42nd AIAA Aerospace Sciences Meeting and Exhibit
05–08 January 2004, Reno, NV

AIAA Paper #2004–0242

CFD with Hybrid Symbolic–Numeric Methodology

José A. Camberos*
U.S. Air Force Research Laboratory, Wright–Patterson Air Force Base, Dayton, OH 45433

Larry A. Lambe†
Multidisciplinary Software Systems Research Corporation, Bloomingdale, IL 60108
and
Department of Informatics, Division of Mathematics
University of Wales, Dean St., Bangor, Gwynedd LL57 1UT, United Kingdom

Richard Luczak ‡
Science Applications International Corporation, Beavercreek, OH 45431

Advances in computer capabilities make the process of generating numerical solutions to complex partial differential equations fairly routine. At the same time, computer power now allows the direct manipulation of mathematical symbols and exact solutions. Symbolic computation engines now widely available include several successful commercial varieties like MATHEMATICA, MAPLE, and MATLAB. Given this, it seems that the time is at hand to revisit the original problem of solving partial differential equations using a combination of numerical calculation and symbolic manipulation. Harnessing the power and potential of symbolic computation requires a re–orientation of the computer and numerical analysts' perspective to achieve advances beyond current techniques. The new methodology emerging re–opens the research field in computational fluid dynamics to possibilities only dreamed of in its infancy. Computer power at the time, however, forced the research direction towards fully discretized, purely numerical calculations. We present several examples of this emerging methodology in re–visiting some familiar, textbook problems. This will introduce the reader to the hybrid methodology with the hope of inspiring new thought in utilizing symbolic manipulation to solve problems in mathematical physics from a fundamental perspective.

Nomenclature

a	generic coefficient
h	mesh spacing
j	space index (subscript)
n	time index (superscript)
t	time variable
u	generic unknown variable
x	space variable
α	generic diffusion coefficient
ν	viscosity coefficient
ε	generic error
Re	Reynolds number
HSN	acronym: **H**ybrid-**S**ymbolic **N**umeric

*Aerospace Engineer and AIAA Senior Member
†Chief Scientist, MSSRC (llambe@mssrc.com)
‡Principal Systems Engineer, SAIC (luczakr@saic.com)

Introduction

Overview

We introduce a method for solving some model equations typically used in testing algorithms based on finite–difference approximations to space and time derivatives. The method is based on utilizing symbolic computation capabilities presently available. First, we give a limited historical overview of symbolic computation and follow with a detailed development of how this power can be used to develop semi–analytic (hybrid) solutions to certain model equations. In this paper, we deal with linear hyperbolic and parabolic equations and present (very) preliminary results for a nonlinear hyperbolic equation. Subsequent works now in progress will deal with elliptic equations and the challenge of boundary–value problems and nonlinear systems of equations. Our long–term vision is to solve the equations of mathematical physics, from solid/fluid mechanics to electromagnetism. The short–term goal is to demonstrate that symbolic computation offers a practical, accurate, and robust methodology for solving nonlinear systems of equations, including, for example, the Euler equations of fluid dynamics.

Symbolic Computation

The use of computers for symbolic computation has been actively pursued since the early years of machine computation. Indeed, FORTRAN (an acronym for *for*mula *tran*slation) was considered to *be* symbolic computation when it was first introduced,[1].[2] In fact, the advent of FORTRAN was a major step in the direction of bringing algebra to machine computation – one really could program in terms of (simple) algebraic formulas using that language. An extension of FORTRAN called ALTRAN was developed by W. S. Brown of Bell Labs ca. 1968.[3] See also Hall (1971)[4] and the articles by Rink and Guru.[5,6]

Another approach to symbolic computation in the 1960's and 70's used formula manipulation algorithms,[7] artificial intelligence techniques,[8] pattern matching,[9] and rewrite rule theory.[10,11] The idea was to begin with a set of "simplifying transformations" which apply in some obvious cases and, using that, attempt to develop a coherent scheme applicable to a larger class of expressions. A good survey of such activities at the time is given in Sammet.[7] There are at least two problems with this approach: Depending on what definition of "expression" is used, there are classes for which the above approach cannot succeed[12] and the kinds of algorithms involved tend to be rather complex (in space and time). Typically, such systems introduced a new programming language, emphasized exact computation (infinite precision), depended on LISP or an underlying kernel and interpreted. These systems allowed for the conventional manipulation of expressions (sum, difference, product, derivative, etc.), although they were not very convenient for use with codes written in other languages like FORTRAN, C, etc. Typically, they offered no user control over execution of internal transformations, which could, at times, lead to confusing or incorrect results. While rewrite rule transformations can be quite useful, they are best used at a high–level in a controlled manner and not as a foundation for basic mathematical operations. In spite of the deficiencies, systems arose which could manipulate mathematical expressions symbolically and which were based on such in methods. For example, MATHLAB (MITRE, 1964, 1968),[13] REDUCE (University of Utah,[14] 1968), and MACSYMA (MIT,[15] 1970's). All these systems were implemented using LISP.

In the 1980's and 90's, a later generation of systems such as MAPLE (University of Waterloo,[16] 1985), MATHEMATICA (WRI,[17] 1988), and AXIOM (IBM 1970's–90's and Numerical Algorithms Group 90's[18,19]) as well as several others (a more complete list can be found at http://www.can.nl/) were based on more modern software engineering techniques. The designers of most of these newer systems made an effort to balance the user's wishes and the features of those systems, but they are still far from reflecting the typical way in which researchers think about and solve problems. Systems of this kind have many similarities, including:

- Modern (graphical) user interfaces,
- introduction of a new programming language,
- an interactive, interpreted environment,
- the usual operations on expressions such as $+$, , $*$, / as well as symbolic differentiation and integration (to some extent),
- built–in routines for solving various classes of equations,
- designed to be a convenient environment for prototyping ideas,
- by their particular unique nature, interaction with other software is not easy and requires special effort,
- by default, they emphasize exact symbolic computation as opposed to approximate numerical computation.

