The Design and Implementation of Usable ODE Software *

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

Abstract

In the last decade it has become standard for students and researchers to be introduced to state-of-the-art numerical software through a problem solving environment (PSE) rather than through the use of scientific libraries callable from a high level language such as Fortran or C. In this paper we will identify the constraints and implications that this imposes on the ODE software we investigate and develop. In particular the way a 'numerical solution' is displayed and viewed by a user dictates that new measures of performance and quality must be adopted. We will use the MATLAB environment and ODE software for initial value problems, boundary value problems and delay problems to illustrate the issues that arise and the progress that has been made.

One of the major implications is the expectation that accurate approximations at 'off-mesh' points must be provided. Traditional numerical methods for ODEs have produced approximations to the underlying solution on an associated discrete, adaptively chosen mesh. In recent years it has become common for the ODE software to also deliver approximations at off-mesh values of the independent variable. Such a feature can be extremely valuable in applications and leads to new measures of quality and performance which are more meaningful to users and more consistently interpreted and implemented in contemporary ODE software. Numerical examples of the robust and reliable behaviour of such software will be presented and the cost/reliability trade-offs that arise will be quantified.

1 Introduction

In the last decade there has been a significant change in the way scientific computing is carried out. Developers of ODE software must be aware of this

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

change if their software is is ever going to be widely used and have an impact. Two environments or areas where ODE software is now widely used are:

- Large Scale Scientific Computation: Characterized by simulations generating massive amounts of data and involving expensive runs on High Performance distributed systems (often located at remote sites). Typical applications often involve data compression, interactive data viewing (or data mining), remote visualization, and the distributed collaboration of experts.
- Problem Solving Environments (PSEs): Often where students are first exposed to scientific computing and where scientists and engineers formulate and solve their problems.

In this paper we will focus on the latter environment but most of our observations and discussion also apply to the former environment.

In a typical PSE the focus is on finding approximate solutions to mathematical models and visualizing these approximate solutions in a realistic and non-distracting fashion. If the underlying mathematical model is a system of ODEs then it is essential that the ODE software produce an approximate solution that can be readily rendered or visualized. In such an environment stringent accuracy is not often required although it should be available as an option.

Because a PSE serves a large community of users, many of whom have little knowledge of numerical software, it is also very important to adopt a user interface (to the ODE software) and documentation that is hierarchichal and structured. Information that is not necessary to define a basic problem should be hidden from the user (and not even discussed at the top level of the documentation). Of course for more difficult problems or more sophisticated applications additional information may be necessary for the software to be effective, but the related discussion and the optional interface to specify the 'additional information' need only be read and understood by those who must supply the information.

1.1 A Typical Application

To appreciate some of the issues that arise in a PSE and the implications it has on the numerical software that can be used consider a typical application where a predator-prey relationship is modeled by the IVP:

$$y_1' = y_1 - 0.1 y_1 y_2 + 0.02t$$

$$y_2' = -y_2 + 0.02 y_1 y_2 + 0.008t$$

with

$$y_1(0) = 30, \quad y_2(0) = 20,$$

where $y_1(t)$ represents the 'prey' population at time t and $y_2(t)$ represents the 'predator' population at time t. The solution can then be visualized as a y/t

solution plot or by a 'phase plane' plot. Consider the use of a standard ODE software package such as one based on an 8th order Runge-Kutta formula to generate the approximate solution.

A scientist may be interested in whether the solutions to this equation are 'almost periodic' (in the sense that the difference between successive local maximums of the populations is constant) and whether the local maximums approach a steady state exponentially (see Figure 1). If only a discrete solution was available then the visualization could only use low-order interpolation such as piecewise linear (Figure 2) or splines (Figure 3) to obtain the off-mesh values required to test this hypothesis. (Note that to produce the results displayed here the 8th order Runge-Kutta method required only 34 steps to determine the approximate solution on the interval $[0, 40]$.) In both of these cases one can observe distracting artifacts caused by the inconsistent off-mesh accuracy.

