On Some Unsolved Problems of *Mathematica*

Evgenii V. Vorozhtsov
Khristianovich Institute of Theoretical and Applied Mechanics,
Siberian Branch of the Russian Academy of Sciences, 4/1 Institutskaya street,
630090, Novosibirsk, Russia
vorozh@itam.nsc.ru

***Abstract*— In this paper, some unsolved problems of the *Mathematica* software package are documented. At first, it is shown, using a number of examples, that the processing (simplification) of rational-fractional expressions involving powers in the general form, has been implemented in Maple more carefully than in *Mathematica*. Then, an error in *Mathematica* is demonstrated, leading to incorrect results at a change of a function's body. For each example of symbolic or symbolic-numeric computations, alternate routes for solving the emerged problem are proposed (where possible). An added problem, related to *Mathematica*'s computing speed is documented, employing examples related to two-dimensional gas dynamics problems. It is shown that *Mathematica* computes rather slowly (about one thousand times slower) compared to a Fortran code.**

Keywords-Mathematica; analytic; numerical; computations; simplification; expressions; Fortran

I. INTRODUCTION

Modern theoretical mathematics and the development of information technologies are strongly connected to powerful software packages, widely applied in the areas of science and engineering, such as *Mathematica* [1], which is one of the most powerful tools supporting scientific research, applied design, and teaching of specialists, postgraduates and students. *Mathematica* was designed as software mainly aiming to automate the work of scientists and mathematicians. The software contains a wide spectrum of mathematical methods and numerical algorithms for conducting theoretical and applied research as well as a powerful system for the visualization of obtained results. Starting from version 2, the subsequent versions of *Mathematica* are the world leaders in the area of symbolic mathematics for PCs, providing the execution of numerical computations, employing elegant graphical output forms, and also the execution of especially laborious analytic computations and transformations. *Mathematica* has been adapted to the most recent operating systems and hardware, making it available to most platforms including Microsoft Windows, Apple Mac and Linux.

Mathematica enables the performance of laborious analytic computations. However, the possibility of errors in the produced results due to internal software design is a rather important issue, considering that the source of such problems is rarely detected by the user. Even after the error has been localized, the end-user will need further time in order to replace the piece of his code where some built-in operators or *Mathematica* functions work incorrectly. Even though, it is usually possible to implement such alternate approaches, due to the large number of available functions, such internal errors and/or imperfections of *Mathematica* reduce its efficiency. Most importantly, in some cases, the erroneous results produced may be externally believable and thus not identified as such. Therefore, when there are no verifying mathematical relations, it is necessary to duplicate the analytic computations by using a different software system, for example, Maple, REDUCE, etc.

In the examples of analytic, symbolic-numeric and numeric computations presented in the following, the licensed copies of the following program packages were used: *Mathematica* 3.0.0 (license L2718-0816), *Mathematica* 4.1.0 (license L2989-2426), *Mathematica* 8.0.4.0 (license L3427-3055), and Maple 7 (license P/N: 01-0701-00-B-SN-A-E-0, serial number 693366959). Further, the licensed copy of the operating system Windows XP Professional (license E85-05798) was used.

The capabilities of seven general-purpose systems for analytic computations, namely Axiom, Derive, Macsyma, Maple, *Mathematica*, MuPAD, and REDUCE were analyzed previously [2] on 542 short problems. *Mathematica* produced completely wrong results in 13 of the 542 examples. Some added examples, not included in the above set, are documented in the present paper. In these cases, *Mathematica* either outputs an incorrect result or outputs a correct result having a bulky form, and one fails in its simplification with the aid of the built-in functions of the system. For each such example, alternate solving routes are proposed where possible.