A completely different approach to symbolic computation, *ExprLib* (MSSRC, 2000[20]), was designed to address some stringent needs of application developers and problem solvers in the area of scientific computing. *ExprLib* has some similarities with other symbolic computation systems in the sense that allow for manipulating expressions, symbolic differentiation, etc. However, it differs from the systems mentioned above in several important ways: (i) It is written in ANSI C and is a compiled library; (ii) applications using the library can therefore be written in standard ANSI C; (iii) the default is to provide expressions with double coefficients (exact coefficients are an option); (iv) expressions are essentially algebraic combinations of compiled functions, e.g. when the user manipulates expressions containing $\sin(x)$, the actual compiled function from the standard ANSI C math library is involved (making calculations very efficient).
Several advantages of the way *ExprLib* is designed and implemented are immediately evident. For example:

- Applications can easily interact with other C or FORTRAN libraries.
- While no pattern matching or rewrite rules are used in implementing low level mathematical operations in the library, higher level rewrite rule transformations can be defined and precisely controlled by the user.
- Problems of industrial strength size may be solved with no cumbersome interface efforts.

- Fast and efficient numeric evaluation of expressions are possible.

- Using today's web/networking technology, it is easy to provide web/browser based wrappers for *ExprLib* applications ("instant user interfaces" for applications).

- By design, the library is a hybrid symbolic–numeric tool.

A reasonable analogy is that *ExprLib* is to mathematical programming as `libc` is to general ANSI C programming, but there are facilities that go beyond such a simple analogy. At the most basic level, *ExprLib* was designed to provide very precise standard, efficient, and robust ways to decompose mathematical expressions into their parse trees, efficiently apply classes of well–defined transformations to such parse trees, and recombine such transformed parse trees back into expressions. The ability to do this and do it efficiently and robustly is quite powerful. For example, it is key to computing complex integrals analytically in closed form as noted in Lambe, et. al.[21] – it allows one to create complex operators with user defined behavior – it even allows the construction of non–smooth and discontinuous operators.

Goals of HSN Computation in CFD

A search on the phrase "hybrid symbolic" using the "Anywhere" field of the online version of Math Reviews <http://www.ams.org/mathscinet/search> returned only 6 hits (on October 23, 2003). Using the phrase "symbolic-numeric" on the same day, 45 hits were returned most of which were not directly related to scientific/engineering problem solving. The search string "hybrid symbolic-numeric" returned 536 hits using advanced search in Google, also on the same day, most of which concerned talks, courses, and statements about research programs (as opposed to published papers). We conclude that the area is still young. Our specific goals for applying HSN to solve the equations of CFD, we include:

- The derivation of approximate analytic solutions of general equations of physics, regardless of their dimensionality, complexity, or nonlinearity,

- parameter studies (by deriving solutions as functions of parameters, e.g. Reynolds number),

- solving otherwise unsolvable CFD problems (e.g. those with arbitrarily high Reynolds numbers),

- investigation of the direct nonlinear interactions between inertia and viscosity at all spatial scales,

- the derivation of arbitrarily high accurate numerical schemes,

- the generation of efficient and user-readable ANSI C/FORTRAN code automatically,

- the derivation of new classes of numerical schemes for CFD (as has been done in electromagnetics).[21]

Work in the areas just cited is currently underway and our attempt complements the direction taken by Poondru and Abdallah[22] for matrix inversion and Verhoff[23] for analytic solutions to the Euler equations. Results and recommendations obtained will be reported in future papers and publications. To begin the realization of these possibilities, we start by rethinking the standard approach to numerical solutions in CFD, starting with the basic model equations classified mathematically as hyperbolic, parabolic, and elliptic. An example of a scalar non–linear equation will also be given, with full results given in a future write–up. We chose *ExprLib* in this work for the reasons stated above, but the content of this paper is independent of any system or language, of course – indeed, we believe that a clever programmer can program almost anything in almost any language. The interested reader is encouraged to implement the ideas we present in any preferred language or system.

Hyperbolic Model Equation

Advection Equation

A basic equation typically used to introduce the student to computational fluid dynamics models the physics of pure advection:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0. \tag{1}$$

We assume a non–zero constant wave speed a and the solution domain is $[0,1] \times [0, t_{max}]$. Let the initial conditions be given by $u(x,0) = u_0(x)$, with periodic boundary conditions $u(0,t) = u(1,t)$. The exact solution $u(x,t) = u_0(x-at)$ is known and easily verified. The solution represents an arbitrarily shaped pulse which is swept along at constant speed a in the positive x direction without changing shape.

There are many methods available to solve the equation numerically. The standard numerical approach requires a discrete approximation to all derivatives, typically a finite–difference formula in space followed by a time–integration technique. It is well known that a fully numerical solution to equation (1) is subject to time–step limitations, namely the CFL condition $a\Delta t/\Delta x \leq 1$. For example, consider here a typical method, explicit Lax–Wendroff,[24,25] that is second–order accurate:

$$u_j^{n+1} = u_j^n - \frac{a\Delta t}{2h}\left(u_{j+1}^n - u_{j-1}^n\right) + \frac{a^2\Delta t^2}{2h^2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \tag{2}$$

where, in standard notation, superscript n refers to the time index and subscript j refers to the space index.

The Lax-Wendroff formula gives a numerical approximation to the equation (1) using centered–differences in space. Recalling its original development, this technique was constructed from the Taylor series

$$u(x,t+\Delta t) = u(x,t) + \Delta t\frac{\partial u}{\partial t} + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t^2} + \ldots + \frac{\Delta t^m}{m!}\frac{\partial^m u}{\partial t^m}. \quad (3)$$

For small enough Δt, the series is convergent. It is possible to prescribe a solution method by replacing all time derivatives using Equation (1). This gives

$$u(x,t+\Delta t) = u(x,t) - a\Delta t\frac{\partial u}{\partial x} + \frac{a^2\Delta t^2}{2}\frac{\partial^2 u}{\partial x^2} + \cdots + \frac{(-a\Delta t)^m}{m!}\frac{\partial^m u}{\partial x^m}. \quad (4)$$

Assuming that the prescribed initial condition is sufficiently differentiable, we can truncate the series at any given order and use symbolic computation to evaluate the spatial derivative analytically. For example, consider the semi-discretized version of the Lax-Wendroff method, which is truncated at the second-order derivative:

$$u^{n+1}(x) \approx u^n(x) - a\Delta t\frac{\partial u^n}{\partial x} + \frac{a^2\Delta t^2}{2}\frac{\partial^2 u^n}{\partial x^2}. \quad (5)$$

By calculating the derivative in a continuous fashion using symbolic computation, a solution to the advection equation is obtained using this explicit, one–step prediction formula (5). We will call this the one-step Hybrid Symbolic Numeric (HSN) method.

