On the numerical solution of chaotic dynamical systems using extend precision floating point arithmetic and very high order numerical methods

Scott A. Sarra, Clyde Meador

Marshall University
One John Marshall Drive, Huntington, WV 25755, USA
sarra@marshall.edu

Received: 11 August 2010 / **Revised:** 28 Mart 2011 / **Published online:** 19 September 2011

Abstract. Multiple results in the literature exist that indicate that all computed solutions to chaotic dynamical systems are time-step dependent. That is, solutions with small but different time steps will decouple from each other after a certain (small) finite amount of simulation time. When using double precision floating point arithmetic time step independent solutions have been impossible to compute, no matter how accurate the numerical method. Taking the well-known Lorenz equations as an example, we examine the numerical solution of chaotic dynamical systems using very high order methods as well as extended precision floating point number systems. Time step independent solutions are obtained over a finite period of time. However even with a sixteenth order numerical method and with quad-double floating point numbers, there is a limit to this approach.

Keywords: chaos, ODEs, numerical methods, extended floating point precision, Lorenz system, implicit Gauss Runge–Kutta methods.

1 Introduction

The numerical solution of nonlinear systems of differential equations that exhibit chaotic behavior, such as the well-known Lorenz equations [1]

$$\begin{aligned} x' &= \sigma(y-x), & x(0) &= x_0, \\ y' &= rx - y - xz, & y(0) &= y_0, \\ z' &= -bz + xy, & z(0) &= z_0, \end{aligned} \tag{1}$$

has proven very challenging. Using existing numerical methods and current day computer technology, researchers have not been able to produce time-step independent numerical solutions of chaotic systems. Solutions with small but different time steps will decouple from each other after a certain (small) finite amount of simulation time. One of the defining traits of a chaotic system is its *sensitive dependence on initial conditions* which is characterized by nearby solutions separating exponentially fast due to the fact that the

system has a positive Liapunov exponent [2]. Thus, approximate solutions starting from exactly the same initial conditions but using a different time step size will drift away from each other if truncation and rounding errors prevent the approximate solution from being accurate to a *sufficient* number of decimal places.

The sensitivity of the system (1) was originally noted by Lorenz in 1963 [1]. While trying to repeat previous numerical work with a second order Runge–Kutta (RK) method, Lorenz wrote down the output of the method and stopped the long calculation. Later the method was restarted with the recorded intermediate values as initial conditions. The final result was a vastly different solution than he was trying to recreate. The disparity was caused by the recorded intermediate values not exactly matching the floating point numbers in memory.

Recent accounts of the difficulty in calculating accurate long time, time-step independent solutions of the Lorenz equations (and other chaotic equations) include the following:

- Reference [3] considers the numerical solution of chaotic differential equations in general with an application to the one-dimensional Kuramoto–Sivashinsky equation. The author concludes that "No computed chaotic solution, which is independent of the integration time-step employed, exists.".
- In [4], the following was concluded about the Lorenz equations: "Similar behavior was noted with Adams–Bashforth methods up to the fifth order; an implicit Crank–Nicholson method; second-order and forth-order Runge–Kutta methods; adaptive methods; and compact time-difference schemes. In no case was a convergent solution obtained for $t \geqslant 20$.".
- Reference [5] discusses the inability to produce time-step independent solutions of the Lorenz equations with second and fourth order RK methods. Comments supporting the conclusions of [5] were made in the note [6].
- Reference [7] uses the arbitrary precision capabilities of Mathematica with up to 800 digits of decimal precision and high order Taylor series methods of up to order 400 to calculate accurate solutions of the Lorenz equations. Extended arbitrary precision floating point arithmetic is much more computationally expensive than the extended fixed precision that we consider in this work.
- Higher order methods (order 4 to 8) with error control and adaptive step-size have been used on the Lorenz equations. The results are summarized in the book [8, p. 245] where a numerical example is explained: "The solution is, for large t, *extremely* sensitive to the errors of the first integration steps. For example, at $t = 50$ the numerical solution becomes totally wrong, even if the computations are preformed in quadruple precision with $tol = 10^{-20}$. Hence the numerical results of all methods would be equally useless and no comparison makes any sense. Therefore we chose $t_{end} = 16$."

The goal of this work is to calculate solutions of the Lorenz equations that are time-step independent up to $t = 100$ in the sense that the maximum error between two computed solutions with small but different time steps is less than a tolerance of $tol = 5\,\mathrm{e} - 14$. In addition to obtaining time step independent solutions, we use two vastly

different types of numerical methods to obtain the solutions. This is because, as pointed out in [9], that a practical way of judging the validity of the numerical results from a non-linear dynamical system is to use two or more different methods to solve the same problem. If the two solutions agree, then we can have some confidence in the computed solutions. However with chaotic systems, we can not claim that the agreeing solutions are the true solutions of the system. The two numerical methods that are used are a 11 stage, order 8, explicit Runge–Kutta method and a 8 stage, order 16, implicit Runge–Kutta method. Constant step sizes are used and the methods are implemented in floating point number systems using double, double-double, and quad-double data types.

