SIAM J. SCI. COMPUT.
Vol. 29, No. 4, pp. 1637–1649

A NEW IMPLEMENTATION OF SYMPLECTIC RUNGE–KUTTA METHODS*

ROBERT I. MCLACHLAN†

Abstract. We propose a "Newton–Taylor" iteration for solving the implicit equations of symplectic Runge–Kutta methods, using the Jacobian of the vector field and matrix-vector multiplications whose extra cost for certain structured problems is negligible. The structure of Hamiltonian ODEs allows this very simple iteration to be effective. The iteration reduces the number of vector field evaluations almost to that of Newton's method, often only one or two per time step, making symplectic Runge–Kutta methods more efficient even at relatively large time steps.

Key words. Runge–Kutta, implicit, symplectic integrators, inexact Newton methods

AMS subject classifications. 65L06, 65P10

DOI. 10.1137/06065338X

1. Introduction. Symplectic integrators based on splitting, such as the leapfrog method for separable Hamiltonian systems, are fast and simple and give very good results for long simulations [8]. They are widely, almost universally, used in applications from accelerator physics to molecular dynamics to celestial mechanics [16]. However, the only symplectic integrators for *general* Hamiltonian systems are implicit, such as the Gaussian Runge–Kutta methods [11], which has tended to limit their popularity. (The implicit equations must be solved to within round-off error, to preserve symplecticity.) Hairer, Lubich, and Wanner [8] have studied the implementation issues and found the simple ("standard") iteration, (4) below, superior to Newton's method; for very small error tolerances it can even compete with high-order methods based on splitting and composition. However, for the relatively low orders and large time steps often used in long symplectic integrations, the convergence of the standard iteration deteriorates, making the method more expensive, which defeats the purpose of using a large time step.

For other applications of implicit methods, there are methods in which the linear equations defining the Newton step are solved only approximately. In the inexact Newton method an (e.g., Krylov or Chebyshev) iterative method is applied [2, 4, 5]; in the Newton-chord method [18] the Jacobian itself is approximated to simplify the solution step. In the Jacobian-free-Newton–Krylov method [12] the Jacobian-vector multiplications are approximated by finite differences, so the Jacobian itself is never formed.

The application to Hamiltonian systems has a number of special features that lead us to propose a very simple inexact Newton method that, for many structured problems, costs only a constant factor close to 1 times the cost of the standard iteration, yet has a convergence rate very close to Newton's method. For the midpoint rule, for example, the method may require only one or two evaluations of the vector field per time step.

*Received by the editors March 1, 2006; accepted for publication (in revised form) January 9, 2007; published electronically August 1, 2007. This work was supported by the Marsden Fund and the New Zealand Institute of Mathematics and Its Applications.
http://www.siam.org/journals/sisc/29-4/65338.html
†Institute of Fundamental Sciences, Massey University, Palmerston North, New Zealand (r.mclachlan@massey.ac.nz).

The method uses the Jacobian of the vector field and performs matrix-vector multiplications. It is efficient when those two operations are cheap compared to the cost of evaluating the vector field itself. Implementations of implicit methods using Newton's method and direct linear solves try to balance the number of Jacobian factorizations against the number of vector field evaluations [19]. In contrast, our goal is to minimize the number of vector field evaluations without performing any operations that cost substantially (e.g., asymptotically, for large systems) more than a vector field evaluation.

First, consider N-body problems with Hamiltonians of the form

$$H = \sum_{i,j=1}^{N} p_i G(\|q_i - q_j\|) p_j + V(\|q_i - q_j\|). \tag{1}$$

The cost of evaluating the vector field directly is $\mathcal{O}(N^2)$. If G, V, and their derivatives are cheap, then the cost is dominated by the cost of computing $\|q_i - q_j\|$, and the Jacobian is available essentially for free. This has been exploited for separable systems ($G = 1$) by designing efficient high-order symplectic integrators using the Jacobian and matrix-vector multiplications [3, 16]. On the other hand, if G and V are special functions, then it often happens that it is relatively cheap to compute their derivatives at the same time. (This happens in the first example below, where G is an exponential or a Bessel function.) The matrix-vector multiplications, on the other hand, are also $\mathcal{O}(N^2)$ and hence carry a constant work overhead compared to the cost of evaluating the vector field.

Second, consider lattice systems with independent variables indexed by a lattice L containing N points and a Hamiltonian of the form

$$H = \sum_{i \in L} F(x_{i+\Delta}), \tag{2}$$

where $\Delta \subset L$ is a fixed template specifying which sites are coupled. The cost of computing the vector field is $\mathcal{O}(N)$. The Jacobian matrix has only $\mathcal{O}(N)$ nonzero entries, and hence matrix-vector multiplications cost $\mathcal{O}(N)$. Again, the cost of forming and multiplying by the Jacobian is negligible for certain functions F.

2. The Newton–Taylor iteration. We describe the algorithm for the implicit midpoint rule. (Its extension to any implicit Runge–Kutta method is straightforward.) For the ODE $\dot{x} = f(x)$ we are required to solve [11]

$$g(z) := z - \frac{1}{2}\Delta t f(x_0 + z) = 0 \tag{3}$$

for z, where Δt is the time step. (The final update is $x_0 \mapsto x_1 := x_0 + 2z$.) The standard fixed-point iteration is

$$z^{k+1} = \frac{1}{2}\Delta t f(x_0 + z^k) = z^k - g(z^k), \tag{4}$$

and Newton's method is

$$z^{k+1} = z^k - g'(z^k)^{-1} g(z^k) = z^k - (I - A_k)^{-1} g(z^k), \tag{5}$$

where $A_k = \frac{1}{2}\Delta t f'(x_0 + z^k)$.

