Vectorized Solution of ODEs in MATLAB with Control of Residual and Error

L.F. Shampine
Mathematics Department
Southern Methodist University
Dallas, TX 75275
shampine@smu.edu

Abstract

Vectorization is very important to the efficiency of computation in the popular problem-solving environment MATLAB. Here we develop an explicit Runge–Kutta (7,8) pair of formulas that exploits vectorization. Conventional Runge–Kutta pairs control local error at the end of a step. The new method controls the extended local error at 8 points equally spaced in the span of a step. This is a byproduct of a control of the residual at these points. A new solver based on this pair, `odevr7`, not only has a very much stronger control of error than the recommended MATLAB solver `ode45`, but competes well at modest tolerances and is notably more efficient at stringent tolerances.

Keywords

MATLAB, vectorization, ordinary differential equations, initial value problems

1 Introduction

The problem-solving environment (PSE) MATLAB [7] is in very wide use. The costs of certain computations in a PSE are quite different from the costs in general scientific computation. In particular, it is possible to reduce run times in MATLAB very substantially by using the compiled functions that are built into the PSE and by vectorizing the computation so as to exploit fast array operations. Advice about how to vectorize code in MATLAB and pointers to other documents about efficient computation are available at [8]. Vectorization is so important to efficient computation in MATLAB that all the programs for approximating $\int_a^b f(x)\,dx$ *require* that $f(x)$ be coded to accept a vector $[x^1, x^2, \ldots, x^k]$ and return a vector $[f(x^1), f(x^2), \ldots, f(x^k)]$. That is because the cost of evaluating $f(x)$ generally depends weakly on the number of arguments when the

computation is vectorized. In [11] we considered how to exploit vectorization when solving numerically a system of n first-order ODEs

$$y' = f(t, y) \tag{1}$$

on an interval $[t_0, t_f]$ with initial value $y_0 = y(t_0)$. Following the advice of the programs provided by MATLAB for solving stiff initial value problems and boundary value problems, we assume that when the function for evaluating f is given a vector with entries $t^1, t^2, \ldots, t^k$ and a matrix with columns $y^1, y^2, \ldots, y^k$, it will return a matrix with columns $f(t^1, y^1), f(t^2, y^2), \ldots f(t^k, y^k)$. With careful coding of this function it is often the case that the cost of evaluating the function with array arguments is not much more than the cost of evaluating it with a single argument. A method was developed in [11] to exploit this kind of vectorization. The `BV78` solver of that paper competes well with the recommended MATLAB solver `ode45` at all tolerances and is considerably more efficient at stringent tolerances.

In the present investigation we develop a method and a solver `odevr7` that also competes well with `ode45` at all tolerances and is considerably more efficient at stringent tolerances. The new solver has a very strong control of error. Conventional solvers like `ode45` and `BV78` are based on a pair of formulas. They control the size of an estimate of the local error of the lower order formula at the end of a step. They advance the integration with the higher order formula, so the actual error is controlled only when the expected asymptotic behavior is evident. The new solver controls the error in the formula used to advance the integration. It controls the size of a residual at 8 points equally spaced in the span of each step. This implies a control of estimates of the extended local error at the 8 points. There is a price to pay for this strong control, but it is a modest one because we have found an analog of the First Same as Last (FSAL) technique for reducing the cost of a successful step. Also, the basic formula is slightly more accurate than the corresponding formula of the BV78 pair. We prove the surprising result that the new pair of formulas has *exactly* the same stability regions as the BV78 pair. Our goal was to develop a solver that has an exceptionally strong error control and is still competitive with a good conventional solver like `ode45`. Certainly we have achieved that with `odevr7` provided that our assumptions about vectorization are valid.

2 Block RK Methods

The explicit block one-step methods suggested by Milne [9] are based on implicit Runge-Kutta (RK) formulas that in the course of a step from t_n to $t_n+h = t_{n+1}$, form an accurate approximation not only at t_{n+1}, but also points equally spaced in the span of the step. The implicit formulas are derived in [11] by collocation: A polynomial $P(t)$ with $P(t_n) = y_n$ is to collocate at equally spaced points $t_{n,j} = t_n + jh/r$ for $j = 0, 1, \ldots, r$. With the notation $y_{n,j} = P(t_{n,j}), f_{n,j} =$

