Joseph L. Awange · Béla Paláncz · Robert H. Lewis · Lajos Völgyesi

Mathematical Geosciences

Hybrid Symbolic-Numeric Methods

This book showcases powerful new hybrid methods that combine numerical and symbolic algorithms. Hybrid algorithm research is currently one of the most promising directions in the context of geosciences mathematics and computer mathematics in general.

One important topic addressed here with a broad range of applications is the solution of multivariate polynomial systems by means of resultants and Groebner bases. But that's barely the beginning, as the authors proceed to discuss genetic algorithms, integer programming, symbolic regression, parallel computing, and many other topics.

The book is strictly goal-oriented, focusing on the solution of fundamental problems in the geosciences, such as positioning and point cloud problems. As such, at no point does it discuss purely theoretical mathematics.

The book delivers hybrid symbolic-numeric solutions, which are a large and growing area at the boundary of mathematics and computer science. **Dr. Daniel Lichtbau**

Earth Sciences

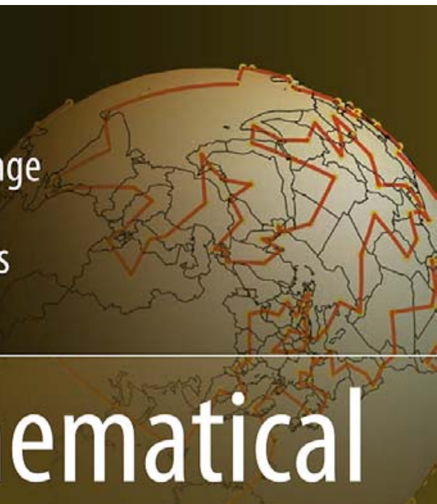
ISBN 978-3-319-67370-7

springer.com

Awange · Paláncz · Lewis · Völgyesi

Mathematical Geosciences

Joseph L. Awange
Béla Paláncz
Robert H. Lewis
Lajos Völgyesi



Mathematical Geosciences

Hybrid Symbolic-Numeric Methods

Springer

Mathematical Geosciences

Joseph L. Awange · Béla Paláncz
Robert H. Lewis · Lajos Völgyesi

Mathematical Geosciences

Hybrid Symbolic-Numeric Methods

Springer

Joseph L. Awange
Spatial Sciences
Curtin University
Perth, WA
Australia

Béla Paláncz
Budapest University of Technology and Economics
Budapest
Hungary

Robert H. Lewis
Fordham University
New York, NY
USA

Lajos Völgyesi
Budapest University of Technology and Economics
Budapest
Hungary

ISBN 978-3-319-67370-7 ISBN 978-3-319-67371-4 (eBook)
https://doi.org/10.1007/978-3-319-67371-4

Library of Congress Control Number: 2017953801

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

Hybrid symbolic-numeric computation (HSNC, for short) is a large and growing area at the boundary of mathematics and computer science, devoted to the study and implementation of methods that mix symbolic with numeric computation.

As the title suggests, this is a book about some of the methods and algorithms that benefit from a mix of symbolic and numeric computation. Three major areas of computation are covered herein. The first part discusses methods for computing all solutions to a system of polynomials. Purely symbolic methods, e.g., via Gröbner bases tend to suffer from algorithmic inefficiencies, and purely numeric methods such as Newton iterations have trouble finding all solutions to such systems. One class of hybrid methods blends numerics into the purely algebraic approach, e.g., computing numeric Gröbner bases or Dixon resultants (the latter being extremely efficient, e.g., for elimination of variables). Another mixes symbolic methods into more numerical approaches, e.g., finding initializations for numeric homotopy tracking to obtain all solutions.

The second part goes into the realm of "soft" optimization methods, including genetic methods, simulated annealing, and particle swarm optimization, among others. These are all popular and heavily used, especially in the context of global optimization. While often considered as "numeric" methods, they benefit from symbolic computation in several ways. One is that implementation is typically straightforward when one has access to a language that supports symbolic computation. Updates of state, e.g., to handle mutations and gene crossover, are easily coded. (Indeed, this sort of thing can be so deceptively simple. baked into the language so to speak, that one hardly realizes symbolic computation is happening.) Among many applications in this part there is, again, that of solving systems of equations. Also covered is mixed-integer programming (wherein some variables are discrete-valued and others continuous). This is a natural area for HSNC since it combines aspects of exact and numeric methods in the handling of both discrete and continuous variables.

