ACADEMIC PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 187 (2003) 298–317

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

# An adaptive $N$-body algorithm of optimal order

C. David Pruett [a,*], Joseph W. Rudmin [b], Justin M. Lacy [c]

[a] *Department of Mathematics and Statistics, James Madison University, Harrisonburg, VA 22807, USA*
[b] *Physics Department, James Madison University, Harrisonburg, VA 22807, USA*
[c] *SETI Institute, 2035 Landing Drive, Mountain View, CA 94043, USA*

Received 22 May 2002; received in revised form 11 December 2002; accepted 14 February 2003

**Abstract**

Picard iteration is normally considered a theoretical tool whose primary utility is to establish the existence and uniqueness of solutions to first-order systems of ordinary differential equations (ODEs). However, in 1996, Parker and Sochacki [Neural, Parallel, Sci. Comput. 4 (1996)] published a practical numerical method for a certain class of ODEs, based upon modified Picard iteration, that generates the Maclaurin series of the solution to arbitrarily high order. The applicable class of ODEs consists of first-order, autonomous systems whose right-hand side functions (generators) are projectively polynomial; that is, they can be written as polynomials in the unknowns. The class is wider than might be expected. The method is ideally suited to the classical $N$-body problem, which is projectively polynomial. Here, we recast the $N$-body problem in polynomial form and develop a Picard-based algorithm for its solution. The algorithm is highly accurate, parameter-free, and simultaneously adaptive in time and order. Test cases for both benign and chaotic $N$-body systems reveal that optimal order is dynamic. That is, in addition to dependency upon $N$ and the desired accuracy, optimal order depends upon the configuration of the bodies at any instant.


*Keywords:* $N$-body problem; Initial-value problems; Picard iteration; Power series; Maclaurin series; Optimal order

## 1. Introduction

Of historical and practical interest, the $N$-body problem occupies a unique place at the heart of classical physics. Newton's stunning success with the two-body problem – the orbit of a planet about the sun – for which he developed both the calculus and the universal theory of gravitation as tools, launched the scientific age of which we are yet a part. The intractability of the three-body problem led Poincaré to consider the qualitative behavior of systems of ordinary differential equations (ODEs) thereby providing the tools necessary for the articulation of nonlinear dynamics and chaos theory in the late 20th century (for example, phase portraits and Poincaré maps).

* Corresponding author. Tel.: +1-540-568-6227.
*E-mail addresses:* pruettcd@jmu.edu (C.D. Pruett), rudminjw@jmu.edu (J.W. Rudmin).

0021-9991/03/$ - see front matter 
doi:10.1016/S0021-9991(03)00101-3

The $N$-body problem remains alive and well in a variety of contexts. Modern methods for simulating the formation of structure in the universe (e.g., both planetary and galactic accretion) exploit $N$-body dynamics [2,3,10]. Moreover, Lagrangian (particle-tracking) methods such as smoothed particle hydrodynamics (SPHs) [9] share much in common with $N$-body algorithms and have been adapted to problems as diverse as liquid-drop oscillations, gravitational collapse, and supernova explosion.

It can be inferred from recent review articles on astrophysical applications of $N$-body dynamics [10] that $N$-body algorithms fall into two categories: $N$-body *integrators*, and $N$-body *simulators*. The distinction, according to Morbidelli [10], is subtle but important. Whereas integrators directly solve Newton's equations of motion without approximations except those inherent in the numerical scheme itself, simulators *model* certain dynamical and physical effects via layers of approximation. The distinction is analogous to the difference between direct numerical simulation (DNS) and large-eddy simulation (LES) in Eulerian computational fluid dynamics. Typically, integrators accurately evolve the motions of individual particles (bodies), which requires exceedingly high accuracy. Integrators scale as $N^2$ in computational effort and are limited by practical considerations to a few hundred bodies. Simulators, on the other hand, which seek statistical properties of large aggregates of particles and in which the specifics of individual particles are less important, favor efficiency at the expense of some degradation in accuracy. Such algorithms enjoy operation counts as low as $N \log N$ and treat $10^8$ particles in reasonable time on supercomputers.

The review articles of Bertschinger [3] and Morbidelli [10] discuss $N$-body algorithms for simulations of structure formation in the universe and the solar system, respectively. A breakthrough in $N$-body simulation technology occurred with the introduction of mixed-variable symplectic methods by Wisdom and Holman in 1991 [16], which tend to preserve volume in phase space. The algorithm was adapted to asynchronous time-steps by Saha and Tremaine in 1994 [14]. Symplectic algorithms tend to use non-adaptive leap-frog time advancement, which is of second order. Low-order without adaptation renders the method unsuitable for close encounters (near collisions) [10], as do basic assumptions of the formulation. Another milestone in $N$-body simulation is marked by the advent of hierarchal tree algorithms, in which individual clusters of particles are treated as a single pseudo-particle located at the center of mass of the cluster. Tree algorithms (e.g. [2,15]) scale as $N \log N$, which permits their application to massive systems. However, they are limited in accuracy to approximately 1% [2].

Direct integration of the classical $N$-body problem has tended to favor classical methods. Among these are Runge–Kutta, Bulirsch–Stoer, and Taylor-series methods [10]. Since 1985, many $N$-body algorithms have exploited some variant of Aarseth-type techniques [1,12], which are hybrid schemes inherently capable of simulation or integration. The Aarseth methods are predictor-corrector methods derived from Taylor-series expansions, typically of 4th-order, and coupled with asynchronous time advancement. Forces are separated into *irregular* (local) and *regular* (distant) contributions which are represented by two different polynomial expansions. Local force contributions are directly computed and the distant ones are predicted (modeled). The choice of nearest neighbors is effected on the basis of local density contrast. The separation of scales in time results in increases in efficiency typically of two orders of magnitude relative to classical methods [12]. However the scheme is optimal only if such separation of scales exists. In simulations, Aarseth-type schemes typically preserve energy to less that one part in $10^{-4}$.

The issue of optimal order arises in both contexts: $N$-body integration and $N$-body simulations. Press and Spergel [12] address the "extrapolability" of Aarseth-type codes, which they define as how far into the future the force may be projected while maintaining pre-established fractional accuracy of the solution. Higher order generally results in greater extrapolability and hence larger time-steps. However, in numerical experiments, they observe a point of diminishing returns, which suggests the existence of an optimal order. Their studies indicate that orders considerably higher than four are indicated, but they are unable to establish precise guidelines for optimizing order. The issue is picked up by Makino [8], who observes from numerical experiments that high-order schemes are favored over low-order during close encounters. In our interpretation, imminent close encounters (near singularities) are first manifest in high-order derivatives.

Thus adaptive low-order schemes run the risk of stepping over close encounters. Makino concludes that optimal order depends both upon the number of bodies $N$ and the desired accuracy. It would therefore seem that optimal order is inherently *dynamic*, in that it depends upon desired accuracy, system size $N$, *and* the system configuration at any given instant.

In 1993, Le Guyader [7] introduced a power-series method for the $N$-body problem that is of arbitrarily high order, and demonstrated the scheme in evolving the solar system to extraordinary accuracy (relative position errors $< 10^{-10}$ in the outer planets) with polynomial orders as high as 25. Despite its relative inefficiency, such a scheme is useful for addressing the issue of optimal order. The method presented herein is also a power-series method of arbitrary order, similar to that of Le Guyader but derived independently. The present method generalizes that of Le Guyader in two fundamental ways. First, the method is derived from the novel perspective of Picard iteration, and thus represents a general approach readily adaptable to a wide class of problems, of which the $N$-body problem is an especially interesting example. Second, whereas the method of Le Guyader is adaptive in time, the present method is adaptive in both time and order.

