ELEKTRYKA 2016
Zeszyt 1 (237) Rok LXII

Marcin SOWA
Silesian University of Technology, Gliwice

SIMPLE C# CLASSES FOR FAST MULTIVARIATE POLYNOMIAL SYMBOLIC COMPUTATION – PART II: ANALYSIS OF NUMERICAL EFFICIENCY

Summary. In part I of the paper the implementation details of a lightweight C# implementation of symbolic computation of multivariate polynomials has been presented. The current part presents the results of an analysis of the numerical efficiency. The symbolic addition/subtraction, differentiation and definite integration operations are studied. The multiplication and exponentiation operators are discussed in part III.

Keywords: symbolic computation, sparse multivariate polynomials, computation time, numerical efficiency, C# implementation

PROSTE KLASY W JĘZYKU C# DO OBLICZEŃ SYMBOLICZNYCH WIELOMIANÓW WIELU ZMIENNYCH – CZĘŚĆ II: ANALIZA EFEKTYWNOŚCI NUMERYCZNEJ

Streszczenie. W części pierwszej niniejszego artykułu przedstawiono szczegóły dotyczące prostej implementacji (w języku C#) obliczeń symbolicznych na wielomianach wielu zmiennych. W niniejszej części zaprezentowano wyniki analizy efektywności numerycznej. Zbadano operacje symboliczne dodawania/odejmowania, różniczkowania symbolicznego oraz całkowania oznaczonego. Operacje mnożenia i potęgowania omówiono w części III.

Słowa kluczowe: obliczenia symboliczne, wielomiany rzadkie wielu zmiennych, czas obliczeń, efektywność numeryczna, implementacja C#

1. INTRODUCTION

The article concerns an analysis of the proposed C# libraries (that have been presented in part I) with an emphasis of its advantages in terms of the numerical efficiency. The analysis also contains results from a critical point of view, where it is indicated what should be improved in the implementation and what could be the drawbacks of the symbolic classes so far only operating on the assumptions given in section 2 of part I.

Several test categories have been proposed, which ascertain the numerical efficiency of an implementation that deals with multivariate polynomials of the general sparse multivariate expanded form given in part I (section 2). The tests require only the following operations being supported:

- addition/subtraction,
- multiplication,
- exponentiation,
- symbolic differentiation,
- symbolic definite integration.

The results of each trial are presented in the form of the following constituents of numerical efficiency:

- the time it has taken for the program to finish the computations,
- the memory needed to store the symbolic expression,

All of the results are compared with equivalent computations made in Mathematica 10 (a 15-day trial version). The software has been selected because the author of this paper has had much experience with it years ago and, most of all, because it is one of the most popular and useful programs for symbolic computation.

Mathematica features a function that allows to check the memory used by a single object, whereas the size of the `SPoly` objects (along with their subordinate `SMono` object lists) is ascertained by applying the C# garbage collector method `GetTotalMemory`. This also includes the amount of memory required to manage the object.

Two of the proposed test categories concern the multiplication and exponentiation operations, therefore they are presented in part III of the paper, where also an original algorithm for sparse multivariate polynomial multiplication is discussed.

2. ADDITION/SUBTRACTION TESTS

The first test comprises of the trials performed in subsection 4.2 of part I i.e.:

- the continuous addition/subtraction of terms with repeating symbolic multipliers,
- the addition of the selected result pairs.

Because the addition and subtraction operations do not differ generally in terms of the numerical efficiency – the subtraction case has been omitted. Mathematica does not feature the fast `(op)=` operation alternative (one that allows to overwrite the original object), however it contains a method called `Sum` that allows to formulate a long expression (provided that the general form of the subsequent terms is known). Hence, actually, `pluseq` in the C# implementation is compared with the performance of the `Sum` method in Mathematica. The results are displayed in Figure 1.