$f(t_{n,j}, y_{n,j})$ and the definition $y_{n,0} = y_n$, the resulting formulas have the form

$$\begin{aligned}
y_{n,1} &= (y_{n,0} + h\,A_{1,0}\,f_{n,0}) + h\,[A_{1,1}\,f_{n,1} + \ldots + A_{1,r}\,f_{n,r}] \\
y_{n,2} &= (y_{n,0} + h\,A_{2,0}\,f_{n,0}) + h\,[A_{2,1}\,f_{n,1} + \ldots + A_{2,r}\,f_{n,r}] \\
\vdots &= \vdots \\
y_{n,r} &= (y_{n,0} + h\,A_{r,0}\,f_{n,0}) + h\,[A_{r,1}\,f_{n,1} + \ldots + A_{r,r}\,f_{n,r}]
\end{aligned}$$

It is shown in [17] that the $y_{n,j}$ are all of local order $r+2$. In particular, this is true of the approximation $y_{n,r} = y_{n+1}$ used to advance the integration, so the implicit RK method is of global order $r+1$. It is also shown that if r is even, $y_{n,r}$ has a higher local order than at intermediate points, namely $r+3$, so the method is of global order $r+2$.

Following Rosser [10], we investigated explicit methods in [11] that are formed by starting with the locally second order approximations $y^{[1]}_{n,j} = y_{n,0} + (jh/r)\,f_{n,0}$ for $j = 1, \ldots, r$ and then making a *fixed* number of simple iterations in the implicit formulas. An iteration begins by evaluating the $f^{[m]}_{n,j} = f(t_{n,j}, y^{[m]}_{n,j})$ and then computing the $y^{[m+1]}_{n,j}$ from

$$\begin{aligned}
y^{[m+1]}_{n,1} &= (y_{n,0} + h\,A_{1,0}\,f_{n,0}) + h\left[A_{1,1}\,f^{[m]}_{n,1} + \ldots + A_{1,r}\,f^{[m]}_{n,r}\right] \\
y^{[m+1]}_{n,2} &= (y_{n,0} + h\,A_{2,0}\,f_{n,0}) + h\left[A_{2,1}\,f^{[m]}_{n,1} + \ldots + A_{2,r}\,f^{[m]}_{n,r}\right] \\
\vdots &= \vdots \\
y^{[m+1]}_{n,r} &= (y_{n,0} + h\,A_{r,0}\,f_{n,0}) + h\left[A_{r,1}\,f^{[m]}_{n,1} + \ldots + A_{r,r}\,f^{[m]}_{n,r}\right]
\end{aligned}$$

Each iteration raises the local order of the approximations $y^{[m]}_{n,j}$ by one (up to a maximum order determined by the order of the underlying quadrature formulas). For the BV78 pair of [11] we took $r = 6$, so after 6 iterations, all the approximations have local order 8. When another iteration is done, the local order remains 8 at all the interior points, but the local error at the end of the step is increased to 9, resulting in a formula of global order 8. Along with the previous iterate, this results in an explicit (7,8) pair that we called BV78. As this pair is implemented in the solver `BV78`, the function $f(t, y)$ is evaluated with a single argument to obtain $f_{n,0}$. Each iteration requires evaluation of f at 6 arguments, which can be accomplished with a single array evaluation. Seven iterations are needed to reach the desired order, so each step costs 8 (array) evaluations. This is much better than the 13 evaluations of conventional (7,8) pairs. An attractive aspect of this kind of formula is that the values $y^{[m+1]}_{n,j}$ are just $P^{[m]}(t_{n,j})$ for the polynomial $P^{[m]}(t)$ that interpolates y_0 and $f^{[m]}_{n,j}$ for $j = 0, 1 \ldots, r$. Accordingly, when we reach the m that provides the desired order, we already have available a continuous extension $P^{[m]}(t)$ for "free". This contrasts with the excellent conventional (7,8) pair of Verner [16] that uses three additional evaluations to form a continuous extension of the lower order formula.

In this paper we investigate the (7,8) pair in this family that has $r = 7$. The explicit formulas are evaluated in exactly the same number of array evaluations as the BV78 pair. It is plausible that the extra collocation point would provide a more accurate formula. Indeed, a result in [17] shows that the underlying implicit formula for y_{n+1} with $r = 7$ has a significantly smaller truncation error than the formula with $r = 6$. Unfortunately, this does not appear to be the case for the corresponding explicit formulas. To investigate the matter we substituted the pair with $r = 7$ into `BV78` and compared the resulting code to `BV78` precisely as we compared `BV78` to `ode45` in [11]. The two pairs behaved almost the same, though there were some problems for which the new pair was a little more efficient.