The third part delves into data modeling. This begins with use of radial basis functions and proceeds to machine learning, e.g., via support vector machine (SVM) methods. Symbolic regression, a methodology that combines numerics with evolutionary programming, is also introduced for the purpose of modeling data. Another area seeing recent interest is that of robust optimization and regression, wherein one seeks results that remain relatively stable with respect to perturbations in input or random parameters used in the optimization. Several hybrid methods are presented to address problems in this realm. Stochastic modeling is also discussed. This is yet another area in which hybrid methods are quite useful.

Symbolic computing languages have seen a recent trend toward ever more high level support for various mathematical abstractions. This appears for example in exact symbolic programming involving probability, geometry, tensors, engineering simulation, and many other areas. Under the hood is a considerable amount of HSNC (I write this as one who has been immersed at the R&D end of hybrid computation for two decades.) Naturally, such support makes it all the easier for one to extend hybrid methods; just consider how much less must be built from scratch to support, say, stochastic equation solving, when the language already supports symbolic probability and statistics computations. This book presents to the reader some of the major areas and methods that are being changed, by the authors and others, in furthering this interplay of symbolic and numeric computation. The term hybrid symbolic-numeric computation has been with us for over two decades now. I anticipate the day when it falls into disuse, not because the technology goes out of style, but rather that it is just an integral part of the plumbing of mathematical computation.

Urbana—Champaign
IL, USA
July 2017

Daniel Lichtblau
Ph.D., Mathematics UIUC 1991
Algebra, Applied Mathematics
Wolfram Research, Champaign

Preface

It will surprise no one to hear that digital computers have been used for numerical computations ever since their invention during World War II. Indeed, until around 1990, it was not widely understood that computers could do anything else. For many years, when students of mathematics, engineering, and the sciences used a computer, they wrote a program (typically in Fortran) to implement mathematical algorithms for solving equations in one variable, or systems of linear equations, or differential equations. The input was in so-called "float" numbers with 8–12 significant figures of accuracy. The output was the same type of data, and the program worked entirely with the same type of data. This is *numerical computing*.

By roughly 1990, computer algebra software had become available. Now it was possible to enter data like $x^2 + 3x + 2$ and receive output like $(x + 2)(x + 1)$. The computer is doing algebra! More precisely, it is doing *symbolic computing*. The program that accomplishes such computing almost certainly uses no float numbers at all.

What is still not widely understood is that often it is productive to have algorithms that do both kinds of computation. We call these *hybrid symbolic-numeric* methods. Actually, such methods have been considered by some mathematicians and computer scientists since at least 1995 (ISSAC 1995 conference). In this book, the authors provide a much-needed introduction and reference for applied mathematicians, geoscientists, and other users of sophisticated mathematical software.

No mathematics beyond the undergraduate level is needed to read this book, nor does the reader need any pure mathematics background beyond a first course in linear algebra. All methods discussed here are illustrated with copious examples.

A brief list of topics covered:

- Systems of polynomial equations with resultants and Gröbner bases
- Simulated annealing
- Genetic algorithms
- Particle swarm optimization
- Integer programming
- Approximation with radial basis functions

- Support vector machines
- Symbolic regression
- Quantile regression
- Robust regression
- Stochastic modeling
- Parallel computations

Most of the methods discussed in the book will probably be implemented by the reader on a computer algebra system (CAS). The two most fully developed and widely used CAS are *Mathematica* and *Maple*. Some of the polynomial computations here are better done on the specialized system *Fermat*. Other systems worthy of mention are *Singular* and *SageMath*.

The second author is a regular user of *Mathematica*, who carried out the computations, therefore frequent mention is made of *Mathematica* commands. However, this book is not a reference manual for any system, and we have made an effort to keep the specialized commands to a minimum, and to use commands whose syntax makes them as self-explanatory as possible. More complete *Mathematica* programs to implement some of the examples are available online. Similarly, a program written in *Fermat* for the resultant method called Dixon-EDF is available online.