In summary, the present paper presents a straightforward and novel computational approach suitable for a wide class of low-dimensional initial-value problems for which exceptionally high accuracy is favored over computational efficiency. The method also provides an appropriate theoretical tool for addressing numerical issues related to optimal order. Based upon Picard iteration, the method is demonstrated for the classical $N$-body problem. Normally a theoretical tool, Picard iteration is adapted in Section 2 to solve autonomous systems of first-order ODEs, and the method is illustrated with simple linear and nonlinear (predator–prey) examples. In Section 3, relevant theoretical considerations are reviewed. Section 4 extends the algorithm to a wide class of initial value problems, and in Section 5, the algorithm is adapted to the classical $N$-body problem. An adaptive time-advancement procedure is developed in Section 6. In Section 7, the adaptive procedure is enhanced to dynamically optimize both time-step and order. This is accomplished by minimizing computational effort while maintaining a pre-specified global accuracy in the solution. If the machine unit round-off error is exploited as the accuracy constraint, the entire algorithm is a parameter-free. The algorithm is demonstrated by evolving both benign (the solar system) and chaotic $N$-body systems to double-precision global accuracy. Concluding remarks and directions for future work are offered in Section 8.

## 2. Picard iteration revisited

Picard iteration has long been exploited to establish the existence and uniqueness of solutions of first-order systems of differential equations (e.g. [4]). However, until recently [11], Picard iteration was assumed impractical as a solution method per se. The advent of computer algebra systems has removed this impracticality. Furthermore, attempts to design practical solution methods based upon Picard iteration have yielded not only successful new algorithms but have raised intriguing theoretical issues. In this section, we briefly review Picard iteration, and demonstrate its application to the solution of the classical predator–prey problem.

Any system of high-order ordinary differential equations can be rendered as a first-order system, namely

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}, t) \ (t > t_0), \quad \mathbf{y}(t_0) = \mathbf{y}^0. \tag{1}$$

The successive Picard iterates $\mathbf{y}^k(t)$ of this system, defined by

$$\mathbf{y}^{k+1} = \mathbf{y}^0 + \int_{t_0}^{t} \mathbf{f}[\mathbf{y}^k(s), s]\,\mathrm{d}s \tag{2}$$

converge uniformly to the exact solution $\mathbf{y}(t)$ for all $|t - t_0| \leqslant T$ provided the vector function $\mathbf{f}$ satisfies the Lipschitz condition, in which case $\mathbf{y}(t)$ is unique [4]. Because non-autonomous systems of ODEs can be rendered autonomous, it suffices to consider

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}) \ (t > t_0), \quad \mathbf{y}(t_0) = \mathbf{y}^0, \tag{3}$$

whereby the Picard iterates are

$$\mathbf{y}^{k+1} = \mathbf{y}^0 + \int_0^t \mathbf{f}[\mathbf{y}^k(s)]\,\mathrm{d}s, \tag{4}$$

where without loss of generality, $t_0 = 0$. Henceforth, according to [11], we will refer to the function $\mathbf{f}$ as the *generator* of the ODE.

### 2.1. A 1D example: the exponential function

Consider the scalar, linear initial value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = y \ (t > 0), \quad y(0) = 1. \tag{5}$$

It is straightforward to show that $y^{k+1}(t) = 1 + \int_0^t y^k(s)\,\mathrm{d}s = 1 + t + t^2/2 + \cdots + t^{k+1}/(k+1)!$ That is, the $k$th Picard iterate is the $k$th-order Maclaurin polynomial.

### 2.2. A 2D example: the predator–prey problem

The classical predator–prey problem provides a simple but useful example to illustrate a practical solution method based upon Picard iteration.

$$\begin{aligned} \frac{\mathrm{d}y_1}{\mathrm{d}t} &= +c_1 y_1 - c_2 y_1 y_2, \\ \frac{\mathrm{d}y_2}{\mathrm{d}t} &= -c_3 y_2 + c_4 y_1 y_2. \end{aligned} \tag{6}$$

Here, $c_1$, $c_2$, etc., are positive constants and $\mathbf{y} = [y_1, y_2]^{\mathrm{T}}$, where $y_1$ and $y_2$ denote the populations (in say, thousands and hundreds) of prey and predator species, respectively. Note that $\mathbf{f}$ is both nonlinear and autonomous. Because the system is non-dissipative, the solution is, for any non-equilibrium initial condition, a closed limit cycle. For specificity, we take $c_1 = 1.1$, $c_2 = 0.9$, $c_3 = 1.0$, and $c_4 = 1.0$, and $\mathbf{y}^0 = [0.6, 0.7]^{\mathrm{T}}$. From the generic initial condition with $t_0 = 0$, the first two Picard iterates are

$$\begin{aligned} y_1^1(t) &= y_1^0 + \left(a y_1^0 - b y_1^0 y_2^0\right) t, \\ y_2^1(t) &= y_2^0 + \left(-c y_2^0 + d y_1^0 y_2^0\right) t \end{aligned} \tag{7}$$

and

$$\begin{aligned} y_1^2(t) &= y_1^0 + \left(c_1 y_1^0 - c_2 y_1^0 y_2^0\right) t + \frac{1}{2}\left[c_1^2 y_1^0 + 2c_1 c_2 y_1^0 y_2^0 + c_2 c_3 y_1^0 y_2^0 - c_2 c_4 \left(y_1^0\right)^2 y_2^0 + c_2^2 y_1^0 \left(y_2^0\right)^2\right] t^2 \\ &\quad - \frac{1}{3} c_2 y_1^0 y_2^0 \left(c_1 - c_2 y_2^0\right)\left(-c_3 + c_4 y_1^0\right) t^3 \\ y_2^2(t) &= y_2^0 + \left(-c_3 y_1^0 + c_4 y_1^0 y_2^0\right) t + \frac{1}{2}\left[c_3^2 y_2^0 - 2c_3 c_4 y_1^0 y_2^0 + c_4^2 (y_1^0)^2 y_2^0 - c_1 c_4 y_1^0 y_2^0 - c_2 c_4 y_1^0 (y_2^0)^2\right] t^2 \\ &\quad + \frac{1}{3} c_4 y_1^0 y_2^0 \left(c_1 - c_2 y_2^0\right)\left(-c_3 + c_4 y_1^0\right) t^3. \end{aligned} \tag{8}$$

What is noteworthy about (8) is that the approximation holds for arbitrary initial state $\mathbf{y}^0$. Thus the *same* equation can be successively advanced in time simply by reseting the initial condition following each step. In general, the higher the Picard iterate, the larger the admissible time-step. Whereas the generation of successively higher Picard iterates rapidly becomes too tedious for hand computation, such calculations are routine for computer algebra systems.

In the sample problem, $\mathbf{f}$ is of polynomial form and degree two, in which case Picard iterate $\mathbf{y}^k$ is of order $2^k - 1$. More generally, for polynomial $\mathbf{f}$ of degree $p$, $q_k = (p^k - 1)/(p - 1)$, where $q_k$ denotes the degree of the $k$th Picard iterate. For arbitrary initial conditions, high order Picard iterates of polynomial generators become intractable, even for computer algebra systems, by, say, iteration eight, in which case there may be no advantage in using a Picard-based method over, for example, classical Runge–Kutta methods. Alternately, the initial conditions (and parameters) can be specified numerically. For example, for the predator–prey system, the 2nd Picard iterate (8) collapses to

$$\begin{aligned} y_1^2(t) &= 0.6 + 0.282t + 0.14187t^2 + 0.023688t^3, \\ y_2^2(t) &= 0.7 - 0.280t + 0.15470t^2 - 0.026320t^3. \end{aligned} \tag{9}$$

For this scenario, the order of the expansion that can be generated by a computer algebra system is quite large. However, the expansion is valid only for a single time-step, following which another expansion must be generated.

In summary, Picard iteration can exploited once to generate a reusable power-series expansion of the solution with symbolic coefficients that incorporate the initial condition, or it can be exploited at each time-step to re-derive a power-series approximation with numerical coefficients.

## 3. Theoretical considerations

The generation of series solutions by Picard iteration would remain a novelty without much practical import were it not for several theoretical considerations that are potentially far reaching.