We replace the iteration matrix $(I - A_k)^{-1}$ by its Taylor series $\sum_{i=0}^{M} A_k^i$ of order M to give the following (Newton–Taylor) outer iteration.

Outer iteration:

$$z^{k+1} = h(z^k) := z^k - \left(\sum_{i=0}^{M} A_k^i \right) g(z^k), \tag{6}$$

which can be evaluated using one vector field, Jacobian evaluation at z^k, and M matrix-vector multiplications.

The analysis of the algorithm is standard in the context of inexact Newton methods [6, 7]. However, it is simpler here because of the simple iteration. Let z^* be the desired solution so that $g(z^*) = 0$, and let $e^k = z^k - z^*$ be the error in the current approximation. Then by Taylor's theorem we have for some $\theta \in [0, 1]$,

$$\begin{aligned} e^{k+1} &= h'(z^k)e^k + \frac{1}{2}h''(z^k + \theta(z^{k+1} - z^k))(e^k, e^k) \\ &= \left(I - \left(\sum_{i=0}^{M} A_k^i \right) (I - A_k) \right) e^k + \frac{1}{2}h''(z^k + \theta(z^{k+1} - z^k))(e^k, e^k) \\ &= A_k^{M+1} e^k + P(e^k, e^k) + \mathcal{O}(\|e^k\|^3), \end{aligned} \tag{7}$$

where $P = \frac{1}{2}h''(z^*) = \mathcal{O}((\Delta t)^2)$. The convergence rate of the modified iteration (6) is therefore $\rho(A_k^{M+1}) = \rho(A_k)^{M+1} = \mathcal{O}((\Delta t)^{M+1})$, where $\rho(A_k)$ is the convergence rate of the standard iteration. Not only is the modified iteration higher-order in the time step, but (at the linear level, i.e., for sufficiently small e^k) one step of the modified iteration is *identical* to $M + 1$ steps of the standard iteration.[1] It is therefore worth increasing M by one precisely when a Jacobian-vector multiplication is cheaper than evaluating the vector field.

We also conclude from (7) that the modified iteration (6) converges for sufficiently small $\|e^0\|$ if and only if the simple iteration converges. We therefore henceforth assume that $\rho(A_k) < 1$.

If M is taken too large, however, then e^{k+1} will be dominated by the quadratic term in (7). Note that as $M \to \infty$, $A_k^{M+1} \to 0$ and $P \to P^\infty$ (say), the iteration tensor for Newton's method. In fact, $P = P^\infty + \mathcal{O}((\Delta t)^{M+1})$, so as $M \to \infty$ the behavior of the modified iteration tends to that of Newton's method. One should therefore adapt M depending on the current iteration, so that $\|A_k^{M+1} e^k\| \approx \|P(e^k, e^k)\|$. This gives the following algorithm, in which (6) is implemented.

Inner iteration:

1. Given z^k, set $w^0 = g(z^k)$;
2. while $\|w^{m+1} - w^m\| > \max(c\|w^0\|^2, tol)$, set $w^{m+1} = w^0 + A_k w^m$;
3. set $z^{k+1} = z^k - w^m$.

Here *tol* is the tolerance, usually close to machine precision, to which the Runge–Kutta equations are to be solved. The ("forcing") parameter c depends on the problem and is related to the norm of P, i.e., to the second derivatives of the vector field [7]. Note that w^0 is the update to z^k in the standard iteration and hence, as long as $\rho(A_k)$ is bounded away from 1, w^0 can be used as an approximation of e^k.

[1] Often, higher-order methods have large error constants, rendering them useless unless the time step is sufficiently small. This does not happen here.

This algorithm will lead to the number M of inner iterations roughly doubling each outer iteration, just as the error decreases in Newton's method.

It is theoretically possible, as in the automatic detection of stiffness [11] and the traditional convergence tests for implicit methods [19], to maintain running estimates of the slowly varying parameters $\rho(A_k)$ and the norm of P, and hence adapt the choice of M more precisely. However, this seems to be more difficult in the present context than in those just cited, because (i) we do relatively few inner iterations and outer iterations, so there is little data on which to base the estimates; (ii) the last iterations may be polluted by round-off error; (iii) the absolute values of the eigenvalues of the Hamiltonian matrix A_k occur with multiplicity 2 (if the eigenvalue is real or imaginary) or 4 (if the eigenvalue is complex), which means that even in the best case, when A_k is not severely nonnormal, an estimate of the convergence rate of the inner iteration can be obtained only from four consecutive iterations; (iv) if the time step is large, as we hope to achieve, then the parameters vary quite quickly (see the first example below).

A more positive viewpoint is to consider these factors as advantages, as they support the use of a very simple algorithm! Most of the experiments reported below use $c = 1$, tol=10^{-15}, and $\|\cdot\|_\infty$ error norms.

To further reduce the number of evaluations of f we terminate the outer iteration (6) at z^{k+1} when $\|g(z^k)\| < tol_2$, where $tol_2 > tol$. Under the model used for terminating the inner iterations, one reasonable choice is to take $tol_2 = \sqrt{tol/c}$, as the final iteration will then take the error down to about $2tol$. This saves one evaluation of f and is the termination criterion used in our experiments. If long-term growth of round-off error is a concern, these criteria may need to be tightened.