A variety of other methods follow immediately by utilizing this technique. For example, the predictor-corrector, well–known as Heun's method, is

$$\begin{aligned} u^* &= u^n - a\Delta t\frac{\partial u^n}{\partial x} \\ u^{n+1} &= u^n - \frac{a\Delta t}{2}\left(\frac{\partial u^*}{\partial x} + \frac{\partial u^n}{\partial x}\right) \end{aligned} \quad (6)$$

Again, the spatial derivatives can be evaluated symbolically, given the initial condition $u_0(x)$. In fact, the generalization of this approach follows immediately by recognizing that this is the same technique used in the numerical solution of an ordinarydifferential equation (ODE). We can therefore write the hybrid symbolic-numeric (HSN) version of the two-step Runge-Kutta method

$$\begin{aligned} u^* &= u^n - \frac{a\Delta t}{2}\frac{\partial u^n}{\partial x} \\ u^{n+1} &= u^n - a\Delta t\frac{\partial u^*}{\partial x} \end{aligned} \quad (7)$$

and of course the workhorse of ODE numerical methods, the four-stage Runge-Kutta method:

$$\begin{aligned} u^* &= u^n - \frac{a\Delta t}{2}\frac{\partial u^n}{\partial x} \\ u^{**} &= u^n - \frac{a\Delta t}{2}\frac{\partial u^*}{\partial x} \\ u^{***} &= u^n - a\Delta t\frac{\partial u^{**}}{\partial x} \\ u^{n+1} &= u^n - \frac{a\Delta t}{6}\left(\frac{\partial u^n}{\partial x} + 2\frac{\partial u^*}{\partial x} + 2\frac{\partial u^{**}}{\partial x} + \frac{\partial u^{***}}{\partial x}\right). \end{aligned} \quad (8)$$

Other well–known "classical CFD methods" like MacCormack's predictor-corrector method[26] can be transformed into a hybrid symbolic-numeric technique:

$$\begin{aligned} u^* &= u^n - a\Delta t\frac{\partial u^n}{\partial x} \\ u^{n+1} &= \tfrac{1}{2}\left(u^n + u^*\right) - \frac{a\Delta t}{2}\frac{\partial u^*}{\partial x}. \end{aligned} \quad (9)$$

For a linear equation with constant coefficient as above, all these methods actually reduce to the one-step HSN method, to some order m indicating the truncation term in the Taylor sequence (4). For non-linear equations, these formulas offer a variety of solution techniques.

Other methods like the two–step version of Lax–Wendroff and the Warming–Beam method can be shown to reduce to one of the above when written in hybrid form. The evident trend here seems to indicate a shift in how the solution to a PDE like (1) should be approached. In standard numerical methods, the PDE is often first discretized in one variable (like space, requiring mesh generation) and then each point subjected to time integration, either point-by-point as in explicit methods or for a collection of points as in implicit methods. In the examples given above, the spatial continuity is retained and time integration applied to the function itself, not a collection of numerical values at given points. This requires no mesh generation and is effectively spatially continuous. The accuracy of the technique can be compared to standard methods by example. We first note that the examples below are simply first–steps towards synthesizing the capabilities of symbolic computation with numerical algorithms that have matured the field of computational fluid dynamics. The hope is not to supersede these methods but to harness the potential of symbolic computation to complement and enhance the power of numerical techniques.

Example 1: Smooth initial function

Consider the case when $a = 1$ in Equation (1) and the initial function u_0 is given by

$$u_0(x) = \sin(2\pi x). \quad (10)$$

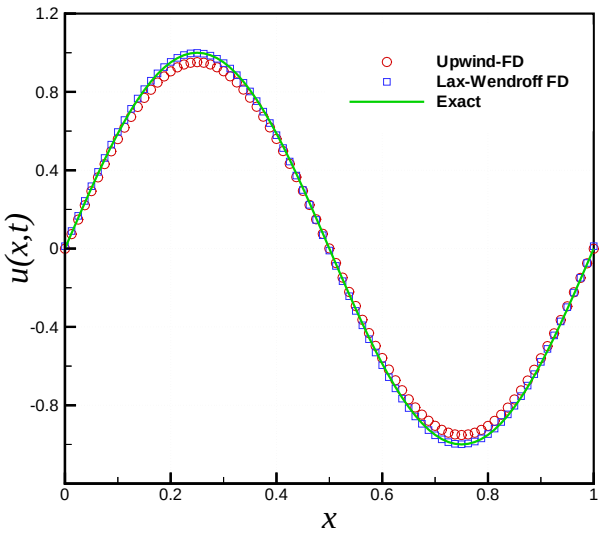

Figure 1 Finite–Difference solution with Lax–Wendroff method.

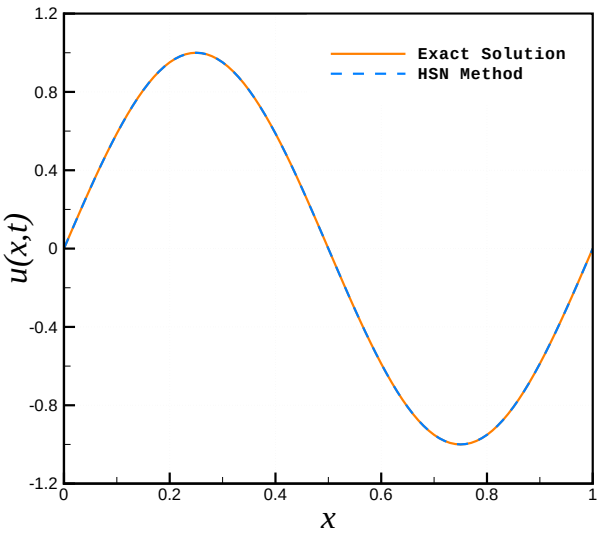

Figure 2 Hybrid Symbolic–Numeric solution based on one-step iterative method.

The exact solution is easily obtained as

$$u(x,t) = \sin\left[2\pi(x-at)\right]. \tag{11}$$

At any given time $t^n = n\Delta t$, the approximate hybrid solution can be expressed in the form

$$u_{HSN}(x) = A_n \sin(2\pi x) + B_n \cos(2\pi x) \tag{12}$$

where the coefficients A_n and B_n depend only on t^n. Clearly, after one period, when $t = 1$, one should have $A_n \approx 1$ and $B_n \approx 0$.

