Tools for the Verification of Approximate Solutions to Differential Equations *

W.H. Enright
Department of Computer Science
University of Toronto
Toronto, Canada M5S 3G4

enright@cs.utoronto.ca

Abstract

Scientific computation is often carried out in general purpose problem solving environments (PSEs) such as MATLAB or MAPLE or in application-specific PSEs such as those associated with large scale, high performance simulations. When practitioners work in these environments it is essential that they have access to state-of-the-art numerical software and tools to visualize approximate solutions produced by such software. It is equally essential (although not yet standard practice) to have available tools that can be used to verify (or validate) that the approximate solution is consistent with the true solution of the underlying mathematical model. In this presentation we will identify a suite of tools that are being developed to address this need for the case where the underlying model is a system of ODEs or PDEs.

The resulting suite of tools can be used to improve the practitioners confidence in the results and allow him to quantify the inherent reliability/cost trade-offs that arise. The collection of generic tools are suitable for use with any numerical method and cover a range of cost/complexity. We will assume that the approximate solution is provided as a piecewise (possibly multivariate) polynomial on an arbitrary mesh and the tools would perform such tasks as measuring the size of the associated defect, estimating the global error , estimating the condition number and checking that a different method generates a consistent approximate solution. The application of the resulting tools to two typical simulations are presented.

*This research was supported by the Natural Science and Engineering Research Council of Canada.

1 Introduction

In the last decade there has been a significant change in the way scientific computing is carried out. Software developers must be aware of this change if their software is is ever going to be widely used and have an impact. Two environments where numerical software is now widely used are large scale scientific computation and problem solving environments (PSEs). The first of these is characterized by simulations generating massive amounts of data and involving expensive runs on High Performance distributed systems (often located at remote sites). Applications often require data compression, interactive data viewing (or data mining), remote visualization and involve the distributed collaboration of experts.

A PSE is often where students are first exposed to scientific computing. It is also the environment where scientists and engineers formulate and investigate approximate solutions to their problems. In this paper we will focus on the latter environment but the tools we develop and the observations we make are also applicable in the former environment.

In the next section we will identify various aspects of a PSE environment that affect how numerical software can be implemented and used effectively. We will then justify and develop four 'verification tools' for use with standard ODE solvers. In the fourth section we will present the application of these tools to two typical simulations and show how the tools we have developed can give insight into and understanding of the quality of the approximate solution. We will conclude with a discussion of current and future extensions of this work.

2 Characteristics of a PSE

Although the basic problem we will consider is to approximate the continuous solution $y(x)$ on a specified interval of interest, $[a, b]$, most numerical methods partition the interval $[a, b]$ into a discrete set of meshpoints $a = x_0 < x_1 < \cdots x_N = b$ and provide approximate solution values, $y_i \approx y(x_i)$, only for these meshpoints. In a PSE the focus is on visualizing and investigating various aspects of approximate solutions. Visualization generally involves graphical representation of a 'subset' of the results. It can involve the use of colour, texture, lighting, animation and sound. Although stringent accuracy is rarely necessary, off-mesh accuracy is frequently required. It is essential that the tools and packages that are provided in a PSE be easy to understand and use. The interface and documentation must be simple and structured in a hierarchichal fashion. (Information should be hidden unless necessary to define the problem.) In this environment ease of use is as important as efficiency.

A consequence of these observations is that software that is used in PSEs should satisfy some extra design requirements. For example, robustness is as important as reliability. In particular it is acceptable for the software to fail, as long as it is done gracefully and early. It is also important to adopt a standard interpretation of error control, and a standard program interface (calling sequence) for methods that perform the same or similar tasks. Similarly options and additional parameters (if required) should be specified in the same way for all methods. This includes mesh-constraints, initial guess of solution, specification of partial derivatives, and accuracy specification. It is also important to adopt a standard representation of an approximate solution. (For example, in the case of solving sytems of ODEs one can choose to represent an approximate solution by a vector of piecewise polynomials.)

An effective PSE must provide tools to:

1. Formulate a mathematical model of the problem.
2. Approximate the solution of the mathematical problem.
3. Visualize an approximate solution.
4. Verify that an approximate solution is consistent with the mathematical model and the requested accuracy.
5. Verify that the mathematical model is well-posed. (That is, quantify the underlying inherent conditioning.)

Of these requirements, the first three are satisfied in virtually all PSEs for most classes of problems, while the latter two requirements are often not satisfied. The verification tools that do address these latter requirements will generally be designed to be used for particular problem classes but they are a very important part of mathematical modeling. In the next section we will develop and justify a collection of such tools for use when solving ODE problems in a PSE. We will use the MATLAB PSE [3] and in particular the numerical method, ode45 to illustrate the details and trade-offs one faces when introducing and using these tools. For an good introduction to MATLAB and its use in scientific computation see [2] and for details and guidance in formulating and visualizing mathematical models in MATLAB involving sytems of ODEs see [4].

