Symbolic Analysis as Universal Tool for Deriving Properties of Non-linear Algorithms – Case study of EM Algorithm

Vladimir Mladenović, Miroslav Lutovac, Dana Porrat

The Higher Technical School of Professional Studies
Nemanjina 2, 12000 Požarevac, Serbia

The University Singidunum
Danijelova 32, 11000 Belgrade, Serbia

The School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem 91904, Israel

Email: vlada@open.telekom.rs, mlutovac@singidunum.ac.rs, dana.porrat@huji.ac.il

Abstract: Many researchers start their work by studying theory in order to get better insight into measured phenomena. Sometimes they cannot get the numeric values of parameters used in published equations. This is even more complicated when the theory is statistical and there are no closed form deterministic solutions. In this paper we introduce an original approach and method to analyzing a popular and frequently cited tutorial paper on Expectation-Maximization (EM) algorithm. The original paper has sufficient information to understand the algorithm. Using tools of Computer Algebra System and methods of Symbolic Processing (SP), the typewriting errors are discovered and the re-derived equations are used for automatic generation of algorithmic code. The examples are evaluated using automatically derived code and the final numeric values agree with the values from the original paper. The derived results are used for further optimization, such as deriving the computational error in the closed form. From the closed form solutions, the precision can be derived in terms of number of iterations, or the minimal number of iterations can be expressed in terms of the required precision. This helps to optimize the algorithm parameters so that the algorithm becomes more efficient.

Keywords: Symbolic processing; EM algorithm; iteration; convergence; ML estimation; Computer algebra system

1 Introduction

Computer algebra is a field of computer science concerned with the development, implementation, and application of algorithms that manipulate and analyze expressions. Some of the issues are developing algorithms (such as algebraic algorithms, symbolic-numeric algorithms), studying how to build computer algebra systems (memory management, higher-order type systems, optimizing compilers), and mathematical knowledge management (representation of mathematical objects). Computer algebra is based on well-defined semantics, compose constructions, and algebraic algorithms. Symbolic computation is concerned with alternative forms, partially-specified domains, and symbolical derivations in general. Computer algebra systems is used to simplify rational functions, factor polynomials, find the solutions to a system of equations, give general formulas as answers, model industrial mathematical problems, and various other manipulations. Symbolic processing can be treated as a transformation of expression trees, such as symbols for operations ("+", "sin"), variables, constants, and simplification (for example, expression equivalence).

The EM (Expectation-Maximization) algorithm is analysed with a new type of processing, called symbolic processing. We illustrate the main drawbacks of numerical tools that have led to erroneous conclusions in the application of the algorithm through a practical example presented in [1].

Using symbolic processing, we derive the error function in closed form as a function of the number of iterations. Next, we determine the minimum number of iterations required to obtain a correct result, and as well as the required number of iterations as a function of the number of accurate bits of the final result of an algorithm. The main aim of this paper is to show how Computer Algebra (CA) and symbolic processing can be used in the design of numeric algorithms, and in discovering the properties of EM algorithm.

2 The Method of Symbolic Processing

Modern research in the field of engineering science usually starts with theoretical analysis. The next step can be simulation in order to prove the theoretical assumptions. The final step is practical implementation and measurements. Usually, the theoretical derivations are made on ideal parameter values, such in infinite number of samples or known signal values for time from $-\infty$ to $+\infty$. Simulation is used for simulation of systems in order to gain insight into their functioning. For a finite number of measured values, one can expect that the result agrees well with the simulation.

Probably, the first critical step is to set the simulation parameters and write a code that fits the theory. The second critical step is to choose the number of iterations or the number of input samples.

Symbolic processing can help to error-free derive simulation code, and to find typewriting errors in published results. This paper presents such an analysis. Next, symbolic processing can be used for finding the processing errors as the closed-form expression for computing the number of required iteration steps; or the error function due to finite word-length [2, 3]. Therefore, symbolic processing can help gain insight into how a system works, which is preferred to experimenting with numeric simulations.

Symbolic processing plays an important role in education. Even more it can be used in practice because the numerical tools may have problems with accuracy and a significant increase in the complexity. Primarily, the analysis area is essentially continuous and infinite [4]. This paper aims to highlight the role of symbolic processing applications, especially where the numerical approaches fail to produce the right algorithms that are important for many areas of engineering research, especially in communications.

Symbolic processing repairs certain disagreement between theoretical performance and numerical simulations. Numerical processing may show unsatisfactory results [5]. It can drive the researchers to erroneous conclusions. Using the symbolic tools, it is easy to find and prove mistakes and gain a better insight into the whole process of analysis, simulation and modeling.

Symbolic processing works with symbols. Any symbol can be replaced by a number when symbolic expressions become impractical, for example, for plotting a response to a specific excitation.