The straightforward symbolic computation of the successive Picard iterates in the example above hinged upon the polynomial form of the generator $\mathbf{f}$. (Henceforth, for clarity, we adopt the convention that *degree* refers to the generator whereas *order* refers to the approximation of the solution.) Many or most systems of ODEs are not originally cast in polynomial form. For example, consider the two-body problem of a satellite orbiting a planet at the fixed location $(x_0, y_0)$. The state vector $[x, y, u, v]^{\mathrm{T}}$ of the satellite is governed by the autonomous system

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= u, \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= v, \\ \frac{\mathrm{d}u}{\mathrm{d}t} &= \frac{GM(x_0 - x)}{[(x - x_0)^2 + (y - y_0)^2]^{(3/2)}}, \\ \frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{GM(y_0 - y)}{[(x - x_0)^2 + (y - y_0)^2]^{(3/2)}}, \end{aligned} \tag{10}$$

where $M$ is the mass of the planet and $G$ is the universal gravitational constant. By defining a new variable, the inverse separation distance $s = 1/\sqrt{(x - x_0)^2 + (y - y_0)^2}$, the system above (10) can be rendered as

$$\begin{aligned}
\frac{\mathrm{d}x}{\mathrm{d}t} &= u,\\
\frac{\mathrm{d}y}{\mathrm{d}t} &= v,\\
\frac{\mathrm{d}u}{\mathrm{d}t} &= GM(x_0 - x)s^3,\\
\frac{\mathrm{d}v}{\mathrm{d}t} &= GM(y_0 - y)s^3,\\
\frac{\mathrm{d}s}{\mathrm{d}t} &= [(x_0 - x)u + (y_0 - y)v]s^3
\end{aligned} \tag{11}$$

for which $\mathbf{f}$ is a 5th-degree polynomial in the five unknowns.

It appears that many generators that arise from physical laws can be rendered polynomial. Following the terminology of Parker and Sochacki [11], we term such generators *projectively polynomial*. Moreover, Carothers [5] has shown that any system whose generator $\mathbf{f}$ is polynomial can be re-written in polynomial form of degree two (although how to re-write a specific $\mathbf{f}$ is not obvious). This observation has potentially far-reaching implications. It would appear that many nonlinear systems of ODEs are quadratic systems in disguise.

But suppose that $\mathbf{f}$ is not polynomial, yet belongs to $C_0^1$. Let $\mathbf{y}(t)$, $t \in [0, T]$, be the unique solution for generator $\mathbf{f}$ and initial condition $\mathbf{y}_0$. The theory [11] states that, for any small number $\epsilon > 0$, there exists a polynomial generator $\mathbf{f}_\mathrm{p}$ with solution $\mathbf{y}_\mathrm{p}(t)$ (for the same initial condition), for which $|\mathbf{y}(t) - \mathbf{y}_\mathrm{p}(t)| < \epsilon$ for all $t \in [0, T]$. This result for ODEs is analogous to the Weierstrass Approximation Theorem for approximating polynomials.

Furthermore, the $k$th Picard iterate $\mathbf{y}_n(t)$ is the $k$th Maclaurin polynomial plus a polynomial of degree greater than $k$ [11]. That is, each successive Picard iterate generates one additional term of the Maclaurin series expansion of the solution. Thus, Picard iteration provides an alternative to conventional methods for developing the Maclaurin-series expansion of the solution. Rather than successively differentiating the generator $\mathbf{f}$, one successively integrates power series in $t$.

Finally, the Maclaurin series is also generated by *modified* Picard iteration. That is, suppose $\tilde{\mathbf{y}}^k(t)$ represents the $k$th modified Picard iterate, which retains terms only through order $k$ in $t$. Then, the following process, termed *modified* Picard iteration, generates the $k$th-order Maclaurin polynomial approximation of $\mathbf{y}(t)$:

$$\tilde{\mathbf{y}}^{k+1} = \mathbf{y}^0 + \int_0^t \mathbf{f}[\tilde{\mathbf{y}}^k(s)]\,\mathrm{d}s. \tag{12}$$

In summary, ODEs of physical origin can often be re-written as first-order autonomous systems with polynomial generators. The Maclaurin-polynomial approximation of the solution of such systems is readily generated term-by-term to arbitrarily high-order by modified Picard iteration, a routine computation for modern computer-algebra systems.

## 4. An algorithm based upon modified Picard iteration

We demonstrate modified Picard iteration by solving the classical predator–prey system (6), with the help of the computer algebra software package Maple. Maple code to generate the successive Maclaurin-series terms for the predator–prey example (with arbitrary parameters and initial conditions) is provided in Appendix A. The $m + 1$ Maclaurin coefficients of the $n$ equations are stored as an $n \times (m + 1)$ matrix $C$, computed only once, automatically converted to Fortran code, and embedded (via an INCLUDE directive)

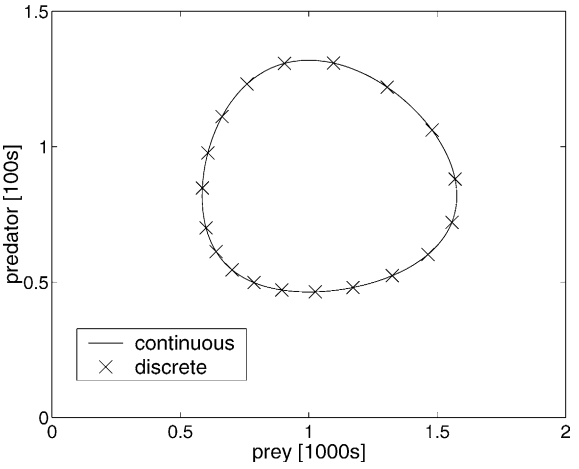


Fig. 1. Continuous and discrete solutions of predator–prey problem: discrete solution by modified Picard method of fixed order $m = 7$ with uniform time intervals.

directly into a generic evaluation subroutine **matrix_C**, which is given in Appendix B. The subroutine is called within a time advancement loop that updates the solution via Horner's method for vectors (Appendix B). The entire algorithm, summarized below, is remarkably compact.

for $l$ from 1 to $s$ by 1 do
  $t = l\Delta t$
  call matrix_C($\mathbf{y}_0$,C)
  $\mathbf{y} = \mathbf{c}_m$
  for $j$ from $m-1$ by $-1$ to 0 do
    $\mathbf{y} = \Delta t\mathbf{y} + \mathbf{c}_j$
  end do
  $\mathbf{y}_0 = \mathbf{y}$
end do

where $\mathbf{c}_j$ is the $n$-vector comprised of the elements of the $j$th column of $C$. Here, for simplicity, the time increment $\Delta t$ is presumed constant, but it need not be.

Fig. 1 presents the solution of the predator–prey system obtained by the Picard-based algorithm above. The integration time is approximately the period of the limit cycle. The figure compares the continuous solution with a discrete solution of 18 steps equally spaced in time and obtained by modified Picard iteration with $m = 7$. Clearly, high order permits an accurate solution despite large time-steps.

## 5. A classical application: the $N$-body problem

Here, we develop and demonstrate a second Picard-based method in which the Maclaurin coefficients are generated numerically rather than symbolically. The method is applied to the classical $N$-body problem.

Let $x_{ij}$ and $v_{ij}$ represent the $i$th position coordinate and the $i$th velocity component, respectively, of particles $j = 1, 2, \ldots, N$ of respective masses $m_j$. Furthermore, let $r_{jk} = [\sum_{i=1}^{3}(x_{ij} - x_{ik})^2]^{1/2}$ denote the separation distances between the centers of particles $j$ and $k$, where $k = 1, 2, \ldots, N$. In classical form, the $N$-body problem consists of $6N$ coupled autonomous first-order ODEs, namely

$$\begin{aligned}\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} &= v_{ij},\\ \frac{\mathrm{d}v_{ij}}{\mathrm{d}t} &= \sum_{k\neq j} m_k(x_{ik}-x_{ij})/r_{jk}^3.\end{aligned} \tag{13}$$

By defining the inverse separation distances to be $s_{jk} = 1/r_{jk}$ $(j \neq k)$, we derive the following polynomial formulation of the $N$-body problem:

$$\frac{\mathrm{d}x_{ij}}{\mathrm{d}t} = v_{ij}, \tag{14}$$

$$\frac{\mathrm{d}v_{ij}}{\mathrm{d}t} = \sum_{k\neq j} m_k(x_{ik}-x_{ij})s_{jk}^3, \tag{15}$$

$$\frac{\mathrm{d}s_{jk}}{\mathrm{d}t} = -s_{jk}^3 \sum_{i=1}^{3}(x_{ik}-x_{ij})(v_{ik}-v_{ij}). \tag{16}$$

Even when symmetry is exploited ($s_{jk} = s_{kj}$), we have paid a price for polynomial form; the number of equations $n$ has been increased to $n = 6N + N(N-1)/2$. The gain is that the system now readily admits a Maclaurin-series solution of arbitrary order.