In this work we numerically explore chaotic solutions of the Lorenz system. It should be noted that much is known theoretically about the existence of chaos in the Lorenz system as well. The first mathematical proof of such chaotic behavior in the Lorenz system was given in reference [10] in 1995. Subsequently, more efficient ways of proving chaos have been proposed [11].

2 Floating point numbers systems

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is the most widely-used standard for floating-point computation. It is followed by virtually every computer used in scientific computing. The current version, IEEE 754-2008, was published in August 2008 and is a revision of the original version that was published in 1985. Properties of three of the binary formats specified by the standard are listed in Table 1. The column labeled dps lists the number of decimal digits to which a base ten number is correctly represented to in the floating point system. In a given floating point number system, every real number is approximated and every floating point operation is performed with a relative error of at most machine epsilon, ε_m.

IEEE double floating point arithmetic is sufficiently accurate for most scientific applications. However, for a rapidly growing body of important scientific computing applications, a higher level of numeric precision is required. A sampling of such applications are surveyed in [12]. An additional example is given by the first author in radial basis functions approximation methods [13]. The addition of the quadruple type to the 2008 IEEE standard is in response to the need for more precise floating point arithmetic. To date, a hardware implementation of the quadruple type does not exist for mainstream desktop computers. When a hardware implementation of quadruple precision is available it will likely be at least twice slower than double precision [14]. The books [15] and [16] can be consulted for more information on floating point arithmetic.

An alternative to the quadruple type is the double-double type, in which the unevaluated sum of two IEEE double numbers is used to represent a number with twice the precision. A simple example is that $8.765 \times 10^5 + 4.32 \times 10^1$ can be used to represent the number 8.765432×10^5. The idea can be extended to a quad-double number that is an unevaluated sum of four IEEE doubles with four times the precision of a double. A freely available, open source C++ library is available that implements algorithms for basic arithmetic operations for the double-double and quad-double types, as well as some algebraic and transcendental functions [17]. Modifying existing C or C++ software to implement

extended precision is in most cases trivial and only requires an include statement and a define statement such as the following:

```
#include <qd/dd_real.h>
#define double dd_real
```

There is a performance penalty for precisions beyond double. Typical execution time of the formats are listed in Tables 1 and 2 where the double time has been normalized to be one. There is a range of execution time penalty factors for each extended type due to several reasons. When implemented in software, the quadruple and double-double types can be expected to run 5 to 10 times slower than the double format, with the exact factor depending on the particular computer platform, the compiler, and the compiler flags that are used. If implemented in hardware, the quadruple format should executive in twice the time that a double does [14].

Table 1. Base 2 IEEE 754-2008 formats.

type	ε_m	bits	dps	exec time
binary32 (single)	$1.2\,\mathrm{e}-7$	32	7	1
binary64 (double)	$2.2\,\mathrm{e}-16$	64	16	1
binary128 (quadruple)	$3.8\,\mathrm{e}-34$	128	33	2 to 10

Table 2. Information on floating point types derived from the IEEE double and implemented in software.

type	bits	ε_m	dps	exec time
double-double	128	$4.9\,\mathrm{e}-32$	31	5 to 10
quad-double	256	$1.2\,\mathrm{e}-63$	62	25 to 100

However, the algorithms for extended, but fixed, precision floating point systems are many times more efficient than the algorithms for arbitrary precision floating point calculations of comparable length. In our benchmarks, the C++ arbitrary precision package available from [17] with the decimal precision set to 62 (the decimal precision of a quad-double) runs 5 times slower than does the fixed quad-double precision code.

3 Runge–Kutta methods

For an autonomous system of ODEs, Runge–Kutta (RK) methods are of the form

$$y^{n+1} = y^n + \Delta t \sum_{i=1}^{s} b_i F(Y_i) \tag{2}$$

where the internal stages are determined by

$$Y_i = y^n + \Delta t \sum_{j=1}^{s} a_{ij} F(Y_j), \quad i = 1, \ldots, s, \tag{3}$$

and where s denotes the number of stages. If $a_{ij} = 0$ whenever $i \leqslant j$ the method is explicit and the equations (3) provide a recursion for computing each Y_i in terms of the proceeding stages. Otherwise, the method is implicit and the Y_i need to be calculated by solving a system of nonlinear algebraic equations. If the system of ODEs is stiff, the nonlinear equations are typically solved by Newton's method, or a variant of Newton's method, as described in [18, p. 118] in order to maintain the good stability properties of the method. Otherwise, if stiffness is not an issue, the equations can be solved by a simple fixed point iteration

$$Y_i^{[k]} = y^n + \Delta t \sum_{j=1}^{s} a_{ij} F\big(Y_j^{[k-1]}\big), \quad i = 1, \ldots, s. \tag{4}$$