A good initial guess is essential for the performance of this method, as one wants the underlying Newton's method to stay close to the root where convergence is quickest. For the midpoint rule, we have used polynomial extrapolation from the previous time steps [8]. The order s of extrapolation is adapted based on the decrease of the backward differences, as follows.

Initialization iteration:

1. Set $w^0 = z_n$;
2. while $\|\nabla^{s+1} z_n\| < \|\nabla^s z_n\|$, set $w^{s+1} = w^s + \nabla^{s+1} z_n$;
3. set $z_{n+1}^0 = w^s$.

The modified iteration also works successfully for higher-order Runge–Kutta methods. The tensor product structure of the linear equations means that only s matrix-vector products are needed per inner iteration for an s-stage method. For systems in $\mathbb{R}^N$, $2N^2 s + 2Ns^2$ operations are required per inner iteration, so the overhead does grow with increasing s, but not at leading ($\mathcal{O}(N^2)$) order.

3. Other inexact Newton iterations (see [10, 13, 14]). We are approximating the solution of $(I - B)\delta z = g$ by the Taylor polynomial $\sum_{i=0}^{M} B^i g$, where for a general Runge–Kutta method with coefficients a_{lj} the matrix B is a block matrix whose (l, j) block is given by $\Delta t a_{lj} f'(Z_j)$, where Z_j are the stage values and $f'(z) = JH''(z)$. We have seen that this is a reasonable approximation, but aren't there better approximations? Krylov methods form the best (in some sense) approximation to δz of the form $p(B)g$ over all polynomials p of degree M; Faber methods use the Faber polynomial, which is the best (in some sense) approximation of $(1-t)^{-1}$ on $\Omega \subset \mathbb{C}$, where Ω contains the spectrum of B. These satisfy an (expensive) long-term recurrence; hence one often chooses Ω to be an ellipse, in which case the Faber polynomials are Chebyshev polynomials which satisfy a 2-term recurrence. In this context,

the Taylor polynomials are themselves the special case of the Chebyshev polynomials, obtained when the ellipse is a circle centered on the origin.

In this case the best approximation property is easily seen. Let $a(t)$ be a complex function analytic on a disk of radius r centered (now for convenience only) on the origin. The powers t^n are orthogonal with respect to the L_2-inner product $(a,b) = \int_{|t|\le r} a(t)\overline{b}(t)\,|dt|^2$, and hence, from the general theory of orthogonal functions, the expansion of $a(t)$ in powers of t—the Taylor polynomial—is the best L_2 approximation to $a(t)$ by a polynomial.

The role of two key features of Hamiltonian systems and of symplectic integrators is now apparent. They both stem from the fact that the Jacobian f' is a Hamiltonian matrix whose eigenvalues exhibit "Hamiltonian symmetry"; namely, they occur in λ, $-\lambda$, $\overline{\lambda}$, $-\overline{\lambda}$ quadruplets. For the midpoint rule, $B = \frac{1}{2}\Delta t f'(Z_1)$ is also Hamiltonian.

First, for the implicit equations to have a solution we need to assume $\mathrm{Re}(\lambda(B)) < 1$, which implies $\mathrm{Re}(\lambda(B)) > -1$. We can also assume $|\mathrm{Im}(\lambda(B))| < 1$, for otherwise the fast oscillations are very poorly represented by the integrator and can even destabilize the calculation. (The currently known integrators that are effective when the fast modes are not resolved are quite different; they are explicit and nonsymplectic [8].) Therefore, the assumption $\rho(B) < 1$ that is required for the Taylor series to converge is not overly restrictive. (Applications where Krylov and Chebyshev iterations are effective typically have $\mathrm{Re}(\lambda(B))$ very large and negative [2, 12].)

Second, the eigenvalue symmetry means that a disk centered on 0, enclosing $\lambda(B)$, will tend to be a much better approximation of $\lambda(B)$ than in the general, non-Hamiltonian, case. Further, the Taylor polynomial requires no knowledge of the spectrum and is the best approximation on all disks centered on the origin (i.e., the best regardless of $\rho(B)$).

The eigenvalue symmetry does mean that an ellipse centered on 0 with real and imaginary axes is never a worse approximation to $\lambda(B)$ than the disk. If this ellipse were known, then a Chebyshev polynomial would be a better approximation to $(1-t)^{-1}$, and would lead to faster convergence, than the Taylor polynomial. Two factors suggest that this advantage will be difficult to exploit in our case. First, for Hamiltonian systems the focal interval of the bounding ellipse can be either real or imaginary, typically alternating between the two, making an accurate estimate of it more difficult; see the first example below. (Existing applications of Chebyshev polynomials use far more iterations than we are proposing here, because they are for $\rho(B)$ large, and hence they can afford to do a few secondary iterations to estimate the ellipse that bounds $\lambda(B)$.) Second, when $\rho(B) < 1$ the convergence advantage of Chebyshev over Taylor polynomials is not that great. For small $\rho(B)$ the Chebyshev convergence factor (see, e.g., [13]) is at best $\frac{1}{2}\rho(B)$ and achieves that only when $\lambda(B)$ is entirely real or entirely imaginary. We leave a fuller comparison of the two approaches for the future.