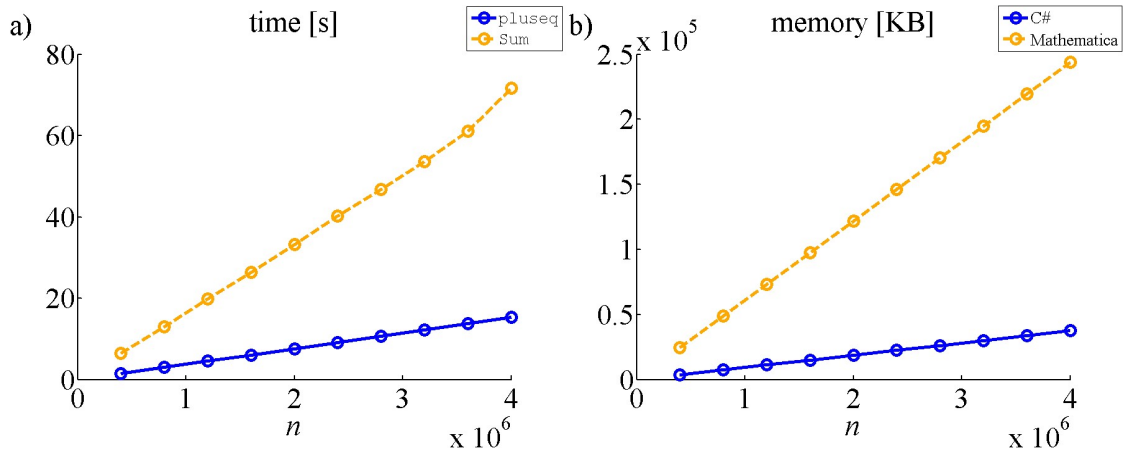

Fig.1. Comparison of the numerical efficiency for sequential adding in C# and Mathematica: a) computation time, b) memory required to store the resulting symbolic expression

Rys.1. Porównanie efektywności numerycznej dodawania sekwencyjnego w implementacji C# i oprogramowaniu Mathematica: a) czasy obliczeń, b) pamięć wymagana do przechowywania wynikowego wyrażenia symbolicznego

The results show that the `pluseq` method is very efficient for sequential additions as the computations are about 3 to 4 times faster than in Mathematica when using `Sum`. The required memory was about 10 times lower for the `SPoly` objects. Obviously Mathematica is prepared to deal with a wider variety of expression classes, which is why, in an obvious assumption, it will be slower and the objects will require more memory. Moreover, the expression string interpretation is not considered when comparing the computation times, which also limits the analysis to a general view at the test, without the distinction of the actual symbolic computations. Fact is, however, that in both the C# program and in Mathematica the results show a nearly linear rise in computation time and required memory, hence one can say that generally the applied algorithms perform similarly but the basic number operations take longer in Mathematica since they must automatically handle the number domains (e.g. as floating point, rational or integer numbers).

The second test, where the results of the first one are added, has its operation components already prepared and does not require string interpretations – hence it could be more useful to determine the computation efficiency. In this case only the + operator is compared as Mathematica does not feature an overtaking one. The results are presented in Figure 2.

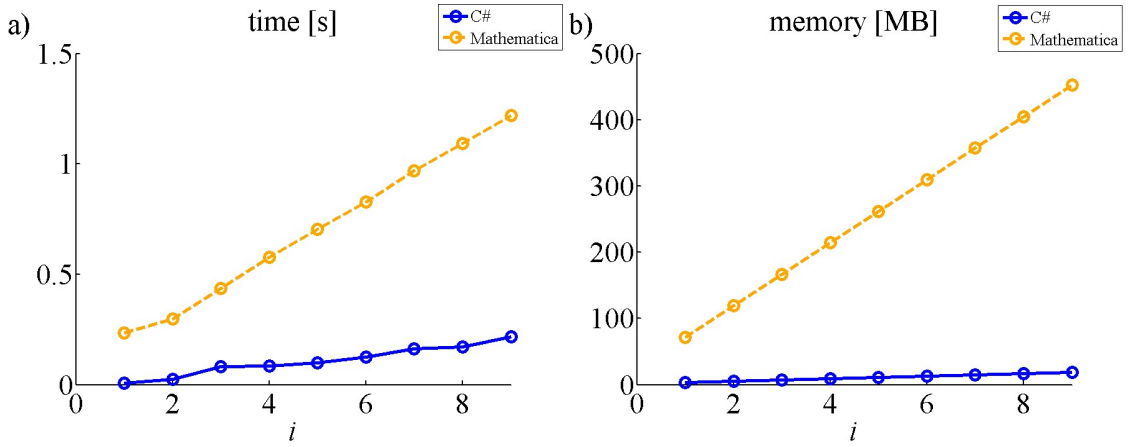

Fig.2. Comparison of the numerical efficiency for polynomial addition in C# and Mathematica: a) computation time, b) memory required to store the resulting symbolic expression
Rys.2. Porównanie efektywności numerycznej dodawania wielomianów w implementacji C# i oprogramowaniu Mathematica: a) czasy obliczeń, b) pamięć wymagana do przechowywania wynikowego wyrażenia symbolicznego

The results display that addition and subtraction operators have been efficiently implemented in terms of the potential computation speed. The resulting objects require relatively low amounts of memory for their storage. One can notice that even though the C# classes have proven to be more efficient, Mathematica still performs the operations very quickly, e.g. for $i = 9$ – polynomials of $8 \cdot 10^5$ and $7.2 \cdot 10^5$ terms have been added in only 1.22s.