The equations are iterated until the increment of two successive approximations satisfies either

$$\Delta^{[k]} = \max_{i=1,\ldots,s} \big\| Y_i^{[k]} - Y_i^{[k-1]} \big\|_\infty \leqslant \delta \tag{5}$$

or $\Delta^{[k]} \geqslant \Delta^{[k-1]}$ which indicates that the increments of the iteration start to oscillate due to roundoff error [19]. The value of δ depends on the floating point system and is taken to be a number slightly large than machine epsilon in the system. In double precision we take $\delta = 1\,\mathrm{e} - 14$, in double-double $\delta = 1\,\mathrm{e} - 28$, and in quad-double $\delta = 1\,\mathrm{e} - 56$. In all cases, the iteration typically converges in 2 to 6 iterations.

Two particular RK methods are used in the numerical experiments. The first is an eighth order explicit method with eleven stages. The a and b coefficients of the RK8 method are given in [20].

The second method, referred to as Gauss16, is an eight stage, sixteenth order, implicit Gauss RK method. The Gauss RK methods (also known as Butcher–Kuntzmann RK methods) are of order $2s$ and have the highest possible order of any RK method relative to the number of stages [21]. The coefficients of Gauss RK methods of order 2, 4, 6, 8, and 10 are listed in [22]. We are unaware of any Gauss RK method that is higher than order 10 being described in the literature. This is most likely because using such a high order method would be overkill when employed using double precision. With such a high order method, reducing a moderately small step size any further would not further reduce the overall error due to roundoff errors. However, we intend to use the method with extended precision and have derived a sixteenth order method. The coefficients of the method are listed with 65 decimal places of accuracy in the appendix.

4 Numerical results

The parameters in the system (1) have been set to $\sigma = 10$, $b = 8/3$, and $r = 28$ as these have been considered by many other authors, for example [1, 4, 5]. The initial conditions $x_0 = 1$, $y_0 = -1$, and $z_0 = 10$ have been used. In all numerical runs, the solution has been approximated from $t = 0$ to $t = 100$ and the numerical solutions have been saved and plotted at intervals of 0.1, i.e. at times $t = 0, 0.1, 0.2, \ldots, 99.9, 100$. Time step

size independent solutions are sought in the sense that solutions calculated with small but different size time steps that have a maximum difference of less than a tolerance of $tol = 5\,\mathrm{e}{-14}$. The RK8 and Gauss16 methods are used with three different floating point types: double, double-double, and quad-double. We only discuss the x variable from the system as the results for y and z are similar. With each numerical method and floating point system, we seek a time step independent solution by refining the time-step in the sequence $\Delta t = 1\,\mathrm{e}-3, 1\,\mathrm{e}-4, 1\,\mathrm{e}-5, 1\,\mathrm{e}-6$, and $1\,\mathrm{e}-7$.

The double precision results are shown in Fig. 1. In the upper left image of the figure, the $\Delta t = 1\,\mathrm{e}-6$ and $\Delta t = 1\,\mathrm{e}-7$ RK8 solutions visually agree to about $t = 42$. At $t = 0.1$ when their difference is first measured, the two solutions differ by $9.8\,\mathrm{e}{-14}$ which exceeds the set tolerance. The double precision Gauss16 $\Delta t = 1\,\mathrm{e}-6$ and $\Delta t = 1\,\mathrm{e}-7$ solutions are compared in the lower images of Fig. 1. As was the case for the RK8 double solutions, the two Gauss16 solutions visually agree to about $t = 42$. The two solutions