3 Verification tools for use with an ODE Solver

If the underlying numerical method provides a continuous approximation to the solution (over the domain of interest) then generic tools for the verification of an approximate solution and the estimation of the mathematical conditioning can be developed. Examples of such tools that would be useful and which we have implemented include those based on assessing the magnitude of the defect (or residual) of the approximate solution; estimating the global error by solving the same problem with a more stringent acuracy request or with a different method; and estimating the global error by determining the exact error associated with the approximate solution of a 'nearby problem' with the same method.

The key assumption we make of the methods which generate the approximate solutions is that, when applied to the IVP,

$$y' = f(x, y), \; y(a) = y_0, \text{ on } [a, b],$$

with a specified accuracy, TOL, the numerical method, M, generates a piecewise polynomial, $z_{TOL}(x)$, defined for $x \in [a, b]$ and satisfying,

$$\|z_{TOL}(x) - y(x)\| \leq K_M TOL.$$

This assumption is satisfied by most state-of-the-art ODE software and certainly by the built-in numerical methods of MATLAB [5]. Note that K_M can depend on the method and the problem and can be interpreted as the 'numerical condition number' associated with method M.

The restrictions we imposed on the tools we produced include:

1. They must be easy to use – If possible the calling sequence should be the same as that used to generate the approximate solution with no more information required.

2. The 'cost' associated with applying the tool should not be much greater than that involved in generating the approximate solution (or visualizing it).

3. The calling sequence should be the same for any method it is applied with. The tool may require some knowledge of the method but this information is transparent to the user. Some of the tools can be independent of the method.

The four verification tools we have implemented for ODEs in MATLAB are:

Check1: A consistency check based on solving the problem with a more stringent value for TOL and returning the difference in the two approximations as a piecewise polynomial. That is we compute,

$$E_{TOL}(x) = z_{TOL}(x) - z_{TOL1}(x),$$

where $TOL1 < TOL$. The 'cost' of computing this is about double the cost of generating $z_{TOL}(x)$.

Check2: A consistency check based on solving the problem with an alternative method, $\tilde{M}$. The piecewise polynomial returned by this routine is,

$$\tilde{E}_{TOL}(x) = z_{TOL}(x) - \tilde{z}_{TOL}(x)$$

where $\tilde{z}_{TOL}(x)$ is the approximate solution associated with $\tilde{M}$. (Note that this check is most credible if the method $\tilde{M}$ is very different from M. For example in our implementation of this tool we use a variable order Adams method when assessing a Runge Kutta method from MATLAB.)

Check3: A check which computes the defect,

$$D_{TOL}(x) = z'_{TOL}(x) - f(x, z_{TOL}(x)).$$

Note that although $D_{TOL}(x)$ is not a piecewise polynomial it is a function that can be evaluated at any point in $[a, b]$.

Check4: A consistency check based on the idea of Zadunaisky [6] where we determine a piecewise polynomial,

$$EST_{TOL}(x) = z_{TOL}(x) - w_{TOL}(x),$$

where $w_{TOL}(x)$ is the approximate solution, produced by method M (applied to the perturbed IVP, $z' = f(x, z) + D_{TOL}(x)$) on the same mesh that was used to determine $z_{TOL}(x)$.

For each of the vector-valued functions returned by Check1, Check2 and Check4 we can interpret its max value for $x \in [a, b]$ to be a measure of the maximum global error. Therefore a crude indicator of the numerical condition number is the ratio of this max value to TOL. Similarly for Check3, the maximum magnitude of the components of $D_{TOL}(x)/TOL$ can be considered an indication of the contribution of the numerical method to the overall numerical conditioning of the problem (it does not reflect the underlying mathematical conditioning of the problem).

4 Two examples of the use of these tools

A predator-prey relationship can be modeled by the well-known IVP:

$$y_1' = y_1 - 0.1y_1y_2 + 0.02x$$

$$y_2' = -y_2 + 0.02y_1y_2 + 0.008x$$

with

$$y_1(0) = 30, \quad y_2(0) = 20, \quad \text{and } x \in [0, 100].$$

where $y_1(x)$ represents the 'prey' population at time x and $y_2(x)$ represents the 'predator' population at time x. The solution can then be visualized as a standard x/y solution plot or by a 'phase portrait' plot.