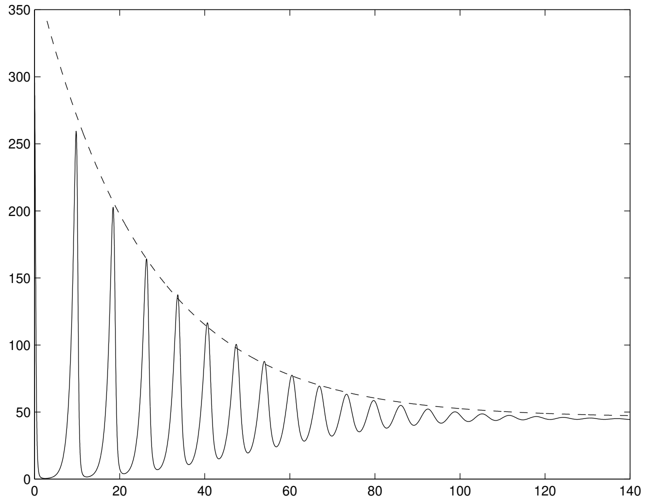
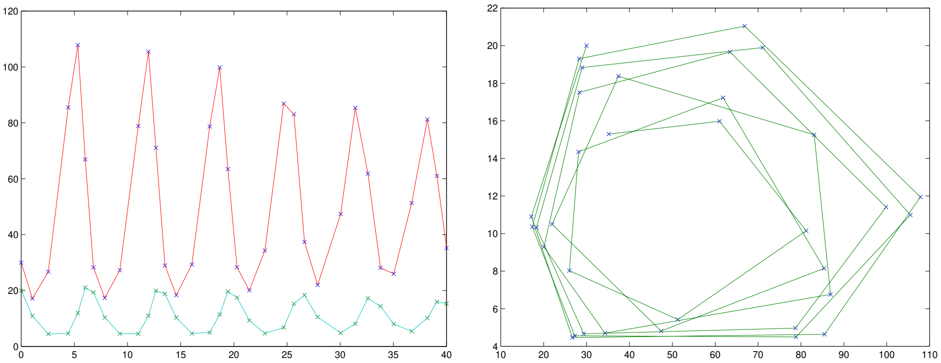


Figure 1: Prey population and exponential decay of local max

The special interpolant used to generate the visualizations presented in Figure 4 are associated with an underlying continuous Runge-Kutta (CRK) formula that has been derived for this particular 8th order Runge-Kutta formula. Note that, in particular, the off-mesh accuracy of this latter piecewise polynomial leads to a much more realistic phase plane portrait and can be reliably used to test the hypothesis under investigation.

1.2 Implications for ODE Software

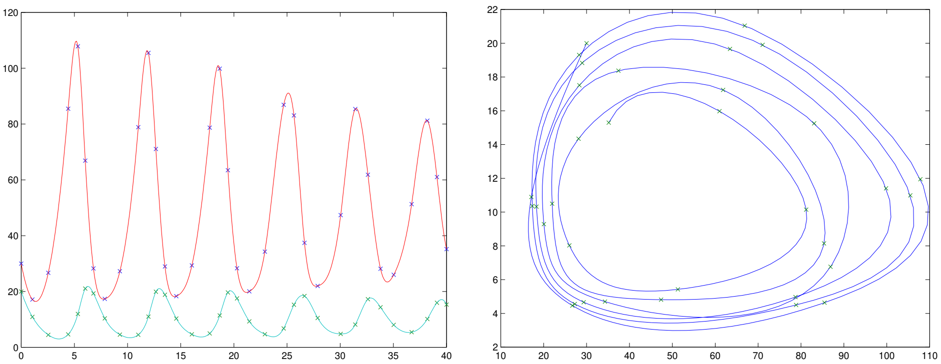
It is also particularly important, in a PSE, that different software packages (designed for the same or similar tasks) that determine an approximate solution, do so in a consistent way. That is, such software should:



(a) Solution Plot

(b) Phase Plane Plot

Figure 2: Visualization Using Piecewise Linear and 8th order Method.