II. INSUFFICNET EFFICIENCY OF SIMPLIFICATION FUNCTIONS

Consider the basic problem of mathematical analysis of the calculation of the sum:

$$S = 2 + 2\cdot 3x + 3\cdot 4x^2 + \ldots + n(n-1)x^{n-2}.$$

The *Mathematica* command:

```
S=Sum[j*(j-1)*x^(j-2),{j,2,n}],
```

where **Sum[…]** is the built-in function of the system, gives the following solution:

$$S=\frac{-2x+nx^n+n^2x^n+2x^{1+n}-2n^2x^{1+n}-nx^{2+n}+n^2x^{2+n}}{(-1+x)^3x} \quad (1)$$

It is clear that this expression can be simplified by dividing its numerator and denominator by *x*. Instead of doing this by hand using the pen and paper one can try to use the built-in function **Simplify[…]**, which is intended for the simplification of expressions. However, instead of the expected simplification of (1), the function produces the same result (1). Another built-in function, **PowerExpand[…],** enables the simplification of expressions involving power functions. But the application of this function to (1) also leaves this expression without change. The *Mathematica* function **Expand[S]** expands a complex fraction *S* into the sum of elementary fractions:

$$S=-\frac{2}{(-1+x)^3}+\frac{nx^{-1+n}}{(-1+x)^3}+\frac{n^2x^{-1+n}}{(-1+x)^3}+\frac{2x^n}{(-1+x)^3}-\frac{2n^2x^n}{(-1+x)^3}-\frac{nx^{1+n}}{(-1+x)^3}+\frac{n^2x^{1+n}}{(-1+x)^3}. \quad (2)$$

It can be seen from (2) that *Mathematica* has performed in each elementary fraction the divisions of both the numerators and denominators by the factor *x*. Therefore, one can now hope that the subsequent application of the function **Simplify**[…] to the sum of fractions (2) would yield the desired result. However, at the simplification of (2), *Mathematica* has multiplied both the numerator and denominator of the final expression by *x,* with the output being again the original expression (1). *Mathematica* also contains the function **Together**[…] which reverts the sum of fractions to a single fraction. Since there is no factor *x* in the denominators of (2), one may hope that this function will produce the result in the desired simplified form. However, the application of this function to (2) leads again to expression (1).

It should be noted that the Maple software produces the desired result for this task, using its **simplify** command, as shown below:

```
S:=simplify(sum(j*(j-1)*x^(j-2),j=2..n));
```

$$S=\frac{x^{(n+1)}n^2-x^{(n+1)}n-2x^nn^2+2x^n+x^{(n-1)}n^2+x^{(n-1)}n-2}{(x-1)^3}.$$

The problem of the ideal incompressible fluid flow around a motionless sphere is a relatively simple example of a three-dimensional fluid dynamics problem. The fluid flow is assumed to be directed at infinity along the $\overline{z}$ axis of a Cartesian coordinate system $O\overline{xyz}$. The solution of this depends on three dimensional coordinates $\overline{x},\overline{y},\overline{z}$. Let $\overline{u},\overline{v},\overline{w}$ be the dimensional components of the fluid velocity vector along the axes $\overline{x},\overline{y},\overline{z}$, respectively, and let $\overline{p}$ be the dimensional pressure of the fluid. The nondimensionalization of independent and dependent variables in the problem under study is given by:

$$x=\frac{\overline{x}}{R},\ y=\frac{\overline{y}}{R},\ z=\frac{\overline{z}}{R},\ u=\frac{\overline{u}}{U_\infty},\ v=\frac{\overline{v}}{U_\infty},\ w=\frac{\overline{w}}{U_\infty},\ p=\frac{\overline{p}}{\rho U_\infty^2},$$

where *R* is the dimensional sphere radius, U_∞ is the freestream velocity and ρ is the fluid density. It is assumed for definiteness that the sphere center lies in the coordinate origin (0,0,0) of the coordinate system *Oxyz*. The exact solution of the problem under consideration is then obtained with the aid of the velocity potential theory so that, for example, the velocity component *u* has the form:

$$u=-\frac{3xz}{r^5}, \quad (3)$$

where $r = (x^2 + y^2 + z^2)^{1/2}$. One can try to obtain expression (3) with the aid of *Mathematica* by performing:

```
r = Sqrt[x^2+y^2+z^2];fi=V0*r*
(1+1/2*(R/r)^3)*z/r; u = D[fi, x];
u1 = u/.{x-> R*x1,y-> R*y1,z-> R*z1}
u1a=PowerExpand[u1/V0];
Print["u1a = ",u1a];
```

