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

Introduction

Symbolic computation is applicable in analyses that require the presentation of relationships between parameters or certain quantities and time 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 case of symbolic computation 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:

\[ x(t) = \sum_{h=0}^{h_{\text{max}}} \sum_{j=1}^{j_{\text{max}}} B_h \cos(h \omega t + \xi_h). \]  

If a presentation using symbols must be made, the above can be written with taking into account that \( B_h = B_h(s) \) and \( \xi_h = \xi_h(s) \), where \( s \) 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:

\[ x(t) = \sum_{k=0}^{k_{\text{max}}} \sum_{j=1}^{j_{\text{max}}} (\alpha_j x_h) \cos(\omega t + \theta_{h,j} + f_{h,j}(\alpha)). \]  

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

\[ \alpha = [\alpha_1 \quad \alpha_2 \quad \ldots \quad \alpha_{m}]. \]
As mentioned, the discussed symbolic expressions are either added, subtracted or multiplied hence a general form of the time function can be derived:

\[
x(t) = \sum_{h=0}^{h_{\text{max}}} \left( \sum_{j=1}^{J} \left( \prod_{i=1}^{n} c_i^{p_{h,j,i}} \right) \right) \cdot \left( h_0 \omega t + \theta_{h,j} + \sum_{k=1}^{m} g_{h,j,k} \alpha_k \right)
\]

(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 \( M \)) for each harmonic represents the number of dependencies on the base symbolic expressions \( c \) and \( \alpha \). The smaller the \( M \) 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 the product:

\[
a_{h,j}(c) = \prod_{i=1}^{n} c_i^{p_{h,j,i}},
\]

(6)

and sum:

\[
f_{h,j}(\alpha) = \sum_{k=1}^{m} g_{h,j,k} \alpha_k,
\]

(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 functions then it can be assumed that:

\[
f_{h,j}(\alpha) = \sum_{p=0}^{p_{\text{max}}} c_{h,j,p} \alpha^p,
\]

(8)

This could aid both in memory and calculation time efficiency (the search for the same existing dependence on \( c \) and \( \alpha \) 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 \( \prod_{i=1}^{\frac{2n}{m+1}} \) elements (including an \( i_1 x i_2 \ldots \times i_{\frac{2n}{m+1}} \) 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 \( A_{h,j} \exp(i \theta_{h,j}) \) complex values.

If additionally the assumption (8) is made, then the dependencies on \( c_i \) and \( \alpha_i \) 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 \( c \) understood as:

\[
c_h = c_1 e^{\alpha_1}
\]

(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 to assume, 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.

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 the 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:

\[
\begin{align*}
    r_i(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3, \\
    r_s(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3.
\end{align*}
\]

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 and in the second variant:

- 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:

\[
\begin{align*}
    r_i(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3, \\
    r_s(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3.
\end{align*}
\]

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 and in the second variant:

- 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:

\[
\begin{align*}
    r_i(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3, \\
    r_s(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3.
\end{align*}
\]

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 and in the second variant:

- 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:

\[
\begin{align*}
    r_i(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3, \\
    r_s(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3.
\end{align*}
\]

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 and in the second variant:

- 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:

\[
\begin{align*}
    r_i(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3, \\
    r_s(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3.
\end{align*}
\]

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 and in the second variant:

- 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:

\[
\begin{align*}
    r_i(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3, \\
    r_s(t) &= (A_1 c_1 \cos(\alpha_1 t + \theta_1 + \alpha_2) + A_2 c_2 \cos(3\alpha_1 t + \theta_2 + \alpha_3))^3.
\end{align*}
\]
The script checking the time and memory used by the operations (12) and (13) is:

```mathematica
For[i1=1, i1<=1, i1++,
  s1=Dimensions[retvec][[1]]; s2=Dimensions[v1,1][[1]]; retvec=multharmsWD2[retvec,v1,s1+s2+1];
];
```

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

```mathematica
t0=SessionTime[]; 
retvec=multharmsWD2[retvec,v1,s1+s2+1]; 
s2=Dimensions[v1,1][[1]]; s1=Dimensions[retvec,1][[1]]; 
t1=SessionTime[]-t0;
Print[t1] 
Print[Floor[ByteCount[r1]/1024]]
```

```mathematica
t0=SessionTime[]; 
retvec=multharmsWD2[retvec,v1,s1+s2+1]; 
s2=Dimensions[v1,1][[1]]; s1=Dimensions[retvec,1][[1]]; 
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. \( r(t) \) and \( r_2(t) \)) and the memory used for the storage of the symbolic expression resulting from the mathematical operation.

| Obtained expression | Time used for operation (s) | Mathematica script (variant 1) | Mathematica script (variant 2) |
|---------------------|-----------------------------|-------------------------------|-------------------------------|
| \( r(t) \)         | 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

| Obtained expression | Memory used (KB) | 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 Q9300 @ 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., Spalek D.: *Analytical solution for certain nonlinear electromagnetic field problems*, Computer Applications in Electrical Engineering Issue 69, (2012).
[6] Sowa M., Spalek 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., Spalek D.: *Cylindrical structure with superconducting layer in a uniform electromagnetic field – analytical solution*. Advanced Methods of the Theory of Electrical Engineering 2011, Klagenfurt, Austria (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.

Artystyczne recenzowanie