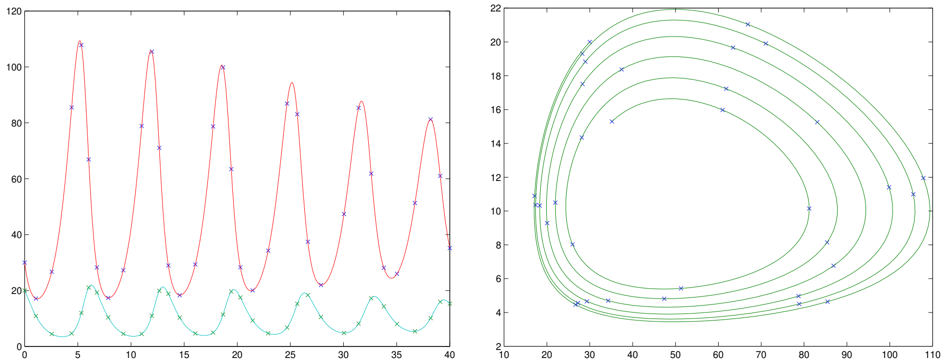


(a) Solution Plot

(b) Phase Plane Plot

Figure 3: Visualization Using Splines and 8th order Method.

- Adopt a standard interpretation of error control, and a standard program interface (calling sequence).
- Require that options and additional parameters (when required) be specified in the same way. (This includes such things as mesh-constraints, initial guess of solution, specification of partial derivatives, and accuracy specification.)
- Adopt a standard representation of the approximate solution. (eg. a vector of piecewise polynomials.)



(a) Solution Plot (b) Phase Plane Plot

Figure 4: Visualization Using Special Interpolation for the 8th order Method.

There are two common approaches for delivering such software. The first is to 'wrap' existing high performance program libraries and use environment-specific 'Application Program Interfaces' or APIs. Examples of this are the mechanisms developed within MATLAB and MAPLE to invoke selected software from the NAG numerical library. (See [3] for more details and discussion of this approach.) The second approach involves the development of low-overhead, low-order, new methods completely within the PSE. (For example, consider the recent sequence of improvements to the ODE component of MATLAB: [16],[14] and a similar improvement in the ODE capability of MAPLE [15].)

There is a need for both approaches as they complement each other. The latter is ideal for rapid prototyping and numerical experiments while the former is often required for final production runs and large scale simulations.

In addition to the above implications that are motivated by considering 'ease of use' there are 'robustness' implications that are particularly relevant for ODE software. The stepsize and error control strategies must be effective for both coarse and fine meshes. That is, as is pointed out in [14], the strategies should be both asymptotically 'correct' and otherwise 'plausible'. It is also important to detect numerical difficulties as early as possible. (Examples of such difficulties include the lack of convergence of an iteration; a discontinuity in a derivative; and an error in the specification of analytic partial derivatives). When such difficulties are detected the action taken may involve the use of alternate strategies or a return with warning message/flag.

1.3 Defect Control for Continuous Methods

We will consider continuous methods which are designed to directly monitor and control the maximum magnitude of an estimate of the defect of a piecewise polynomial $u(t)$ that is delivered as the approximate solution to an ODE. We will focus on methods based on an order p CRK formula but most of the discussion and analysis will apply to other classes of continuous methods such as those based on multistep formulas or collocation formulas. As is discussed in detail in [7], these assumptions will make it possible to ensure that the resulting software is suitable for use in a PSE.

For a piecewise polynomial approximation $u(t)$ we define the defect, $\delta(t)$, for $t \in (t_0, t_F)$ by

$$\delta(t) = u'(t) - f(t, u(t)).$$

Note that our assumption that $u(t)$ be order p implies, for sufficiently smooth problems,

$$u'(t) = y'(t) + O(h^{p-1}).$$

Furthermore, since

$$\begin{aligned} \delta(t) &= u'(t) - f(t, u(t)) - y'(t) + f(t, y(t)) \\ &= (u'(t) - y'(t)) + (f(t, y(t)) - f(t, u(t))), \end{aligned}$$

for differential equations that are Lipschitz continuous, $||\delta(t)||$ will be at worst $O(h^{p-1})$.