The following notations are used here: **fi** is the velocity potential, **V0** = U_∞, **x1, y1** and **z1** are the dimensionless spatial coordinates and **u1a** is the dimensionless velocity component along the *x*-axis. The following expression is obtained for **u1a** as a result:

$$u1a=-\frac{3R^5x1\,z1}{2(R^2x1^2+R^2y1^2+R^2z1^2)^{5/2}}. \quad (4)$$

It is shown from (4) that the function **PowerExpand[u1/V0]** has "missed" the presence of the common factor $R^{2\cdot5/2} = R^5$ in the denominator.

```
u1b= Simplify[u1/V0]; Print["u1b = ",u1b];
```

$$u1b = -\frac{3R^5 x1\, z1}{2(R^2(x1^2 + y1^2 + z1^2))^{5/2}}.$$

The function **Simplify[u1/V0**] was able to detect the common factor R^2 only in the base of the power function.

And, finally,

```
u1c=PowerExpand[u1b]; Print["u1c = ",u1c];
```

$$u1c = -\frac{3x1\, z1}{2(x1^2 + y1^2 + z1^2)^{5/2}}. \quad (5)$$

It is seen from (5) that the combination of the functions **Simplify[…]** and **PowerExpand[…]** has led to the correct result.

It is interesting to solve the same task with Maple and to see how many steps are needed to obtain (5):

```
u:=-3*R^3*V0*x*z/(2*(x^2+y^2+z^2)^(5/2));
 u1c:=simplify(subs(x=x1*R,y=y1*R,z=z1*R,
          u/V0),assume=positive);
```

III. SOME PECULIARITIES OF THE PROCESSING OF COMPLEX QUANTITIES IN *MATHEMATICA*

Beside the capability of executing the basic arithmetic operations on complex numbers, in *Mathematica* the built-in functions **Re[z], Im[z], Conjugate[z], Abs[z],** and **Arg[z]** are also included. The purpose of each of these functions is clear from its name. The letter **I** is reserved in the system to denote the imaginary unit **i** = $\sqrt{-1}$.

Example:

```
z = 5 - 4I; Print["Re(z) = ", Re[z],
        "; Im(z) = ", Im[z]];
```

Answer: Re(z) = 5; Im(z) = –4.

Let now $z = \cos^2 x + \mathbf{i}\ \mathrm{tg}(x)/x$, where x is a real variable. The application of function **Re[z]** in this case gives the answer Re(z) = Re($\cos^2(x)$) – Im(tg(x)/x) because it is assumed in *Mathematica* that each of the functions $\cos^2(x)$ and tg(x)/x may be complex. The function **ComplexExpand[z]** performs the simplification of the complex number $z = a + \mathrm{i}b$ under the assumption that both expressions a and b are real.

Example:

```
z = Cos[x]^2 + I*Tan[x]/x;
   ComplexExpand[Re[z]]
```

The answer is obtained in the form Re(z) = $\cos^2 x$. However, the operation **zim1 = ComplexExpand[Im[z]]** leads to:

$$\mathrm{Im}(z) = \frac{\sin(2x)}{x(1+\cos(2x))} \quad (6)$$

instead of the expected:

$$\mathrm{Im}(z) = \frac{\mathrm{tg}(x)}{x}. \quad (7)$$

Using the well-known trigonometric relations it is easy to transform (6) to (7). The right-hand side of (6), however, requires 20 symbols, and the right-hand side of (7) requires 7 symbols, if one counts the horizontal bar separating the numerator from the denominator as a single symbol. The built-in function **ByteCount[expr]** available in *Mathematica* counts the number of bytes used internally by the system to store the expression **expr**. The call **ByteCount[zim1]** gives for the right-hand side of (6) the answer 488 bytes. Let us now try to simplify expression (6) with the aid of the function **Simplify[…]:** the command **zim2 = Simplify[zim1]** gives (7). **ByteCount[zim2]** gives 160 bytes. This means that the internal representation of (6) in *Mathematica* is three times longer compared to (7).

The danger of the appearance of bulky expressions like (6) in the process of symbolic manipulations consists of the fact that if (6) is used at the next stages of the analytic procedure then this may lead to a multiple "swell" of the final expressions. Further, bulky expressions deteriorate the visibility and understanding of final results. It is, therefore, desirable to execute the multi-stage analytic procedures in the interactive regime. As soon as a bulky expression of type (6) suddenly appears at a stage, one can try to simplify it with the aid of the following functions: **Simplify[…],ExpandAll[…]**, **FullSimplify[…],Expand[…], PowerExpand[…].**

Let us now see how the Maple system solves the task of finding the imaginary part of the complex number $z = \cos^2 x + \mathbf{i}$ tg(x)/x. The function **evalc(…)** in the Maple system is an analog of the function **ComplexExpand[…]** of *Mathematica*.