The implicit formulas underlying both pairs are shown in [17] to be A-stable, but, of course, the explicit formulas have finite stability regions. The stability regions for the two formulas of BV78 appear as Figure 1 in [11]. When the author computed the corresponding stability regions for the new (7,8) pair, he was surprised to find that the regions were the *same* as those of the BV78 pair! This is an interesting consequence of the form of the formulas. The stability region is obtained from the stability polynomial, which results from applying the formula to the test equation $y' = \lambda y$. Like Rosser [10], we use as initial approximation the same polynomial of degree 1 for each choice of r. Each iterate constructs a new polynomial by applying a set of quadrature formulas to the current polynomial. As long as the degree of this polynomial is no greater than the degree of precision of the quadrature formulas, the polynomial is independent of r. In the present circumstances the stability polynomials are the same, so the new explicit (7,8) pair has exactly the same stability regions as those of BV78.

The (7,8) pair with $r = 7$ costs the same number of array evaluations as the BV78 pair with $r = 6$, it has the same stability, and is only a little more efficient. Nevertheless, it has some important advantages. In this section we show how to reduce the cost of a typical step. This improves not only efficiency, but also stability. In the next section we show how to achieve a very much stronger control of error.

The (4,5) Runge–Kutta pair derived by Dormand and Prince [1] that is implemented in the recommended MATLAB ODE solver `ode45` is FSAL (First Same As Last). A pair of this kind first forms the result y_{n+1} that will be used to advance the integration. The function $f(t, y)$ is then evaluated at (t_{n+1}, y_{n+1}), and possibly other arguments. The values are used to compute the other formula in the pair and then the local error of the lower order formula is estimated by comparison. If the step is a success, and most are, the value $f(t_{n+1}, y_{n+1}) = f_{n+1,0}$ is the first value needed in the next step and so is "free" in that step. A pair might be derived so that the integration is advanced with either the lower or the higher order formula. Generally the higher order formula is used, which is known as local extrapolation, because if the error estimate is valid, the higher order formula is the more accurate. However, the BDFs that are so popular for solving stiff problems are not implemented with local extrapolation because the companion formulas have unsatisfactory stability. The Dormand/Prince pair is

typical of popular explicit Runge–Kutta pairs in that the higher order formula is actually more stable, so there is quite a good case for local extrapolation. The same is true of the BV78 pair, so we do local extrapolation in the `BV78` solver. It struck the author that something analogous to FSAL was possible with the explicit block RK formulas if local extrapolation is *not* done. To be specific, we discuss this for the (7,8) pair. After forming the results $y_{n,j}$ of global order 7, we form all the $f(t_{n,j}, y_{n,j})$ in a single array evaluation to construct the result of global order 8 used for error estimation. If the step is a success, we advance the integration with $y_{n+1} = y_{n,r}$, so we have available the value $f(t_{n+1}, y_{n+1})$ that we need to start the next step. Just as with an FSAL pair like DOPRI5, this reduces the cost of a successful step by one evaluation.

We had hoped that increasing r would provide more stable explicit formulas. Although that did not prove to be the case, it is shown in [11] that the stability of the BV78 pair is quite satisfactory. Indeed, the stability regions of the DOPRI5 pair used in `ode45`, as seen in Figure 7.4 of [1], are both uniformly smaller than the region for the block RK formula of order 7. The DOPRI5 pair is FSAL and costs only 6 evaluations per successful step. As we have just seen, something similar is true of the new (7,8) pair so that a successful step costs only 7 evaluations per step. Comparing the regions and the cost per step, it appears that the stability of the new `odevr7` is comparable to `ode45`. It is stated in [11] that the average radius of the stability region for the formula of order 8 in the BV78 pair is about 4.66 and the average radius of the region for the formula of order 7 is about 4.26. Although the `BV78` program advances the integration with the more stable formula of order 8, this costs 8 array evaluations per step. Each successful step in `odevr7` using the formula of order 7 costs only 7 evaluations. Taking into account the sizes of the stability regions and the costs per step, `odevr7` has a small theoretical advantage. In [11] we compared numerically the efficiency of `BV78` and `ode45` when applied to problem K7 of a test set of Krogh [6]. This problem appears in the test set to illuminate the behavior of a solver when stability is an issue. Figure 1 shows how much accuracy is achieved for a given number of function calls for all three solvers. In this no distinction is made between calling the function with one, six, or seven arguments. As we expected on theoretical grounds, the codes perform much the same, but `odevr7` is somewhat more efficient.

