TAYLOR AND MACLAURIN SERIES EXPANSION USING COMPILER-COMPILER TOOLS

MIR MOHAMMAD REZA ALAVI MILANI, HÜSEYIN PEHLİVAN, SAHEREH HOSSEINPOUR

Abstract: In current age that leads to the turning from industrial society to ultra-industry or information society, it is natural that, the information, science and knowledge are the fundamental properties for human and society. Some people believe that the modern age will be a different world where the information technology will be the leading. In this way, using of these instruments in teaching and education can cause some suitable changes in education systems. In this article, we propose a methodological approach for automatic solving of math problems, especially in terms of the calculation of Taylor and Maclaurin series, with the aim of the practicing of mathematics subjects, by using of Computer Algebra System (CAS) tools. The approach is illustrated on a currently developed instance of CAS through all the intermediate stages from the input of mathematical expression to expansion of the related series. The paper also addresses some specific fields such as the simplification and automatic production of mathematical expressions. One of major benefits of the proposed system is that students can access many examples of similarly solved problems in an appropriate way without spending time and cost, particularly in order to comprehend a mathematical topic well. The generalization of virtual teaching/education and the necessity of creating the practice and evaluation systems in this field make the significance of this study clear.

Keywords: ICT, Computer Algebra System, JavaCC, Self-Evaluation

Introduction: In modern world, the math science is as a blood in the great body of other sciences. The teaching of math is not only a science but also it is as a model for correct teaching to other sciences. Creative, innovative and daring minds, in responding to the questions around themselves, undoubted resulted from a system that, the math knowledge, has the ability to revive it inherently. In current astonished millennium, the discovery of the globe, throwing the satellites, making atomic submarines and entering to the most complicated world, by having tens, hundreds and thousands of modern technologies which every of them can only answer to some of the human society problems. So, it can be said that correct teaching of math means the correct teaching of all sciences. Therefore, the teaching of mathematics is much more important.

When the teacher, teaches a lesson to a group of students, with the all advantages the face to face teaching process is happened. But when the teacher asks that "Is there any question or problem?" other new teaching instruments, show their capabilities. Then, the strong student wants new learning materials but the weak student has the problem in learning of the taught materials. After solving the students' problems, the needs for helpful teaching instruments for deepening of understanding comes out, and exactly after that, the repetition of materials is so necessary to establish and experience, which causes the static learning of materials in the students' mind.

One of the main steps in the teaching process is practice and repetition. Also one of the fundamental weak points of traditional education system is the limitation of doing the practices, because of the time limitation and the other obstacles like the extra time-consuming in writing and the students' uncertainty in doing of the practices correctly, and the spending of much more time and the limitation in observing of those practices by teacher, too.

In practice section, all the students should confronted with the teaching process individually, because everyone develops based on his/her talent, ability and power and the pioneer students never stop for others, as the weak students, don't have to pass from a part of learning materials which haven't learned correctly because of the weak learning. So, it is clear that all the things that stated in instrument which helps to teaching process is not limited to school environment and the student can have all of them in his/her home and don't need to bring and take the books and notebooks between school and home.

One of basic strategies for mathematical education is the use ofcomputer algebra system (CAS). CAS can be used in academic evaluation. For long, one of main concerns has been considered to be the design of standard questions consistent with educational level. The step-by-step solution of problems can help students in self-evaluation and learning through problem solving. In this paper, a CAS-like system is presented for the step-by-step solution of problems. The proposed system also deals with the production of new questions using the templates that are derived from the solved questions or entered by the users. The core calculation mechanism of the system, which

is based on symbolic calculation techniques, can be coded by using a Java-like programming language. The input data of the system are textual expressions used for the representations of mathematical problems, which can be parsed via a compiler-compiler tool such as JavaCC. The system currently focuses on the solutions of various problems associated with the subject of derivative, however, can be easily extended to cover the other subjects of general mathematics.