Therefore, writing programs using symbolic processing can be seen as a set of instructions that manipulate the symbols and can be used to perform a much wider range of activities [6]. The efficiency of symbolic processing becomes more important if systems and signals are more complex.

There is no algebraic solution for all forms of processing, and symbolic processing will not be able to solve all problems. Even then, the final result can be expressed in a form of special function that can be computed for known numerical values of system parameters.

In the next section, we will demonstrate the derivation of the EM algorithm using symbolic processing.

3 The Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm is ideally suited to estimate the parameters of a probability function, especially when direct access to all of the data is impossible [5]. A brief description of the algorithm can be found in literature [5] that is suitable for signal processing practitioners. An illustrative example is also presented there. Unfortunately, the algorithmic steps there disagree with the theoretical statements, and do not provide the numeric results given in Table of [5].

Those who want to test the knowledge learned from the example may be confused about the reason for such disagreements. The error can be in the initial statements, in the derived algorithmic steps, or it can be typewriting error.

A computer algebra system (specifically Mathematica) and symbolic processing are used in this paper to start analysis from the theoretical statements, derive the algorithmic code, and calculate the final results.

It is important to emphasize the role of the new analysis of the EM algorithm because EM is the backbone for algorithms that are used in significant analyse, measurement data processing and simulations [1, 7, 8].

The main goal of EM is approximation Maximum Likelihood from incomplete data. The most general form starts from the pdf of the incomplete data:

$$g(y|\theta) = \int_{\chi(y)} f(x|\theta)dx \tag{1}$$

where $f(x|\theta)$ is the probability density function of complete data, and θ is set of parameters.

An observation y determines a subset of χ that is denoted as $\chi(y)$.

The EM algorithm consists of two steps: the E-step and the M-step. The E-step (Expectation step) takes an expectation with respect to the unknown variables using the current estimate of the parameter and conditioned upon the observation. The M-step (Maximization step) then provides a new estimation of the parameters, in that it produces a maximum-likelihood (ML) estimates of parameters when there is many-to-one mapping. These two steps are iterated until convergence [5].

For the E-step compute:

$$Q(\theta|\theta^{[k]}) = E[\log f(x|\theta)|y, \theta^{[k]}] \tag{2}$$

where $Q(\cdot)$ is $Q-$ function.

For the M-step compute:

$$\theta^{[k+1]} = \arg\max_{\theta} Q(\theta \big| \theta^{[k]}) \tag{3}$$

4 An Example Problem

The example problem is to find the unknown value of *p* using the EM algorithm. In this example the initial value of p is defined as a symbol, and we know that its true value is 1/2 (as specified in the example in [9]). The application of EM to this example is calculated symbolically (using Mathematica), and we show mathematical description.

The expressions which are derived in this paper, were not derived in the paper [5] and cannot be obtained by hand. On the other side, the estimated parametar *p* is obtained with the value of 0.52 in the paper [5], but nowhere it explains the deviation from the true value of 0.5. We obtain the exact value of 0.5 and explain the way how we get it.

This section repeats Example from [5]. The application of EM to this example is calculated symbolically (using Mathematica), and we show mathematical description. Let a set of random variables is $\{X_1, X_2, \ldots, X_m\}$. For example X_1 represents the number of round dark objects, X_2 represents the number of square dark objects, and X_m represents the number of light objects. Let x_1, x_2, …, x_m be a sequence of outcomes of the random variables X_1, X_2, …, X_m that have been observed. It is assumed that X_i is independent of X_j for $i \neq j$. If the number of outcomes (total number of observed objects) is n, then x_i are nonnegative integers such that

$$\sum_{i=1}^{m} x_i = n \tag{4}$$

If X_1, X_2, …, X_m are mutually exclusive events with $P(X_1 = x_1) = p_1$, ..., $P(X_m = x_m) = p_m$, then the probability X_1 that occurs x_1 times, ..., X_m occurs x_m times is given by

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m) = n! \prod_{i=1}^{m} \frac{p_i^{x_i}}{x_i!} \tag{5}$$

where p_i are constants with $p_i > 0$ and

$$\sum_{i=1}^{m} p_i = 1 \tag{6}$$

The joint distribution of X_1, X_2, …, X_m is a multinomial distribution and $P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m)$ is given by the corresponding coefficient of the multinomial series

$$\left(p_1 + p_2 + \cdots + p_m\right)^n \tag{7}$$

Assume further that probabilities are also known As an example, assume that the objects are known to be trinomial distributed, that is $m = 3$, as in [5]. Assume further that probabilities are also known