Although a variety of finite-difference formulas are available, here we will compare only the upwind, Lax-Wendroff, and Runge-Kutta varieties with the hybrid symbolic–numeric method described above.

Figure (1) shows the finite–difference upwind and Lax-Wendroff techniques with a grid of 81 discrete points. Figure (2) shows the hybrid symbolic-numeric method using the one-step Taylor series formula (5) truncated at second–order. Comparing the two figures graphically illustrates the ideas we are attempting to convey: Utilizing the symbolic capability available in modern computers allows one to obtain partial or fully analytic solutions to the problems of mathematical physics. In addition, whereas the finite-difference solution is limited by the CFL condition requiring small time-steps, the hybrid solution has no CFL restrictions. However, a limit on the time steps is imposed due to the truncation error that creeps in, requiring the inclusion of more terms in the Taylor series. For a finite–difference solution, the time step limit is $\Delta t \le a\Delta x$, which gives $\Delta t = 0.0167$ for this example. The hybrid method truncated at second order has a limit of about 0.10 for Δt; larger time steps are possible by adding more terms in the series (4).

To measure the accuracy of the HSN solution numerically, we may represent the error as

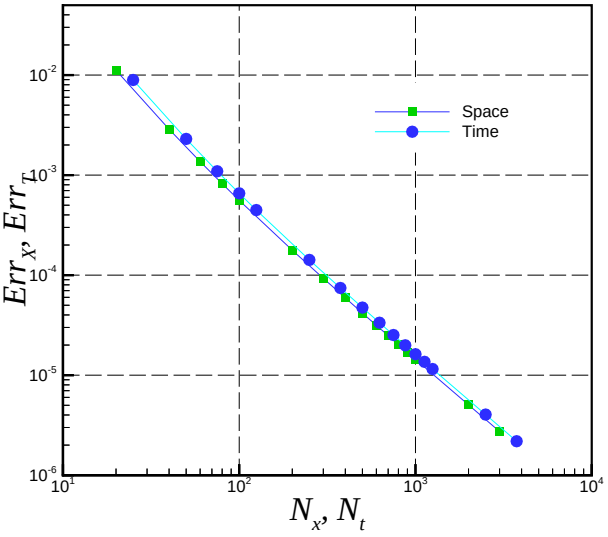

Figure 3 Numerical error associated with finite-difference method (Lax-Wendroff).

$$\varepsilon \equiv u_{HSN}(x) - u(x,t^n), \tag{13}$$

where $u(x,t^n)$ is the exact solution at $t = t^n$. The accuracy is thus quantified by the norm

$$\|\varepsilon\|_{L_2(0,1)} = \sqrt{\int_0^1 \varepsilon^2 dx} \tag{14}$$

For a given initial condition, a closed form expression for the norm may be obtained, depending on the function $u_0(x)$. Otherwise, for an arbitrary initial function, the integral may be evaluated numerically by a high–accuracy quadrature formula. In our tests, we used a corrected form of Simpson's rule. For this example,

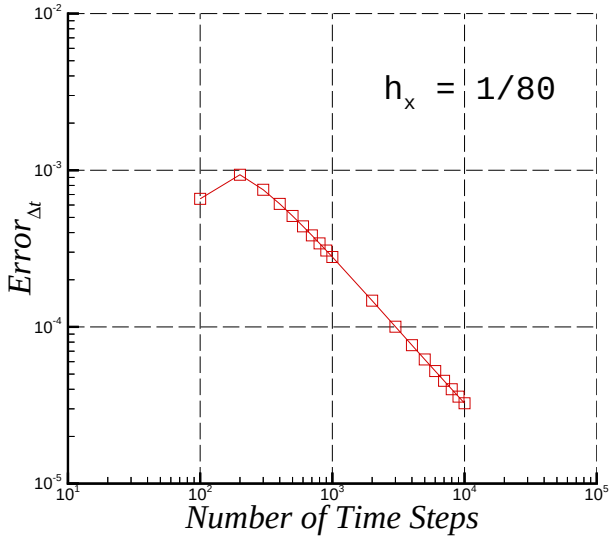

Figure 4 Numerical error with fixed grid, decreasing time-step size.

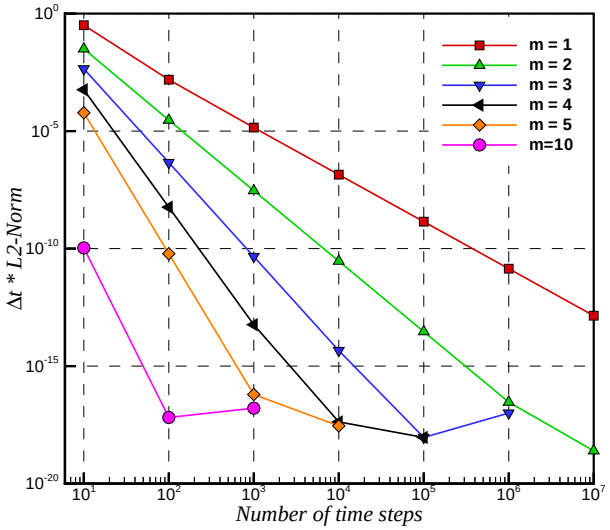

Figure 5 Convergence rates with decreasing step size for HSN solution of scalar advection equation.

the error norm gives

$$\sqrt{\tfrac{1}{2}[B_n^2 + (A_n - 1)^2]} \tag{15}$$

For reference, we present the results of the HSN method to solve the hyperbolic model equation with initial conditions as given above. These numerical values are given in Table 1 in the Appendix, obtained by using (3) with $m = 1$, where $N_{\Delta t}$ denotes number of time steps.