Our scheme for reducing the cost per step requires that we advance the integration with the lower order formula of the pair. This affects the behavior of the solver with respect to a change of tolerance. An error per step control (EPS) controls the local error to be no more than a tolerance τ in stepping from t_n to $t_n + h$ and an error per unit step control (EPUS) controls it to be no more than $h\tau$. The paper [12] works out the behavior of the global error of a one-step method with each of the four possibilities of error control and local extrapolation. An error per step control with local extrapolation results in a global error that is asymptotically proportional to the tolerance τ. This combination is found in `ode45` and `BV78`. The popular BDF codes use EPS and do not do local extrapolation. A one-step method of global order p implemented in this way has a global error proportional to $\tau^{p/(p+1)}$. Accordingly, the imple-

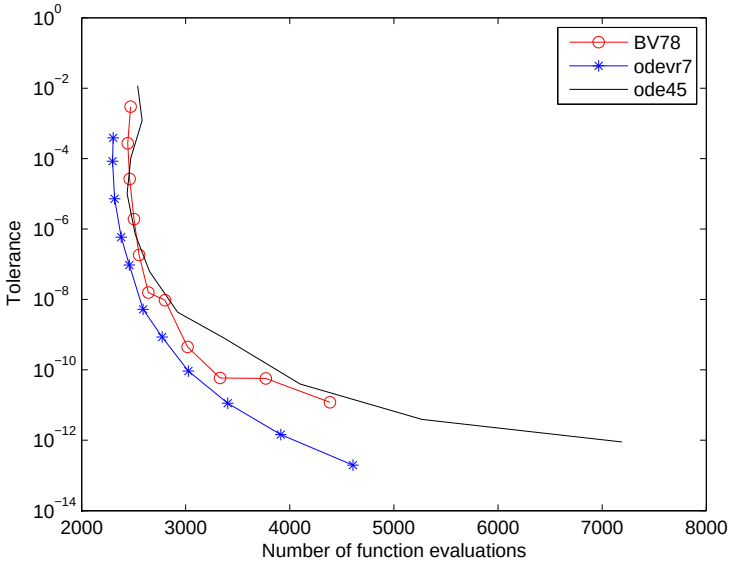

Figure 1: K7 tests stability along the negative real axis.

mentation of the new (7,8) pair in `odevr7` has a global error proportional to $\tau^{7/8}$. As with the popular BDF codes, at orders this high, the global error is sufficiently close to being proportional to the tolerance that users do not seem to be troubled by the fact that it is not quite proportional. The experiments reported in §4 consider the behavior of the global error as a tolerance is successively reduced by a factor of 10. At stringent tolerances, it can be seen in plots like Figure 2 that the reduction when using `odevr7` is not as close to being an order of magnitude as it is with `BV78`.

3 Error Control

In this section we exploit an important difference between the two (7,8) block RK formulas to control error in a much more robust way. In §2 we saw that for the BV78 pair, all the approximations $y_{n,j}$ corresponding to the lower order formula have local order 8, but for the higher order formula, only the approximation at the end of the step is of local order 9. For the new pair with block size $r = 7$, all the approximations $y_{n,j}$ corresponding to the higher order formula are of local order 9. Accordingly, we can compare these approximations to those of local error 8 to estimate the local error in *all* of the intermediate approximations, not merely the one used to advance the integration. This is quite remarkable. At each step we compute 7 new approximations evenly spread throughout the step and we can estimate the local error of each in exactly the same way that

the local error at the end of the step is ordinarily estimated. Not only do we get an extraordinarily robust assessment of the local error, but we also account for the fact that a solution might change substantially over the course of a step with formulas of order this high, especially formulas involving so many stages.

A more formal statement of this control of local error will be needed for subsequent developments. The local solution $u(t)$ through (t_n, y_n) is defined by

$$u'(t) = f(t, u(t)), \qquad u(t_n) = y_n \tag{2}$$

Typical methods approximate the local solution at the end of a step to $t_n + h$, but we consider here methods that also produce a polynomial $P(t)$ that approximates $u(t)$ throughout the span of the step. How well it does this is measured by

$$\max_{t_n \le t \le t_n + h} \|P(t) - u(t)\| = \|\|P - u\|\| \tag{3}$$

The usual local error is the quantity $P(t_n + h) - u(t_n + h)$, so at other t in the span of the step, the quantity $P(t) - u(t)$ is sometimes called the *extended local error*. When the meaning is clear, we shorten this to *local error*. The local errors of the approximate solutions $y_{n,j}$,