$$p_1 = \frac{1}{4} \tag{8}$$

$$p_2 = \frac{1}{4} + \frac{p}{4} \tag{9}$$

Then, from (6) follows

$$p_3 = 1 - \left(p_1 + p_2\right) = \frac{1}{2} - \frac{p}{4} \tag{10}$$

We can use a notation for probability density function

$$f(x_1, x_2, x_3 \mid p) = P(X_1 = x_1, X_2 = x_2, X_3 = x_3) \tag{11}$$

For the unknown parameter p, the function becomes

$$f(x_1, x_2, x_3 \mid p) = \frac{n!}{x_1! x_2! x_3!} \left(\frac{1}{4}\right)^{x1} \left(\frac{1}{4} + \frac{p}{4}\right)^{x2} \left(\frac{1}{2} - \frac{p}{4}\right)^{x3} \tag{12}$$

Suppose that a feature extractor is employed that can distinguish the color of the objects, but cannot distinguish the shape. From the measurement, the number of dark objects is known, $y_1 = x_1 + x_2$, but the number of x_1 or x_2 is unknown. The basic idea of EM algorithm is to find expressions that can be used for iterated computing of x_1 or x_2 based on known $y_1 = x_1 + x_2$ and distribution function.

4.1 Expectation Step

In the E-step, the expected values of x_1, x_2, and x_3, have to be computed using initial estimate and measured data. The algorithm is derived using the knowledge of the multinomial distribution.

Starting with measured y_1 (where $y_1 = x_1 + x_2$) we have to derive expressions that can be used in iterated algorithm in order to find the number of objects x_1 and x_2, and the parameter p. In other words, we have to derive expected values of x_1 and x_2 for measured y_1 and estimate of p in the k th iteration

$$x_1^{[k+1]} = E[x_1 \mid y_1, p^{[k]}] \tag{13}$$

$$x_2^{[k+1]} = E[x_2 \mid y_1, p^{[k]}] \tag{14}$$

The expected values can be derived using the multinomial distribution (assuming that $x_1 + x_2 + x_3 = n$)

$$P(X_1 = x_1, X_2 = x_2, X_3 = x_3) = n! \frac{p_1^{x_1}}{x_1!} \frac{p_2^{x_2}}{x_2!} \frac{p_3^{x_3}}{x_3!} \tag{15}$$

Notice that $(X_1 + X_2, X_3)$ is binomial with class probability $(p_1 + p_2, p_3)$, then (assuming $p_1 + p_2 + p_3 = 1$, $y_1 + x_3 = n$) the probability $P(Y_1 = y_1, X_3 = x_3)$ becomes

$$P(Y_1 = y_1, X_3 = x_3) = n! \frac{(p_1 + p_2)^{y1}}{y_1!} \frac{p_3^{x3}}{x_3!} \tag{16}$$

Assume that the probability of some event x is known, say $P(x)$, and the number of events can be from 0 to n, the expectation value of x is defined by

$$E[x] = \sum_{x=0}^{n} x\, P(x) \tag{17}$$

Noticing that $y_1 = x_1 + x_2$ and $0 \le x_1 \le y_1$, expectation of x_1 requires to know conditional probability $P(x_1)$,

$$E[x_1] = \sum_{x=0}^{y_1} x_1\, P(x_1) \tag{18}$$

Assuming that y_1 has occurred, the conditional probability $P(x_1)$ equals

$$P(x_1) = \frac{P(X_1 = x_1, Y_1 = y_1)}{P(Y_1 = y_1)} \tag{19}$$

Applying those definitions on $x_1^{[k+1]}$, the iterated algorithm can be derived. Unfortunately, the presented relations in [5] disagree with expected.

4.2 Maximization Step

The maximum of some function, say $f(x_1, x_2, x_3 \mid p)$, can be obtained by taking the derivative of the function f with respect to parameter p, equating to zero, and solving for p. Since the logarithm is monotonically increasing, maximizing $\log(f)$ is equivalent to maximizing f.

The function f is known in terms of the parameter p, and we can try to solve for p the following equation

$$\frac{\mathrm{d}}{\mathrm{d}p}\log\left(f(x_1, x_2, x_3 \mid p)\right) = 0 \tag{20}$$

If solution exists, it becomes the second part of EM algorithm.

5 Implementation of Computer Algebra System Code into EM Algorithm

In this paper, we re-derive the expressions obtained in [5] because they do not produce the expected results in the numeric example presented in [5].

Computer algebra systems (CAS), such as Mathematica [10], can be used for derivation of all equations required for algorithm. The main motive is to avoid manual derivation and thus derive all relations without error.

All derivations are in a document called notebook that looks like technical paper in electronic form. There are cells with textual description, such as title, section head, text with formula, but also cells named input and output. In the input cells we write expressions that can be evaluated.