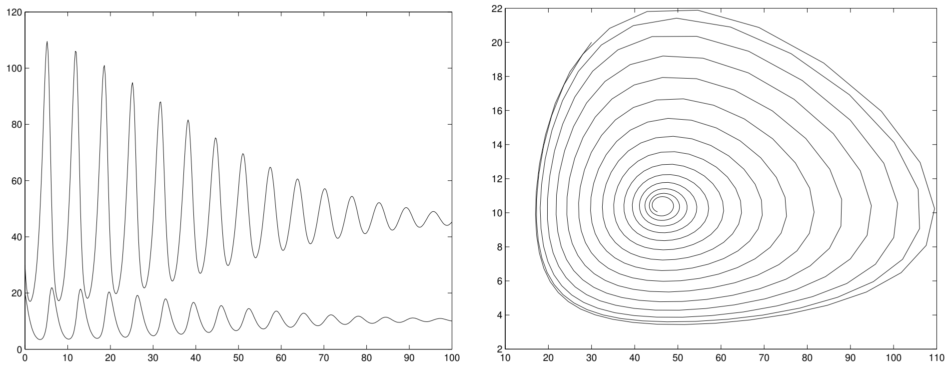
A second example is the well-known Lorenz problem which arises in the study of dynamical systems and is known to have solutions which are potentially poorly conditioned. We will consider the IVP,

$$\begin{aligned} y_1' &= 10(y_2 - y_1) \\ y_2' &= y_1(28 - y_3) - y_2 \\ y_3' &= y_1y_2 - \frac{8}{3}y_3 \end{aligned}$$

with

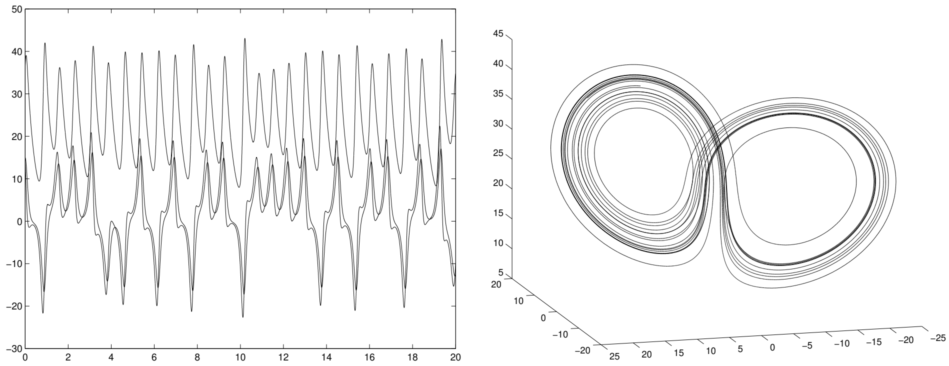
$$y_1(0) = 15, \ y_2(0) = 15, \ y_3(0) = 36, \quad \text{and } x \in [0, 20].$$

Using ode45 to investigate solutions of these problems allows us to illustrate the ability of our tools to detect potential conditioning difficulties. Figures 1-2 show visualizations of numerical solutions to these problems determined by ode45 and



(a) Solution Plot (b) Phase Portrait

Figure 1: The approximate solution produced by ode45 for the predator-prey problem



(a) Solution Plot (b) Phase Portrait

Figure 2: The approximate solution produced by ode45 for the Lorenz problem

tables 1-4 and figures 3-10 report the results of applying our verification tools to a selection of numerical solutions computed by ode45.

In the tables, whenever a maximum value is reported it has been determined numerically by sampling over 500 equally spaced points in the interval of integration. In each table we report the number of steps required by ode45 to generate the numerical solution, NST1 and the number of extra steps, NST2 required by the respective tool. (This latter count quantifies the cost of applying the tool relative to that of generating the approximate solution.) For

Check1, Check2 and Check4 results also include the indicator of the numerical condition number, CoNum. Similarly for Check3 we include the indicator of the contribution of the method to the numerical conditioning, NCon.

The method ode45 allows one to specify a mixed relative and absolute error tolerance by setting ATOL and RTOL respectively. For the predator-prey problem, as figure 1 shows, the components of the solution remain bounded and positive and a relative error control is appropriate. We have chosen to use a range of RTOL values that would be typical in a PSE – ($10^{-3}, 10^{-5}$and 10^{-7}). In cases where we report a norm of a vector (as one of the statistics) we tabulate for each component of the vector, the magnitude of that component relative to the magnitude of the corresponding solution component. That is we report a measure of the component-wise relative error. For the Lorenz problem, as figure 2 shows, $y_1(x)$ and $y_2(x)$ oscillate around zero while $y_3(x)$ remains positive. A mixed relative/absolute error control is appropriate in this case and we have used RTOL = ATOL with a range of values – (10^{-6}, 10^{-8}, and 10^{-10}).