Let $x_{ijl}$ denote the $l$th order coefficient of the Maclaurin series for $x_{ij}$. That is, $x_{ij}(t) = \sum_{l=0}^{m} x_{ijl}t^l$. Let similar notation hold for $v_{ij}$ and $s_{jk}$. Suppose that coefficients are known for terms through order $m-1$. In light of the theoretical considerations mentioned previously, the coefficients for the $m$th term is obtained simply by integrating (14) with respect to $t$, to yield

$$x_{ijm} = v_{i,j,m-1}/m. \tag{17}$$

Similarly, by integrating (15), we obtain

$$v_{ijm} = \sum_{k=1}^{n} m_k \sum_{l=0}^{m-1}(x_{ikl}-x_{ijl})(s^3)_{j,k,m-l-1}/m. \tag{18}$$

The coefficients of nonlinear terms are generated by successive Cauchy products. For example,

$$(s^2)_{jkm} = \sum_{l=0}^{m} s_{jkl}s_{j,k,l-m}, \quad (s^3)_{jkm} = \sum_{l=0}^{m}(s^2)_{jkl}s_{j,k,l-m}. \tag{19}$$

Finally, by defining

$$a_{jkm} = \sum_{l=1}^{m}\sum_{i=1}^{3}(x_{ijl}-x_{ikl})(v_{i,j,m-l}-v_{i,k,m-l}) \tag{20}$$

we obtain from (16)

$$s_{jkm} = -\sum_{l=1}^{m-1}(s^3)_{jkl}a_{j,k,m-l}/m. \tag{21}$$

The essence of the algorithm consists of two procedures inside a time-advancement loop. First, via (17)–(21) the following coefficient vectors and matrices are generated for $l = 1, 2, \ldots, m$: $\mathbf{x}_{jl} = [x_{1jl}, x_{2jl}, x_{3jl}]$,

$\mathbf{v}_{jl} = [v_{1jl}, v_{2jl}, v_{3jl}]$, and $S_l$, whose entries are $s_{jkl}$. Subsequently, the solution is advanced in time merely by evaluating the Maclaurin polynomials. For example, the position update is accomplished by

$$\mathbf{x}_j(\Delta t) = \mathbf{x}_{j0} + \mathbf{x}_{j1}\Delta t + \mathbf{x}_{j2}\Delta t^2 + \cdots + \mathbf{x}_{jm}\Delta t^m. \tag{22}$$

Recall that the update stage is efficiently accomplished by Horner's method.

The algorithm is identical to that given at the end of Section 4 with the exception that the coefficients are generated rather than simply evaluated at each time-step. The generation step requires just 50 lines of computer code and $O[(mN)^2]$ machine operations. In contrast, the Horner update requires but $O(mn) = O(mN^2)$ operations.

Finally, a comment is in order on the relationship between the present method and that of Le Guyader [7]. The methods are similar though not identical. Both exploit polynomial generators, whose forms nevertheless differ. The polynomial form of the present method permits all nonlinear terms of all orders to be generated simply as Cauchy products. In contrast, there appears to be no standard method for the generation of nonlinear terms in [7].

In the next section, we discuss an adaptive time advancement for the Picard-based algorithm of this section.

## 6. Time-step adaptation

A number of venerated theorems exist in the analysis of ODEs that establish the connection between local and global truncation errors. Simply put, controlling local truncation error tends to control global error as well, provided the generator $\mathbf{f}$ satisfies the Lipschitz condition, stated here for autonomous systems.

$$|\mathbf{f}(\mathbf{y}_2) - \mathbf{f}(\mathbf{y}_1)| \leqslant L|\mathbf{y}_2 - \mathbf{y}_1|, \tag{23}$$

where $L$ is a finite constant [4]. If $\mathbf{f}$ is continuously differentiable, as is the case for polynomial generators, then the Lipschitz condition is satisfied as a strict equality with $L = \sup_D |\partial\mathbf{f}/\partial\mathbf{y}|$, where $D$ is the domain [4]. Moreover, for any one-level numerical integration scheme, the cumulative (global) error $E$ for an integration over the time interval $0 \leqslant t \leqslant T$ is at most [4]

$$\frac{\delta}{L}(\mathrm{e}^{LT} - 1) \tag{24}$$

if the relative error per step is at most $\delta$. The linear approximation of (24) suggests the working relationship

$$E \approx \delta T, \tag{25}$$

which is useful provided neither the Lipschitz constant $L$ nor the integration time $T$ is too large.

For reasons to be discussed later, we focus on the velocity components of the system, namely $\mathbf{v}_j$. Let $EV$ be the global error in $\max_j ||\mathbf{v}_j||_2$ and $e_l$ be the relative error during time-step $l$ whose length is $\Delta t_l$. Global error approximation (25) suggests $EV = \sum_{l=1}^{s} e_l$, $s$ being the number of time-steps. Let $\epsilon$ be the global tolerance in velocity error. It follows that $|EV| \leqslant \epsilon$ provided $|e_l| \leqslant \epsilon_l$, where

$$\epsilon_l = \frac{\epsilon \Delta t_l}{T}. \tag{26}$$

This defines the allowable relative error per step, presuming that the velocity components have been scaled by an appropriate reference velocity.

Let $\mathbf{v}_j(t_{l+1})$ be the exact velocity at time $t_l$ and let $\mathbf{w}_j^m(t_{l+1})$ be its $m$th-order Maclaurin polynomial approximation via (22). The magnitude of the error of the step from time $t_l$ to $t_{l+1}$ is $e_l = ||\mathbf{v}_j(t_{l+1}) - \mathbf{w}_j^m(t_{l+1})||_2$, which we approximate using the $m+1$ term of the Maclaurin series. That is,

$$e_l \approx \max_j ||\mathbf{v}_{j,m+1}(t_l)||_2 \Delta t_l^{m+1}. \tag{27}$$

By requiring $e_l < \epsilon_l$, we have

$$\max_j ||\mathbf{v}_{j,m+1}(t_l)||_2 \Delta t_l^m \leqslant \frac{\epsilon \Delta t_l}{T} \tag{28}$$

from which it follows that:

$$\Delta t_l \leqslant \left[ \frac{\epsilon}{\max_j ||\mathbf{v}_{j,m+1}(t_l)||_2 T} \right]^{1/m}. \tag{29}$$

### 6.1. A three-body example

The three-body problem with close encounters is notoriously ill-conditioned because it admits chaotic solutions that manifest extreme sensitivity to initial conditions. The following example demonstrates the power of the present method for an ill-conditioned three-body problem comprised of the earth, its moon, and a "spacecraft".

The masses, initial positions, and initial velocities of the bodies are provided in Table 1. The "spacecraft" is given significant mass so that it too exerts a small but non-negligible gravitational force. Data are only approximately related to realistic values. For all bodies, the initial coordinate $x_3$ and velocity $v_3$ are assumed zero so that the orbits remain co-planar. Masses are scaled relative to the mass of the earth; distances are in earth radii. Velocity is normalized by the orbital velocity at the earth's surface, for which the orbital period is $2\pi$. The integration time ($T = 3200$) is in excess of 500 orbital periods.

