediate or final form, the expressions often become larger than necessary, hence much more memory is used,
- because of the expressions being larger, operations between the symbolic expressions have a longer completion time,
- separate functions used for simplifying large expressions and putting them into the desired form need to be implemented because one can never guarantee that the result will be obtained in the desired form.

However the author has made an effort so that the *Mathematica* scripts used in the comparative analysis will be improved in order for them to be more efficient in the sense of the chosen criteria of efficiency. For the comparative analysis, two exemplary operations made on symbolic expressions in steady state are brought forth. The results of these operations are the time functions as follows:

$$r_1(t) = \left(A_1 c_1 \cos(\omega_0 t + \theta_1 + \alpha_1) + A_2 c_2 \cos(3\omega_0 t + \theta_2 + \alpha_2)\right)^9, \quad (12)$$

$$r_2(t) = \left(A_1 c_1 \cos(\omega_0 t + \theta_1 + \alpha_1) + A_2 c_2 \cos(3\omega_0 t + \theta_2 + \alpha_2)\right)^4 \cdot \cdot \left(A_3 c_3 \cos(\omega_0 t + \theta_3 + \alpha_3) + A_4 c_4 \cos(3\omega_0 t + \theta_4 + \alpha_4)\right)^7. \quad (13)$$

Two variants of a *Mathematica* function have been proposed to deal with the multiplication of symbolic expressions in steady state. They both differ only by the function **Reduc** that changes the form of the symbolic expression. The script is as follows:

```
multharmsWD[v1_,v2_,harms_]:=Module[{s1,s2,retv,i1,i2},
   s1=Dimensions[v1,1][[1]];
   s2=Dimensions[v2,1][[1]];
   retv=ConstantArray[0,harms+1];
   For[i1=1,i1≤s1,i1++,
    For[i2=1,i2≤s2,i2++,
      If[ i1+i2-2≤harms,
       retv[[ i1+i2-1 ]]+=v1[[i1]]*v2[[i2]]/2;];
      If[ Abs[i1-i2]≤harms,
       If[i1-i2≥0,
         retv[[i1-i2+1]]+=
          v1[[i1]]* Reduc[Conjugate[ v2[[i2]] ]]]/2;
         ,
         retv[[i2-i1+1]]+=
          v2[[i2]]* Reduc[Conjugate[ v1[[i1]] ]]]/2;
         ];
       ];
      ];
    ];
   retv];
```

In the first case the **Reduc** function is:

```
Reduc[x_]:=TrigToExp[ComplexExpand[x]];
```

and in the second variant:

```
Reduc[x_]:=TrigToExp[Simplify[ComplexExpand[x]]];
```

An additional function can be written for an exponentiation of the symbolic expression:

```
harmsdonWD[v1_,n_]:=Module[{i1,s1,retvec,s2},
   retvec=v1;
```

```
   For[i1=1,i1≤n-1,i1++,
    s1=Dimensions[retvec,1][[1]];
    s2=Dimensions[v1,1][[1]];
    retvec=multharmsWD2[retvec,v1,s1+s2+1];
    ];
   retvec
   ];
```

The script checking the time and memory used by the operations (12) and (13) is:

```
t0=SessionTime[];
r1=harmsdonWD[{0,(25+410I)*A1*Exp[I*fi1],0,(45+86I)*A3*
    Exp[I*fi3]},9];
t1=SessionTime[]-t0;
Print[t1]
Print[Floor[ByteCount[r1]/1024]]

t0=SessionTime[];
r2=multharsWD[harmsdonWD[{0,(25+410I)*A0*Exp[I*fi0],0,(45
    +86I)*A1*Exp[I*fi1]},4],
harmsdonWD[{0,(41+41I)*A2*Exp[I*fi2],
    0,(34+26I)*A3*Exp[I*fi3]},7],50];
t1=SessionTime[]-t0;
Print[t1]
Print[Floor[ByteCount[r2]/1024]]
```

Naturally, both scripts allowed to obtain different (though equivalent) expressions at a different computing duration. It is possible that when using other functions, different results could be obtained.

Results of both quantities determining the efficiency (in the understanding of the chosen criteria) are presented in Tables 1 and 2. These are respectively – the time needed to perform the given operation in order to obtain the symbolic expressions (i.e. $r_1(t)$ and $r_2(t)$) and the memory used for the storage of the symbolic expression resulting from the mathematical operation.