In visualizing the results of the verification tools (that is the respective piecewise-polynomials or function) for the Lorenz problem (where exponential growth in the global error, as a function of the length of the integration interval is to be expected) we found that standard plots were not particularly helpful. We found it more meaningful to present plots of the logarithm of the magnitude of the respective global error estimate for each component of the approximate solution and this is what is presented in figures 3-10. From these results one observes a consistent estimate of the condition number for each of the relevant tools (Check1, Check2 and Check4) at each tolerance. This condition number estimate can be seen to be exponential in the length of the interval of integration. That is, if the interval were $[0, T]$, then the condition number estimate for each component roughly satisfies

$$CoNum \approx 10^{0.45T}.$$

This observation can be used to help decide when a higher order method should be used (because of the need for stringent values of TOL) and when higher precision (such as quadruple precision) would be necessary in a computation.

Table 3 and figures 8-10 reveal that the size of the defect and the associated potential contribution to the overall mathematical conditioning are both greater than would be expected, especially at the more stringent values of $RTOL$. The reason for this is that the piecewise polynomial returned by ode45 has a corresponding defect that is only $O(h^4)$ while it is well known that a piecewise polynomial with an $O(h^5)$ defect is available for any fifth order numerical method(see for example [1]). We have derived such a generic interpolant at a cost of two extra derivative evaluations per step and the resulting piecewise polynomial, $\bar{z}_{TOL}(x)$, has an associated defect, $D2(x)$. Table 3 reports the maximum magnitude of this defect and the corresponding indicator of the numerical contribution to the overall mathematical conditioning, NCon2.

We have implemented these tools as MATLAB functions and used them when solving a small set of sample test problems. The numerical experience reported here for ode45 on the two problems is typical of our experience with other problems. The tests show, as one would expect from the definitions, that Check2 gives a less accurate estimate of the magnitude of the global error (than that provided by the more expensive verification tools, Check1 and Check3). On the other hand, all the tools give a consistent confirmation when the results are reliable. In particular they allow one to have confidence in the quality of solution and they all indicate that the predator-prey problem is well conditioned over the specified range of TOL values and that the Lorenz problem is poorly conditioned (but that accurate approximate solutions are possible as long as appropriate values of TOL are specified).

$RTOL$	$max\|E(x)\|_R$	CoNum	NST1	NST2
10^{-3}	$.16 \cdot 10^{-1}$	15.8	98	232
	$.26 \cdot 10^{-1}$	26.2		
10^{-5}	$.31 \cdot 10^{-4}$	3.1	232	582
	$.38 \cdot 10^{-4}$	3.8		
10^{-7}	$.23 \cdot 10^{-6}$	2.3	582	1461
	$.22 \cdot 10^{-6}$	2.2		

(a) Predator-Prey Problem

TOL	$max\|E(x)\|_R$	CoNum	NST1	NST2
10^{-6}	$.35 \cdot 10^{2}$	$.35 \cdot 10^{8}$	1086	2754
	$.21 \cdot 10^{3}$	$.21 \cdot 10^{9}$		
	$.31 \cdot 10^{1}$	$.31 \cdot 10^{7}$		
10^{-8}	$.85 \cdot 10^{2}$	$.85 \cdot 10^{10}$	2754	6914
	$.23 \cdot 10^{2}$	$.23 \cdot 10^{10}$		
	$.64 \cdot 10^{0}$	$.64 \cdot 10^{8}$		
10^{-10}	$.23 \cdot 10^{0}$	$.23 \cdot 10^{10}$	6914	17358
	$.63 \cdot 10^{-1}$	$.63 \cdot 10^{9}$		
	$.13 \cdot 10^{-1}$	$.13 \cdot 10^{9}$		

(b) Lorenz Problem

Table 1: Check1 Results for ode45

$RTOL$	$max\|\tilde{E}(x)\|_R$	CoNum	NST1	NST2
10^{-3}	$.20 \cdot 10^{-1}$	19.5	98	271
	$.28 \cdot 10^{-1}$	28.5		
10^{-5}	$.11 \cdot 10^{-3}$	10.6	232	515
	$.12 \cdot 10^{-3}$	12.1		
10^{-7}	$.91 \cdot 10^{-6}$	9.1	582	772
	$.12 \cdot 10^{-5}$	12.0		

(a) Predator-Prey Problem

TOL	$max\|\tilde{E}(x)\|_R$	CoNum	NST1	NST2
10^{-6}	$.64 \cdot 10^{2}$	$.64 \cdot 10^{8}$	1086	1530
	$.25 \cdot 10^{4}$	$.25 \cdot 10^{10}$		
	$.13 \cdot 10^{1}$	$.13 \cdot 10^{7}$		
10^{-8}	$.42 \cdot 10^{3}$	$.42 \cdot 10^{11}$	2754	2291
	$.17 \cdot 10^{2}$	$.17 \cdot 10^{10}$		
	$.13 \cdot 10^{1}$	$.13 \cdot 10^{9}$		
10^{-10}	$.18 \cdot 10^{0}$	$.18 \cdot 10^{10}$	6914	3141
	$.52 \cdot 10^{-1}$	$.52 \cdot 10^{9}$		
	$.11 \cdot 10^{-1}$	$.11 \cdot 10^{9}$		