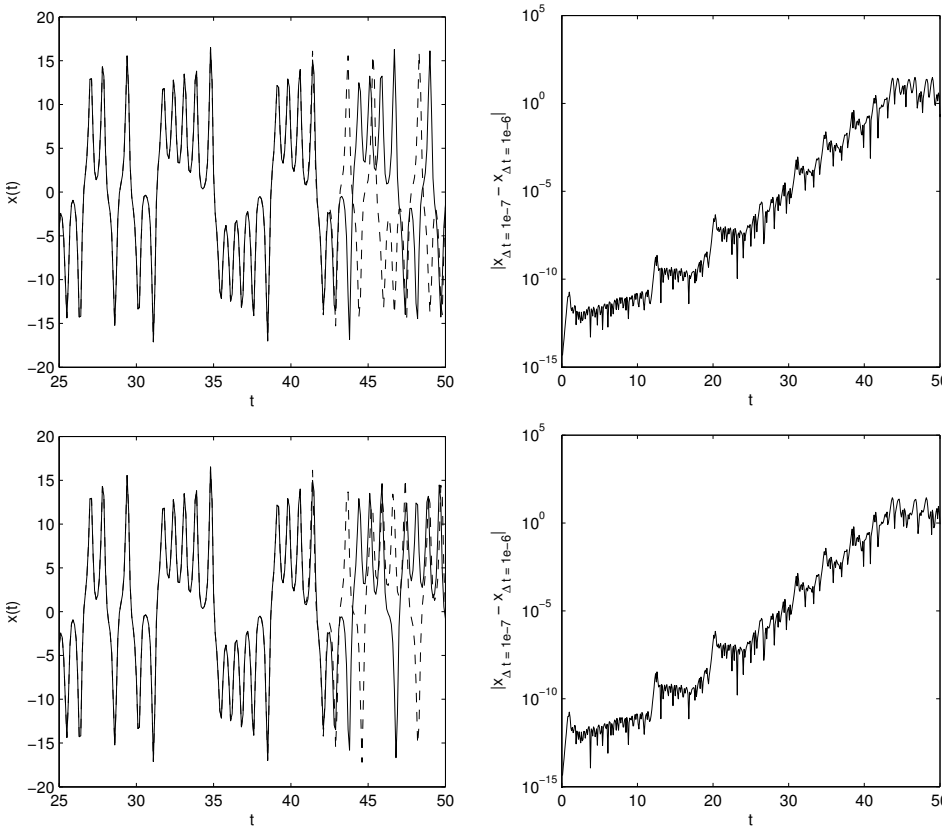

Fig. 1. Double precision, $\Delta t = 1\,\mathrm{e}-6$ and $\Delta t = 1\,\mathrm{e}-7$. Upper: RK8. Lower: Gauss16. Left: $x(t)$ solutions visually agree until about $t = 42$ for both methods. Right: logarithmic plot of the difference of the two solutions.

differ by more than the tolerance at $t = 0.3$. At first it may seem odd that the much more accurate Gauss16 method does not produce time step independent solutions over a longer period of time than does the RK8 methods. However, for such high order numerical methods, the local truncation errors of both methods may be much smaller than machine epsilon when even moderately small time steps are used. At this point, it is the lack of accuracy and rounding errors in the floating point system that is preventing a longer time period of agreement.

The double-double precision results with $\Delta t = 1\,\mathrm{e}-6$ and $\Delta t = 1\,\mathrm{e}-7$ are shown in Fig. 2. In the upper images of the figure, the RK8 solutions are visually indistinguishable up to about $t = 78$. The two RK8 numerical solutions agree to within the tolerance through time $t = 46.6$. In the lower images, the Gauss16 solutions are also visually indistinguishable up to about $t = 78$. The two Gauss16 numerical solutions agree to within the tolerance through time $t = 46.5$.