Using the proposed system, it is possible to create a suitable environment in which students can study on many mathematical problems generated according to the level of their individual skills. One of major benefits of the system is that students can access many examples of similarly solved problems in an appropriate way without spending time and cost, particularly in order to comprehend a mathematical topic well. Besides the step-by-step solution of problems, if possible, the different solution ways for the related problem is also provided, which helps students to learn better, and have them get familiar with different aspects of the solution. On the other hand, the system allows educators to design similar questions for their own educational standards, and to produce some tests and quizzes for students more easily and rapidly.

Computer Algebra Systems (CAS): CAS is systems that perform arithmetic operations on functions, formulas, symbols and numbers without any computational errors. Usually, the results of this type of system are without error and ambiguity. These systems are very diverse, but usually some type of system that runs on standard mathematical operations, simplifying polynomials and functions, is more focused. Some features of this system are as follows:

- Simplification of complex functions
- Differentiation
- Definite and indefinite integration
- Limit
- Differential Equations
- Series
- Linear and non-linear equations
- Operations on vectors and matrices

Modern CAS systems have conversational features and graphical and numerical representations. Many general mathematical problems are being developed and presented by these systems. CAS systems are classified to two groups: General purposes; and specific purposes. Systems with general purposes are designed and created for general mathematical intents, and focused on general mathematical operations. AXIOM, REDUCE, MACSSYMA, MAPLE, MATHEMATICA and DERIVE are examples of these types of systems.

Systems with specific purposes insert operations, which are mathematical and or physical operations in general, on specific environments. Data structures in these types of systems are usually designed by a work environment. Examples of these systems are CAYLEY, which is designed on group theory, and DELIA, which is designed on differential equations.

With increasing CAS systems, the thought of use of these systems in mathematical teaching was created; there were different discussions about the use of these systems in mathematical curricula (Kutzler, 2000;Waits & Demana, 1999). The application of CAS systems to teaching programs has been focused from two dimensions: First dimension, the use of these systems as part of mathematical teaching programs like instructing materials including solving equations, derivates, etc. (Judson, 1990; Mayes, 1995; Palmiter, 1991; Runde, 1997); and second dimension, the development of teaching programs based on technology, that is changing teaching structures and methods based on the use of technology (Brown, 1998; Drijvers, 1998; Heid, 1997; Herget, Heugl, Kutzler & Lehmann, 2000).

In recent years, different research on the impacts of use of CAS systems on improvement in mathematical teaching has been conducted, showing the favorable impact of use of CAS systems in mathematical teaching (Bennett, 1995; Day, 1996; Heid, 1988; Tall, 1996). Also, studies on the use of technological educational systems for teaching the derivative subject have been performed (Heid, 1984, 1988; Judson, 1988; Palmiter, 1991; Repo, 1994).

Efforts made on the use of CAS systems in mathematical teaching have usually been as non-numerical solutions and, in some cases, graphical representations, which the results suggest improvement in educational quality while the utilization of these systems.

Proposed System: In this paper we seek to present a method to calculate the Taylor expression symbolically by using the common tools of compiler designs as well as utilizing the CAS methods. Before addressing how it is done, however, first the expression received from the user should ideally be examined in terms of the verified entry (the rules related to a mathematical expression) and the resulting process should be transformed to a tree structure known as parse tree. Therefore, we examine the suggested system in separate phases with the following titles.

- Lexical and syntax analysis, and parse tree construction
- Conversion of parse tree into Taylor series
- Simplification and presentation
- Automatic creation of mathematical expressions

Lexical and Syntax Analysis, and Parse Tree Construction: We use the Java programming language to implement the proposed system, and a compiler-compiler tool, called JavaCC, which can generate code segments required for both scanner and parser. The analysis process consists of the lexical and formal examination of the input data (i.e. a particular mathematical expression). The general form of an expression is grammatically defined using a context-free grammar (CFG). Since JavaCC generates LL(k) parsers, this grammar must be LL(k) one. We illustrates how to transform a CFG grammar into an LL(k) one below.

The design of a scanner and parser is based on the lexical and phrase structure of mathematical expressions. Thus, a CFG grammar is developed to recognize and parse such expressions, as seen in Table 1.