(b) Lorenz Problem

Table 2: Check2 Results for ode45

$RTOL$	$max\|D(x)\|_R$	NCon	$max\|D2(x)\|_R$	NCon2	NST1
10^{-3}	$.13 \cdot 10^{1}$	17.4	$.12 \cdot 10^{1}$	15.5	98
	$.44 \cdot 10^{1}$	16.6	$.14 \cdot 10^{1}$	17.3	
10^{-5}	$.18 \cdot 10^{0}$	18.8	$.19 \cdot 10^{-1}$	6.5	232
	$.42 \cdot 10^{-1}$	15.0	$.38 \cdot 10^{-2}$	6.8	
10^{-7}	$.24 \cdot 10^{-3}$	69.8	$.16 \cdot 10^{-4}$	2.4	582
	$.42 \cdot 10^{-3}$	60.0	$.44 \cdot 10^{-4}$	2.4	

(a) Predator-Prey Problem

TOL	$max\|D(x)\|_R$	NCon	$max\|D2(x)\|_R$	NCon2	NST1
10^{-6}	$.63 \cdot 10^{-3}$	9.8	$.20 \cdot 10^{-3}$	1.6	1086
	$.75 \cdot 10^{-3}$	18.0	$.70 \cdot 10^{-4}$	1.6	
	$.82 \cdot 10^{-3}$	13.0	$.12 \cdot 10^{-3}$	1.0	
10^{-8}	$.19 \cdot 10^{-4}$	16.0	$.15 \cdot 10^{-5}$	1.1	2754
	$.77 \cdot 10^{-4}$	26.0	$.11 \cdot 10^{-6}$	1.7	
	$.37 \cdot 10^{-4}$	39.0	$.24 \cdot 10^{-5}$	.82	
10^{-10}	$.65 \cdot 10^{-7}$	40.0	$.13 \cdot 10^{-7}$	1.1	6914
	$.67 \cdot 10^{-6}$	59.0	$.99 \cdot 10^{-8}$	1.4	
	$.11 \cdot 10^{-4}$	87.0	$.87 \cdot 10^{-7}$	.84	

(b) Lorenz Problem

Table 3: Check3 Results for ode45

$RTOL$	$max\|EST(x)\|_R$	CoNum	NST1
10^{-3}	$.34 \cdot 10^{-1}$	33.7	98
	$.18 \cdot 10^{-1}$	18.2	
10^{-5}	$.25 \cdot 10^{-4}$	2.5	232
	$.26 \cdot 10^{-4}$	2.6	
10^{-7}	$.24 \cdot 10^{-6}$	2.4	582
	$.22 \cdot 10^{-6}$	2.2	

(a) Predator-Prey Problem

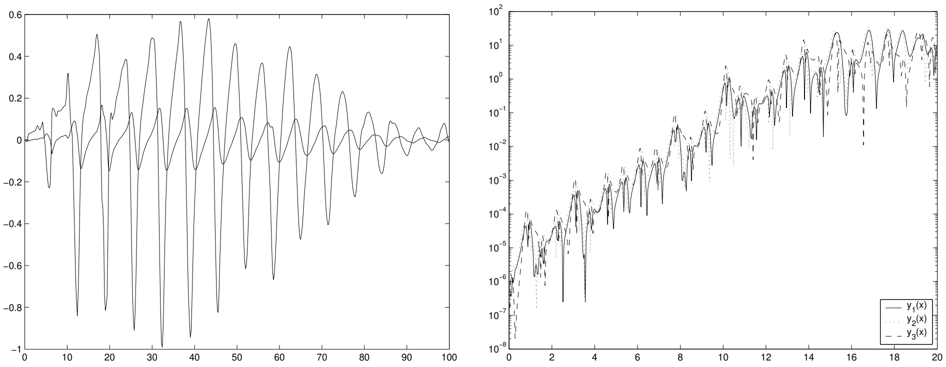
TOL	$max\|EST(x)\|_R$	CoNum	NST1
10^{-6}	$.27 \cdot 10^{3}$	$.27 \cdot 10^{9}$	1086
	$.24 \cdot 10^{2}$	$.24 \cdot 10^{8}$	
	$.35 \cdot 10^{1}$	$.35 \cdot 10^{7}$	
10^{-8}	$.51 \cdot 10^{1}$	$.51 \cdot 10^{9}$	2754
	$.41 \cdot 10^{2}$	$.41 \cdot 10^{10}$	
	$.28 \cdot 10^{1}$	$.28 \cdot 10^{9}$	
10^{-10}	$.17 \cdot 10^{0}$	$.17 \cdot 10^{10}$	6914
	$.62 \cdot 10^{-1}$	$.62 \cdot 10^{9}$	
	$.14 \cdot 10^{-1}$	$.14 \cdot 10^{9}$	