Fig. 2 compares time-step adaptations for solutions with $m = 8$, $m = 12$, and $m = 16$, for $T = 3200$. The computations are performed in double precision on a PC for which the machine unit roundoff error $u = 2.2 \times 10^{-15}$. For all three calculations $\epsilon = u$. At $t = 386$ the spacecraft narrowly misses the moon and its orbit is dramatically altered by the slingshot effect, as shown by the inset in the first sub-plot. Subsequent sub-plots for $m = 8$, $m = 12$, and $m = 16$ show only the region of the close encounter. For clarity the plot density is greatly reduced. Symbols show positions at every 80th, every 20th, and every 10th time-steps for the $m = 8$, $m = 12$, and $m = 16$ calculations, respectively. The plots reveal the ability of the current fixed-order scheme to dramatically adapt its time-step during close encounters. Moreover, by comparing the sub-plots associated with differing orders $m$, one gleans an indication of the relationship between time-step and order for fixed global accuracy.

The results of the computations shown in Fig. 2 agree at the final time ($t = T$) to 10 decimal places, as do computations with $m = 10$, 20, 30, 36, and 40. A computation with the same global error tolerance and $m = 5$ agrees with the other results to only 7 digits. Whereas the $m = 5$ computation requires 47,000 time-

Table 1
Masses and initial data for three-body problem

| Body | Mass, $M_j$ | $x_1$ | $x_2$ | $v_1$ | $v_2$ |
|---|---|---|---|---|---|
| Earth | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Moon | 0.0125 | 42.4 | −42.4 | 0.095 | 0.095 |
| Craft | 0.00001 | −1.0 | 0.0 | 0.0 | 1.4034 |

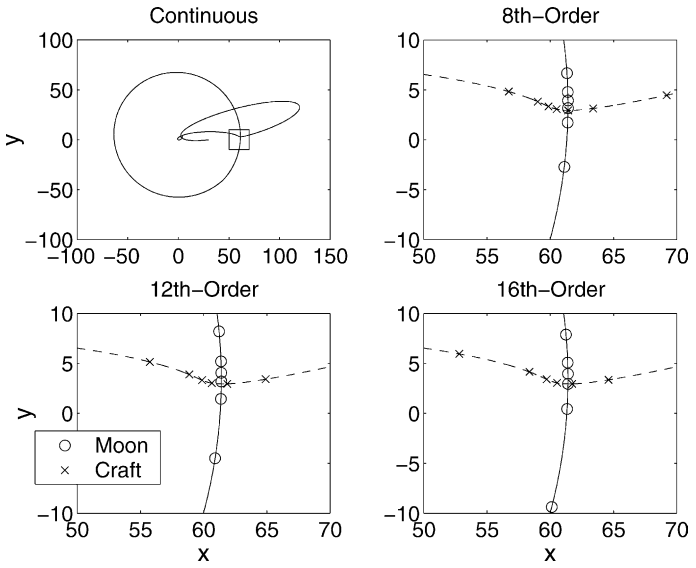


Fig. 2. Positions of Moon and spacecraft showing time-step adaptation during close encounter, for selected orders of Maclaurin polynomial: (continuous) – reference solution with inset of close encounter; (8th-order) – position shown at every 80th time-step; (12th-order) – every 20th time-step shown; (16th-order) – every 10th time-step shown.

steps, the $m = 12$ computation requires just over 500. We infer that, over such long-time integration, there accumulates a 4–5 digit loss due to round-off error for most computations except that with $m = 5$, for which there is an apparent 7-digit loss. These suspicions appear to be confirmed during single precision $\epsilon = u = 2^{-23} \approx 10^{-7}$ calculations which agree at $t = T$ to only two significant digits for $m = 8$, $m = 12$, and $m = 16$, again suggesting a 5-digit loss to round-off error. A reference calculation with the Bulirsch–Stoer method suffers commensurate round-off errors.

## 7. Optimally adapted time advancement

As implied previously, the number of floating-point operations (FLOPS) required per time-step for the $N$-body solver is $\mathrm{O}[(mN)^2]$. More specifically, let $P(m,N)$ be the function that represents the computational cost in FLOPS per time-step of generating the Maclaurin coefficients to order $m$ for $N$ bodies. In recognition that different machines may require differing numbers of clock cycles for addition, subtraction, multiplication, and division, we define

$$P(m,N) = w_a Pa(m,N) + w_s Ps(m,N) + w_m Pm(m,N) + w_d Pd(m,N), \tag{30}$$

where, for example, $w_a$ is the relative cost (computational weight) of addition and $Pa(m,N)$ is the number of additions. Specifically,

$$\begin{aligned} Pa(m,N) &= \frac{5}{4}N(N-1)(m+1)(m+2) + \frac{3}{2}N^2 m(m+1) + \frac{1}{2}N(N-1)m + 6mm + 4N^2 \\ Pd(m,N) &= mN. \end{aligned} \tag{31}$$

Similar to $Pa(m,N)$, $Ps(m,N)$ and $Pm(m,N)$ are each $\mathrm{O}[(mN)^2]$. If the weights are unknown, we simply use the default values, $w_a = w_s = w_m = w_d = 1$. On the other hand, in years past, programmers who wished to

optimize performance on Cray supercomputers, for example, were advised that $w_d = 3w_m$ because division on those platforms was accomplished in three steps: reciprocal approximation, Newton refinement, and multiplication by the reciprocal.

Let us now define $C(m,N)$ to be the computational cost per unit time at time $t_l$; that is,

$$C(m,N) = \frac{P(m,N)}{\Delta t_l}, \tag{32}$$

where $\Delta t_l$ is related to $m$ and $N$ through (29). Because we have chosen to compute $\Delta t_l$ by constraining velocity error, and velocity is a smooth function, $C(m,N)$ is also a smooth function. Our goal now is to choose both $m$ and $\Delta t_l$ so as to minimize $C(m,N)$ at each time-step, while maintaining the requisite tolerance on local truncation error. The obvious approach is to differentiate and minimize $C(m,N)$ with respect to $m$ at each step. This, it turns out, is unnecessary. Fig. 3 compares CPU time, estimated on the basis of the cost function $C$, with actual CPU time for the three-body example, both as functions of order $m$. Here, the global relative error tolerance $\epsilon$ is set at the unit roundoff error for double precision; that is $\epsilon = 2^{-52} \approx 10^{-16}$. The estimated CPU time is simply the number of flops arbitrarily scaled by an average mega-flop (Mflop) rate, in this case 50 MFlops. Because the cost function does not take into consideration how the machine exploits cache, it can be considered only a rough indicator of computational effort. However, from Fig. 3, three things are obvious: (1) the cost function reflects the trends in actual CPU time, (2) the cost function is smooth, and (3) increasing order rapidly diminishes cost to a point, beyond which cost increases very gradually. Consequently, we opt for the simplest possible scheme. As we compute the successive Maclaurin coefficients of ascending orders $k$, we simultaneously evaluate $\Delta t_l$ via (29), $P(m,N)$ by (30) and $C(k,N)$ by (32). If $C(k+1,N) < C(k,N)$ we continue to the next higher order. Otherwise, we stop at order $k$; that is, we set the maximum order $m = k$. For the example shown in Fig. 3, the estimated and actual minima occur at $m = 19$ and $m = 23$, respectively. However, because of the characteristic shape of $C$, there is very little penalty for not precisely hitting the optimal $m$.

If the default values of the weights are used, and if the machine unit roundoff error $u$ is selected as the tolerance (that is, $\epsilon = u$) then the present optimal-order algorithm is entirely parameter free. In Fortran 90, for example, the intrinsic function EPSILON returns the value of $u$. Thus, the parameter-free implementation of the algorithm returns velocities whose relative truncation errors are bounded by machine epsilon.

Fig. 4 compares results of the fixed $m$ and parameter-free versions of the algorithm for the three-body example. Results are shown for both single precision (SP) and double precision (DP) values of $u$, $2^{-23}$ and

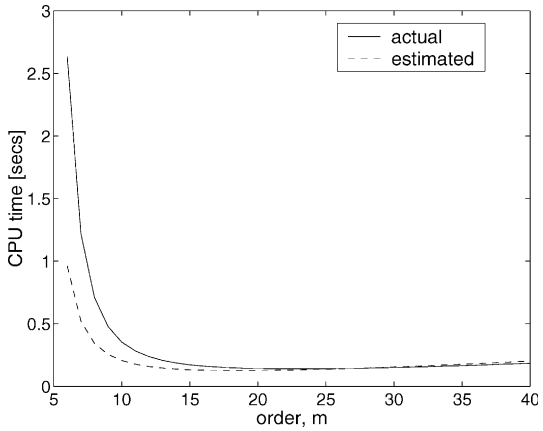


Fig. 3. Actual CPU time in seconds as measured by system clock vs. CPU time estimated from Eqs. (30) and (31), for three-body example with double-precision accuracy.