In [7] we have shown that by adding a few extra stages (to an existing order p CRK) one can derive an improved order p local interpolant, $\bar{u}_i(t)$, (associated with the same underlying discrete formula) which has a defect $\bar{\delta}(t)$ that is $O(h^p)$ and a continuous local error that is $O(h^{p+1})$. With such a $\bar{u}_i(t)$, one can sample the defect at any point (or set of points) in the current step and this can be used to determine an estimate of its maximum magnitude. A stepsize and error control based on such estimates has been found to be reliable and effective for a wide class of practical problems and has been called 'relaxed defect control'.

We have also shown how one can add a small number of extra stages (to an existing order p CRK) to derive an improved order p local interpolant, $\tilde{u}_i(t)$, which has a defect that is $O(h^{p+1})$ and for which a one-point sampling of the defect provides an asymptotically correct estimate of its maximum magnitude. A stepsize and error control strategy based on such asymptotically correct estimates is called 'strict defect control'.

The results reported in Table 1 show the number of stages required for some typical order p CRK formulas (see [4] for the definition of these particular CRKs). In this table s is the number of stages required to determine the discrete approximate solution, $\bar{s}$ is the number of stages necessary to produce a CRK with an $O(h^p)$ defect and relaxed defect control, while $\tilde{s}$ is the number of stages necessary to produce a CRK with an $O(h^p)$ defect and strict defect control.

2 Suitable ODE Software

In [7] we consider in detail the advantages and cost/reliability trade-offs when implementing ODE software using defect control. In particular, it is an attractive alternative (to standard local error control) when off-mesh solution values

Formula	p	s	$\bar{s}$	$\tilde{s}$
CRK4	4	4	6	7
CRK5	5	6	9	11
CVSS6B	6	7	11	14
CVSS7	7	9	15	20
CVSS8	8	13	21	28

Table 1: Cost per step of some CRK formulas

must be provided and when a consistent accuracy/TOL relationship is desired (across a collection of different ODE software packages). In the PSE environment we are considering such methods can be directly adopted and 'defect control' becomes a criteria that leads naturally to a standard interpretation of error control and user interface (for all ODE software). We have implemented and tested several CRK formulas based on this approach for IVPs and reported (in [7] and the references cited there) on how well this software performs over a range of accuracy requests. In particular we have shown that if strict defect control is used when solving an IVP, one may observe a 10-20% increase in cost but the tight and consistent control of the maximum defect and the maximum global error is a significant advantage and the resulting increase in reliability and robustness often justifies the extra cost. (In the software we have developed for IVPs we adopt strict defect control as the default but, as an option, allow relaxed defect control to be used).

An additional benefit that can be realized when defect control is used in IVP software is that very effective generic schemes can be used for event detection and discontinuity detection/crossing (see [17], [12] for details).

2.1 BVP Software

Consider the two-point BVP:

$$y' = f(t, y), \quad t \in [t_0, t_F], \quad g(y(t_0), y(t_F)) = 0.$$

MIRKDC ([9]) and bvp4c ([14]) are two recently developed software packages that attempt to produce an approximate solution that satisfies:

$$u'(t) = f(t, u(t)) + \delta(t),$$

with

$$||\delta(t)|| \leq TOL,$$

and

$$g((u(t_0), u(t_F)) = 0.$$

Some advantages of such BVP software are:

	$O(h^{2k})$ local interp				$O(h^{2k+1})$ local interp			
$mmin =$	1	2	3	4	1	2	3	4
$k = 1$	1	-	-	-	1	-	-	-
$k = 2$	1	1	-	-	2	1	-	-
$k = 3$	3	1	1	-	6	2	1	-
$k = 4$	8	3	1	1	13	6	2	1
$k = 5$	16	8	3	1	23	13	6	2
$k = 6$	27	16	8	3	36	23	13	6
$k = 7$	41	27	16	8	52	36	23	13