(b) Lorenz Problem

Table 4: Check4 Results for ode45

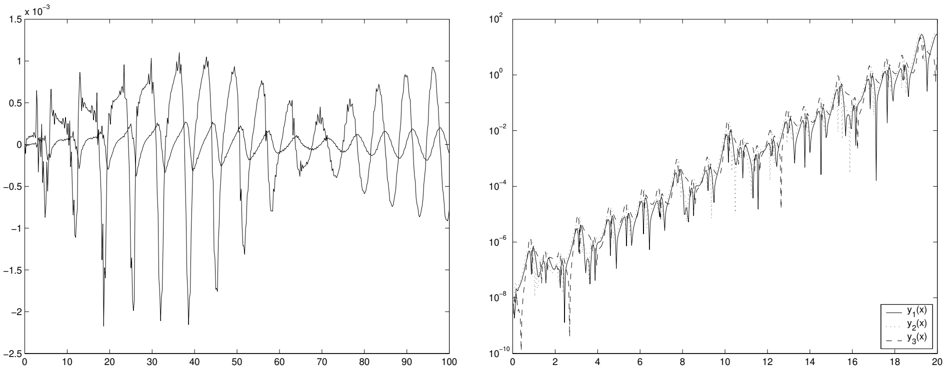
5 Discussion and future extensions

We have implemented these verification tools as standard MATLAB functions and demonstrated that they can be used in an effective way to quantify the quality of an approximate solution. These tools could have options which would be specified in a straightforward way that is identical to (or at least consistent with) the way options are set when using the ODE methods of MATLAB. Possible options include, the number of sample points used to 'visualize' or display the function associated with each verification tool (we have used a default value of 500); the reference alternative method to use in Check2 (we have used the method ode113 as the default); and the stringent value to use for $TOL1$ in Check2 (we have used $TOL/100$ as the default value). Such options could be changed from their respective default values in the same way that the default



(a) Predator-Prey Problem, $TOL = 10^{-3}$ (b) Lorenz Problem, $TOL = 10^{-6}$

Figure 3: $E_{TOL}(x)$ for ode45

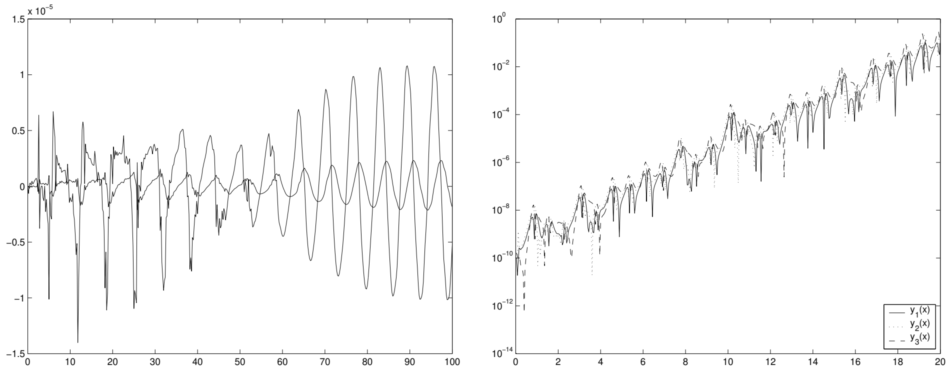


(a) Predator-Prey Problem, $TOL = 10^{-5}$ (b) Lorenz Problem, $TOL = 10^{-8}$

Figure 4: $E_{TOL}(x)$ for ode45

options are now changed for the different ODE methods of MATLAB.

