Computer Physics Communications 185 (2014) 1461–1472

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Automatic code generator for higher order integrators[☆]

Asif Mushtaq[a], Kåre Olaussen[b,*]

[a] Institutt for Matematiske Fag, NTNU, N-7491 Trondheim, Norway
[b] Institutt for Fysikk, NTNU, N-7491 Trondheim, Norway

ARTICLE INFO

Article history:
Received 4 October 2013
Received in revised form
20 December 2013
Accepted 20 January 2014
Available online 30 January 2014

Keywords:
Splitting methods
Modified integrators
Higher order methods
Automatic code generation

ABSTRACT

Some explicit algorithms for higher order symplectic integration of a large class of Hamilton's equations have recently been discussed by Mushtaq et al. Here we present a Python program for automatic numerical implementation of these algorithms for a given Hamiltonian, both for double precision and multi-precision computations. We provide examples of how to use this program, and illustrate behavior of both the code generator and the generated solver module(s).

Program summary

Program title: HOMsPy: Higher Order (Symplectic) Methods in Python

Catalogue identifier: AESD_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AESD_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html

No. of lines in distributed program, including test data, etc.: 19423

No. of bytes in distributed program, including test data, etc.: 1970283

Distribution format: tar.gz

Programming language: Python 2.7.

Computer: PCs or higher performance computers.

Operating system: Linux, MacOS, MSWindows.

RAM: Kilobytes to a several gigabytes (problem dependent).

Classification: 4.3, 5.

External routines: SymPy library [1] for generating the code. NumPy library [2], and optionally mpmath [3] library for running the generated code. The matplotlib [4] library for plotting results.

Nature of problem:
We have developed algorithms [5] for numerical solution of Hamilton's equations.

$$\dot{q}^a = \partial H(\boldsymbol{q}, \boldsymbol{p})/\partial p_a a, \qquad \dot{p}_a = -\partial H(\boldsymbol{q}, \boldsymbol{p})/\partial q^a, \quad a = 1 \ldots N \tag{1}$$

for Hamiltonians of the form

$$H(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + V(\boldsymbol{q}) = (1/2)\boldsymbol{p}^T \boldsymbol{M}\boldsymbol{p} + V(\boldsymbol{q}), \tag{2}$$

with M a symmetric positive definite matrix. The algorithms preserve the symplectic property of the time evolution exactly, and are of orders τ^N (for $2 \le N \le 8$) in the timestep τ. Although explicit, the algorithms are time-consuming and error-prone to implement numerically by hand, in particular for larger N.

[☆] This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

[*] Corresponding author. Tel.: +47 73593652; fax: +47 73593372.
E-mail addresses: Asif.Mushtaq@math.ntnu.no (A. Mushtaq), Kare.Olaussen@ntnu.no (K. Olaussen).

0010-4655/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cpc.2014.01.012

Solution method:
We use computer algebra to perform all analytic calculations required for a specific model, and to generate the Python code for numerical solution of this model, including example programs using that code.

Restrictions:
In our implementation the mass matrix is assumed to be equal to the unit matrix, and $V(q)$ must be sufficiently differentiable.

Running time:
Subseconds to eons (problem dependent). See discussion in the main article.

References:

[1] SymPy Development Team, http://sympy.org/.
[2] NumPy Developers, http://numpy.org/.
[3] F. Johansson et al., Python library for arbitrary-precision floating-point arithmetic, http://code.google.code/p/mpmath/ (2010).
[4] J.D. Hunter, Matplotlib: A 2D graphics environment, Computing in Science and Engineering 9, 90–95 (2007).
[5] A. Mushtaq, A. Kværnø, K. Olaussen, Higher order Geometric Integrators for a class of Hamiltonian systems, International Journal of Geometric Methods in Modern Physics, vol. 11, no. 1 (2014), 1450009-1–1450009-20. DOI: http://dx.doi.org/10.1142/S0219887814500091. arXiv.org:1301.7736.