Table 1. Time needed to perform the operations of the chosen symbolic expressions

	Time used for operation (s)		
Obtained expression	Program using the symb class	Mathematica script (variant 1)	Mathematica script (variant 2)
$r_1(t)$	0.046	43.166	58.400
$r_2(t)$	1.109	198.641	255.254

Table 2. Memory required to store the given symbolic expression

	Memory used (KB)		
Obtained expression	Program using the symb class	Mathematica script (variant 1)	Mathematica script (variant 2)
$r_1(t)$	144	73293	41097
$r_2(t)$	2664	114316	27611

A clear relation can be noticed that both the author's program and *Mathematica* take more time to complete the calculation of obtaining $r_2(t)$. It is however interesting that the second variant of the *Mathematica* code needs more memory to store $r_1(t)$ rather than the more complicated expression $r_2(t)$. It is clear that more studies need to be performed on the choice of the *Mathematica* functions that change the form of symbolic expressions. It is possible that a more efficient script can be written than the ones proposed in this paper.

The proposed algorithm has proven to be useful because of its efficiency in terms of the chosen criteria. Very complicated and large symbolic expressions were computed in a very short time. Additionally the symbolic expressions resulting from the operations given in (12) and (13) require an insignificant amount of memory in comparison to the *Mathematica* scripts.

The calculations have been performed on Windows 7 Professional 64Bit SP1 on Intel™ Core™2 Quad CPU Q3900 @ 2.5GHz and total available 3.25GB RAM. Both *Mathematica* and the author's program have used a single core for the calculations. *Mathematica for Students 8* has been used.

4. Conclusions

The author's proposition of an implementation of specialized symbolic computation for steady state problems has been presented. The mathematical basis has been given that determined the rules for the algorithm to follow.

In accordance with a derived mathematical basis an analysis has been performed that allowed to choose the proper algorithm which could be effective in terms of the chosen criteria.

The chosen implementation has been explained with the aid of a general figure along with an exemplary set of object chains in order to clarify the symbolic expression presentation.

To verify whether the implementation is useful, it was ascertained if the algorithm allows to store a symbolic expression with the use of a small amount of memory. Furthermore, another verification has been made (which is also affected by the previous criterion) i.e. the time needed to perform certain operations on symbolic expressions has been observed.

The implementation has been made for chosen problems but certain propositions are also mentioned to extend its use for similar mathematical tasks.

In order to perform a comparative analysis, *Mathematica* has been chosen as it allows to perform a wide variety of operations on symbolic expressions. Additionally, it contains functions that allow for various presentations of symbolic expressions (their reductions, expansions etc.).

The author has written two scripts in *Mathematica* that allow to perform the comparative analysis in terms of the mentioned criteria of efficiency. It has been noticed that, in accordance with the intent, the proposed implementation is efficient i.e. a lot less memory is used to store symbolic expressions and the calculations are performed a lot quicker. This supports the idea of applying such specialized schemes of symbolic computation for certain problems instead of general algorithms during which all types of different symbolic expression forms are supported.

References

[1] Mathematica, Wolfram, http://www.wolfram.com/mathematica/
[2] Maple, Maplesoft, http://www.maplesoft.com/products/maple/
[3] http://www.mathworks.com/products/matlab/
[4] http://www.mathworks.com/products/symbolic/
[5] Sowa M., Spałek D.: *Analytical solution for certain nonlinear electromagnetic field problems,* Computer Applications in Electrical Engineering Issue 69, (2012).
[6] Sowa M., Spałek D.: *Nonlinear boundary condition application: numerical-symbolic scheme of formulation,* 35th International Conference of Electrotechnics and Circuit Theory IC-SPETO 2012. Gliwice-Ustroń (2012).
[7] Sowa M., Spałek D.: *Cylindrical structure with superconducting layer in a uniform electromagnetic field – analytical solution.* Advanced Methods of the Theory of Electrical Engineering 2011, Klatovy, Czech Republic (2011).

Mgr inż. Marcin Sowa
e-mail: marcin.sowa@polsl.pl

PhD student at the Institute of Industrial Electrical Engineering and Informatics in Faculty of Electrical Engineering at Silesian University of Technology. The author or co-author of 13 works published in Przegląd Elektrotechniczny and other journals, where he discussed the ongoing results of selected analytical and numerical methods for solving nonlinear boundary problems in the theory of the electromagnetic field

Artykuł recenzowany