Table 2: Extra stages required for accurate interpolants

- The cost of determining $u(t)$ is small relative to that of determining the discrete solution. This is due to the observation that generally $u_i(t)$ only needs to be determined after an acceptable converged discrete solution has been computed (this often involves the solution of a sequence of nonlinear systems associated with preliminary coarse meshes).
- The size of the defect can be used in the mesh refinement strategy (even before convergence has been achieved on an initial coarse mesh).
- $u(t)$ can be used to generate an initial guess for use with continuation.

Collocation software for BVPs (such as COLSYS/COLNEW [1],[2]) can be applied directly to high order systems:

$$y^{(m)} = f(t, y, y' \cdots y^{(m-1)}),$$

or to mixed order systems:

$$y_j^{(m_j)} = f_j(t, z(y)), \quad j = 1 \cdots d.$$

In recent work we ([10],[11]) and others ([13]) have developed more accurate local interpolants for use with COLSYS/COLNEW. In particular we have used the idea of explicit CRKs (applied to the equivalent IRK interpretation of the collocation formulas) to derive $O(h^{2k})$, piecewise polynomials, $u(t)$. The approach has been applied to mixed order systems where the resulting $u_j(t) \in C^{m_j}[t_0, t_F]$ and the off-mesh accuracy is optimal: $u^{(r)}(t) = y^{(r)}(t) + O(h^{2k-r})$, for $r = 0 \cdots m_j$. The associated continuous local error of these new interpolants can be of order $2k$ or $2k+1$. The cost and accuracy of these new interpolants when solving mixed order problems using COLNEW is summarized in Table 2 (where mmin is the minimum of $m_1, m_2 \cdots m_d$).

Detailed numerical results showing the quality of these interpolants are presented in [11]. These results show that these interpolants provide, at very little extra cost, more accurate approximations even at coarse meshes. The next step is to investigate how the internal strategies/heuristics of COLNEW can be modified to make direct use of these interpolants for such tasks as mesh selection and error control.

2.2 DDE Software

A class of numerical methods based on CRK formulas with defect control has been developed for systems of retarded and neutral DDEs of the form,

$$
\begin{aligned}
y' &= f(t, y, y(t-\sigma_1(t,y)), y'(t-\sigma_2(t,y))), \\
y(t) &= \phi(t), \;\; t \leq t_0,
\end{aligned}
$$

where $\sigma_1(t,y)$ and $\sigma_2(t,y)$ are positive scalar functions. One particular formula from this class, CVSS6B, has been implemented in DDVERK [8]. This software package is designed to effectively handle problems with multiple delays, vanishing delays and frequent discontinuities.

Several complications can arise when solving DDEs which can lead to numerical difficulties which are not handled well by current DDE software. Examples include:

- Efficiently solving the implicit equations that inevitably arise when the delay is small (relative to the stepsize).
- Choosing the right value to use to approximate $y^{'}(t-\sigma)$ or $y(t-\sigma)$ when a discontinuity in the derivative exists at t_{i-j} and $t-\sigma$ is very close to t_{i-j}. (On problems with state-dependent delays this can cause poor performance as $t-\sigma$ may not be accurate enough.)
- Detecting when, on problems with multiple delays, different components may be signaling the same discontinuity. (It is important to avoid introducing 'extra' artificial discontinuities into the approximate solution.)

To address these difficulties we must either develop better heuristics and/or strategies or restrict the type of problems that our software can solve. For example, in the recently introduced DDE solver dde23 of MATLAB [18], only constant delays are allowed. This restriction makes it possible to readily resolve these difficulties in a robust and effective way.

3 Other Related Developments

An ideal PSE provides tools that make it easy to:

1. Formulate a mathematical model of the problem.
2. Approximate the solution of the mathematical problem.
3. Visualize the approximate solution.
4. Validate (or verify) an approximate solution.
5. Validate (or verify) the mathematical model.