The trend of results in the tables shown indicate that with $\Delta t \to 0$ the following hold: $A \to 1$, $B \to 0$, $norm \to 0$. When $norm$ vs. number of time steps is presented in logarithmic scale, then the curve is a straight line. In standard finite-difference methods, one quantifies the error using the formula

$$\|\varepsilon_{\Delta x}\| = \Delta x \sqrt{\sum \varepsilon^2}. \tag{16}$$

On a log–log plot of (16) against the number of points, the order of the method would be given by the slope of a line. In Figure (3) the slope of the lines equals two, representing the second–order accuracy of the Lax–Wendroff method in space and time. Formula (3) with $m = 2, 3, 4, 5, 10$ was used to approximate the solution of the model equation and given initial conditions at time $t^n = 1$. Detailed results are shown in the tables in the Appendix and a plot of these calculations is shown in Figure (5). For comparison with the finite–difference error calculations, the error norm is multiplied by the time–step size and plotted on a log–log scale. The results, for example, show that for $m = 3$, which is the HSN formula with three terms in the series, is better than third–order accurate in time. The figure also demonstrates that to achieve more accurate results, more terms in the series can be retained without decreasing the step size. What became evident in obtaining the results shown in Figure (4) and Figure (5) is that the finite–difference method slowed down considerably as the time–step size was decreased for a given mesh size. This became even more dramatic as both the space and time meshes were refined. Note the difference in scale: The error for the finite–difference method did not reach machine zero, while the HSN method easily reached machine zero even at a million time steps. It will be interesting to calculate the operation counts and computation time as compared to the finite–difference method.

A few additional observations can be derived from Figure (5). The HSN scheme generates very accurate results for expressions (ExprLib) with double (C language) coefficients. It can be seen that for $m = 5$ the curve is a straight line between $nts = 10$ and $nts = 1000$. By extrapolation it could be expected that the $norm$ value at $nts = 10^5$ should be in the order of 1.0×10^{-18}. This number, however, goes beyond double (C language) representation of numbers, and the value obtained is about 1.0×10^{-14}. In such cases a multiple precision version of the library (MP-ExprLib) can be used to achieve higher accuracy.

Example 2: Step Function

Tracking and resolving discontinuities is of prime importance in CFD. An model of this capability for finite–difference methods is given by the unit step function. Here, we will study the case when $a = 1$ in equation (1) and the initial condition u_0 is given by the hyperbolic tangent function:

$$u_0(x) = \tfrac{1}{2}(1 - \tanh(10x)). \tag{17}$$

The exact solution is easily obtained by $u(x,t) = u_0(x-at)$,

$$u(x,t) = \tfrac{1}{2}(1 - \tanh(10(x-at))). \tag{18}$$

To solve equation (1) with initial condition (17) we will apply one–step method based on the Taylor series (3). We will call the method m–degree when it uses first $m+1$ terms in (3). To measure the accuracy of the HSN solution numerically, the L_2–norm is used:

$$\|\varepsilon\|_{L_2(a,b)} = \sqrt{\int_a^b \varepsilon(x)dx} \tag{19}$$

where the error is again defined as the difference at time $t = t_n$,

$$\varepsilon = u_{\text{exact}}(x,t^n) - u_{HSN}(x). \tag{20}$$

The integral is evaluated numerically with a high–accuracy quadrature formula. The limits of integration are determined from the location of the jump: For an advection speed of $a = 1$ and the initial condition places the $\tanh(x)$ function centered at the origin, the jump moves to the right. So $t^n \pm 0.5$ are the specified limits of integration that span the jump thickness, which is the region of interest.

For $n = 10$ time steps with time step $\Delta t = 0.01$, and approximation degree $m = 2$, the HSN scheme implemented in C utilizing *ExprLib* generates the following result:

$$\begin{aligned} u_{HSN}(x) = {} & -1.18794 \times 10^{-05} \tanh(10x)^{21} \\ & -0.0001188 \tanh(10x)^{20} - 0.0005421 \tanh(10x)^{19} \\ & -0.0015006 \tanh(10x)^{18} - 0.0028540 \tanh(10x)^{17} \\ & -0.0041111 \tanh(10x)^{16} - 0.0050825 \tanh(10x)^{15} \\ & -0.0062731 \tanh(10x)^{14} - 0.0083101 \tanh(10x)^{13} \\ & -0.0113005 \tanh(10x)^{12} - 0.0150604 \tanh(10x)^{11} \\ & -0.0196581 \tanh(10x)^{10} - 0.0254695 \tanh(10x)^{9} \\ & -0.0329481 \tanh(10x)^{8} - 0.0425778 \tanh(10x)^{7} \\ & -0.0549901 \tanh(10x)^{6} - 0.071078 \tanh(10x)^{5} \\ & -0.0921081 \tanh(10x)^{4} - 0.119923 \tanh(10x)^{3} \\ & -0.157332 \tanh(10x)^{2} - 0.209091 \tanh(10x) \\ & + 0.88034 \end{aligned} \tag{21}$$

with error norm (19) totalling 4.6086×10^{-04}. The exact and HSN solutions are visually indistinguishable in Figure (6).

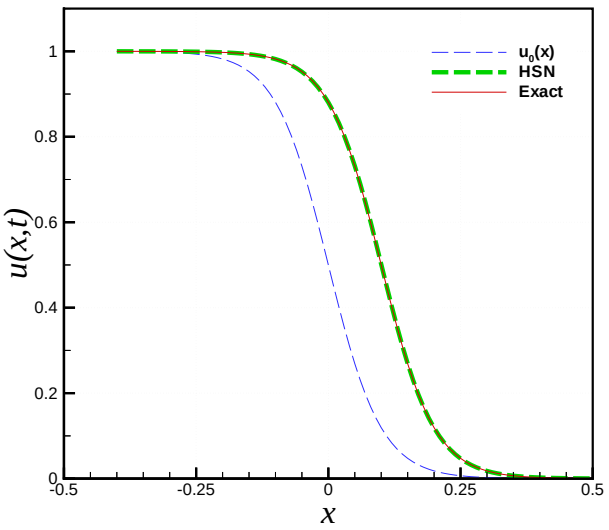

Figure 6 Exact and HSN solution for advection equation with step–function initial condition.

While the finite–difference solution to these initial conditions is straight–forward, it is still limited by the CFL condition. The interesting challenge that arises with the HSN method is the accumulation of terms evident in (21). This challenge is being met in several ways and requires further careful study since it becomes a significant factor in long–time calculations for non–linear equations and we will present results of this study in a subsequent publication.

Model Parabolic Equation

In this section we present solutions to the model parabolic equation of the general form

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{22}$$

where α is a positive constant representing a diffusion coefficient. We assume the solution domain is $[0,1] \times [0, t_{max}]$. Let the initial conditions be given by $u(x,0) = u_0(x)$, with specified boundary conditions $u(0,t) = u_1(t)$, $u(1,t) = u_2(t)$. The standard approach again requires a discrete approximation to both derivatives, typically a finite-difference formula in space followed by time–integration. Like the model hyperbolic equation, a time–step constraint is typically enforced to maintain numerical stability. Most methods that work with the hyperbolic equation work with the parabolic equation, with the caveat that a best method for the parabolic equation may not be optimal for the hyperbolic equation.