Thus, we have in Maple:

```
z:= cos(x)^2 + tan(x)/x*I; evalc(Im(z));
```

The answer is (7). Thus, the function **evalc(…)** immediately outputs the answer, which is optimal from the viewpoint of its length.

Let us return to *Mathematica*. In a search for a simple alternative way of obtaining Im(z) in the optimal form (7) one can use the following definition of the imaginary part of the number z: Im(z) is the coefficient affecting **i**.

To find the coefficients affecting some variable, the function **`Coefficient[…]`** is provided.

Let us use it:

```
z = Cos[x]^2 + I*Tan[x]/x;
zim = Coefficient[z, I]
```
(8)

In this case, *Mathematica* prints the following diagnostic message: "`Coefficient:: ivar : i is not a valid variable. >>`" and does not output any result. One can circumvent this situation by replacing **`I`** in (8) with a different letter, say **`j`**: **`Coefficient[z/.I -> j, j]`**. The answer is then obtained in the optimal form (7).

Summarizing the contents of this and foregoing sections one can draw the following conclusion: the simplification of analytic expressions has been implemented more carefully in Maple than in *Mathematica*.

IV. AN EXAMPLE OF INCORRECT WORK OF M*ATHEMATICA* AT A CHANGE OF A FUNCTION'S BODY

As a preliminary discussion, we assume that there is a variable parameter **`M`** in the user's *Mathematica* code, which must be specified at the beginning of the code. To conserve the information about already computed variants it is important to store in the program the information about already considered values of the parameter **`M`**. This is also convenient from the viewpoint that it will be easy to return again to one of the computed runs, which proved to be the best among all other variants. It is very simple to store the information about already used values of **`M`** in the following way: **`M = 5; M = 10; M = 15; M = 20`**. Here the last assignment operator cancels any of the foregoing assignment operators of the form **`M =`** …, so that one can be sure that the *Mathematica* program of the user will execute the next run with the value **`M = 20`**. This is always the case in practice.

Let us now consider the following *Mathematica* program:

```
f[x_]:=x*x/;x>1; expr1=y^2-f[x]*z/.x-> 2
```

Here the first command defines the function in the case when **`x>1`**. The system yields a correct result of the symbolic computation in the second command: $expr1 = y^2 - 4z$. Let us now assume that it is necessary to replace the body of the function **`f[x_]`** at some stage of the analytic computation with a different body, for example, with the body

```
f[x_]:=-x/;x ≠ 1;
```

It is obvious that this new definition of the function should cancel the foregoing definition of this function. Let us now assume that it is necessary to calculate the expression:

```
expr2=expr1^2-f[x]*z/.x-> 2;
```

in the next line of the *Mathematica* code. *Mathematica* outputs the following answer: $expr2 = (y^2 - 4z)^2 - 4z$. This answer is obviously incorrect because **`f[2]=-2`** in accordance with the new definition of function **`f[x_].`** That is, *Mathematica* has ignored the new definition of the function **`f[x_]:=-x/;x ≠ 1`** and continues using the old definition **`f[x_]:=x*x/;x>1.`** Below we present the solution of the given problem within the *Mathematica* framework:

```
f1[x_]:=-x/;x ≠ 1;
expr2a=expr1^2-f[x]*z/.f[x]-> f1[x]
expr2=expr2a/.x-> 2
```

This time, *Mathematica* has produced the correct result: $expr2 = (y^2 - 4z)^2 + 2z$.

V. SLOW NUMERICAL COMPUTATIONS

Many engineering problems are described by systems of partial differential equations. Common examples include problems of fluid dynamics, magnetohydrodynamics, elasticity theory etc. Problems whose solutions depend on two or three spatial variables and the time variable t are considered the most laborious computationally. Such problems are commonly computed with the use of advanced supercomputers and parallel computations.