Table 1: ACFG grammar for mathematical expressions

E →E "+" E (Plus)	E →"exp" "(" E ")" (Euler)
E →E "-" E (Minus)	E →"log" "(" E ")" (Log)
E →E "*" E (Times)	E →"sqrt" "(" E ")" (Sqrt)
E →E "/" E (Divide)	E →"sin" "(" E ")" (Sin)
E →E "^" E (Power)	E →"max" "(" E "," E ")" (Max)
E →"x" (Var)	E → (D)+ ("." (D)+)? (Num)
E →"+" E	E →"(" E ")"
E →"-" E	D →["0"-"9"]

Table 1 shows a grammar which does not consider operator priorities and associativities in the evaluation of expressions. In addition to the form of the input data the grammar must also represent the appropriate priority level and associativity of operators. This task is followed by the conversion of the grammar to the LL(k) one. The resulting LL(k) grammar is shown in Table 2, where k=1.

Table 2: An LL(1)grammar for mathematical expressions, including operator priority and associativity.

Rule	Method	Rule	Method
S →E $	parse()	R →"x"	element()
E →T E' E' →("+" | "-") T E' E' →	expr()	R →"exp" "(" E ")" R →"log" "(" E ")" R →"sqrt" "(" E ")" R →"sin" "(" E ")" R →"max" "(" E "," E ")" R →(D)+ ("." (D)+)? R →"(" E ")" D →["0"-"9"]	
T →F T' T' →("*" | "/") F T' T' →	term()		
F →("+" | "-")? P	unary()		
P →R ("^" P)?	power()		

A scanner splits the input data into a sequence of tokens which are the atomic parse elements of that input data. For mathematical expressions, "+", "-" and "*"are the typical examples of tokens. On the other hand, as there is a separate token class defined for each different operator in a mathematical expression, all numbers in that expression are represented by only one token class, called NUMBER; for example, a scanner always generates token NUMBER for the numbers "5" and "0.1". The list of token defined for the proposed system is given in Table 3.

Table 3: Sampletoken declarations

//EvalParse.jj	(Continued)
SKIP : { " " | "\t" | "\r" }	| < POWER: "^" >
TOKEN : { < NUMBER : (["0"-"9"])+	| < EXP: "exp" >
("." (["0"-"9"])+)? >	| < LOG: "log" >
}	| < SQRT: "sqrt" >
TOKEN : { < EOL : "\n" > }	| < SIN: "sin" >
TOKEN : /* OPERATORS */	| < MAX: "max" >
{	| < X: "x" >
< PLUS: "+" >	| < COM: "," >
| < MINUS: "-" >	| < LPR: "(" >
| < TIMES: "*" >	| < RPR: ")" >
| < DIVIDE: "/" >	}

For example, using the definitions of tokens in Table 3, the mathematical expression "(x+1)*2-x^sin(x)" is scanned as the token sequence "LPR X PLUS NUMBER RPR TIMES NUMBER MINUS X POWER SIN LPR X RPR".

The token sequences into which the input expressions are separated then are analyzed by a parser through the set of grammar rules. In this stage of analysis, an expression from true syntactic form leads to the construction of a parse tree. Whit the definition in Table 4, the related source code for a parser can be automatically generated by JavaCC.

Table 4: The definition ofthe grammar rules inJavaCC

```
//EvalParse.jj
void parse() : { }
{
expr() (<EOF> | <EOL>)
}
void expr() : { }
{
term()( <PLUS> term()
| <MINUS> term()
)*
}
void term() : { }
{
unary() ( <TIMES> unary()
| <DIVIDE> unary()
)*
}
void unary() : { }
{
<PLUS> power()
(Continued)
| <MINUS> power()
| power()
}
void power() : { }
{
element() ( <POWER> power() )?
}
void element() : { }
{
<NUMBER>
| <X>
| <LPR> expr() <RPR>
| <EXP><LPR> expr() <RPR>
| <LOG><LPR> expr() <RPR>
| <SQRT><LPR> expr() <RPR>
| <SIN><LPR> expr() <RPR>
| <MAX><LPR> expr() <COM>
expr() <RPR>
}
```