Example

We will study the case when $\alpha = 1$ in equation (22) and the initial condition is given by

$$u(x,0) = \sin(\pi x). \tag{23}$$

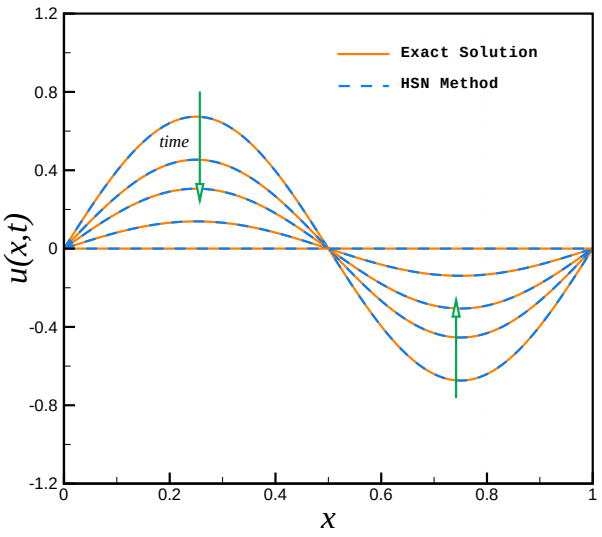

Figure 7 Exact and HSN solution to parabolic (diffusive) model equation.

The problem has exact solution of the form[27]

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x) \tag{24}$$

which is sine wave decaying in amplitude with time. To solve equation (22) with initial condition (23) we will apply one-step method based on the Taylor series (3). We will call the method m-degree when it uses first $m+1$ terms in (3). At any time $t^n = n\Delta t$ the approximate hybrid solution can be expressed in the form

$$u_{HSN}(x) = A_n \sin(\pi x) \tag{25}$$

where the coefficient A_n depends only on t^n. To measure the accuracy of the HSN solution numerically, we again use the L_2–norm (14) and evaluate the integral analytically:

$$\|u(x,t) - u_{HSN}(x)\|_{L_2(0,1)} = \frac{1}{\sqrt{2}}\left|A_n - e^{-\pi^2 t^n}\right|. \tag{26}$$

Taking the number of time steps $n = 10$, time step $\Delta t = 0.01$, and approximation degree $m = 2$, the one–step HSN scheme implemented in C utilizing *ExprLib* generates the following result:

$$u_{\text{HSN}} = 3.733515337236092 \times 10^{-01}\sin(\pi x). \tag{27}$$

The error norm (26) at the end of these 10 steps is approximately 4.5516×10^{-04}; this can be reduced very effectively and with little increased cost either by taking smaller time steps or increasing the approximation degree m. The physics modelled by the parabolic equation represent the process of diffusion. Figure (7) demonstrates that the HSN method, like a stable finite–difference technique, gradually smoothes out the given initial condition with time and matches the exact, time–decaying solution quite well.

Nonlinear Burgers Equation

In this section we have preliminary results for using the HSN methodology for non–linear equations, represented ubiquitously by Burgers equation. The non-dimensional form of the viscous Burgers equation can be written as

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2} \tag{28}$$

where **Re** is the Reynolds number ($\mathbf{Re} = u_r l_r/\nu$, u_r is a characteristic velocity, l_r is a characteristic length, ν is a viscous–like coefficient). The initial condition

$$u(x,0) = u_0(x), \tag{29}$$

where u_0 is a given initial function. An interesting HSN scheme for this equation can be interpreted as a variation of Heun's method for ordinary differential equations, as mentioned above. The approach was detailed by Derickson and Pielke.[28] We review it here and give some results.

Assume that an equally spaced partition $0 = t_0 < t_1 < \ldots$ of the time interval $[0,\infty)$ has been given. Integrating equation (28) over the discrete time interval $\Delta t = t_{n+1} - t_n$ yields

$$\int_{t_n}^{t_{n+1}} \frac{\partial u}{\partial t}dt = \int_{t_n}^{t_{n+1}} \left(-u\frac{\partial u}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}\right)dt \tag{30}$$

It is clear that left hand side in (30) can be written as

$$\int_{t_n}^{t_{n+1}} \frac{\partial u}{\partial t}dt = u^{n+1} - u^n \tag{31}$$

where $u^{n+1} = u^{n+1}(x) = u(x,t_{n+1})$, $u^n = u^n(x) = u(x,t_n)$. The integral on the right hand side of (30) can be approximated by using the trapezoidal rule

$$\int_{t_n}^{t_{n+1}} \left(-u\frac{\partial u}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}\right)dt \approx \tag{32}$$

$$\frac{1}{2}\left[\left(-u\frac{\partial u}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}\right)^n + \left(-u\frac{\partial u}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}\right)^{n+1}\right]$$

The first term of the right hand side in (32) is simply

$$\left(-u\frac{\partial u}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}\right)^n = -u^n\frac{\partial u^n}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u^n}{\partial x^2} \tag{33}$$

The second term of the right hand side in (32) is approximated by a two–step iterative method. Putting (31) into (30) and averaging the integrand in (30) at time $t = t_n$ yields

$$u^* - u^n \approx \Delta t\left(-u^n\frac{\partial u^n}{\partial x} + \frac{1}{\mathbf{Re}}\frac{\partial^2 u^n}{\partial x^2}\right) \tag{34}$$

The initial approximation u^* of u at time t_{n+1} is provided by (34). The formula for u^* in (34) can be used

to calculate the derivatives. In the second iterative step one can derive

$$\left(-u\frac{\partial u}{\partial x}+\frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}\right)^{n+1}=-u^*\frac{\partial u^*}{\partial x}+\frac{1}{\mathbf{Re}}\frac{\partial^2 u^*}{\partial x^2} \quad (35)$$

Putting all the above together, we can summarize the entire procedure as the two–step predictor–corrector method

$$u^* \;=\; u^n+\Delta t\,(U^n) \quad (36)$$

$$u^{n+1} \;=\; u^n+\frac{\Delta t}{2}\,(U^n+U^*) \quad (37)$$

where

$$U=-u\frac{\partial u}{\partial x}+\frac{1}{\mathbf{Re}}\frac{\partial^2 u}{\partial x^2}, \quad (38)$$

and u^*, etc., are calculated using (34). We will consider the initial value of this sequence as given by (29).