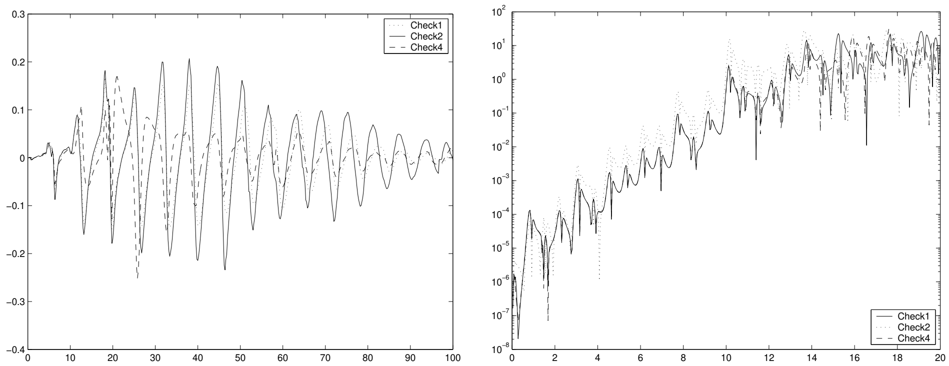
We are currently developing tools, similar to those introduced here, for use on other problem classes in the MATLAB PSE. In particular we are experimenting with preliminary versions of similar tools for use with boundary value problems and delay problems. We are also considering the design of such tool for classes of 2D and 3D partial differential equations (PDEs). For PDEs the costs involved in generating and visualizing the approximate solution are much greater than that for ODEs and the complexity and cost of each verification tool will also be greater.



(a) Predator-Prey Problem, $TOL = 10^{-7}$

(b) Lorenz Problem, $TOL = 10^{-10}$

Figure 5: $E_{TOL}(x)$ for ode45



(a) $y_2(x)$ of Predator-Prey Problem, $TOL = 10^{-3}$

(b) $y_3(x)$ of Lorenz Problem, $TOL = 10^{-6}$

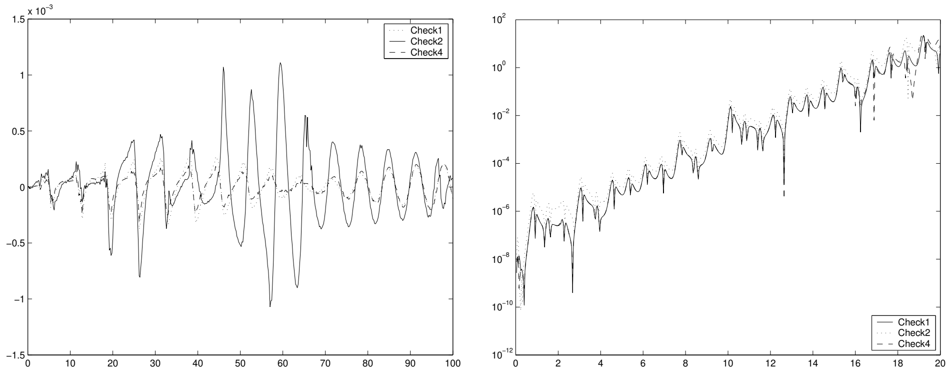
Figure 6: Check1, Check2 and Check4 Results for ode45

References

[1] W.H. Enright, K.R. Jackson, S.P. Nørsett and P.G. Thomsen, Interpolants for Runge-Kutta formulas, ACM Trans. Math. Soft., 12, 1986, pp.193-218.

[2] D.J. Higham and N.J. Higham, *Matlab Guide*, SIAM, 2000.

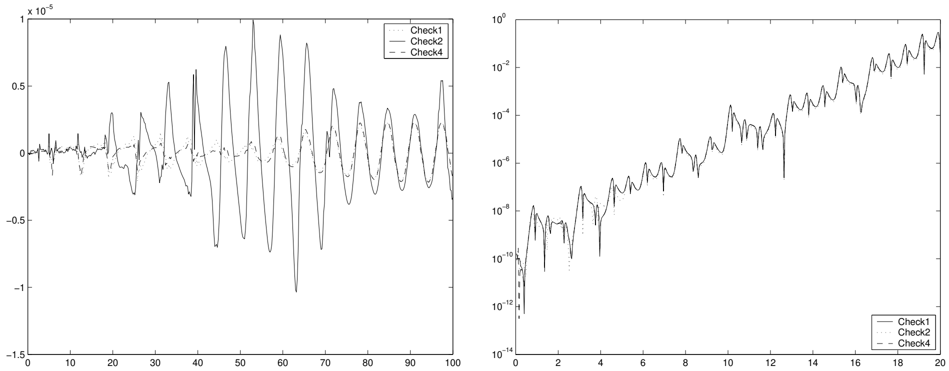
[3] The MathWorks, Inc., MATLAB 6.0, Natick MA, 2000.