5.1 Entering Knowledge in CAS

The simplest way to re-derive equations is to start with definitions. The first line of the first input cell can be the definition of the trinomial distribution

$$\texttt{f[x1\_, x2\_, x3\_, p1\_, p2\_, p3\_] := (x1 + x2 + x3) !}\ \frac{\texttt{p1}^{\texttt{x1}}}{\texttt{x1 !}}\ \frac{\texttt{p2}^{\texttt{x2}}}{\texttt{x2 !}}\ \frac{\texttt{p3}^{\texttt{x3}}}{\texttt{x3 !}}$$

By executing the input cell there is no visible output cell. As a matter of fact, we define new knowledge that becomes a part of the CAS. The meaning of the code is that we define a function called **`f`**, a function with 6 arguments, (**`x1`**, **`x2`**, **`x3`**, **`p1`**, **`p2`**, **`p3`**), where instead of any argument with **_** we can use any other symbol or number (instead of **`x1_`** we can use x_1). The symbol **`:=`** tells to CAS to remember the expression that follows, but without execution, and so there is no output cell.

Once the knowledge is inputted to CAS, we can use for presentation or evaluation. The mathematical expressions may look different than a code. Let us replace symbolic arguments with symbols that appear in textbooks (use x_1 instead of **`x1`**), and present in the traditional form (transform the expression using command **`//TraditionalForm`**).

This expression still does not look same as equation (15). Therefore, we evaluate the expression again for the same symbolic arguments, and in the next line use the expression (% stays for previous expression) and apply substitution as in equation (4), and finally, display the result in the traditional form

```
In[2]:=  f[x₁, x₂, x₃, p₁, p₂, p₃] // TraditionalForm
```

Out[2]//TraditionalForm=

$$\frac{(x_1 + x_2 + x_3)!\, p_1^{x_1}\, p_2^{x_2}\, p_3^{x_3}}{x_1!\, x_2!\, x_3!}$$

Now, we obtain the same result as equation (15). This means that inputted knowledge is visually identical as in textbook. Notice that the last output is a result of the line that follows input In[3], that is actually input In[4], and the displayed output is Out[4].

The binomial distribution can be defined in a similar way

```
In[3]:=  f[x₁, x₂, x₃, p₁, p₂, p₃];
         % /. x₁ + x₂ + x₃ → n // TraditionalForm
```

Out[4]//TraditionalForm=

$$\frac{n!\, p_1^{x_1}\, p_2^{x_2}\, p_3^{x_3}}{x_1!\, x_2!\, x_3!}$$

This defines a function called f, a function with 5 arguments, (**`y1`**, **`y2`**, **`p1`**, **`p2`**, **`n`**). The head of the function is the same as for the trinomial distribution, but the difference is in the number of arguments. Instead of any argument with _ we can use any other symbol or number (instead of **`y2_`** we can use x_3, and instead of **`p1_`** we can use $p_1 + p_2$). Replacing arguments with appropriate symbols, and using the command for displaying expression in the traditional form, equation (16) is derived

```
In[6]:=  f[y₁, x₃, p₁ + p₂, p₃, n] // TraditionalForm
```

Out[6]//TraditionalForm=

$$\frac{n!\, p_3^{x_3}\, (p_1 + p_2)^{y_1}}{x_3!\, y_1!}$$

5.2 Derivation of The Expectation Step

In order to find out expectation of x_1, the conditional probability $P(x_1)$ should be computed using equation (19)

```
              f[x1, x2, x3, p1, p2, p3]
In[7]:= cp1 = ------------------------------------------ ;
              f[x1 + x2, x3, p1 + p2, p3, x1 + x2 + x3]
        Px1 = cp1 /. x2 → y1 - x1 // Simplify;
        % /. {p1 → p₁, p2 → p₂, x1 → x₁, y1 → y₁};
        % // TraditionalForm
```

Out[10]//TraditionalForm=

$$\frac{y_1!\, p_1^{x_1} (p_1+p_2)^{-y_1} p_2^{y_1-x_1}}{x_1!\,(y_1-x_1)!}$$

In the nominator the trinomial distribution is used, while in the denominator the binomial distribution is applied. The command **`Simplify`** is use to find the simplest form of the expression. Next, the derived expression is used in equation (18), and the final result is not the same as in [5]

```
               y1
In[11]:= ex1 = Σ  x1 Px1;
              x1=0
         % /. {p1 → p₁, p2 → p₂, y1 → y₁}
         % // TraditionalForm
```

Out[12]=

$$\frac{p_1\, y_1}{p_1+p_2}$$

Out[13]//TraditionalForm=

$$\frac{p_1\, y_1}{p_1+p_2}$$

The same procedure is used for deriving expected value of x_2, but this time the same result is as in [5]