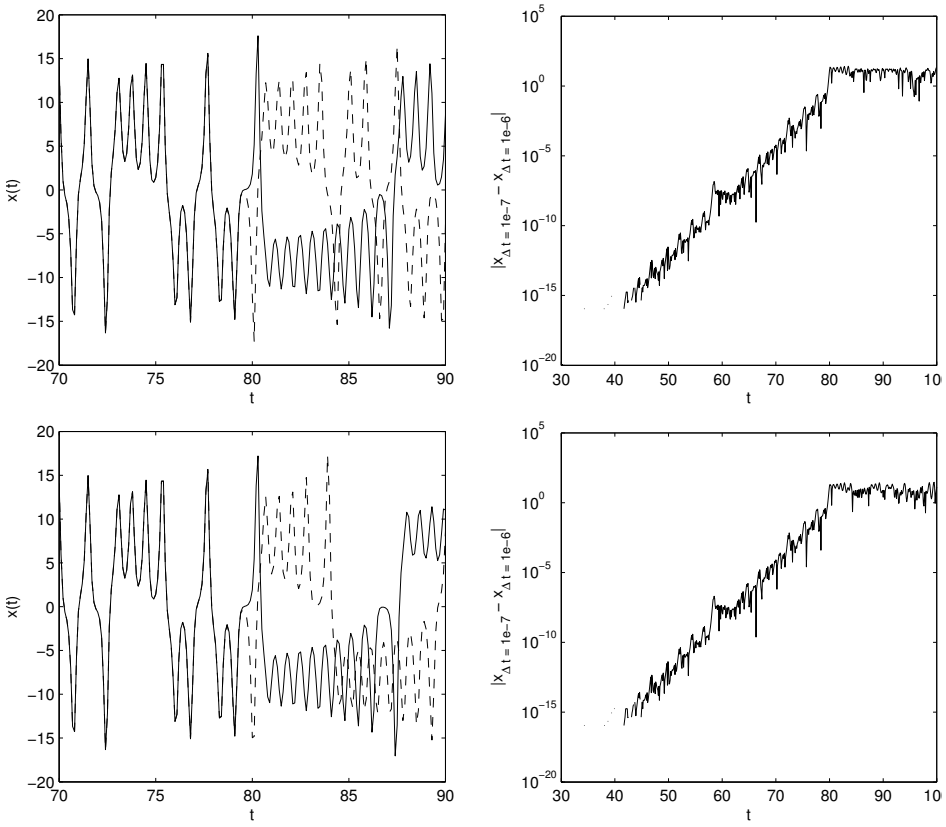

Fig. 2. Double-double precision, $\Delta t = 1\,\mathrm{e}-6$ and $\Delta t = 1\,\mathrm{e}-7$. Upper: RK8. Lower: Gauss16. Left: $x(t)$ solutions visually agree until about $t = 78$ for both methods. Right: logarithmic plot of the difference of the two solutions.

The quad-double precision RK8 results with $\Delta t = 1\,\mathrm{e} - 6$ and $\Delta t = 1\,\mathrm{e} - 7$ have a maximum difference of $2.83\,\mathrm{e}{-6}$. The solutions are visually indistinguishable to $t = 100$, but only agree to within the tolerance through time $t = 72.7$. The time step size is halved and another run is computed with $\Delta t = 5\,\mathrm{e} - 8$ and compared to the $\Delta t = 1\,\mathrm{e} - 7$ solution. The two solutions agree to within the tolerance to $t = 100$ which is the last time their difference was measured. The two solutions agree to within the double precision machine epsilon through time $t = 96.6$. The quad-double solution with $\Delta t = 1\,\mathrm{e} - 7$ is time step size independent up to $t = 100$.

The Gauss16 quad-double solutions with $\Delta t = 1\,\mathrm{e} - 3 \Delta t = 1\,\mathrm{e} - 4$ agree to within the tolerance to $t = 75.4$. Reducing the size of the time step once more we find that the Gauss16 quad-double solutions with $\Delta t = 1\,\mathrm{e} - 4 \Delta t = 1\,\mathrm{e} - 5$ not only agree within the tolerance to $t = 100$ but agree to the 16 decimal places. Additionally, the solutions also agree with the RK8 quad-double solutions to 16 decimal places.

The quad-double RK8 $\Delta t = 1\,\mathrm{e} - 7$ and $\Delta t = 5\,\mathrm{e} - 8$ and Gauss16 $\Delta t = 1\,\mathrm{e} - 4$ and $\Delta t = 1\,\mathrm{e} - 5$ all agree to within the tolerance up to $t = 100$. Additionally, the quad-double RK8 $\Delta t = 5\,\mathrm{e} - 8$ and Gauss16 $\Delta t = 1\,\mathrm{e} - 4$ and $\Delta t = 1\,\mathrm{e} - 5$ are in agreement up to 16 decimals places to $t = 100$. This solution is taken as our time step independent solution. The solution is plotted in Fig. 3 and is tabulated at intervals of $t = 10$ in Table 3. In Table 4, the agreement with the time step independent solution and RK8 and Gauss16 in double-double and double precision with $\Delta t = 1\,\mathrm{e} - 7$ is listed.

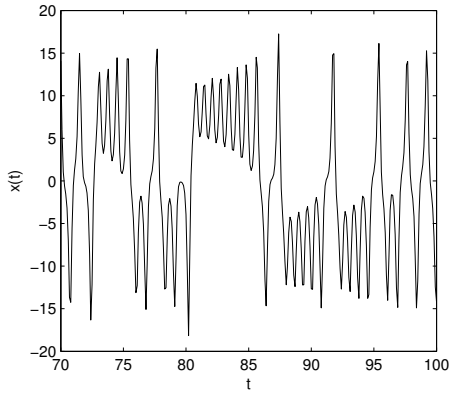

Fig. 3. The time step size independent (16 decimal places) $x(t)$ solution.

Another number that was recorded in all numerical runs was the maximum absolute value of $x(t)$ over the time interval $[0, 100]$. To fifteen decimal places, all RK8 double-double, and quad-double runs with $\Delta t = 1\,\mathrm{e} - 5$ to $5\,\mathrm{e} - 8$ agree that

$$\max_{t \in [0,100]} \left| x(t) \right| = 18.422269920984803. \tag{6}$$

All Gauss16 runs with double-double and $\Delta t = 1\,\mathrm{e} - 4$ to $1\,\mathrm{e} - 7$ and quad-double runs with $\Delta t = 1\,\mathrm{e}{-2}$ to $1\,\mathrm{e}{-5}$ agree with the above number to fifteen decimal places as well.