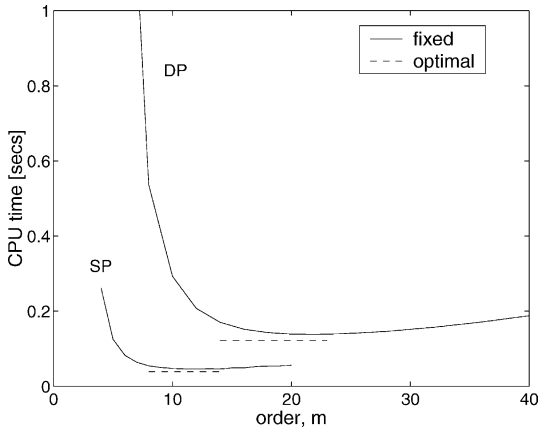


Fig. 4. Actual CPU time for fixed-order algorithm vs. time for dynamic optimal-order algorithm, for three-body example: (solid lines) fixed-order method for single- (SP) or double-precision (DP) relative global error; (dashed lines) optimal-order method, with length of line showing dynamic range of order $m$.

$2^{-52}$, respectively. For single precision, the fixed-order method attains minimum CPU time whenever the order $m = 12$. In contrast, the corresponding minimum for double precision occurs at $m = 21$. For both single and double precision, the parameter-free algorithm beats the best times of the fixed-order method because the optimization is dynamic, as indicated by the lengths of the dashed lines, which show the range of the order $m$ over the duration of the calculation. For the single-precision computation, the order varies from 8 to 14. In contrast, for the double precision result, $m$ varies from 14 to 23. Fig. 4 corroborates the observations of Press and Spergel [12] that increasing order improves efficiency, but only to a point, and that orders higher than four are desirable.

### 7.1. Practical considerations

The current algorithm minimizes CPU time and maximizes accuracy at the expense of storage. Storage intensive, the algorithm requires memory for $m + 1$ vectors for the Maclaurin coefficients of $n$ state variables, relative to, for example, four vectors for the classic fourth-order Runge–Kutta scheme. Moreover, for the parameter-free algorithm in which $m$ floats, the actual storage is not known a priori. A natural way in which to deal with this uncertainty is to use dynamic memory allocation in conjunction with linked lists, which both *C* and Fortran 90 support. As a linked list, the storage vector for coefficients of order $k$ incorporates a pointer to the vector for order $k + 1$. Thus, memory can be both allocated or deallocated as the method adjusts the order.

For some purposes (graphical, for example) the coarse time-steps permitted by a high-order algorithm may be undesirable. For these purposes, fine-grained results are efficiently generated directly from the Maclaurin polynomial simply by evaluating it at several intermediate times that range over the full time interval $\Delta t_l$. Because the global operation count of the algorithm is dominated by generating the Maclaurin coefficients, additional Horner updates do not add substantially to the computational effort.

### 7.2. An N-body example: evolution of the solar system

Here we consider the present method for evolution of the solar system. For reference, we also provide a reference solution computed by the Bulirsch–Stoer method. The reference method, which combines Richardson extrapolation with a second-order modified midpoint method [13], represents the de facto state-

of-the-art for classical ODE integrators [13]. Initial data are taken from *Ephemerides of Minor Planets* [6] for Epoch 1997, December 18.0 (Julian Date). The sun and the nine planets without moons are considered; thus $N = 10$. Cartesian position coordinates $(x, y, z)$ are given in astronomical units [AU] referenced from the center of the Sun, which is located at the origin. The first two coordinates are in the nominal plane of the ecliptic. The third coordinate $z$ is normal to the ecliptic. The natural time unit [NTU] is mean Earth years (365.25689833 Earth days) divided by $2\pi$. Velocities are given in [6] in AU/day and are converted in the program to AU/NTU. The period of integration is 1000 NTUs, roughly 159 Earth years, which is just short of the period of Neptune. There are no relativistic corrections for the orbit of Mercury.

Because of periodic orbits without close encounters, the problem of evolving the solar system is benign relative to the three-body example studied previously. Both the present method and the Bulirsch–Stoer reference performed well. The parameter-free modified Picard method worked as expected the first time: successfully to completion, accurately, and without adjustment. The Bulirsch–Stoer method required modest experimentation before an appropriate error tolerance and normative time-step were found. Twice the reference method failed, for different reasons, when the tolerance was too restrictive. The methods agreed to at least nine significant digits in all three position coordinates of all 10 bodies at the final time. The Bulirsch–Stoer method maintained constant angular momentum of the entire system to 12 digits; the parameter-free Picard method to 14 digits. The one-parameter ($\epsilon$) Picard method with $\epsilon = 10^{-6}$ matched the accuracy of the optimally tuned Bulirsch–Stoer algorithm. However, at equivalent accuracy it ran considerably slower. On the same platform (an SGI Octane with a single 270 MHz IP30 processor), Bulirsch–Stoer ran to completion at 8520 time-steps in 138 s (including output). The present method took 11446 time-steps in 970 s (also including output). For the present method, the average time-step was .087 NTUs, or approximately 5 Earth days. For this benign problem there was relatively little adjustment in time increments. For $\epsilon = 10^{-6}$, the order $m$ remained nearly constant at 13, with infrequent excursions to as low as 9 and as high as 14.

### 7.3. Gravitational collapse of an N-body system

A final test of the present method, the partial gravitational collapse of an $N$-body system, is far less benign. We consider a sphere radius $r = N^{1/3}$ in which $N$ particles are randomly distributed in position with random initial velocities. Specifically, the Euclidean norm of the position vector of each particle is a random variable uniformly distributed on $[0, r]$, and the particle's speed is uniformly distributed on $[0, 1]$. The directions of position and velocity vectors are assigned in spherical coordinates by angles $\phi$ and $\theta$ uniformly distributed on $[0, \pi]$ and $[0, 2\pi]$, respectively. Such systems tend to eject "hot" particles followed by successive collapses. The greater $N$ or the smaller $r$, the greater the probability of a direct collision in finite time. Near misses are the norm; as a result, some particles exhibit chaotic trajectories. In general, such problems severely challenge adaptive integrators.

Here we compare the present (Picard) method against the Bulirsch–Stoer reference, for gravitating systems of $N = 2, 4, 8, 16$, and 32 bodies, each of mass unity. The specific initial conditions (in rectangular Cartesian coordinates) for the $N = 32$ case are attached as Table 3 in Appendix C. The results are summarized in Table 2 below. The integration time is $0 \leqslant t \leqslant 0.5$. At the final time, the $N = 16$ solutions agree to 8–9 significant digits and the $N = 32$ solutions agree to 4–5 significant digits. We attribute the discrepancy to the round-off errors of both methods. For large $N$, some numerical experimentation is required to avoid initial conditions that result in direct collisions within the time span of interest. The Bulirsch–Stoer algorithm (with polynomial rather than rational polynomial extrapolation) is taken directly from *Numerical Recipes* [13] without modification. For roughly equivalent accuracy ($\epsilon = 10^{-14}$ for both methods), the Bulirsch–Stoer algorithm runs approximately 10 times faster than the present method. However, it is clear that the method struggles with near collisions. The authors of *Numerical Recipes* report, "Keep a watch for failed steps. If these are not rare, then Bulirsch–Stoer is in trouble". As shown