In[14]:= $$\texttt{cp2} = \frac{\texttt{f[x2, x1, x3, p2, p1, p3]}}{\texttt{f[x1 + x2, x3, p1 + p2, p3, x1 + x2 + x3]}};$$

```
Px2 = cp2 /. x1 → y1 - x2 // Simplify;
```

In[16]:= $$\texttt{ex2} = \sum_{\texttt{x2=0}}^{\texttt{y1}} \texttt{x2 Px2};$$

```
% /. {p1 → p₁, p2 → p₂, y1 → y₁};
% // TraditionalForm
```

Out[18]//TraditionalForm=

$$\frac{p_2\, y_1}{p_1 + p_2}$$

Sum of expected values of x_1 and x_2 should be y_1, that is obvious from the derived expressions using CAS, but also proves that one of two results in [5] is not correct. Now, it is obviously that the sign – in the denominator of the expectation of x_1 is not correct in [5, Box 2, equation (37)]. Also, in the original text [5], it used y and y_1 for the same variable, that may be also confusing for some readers.

The presented derivation in this paper shows that all results can be found out automatically, and intermediate results and defined functions can be visually identified to be the same with those from textbooks.

Once the general expressions are derived, a particular solution follows as a special case. For example, if the probabilities are known, they can be denoted by subscript **e**

In[19]:= $$\mathbf{p_{1\,e}} = \frac{1}{4};$$

$$\mathbf{p_{2\,e}} = \frac{1}{4} + \frac{p}{4};$$

The expected value of x_1 for this specific case can be derived by substituting parameters in the general solution for x_1

In[21]:=
```
ex1 /. {p1 → p₁ₑ, p2 → p₂ₑ, y1 → y₁};
% // TraditionalForm
```

Out[22]//TraditionalForm=

$$\frac{y_1}{4\left(\frac{p}{4} + \frac{1}{2}\right)}$$

Following the same procedure, the expected value of x_2 is derived

```
In[23]:= ex2 /. {p1 → p1e, p2 → p2e, y1 → y1};
         % // TraditionalForm
```

Out[24]//TraditionalForm=

$$\frac{\left(\frac{p}{4}+\frac{1}{4}\right)y_1}{\frac{p}{4}+\frac{1}{2}}$$

The derived results differ from that in [5, equations (5) and (6)], and instead of p, the automatically derived expressions have $p/2$.

5.3 Derivation of The Maximization Step

For known probabilities of a specific case, the trinomial distribution can be expressed in terms of x_1, x_2, x_3, and the parameter p

```
In[25]:= f[x1, x2, x3, p1, p2, p3];
         % /. {p1 → p1e, p2 → p2e, p3 → (1 - p1e - p2e)};
         fe = % /. {x1 → x1, x2 → x2, x3 → x3};
         % /. x1 + x2 + x3 → n;
         % // TraditionalForm
```

Out[29]//TraditionalForm=

$$\frac{4^{-x_1}\, n!\left(\frac{1}{2}-\frac{p}{4}\right)^{x_3}\left(\frac{p}{4}+\frac{1}{4}\right)^{x_2}}{x_1!\, x_2!\, x_3!}$$

The same expression is presented in [5]. Since this is a continuous function in p, the first derivative can be computed

```
In[30]:= d = D[Log[fe], p] // Simplify
```

Out[30]= $\frac{(-2+p)\; x_2 + (1+p)\; x_3}{(-2+p)\;(1+p)}$

The parameter p can be expressed in terms of for the extreme (and again, the same expression is presented in [5])

```
In[31]:= sol1 = Solve[d == 0, p] // Flatten
```

Out[31]= $\left\{p \to \frac{2\, x_2 - x_3}{x_2 + x_3}\right\}$

But, the value of x_2 is unknown, Instead of x_2, we can use the expected value of x_2, that was derived in the expectation step. In order to obtain the same form of the final results as in [5], some rearrangements can be evaluated (to collect all terms in numerator and denominator that are multiplied by p)

```
In[32]:= sol1 /. x_2 → ex2;
         s = % /. {p1 → p_1e, p2 → p_2e, y1 → y_1} // Simplify
                        Collect[Numerator[p /. s], p]
         pnew = ─────────────────────────────────────
                   Collect[Denominator[p /. s], p]
```

Out[33]=
$$\left\{p \to \frac{-(2+p)\,x_3 + 2\,(1+p)\,y_1}{(2+p)\,x_3 + (1+p)\,y_1}\right\}$$

Out[34]=
$$\frac{-2\,x_3 + 2\,y_1 + p\,(-x_3 + 2\,y_1)}{2\,x_3 + y_1 + p\,(x_3 + y_1)}$$

The final result is not the same as in [5]. It seems that $2p$ is used instead of p in the recurrence relation in [5].