When discussing single addition/subtraction operations – the proposed implementation features an implementation that works similarly like in any CAS. A new element is the overtaking operator (but, as mentioned in part I, it is only suggested in critical cases because of how it makes the partaking objects unusable after the operation). The author would only like to point out that lexicographical ordering implemented into a symbolic computation library positively influences the efficiency of a (later on implemented) addition/subtraction operator.

As for sequential addition/subtraction operations – in the author's opinion a mutable symbolic class (at least for polynomials) implemented in a CAS could be useful and convenient when building long expressions. It also allows to overwrite the original object when performing a += or −= operation, instead of completely rebuilding it just for the sake of adding a single new term.

3. CONCERNING THE REQUIRED MEMORY

In a polynomial structure, like the one proposed for the C# implementation, the memory required for the storage of n monomials is proportional to this number (when excluding the

objects contained in the `SPoly` object alone) as each `SMono` object contains the information on the variable powers and the real-valued multiplier. An exemplary test has been performed to ascertain the amount of memory required for an `SPoly` object with respect to the amount of stored monomial terms. The results (in comparison to the same test performed in Mathematica) are presented in Figure 3. The polynomial in the example is (in each case) a function of three base variables.

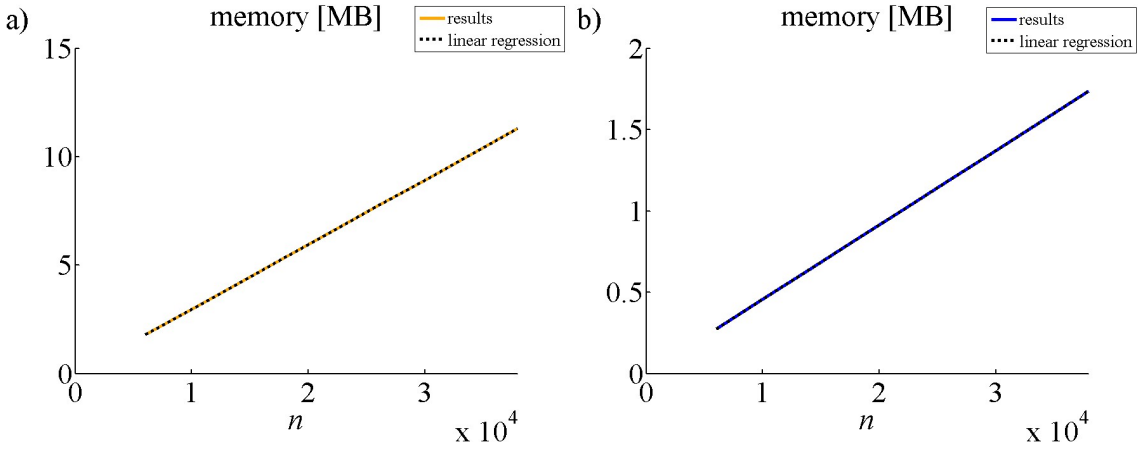

Fig.3. Memory required to store an exemplary symbolic polynomial (function of three variables), with respect to its number of terms, along with a comparison with the plot's linear regression: a) Mathematica, b) the C# implementation

Rys.3. Porównanie wymaganej pamięci do przechowywania przykładowego wielomianu w postaci symbolicznej (funkcji trzech zmiennych) z ilością jego wyrazów przy jednoczesnym porównaniu z liniową regresją wykresu: a) wyniki oprogramowania Mathematica, b) wyniki dla implementacji C#

Both in Mathematica and the C# implementation the proper, expected attribute is observed, where the memory required for the storage of the symbolic expression is proportional to the number of terms in that expression. In the example Mathematica required 312 bytes per monomial while the proposed C# implementation required 48 bytes for each term.