Table 2
Current method vs. reference (Bulirsch–Stoer) method for gravitating $N$-body systems with random initial positions and velocities

| $N$ | Picard | | B–S, $\epsilon = 10^{-14}$ | | B–S, $\epsilon = 10^{-15}$ | |
|---|---|---|---|---|---|---|
| | CPU (s) | $m_{\max}$ | CPU (s) | OK/BAD | CPU (s) | OK/BAD |
| 2 | .03 | 17 | .003 | 3/0 | .004 | 4/0 |
| 4 | .23 | 19 | .025 | 4/2 | .050 | 6/2 |
| 8 | 6.2 | 21 | .617 | 20/10 | 1.21 | 35/22 |
| 16 | 402. | 24 | 41.8 | 484/293 | 303. | 4140/4393 |
| 32 | 1847. | 25 | 180. | 570/365 | Underflow | Failed |

in Table 2, the ratio of failed (BAD) to successful (OK) steps grows quickly with $N$. It is interesting to note that, for a slightly more stringent error tolerance of $\epsilon = 10^{-15}$, the number of failed steps exceeds successful ones for $N = 16$ and the reference method experiences an underflow error in step size $h$ for $N = 32$. In contrast, the present method continues to perform at $\epsilon = u = 2^{-52}$, with only a modest increase in CPU time. An indication of the chaoticity of the system is shown in Fig. 5, which shows the trajectory of particle 18 for the $N = 32$ integration.

A drawback of the present method is that it is memory intensive, requiring $\approx 3/2N^2 m_{\max}$ memory locations, relative to $12N$ for the reference method. In addition to CPU time, Table 2 presents the maximum order $m_{\max}$ for the optimal-order variant of the current method. High memory demands for the $N$-body problem are an indirect consequence of the requirement that the generator be of polynomial form.

In summary, some advantages of the current algorithm relative to existing ones are its extraordinary accuracy, its simplicity, the freedom from arbitrary parameters, its robustness, and its built-in interpolation feature. On the other hand, the method is admittedly memory-intensive and typically slower than the Bulirsch–Stoer algorithm, the state-of-the-art for general systems of ODEs. The Picard-based method is at its best for problems in which drastically adaptive stepping is necessary. Present results confirm Makino's assertion [8] that optimal order depends both upon $N$ (Table 2) and accuracy (Fig. 4). To that we would add, based upon all tests cases and Fig. 4, that optimal order is a dynamic function of the instantaneous configuration, with close encounters favoring higher order as observed by Makino.

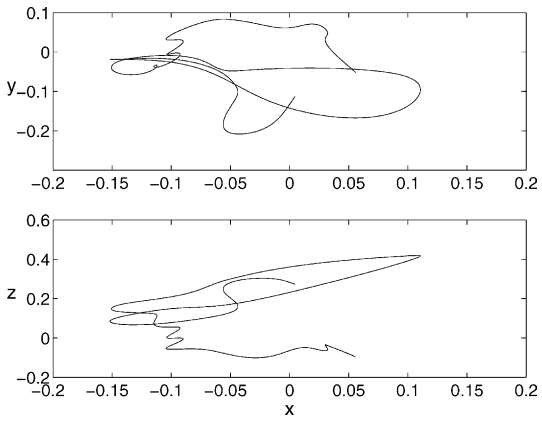


Fig. 5. Position coordinates of chaotic trajectory of particle 18 for gravitating system of $N = 32$, showing effects of near collisions.

Table 3
Random initial conditions for gravitating system with $N = 32$

| $x_1$ | $x_2$ | $x_3$ | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|---|---|
| −0.451781 | −0.668711 | 2.130823 | −0.390649 | 0.170636 | −0.139339 |
| 0.139627 | 2.417780 | −0.184771 | 0.059581 | 0.064517 | 0.278170 |
| −0.012178 | 0.005604 | 3.094713 | 0.002131 | 0.000994 | 0.523462 |
| 0.957195 | −1.649651 | −1.648189 | 0.483751 | −0.143965 | −0.450655 |
| 1.832367 | −2.058355 | −1.196428 | −0.005011 | 0.004957 | 0.028043 |
| −1.323243 | 0.092987 | 1.958540 | −0.304847 | −0.254990 | −0.331886 |
| 2.619225 | −0.531646 | −1.001710 | 0.049874 | 0.029157 | −0.094518 |
| 0.304035 | −0.493170 | 0.062824 | 0.083240 | 0.083887 | −0.017629 |
| 1.761668 | 1.260788 | −1.938820 | −0.013529 | −0.003297 | −0.510776 |
| −0.010658 | −0.013185 | 0.575171 | −0.296793 | 0.022484 | 0.041705 |
| −0.031451 | −0.072838 | −1.528530 | −0.116508 | 0.019669 | 0.363987 |
| −0.110800 | 0.268477 | 0.510285 | 0.624336 | 0.021235 | 0.642547 |
| 0.359750 | −0.279393 | 1.018212 | −0.041297 | −0.073610 | −0.612759 |
| 0.086059 | 0.169721 | −0.086404 | −0.602516 | 0.048723 | 0.463746 |
| −0.094330 | −0.004586 | 2.926841 | 0.100003 | 0.013467 | −0.297018 |
| 0.023506 | −1.690861 | 1.241850 | −0.597471 | 0.093746 | 0.015160 |
| −0.429119 | 0.017477 | 0.442344 | −0.001187 | 0.109198 | 0.407929 |
| 0.056293 | −0.054437 | −0.100358 | 0.008737 | 0.177400 | 0.371940 |
| 0.147657 | −0.220578 | 0.563181 | −0.090589 | −0.031124 | −0.793053 |
| −0.228752 | 1.036050 | 2.857207 | 0.032978 | 0.000263 | 0.714455 |
| −0.494299 | 0.998871 | 1.570144 | 0.031275 | 0.239224 | −0.349678 |
| 0.249536 | 0.849006 | −0.812883 | −0.097460 | 0.714458 | −0.455581 |
| 0.845137 | −0.689624 | 1.870891 | −0.007234 | 0.002523 | −0.024845 |
| −0.693832 | −0.071945 | 3.039076 | −0.672755 | −0.002692 | −0.003527 |
| −0.351586 | 0.158041 | −0.278988 | 0.001492 | −0.039310 | −0.668484 |
| −0.029971 | 0.663645 | −1.956217 | 0.027927 | −0.015388 | 0.249257 |
| −0.664721 | −2.549115 | 0.657397 | 0.327532 | 0.161669 | 0.044897 |
| −0.480630 | −0.226497 | 0.104536 | −0.105470 | −0.057948 | −0.135116 |
| 1.206708 | −0.660459 | 2.342273 | 0.092884 | 0.195367 | 0.458886 |
| 2.026496 | 0.609431 | 2.289365 | 0.207987 | −0.306392 | −0.210477 |
| −0.195273 | 0.012207 | −0.066007 | −0.667363 | −0.207305 | 0.144972 |
| 0.030465 | 0.050829 | 0.016035 | −0.024180 | −0.040752 | 0.048022 |

Although we have focused on the $N$-body problem, the method extends naturally to any system of ODEs whose generator $\mathbf{f}$ can be written in polynomial form. Because the coefficients of nonlinear terms of high degree can be computed by successive Cauchy products of lower order coefficients, the operation count for nonlinear problems scales as $nm^2 \log_2 p$, where $n$ is the number of equations, $m$ is the order of the Maclaurin polynomial, and $p$ is the polynomial degree of the generator. As a practical consequence, the algorithm is effectively quadratic in $m$ for *all* nonlinear systems, in which case the number of operations is easily counted or estimated. However, the intermediate computations necessary to evaluate high-degree terms may require additional storage.

## 8. Conclusions

The present paper adapts the work of Parker and Sochacki [11] to the solution of the classical $N$-body problem. The solution method is based upon modified Picard iteration, which generates successive approximations of the Maclaurin series of the solution to arbitrarily high order, provided that the right-

hand side (the generator) can be cast in polynomial form. The method is well-suited to the simultaneous optimization of time-step and order. An adaptive, optimal-order variant of the algorithm reveals that optimal order is inherently dynamic, dependent upon the number of bodies $N$, the specified error global tolerance $\epsilon$, and the instantaneous configuration. Advantages of the method are its simplicity, its robustness, its extraordinary accuracy, that it inherently contains a built-in high-order method for interpolation between coarse time-steps, and that it can be rendered completely parameter free. Its disadvantages include computational inefficiency relative to standards such as Bulirsch–Stoer, and that it is memory intensive.