$$\max_{j=0,\ldots,r} \|P(t_{n,j}) - u(t_{n,j})\| = \|\|P - u\|\|_r \tag{4}$$

are of particular interest for the methods we study. For our investigation of the new formula implemented in `odevr7`, it will be convenient to write $y_{n,j} = y_{n,j}^{[6]}$ for the locally eighth order results that are returned as approximations to $y(t_{n,j})$ and $P(t)$ for the polynomial with $P(t_{n,j}) = y_{n,j}$, $j = 0, \ldots, 7$. We further write $y_{n,j}^* = y_{n,j}^{[7]}$ for the locally ninth order results that are used to estimate the local errors of the $y_{n,j}$. In this notation a conventional estimate of the local error of $y_{n,j}$ is

$$y_{n,j}^* - y_{n,j} = u(t_{n,j}) - P(t_{n,j}) + O(h^9) \qquad \text{for } j = 0, \ldots, 7$$

With this we have an estimate of the maximum local error at 8 points equally spaced in the interval of interest,

$$\max_{j=0,\ldots,r} \|y_{n,j}^* - y_{n,j}\| = \|\|P - u\|\|_r + O(h^9)$$

Rather than control this quantity directly, we have preferred in `odevr7` to control a scaled residual. In the rest of this section we study how the two kinds of controls are related.

The residual of a smooth approximate solution $Q(t)$ is

$$R(t) = Q'(t) - f(t, Q(t))$$

When we recognize that

$$f_{n,j}^{[6]} = f(t_{n,j}, y_{n,j}^{[6]}) = f(t_{n,j}, P(t_{n,j}))$$

and

$$f_{n,j}^{[5]} = \frac{dP^{[5]}}{dt}(t_{n,j}) = P'(t_{n,j})$$

we find that the residual of $P(t)$ at the nodes $t_{n,j}$ is

$$R(t_{n,j}) = P'(t_{n,j}) - f(t_{n,j}, P(t_{n,j})) = f_{n,j}^{[5]} - f_{n,j}^{[6]} \tag{5}$$

From the definition of the iterates we find that for given m,

$$\begin{aligned}
y_1^{[m+1]} - y_1^{[m]} &= h\left[A_{1,1}\left(f_1^{[m]} - f_1^{[m-1]}\right) + \ldots + A_{1,r}\left(f_r^{[m]} - f_r^{[m-1]}\right)\right] \\
y_2^{[m+1]} - y_2^{[m]} &= h\left[A_{2,1}\left(f_1^{[m]} - f_1^{[m-1]}\right) + \ldots + A_{2,r}\left(f_r^{[m]} - f_r^{[m-1]}\right)\right] \\
\vdots &= \vdots \\
y_r^{[m+1]} - y_r^{[m]} &= h\left[A_{r,1}\left(f_1^{[m]} - f_1^{[m-1]}\right) + \ldots + A_{r,r}\left(f_r^{[m]} - f_r^{[m-1]}\right)\right]
\end{aligned}$$

Using the new notation and (5) in these equations for $m = 6$ leads to

$$\begin{aligned}
y_1^* - y_1 &= -h\left[A_{1,1}\, R(t_1) + \ldots + A_{1,r}\, R(t_r)\right] \\
y_2^* - y_2 &= -h\left[A_{2,1}\, R(t_1) + \ldots + A_{2,r}\, R(t_r)\right] \\
\vdots &= \vdots \\
y_r^* - y_r &= -h\left[A_{r,1}\, R(t_1) + \ldots + A_{r,r}\, R(t_r)\right]
\end{aligned}$$

It is then straightforward to show that

$$\max_{j=0,\ldots,r} \|y_{n,j}^* - y_{n,j}\| \le h\, \|A\|_\infty\, |||R(t)|||_r \tag{6}$$

At each step the `odevr7` program controls the size of the scaled residual at the nodes of the step, which is to say that it controls $h\,|||R(t)|||_r$. For the new formula, $\|A\|_\infty \approx 0.96$, so this also controls the extended local error at 8 points evenly spread throughout $[t_n, t_n + h]$.