4. ANALYTICAL DIFFERENTIATION TEST

The following test is one where the partial derivatives of a symbolic expression are obtained. For a polynomial $S(a, b, c)$ (obtained in the first example of section 2) with various numbers of terms (denoted by n): first a partial derivative $\frac{\partial S}{\partial a}$ is computed and then a mixed derivative $\frac{\partial^2 S}{\partial a \partial b}$ is obtained. The computation times of these trials are depicted in Figure 4.

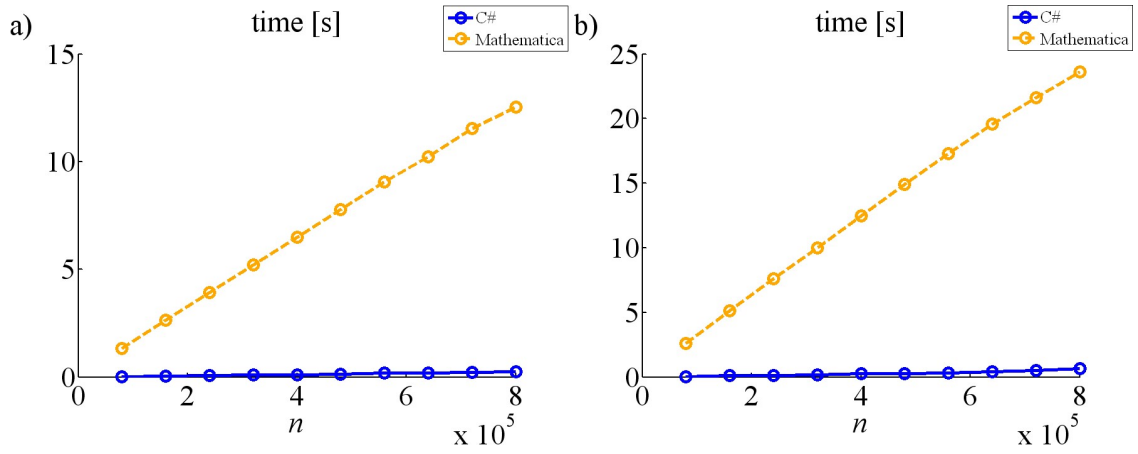

Fig.4. Comparison of the computation time for the evaluation of a partial derivative in the C# implementation and Mathematica: a) first order derivative $\frac{\partial S}{\partial a}$, b) mixed derivative $\frac{\partial^2 S}{\partial a \partial b}$

Rys.4. Porównanie czasów symbolicznego wyznaczania pochodnych cząstkowych za pomocą implementacji C# i oprogramowania Mathematica: a) pierwsza pochodna $\frac{\partial S}{\partial a}$, b) pochodna mieszana $\frac{\partial^2 S}{\partial a \partial b}$

The computation time for the C# implementation in both trial series is around 40 times faster for each run. What obviously makes the C# implementation so fast with differentiation is that its base symbolic variables are identified only through their indices, not requiring searches for the proper character or string (or their interpretation). As expected, the computation of the mixed partial derivative takes, in all of the examples, around twice as long as for the single partial derivative, because the operation is performed two times, almost for the same amount of terms.

5. DEFINITE INTEGRATION WITH NUMERICAL BOUNDS

The next two tests (explained in this section and section 6) concern definite integrations. The first test deals with integration bounds defined as numerical values. First (what is not included in the measured computation time) an expression $S(a, b, c, d, e)$ is built according to the pseudo-code (where `trial_i` denotes the number of the trial):

```
S1=0; S2=0; S3=0; itmax=20*trial_i;
For it=1 to itmax:
  i = it - 1; j = itmax - it; k = it + 4;
  S1+= a^i * b^i * c^k;
```

```
    S2+= a^k * b^i + c^k;
    S3+= a^j * b^k + c^i;
End
S = S1 + S2 + S3;
```

The actual test involves the evaluation of the expression:

$$I = \int_{a=-1}^{1} \int_{b=-1}^{1} S \, \mathrm{d}b \, \mathrm{d}a. \tag{1}$$

The computation times for each performed trial are presented in Figure 5.