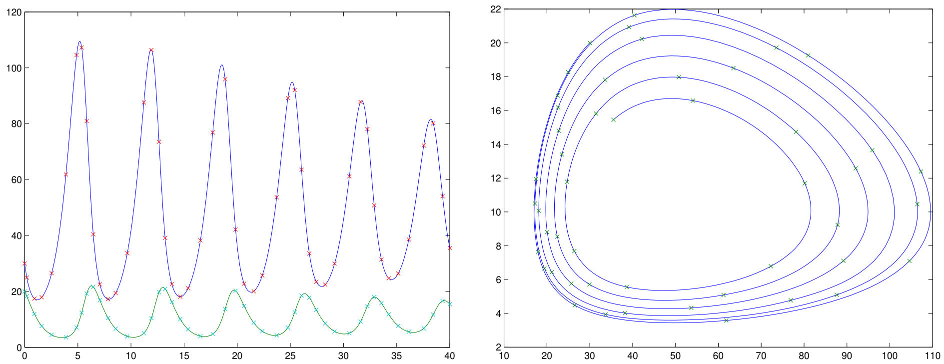
So far we have focused on software for the second of these tasks but the fourth task is closely related and is important as well.

3.1 Validation of Approximate solutions of ODEs

If the underlying software provides a continuous approximation to the solution (over the domain of interest) generic tools for validating or verifying an approximate solution can be developed for the PSE. Examples are tools or packages which would:

- Generate guaranteed error bounds using an (expensive) validated ODE solver.
- Solve the same ODE with an unrelated method or a more stringent error tolerance and check the difference between the two either graphically or by sampling over a selection of predetermined (or random) points.
- Sample the defect (or residual) over a selection of predetermined (or random) points.
- Estimate the global error by integrating a 'nearby' problem where the exact solution is known (using the fact that $u(t)$ satisfies the perturbed ODE, $z^{'} = f(t, z) + \delta(t)$.)

As an example of a validity check on the approximate solution determined by the 8th order CRK (visualized in Figure 4) we could solve the problem using the builtin MATLAB routine ode45 (based on a continuous 5th order CRK) and obtain the results visualized in Figure 5. Note that as far as these visualizations are concerned the approximate solutions are indistinguishable (from those of Figure 4) and this would serve as a validity check of either approximate solution.



(a) Solution Plot

(b) Phase Plane Plot

Figure 5: Visualization Using ode45 and its associated Interpolant

3.2 Extension to higher dimensions (PDEs)

In a recent sequence of investigations (see [5], [6]) we have shown that the approach of extending a discrete approximate solution to a 'continuous' approximate solution using piecewise polynomial to deliver off-mesh accuracy can be effectively applied to PDE problems. In particular generic multivariate interpolation/pseudo-collocation can be used to make visualization and data storage/transmission efficient and convenient. This is particularly relevant in large scale simulations where large volumes of data must be manipulated and displayed in a meaningful and efficient manner. The key observation is that one can represent the approximate solution to a PDE (a multivariate piecewise polynomial) by coarse mesh, accurate, discrete meshpoint information only. At a relatively small cost the polynomial associated with any mesh element can be computed (from local mesh information only) and used locally for visualization and or other data mining or data analysis application.

As an example of the potential of this approach consider the approximate solution of the Wave Equation in 2D (the vibrating membrane problem):

$$u_{tt} - .25(u_{xx} + u_{yy}) = 0,$$

on the domain: $0 \leq t \leq 2$, $0 \leq x \leq 2$, $0 \leq y \leq 2$, with boundary conditions: $u(t, x, y) = 0$, and initial conditions:

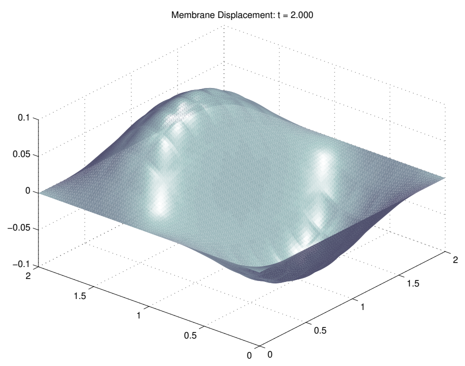
$$u(0, x, y) = 0.1 sin(\pi x) sin(\pi y/2), \;\; u_t(0, x, y) = 0.$$

To visualize the solution to this type of problem it is now common to use animation and produce a 'movie' showing the approximate solution as it passes through one complete cycle or oscillation. In this example one must use at least 100 images to produce a 'smooth' animation covering the range $0 \leq t \leq 2$. Each image or 'snapshot' corresponds to the state of the system at a prescribed point in time and can be represented by a surface plot or a contour plot.

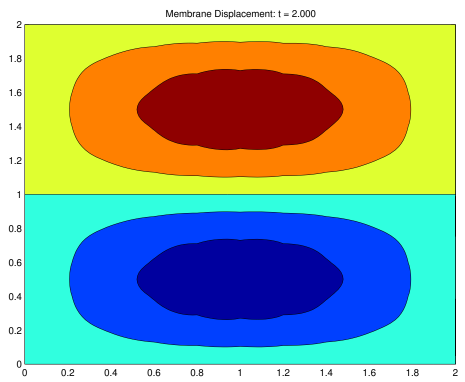
We have investigated this problem in MATLAB and found that, to generate a realistic (non-distracting) image, the renderer required the approximate solution values on a fine grid of at least 100×100 points. The complete animation would then require that the approximate solution be evaluated on a fine mesh of $100 \times 100 \times 100$ points. This would comprise a large amount of data and require an expensive computation if only a discrete solution could be obtained from the PDE software package that is to be used. Our approach would use the PDE package to generate an accurate discrete solution on a coarse mesh and then use the low-cost tri-cubic piecewise interpolant to generate the 'fine-mesh' data values required for visualization.

Figures 6–7 show a single snapshot (corresponding to $t = 2.0$ in the animation) and two choices for the coarse mesh. In both cases the renderer is supplied with the approximate solution evaluated on the fine-mesh (100×100) partitioning of the domain. In Figure 6 a $10 \times 10 \times 10$ coarse mesh is used (corresponding to a data compression ratio of 1000:1). The corresponding image is acceptable but artifacts of the underlying coarse grid could be distracting. In Figure 7 a

$20 \times 20 \times 20$ coarse mesh is used (corresponding to a data compression ratio of 125:1) and there are no distracting artifacts.

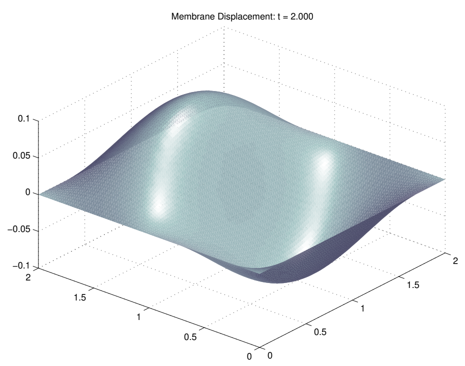


(a) Surface Plot

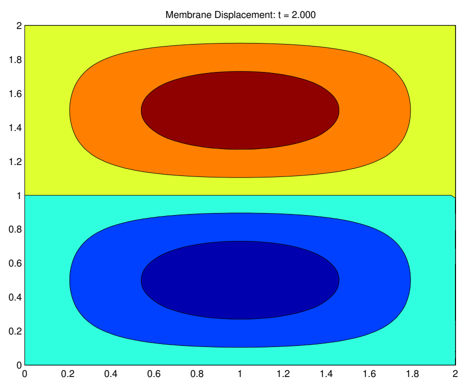


(b) Contour Plot

Figure 6: Snapshot generated using Tri-cubic Interpolant on $10 \times 10 \times 10$ coarse mesh corresponding to $t = 2.0$.