Table 3. Time step independent solutions computed with quad-double precision and Gauss16 with $\Delta t = 1\,\mathrm{e} - 4$ and RK8 with $\Delta t = 5\,\mathrm{e} - 8$.

t	$x(t)$
10	6.0522357030842335
20	3.0798989869880050
30	−7.5894934859019713
40	6.7582931863137214
50	1.4275216839127140
60	−4.9386364320497773
70	14.0746063398783966
80	−1.4271159848437984
90	−12.6554314800994861
100	−14.2975549270969643

Table 4. The agreement with the time step independent solutions of RK8 and Gauss16 in double-double and double precision with $\Delta t = 1\,\mathrm{e} - 7$.

t	RK8-dd	Gauss8-dd	RK8-d	Gauss16-d
10	6.0522357030842	6.0522357030842335	6.052235703	6.052235703
20	3.0798989869880	3.0798989869880050	3.079898	3.079898
30	−7.5894934859019	−7.5894934859019713	−7.589	−7.589
40	6.7582931863137	6.7582931863137214	6	6
50	1.42752168391	1.42752168391	−	−
60	−4.93863643	−4.9386364	−	−
70	14.0746	14.0746	−	−
80	−	−	−	−
90	−	−	−	−
100	−	−	−	−

5 Conclusions

Using quad-double floating point arithmetic, we have calculated time step independent solutions up to $t = 100$ with two different numerical methods, an eighth order explicit Runge–Kutta method and a sixteenth order implicit Runge–Kutta method that agree to 16 decimal places. The solutions may serve as benchmarks in other numerical studies.

Calculating numerical solutions of chaotic dynamical systems that are time step independent over long time periods remains a challenging problem when using computer technology that is currently available. Standard double precision floating point number systems are not adequate for this purpose. A quadruple precision type has been added to the IEEE 754 standard that was revised in 2008. However, to date, the quadruple type has not been implemented in hardware or software. Software packages are available that implement double-double and quad-double floating point arithmetic that offer two and four times more decimal precision than the double type. A performance penalty is incurred in using software floating point systems, however the fixed extended types are

more efficient than the arbitrary precision extended types where the decimal precision is user specified. Although not yet available, it is possible to extend the quad-double algorithms [23] to develop floating point systems that have 8 or 16 times the decimal precision of an IEEE double. These more precise floating point systems will be necessary for obtaining time-step independent numerical solutions over longer time periods.

Despite employing very precise floating point number systems and very high order numerical methods, we were unable to compute time step size independent solutions significantly beyond $t = 100$, which cannot be considered a large time in computational chaos. If even more precise floating point numbers systems than quad-double were used with even higher order numerical methods, this finite interval of time step independent solutions could be extended. However, extending the finite interval to an infinite interval is impossible due to the chaotic nature of the true solution.

As the numerical experiments that have been presented in this work were being completed, the authors became aware of emerging technology that potentially make computing with the quad-double software hundreds of times faster. In [24] the authors describe a port of the quad-double package that runs on general purpose graphics processor units (gpgpus). The software is freely available from `http://code.google.com/p/gpuprec/`. In benchmarks, the gpgpu based quad-double package ran up to 27 times faster than the cpu quad-double code. The gpgpu benchmarks were made using the Nvidia 280. Modern gpgpus such as the Nvidia Tesla M2070 claim to offer float point operations that are up to 250 times faster than their cpu based counterparts. The new gpgpu technology and the gpgpu based quad-double package should allow numerical experiments, such as we have conducted in this work, to be performed much more efficiently than can be done with a computer with only cpus.

Appendix: Guass16 coefficients

The coefficients of an implicit 8 stage Gauss Runge–Kutta that is sixteenth order. The coefficients are accurate to 65 decimal places.

$$
\begin{aligned}
a_{11} &= 0.025307134072594064788132838577490547528848522762921239264750924 51\\
a_{12} &= -0.009105943305970075021363652104664350473117272473296864293486235 86\\
a_{13} &= 0.006280831147030473980609420737807597954258358376305131883423226 51\\
a_{14} &= -0.004483015613054751441913314235301548147590609166011393022915256 11\\
a_{15} &= 0.003078491368326779889853830649117115206609020301678813700709456 60\\
a_{16} &= -0.001917675254636952340925635119494438564064489383086963501632983 04\\
a_{17} &= 0.000972757664059263526724523824088790494697049029233802386038858 37\\
a_{18} &= -0.000277508327116919222898446613780209213758196598469785552707879 67
\end{aligned}
$$