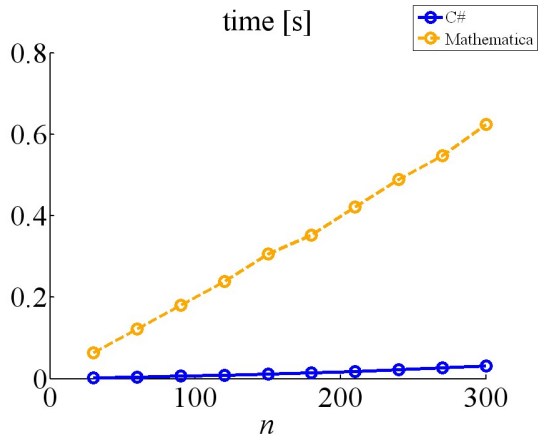

Fig.5. Comparison of the computation time for the evaluation of a definite integral (with bounds defined by numerical values) in the C# implementation and Mathematica; n denotes the number of terms in S

Rys.5. Porównanie czasów symbolicznego całkowania (z granicami zdefiniowanymi jako wartości numeryczne) za pomocą implementacji C# i oprogramowania Mathematica; n oznacza liczbę wyrazów w wyrażeniu S

The computations have been performed, in average, approximately 25 times faster with the C# implementation. Additionally, when the integration bounds are in advance assumed as numerical values, the implementation utilizes the formula:

$$\int_{c(\boldsymbol{s})}^{d(\boldsymbol{s})} \zeta(\boldsymbol{s}) \, \mathrm{d}s_l = \sum_{i=1}^{N} \frac{a_i}{k_{i,l}+1} \left(d^{k_{i,l}+1} - c^{k_{i,l}+1} \right) \prod_{j=1, j\neq i}^{n} s_j^{\,k_{i,j}}. \tag{2}$$

(previously also discussed in section 5 of part I). This leads to a decrease in computation time (in average by 10% for the quoted example).

6. DEFINITE INTEGRATION WITH BOUNDS IN SYMBOLIC FORM

The trials of this subsection concern the evaluation of the expression:

$$I = \int_{a=1}^{c} \int_{b=1-a}^{e} S \, \mathrm{d}b \, \mathrm{d}a. \tag{3}$$

where S is obtained in the same manner as in the previous section. Mathematica leaves its expressions in either a factored or expanded form, depending on how they were built. In the case of the previous section (bounds in numerical form) the expressions existed in an expanded form, however, after applying an integration with bounds represented by symbolic expressions – it is possible that the polynomials will be left factored. Both cases have been studied in Mathematica: for the expressions left in their default state and for the integration to be performed on an enforced expanded form. The results, in comparison with the performance of the C# implementation, are presented in Figure 6.

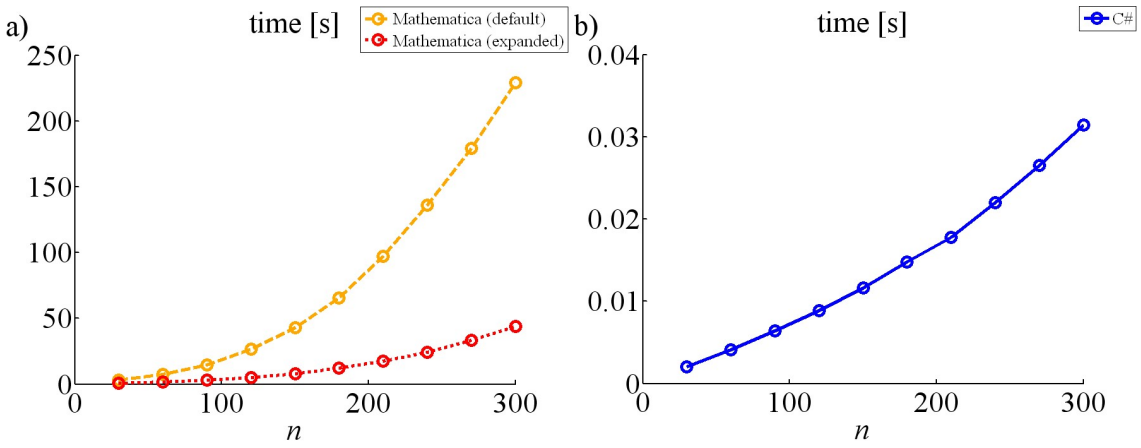

Fig.6. Comparison of the computation times for the evaluation of a definite integral (with bounds defined by symbolic expressions) in: a) Mathematica (in the default case and the case with an enforced expanded form), b) the C# implementation

Rys.6. Porównanie czasów symbolicznego całkowania (z granicami zdefiniowanymi jako wyrażenia symboliczne) za pomocą: a) oprogramowania Mathematica (w przypadku postaci domyślnej wyrażenia oraz w przypadku wymuszenia postaci rozwiniętej), b) implementacji w języku C#

As can be observed from the figure, the definite integration is very efficient in the C# implementation as it has completed the test, in average, around 4200 times faster than in Mathematica with the default form of the expressions and approximately 800 times faster than in the case with the expanded polynomial forms. What is interesting – it seems that the symbolic integration alone isn't as slow in Mathematica but the substitutions (especially when a variable is substituted by a symbolic expression) require much time for their completion.