The presented derivation in this section shows that the final result can be found out automatically following the mathematical notation from textbooks or published papers.

5.4 Derivation of The Code

Once the expressions are derived from inputting the knowledge into CAS, the algorithm in the form of code can be simply transformed.

The code can be derived for some input parameters, such as the total number of objects, number of dark or light objects, initial guess for p, required accuracy, or probabilities

```
In[35]:= Clear[k, p]
         n = 100;
         y1 = 63;
         x3 = n - y1;
         p[0] = 0;
         a = 10^-7;
                 1
         p_1e =  ─ ;
                 4
                 1   p
         p_2e =  ─ + ─ ;
                 4   4
```

The expressions used in the code is a special case of the general expressions in terms of the known parameters or the previous value of p

```
In[43]:= p_new = pnew /. {x_3 → x3, y_1 → y1} /. p → p[k - 1];
         x1_new = ex1 /. {p1 → p_1e, p2 → p_2e} /. p → p[k - 1];
         x2_new = ex2 /. {p1 → p_1e, p2 → p_2e} /. p → p[k - 1];
```

Iterated algorithm can be a while loop, in that new number ob objects or new *p* is calculated

```
In[46]:= e = 1; k = 1;
         While[e > a, {
           p[k] = pnew;
           x1[k] = x1new;
           x2[k] = x2new;
           e = p[k] - p[k - 1];
           k = k + 1}]
```

Finally, the results of iterations can be viewed as table of values

```
Table[N[{i, x1[i], x2[i], p[i]}, 8], {i, 1, k - 1}] // TableForm
```

Number of displayed digits can be also specified

```
Out[48]//TableForm=
        1.0000000    31.500000    31.500000    0.37956204
        2.0000000    26.475460    36.524540    0.49029997
        3.0000000    25.298157    37.701843    0.51409288
        4.0000000    25.058740    37.941260    0.51883995
        5.0000000    25.011514    37.988486    0.51977276
        6.0000000    25.002255    37.997745    0.51995551
        7.0000000    25.000441    37.999559    0.51999129
        8.0000000    25.000086    37.999914    0.51999829
        9.0000000    25.000017    37.999983    0.51999967
        10.000000    25.000003    37.999997    0.51999993
        11.000000    25.000001    37.999999    0.51999999
```

The values from table are the same as in [5]. Therefore, using CAS, we derive all expressions and code for EM algorithm. Also, we avoid typewriting errors in the original text, prove all of the steps, and compute the numerical results for a specific case. For example, it is obvious that y_1 cannot be 100 [5, page 50 after equation (8)], because $n = 100$ and $y_1 < n$. A typewriting error is in the derivation of expectation step [5, page 53, Box 2] where index in the summation formula is from 10. Another typewriting error is in the derivation of expectation step [5, page 49, equation (6)] where x_2 should be used instead of x_1.

6 Analytical Solution of The EM Algorithm

The EM algorithm can be further analyzed by employing the symbolic expressions derived using CAS. Instead of the numeric analysis, a closed form expressions can be derived. This can help us to gain insight into how some system works and understand the properties of the algorithm.

6.1 Finding Closed Form Solution

Firstly, we generate an equation that describes iterated computing of parameter p, knowing the initial value

```
In[49]:= Clear[k, p]

In[50]:= eq1 = {p[k + 1] == (pnew /. p → p[k]), p[0] == 0}
```

Out[50]=
$$\left\{p[1+k] == \frac{-2\,x_3 + 2\,y_1 + p[k]\,(-x_3 + 2\,y_1)}{2\,x_3 + y_1 + p[k]\,(x_3 + y_1)},\ p[0] == 0\right\}$$

Command **`RSolve`** can be used to derive a closed form expression for p in terms of the iteration step k and other parameters kept as symbols

```
In[51]:= sol2 = RSolve[eq1, p[k], k] // Flatten // FullSimplify
```

Out[51]=
$$\left\{p[k] \to -\frac{2\,(x_3 - y_1)\left(\left(-1 - \frac{x_3}{y_1}\right)^k - \left(-3 - \frac{3\,y_1}{x_3}\right)^k\right)}{2\left(-1 - \frac{x_3}{y_1}\right)^k (x_3 - y_1) - (x_3 + y_1)\left(-3 - \frac{3\,y_1}{x_3}\right)^k}\right\}$$

In the specific case, when numeric values of some parameters are known, the closed form expression can be in terms of k, only

```
In[52]:= sol3 = sol2 /. {x₃ → x3, y₁ → y1} // Simplify
```