The other facility of JavaCC is that it can generate parse trees for each input expression. For example, the tree representation of the above expression can be provided as seen in Figure 1

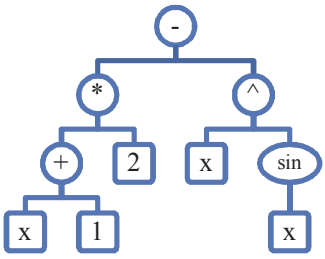

Figure 1 : The tree representation of an expression

Construction of the Taylor Series: For the Taylor series expansion of an input function f(x), the system calculates the successive derivatives of the function, and represents the related series through Equation 1.

$$f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 + \cdots \quad \text{(Equation 1)}$$

The derivative of an expression is symbolically obtained by using the basic derivative rules, for example," the derivative of the sum of two expressions equals the sum of their derivatives". The new expression produced by the derivation stage possibly has a considerable number of terms and thus is followed by a simplification one. Therefore, the following functions are required to design the related system.

- Function to calculate the derivative symbolically
- Function to perform the simplification symbolically
- Function to find Taylor series symbolically using by Equation 1.

Derivative Calculator: The derivative of an

expression through a parse tree can be performed as follows. By visiting each node of the tree, a basic derivative operation is performed, depending on the basic operator type residing in that node. For example, if the visited node has an operator +, the left and right children of that node are obtained by the derivative function as two separate trees, and their sum is returned to the caller as the derivative of the node, which is expressed by the pseudo-code in Figure 2.

```
if OpNode instanceof Pluse then
        L=LChild.Drive()
        R =RChild.Drive()
        Return Plus(L,R)
End if
```

Figure 2 : Pseudo-code for Plus class of Derivative

The procedure explained above will recursively be repeated for every node type, depending on their operators and according to the derivative rules. At the end of this procedure, the obtained tree is related to the derivative of the expression. The code fragment in Figure 3, for example, shows the derivative function associated with a syntax class Plus, coded in the Java language.

```
class Plus extends Exp {
  public Exp exp1, exp2;

  public Plus(Exp e1, Exp e2) {
    exp1 = e1; exp2 = e2;
  }

  public Exp Driv() {
    return new Plus(exp1.Driv(), exp2.Driv());
  }

}
```

Figure 3 : Java code of Derivative function for Plus class

The next task is the simplification of the resulting tree. Therefore, having done the derivative of the function, simplification, which will be addressed in the next section, will be called.

Simplification: The next task is the simplification of the resulting tree, which is one of the most important and underlying operations in symbolic computation systems. This has many difficulties in the implementation because some concepts of simplification are naturally challenging. For example, responding the question “which of the states shown in the following figure is simplified?” justifies this claim.

In this step, reusing the tree obtained from the previous step, all nodes associated with the tree are visited again; this time the simplification operations are performed for every node. For example, if the visited node has an operator +, which is connected to two child nodes with numbers, the sum of these two numbers are returned as the simplified form of the node.

The pseudo-code in Figure 4 represents this example.

```
if OpNode instanceof Pluse then
        If RChild and LChild instanceof Num then return Num(RChild.val()+LChild.val())
End if
```

Figure 4 : Pseudo-code for a node with two Num child

In the above state, if both child nodes are of the same variable, the addition operation can be done as Figure 5.

```
if OpNode instanceof Plus then
        If RChild and LChild instanceof Num then return new Num(RChild.val()+LChild.val())
        If RChild and LChild instanceof Var then return new Var(RChild.val()+LChild.val())
End if
```

Figure 5 : Pseudo-code for a node with two Num or Var Child .

Note that, in the above pseudo-code, the class Var is used as a variable, and the coefficient of variable x is placed inside the Var objects. For example, Var(3) is used to represent the expression 3*x. The simplification stage can encounter some particular expressions. To illustrate this, let us consider more complex expressions represented by a tree such as Plus(Var(5), Plus(Num(3),Var(2)))
The represented expression is 5*x+3+2*x. Since the child nodes are not of the same type, the parent one is not able to simplify, while the expression is equivalent to the simplified one 7*x+3, represented by Plus(Var(7),Num(3)). This problem can be resolved by the transformation of the binary tree into multiple one and then applying the simplification. Figure 1 shows the case graphically.
In this way, the simplifying function can generate the simplest form of the expression.