To better understand this error control, we derive a general result relating control of (extended) local error to control of a scaled residual. This aspect of the present investigation is closely related to the work of [14, 15]. Suppose now that $P(t)$ is an approximate solution that has the correct value at the beginning of the step and satisfies the ODEs with residual $R(t)$,

$$P'(t) = f(t, P(t)) + R(t), \qquad P(t_n) = y_n$$

Subtracting the equation (2) satisfied by the local solution $u(t)$ from this equation for $P(t)$, integrating the difference from t_n to t, and accounting for the values at the beginning of the step leads first to

$$P(t) - u(t) = \int_{t_n}^{t} [f(x, P(x)) - f(x, u(x))]\, dx + \int_{t_n}^{t} R(x)\, dx$$

and then to

$$\|P(t) - u(t)\| \leq \int_{t_n}^{t} \|f(x, P(x)) - f(x, u(x))\|\, dx + \int_{t_n}^{t} \|R(x)\|\, dx$$

As usual in the study of numerical methods for ODEs, we suppose that $f(t, y)$ satisfies a Lipschitz condition with constant L. It then follows easily that

$$|||P - u||| \leq hL\, |||P - u||| + h\, |||R|||$$

When solving non-stiff problems, it is generally assumed that hL is rather smaller than one. To be concrete, if we assume that $hL \leq 1/2$, then

$$|||P - u||| \leq \frac{h}{1 - hL}\, |||R||| \leq 2h\, |||R||| \tag{7}$$

With this modest assumption on the step size we find that a control of the scaled residual $h\, |||R|||$ provides a control of the extended local error. We favor a residual control because it is meaningful even when the asymptotic results about accuracy that justify estimates of local error are of dubious validity. In principle we can always compute a good estimate of the size of the residual because we can evaluate $R(t)$ wherever we like.

It is useful to regard the inequality (6) for explicit block RK formulas as a discrete analog of the general result (7), but we must recognize that there are some important differences. Obviously one takes into account only $r + 1$ points in the span of the step and the other, the whole interval $[t_n, t_n + h]$. The quantity on the left side of (6) is only an (asymptotically correct) *estimate* of the extended local error at the points of interest. Unfortunately, we have no theoretical results that say the discrete norms of (6) approximate well the continuous norms of (7). Certainly it is plausible that sampling the residual at 8 points equally spaced throughout the step would provide a good estimate of the maximum value of the residual, but we have not shown that for the scheme of `odevr7`. There are methods with continuous extensions for which there are asymptotic results that provide the locations of extrema of the residual, see for example [3, 13]. With this information an asymptotically correct estimate of the maximum residual can be readily computed. Alternatively, a quadrature formula is used to obtain an asymptotically correct estimate of an integral norm of the residual for the method of [5]. Enright and Li [2] discuss estimation of the size of the residual and in particular, how an improved estimate can lead to a better performance. Schemes with asymptotically correct estimates of the size of the residual are used in [14, 15] to provide a control not only of the size of a scaled residual, but also a control of the extended local error as in (7).

In `odevr7` we control the size of a scaled residual at 8 points equally spaced in the span of each step. We have shown that this bounds asymptotically correct estimates of the extended local error at those 8 points. It is conventional to control an estimate of the local error only at the end of a step. Also, it is conventional to do local extrapolation with the consequence that the error actually incurred is smaller than the quantity being controlled only if the expected

asymptotic behavior is evident. Certainly we have a far stronger control of error in `odevr7`. We have *not* proved that the 8 equally spaced samples provide an asymptotically correct estimate of the residual for the method of `odevr7`. Indeed, we do not believe that to be true. Still, with this many samples spread throughout the interval we think it reasonable to expect that `odevr7` will enjoy some of the robustness of a full control of the residual.

4 Illustrative Computations

In [11] we compared `BV78` and `ode45` using two standard sets of test problems [4, 6]. Here we compare these solvers to `odevr7` in the same way. Full details of the tests are reported in [11], so here we provide only an overview. We have compared the three solvers on both sets of test problems, but the results are consistent, so for brevity we report here only what happened with Krogh's set. The programs for both sets and `odevr7` itself are available at `http://faculty.smu.edu/shampine/current.html`. There are 10 problems in the set, but we must exclude K5 because it is a pair of second order ODEs and none of the solvers can treat such problems directly. For this reason Krogh includes in his set the problem K4, which is K5 written as a first order system. We display results for this problem in Table 1 under the heading "K4_5". The problems are to be solved with either pure relative or pure absolute error. The solvers were applied to each of the 9 problems of the test set with tolerances $10^{-2}, 10^{-3}, \ldots, 10^{-12}$. Tolerances were excluded from the comparison when one of the solvers had an error bigger than 1. Using reference values obtained as in [11], we compared the accuracies at 200 points equally spaced in the interval of integration. In this way we test not only the accuracy of the method, but also the accuracy of the continuous extension. Moreover, the cost of evaluating the continuous extension is included in the run times reported. As emphasized in [11], it is difficult to obtain consistent run times in MATLAB, especially when the run times are small in absolute terms, *as they are for these test problems.* To reduce the effects of this, we report run times summed over the whole range of tolerances. For each test problem, the entries of Table 1 were obtained in a single run of a program preceded by a "`clear all`" command. We have done this repeatedly and found the times to be consistent, though the last digit displayed might vary. Nevertheless, *the run times displayed in the table should be considered only a rough guide as to relative costs.* We summarize these results by comparing the total time required to solve this wide range of problems over a wide range of tolerances. As we found already in [11], the total run time of `ode45` is about 2.8 times that of `BV78` and we now see that it is about 2.7 times that of `odevr7`.