The ability to perform an on-line analysis of the results obtained through the simulation of multidimensional problems increases significantly with the use of graphical representations. ANSYS Fluent (www.ansys.com) is a well known software that integrates not only the programs for the numerical solution of gas dynamics problems, structural mechanics, and electromagnetics but also a number of computer graphics procedures for the visualization of the results of multidimensional computations.

Since the *Mathematica* software had already integrated initially the means for symbolic manipulations, numerical computations and graphical visualization of the results with the aid of numerous built-in functions, it appeared to be attractive for the specialists concerned with numerical modeling of various applied problems. Examples of the numerical solutions of some relatively simple one-, two-, and three-dimensional problems of continuum mechanics were presented in [3], which may be computed with acceptable CPU time expenses within the framework of *Mathematica* at a fairly moderate total number of spatial computing mesh nodes. An attempt to apply *Mathematica* for the numerical solution of two-dimensional problems of gas dynamics on curvilinear spatial computing meshes was implemented successfully in [4]. But it turned out that the developed *Mathematica* code had a computational speed, which was many times smaller than the speed of the corresponding Fortran code. The developers of the considered system recommended the use of the built-in function **`Compile[…]`** to accelerate the numerical computations in *Mathematica*. However, its application reduced the CPU time needed for solving the two-dimensional

problems of gas dynamics only by the factor of 5/3 = 1.67 [4] rather than by the factor of 20, as it was declared in [1].

The Euler equations, which represent one of the basic mathematical models for gas dynamics problems, have the following form for the two-dimensional flow of an inviscid, non-heat-conducting compressible gas:

$$\partial w/\partial t + \partial f(w)/\partial x + \partial g(w)/\partial y = 0, \quad (9)$$

where x and y are the Cartesian spatial coordinates, t is the time, and

$$w = (\rho, \rho u, \rho v, \rho E)^T,\ f(w) = (\rho u, \rho u^2 + p, \rho uv, \rho uH)^T,$$
$$g(w) = (\rho v, \rho vu, \rho v^2 + p, \rho vH)^T. \quad (10)$$

Here $p,\rho,u,$ and v denote the pressure, density, and velocity components; $E = \varepsilon + (1/2)(u^2 + v^2)$, ε is the specific internal energy, $H = E + (p/\rho)$. The superscript T denotes the transposition operation. The ideal gas equation of state $p = (\gamma - 1)\rho\varepsilon$, $\gamma = \text{const} > 1$ is used. An explicit TVD method as applied to equations (9) and (10) was described in detail in [5], and therefore, the computational formulas of this method are not presented here. The method has either the second or third order of accuracy in spatial variables and in the subregions of smooth flow depending on the numerical value of the available weight parameter ϕ.

The method uses the flux limiters as well as the Roe averaging. As in other TVD methods, a switching from the higher approximation order to the first approximation order is carried out in the regions of shock waves and contact discontinuities for the purpose of obtaining monotonous profiles of numerical solution [5].

The problem of the oblique shock reflection from a wall is frequently used for verification of new numerical methods for solving (9) and (10). The exact solution of this test problem represents a piecewise constant function and is found numerically with machine accuracy with the aid of the theory of the oblique shock waves [3]. The value $\varphi = \pi/6$ for the angle between the incident shock and the x axis was used in the computational examples presented below, as shown in Figure 1.

Numerical solution was found by the pseudo-unsteady method. At the initial moment of time, the entire flow field was specified equal to the values of the undisturbed supersonic free stream, which were specified for subregion 1, as shown in Figure 1, that is the initial gas flow was parallel with the x-axis.

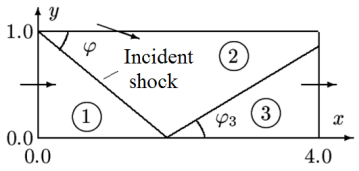

Fig. 1. Spatial region in the problem of the oblique shock wave reflection.