Taylor Series Calculator: In this step, having the underlying function, a tree related to the Taylor expansion can conveniently be created. To do this, one can do as Figure 6.

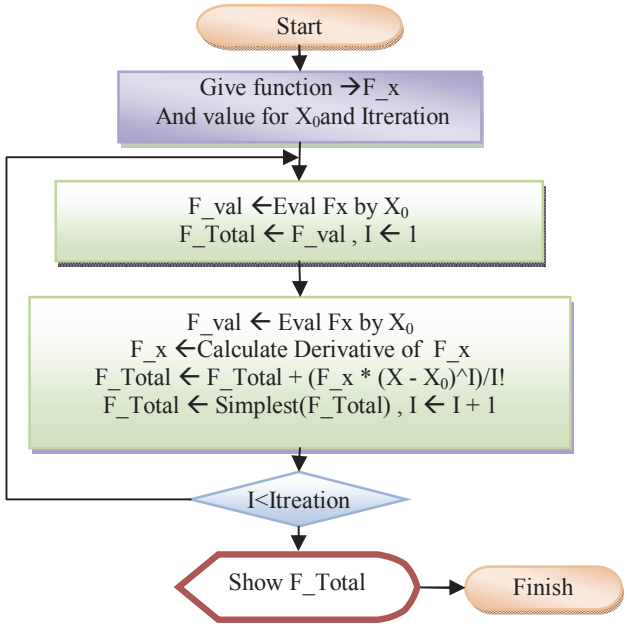

Figure6 : Flowchart for Taylor Series Calculator

According to the flowchart above, having received an expression from the user and having generated an initial parse tree, as well as by considering Equation 1, trees related to successive derivatives are generated. Consider that for calculating the nth derivative, the simplified form of the (n-1)th derivative is used. Every resulting derivative is trees, which according to Equation 1 should be connected under a single tree, known as Taylor expression tree. Of course, the number of derivatives is appropriately received from the user.
Notice that in generating the Taylor expression, the simplification function is used not only after every differentiation, but also called after the final generation of the Taylor expression.

Simplification and Presentation : At the end, the Taylor expression tree should comprehensively and suitably be transformed to a mathematical expression to be represented. For this, a function should be designed that takes a tree as an input and then returns a mathematical expression related to that tree as a string.
Performed by using the concepts of classes and recursive calling, by meeting every node and depending on the operator type of the node, this function generates and returns appropriate strings.For example, entering a node with the operator +, it returns a value of pseudo- code as Figure 7.

```
If OpNode instanceof Plus then
        String SR = RChild.Print()
        String SL = LChild.Print()
        Return SR + "+" + SL
  End if
```

Figure 6 : Plus class Pseudo-code for mathematical transformation function.

Of course, it should be noticed that for representing mathematical expressions, the appropriate use of parentheses is of specific importance, which should specifically be considered in the design of the function.

Creating an equation as question by a specific template automatically: Using the concepts of compilers and having a tree related to a mathematical expression, a new mathematical expression is automatically produced. First, a tree related to a question is created, and then changing the constant expressions to the variable expressions a template of the question is generated.

Now, having a tree template of a question, a function can be designed (by the criteria considered) to traverse through the tree and to produce random numbers; with replacing the numbers produced diverse and similar questions to the question can be generated. These questions can be presented to students, and used as a tool of their learning assessment. Also, by using an appropriate programming environment, teachers can be helped to generate such questions to design standard, diverse questions.