Each of our programs plots efficiency in terms of the accuracy achieved versus the number of array evaluations. The performance differs from problem to problem, but Figure 2 shows what might be described as a typical plot. The problem K9 is a two body problem in elliptic motion with eccentricity 0.6. `BV78` is somewhat more efficient than `odevr7`, especially at the most stringent

Table 1: Run times for test set of Krogh [6].

Problem	$K1$	$K2$	$K3$	$K4\_5$	$K6$	$K7$	$K8$	$K9$	$K10$
`BV78`	0.2	0.5	0.5	0.6	0.6	0.9	0.8	0.9	0.5
`odevr7`	0.2	0.5	0.5	0.6	0.6	1.0	0.9	1.0	0.5
`ode45`	0.5	1.8	1.7	2.0	1.9	1.2	2.1	2.7	1.7

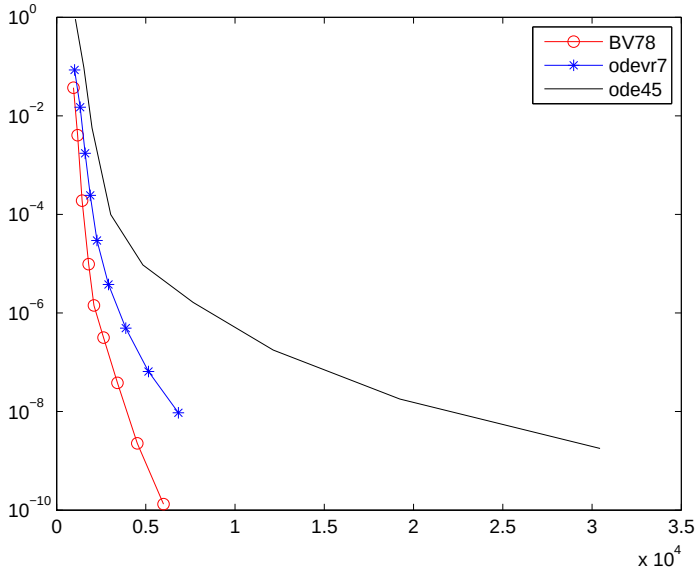

Figure 2: K9 is a two body problem with eccentricity 0.6.

tolerances where the fact that it integrates at order 8 is most important. Though `BV78` is typically more efficient in this sense for Krogh's test set, `odevr7` is comparably efficient for some problems and more efficient for a few. As we saw in Figure 1, `odevr7` is somewhat more efficient than `BV78` when stiffness is an issue. The performance of the solvers on the restricted three body problem K10 is interesting. Figure 3 shows that `odevr7` solves K10 a little more efficiently than `BV78`. The fact that it performs better at crude tolerances is not unusual, but the fact that it performs better at the most stringent tolerances is.

The new `odevr7` has a very much stronger control of error than `BV78`. It is less efficient than `BV78`, but it is competitive because each step is cheaper and the formulas are a little more accurate. That, however, is not the important conclusion from these tests and analysis: By virtue of vectorization, `odevr7` has a very much stronger control of error than `ode45`, but still competes well at modest tolerances and is notably more efficient at stringent tolerances.