(a) Surface Plot



(b) Contour Plot

Figure 7: Snapshot generated using Tri-cubic Interpolant on $20 \times 20 \times 20$ coarse mesh corresponding to $t = 2.0$.

4 Summary and Conclusions

We have discussed how the expectations of what ODE software should deliver have changed over the last decade. These changes have been largely dictated by the way ODE software is being used by scientists, engineers and students who invoke numerical software and investigate the approximate solutions interactively from a familiar PSE. The software must provide a continuous approximation to the solution of the problem and it must satisfy additional non-traditional requirements.

In particular the software must be easy to use (especially for first time users) and this implies that all software for solving the same class of problems should share a common interface and software for solving similar problems (eg. BVPs and IVPs) should have a consistent interface.

We have also discussed how ODE software based on defect control is able to satisfy these expectations and we have identified specific packages that have been developed for IVPs, for BVPs and for DDEs. We expect that in the near future several existing ODE methods will be revised and 'repackaged' so they can be effectively used in PSEs.

References

[1] U. Ascher, J. Christiansen and R. Russell, Collocation software for boundary value ODEs, ACM Trans. Math. Soft., 7, 2, 1981, pp. 209-222.

[2] G. Bader and U. Ascher, A new basis implementation for a mixed order boundary value ODE solver, SIAM J. Sci. Comp., 8, 1987, pp. 483-500.

[3] R. Boisvert and P. Tang (eds), Software Architecture for Scientific Computing Applications, Proceedings of IFIP WG2.5, Working Conf. 8, Ottawa, October, 2000.

[4] W.H. Enright, The relative efficiency of alternative defect control schemes for high order Runge-Kutta formulas, SIAM J. Numer. Anal., 30, 5, 1993, pp. 1419-1445.

[5] W.H. Enright, Accurate approximate solution of PDEs at off-mesh points, ACM Trans. Math. Soft., 26, 2, 2000, pp. 274-292.

[6] W.H. Enright, Hierarchichal Representation and Computation of Approximate Solutions in Scientific Simulations, Proceedings of ..., to appear, 2001.

[7] W.H. Enright, Continuous numerical methods for ODEs with defect control, JACM, 125, 2001, pp. 159-170.

[8] W.H. Enright and H. Hayashi, A delay differential equation solver based on a continuous Runge-Kutta method with defect control, Numerical Algorithms, 16, 1997, pp. 349-364.

[9] W.H. Enright and P.H. Muir, A Runge-Kutta type boundary value ODE solver with defect control, SIAM J. Sci. Comp., 17, 1996, pp.479-497.

[10] W.H. Enright and P.H. Muir, Superconvergent interpolants for the collocation solution of BVODEs, SIAM J. Sci. Comp., 21, 1999, pp. 227-254.

[11] W.H. Enright and R. Sivasothinathan, Superconvergent interpolants for collocation methods applied to mixed order BVODEs, ACM Trans. Math. Soft., 26, 3, 2000, pp. 323-351.

[12] W.H. Enright, K.R. Jackson, S.P. Nørsett and P.G. Thomsen, Effective solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants, Appl. Math. Comput., 27, 1988, pp.313-335.

[13] H. Jin and S. Pruess, Uniformally superconvergent approximations for linear two-point boundary value problems, SIAM J. Numer. Anal., 35, 1, 1998, pp. 363-375.

[14] J. Kierzenka and L.F. Shampine, A BVP solver based on residual control and the MATLAB PSE, Report 99-1, Math. Dept., Southern Methodist University, 1999.

[15] L.F. Shampine and Robert M. Corless, Initial value problems for ODEs in problem solving environments, JACM, 125, 2001, pp. 31-40.

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

[17] L.F. Shampine and S. Thompson, Event location for ordinary differential equations, Comp. & Maths. with Appls., 39, 2000, pp. 43-54.

[18] L.F. Shampine and S. Thompson, Solving DDEs in MATLAB, Manuscript