Despite its relative inefficiency, the current algorithm is well-suited for certain applications, among these accurate integration of chaotic $N$-body systems, where close encounters are the norm, and as a platform for numerical experiments on optimal order. Because of its robustness and freedom from arbitrary parameters, the method could be adapted to hybrid $N$-body integrators, where, for example, a more efficient method is used until a close encounter, at which time the scheme hands off to the current method for negotiating a difficult passage.

The robustness and accuracy of the method are demonstrated for both benign (e.g., the solar system) and chaotic $N$-body systems.

The inefficiency of the method stems from the fact that the operation count scales as $m_{\max}^2 N^2$, where $m_{\max}$ is the maximum order of the Maclaurin polynomial approximation. Order $N^2$ computational effort is normative for $N$-body integrators (relative to $N$-body simulators). However, because $m_{\max}$ can be quite high, say 25 or more for double-precision accuracy, it is desirable to explore methods for reduction of order. One possibility is consideration of rational polynomial generators, which should be better at representing near singularities. In addition to further exploration of this possibility, future efforts include the development and implementation of dynamic memory allocation and deallocation within the algorithm, and adaptations suitable for parallel processing.

Finally, although the solution method was developed expressly for the $N$-body problem, it is applicable to a wide class of ordinary differential equations of initial-value type.

## Acknowledgements

The authors are extremely grateful to their mathematics colleagues J.S. Sochacki, G.E. Parker, D.C. Carothers, and P.G. Warne, and their physics colleagues W.H. Ingham and D.W. Peterson, for numerous clarifications and insights. Moreover, the timely references and insightful suggestions of the reviewers are also greatly appreciated.

## Appendix A. Maple code for predator–prey ODE

The Maple code given below generates Maclaurin coefficients for the classical predator–prey program. By appropriately changing the number of equations $n$ and modifying the generator **rhs_f**, the code can be readily adapted to any autonomous polynomial first-order system.

$p$ = number of modified Picard iterates
$n$ = number of equations
$C[i,j]$ = storage array for Maclaurin coefficients

```
> restart: with(linalg):
> p := 8; n := 2;
> y := array(1..n); C := array(1..n,0..p);
```

Define (autonomous) generator, $n$-vector function $\boldsymbol{f}$.

```
> rhs_f := [ a*y[1] - b*y[1]*y[2],
>          -c*y[2] + d*y[1]*y[2] ];
> f := unapply(rhs_f,y);
```

Establish (generic) initial condition ($i = 0$) for Picard iteration.

```
> y0 := evalm(y);
> for i from 1 by 1 to n do
>   C[i,0] := y0[i];
> od;
```

Create linear (Euler) approximation to introduce $t$ explicitly.

```
> y_n := evalm(y0 + f(y0)*t);
> for i from 1 by 1 to n do
>   C[i,1] := coeff(y_n[i],t,1);
> od;
```

Do $p$ modified Picard iterates; build Maclaurin series term-by-term.

```
> for j from 1 by 1 to p-1 do
>   temp := simplify(f(y_n));
>   k := j + 1;
>   for i from 1 by 1 to n do
>     C[i,k] := simplify(coeff(temp[i],t,j) / k);
>     y_n[i] := simplify(y_n[i] + C[i,k]*t^k);
>   od:
> od;
```

Convert coefficient matrix C to optimized Fortran source code.

```
> fortran(C,optimized,precision=double,filename="mat_c.h");
```

## Appendix B. Fortran code for modified Picard method

### *B.1. Generic module for evaluating Maclaurin coefficients*

Current versions of Maple generate Fortran 77 code but not Fortran 90 code. Provided below is a generic Fortran 77 subroutine in which to imbed the Fortran 77 output of the Maple code provided in Appendix B:

```
      SUBROUTINE matrix_C (n, m, y, C)
        DOUBLE PRECISION y(1:n), C(1:n,0:m)
c
c Parameter definition here; parameters shown for predator–prey example
c
        c1 = 0.9D0
        c2 = 1.1D0
        c3 = 1.0D0
        c4 = 1.0D0
c
c Include Maple-generated Fortran 77 here
c
        INCLUDE 'mat_C.h'
c
        RETURN
      END SUBROUTINE matrix_C
```

### *B.2. Time advancement via Horner's method*

Fortran 90 code for advancing the solution $\mathbf{y}(t)$ forward in time from initial state $\mathbf{y}^0$ via Horner's method is provided below. Here, the time increments are constant, but they need not be.

```
DO time_step = 1, max_steps
  CALL matrix_C (n, m, y0, C) ! y0 is initial state
  time = time_step * dt
  y = C(1:n,order) ! accumulate in highest-order register
  DO j = order-1, 0, -1 ! Horner's method for state vector y
    y = dt*y + C(1:n,j)
  END DO
  IF (mod(time_step,skip) = = 0) WRITE (*,*) time, y
  y0 = y ! reset initial state and start over
END DO
```

If the programs containing the code above are entitled **main.f90** and **subs.f**, respectively, the following UNIX script accomplishes compilation and linking to create the executable file **run.x**:

```
f77 -c subs.f
f90 -c main.f
f90 subs.o main.o -o run.x
```

## Appendix C. Initial conditions for gravitating *N*-body system

See Table 3.

## References

[1] S.J. Aarseth, Direct methods for *N*-body simulations, in: J.U. Brackbill, B.I. Cohen (Eds.), Multiple Time Scales, Academic Press, New York, 1985, pp. 377–418.

[2] U. Becciani, V. Antonuccio-Delogu, M. Gambera, A modified parallel tree code for $N$-body simulation of the large-scale structure of the universe, J. Comput. Phys. 163 (2000) 118–132.
[3] E. Bertschinger, Simulations of structure formation in the universe, Annu. Rev. Astron. Astrophys. 36 (1998) 564–599.
[4] G. Birkhoff, G.-C. Rota, Ordinary Differential Equations, Ginn and Company, New York, 1962.
[5] D.C. Carothers, Some results on systems of polynomial differential equations and projectively polynomial functions, Presented at the 20th Annual Southeastern-Atlantic Regional Conference on Differential Equations, Virginia Tech, October 20–21, 2000.
[6] Ephemerides of Minor Planets, Institute of Applied Astronomy of the Russian Academy of Sciences, St. Petersburg, 1997, p. 148.
[7] Cl. Le Guyader, Solution of the $N$-body problem expanded into Taylor series of high orders. Applications to the solar system over large time range, Astronomy Astrophys. 272 (1993) 687–694.
[8] J. Makino, Optimal order and time-step criterion for adaptive Aarseth-type $N$-body integrators, Astrophys. J. 369 (1991) 200–212.
[9] J.J. Monaghan, Particle methods for hydrodynamics, Comput. Phys. Rep. 3 (1985) 71–124.
[10] A. Morbidelli, Modern integrations of solar system dynamics, Annu. Rev. Earth Planet. Sci. 30 (2002) 89–112.
[11] G.E. Parker, J.S. Sochacki, Implementing the Picard iteration, Neural, Parallel, Sci. Comput. 4 (1996) 97–112.
[12] W.H. Press, D.N. Spergel, Choice of order and extrapolation method in Aarseth-type $N$-body algorithms, Astrophys. J. 325 (1988) 715–721.
[13] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in Fortran 77, second ed., The Art of Scientific Computing, Cambridge University Press, Cambridge, 1992.
[14] P. Saha, S. Tremaine, Long-term planetary integration with individual time steps, Astronomical J. 108 (1994) 1962–1969.
[15] W.L. Sweatman, The development of a parallel $N$-body code for the edinburgh concurrent supercomputer, J. Comput. Phys. 111 (1994) 110–119.
[16] J. Wisdom, M. Holman, Symplectic maps for the $N$-body problem, Astronomical J. 102 (1991) 1528–1538.