Example Calculation

Consider the initial condition, as used Derickson and Pielke[28]

$$u(x,0)=A\sin(\kappa x). \quad (39)$$

Using the HSN method as described above, we obtain the following analytic approximation of the Burgers equation after the first time step using *ExprLib* :

$$\begin{aligned}u_{\mathrm{HSN}}(x)=&\\ &A\left[1-A\kappa\Delta t\cos(\kappa x)\right]\sin(\kappa x)\\ &+\tfrac{1}{2}A^3\kappa^2\Delta t^2\left[A\kappa\Delta t\cos(\kappa x)-1\right]\sin^3(\kappa x)\\ &+A^3\kappa^2\Delta t^2\left[1-\tfrac{1}{2}A\kappa\Delta t\cos(\kappa x)\right]\cos^2(\kappa x)\sin(\kappa x)\\ &+\frac{\Delta t}{\mathbf{Re}}\Big\{A^3\kappa^4\Delta t^2\left[\tfrac{1}{2}\sin^3(\kappa x)-\cos^2(\kappa x)\sin(\kappa x)\right]\\ &+3A^2\kappa^3\Delta t\cos(\kappa x)\sin(\kappa x)-A\kappa^2\sin(\kappa x)\Big\}. \quad (40)\end{aligned}$$

Subsequent time steps increase the number of terms exponentially. However, note that the last term is proportional to the step–size divided by the Reynolds number. For high Reynolds number, the term will be very small and subject to elimination without loss of accuracy. This process was performed by hand in Derickson and Pielke,[28] a slow and tedious procedure. Fortunately, with *ExprLib* , this is automated and in fact a pruning algorithm is available to facilitate writing routines to do this. Details of the method will be presented in a subsequent publication. For the purposes of this paper, we have introduced one approach to utilizing the power of symbolic computation for developing approximate analytic solutions to the equations of interest.

Conclusions

We have presented a new approach to the solution of typical problems in CFD. While several existing commercial software packages for symbolic computation contain many of the desirable features for developing a hybrid methodology, we have chosen to carry out this work using an ANSI C library so that we can take advantage of the efficiency of compiled code as well as a standardized and portable environment. Since *ExprLib* is written in ordinary C, we are able to take advantage of inter-operability with compiled legacy code such as libraries written in FORTRAN. Through the use of these methods, it is clear that "industrial strength" hybrid symbolic–numeric software solutions to real life problems are attainable.

The future of HSN methods seems bright, offering the possibility of developing new and interesting methods that were not feasible or evident before. We foresee the continued development of this methodology.

Acknowledgments

This work was sponsored by a grant from the Chief Scientist Innovative Research Fund of the Air Vehicles Directorate, U. S. Air Force Research Laboratory. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

References

[1]Backus, J.W., e. a., "The FORTRAN Automatic Coding System," Proceedings of the Western Joint Computing Conference 1957, pp.188-198.

[2]Backus, J., "The Syntax and Semantics of the Proposed International Algebraic Language of Zürich ACM-GAMM Conference," Proceedings of the International Conference on Information Processing, UNESCO, 1959, pp.125-132.

[3]Hall, A., "The ALTRAN System for Rational Function Manipulation - A Survey," *Communications of the ACM*, Vol. 14, No. 8, 1971, pp. 517–521.

[4]Hall, Jr., A. D., "Solving a problem in eigenvalue approximation with a symbolic algebra system," *SIAM J. Comput.*, Vol. 4, 1975, pp. 163–174.

[5]Rink, R. A. and Guru, B. P., "Analytical solutions for a class of nonlinear differential equations using ALTRAN," *Nordisk Tidskr. Informationsbehandling (BIT)*, Vol. 16, No. 2, 1976, pp. 161–171.

[6]Rink, R. A. and Guru, B. P., "Automating analytical solutions of nonlinear differential equations," *Computational methods in nonlinear mechanics (Proc. Internat. Conf., Austin, Tex., 1974)*, Texas Inst. Comput. Mech., Austin, Tex., 1974, pp. 57–66.

[7]Sammet, J. E., "An annotated descriptor-based bibliography on the use of computers for non-numerical mathematics," *Comput. Rev.*, Vol. 7, 1966, pp. B1–B31.

[8]Winston, P. H., *Artificial Intelligence*, Addison Wesley, Reading, MA, 1992.

[9]Apostolico, A. and Galil, Z., editors, *Pattern matching algorithms*, Oxford University Press, 1997.

[10]Knuth, D. E. and Bendix, P. B., "Simple word problems in universal algebras," *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, Pergamon, Oxford, 1970, pp. 263–297.

[11]Marché, C., "Normalized rewriting: a unified view of Knuth-Bendix completion and Gröbner bases computation," *Symbolic rewriting techniques (Ascona, 1995)*, Birkhäuser, Basel, 1998, pp. 193–208.

[12]Richardson, D., "Some unsolvable problems involving elementary functions of a real variable," *J. Symbolic Logic*, Vol. 33, 1968, pp. 514–520.

[13]Engelman, C., "The legacy of MATHLAB 68," *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, 1971, pp. 29–41.

[14]Hearn, A. C., Marti, J. D., et al., "REDUCE user's manual, version 3.0 (standard LISP report)," Rand publication cp78 (4/83) (uucs-78-101) (university of utah, technical report tr-6, 1978), Rand Corporation, Santa Monica, CA, USA, 1983.

[15]Pavelle, R., "Macsyma: capabilities and applications to problems in engineering and the sciences," *Applications of computer algebra (Philadelphia, Pa., 1984)*, Kluwer·Nijhoff, Boston, MA, 1985, pp. 1–61.

[16]Heck, A., *Introduction to Maple*, Springer-Verlag, New York, 1993.

[17]Wolfram, S., *The Mathematica book*, Wolfram Media, Inc., Champaign, IL, 4th ed., 1999.

[18]Jenks, R. D. and Sutor, R. S., *AXIOM*, Numerical Algorithms Group Ltd., Oxford, 1992, The scientific computation system, With a foreword by David V. Chudnovsky and Gregory V. Chudnovsky.

[19]Lambe, L. and Luczak, R., "Object-oriented mathematical programming and symbolic/numeric interface," *Math. Comput. Simulation*, Vol. 36, No. 4-6, 1994, pp. 493–503, Third International Conference on Expert Systems for Scientific Computing (West Lafayette, IN, 1993).