The criterion for the difference solution w^n convergence to the limiting stationary solution was taken in the form $Res(n) < \varepsilon$, where ε is the user-specified small positive number;

$$Res(n) = \max_{j,k} \left\{\max\left(|\, R^n_{1,j,k}\,|, |\, R^n_{2,j,k}\,|, |\, R^n_{3,j,k}\,|, |\, R^n_{4,j,k}\,|\right)\right\}, \quad (11)$$

where

$$\left\{R^n_{1,j,k}, R^n_{2,j,k}, R^n_{3,j,k}, R^n_{4,j,k}\right\}^T = \frac{\hat{f}^n_{j+1/2,k} - \hat{f}^n_{j-1/2,k}}{h_1} + \frac{\hat{g}^n_{j,k+1/2} - \hat{g}^n_{j,k-1/2}}{h_2},$$

n is the time layer number ($n = 0,1,2,\ldots$), $\hat{f}^n_{j\pm1/2,k}$, $\hat{g}^n_{j,k\pm1/2}$ are the difference approximations of fluxes $f(w)$ and $g(w)$ in (9) by the method of [5], h_1 and h_2 are the steps of the uniform rectangular computational grid in the (x,y) plane.

For the numerical solution of the given problem, two codes were developed: one in *Mathematica* and the other in Fortran 90. Both results have coincided. All the computations were carried out on a PC with an 3GHz Intel processor. The *Mathematica* code was run both in *Mathematica* 4.1.0 and in *Mathematica* 8.0.4.0. To execute 2600 time steps the *Mathematica* 4.1.0 and *Mathematica* 8.0.4.0 needed 22457 and 27312 seconds, respectively. Fortran needed only 21.45 seconds for the same computation. Thus, *Mathematica* 4.1.0 computes this two-dimensional task 1047 times slower than the Fortran code. In the case of *Mathematica* 8.0.4.0, the corresponding slow-down factor is equal to 27312/21.45 = 1273.29. The computations of the same problem were also done in *Mathematica* 4.1.0 and in Fortran 90 on a coarser grid of 80x20 cells, and 1600 time steps were done. In this case, *Mathematica* 4.1.0 needed 3455 sec. of CPU time, and the Fortran code needed only 3.36 sec. of CPU time. The corresponding slow-down ratio is 3455/3.36 = 1028.28. The discrepancy in the slow-down factors of 1047 and 1028 is explained by the well-known fact that a larger number of the intermediate printouts leads to larger CPU time expenses in *Mathematica*. In both runs — on the 160x40 grid and on the 80x20 grid — the information about the solution residual was printed every 50 time steps. On the 160x40 grid, 2600 time steps were executed, and on the 80x20 grid, a much smaller number of 1600 time steps were done.

One can assume with an insignificant error that the two-dimensional gas dynamics problems are solved in *Mathematica* 4.1.0 one thousand times slower compared to Fortran. In the case of *Mathematica* 8.0.4.0, the corresponding slow-down factor is higher and is equal to about 1270. Such unfavorable slow-down factors make the considered versions of *Mathematica* inapplicable for numerical solution of complex multidimensional problems of fluid dynamics.

It should be noted that *Mathematica* has been written using the C language. This language was developed at later times compared to the Fortran language. Therefore, the C compilers have absorbed the later developments in the area of the programming theory. In this connection, it is reasonable to expect that C compilers should generate exe files which should

be no less efficient compared to the exe files created by Fortran compilers. On this background, an extremely slow speed of numerical computations in the arithmetic of floating-point machine numbers in *Mathematica* looks rather strange.