(a) $y_2(x)$ of Predator-Prey Problem, $TOL = 10^{-5}$

(b) $y_3(x)$ of Lorenz Problem, $TOL = 10^{-8}$

Figure 7: Check1, Check2 and Check4 Results for ode45



(a) $y_2(x)$ of Predator-Prey Problem, $TOL = 10^{-7}$

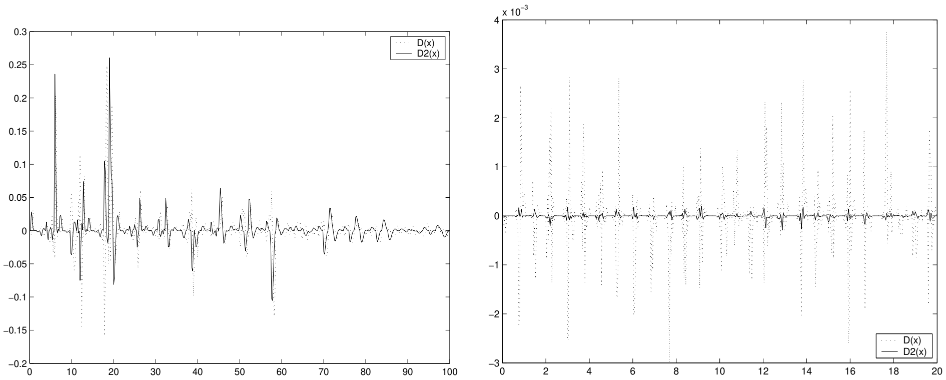
(b) $y_3(x)$ of Lorenz Problem, $TOL = 10^{-10}$

Figure 8: Check1, Check2 and Check4 Results for ode45

[4] L.F. Shampine, I. Gladwell and S. Thompson, *Solving ODEs in in the MATLAB Problem Solving Environment*, To Appear, 2002.

[5] L.F. Shampine and M.W. Reichelt. The MATLAB ODE suite, SIAM J. Sci. Comp., 18, 1997, pp.1-22.

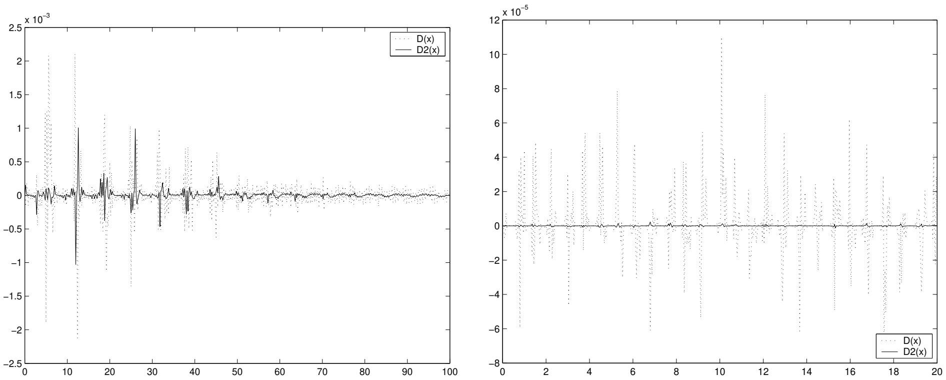
[6] P.E. Zadunaisky, On the estimation of errors propagated in the numerical integration of ordinary differential equations, Numer. Math., 27, 1976, pp. 21-39.



(a) $y_2(x)$ of Predator-Prey Problem, $TOL = 10^{-3}$

(b) $y_3(x)$ of Lorenz Problem, $TOL = 10^{-6}$

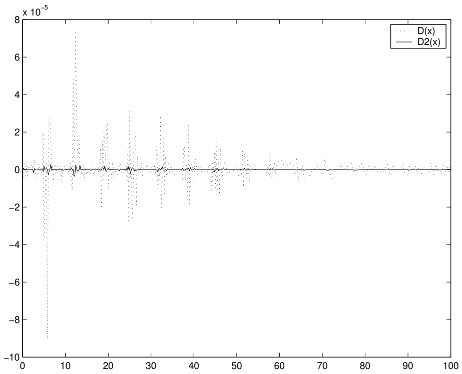
Figure 9: Check3 Results for ode45



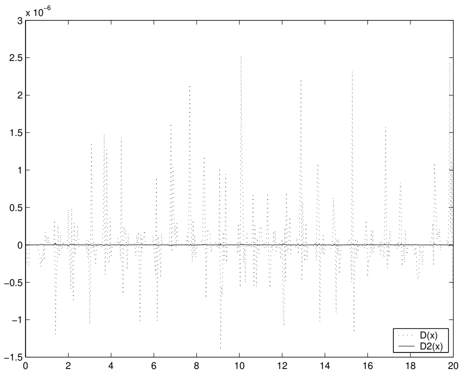
(a) $y_2(x)$ of Predator-Prey Problem, $TOL = 10^{-5}$

(b) $y_3(x)$ of Lorenz Problem, $TOL = 10^{-8}$

Figure 10: Check3 Results for ode45



(a) $y_2(x)$ of Predator-Prey Problem, $TOL = 10^{-7}$



(b) $y_3(x)$ of Lorenz Problem, $TOL = 10^{-10}$

Figure 11: Check3 Results for ode45