1. Introduction

The Hamilton equations of motion (1) play an important role in physics and mathematics. They often require numerical methods to compute a solution [1–3]. A well-behaved class of such methods are the *symplectic solvers*, which preserve symplecticity of the time evolution exactly. One simple way to construct a symplectic solver is to split the time evolutions into *kicks*,

$$\dot{q}^a = 0, \qquad \dot{p}_a = -\frac{\partial V(\boldsymbol{q})}{\partial q^a}, \tag{3}$$

which is straightforward to integrate to give

$$q^a(t+\tau) = q^a(t), \tag{4}$$

$$p_a(t+\tau) = p_a(t) - \tau\frac{\partial V(\boldsymbol{q}(t))}{\partial q^a}, \tag{5}$$

followed by *moves*,

$$\dot{q}^a = \frac{\partial T(\boldsymbol{p})}{\partial p_a} = \sum_b M^{ab}p_b, \qquad \dot{p}_a = 0. \tag{6}$$

The final scheme can be written as

$$q^a(t+\tau) = q^a(t) + \tau\sum_b M^{ab}p_b(t+\tau),$$

$$p_a(t+\tau) = p_a(t) - \tau\frac{\partial V(\boldsymbol{q}(t))}{\partial q^a}.$$

This scheme was already introduced by Newton [4] (as more accessible explained by Feynman [5]). A symmetric scheme can be constructed by performing a *kick* of size $\frac{1}{2}\tau$, a *move* of size τ, and a *kick* of size $\frac{1}{2}\tau$ (and repeating). This is often referred to as the Störmer–Verlet method [6,7]; it has a local error of order τ^3. The solution provided by this method can be viewed as the exact solution of a slightly different Hamiltonian system, with a Hamiltonian H_{SV} which differ from (2) by a term proportional to τ^2. For this reason the scheme respects long-time conservation of energy to order τ^2. It will also exactly preserve conservation laws due to Nöther symmetries which are common to $T(\boldsymbol{p}) = \frac{1}{2}\boldsymbol{p}^T\boldsymbol{M}\boldsymbol{p}$ and $V(\boldsymbol{q})$, like momentum and angular momentum which are often preserved in physical models [8].

Recently Mushtaq et al. [9,10] proposed some higher order extensions of the Störmer-Verlet scheme. These extensions are also based on the *kick–move–kick* idea, but with modified Hamiltonians,

$$H_1 \equiv T_{\text{eff}} = \frac{1}{2}\boldsymbol{p}^T M\boldsymbol{p} + \sum_{k\geq 1} T_{2k}(\boldsymbol{q},\boldsymbol{p}), \tag{7a}$$

$$H_2 \equiv V_{\text{eff}} = V(\boldsymbol{q}) + \sum_{k\geq 1} V_{2k}(\boldsymbol{q}), \tag{7b}$$

where T_{2k} and V_{2k} are proportional to τ^{2k}. I.e., the proposal is to replace $V(\boldsymbol{q})$ in Eq. (3) by $V_{\text{eff}}(\boldsymbol{q})$, and $T(\boldsymbol{p})$ in Eq. (6) by $T_{\text{eff}}(\boldsymbol{q},\boldsymbol{p})$. The goal is to construct V_{eff} and T_{eff} such that the combined *kick–move–kick* process corresponds to an evolution by a Hamiltonian H_{eff} which lies closer to the Hamiltonian H of Eq. (2). The difference being of order τ^{2N+2} when summing terms up to $k = N$ in Eqs. (7).

One problem with this approach is that T_{eff} in general will depend on both $\boldsymbol{q}$ and $\boldsymbol{p}$; hence the *move*-steps of Eq. (6) can no longer be integrated explicitly. To overcome this problem we introduce a generating function [3]

$$G(\boldsymbol{q},\boldsymbol{P};\tau) = \sum_{k\geq 0} G_k(\boldsymbol{q},\boldsymbol{P})\,\tau^k \tag{8}$$

such that the transformation $(\boldsymbol{q},\boldsymbol{p}) \to (\boldsymbol{Q},\boldsymbol{P})$ defined by

$$p_a = \frac{\partial G}{\partial q^a}, \tag{9a}$$

$$Q^a = \frac{\partial G}{\partial P_a}, \tag{9b}$$

preserves the symplectic structure exactly, and reproduce the time evolution generated by T_{eff} to order τ^N. Here Q^a is shorthand for $q^a(t+\tau)$, and P_a shorthand for $p_a(t+\tau)$. Eq. (9a) is implicit and in general nonlinear, but the nonlinearity is of order τ^3 (hence small for practical values of τ). In the numerical code we solve (9a) by straightforward iteration (typically two to four iterations in the examples we have investigated).

The rest of this paper is organized as follows: In Section 2 we introduce compact notation in which we present the general explicit expressions for $T_{\text{eff}}(\boldsymbol{q},\boldsymbol{p})$, $V_{\text{eff}}(\boldsymbol{q})$, and $G(\boldsymbol{q},\boldsymbol{Q})$. Because of their compactness these expressions are straightforward to implement in SymPy.

In Section 3 we provide examples of how to use the code generator on specific problems. This process proceeds through two