The authors:

July 2017

Joseph L. Awange
Perth, Australia

Béla Paláncz
Budapest, Hungary

Robert H. Lewis
New York, USA

Lajos Völgyesi
Budapest, Hungary

Acknowledgements

The authors wish to express their sincere thanks to *Dr. Daniel Lichtblau* for his helpful comments and for agreeing to write a foreword for our book. *R. Lewis* and *B. Paláncz* highly appreciate and thank *Prof. Jon Kirby* the Head of the Department of Spatial Sciences, Curtin University, Australia for his hospitality and his support of their visiting Curtin. *B. Paláncz* wishes also thank the TIGeR Foundation for the partial support of his staying in Perth. This work was funded partially by OTKA project No. 124286.

Contents

Introduction

Numeric and Symbolic Methods—What are they?

Basically, a *numeric* (*or numerical*) *method* is one that could be done with a simple handheld calculator, using basic arithmetic, square roots, some trigonometry functions, and a few other functions most people learn about in high school. Depending on the task, one may have to press the calculator buttons thousands (or even millions) of times, but theoretically, a person with a calculator and some paper could implement a numerical method. When finished, the paper would be full of arithmetic.

A *symbolic method* involves algebra. It is a method that if a person implemented, would involve algebraic or higher rational thought. A person implementing a symbolic method will rarely need to reach for a calculator. When finished, there may be some numbers, but the paper would be full of variables like x, y, z.

Students usually meet the topic of quadratic equations in junior high school. Suppose you wanted to solve the equation $x^2 + 3x - 2 = 0$. With a handheld calculator, one could simply do "intelligent guessing." Let us guess, say, $x = 1$. Plug it in, get a positive result. OK, that is too big. Try $x = 0$; that is too small. Go back and forth; stop when satisfied with the accuracy. It does not take long to get $x = 0.56155$, which might well be considered accurate enough. Furthermore, it is easy to write a computer program to implement this idea. That is a numeric method.

But wait. There is another answer, which the numeric method missed, namely -3.56155. Even worse, if one were to continue this method on many problems, one would soon notice that some equations do not seem to have solutions, such as $x^2 - 2x + 4 = 0$. A great deal of effort could be expended in arithmetic until finally giving up and finding no solution.

The problem is cured by learning algebra and the symbolic method called the quadratic formula. Given $ax^2 + bx + c = 0$ the solution is $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. It is now immediately obvious why some problems have no solution: it happens precisely when $b^2 - 4ac < 0$.

In the previous example, $x^2 + 3x - 2 = 0$, we see that the two roots are exactly $(-3 \pm \sqrt{17})/2$. There is no approximation whatever. Should a decimal answer correct to, say, 16 digits be desired, that would be trivially obtained on any modern computer.

There is more. Not only does the symbolic method concisely represent all solutions, it invites the question, can we define a new kind of number in which the negative under the square root may be allowed? The symbolic solution has led to a new concept, that of complex numbers!

Symbolic methods may be hard to develop, and they may be difficult for a computer to implement, but they lead to insight.

Fortunately, we are not forced into a strict either/or dichotomy. There are symbolic-numeric methods, hybrids using the strengths of both ideas.

Numeric Solution

In order to further illustrate numeric, symbolic, and symbolic-numeric solutions, let us consider an algebraic system of polynomial equations. For such systems, there may be no solution, one solution, or many solutions. With numerical solutions, one commonly utilizes iterative techniques starting from an initially guessed value. Let us start with a two variable system of two equations $f(x, y) = 0$ and $g(x, y) = 0$,

$$f = (x-2)^2 + (y-3)^2,$$
$$g = \left(x - \tfrac{1}{2}\right)^2 + \left(y - \tfrac{3}{4}\right)^2 - 5.$$

This actual problem has two real solutions, see Fig. 1.