7. COMPARISON OF FACTORED AND EXPANDED FORM EFFICIENCY

In the previous test the expansion of the polynomials, before their integration, allowed to reduce the total computation time. This subsection presents an example where it could be profitable to leave the expressions in a factored form. The following operation is considered:

$$I = \int_{a=1}^{c} (a+b+c+d+e+f)^{2i} \, \mathrm{d}a, \tag{4}$$

where i denotes the number of the trial. In the first case an expansion of the exponentiation is performed before the integration and in the second case no expansions are performed – just simply the integration is performed. The results are presented in Figure 7 (the example has also been computed in the C# implementation – however, only for the handled case of expanded forms).

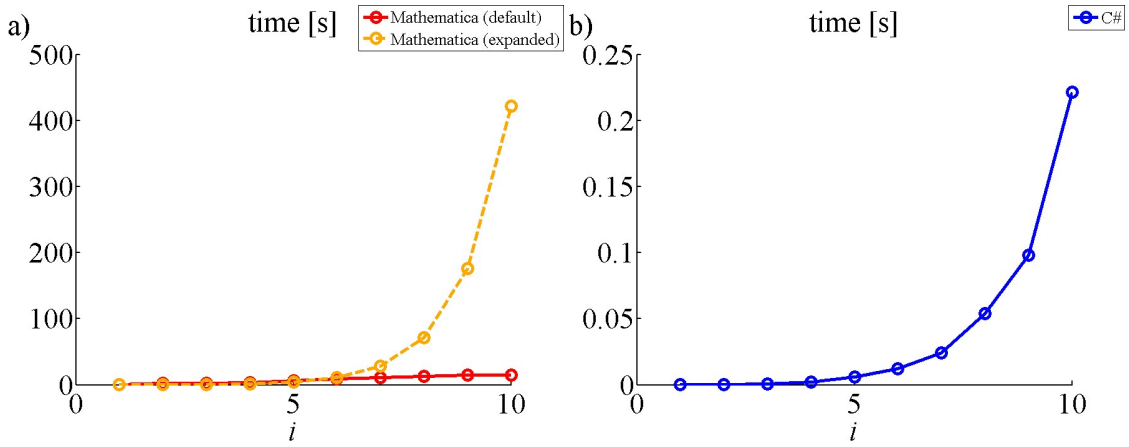

Fig.7. Comparison of the computation times for the evaluation of the definite integral of equation (4) in: a) Mathematica (in the default case and the case with an enforced expanded form), b) the C# implementation

Rys.7. Porównanie czasów symbolicznego całkowania opisanego równaniem (4) za pomocą: a) oprogramowania Mathematica (w przypadku postaci domyślnej wyrażenia oraz w przypadku wymuszenia postaci rozwiniętej), b) implementacji w języku C#

The results obtained in Mathematica are interesting as for $i \leq 5$ the expanded form is advantageous – however, for the other cases the computations are up to 30 times faster (and rising) when the expression is left in its factored form. Obviously much time is saved by not needing to compute the exponentiation. The C# implementation is still faster as the definite integration alone is performed more efficiently.

In the next trials the expression F is considered:

$$\begin{aligned} F = (&xy^3z^2 + x^2y^2z + xy^3z + xy^2z^2 + y^3z^2 + y^3z + \\ &+ 2y^2z^2 + 2xyz + y^2z + yz^2 + y^2 + 2yz + z)^k, \end{aligned} \tag{5}$$

which is actually taken from a benchmark used in [1] and will also be used in part III of this paper. The following expression is evaluated:

$$I = \int_{x=0}^{1} \int_{y=0}^{1} \int_{z=0}^{1} F^{i+1} \, \mathrm{d}z \, \mathrm{d}y \, \mathrm{d}x. \tag{6}$$

The results for various i are presented in Figure 8. The efficiency of the C# implementation is also studied.