[20]Lambe, L. A., "Overview of ExprLib: An ANSI C library for Hybrid Numeric & Symbolic Computation," August 2000, Multidisciplinary Applications and Interoperable Computing 2000, ASC/MSRC, Wright–Patterson AFB, Dayton, OH, Proceedings on CD.

[21]Lambe, L., Luczak, R., and Nehrbass, J., "A New Finite Difference Method for the Helmholtz Equation Using Symbolic Computation," *International Journal of Computational Engineering Science*, Vol. 4, No. 2, 2003.

[22]Poondru, S. and Abdallah, S., "Direct Matrix Inversion in Symbolic form for Solutions of the Transport Equation," AIAA Paper 2003–0251, *January*2003, Reno, NV.

[23]Verhoff, A., "A Compact Formulation for Analytical Solution of the 2D Euler Equations," AIAA Paper 2003–3918, *June*2003, Orlando, FL.

[24]Tannehill, J. C., Anderson, D. A., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer, 2nd ed.*, Taylor & Francis, Inc., Philadelphia, PA, 1997.

[25]Lax, P. D. and Wendroff, B., "Systems of Conservation Laws," *Commun. Pure Appl. Math.*, Vol. 13, 1960, pp. 217–2237.

[26]MacCormack, R. W., "The Effect of Viscosity in Hypervelocity Impact Cratering," AIAA Paper 69–0354, 1969, Cincinnati, OH.

[27]Carrier, G. and Pearson, C. E., *Partial Differential Equations: Theory and Technique, 2nd ed.*, Academic Press, Inc., Boston, MA, 1988.

[28]Derickson, R. G. and Pielke, Sr., R. A., "A preliminary study of the Burgers equation with symbolic computation," *Journal of Computational Physics*, Vol. 162, 2000, pp. 219–244.

Appendix

Table 1 (m = 1)

Δt	$N_{\Delta t}$	A	B	$norm$
$1.0e-01$	10^1	$4.1265851168016621e+00$	$3.2919607341010519e+00$	$3.2103379859034158e+00$
$1.0e-02$	10^2	$1.2177068419842305e+00$	$1.0044860504616157e-02$	$1.5410575633198118e-01$
$1.0e-03$	10^3	$1.0199349143076459e+00$	$8.4329693742445609e-05$	$1.4096239213890230e-02$
$1.0e-04$	10^4	$1.0019758699537735e+00$	$8.2846756418683033e-07$	$1.3971511658699446e-03$
$1.0e-05$	10^5	$1.0001974115707357e+00$	$8.2699743094458242e-09$	$1.3959106047440058e-04$
$1.0e-06$	10^6	$1.0000197394036081e+00$	$8.2712885081854843e-11$	$1.3957866147995319e-05$
$1.0e-07$	10^7	$1.0000019739227146e+00$	$8.8386586653978167e-13$	$1.3957741370428359e-06$

Table 2 (m = 2)

Δt	$N_{\Delta t}$	A	B	$norm$
1.0e-01	10^1	1.1335321492508006e+00	-4.2504639359912272e-01	3.1503592778852896e-01
1.0e-02	10^2	1.0001863097087527e+00	-4.1300598124052118e-03	2.9233632481800885e-03
1.0e-03	10^3	1.0000001939636531e+00	-4.1341220645073971e-05	2.9232979204345813e-05
1.0e-04	10^4	1.0000000001947400e+00	-4.1341697662250754e-07	2.9232998005966072e-07
1.0e-05	10^5	1.0000000000001885e+00	-4.1341670495692791e-09	2.9232975583476438e-09
1.0e-06	10^6	9.9999999999994460e-01	-4.1326888724138408e-11	2.9222549519092303e-11
1.0e-07	10^7	9.9999999999652600e-01	-5.2378699888375371e-13	2.4842524975968665e-12

Table 3 ($m = 3$)

Δt	$N_{\Delta t}$	A	B	$norm$
1.0e-01	10^1	9.4440107148153329e-01	-2.9577414238667794e-02	4.4531249058651606e-02
1.0e-02	10^2	9.9993514811838524e-01	-3.2624666143500384e-06	4.5915194856301901e-05
1.0e-03	10^3	9.9999993506146145e-01	-3.2641793736900521e-10	4.5919061059787826e-08
1.0e-04	10^4	9.9999999993506528e-01	-2.4160546248247696e-14	4.5915787040023443e-11
1.0e-05	10^5	9.9999999999986999e-01	-3.7056458956685018e-15	9.1966249433913722e-14
1.0e-06	10^6	9.9999999999980338e-01	-1.4183590001111901e-11	1.0030276290804307e-11

Table 4 ($m = 4$)

Δt	$N_{\Delta t}$	A	B	$norm$
1.0e-01	10^1	9.9591991621433018e-01	7.0133088801552336e-03	5.7373157986357181e-03
1.0e-02	10^2	9.9999995729234692e-01	8.1490216532127223e-07	5.7701364051335795e-07
1.0e-03	10^3	9.9999999999957490e-01	8.1605892274254275e-11	5.7704862739741451e-11
1.0e-04	10^4	9.9999999999995193e-01	3.5446555534713523e-14	4.2234101429248665e-14
1.0e-05	10^5	9.9999999999986999e-01	-3.7056458548658768e-15	9.1966249433091678e-14

Table 5 (m = 5)

Δt	$N_{\Delta t}$	A	B	$norm$
1.0e-01	10^1	1.0007315025674723e+00	4.3734741635506136e-04	6.0264781125093101e-04
1.0e-02	10^2	1.0000000085330334e+00	4.5999962485016528e-10	6.0425267275620249e-09
1.0e-03	10^3	1.0000000000000875e+00	2.8481506180214369e-15	6.1894416871085785e-14
1.0e-04	10^4	9.9999999999997902e-01	3.4397533026655269e-14	2.8491100338508877e-14

Table 6 ($m = 10$)

Δt	$N_{\Delta t}$	A	B	$norm$
1.0e-01	10^1	1.0000000008211964e+00	-1.2644522898953953e-09	1.0661151892157392e-09
1.0e-02	10^2	1.0000000000000009e+00	2.5060732568867393e-16	6.5255840233263447e-16
1.0e-03	10^3	9.9999999999997813e-01	-6.5933467511678016e-15	1.6152864741837129e-14