For the midpoint rule—or indeed for any 1-stage Runge–Kutta method—the iteration matrix B is Hamiltonian. This still holds for almost all 2-stage Runge–Kutta methods, as we now show. If $a_{12}a_{21} \neq 0$, then B is a Hamiltonian matrix because it can be written as a product of an antisymmetric and a symmetric matrix:

$$\begin{aligned} B &= \Delta t \begin{pmatrix} a_{11}JH''(Z_1) & a_{12}JH''(Z_2) \\ a_{21}JH''(Z_1) & a_{22}JH''(Z_2) \end{pmatrix} \\ &= \Delta t \begin{pmatrix} \frac{a_{11}}{a_{21}}J & J \\ J & \frac{a_{22}}{a_{12}}J \end{pmatrix} \begin{pmatrix} a_{21}H''(Z_1) & 0 \\ 0 & a_{12}H''(Z_2) \end{pmatrix}. \end{aligned}$$

(The symplectic structure, given by the first matrix factor, is noncanonical, but this does not affect the eigenvalue symmetry.) On the other hand, if $a_{12} = 0$, say, then B is block lower triangular and not Hamiltonian. However, its eigenvalues are $\Delta t a_{11}\lambda(JH''(Z_1)) \cup \Delta t a_{22}\lambda(JH''(Z_2))$, which do have Hamiltonian symmetry.

For Runge–Kutta methods with ≥ 3 stages the iteration matrix is not in general Hamiltonian, and its eigenvalues need not have Hamiltonian symmetry, numerical examples being easy to construct. In fact, attempting to express the iteration matrix for a 3-stage Runge–Kutta method in the form of an antisymmetric matrix $\alpha_{lj} \otimes J$ times the symmetric matrix $\sum_k \beta_{ljk} H''(Z_k)$ (where $\alpha_{lj} = \alpha_{jl}$, $\beta_{ljk} = \beta_{jlk}$) yields, on eliminating the variables α_{lj}, β_{ljk}, the result that such a factorization exists only if $a_{13}a_{21}a_{32} = a_{12}a_{23}a_{31}$, which does not hold for the 3-stage Gaussian Runge–Kutta. Nevertheless, the iteration matrix will always be *nearly* Hamiltonian, because the matrices $H''(Z_k)$ are all equal to within $\mathcal{O}(\Delta t)$, and in the case of equal matrices such a factorization always exists, namely $a_{lj}JH''(Z) = (a_{lj} \otimes J)(I \otimes H''(Z))$. Therefore the eigenvalues of the iteration matrix will always be close to obeying the same symmetry as those of Hamiltonian matrices.

In other applications, the use of preconditioners is vital [5, 12]. Preconditioning will work here too—decreasing $\rho(B)$, but also destroying, in general, the Hamiltonian structure of B—although not generally as well as in the stiff, dissipative case. We do so few inner iterations anyway that the preconditioner has to be very cheap or extremely effective. An example of a use of a preconditioner, or equivalently a Newton-chord iteration, for a Hamiltonian system is given in [1] for the Korteweg–de Vries equation $u_t = uu_x + u_{xxx}$. The Jacobian is frozen at the stiff term ∂_{xxx}. A second example is given in the solar system study below.

4. Examples. The following example arose in a study of the dynamics of certain curves called momentum sheets in a generalized Camassa–Holm equation that is thought to govern certain classes of solitary waves in shallow water [15]. The curve motion is governed by a PDE which is discretized by placing particles at positions $q_i \in \mathbb{R}^2$ on the curve. Each particle carries momentum $p_i \in \mathbb{R}^2$, and the Hamiltonian is

$$H = \sum_{i=1}^{n} (p_i \cdot p_j) G(\|q_i - q_j\|), \tag{8}$$

which describes geodesics on a Riemannian manifold with metric the inverse of the matrix of G's. The three kernels $G(r)$ of most interest in the application were the Gaussian $\exp(-r^2)$, the modified Bessel functions $K_0(r)$ and $rK_1(r)$ (with the singularity at $r = 0$ appropriately removed). Note that we have, e.g., $K_0'(r) = -K_1(r)$ and $K_0''(r) = K_0(r) + K_1(r)/r$. Since K_0 and K_0' already occur in the vector field, no new Bessel functions have to be computed for the Jacobian.

The motion of the sheet for the Gaussian kernel is shown in Figure 1 for time 50, during which it undergoes a large-scale evolution. There are 50 particles, and the total dimension of the system is 200. To give an idea of the time scale of this problem, both the standard and modified iterations converge at $\Delta t = 6$ but not at $\Delta t = 7$. The final position error is 0.7% at $\Delta t = 1$ and 15.3% at $\Delta t = 5$. The maximum relative energy error during the simulation is 0.2% at $\Delta t = 1$. The spectrum is also shown in Figure 1. The focal interval of the bounding ellipse changes from imaginary to real, to imaginary, and back to real, although the ellipse never becomes very eccentric, and a circle is always a reasonable approximation to the spectrum.