$$
\begin{aligned}
a_{21} &= 0.054759321767554320283190797385002248268758043645510051810823692 57\\
a_{22} &= 0.055595258613343617636088998606560221107532717512812391181477322 32\\
a_{23} &= -0.013639796235781669253030662069976092247068025242869062572834181 17\\
a_{24} &= 0.008149708858360550536040427483729163268358221621906200141441829 26\\
a_{25} &= -0.005215352089147153380243369774637649244931202370908292126348767 53\\
a_{26} &= 0.003139752985463668942345006901156272652894716207562353582102429 80
\end{aligned}
$$

$$a_{27} = -0.00156493491094894237579511140443043454693422650166514567168147496$$
$$a_{28} = 0.00044280230434223781562694463468105232280388931966908761993400219$$

$$a_{31} = 0.04858753599891288094316014734743885745625614201333994426404020094$$
$$a_{32} = 0.12085952499717316744165374300348485377887792061396792806062120607$$
$$a_{33} = 0.07842666146947182183449055049665032831508224975068373442256598626$$
$$a_{34} = -0.01597510336187843147567011385569651343428866826004589980387830362$$
$$a_{35} = 0.00837173272022616378691032206439061467670572012238163428187991288$$
$$a_{36} = -0.00464346586210447979012617007024473073788065378785656721211938560$$
$$a_{37} = 0.00222571477528489971831530140789318479957698318588276219817862214$$
$$a_{38} = -0.00061880569525051536760330498853976937531181519854996508671678873$$

$$a_{41} = 0.05186552097058123456531207796496692699577807729965268252186029215$$
$$a_{42} = 0.10619349014834840687886945008258241588778041538431551297916346689$$
$$a_{43} = 0.17067113427455362128471078460399407590399126470854040670372947278$$
$$a_{44} = 0.09067094584459049574128761231929890304853650997358263513120576689$$
$$a_{45} = -0.01602104132102501316870521779380082716174561379583905402608604165$$
$$a_{46} = 0.00724120656122226986180306301493880973632538238413779187600040811$$
$$a_{47} = -0.00319781431036077032248274309542092890356476150527201930790883803$$
$$a_{48} = 0.00085923658426485268946690172334863415951981909439515296344123599$$

$$a_{51} = 0.04975503156092327688679877543163246089817722643144732556606061303$$
$$a_{52} = 0.11438833153704800559466074030854137111863019653089680167086348268$$
$$a_{53} = 0.14961211637772137380717803797836184689383911711722967696913156442$$
$$a_{54} = 0.19736293301020600465128044243239863325881863374300432428849757543$$
$$a_{55} = a_{44}$$
$$a_{56} = -0.01381781133560997761572968361069341927382676520717293785859750024$$
$$a_{57} = 0.00499702707833882839330854713053802632728501964130926938379117774$$
$$a_{58} = -0.00125125282539310498904640080998583193808103177381020399235844312$$

$$a_{61} = 0.05123307384043864494386898214352086443300886072439244361621863776$$
$$a_{62} = 0.10896480245140233555386269580522725741548845183974202016477602249$$
$$a_{63} = 0.16149678880104812345910727106354538736804515328922403605725135813$$
$$a_{64} = 0.17297015896895482769566490257420719142036729982478363598053162089$$
$$a_{65} = 0.19731699505105942295824533849429431953136168820721117006628983741$$
$$a_{66} = a_{33}$$
$$a_{67} = -0.00966900777048593216947574579036441156381248558834314569766656142$$
$$a_{68} = 0.00202673214627524863310552980754223760144090351250253426546164808$$

$$a_{71} = 0.05017146584084589176063873252030004273489315620617339090956784684$$
$$a_{72} = 0.11275545213763617764797310861755087676199966152728992803463611960$$
$$a_{73} = 0.15371356995347997472663609409214438397726978329380511526302954273$$
$$a_{74} = 0.18655724377832814486281859441323545534200422231807356238876030132$$
$$a_{75} = 0.17319218283082044094653479715486864282871479832525907012096970452$$
$$a_{76} = 0.17049311917472531292201176306327674887723252474423653141796615371$$
$$a_{77} = a_{22}$$
$$a_{78} = -0.00414505362236619070692512023002115321106099811966757328132184354$$

$$a_{81} = 0.05089177647230504879916412376876130427145524212431226408220972870 7$$
$$a_{82} = 0.11021775956262797174545347338903165172036838599639097997691578627 5$$