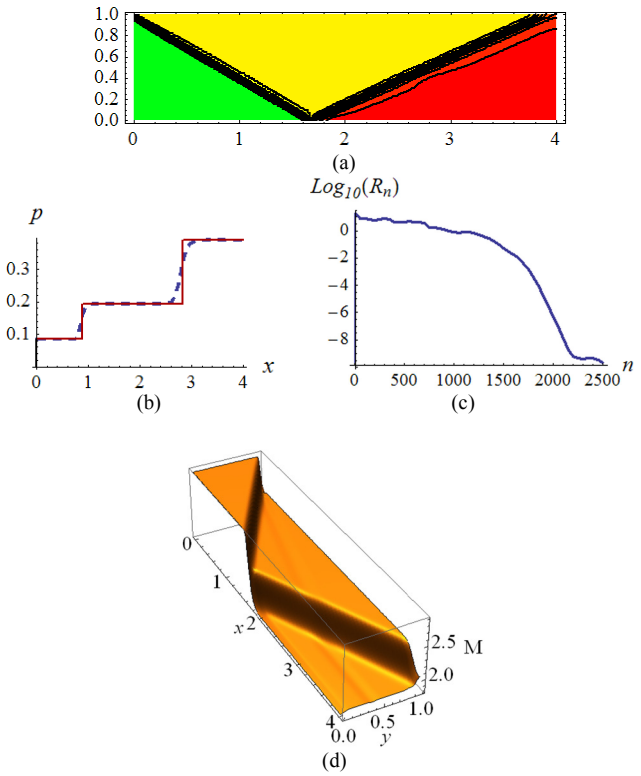

Fig. 2. The problem of the oblique shock wave reflection from a solid wall: (a) Mach number contours; (b) gas pressure in section $y = 0.5$; (c) residual (11) as a function of the number of time steps n; (d) the surface $M = M(x,y)$.

Mathematica has a large number of the built-in functions of computer graphics, which enable the obtaining of high-quality graphs of curves, pictures of surfaces, contour plots, etc. In order to use these possibilities for the purpose of a rapid on-line visualization of the results of numerical solution of complex multidimensional problems of fluid dynamics and mathematical physics the present author has implemented a simple interface between Fortran and *Mathematica*. The realization of this interface reduces to the following: upon termination of the basic computation by Fortran, an output of the computed results to the external file or several external files in *Mathematica* format was implemented. And a small program is then written in *Mathematica* which ensures the graphical visualization of computed results in a form suitable for the user. The exit from the Fortran shell and entry into the *Mathematica* program needs about 3 seconds of the user's time, and then the *Mathematica* program is started by pressing the keys Shift/Enter. It reads the arrays of numerical results from the external files created by the Fortran code, and the user can see in a few seconds the computed results in a graphical form on the monitor screen.

Figure 2 shows the results of numerical computation, by Fortran, of the problem of the oblique shock wave reflection from a solid wall, which were obtained by the steadying computation using the method of [5] on a uniform grid of 160x40 cells; the parameter of the method $\phi = 1/3$, which ensures the third-order accuracy of the method in spatial variables in smooth flow subregions. All these graphical results were obtained with the aid of the above-described small *Mathematica* 8 program, which reads the arrays of numerical results from external files created by Fortran. Figure 2(a) shows the lines of the constant Mach number $M = \sqrt{u^2 + v^2}/c$, where $c = \sqrt{\gamma p/\rho}$ is the sound velocity. The solid line in Figure 2(b) is the exact solution, and the dashed line is the result of the computation by the method of [5]. Figure 2(c) shows the residual (11) in the case of the method of [5]. It can be seen that 2400 time steps are needed to ensure a drop of the solution residual norm to the level of machine round-off errors $10^{-10} - 10^{-12}$.

VI. CONCLUSIONS

1. As the above-presented examples show, the processing (simplification) of the rational-fractional expressions involving power functions is implemented in Maple more thoroughly than in *Mathematica*.
2. An error in *Mathematica* is demonstrated, which leads to an incorrect result of the user's program at a change of a function's body.
3. It is shown by the example of numerical solution of a well-known model of a two-dimensional gas dynamics problem that the two-dimensional problems of gas dynamics are computed slower in different versions of *Mathematica,* by factors ranging from 1000 to 1273 compared to Fortran. Such an unfavorable ratio of the CPU time expenses makes the considered *Mathematica* versions unacceptable for numerical solution of complex multidimensional fluid dynamics problems.

REFERENCES

[1] S. Wolfram, The *Mathematica* book, 3rd Edition, Cambridge University Press, Cambridge, UK, 1996

[2] M. Wester, "A critique of the mathematical abilities of CA systems" In: Computer Algebra Systems: A Practical Guide, John Wiley & Sons, Chichester, United Kingdom, pp. 25–60, 1999

[3] S. P. Kiselev, E. V. Vorozhtsov, V. M. Fomin, Foundations of fluid mechanics with applications: Problem solving using *Mathematica.* Birkhäuser, Boston, 1999

[4] V. G. Ganzha, E. V. Vorozhtsov, "Implementation of aerodynamic computations with *Mathematica*", 2nd Workshop on Computer Algebra in Scientific Computing, pp. 101–114, Munich, 1999

[5] S. R. Chakravarthy, S. Osher, "A new class of high accuracy TVD schemes for hyperbolic conservation laws", AIAA Paper, No. 85-0363, 1985