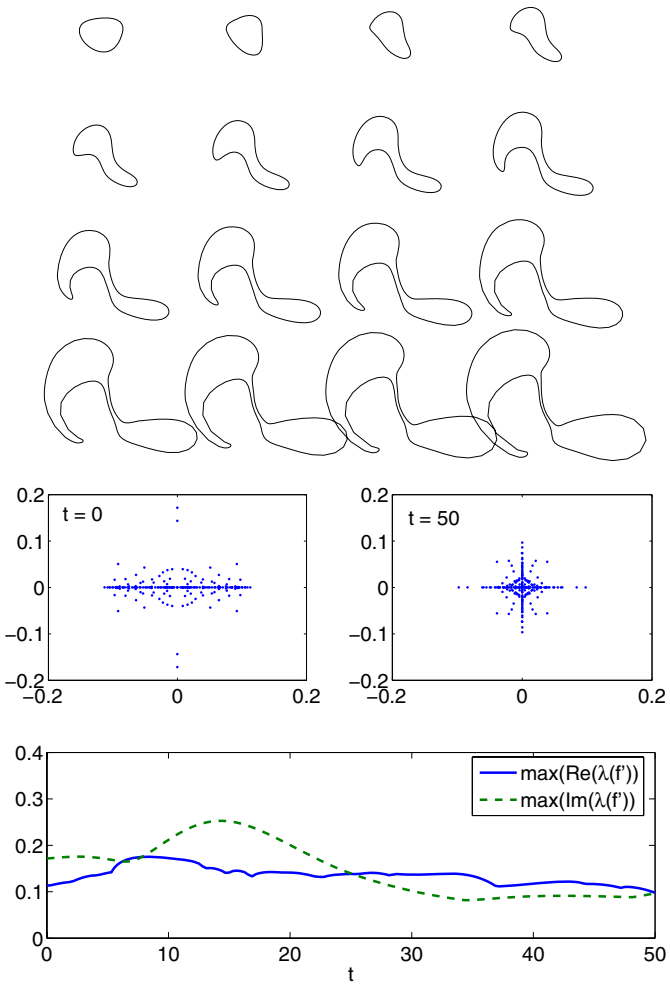

FIG. 1. *Top: Snapshots of the motion of the geodesic curve motion, for* $t \in [0, 50]$. *Middle: Spectrum of the Jacobian at the initial and final times. Bottom: Evolution of the eigenvalues of maximum real and imaginary parts.*

The results are shown in Table 1 and Figure 2. The number of outer iterations (equivalently, number of vector field evaluations) per time step is very close to that used by Newton's method. For the Gaussian kernel with the midpoint rule, one could use $\Delta t = 0.4$ and do about 1 evaluation of f per step (the standard iteration needs 6), achieving a final position error of 0.0011; or one could use $\Delta t = 0.8$ and do about 2 evaluations of f per step (the standard iteration needs 10), achieving a final position error of 0.0045. Clearly, in this problem there is no advantage in the larger time step. But even $\Delta t = 0.4$ is relatively large for this problem and is analogous to the large time steps often used in molecular dynamics problems.

A second observation is that even with the simplest implementation possible, namely using $c = 1$ in the termination criterion for the inner iteration, we have $\overline{N}_{\text{outer}} + \overline{N}_{\text{inner}} \approx \overline{N}_{\text{std}}$ across the whole range of time steps. The algorithm is roughly equivalent to exchanging most iterates of the standard iteration (4) with a matrix-vector multiplication.

TABLE 1

Results for the curve geodesic problem (8) *shown in Figure* 1 *with* 50 *points and dimension* 200. *The integration time is* 50 *and the time step is* $\Delta t = 50/\text{steps}$. $\overline{N}_{\text{outer}}$ *is the mean number of outer iterations (= number of vector field evaluations) per time step used by the modified iteration equation* (6) *(including the initial steps, when the order of the initialization is necessarily lower and more iterations are needed).* $\overline{N}_{\text{inner}}$ *is the mean number of inner iterations (= number of matrix-vector multiplications) per time step.* $\overline{N}_{\text{std}}$ *is the mean number of vector field evaluations per time step used by the standard iteration* (4). *CPU times in seconds are shown for the two methods in a MATLAB implementation. The same extrapolation method is used for initialization for all cases. Its order ranges from* 4 *(for large time steps) to* 14 *(for small time steps).*

$G(r)$	Order	Steps	$\overline{N}_{\text{std}}$	$\overline{N}_{\text{outer}}$	$\overline{N}_{\text{inner}}$	CPU_{std}	CPU
e^{-r^2}	2	10	42.10	4.60	38.50	6.30	1.77
		14	28.07	3.79	24.50	6.03	1.99
		20	20.65	3.30	17.85	6.18	2.37
		28	16.39	3.04	13.75	6.87	3.12
		40	13.13	2.53	10.95	7.87	3.52
		57	10.49	2.07	8.56	8.82	4.10
		80	8.26	1.90	6.50	9.80	5.34
		113	6.09	1.34	4.89	10.29	5.36
		160	4.32	1.06	3.24	10.36	6.02
		226	3.27	1.03	2.24	11.19	8.19
		320	2.98	1.02	1.96	14.53	11.36
e^{-r^2}	4	6	35.83	5.17	37.17	5.20	2.00
		10	23.00	4.10	22.30	5.30	2.52
		20	15.30	3.10	13.40	7.11	3.74
		57	9.32	2.05	7.63	12.38	6.99
		160	4.35	1.07	3.36	17.04	10.73
$K_0(r)$	2	20	19.55	2.45	17.20	12.74	2.82
		40	8.95	1.50	7.48	11.88	3.04
		80	4.90	1.16	3.85	11.90	4.64
		160	3.27	1.05	2.19	15.87	8.25

For the Gaussian kernel and fourth-order 2-stage Gaussian Runge–Kutta, the position errors are now only 0.2% at $\Delta t = 5$ and 4×10^{-6} at $\Delta t = 1$, and the maximum relative energy errors 0.3% at $\Delta t = 5$ and 3×10^{-5} at $\Delta t = 1$. The improvements to the fourth-order method are very similar to those enjoyed by the second-order method. The number of outer iterations is reduced slightly, and the number of inner iterations is significantly reduced at large time steps, because the spectral radius of (a_{ij}) is reduced by a factor 0.58. For the same reason, the fourth-order method converges for larger time steps, failing to converge only at $\Delta t = 9$. For this problem, the fourth-order method always gives a smaller error for a given work (see Figure 2). Both methods can be used satisfactorily at large time steps.