Result and Conclusion: In this work, a CAS-based system was developed for teaching and automatically solving problems related to derivates. By the step-by-step representation of a problem, students can be helped to learn this mathematical subject. Also, the possibility of generating different solutions to a problem and automatically generating a similar new problem to the problem solved can improve students, learning capabilities. So, students will be able to automatically access the infinite number of questions, without spending cost and time, and observe their different solutions step by step. This feature helps them increase their ability to solve a problem, and prepares them for tests. On the other hand, utilizing the system teachers will be able to produce and use diverse questions based on templates designed for each grade, saving time.

References:

1. Bennett, G.: Calculus for general education in a computer classroom, In International DERIVE Journal, 2(2), pp 3-11, (1995).
2. Brown, R.: Computer algebra systems in the junior high school, In 3rd International Derive/TI-92 Conference, Gettysburg, PA, (1998, July).
3. Day, R. P.: Algebra and technology, In Journal of Computers in Mathematics and Science Teaching, 12(1), pp 29-36, (1996).
4. Drijvers, P.: Assessment and the new technologies, In International Journal of Computer Algebra in Mathematics Education, 5, pp 81-93, (1998).
5. Dunham, P. H., & Dick, T.P.: Research on Graphing Calculators, In The Mathematics Teacher, 87 (6), pp 440-445, (1994)
6. Heid, M. K.: The technological revolution and the reform of school mathematics, In American Journal of Education, 106(1), pp 5 – 57, (1997).
7. Heid, M. K.: An exploratory study to examine the effects of re sequencing skills and concepts in an applied calculus curriculum through use of the microcomputer, In Dissertation Abstracts International, (University of Maryland), (1984).
8. Heid, M. K.: Re sequencing skills and concepts in applied calculus using the computer as a tool, In Journal for Research in Mathematics Education, 19(1), pp 3-25, (1988).
9. Herget, W., Heugl, H. Kutzler, B., & Lehmann, E.: Indispensable manual calculation skills in a CAS environment, In Exam questions and basic skills in technology-supported mathematics teaching
17. Mayes, R. L.: The application of a Computer (Proceedings Portoroz 2000) pp. 13-26, Hagenberg, Austria: bk teachware, (2000).
10. Judson, P. T.: Effects of modified sequencing of skills and applications in introductory calculus, In Dissertation Abstracts International, (University of Texas at Austin, 1988), 49/06, (1988).
11. Judson, P.: Elementary business calculus with computer algebra, In Journal of Mathematical Behavior, 9(2) , pp 153 – 157, (1990).
12. Repo, S.: Understanding and reflective abstraction: Learning the concept of derivative in a computer environment, In International DERIVE Journal, 1(1), pp 97-113, (1994).
13. Runde, D. C.: The effect of using the TI-92 on basic college algebra students' ability to solve word problems, In Manatee Community College, ERIC Document Reproduction Services No. ED409046. (1997).
14. Smal, D.B., & Hosack, J.M.: Computer Algebra System, Tools for Reforming Calculus Instruction. In R.G. Douglas (Ed.), Toward a Lean and Lively Calculus, (143-155), Washington, DC: The Mathematical Association of America, (1986)
15. Tall, D.: Functions and calculus, In Dordrecht, the Netherlands (Vol. 1).: Kluwer Academic Publishers, (1996)
16. Kutzler, B.: Two-tier examinations as a way to let technology, In Exam questions and basic skills in technology-supported mathematics teaching (Proceedings Portoroz 2000) pp. 121-124, Hagenberg, Austria: bk teach ware, (2000).

Algebra System as a tool in college algebra, In

School Science and Mathematics, 2, pp 61-67, (1995).
18. Palmiter, J. R.: Effects of computer algebra systems on concept and skill acquisition in calculus, In Journal for Research in Mathematics Education, 22(2), pp 151-156, (1991).
19. Waits, B. K., and Demana, F.: Calculators in mathematics teaching and learning: past, present, and future, In M. J. Burke (Ed.), Learning mathematics for a new century: 2000 Yearbook pp. 51-66, Reston, VA: National Council of Teachers of Mathematics, (1999)

***

Karadeniz Technical University, Computer Engineering Department,Trabzon, Turkey
milani@ktu.edu.tr, Pehlivan@ktu.edu.tr, Hoseinpour@ktu.edu.tr