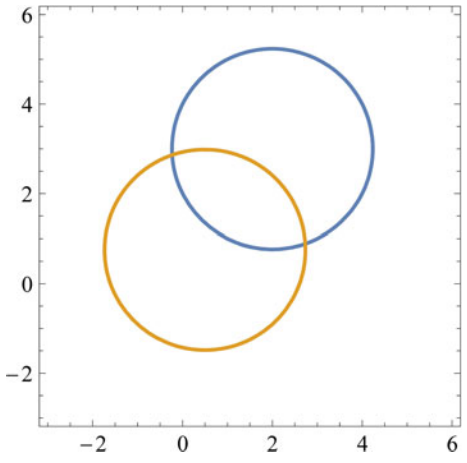


Fig. 1 Geometrical representation of a multivariate polynomial system

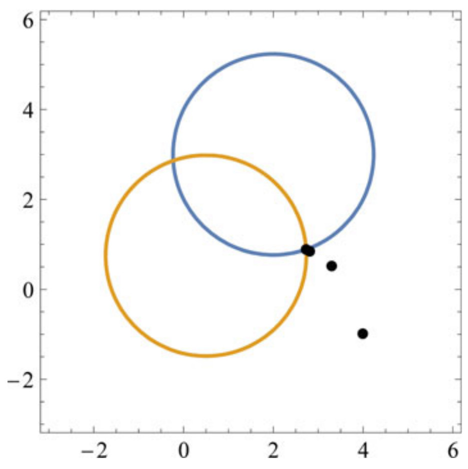


Fig. 2 Local solution with initial guess and iteration steps

A numeric solution starts with the initial value and proceeds step-by-step locally. Depending on the method, we expect to converge to one of the solutions in an efficient manner. Employing the initial value (4, −1) and a multivariate Newton's method, the solution after seven steps is (2.73186, 0.887092). Let us visualize the iteration steps, see Fig. 2.

However, if the initial guess is not proper, for example (0, 0), then, we may have a problem with the convergence since the Jacobian may become singular.

Symbolic Solution

Let us transform the original system into another one, which has the same solutions, but for which variables can be isolated and solved one-by-one. Employing Gröbner basis, we can reduce one of the equations to a univariate polynomial,

$$gry = 2113 - 3120y + 832y^2,$$

$$grxy = -65 + 16x + 24y.$$

First, solving the quadratic equation gry, we have

$$y = \tfrac{1}{104}\left(195 - 2\sqrt{2639}\right),$$

$$y = \tfrac{1}{104}\left(195 + 2\sqrt{2639}\right).$$

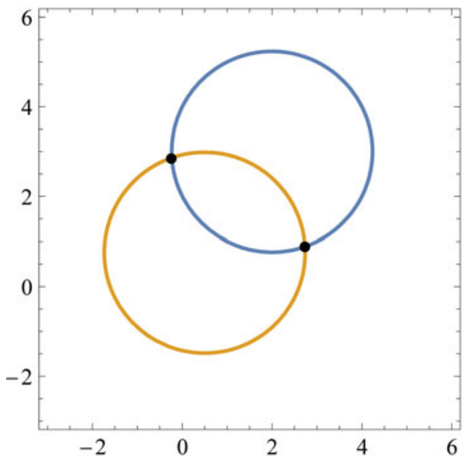


Fig. 3 Global solution—all solutions without initial guess and iteration

Then employing these roots of y, the corresponding values of x can be computed from the second polynomial of the Gröbner basis as

$$x = \tfrac{1}{104}\left(130 + 3\sqrt{2639}\right),$$

$$x = \tfrac{1}{104}\left(130 - 3\sqrt{2639}\right).$$

So, we have computed both solutions with neither guessing nor iteration. In addition, there is no round-off error. Let us visualize the two solutions, see Fig. 3:

Let us summarize the main features of the symbolic and numeric computations:

Numeric computations:

- usually require initial values and iterations. They are sensitive to round-off errors, provide only one local solution,
- can be employed for complex problems, and the computation times are short in general because the steps usually translate directly into computer machine language.

Symbolic computations:

- do not require initial values and iterations. They are not sensitive for round-off errors, and provide all solutions,
- often cannot be employed for complex problems, and the computation time is long in general because the steps usually require computer algebra system software.

Ideally, the best strategy is to divide the algorithm into symbolic and numeric parts in order to utilize the advantages of both techniques. Inevitably, numeric computations will always be used to a certain extent. For example, if polynomial *gry* above had been degree, say, five, then a numeric univariate root solver would have been necessary.

Hybrid (symbolic-numeric) Solution

Sometimes, we can precompute a part of a numerical algorithm in symbolic form. Here is a simple illustrative example.

Consider a third polynomial and add it to our system above:

$$h = \left(x + \tfrac{1}{2}\right)^2 + \left(y - \tfrac{7}{4}\right)^3 - 5.$$

In that case, there is no solution, since there is no common point of the three curves representing the three equations, see Fig. 4.