The speedup in CPU time is not as great as the reduction in function evaluations. It is generally a factor of about 2 in this MATLAB implementation—more for larger time steps, a bit less if the order is increased to 4.

The following refinement of the algorithm yields further improvement in the CPU times: On outer iterations after the first, update the Jacobian only if the norm of the previous update of z is greater than some chosen tolerance. This use of inexact Jacobians slows the convergence of both the inner and outer iterations, but (depending on the complexity of the Jacobian) can nevertheless lead to an overall speedup. In the first example of Table 1 (Gaussian kernel with the midpoint rule), with tolerance 0.1 the speedup factor was in the range 1.3–1.38 for the critical range of 20–80 time steps ($0.625 \le \Delta t \le 2.5$). Clearly, the choice of the tolerance is crucial and is problem-dependent.

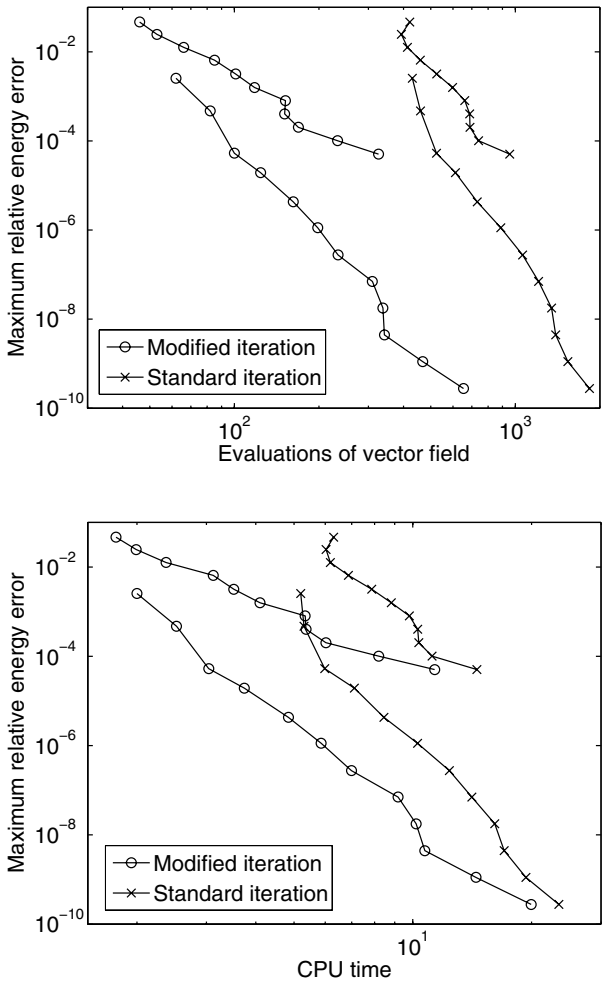

FIG. 2. *Curve geodesics problem. The data from Table* 1 *compared for efficiency* ($G(r) = \mathrm{e}^{-r^2}$, *orders* 2 *and* 4*).*

Sample results are also shown for the more expensive Bessel function kernel $K_0(r)$ with the midpoint rule. $K_0(r)$ can be evaluated in about 78 multiplications, and $K_0(r)$ and $K_1(r)$ together in about 90 multiplications, in MATLAB, which calls a Fortran subroutine. Now the vector field evaluations occupy a larger fraction of the CPU time, and hence the observed speedups are greater.

Two further examples are taken from the results in [8] in a direct comparison with the Fortran code `gnicode`.[2] They are both for separable Hamiltonians of the form $H = \left(\sum_{i=1}^{n} p_i^2/m_i\right) + V(q)$, and in [8] the standard implementation of Gaussian Runge–Kutta methods is compared to explicit composition methods. For these Hamiltonians the momenta at the internal stages can be eliminated, leaving a nonlinear equation for the positions whose Jacobian is now a block matrix with (l, j) block given by $\delta_{lj} I - ((\Delta t A)^2)_{lj} V''(Q_j)$, where A is the matrix of the Butcher tableau and the Q_j are the stage values of the positions. This halves the number of standard iterations,

[2]The results for `gnicode` shown in Table 2 are better than those given in [8], because the code was improved further since the book was published [9].