$$a_{83} = 0.158770998193580596009906736112795095194228988884454432346764955579$$
$$a_{84} = 0.178263400320854211592721393989480690890463999645486456561702077174$$
$$a_{85} = 0.185824907302235742924488538873899354244663629113176663285326789898$$
$$a_{86} = 0.150572491791913169688371680255493058675906141125062336961708746019$$
$$a_{87} = 0.120296460532657310293541649317784792688182707498921646656440880515$$
$$a_{88} = a_{11}$$

$$b_1 = 0.050614268145188129576265677154981095057697045525842478529501849032$$
$$b_2 = 0.111190517226687235272177997213120442215065435025624782362954644646$$
$$b_3 = 0.156853322938943643668981100993300656630164499501367468845131972537$$
$$b_4 = 0.181341891689180991482575224638597806097073019947165270262411533783$$
$$b_5 = b_4$$
$$b_6 = b_3$$
$$b_7 = b_2$$
$$b_8 = b_1$$

References

1. E. Lorenz, Deterministic nonperiodic flow, *Journal of Atmospheric Sciences*, **20**, pp. 130–141, 1963.
2. S.H. Strogatz, *Nonlinear Dynamics and Chaos*, Westview Press, 1994.
3. L.-S. Yao, Is a direct numerical simulation of chaos possiblle? A study of a model nonlinearity, *Int. J. Heat Mass Transfer*, **50**, pp. 2200–2207, 2007.
4. L.-S. Yao, Computed chaos or numerical errors, textitNonlinear Anal. Model. Control, **15**(1), pp. 109–126, 2010.
5. J. Teixeira, C. Reynolds, K. Judd, Time step sensity of nonlinear atmospheric models: Numerical convergence, truncation error growth, and ensemble design, *Jornal of the Atmospheric Sciences*, **64**, pp. 175–189, 2007.
6. L.-S. Yao, D. Hughes, Comment on "Time step sensitivity of nonlinear atmospheric models: Numerical convergence, truncation error growth, and ensemble design" by J. Teixeira, C. Reynolds, K. Judd, *Journal of the Atmospheric Sciences*, **65**, pp. 681–682, 2008.
7. S. Liao, On the reliability of computed chaotic solutions of non-linear differential equations, *Tellus A*, **61**(4), pp. 550–564, 2009.
8. E. Hairer, S. Norsett, G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer Ser. Comput. Math., Vol. 1, Springer, 2000.
9. T.S. Parker, L.O. Chua, *Practical Numerical Algorithms for Chaotic Systems*, Springer-Verlag, New York, 1989.
10. K. Mishaikow, M. Mrozek, Chaos in Lorenz equations: A computer assisted proof, *Bull. Am. Math. Soc.*, **32**(1), pp. 66–72, 1995.
11. P. Zgliczynski, Fixed point index for iterations of maps, topological horseshoe and chaos, *Topol. Methods Nonlinear Anal.*, **8**, pp. 169–177, 1996.
12. D.H. Bailey, High-precision arithmetic in scientific computation, *Comput. Sci. Eng.*, **7**(3), pp. 54–61, 2005.

13. S.A. Sarra, Radial basis function approximation methods with extended precision floating point arithmetic, *Eng. Anal. Bound. Elem.*, **35**(1), pp. 68–76, 2011.

14. F. de Dinechin, G. Villard, High precision numerical accuracy in physics research, *Nucl. Instrum. Meth. A*, **559**, pp. 207–210, 2006.

15. J.-M. Muller et all., *Handbook of Floating-Point Arithmetic*, Birkhäuser, Boston, 2010.

16. M. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, 2001.

17. D. Bailey, Y. Hida, X. Li, B. Thompson, High precision software, `http://crd.lbl.gov/~dhbailey/mpdist/`.

18. E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer Ser. Comput. Math., Vol. 14, Springer, 2000.

19. E. Hairer, R.I. McLachlan, A. Razakarivony, Achieving Brouwer's law with implicit Runge–Kutta methods, *BIT*, **48**(2), pp. 231–243, 2008.

20. G.J. Cooper, J.H. Verner, Some explicit Runge–Kutta methods of high order, *SIAM J. Numer. Anal.*, **9**(3), pp. 389–405, 1972.

21. J. Butcher, *Numerical Methods for Ordinary Differential Equations*, Wiley, 2003.

22. J.C. Butcher, Implicit Runge-Kutta processes, *Math. Comput.*, **18**(85), pp. 50–64, 1964.

23. Y. Hida, X.S. Li, D.H. Bailey, Algorithms for quad-double precision floating point arithmetic, in: *Proceedings of the 15th IEEE Symposium on Computer Arithmetic, IEEE Computer Society*, pp. 155–162, 2001.

24. M. Lu, B. He, Q. Luo, Supporting extended precision on graphics processors, in: *Proceedings of the Sixth International Workshop on Data Management on New Hardware, DaMoN'10*, pp. 19–26, 2010.