Out[52]=
$$\left\{p[k] \to -\frac{13\left(\left(\frac{100}{63}\right)^k - \left(\frac{300}{37}\right)^k\right)}{13\left(\frac{100}{63}\right)^k + \left(\frac{12}{37}\right)^k 25^{1+k}}\right\}$$

The numeric values of p in each iteration step can be displayed in table format

```
In[53]:= Table[N[p[k] /. sol3, 8], {k, 1, 11}] // TableForm

Out[53]//TableForm=
        0.37956204
        0.49029997
        0.51409288
        0.51883995
        0.51977276
        0.51995551
        0.51999129
        0.51999829
        0.51999967
        0.51999993
        0.51999999
```

Closed form expression can be used to find out the exact value after infinite number of iteration

```
In[54]:= Limit[p[k] /. sol3, k → +Infinity]
         % // N
Out[54]= 13/25
Out[55]= 0.52
```

Any of the parameters from the general solution can be changed to compute another set of values, such as for any value of n and specified exact value for p (named **pTrue**)

```
In[56]:= Clear[n]
         pTrue = 1/2;
         p1e = 1/4 /. p → pTrue;
         p2e = 1/4 + p/4 /. p → pTrue;
         y1 = Round[p1e n] + Round[p2e n];
         x3 = n - y1;
         Limit[p[k] /. sol2 /. n → 100, k → +Infinity]
         % // N
Out[62]= 13/25
Out[63]= 0.52
```

Since we already have a limit value as closed form solution in terms of n, the exact values can be computed for a range of different n

```
nrange = Range[50, 150];
pLimit = Limit[p[k] /. sol2 /. n → nrange, k → Infinity];
ListPlot[Transpose[{nrange, pLimit}],
  Filling → Axis, AxesOrigin → {50, 0.45}, AxesLabel → {n, p}]
```

Those results can be illustrated as it is shown in Fig. 1. The probability can be from the range 0.45 to 0.55, depending on n. Regardless of the fact that initial value of p is 1/2, named **pTrue** in the code, the computed value of is from the range 0.45 to 0.55 as exact result of the algorithm after infinite number of iterations. The exact value $p = 0.5$ can be obtained for a large number of objects, or as a mean value for several n.

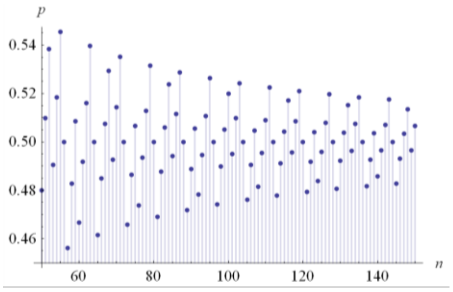

Figure 1
Estimate value of *p* in terms of the number of observed objects *n*

6.2 Finding Number of Iterations k

Usually, the errors in iterated algorithms are estimated. Since we already have the exact value after infinite number of iterations, and the current value at any iteration, the error can be expressed in closed form

```
error = (Limit[p[k] /. sol2 /. n → 100, k → +Infinity] -
       p[k] /. sol2 /. n → 100) // Simplify
```

$$\frac{494}{325 + 625 \left(\frac{189}{37}\right)^k}$$

$$a = \frac{1}{2^b}$$

$$2^{-b}$$

In addition, we specify the acceptable error as $a = 1/2^b$, where b is the number of accurate bits. Solving the equation when error is equal to required number of bits, we can determine b in terms of the iteration index k

```
sol4 = Solve[error == a, b] // Flatten // Simplify
```

$$\left\{b \to -\frac{\text{Log}\left[\frac{494}{325+625\left(\frac{189}{37}\right)^k}\right]}{\text{Log}[2]}\right\}$$

For tree different number of iteration, it follows that 4 iterations are sufficient for 8 bits,

```
b /. sol4 /. k → {3, 6, 10} // N
b /. sol4 /. k → {4, 7, 11} // N
```

```
{7.40333, 14.4561, 23.8672}

{9.7516, 16.8089, 26.22}
```

Figure 2 shows the number of accurate bits is liner function of number of iterations

```
Plot[b /. sol4, {k, 1, 11},
 PlotStyle → Thick, AxesLabel → {k, b},
 GridLines → {{4, 7, 11}, {8, 16, 24}},
 Ticks → {{0, 2, 4, 7, 9, 11}, {0, 8, 16, 24}}]
```