TABLE 2

Kepler problem. One period of the 2D Kepler problem with period 2π and eccentricity 0.6 is integrated using the standard iteration in the `gnicode` *Fortran implementation and using the modified iteration in an adaptation of the same code. CPU times are reported for the GNU Fortran optimizing compiler 3.4 on a 1GHz G4 Mac. The same initialization is used in each case (extrapolation for order 2, and an order $s+2$ Runge–Kutta-type method for order $2s \geq 4$). N_f is the total number of vector field evaluations including those needed for initialization.*

Order	Δt	N_f Std	N_f Mod.	N_{inner}	$\overline{N}_{\text{outer}}$ Std	$\overline{N}_{\text{outer}}$ Mod.	$\overline{N}_{\text{inner}}$	CPU time Std	μs Mod.
2	$2\pi/50$	383	110	242	7.68	2.20	4.84	349	185
	$2\pi/100$	399	142	229	3.99	1.42	2.29	439	281
	$2\pi/200$	511	225	284	2.56	1.12	1.42	653	494
	$2\pi/400$	590	407	438	1.47	1.02	1.09	988	925
4	$2\pi/25$	375	143	110	7.50	1.86	4.40	261	133
	$2\pi/50$	547	235	142	5.47	1.35	2.84	379	203
	$2\pi/100$	881	427	194	4.41	1.13	1.94	611	352
	$2\pi/200$	1405	805	279	3.51	1.01	1.40	989	647
	$2\pi/400$	2521	1601	447	3.15	1.00	1.12	1764	1252
8	$2\pi/25$	525	213	75	5.25	1.63	3.00	327	207
	$2\pi/50$	791	343	99	3.96	1.22	1.98	494	316
	$2\pi/100$	1215	627	141	3.04	1.07	1.41	771	557
	$2\pi/200$	1707	1203	222	2.13	1.00	1.11	1143	1039
	$2\pi/400$	2779	2403	401	1.74	1.00	1.00	1913	2045
12	$2\pi/25$	679	265	70	4.53	1.43	2.80	414	274
	$2\pi/50$	951	447	83	3.17	1.16	1.66	591	424
	$2\pi/100$	1255	805	123	2.09	1.01	1.23	829	745
	$2\pi/200$	1923	1605	206	1.60	1.00	1.03	1328	1444
	$2\pi/400$	3367	3205	401	1.40	1.00	1.00	2371	2864

somewhat reduces the number of outer iterations, and halves the number of inner iterations and the size of the matrices involved. We present results for the relative energy errors; results for the global errors are similar.

The first example is the two-dimensional Kepler problem with $H = \frac{1}{2}(p_1^2 + p_2^2) + 1/\sqrt{q_1^2 + q_2^2}$, for which results are given in Table 2. Despite the low dimension, the completely different implementation, and the fact that the vector field evaluations do not dominate the CPU time (the single square root costs only about 7 multiplications) the results are strikingly similar to those for the curve geodesic problem.

The second example is the outer solar system problem. The sun, four outer planets, and Pluto are evolved in standard coordinates for 500,000 days. Again, the problem is not very large, and the $n(n-1)/2 = 15$ square roots actually number less than the $3n = 18$ degrees of freedom. Still, the results are similar to those of the previous two examples. For this problem due to the scaling of the variables it was advantageous to adapt the forcing parameter c that controls the termination of the inner iteration. The results are shown for $c = 100$ in Figure 3. The order 4 implicit and explicit methods now have equal performance for time steps less than about 120 days (indicated by an arrow). The order 8 and 12 methods beat the explicit methods for relative energy errors less than about 10^{-7}.

Because the system is nearly integrable, a good preconditioner is available in the form of the product of the exact inverses of the Jacobians of the sun-planet two-body systems. These can be calculated explicitly in terms of scalar products. The convergence factor $\rho(A)$ is reduced dramatically, so that only one inner iteration per Newton step is required. However, the preconditioning is expensive, and it is not yet clear whether the extra level of complexity is justified. (Besides, explicit methods can also exploit the near integrability.)

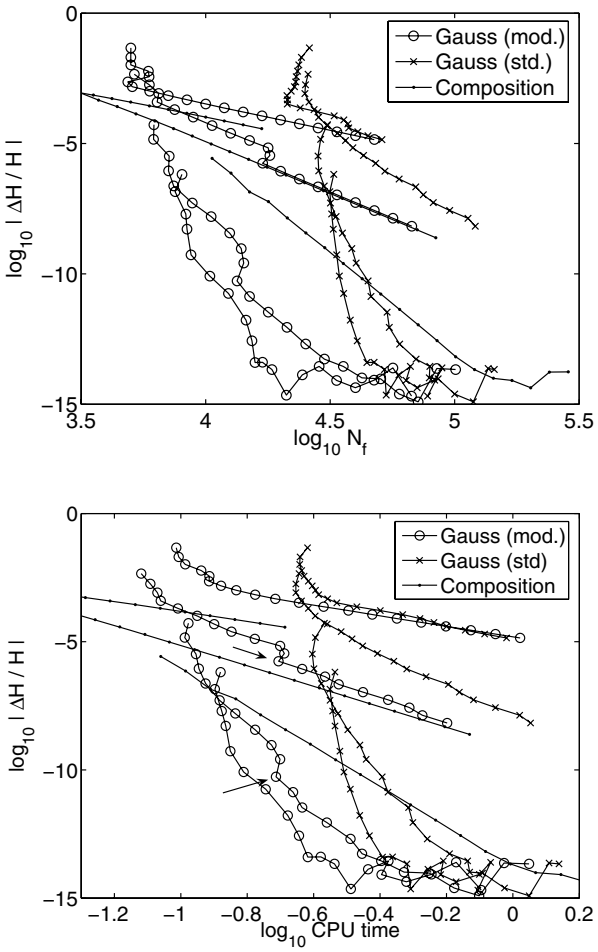

FIG. 3. *Outer solar system problem. Gaussian Runge–Kutta methods of orders* 2, 4, 8, *and* 12 *compared in the standard and modified Fortran implementations. Results for three explicit composition methods are also shown, of orders* 2 *(*1 *stage leapfrog),* 4 *(*5 *stages), and* 8 *(*17 *stages),* `gnicode`*'s methods* 21, 45, *and* 817, *respectively. The performance of the implicit methods is best when the mean number of outer iterations drops to* 1, *which occurs at the points marked by arrows. These correspond to a time step of* 120 *days for order* 4 *and* 240 *days for order* 8. *The order* 12 *results are influenced by round-off for time steps less than* 300 *days.*