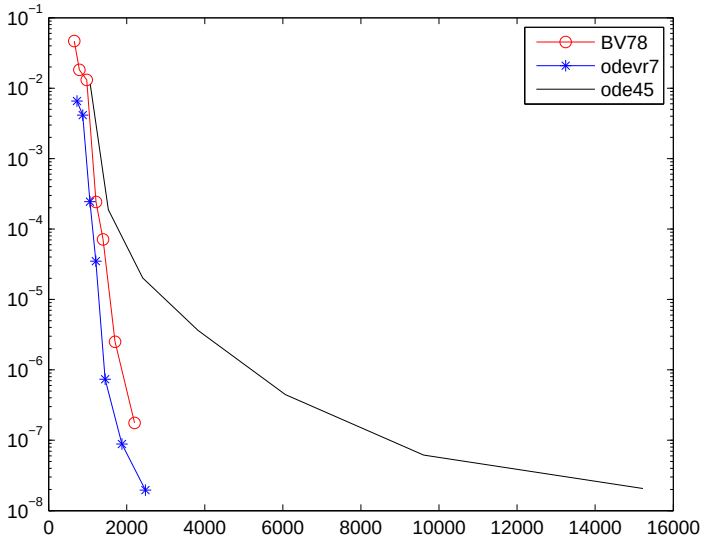

Figure 3: K10 is a restricted three body problem.

5 Conclusions

We assume that evaluation of $f(t, y)$ in the ODEs (1) is vectorized and that the cost of evaluating this function with several arguments is not much greater than the cost of evaluating it with a single argument. The solver `BV78` developed in [11] then competes well with the recommended MATLAB solver `ode45` at all tolerances and is considerably more efficient at stringent tolerances. The same is true of the solver `odevr7` developed here. Indeed, for a wide range of standard test problems solved for a wide range of tolerances, `ode45` had a run time that was about 2.7 times the run time of `odevr7`. This is gratifying when it is appreciated that the new solver has a remarkably strong control of error. The `ode45` solver has a conventional control of the size of an estimate of the local error of the formula of order 4. However, it advances the integration with the formula of order 5 (local extrapolation), so the size of the local error of this formula is controlled only when the tolerance is sufficiently small. The `odevr7` solver advances the integration with a formula of order 7. Not only does it control the size of an estimate of the local error of this formula at the end of the step, but even the size of estimates of the extended local error at 7 other points equally spaced in the span of the step. In `odevr7` this control of the extended local error is a byproduct of the control of the size of the residual of a continuous extension at 8 points equally spaced throughout the span of the step.

References

[1] J.R. Dormand, *Numerical Methods for Differential Equations a Computational Approach*, CRC Press, Boca Raton, FL, 1996.

[2] W.H. Enright and Y. Li, The Reliability/Cost Trade-off for a Class of ODE Solvers, *Numer. Alg.*, to appear.

[3] D.J. Higham, Robust Defect Control with Runge–Kutta Schemes, *SIAM J. Numer. Anal.* 26: 1175–1183 (1989).

[4] T.E. Hull, W.H. Enright, B.M. Fellen, and A.E. Sedgwick, Comparing Numerical Methods for Ordinary Differential Equations, *SIAM J. Numer. Anal.* 9: 603–637 (1972).

[5] J. Kierzenka and L.F. Shampine, A BVP Solver Based on Residual Control and the MATLAB PSE, *ACM Trans. Math. Softw.* 27: 299–316 (2001).

[6] F.T. Krogh, On Testing a Subroutine for the Numerical Integration of Ordinary Differential Equations, *J. ACM* 20: 545–562 (1973).

[7] MATLAB, The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760.

[8] The MathWorks, Code Vectorization Guide, available at `http://www.mathworks.com/support/tech-notes/1100/1109.html`.

[9] W.E. Milne, *Numerical Solution of Differential Equations*, Dover, Mineola, NY, 1970.

[10] J.B. Rosser, A Runge-Kutta for All Seasons, *SIAM Rev.* 9: 417–452 (1967).

[11] L.F. Shampine, Vectorized Solution of ODEs in MATLAB, available at `http://faculty.smu.edu/shampine/current.html`.

[12] L.F. Shampine, Local Error Control in Codes for Ordinary Differential Equations, *Appl. Math. Comp.* 3: 189–210 (1977).

[13] L.F. Shampine, Interpolation for Runge–Kutta Methods, *SIAM J. Numer. Anal.* 22: 1014–1027 (1985).

[14] L.F. Shampine, Solving ODEs and DDEs with Residual Control, *Appl. Numer. Math.* 52: 113–127 (2005).

[15] L.F. Shampine and J. Kierzenka, A BVP Solver that Controls Residual and Error, *JNAIAM* 3: 27–41 (2008).

[16] J.H. Verner, A 'Most Efficient' Runge-Kutta (13:7,8) Pair, available at `http://www.math.sfu.ca/~jverner/`.

[17] H.A. Watts and L.F. Shampine, A-Stable Block Implicit One Step Methods, *BIT* 12: 252–256 (1972).