However, we can look for a solution of this overdetermined system in the minimal least squares sense by using the objective function

$$G = f^2 + g^2 + h^2,$$

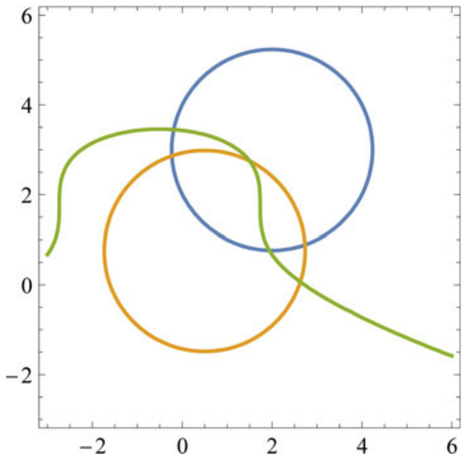


Fig. 4 Now, there is no solution of the overdetermined system

or

$$G = \left(-5+(-2+x)^2+(-3+y)^2\right)^2+\left(-5+\left(\frac{1}{2}+x\right)^2+\left(-\frac{7}{4}+y\right)^3\right)^2$$
$$+\left(-5+\left(-\frac{1}{2}+x\right)^2+\left(-\frac{3}{4}+y\right)^2\right)^2$$

and minimizing it.

Employing Newton's method, we get

$x = 2.28181, y = 0.556578.$

The computation time for this was 0.00181778 s. The solution of the overdetermined system can be seen in Fig. 5.

Here, the gradient vector as well as the Hessian matrix is computed in numerical form in every iteration step. But we can compute the gradient in symbolic form:

$$\text{grad} = \left(\frac{1}{32}\left(2x(173+192(-2+x)x)+216xy-16(41+26x)y^2+\right.\right.$$
$$\left.64(1+2x)y^3+3(-809+740y)\right)-\frac{137829}{512}+\frac{555x}{8}+\frac{27x^2}{8}+\frac{60527y}{128}-$$
$$\left.41xy-13x^2y-\frac{6321y^2}{16}+6xy^2+6x^2y^2+\frac{767y^3}{4}-\frac{105y^4}{2}+6y^5\right).$$

Employing this symbolic form the computation time can be reduced. The running time can be further reduced if the Hessian matrix is also computed symbolically,

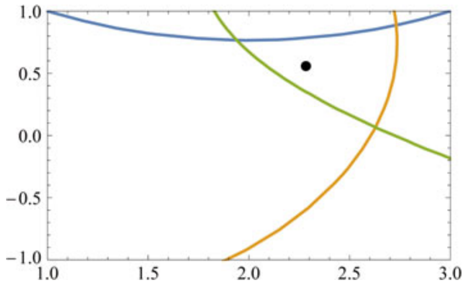


Fig. 5 The solution of the overdetermined system

$$H = \begin{bmatrix} \frac{173}{16} + 12x(-4+3x) + \frac{27y}{4} - 13y^2 + 4y^3 & \frac{555}{8} + y(-41+6y) + x\left(\frac{27}{4} + 2y(-13+6y)\right) \\ \frac{555}{8} + y(-41+6y) + x\left(\frac{27}{4} + 2y(-13+6y)\right) & \begin{array}{c} \frac{60527}{128} - 41x - 13x^2 - \frac{6321y}{8} \\ + 12xy + 12x^2y + \frac{2301y^2}{4} - 210y^3 + 30y^4 \end{array} \end{bmatrix}.$$

Now, the computation time is less than half of the original one.

So using symbolic forms, the *computation time can be reduced* considerably. This so-called *hybrid computation* has an additional advantage too, namely the symbolic part of the algorithm does *not generate round-off errors.*

Another approach of applying the hybrid computation is to *merge* symbolic evaluation with numeric algorithm. This technique is illustrated using the following example.

Let us consider a linear, nonautonomous differential equation system of n variables in matrix form:

$$\frac{d}{dx}y(x) = A(x)y(x) + b(x),$$

where A is a matrix of $n \times n$ dimensions, $y(x)$ and $b(x)$ are vectors of n dimensions, and x is a scalar independent variable. In the case of a boundary value problem, the values of some dependent variables are not known at the beginning of the integration interval, at $x = x_a$, but they are given at the end of this interval, at $x = x_b$. The usually employed methods need subsequent integration of the system, because of their trial–error technique or they require solution of a large linear equation system, in the case of discretization methods. The technique is based on the symbolic evaluation of the well-known Runge–Kutta algorithm. This technique needs only one integration of the differential equation system and a solution of the linear equation system representing the boundary conditions at $x = x_b$.