5. Conclusions. Even if the vector field has no particular structure, the modified iteration (6) is likely to be an improvement over the standard iteration (4). The only unavoidable requirement is that we need $\rho(A) < 1$; i.e., the problem must not be stiff. However, the case $\rho(A) > 1$ takes us into the entirely different realm of highly oscillatory systems [8]. Therefore the present algorithm is most suited to nonstiff problems in which implicit methods are needed for their geometric properties such as symplecticity.

Many large physical systems have an appropriate structure which reduces the cost of the integration considerably, down to one or two vector field evaluations per time step. Of course, some systems have so much structure that it is possible to exploit it directly in solving the implicit Runge–Kutta equations. For example, for N-body systems one could use a fast multipole method, and for smooth lattice systems such as

those arising as semidiscretizations of PDEs one could use multigrid. However, these approaches are not only vastly more complicated than the present proposal, they are also much more problem-specific.

It is tempting to ask whether the information contained in the Jacobians can be used for purposes other than just speeding up the convergence. Unfortunately, there are no symplectic multiderivative Runge–Kutta methods [8], nor are there any "elementary differential" Runge–Kutta methods using just the Jacobian—the simplest is the midpoint rule applied to the modified Hamiltonian $H + \frac{1}{24}(\Delta t)^2 f^T H'' f$, which requires the second derivative of the vector field f. There are Runge–Kutta-like methods using f' [17], but these are not very developed.

The algorithm presented here is a reasonable one because of the special (Hamiltonian) structure of the problem. It is tempting to consider further specializations of this structure for which the algorithm can be improved further. Directions for future research include automatic selection of the tolerances and exploring cases in which inexact Jacobians, Chebyshev iteration, and/or preconditioners should be used.

Acknowledgements. I would like to thank Stephen Marsland for discussions on the momentum sheet example, and Ander Murua for discussions on the use of Chebyshev iteration.

REFERENCES

[1] U. M. Ascher and R. I. McLachlan, *Multisymplectic box schemes and the Korteweg–de Vries equation*, Appl. Numer. Math., 48 (2004), pp. 255–269.
[2] L. Bergamaschi, M. Caliari, and M. Vianello, *Efficient approximation of the exponential operator for discrete 2D advection-diffusion problems*, Numer. Linear Algebra Appl., 10 (2003), pp. 271–289.
[3] S. Blanes and P. C. Moan, *Practical symplectic partitioned Runge–Kutta and Runge–Kutta–Nyström methods*, J. Comput. Appl. Math., 142 (2002), pp. 313–330.
[4] P. N. Brown and A. C. Hindmarsh, *Matrix-free methods for stiff systems of ODE's*, SIAM J. Numer. Anal., 23 (1986), pp. 610–638.
[5] P. N. Brown and A. C. Hindmarsh, *Reduced storage matrix methods in stiff ODE systems*, J. Appl. Math. Comput., 31 (1989), pp. 40–91.
[6] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.
[7] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Sci. Comput., 17 (1996), pp. 16–32.
[8] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer, Berlin, 2002.
[9] E. Hairer, *personal communication*, University of Geneva, 2006.
[10] M. H. Gutknecht and S. Röllin, *The Chebyshev iteration revisited*, Parallel Computing, 28 (2002), pp. 263–283.
[11] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations*. II. *Stiff and Differential-Algebraic Problems*, 2nd ed., Springer, Berlin, 1996.
[12] D. A. Knoll and D. E. Keyes, *Jacobian-free Newton–Krylov methods: A survey of approaches and applications*, J. Comput. Phys., 193 (2004), pp. 357–397.
[13] X. Li, *The convergence rate of the Chebyshev SIM under a perturbation of a complex line segment spectrum*, Linear Algebra Appl., 230 (1995), pp. 47–60.
[14] T. A. Manteuffel, *The Tchebyshev iteration for nonsymmetric systems*, Numer. Math., 28 (1977), pp. 307–327.
[15] R. I. McLachlan and S. R. Marsland, *The Kelvin–Helmholtz instability of momentum sheets in the Euler equations for planar diffeomorphisms*, SIAM J. Appl. Dyn. Sys., 5 (2006), pp. 726–758.
[16] R. I. McLachlan and G. R. W. Quispel, *Geometric integrators for ODEs*, J. Phys. A, 39 (2006), pp. 5251–5285.
[17] S. Miesbach and H. J. Pesch, *Symplectic phase flow approximation for the numerical integration of canonical systems*, Numer. Math., 61 (1992), pp. 501–521.

[18] R. Schreiber and H. B. Keller, *Driven cavity flows by efficient numerical techniques*, J. Comput. Phys., 49 (1983), pp. 310–333.
[19] L. F. Shampine, *Implementation of implicit formulas for the solution of ODEs*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 103–118.