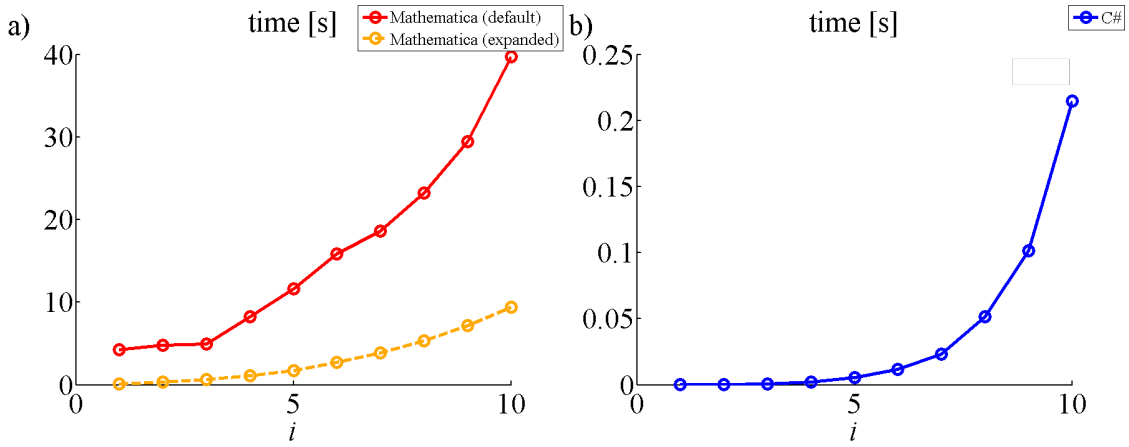

Fig.8. Comparison of the computation times for the evaluation of the expression in equation (6) in: a) Mathematica (in the default case and the case with an enforced expanded form), b) the C# implementation

Rys.8. Porównanie czasów symbolicznego wyznaczania całki opisanej równaniem (6) za pomocą: a) oprogramowania Mathematica (w przypadku postaci domyślnej wyrażenia oraz w przypadku wymuszenia postaci rozwiniętej), b) implementacji w języku C#

It seems that in certain cases the drawbacks of integrating a factored polynomial outweigh the potential benefits (that concern saving the time used otherwise for the expansion of the polynomial exponentiation). Generally, polynomials in expanded forms are advantageous in perspective of predictable efficiency as the computation time can be often related to the amount of operations. On the other hand – polynomials left by default in a factored form are a logical choice since then (in most cases) initially they require less memory for their storage.

8. CONCLUSIONS

The C# implementation of symbolic computations on multivariate polynomials has been subjected to tests that have verified its efficiency for the supported operations. Addition/subtraction, differentiation and definite integration (including the case with bounds defined as symbolic expressions) trials have been performed to ascertain the efficiency of the implementation. The trials for multiplication and exponentiation are presented in part III of

the paper along with a detailed analysis of the implementation of the actual polynomial multiplication algorithm.

In base operations the C# implementation has been more efficient because of simpler data structures being supported. This means that optimizations at the base level of the polynomial structures could greatly improve a CAS in terms of computation speed. The researchers of Maple seem to properly improve their base symbolic structures, where not only has a special data structure been implemented for polynomials [2] but also its operation is optimized at the bit-level. These types of optimizations (also like in the case of the TRIP CAS [3]) tend to make computations incredibly efficient.

As to a general advantage of the C# implementation on the algorithm level – most of all the definite integrations (especially in the part where symbolic substitutions are made) are considerably more efficient in comparison to Mathematica.

BIBLIOGRAPHY

1. Monagan M., Pearce R.: Sparse polynomial powering using heaps. “Computer Algebra in Scientific Computing”, Springer, 2012, s.164-167.
2. Monagan M., Pearce R.: POLY: a new polynomial data structure for Maple 17. “ACM Communications in Computer Algebra” 2013, 46 (3/4), s.164-167.
3. Gastineau M., Laskar J.: Development of TRIP: Fast sparse multivariate polynomial multiplication using burst tries. “Computational Science-ICCS” 2006, s.446-453.

Dr inż. Marcin Sowa
Silesian University of Technology
Faculty of Electrical Engineering
Institute of Electrical Engineering and Computer Science
ul. Akademicka 10
44-100, Gliwice
e-mail: Marcin.Sowa@polsl.pl