The well-known fourth-order Runge–Kutta method, in our case, can be represented by the following formulas:

$$R1_i = \mathrm{A}(x_i)y(x_i) + b(x_i),$$

$$R2_i = \mathrm{A}\left(x_i + \tfrac{h}{2}\right)\left(y(x_i) + \tfrac{R1_i h}{2}\right) + b\left(x_i + \tfrac{h}{2}\right),$$

$$R3_i = \mathrm{A}\left(x_i + \tfrac{h}{2}\right)\left(y(x_i) + \tfrac{R2_i h}{2}\right) + b\left(x_i + \tfrac{h}{2}\right),$$

$$R4_i = \mathrm{A}(x_i + h)(y(x_i) + R3_i h) + b(x_i + h)$$

and then the new value of $y(x)$ can be computed as:

$$y_{i+1} = y(x_i) + \frac{(R1_i + 2(R2_i + R3_i) + R4_i)h}{6}.$$

A symbolic system like *Mathematica*, is able to carry out this algorithm not only with numbers but also with symbols. It means that the unknown elements of $y_a = y(x_a)$ can be considered as unknown symbols. These symbols will appear in every evaluated y_i value, as well as in $y_b = y(x_b)$ too.

Let us consider a simple illustrative example. The differential equation is: $\left(\frac{d^2}{dx^2}y(x)\right) - \left(1 - \frac{x}{5}\right)y(x) = x.$

The given boundary values are:

$$y(1) = 2$$

and

$$y(3) = -1$$

After introducing new variables, we get a first-order system,

$$y1(x) = y(x)$$

and

$$y2(x) = \frac{d}{dx}y(x)$$

the matrix form of the differential equation is:

$$\left[\frac{d}{dx}y1(x),\ \frac{d}{dx}y2(x)\right] = \begin{bmatrix} 0 & 1 \\ 1 - x/5 & 0 \end{bmatrix}[y1(x),\ y2(x)] + [0,\ x].$$

Employing *Mathematica*'s notation:

```
A[x_]:={{0,1},{1-1/5 x,0}};
b[x_]:={0,x};
x0=1;
y0={2.,s}
```

The unknown initial value is s. The order of the system $M = 2$. Let us consider the number of the integration steps as $N = 10$, so the step size is $h = 0.2$.

```
ysol=RKSymbolic[x0,y0,A,b,2,10,0.2];
```

The result is a list of list data structure containing the corresponding (x, y) pairs, where the y values depend on s.

```
ysol[[2]][[1]]
{{1,2.},{1.2,2.05533+0.200987 s},{1.4,2.22611+0.407722
s},
{1.6,2.52165+0.625515 s},
{1.8,2.95394+0.859296s}, {2.,3.53729+1.11368s},
{2.2,4.28801+1.39298 s},
{2.4,5.22402+1.70123 s},{2.6,6.36438+2.0421 s},
{2.8,7.72874+2.41888 s},{3.,9.33669+2.8343 s}}
```

Consequently, we have got a symbolic result using traditional numerical Runge–Kutta algorithm.

In order to compute the proper value of the unknown initial value, *s*, the boundary condition can be applied at $x = 3$. In our case $y1(3) = -1$.

```
eq=ysol[[1]][[1]]==-1
9.33669+2.8343 s==-1
```

Let us solve this equation numerically, and assign the solution to the symbol *s*:

```
sol=Solve[eq,s]
{{s -> -3.647}}
s=s/.sol
{-3.647}
s=s[[1]]
-3.647
```

Then, we get the numerical solution for the problem:

```
ysol[[2]][[1]]
{{1,2.},{1.2,1.32234},{1.4,0.739147},{1.6,0.240397},
{1.8,-0.179911}, {2.,-0.524285},{2.2,-0.792178},
{2.4,-0.980351},{2.6,-1.08317},{2.8,-1.09291},
{3.,-1.}}
```

The truncation error can be decreased by using smaller step size *h*, and the round-off error can be controlled by the employed number of digits.