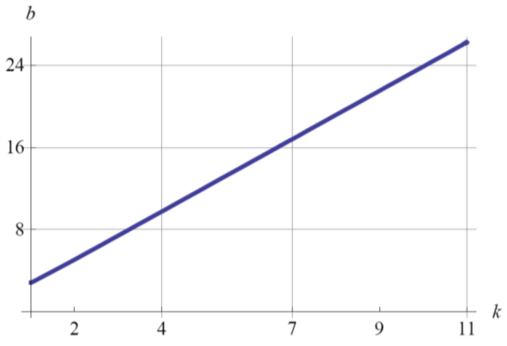

Figure 2
Number of exact bits of estimated *p* vs. number iterations *k*

Instead of number of accurate bits, we can set number of accurate decimal digits

$$a = \frac{1}{10^{\text{digits}}}$$

$$10^{-\text{digits}}$$

Solving the equation when error is equal to required number of accurate decimal digits, we can determine minimal number of iterations

```
sol5 = Solve[error == a, k] // Flatten // Simplify
```

$$\left\{k \to \text{Log}\left[-\frac{13}{25} + 247 \times 2^{1+\text{digits}}\, 5^{-4+\text{digits}}\right] \Big/ \text{Log}\left[\frac{189}{37}\right]\right\}$$

The minimal number of iterations is a function of digits

```
k /. sol5 /. digits → Range[0, 15, 3] // N
```

```
{-0.801956, 4.0911, 8.32723, 12.563, 16.7987, 21.0344}
```

For a 6-digit precision, the minimal number of iterations is 9

```
Ceiling[k /. sol5 /. digits → 6 // N]
```

```
9
```

Recalling the table with results, we can see that the 9th iteration has 6-digit precision.

This result can be used for fixed number of iterations instead of while loop. This is more suitable for numeric algorithms, especially when cash instructions can improve the speed of computing (no branch at the end of set of instructions).

In a similar way, by increasing n, the maximal number can be determined for the required precision. For n from 20 to 10000, 9th iteration are sufficient for 6-digit precision.

Conclusions

In this paper, we analyze EM (Expectation-Maximization) algorithm using a new approach and method named symbolic processing. We present a straightforward systematic procedure for automatically derivation of expressions starting from the basic knowledge of the phenomena, automatic derivation of the code, processing with the code and reevaluate the numeric results from published papers. Even more, a closed form solution can be derived, and symbolic optimization may be performed. The presented procedure can be used as template for symbolic processing instead of the classic numeric processing and manual derivations of new advanced challenging algorithms. We introduce an original approach to analyzing a popular and frequently cited tutorial paper on Expectation-Maximization (EM) algorithm, but with a number of typewriting errors. We prove that a more elegant and accurate solution can be obtained by using symbolic processing. Also, we provide better solutions for the exact values of the unknown parameters of probability.

References

[1] J. A. Fessler and A. O. Hero, “Space-Alternating Generalized Expectation Maximization Algorithm”, IEEE Trans. Signal Processing, Vol. 42, pp. 2664-2677, Oct. 1994

[2] M. D. Lutovac and D. V. Tošić, “Symbolic Analysis and Design of Control Systems using Mathematica”, International Journal of Control, Special Issue on Symbolic Computing in Control, Vol. 79, No. 11, 2006, pp. 1368-1381

[3] M. Lutovac, J. Ćertić, Lj. Milić, “Digital Filter Design Using Computer Algebra Systems,” Circuits Syst. Signal Process., Vol. 29, No. 1, 2010, pp. 51-64

[4] M. Lutovac, D. Tošić, “Symbolic Signal Processing and System Analysis”, Facta Universitatis, Electronics and Energetics, Vol. 16, No. 3, 2003, pp. 423-431

[5] T. Moon, “The Expectation-Maximization Algorithm”, IEEE Signal Processing Magazine, 1997, pp. 47-60

[6] M. D. Lutovac, V. M. Mladenović, “Development of Aeronautical Communication System using Computer Algebra Systems”, Proc. XXV Simposium of a New Technologies in Post Office and Telecommunication Traffic - PosTel 2007, Belgrade, December 2007

[7] B. H. Fleury, D. Dahlhaus, R. Heddergott, and M. Tschudin, “Wideband Angle of Arrival Estimation using the SAGE Algorithm”, Proc. IEEE Fourth Int. Symp. Spread Spectrum Techniques and Applications (ISSSTA ’96), Mainz, Germany, Sept. 1996, pp. 79-85

[8] D. Porrat, “The Diffuse Multipath Component and Multipath Component Visibility”, The 5th European Conference on Antennas and Propagation, EUR Congressi in Rome, Italy, EuCAP 2011, 11-15 April 2011

[9] M. D. Lutovac, V. M. Mladenović, “Post-Processing with Advanced Symbolic Simulation and Design using SchematicSolver and Mathematica”, 9th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services- TELSIKS – IEEE societies: MTT, AP, Communications, and Region 8, October 7-9, 2009, Nis, Serbia&Montenegro

[10] S. Wolfram, The Mathematica Book, Cambridge: Cambridge University Press, 2003