Proceedings of the International MultiConference of Engineers and Computer Scientists 2015 Vol I,
IMECS 2015, March 18 - 20, 2015, Hong Kong

Scientific Computing with HOMsPy

Asif Mushtaq, Anne Kværnø, and Kåre Olaussen

***Abstract*—The development of HOMsPy is motived by our own research work. This program solves the Hamiltonian problems by proposed *kick-move-kick* higher order scheme with an automatically generated numerical solver. These proposed systematic algorithms are extensions of the Störmer-Verlet method and increases the accuracy of the integration for a large class of Hamiltonian systems.**

***Index Terms*—HOMsPy, Störmer-Verlet, code-generation, Hamilton-equations, symplecticity.**

I. Introduction

HIGHER Order Methods in Python (HOMsPy), is a collection of Python routines designed to generate numerical code for solving the differential equations generated by Hamiltonian of the form,

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

These methods preserve the symplectic property exactly. The main goal of this program is to provide a framework for solving the Hamilton's equations by some higher order symplectic algorithms proposed in [1], [2], using a symbolic program, published in [3], which automatically constructs the numerical solver for each specific Hamiltonian problem. The implemented symplectic schemes are based on extensions of the Störmer-Verlet method. Explicit implementation of the numerical code for a specific potential may be rather laborious and erroneous to do by hand, since repeated differentiation (with respect to many variables) and multiplication by lengthy expressions are often involved.

We have therefore written a code-generating program using the `sympy` symbolic manipulation package. This takes a given potential V as input, performs all the necessary algebra symbolically, and automatically writes a python module for solving one full timestep τ to the higher order (or selected order) of accuracy. The program also writes a runfile example (driver module) which demonstrates how the solver module can be used.

II. Important features of the program

Important aspects of this program are listed below:

- The program can handle Hamiltonian problems of the form
$$H(\boldsymbol{q},\boldsymbol{p}) = T(\boldsymbol{p}) + V(\boldsymbol{q}), \tag{2}$$
where $T(\boldsymbol{p}) = \frac{1}{2}\boldsymbol{p}^{\boldsymbol{T}}\boldsymbol{p}$ is the kinetic term and $V(\boldsymbol{q})$ is potential term. This program is very efficient for a large class Hamiltonian where potential term is sufficiently differentiable.
- The program has options for generating double-precision (DP) and/or multi-precision (MP) solver(s) for method(s) up to eight order in the timestep τ, and provides an automatic code generating environment.
- For a given Hamiltonian, a set of driver modules are automatically generated. In these, several parameters are given default values to this program. Further, initial values are generated randomly. It is straightforward for the user to change those.

Manuscript received December 03, 2014.

A. Mushtaq is with the Department of Mathematical Sciences, NTNU, N-7048 Trondheim, Norway. e-mail: Asif.Mushtaq@math.ntnu.no.

A. Kværnø is with the Department of Mathematical Sciences, NTNU. e-mail: Anne@math.ntnu.no.

K. Olaussen is with the Department of Physics, NTNU. e-mail: Kare Olaussen@ntnu.no.

III. Installation and configuration

This program, as well as the generated solver and driver modules, is written in the Python programming language, using the `sympy` [4], `numpy` [5], and optionally `mpmath` [6] libraries. In addition `matplotlib` [7] is used for plotting. In this section the focus of discussion will be the installation and configuration of necessary softwares as well as HOMsPy. For the basic understanding on scientific computations in Python and for general installation in detail, see [8].

A. Prerequisites

We have used Python version 2.7.x, including the packages `sympy`, `numpy`, `mpmath` (for multi-precision calculations), and `matplotlib` for all development and testing. We have registered that incompatible combinations of these packages can lead to problems.

B. Installation of HOMsPy

- Download `aesd_v1_0.tar.gz` or later versions from
http://cpc.cs.qub.ac.uk/summaries/AESD_v1_0.html,
or ask the author for a copy of HOMsPy.tar.
- HOMsPy contains three subdirectories:

  kimoki: The directory containing the code generating module.
  examples: A directory containing a single file, named as *makeExamples.py*. By running *makeExamples.py* eight new files will be generated, four solver modules (VibratingBeam.py, AnharmonicOscillator.py, AnharmonicOscillatorMP.py, TwoDPendulumMP.py), and four runfile examples known as driver modules (runVibratingBeam.py, runAnharmonicOscillator.py, runAnharmonicOscillatorMP.py, runTwoDPendulumMP.py). By running each runfile example two .png plots will be generated, <example> _ soln.png and <example> _EgyErr.png. Each runfile example will also generate several intermediate Python pickle (.pkl) files. These can normally be deleted after use.
  demo: This directory demonstrates how the examples directory should look like after running *makeExamples.py*, and the runfile examples. Note that the figures

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

will not look identical, because the examples are solved with random initial conditions. This directory in addition contains six .log-files with output which is normally printed to screen. These files contains information about how long it takes to run the various programs.

IV. Using HOMsPy on a simple pendulum problem

Let us start with a simple illustrative example. Consider the pendulum problem defined by the Hamiltonian

$$H(q,p) = \frac{1}{2}p^2 - \cos(q). \tag{3}$$

The corresponding set of ODEs are

$$\dot{q} = p, \quad \dot{p} = -\sin(q). \tag{4}$$

User can include the following code snippet in *makeExamples.py* to get the solver as well as driver modules. One can opt the following procedure in oder to solve the pendulum problem by HOMsPy:

- First write a function specifying the Hamiltonian, giving symbolic names for the coordinates (q) and momentum (p) to be used (and optionally also additional parameters), and the symbolic expression of the potential V. In the present example there is just one coordinate and one momentum involved, with $V = -\cos(q)$. A code snippet for the pendulum problem is shown below, with the potential defined on line 9.

Creating a module for solving a pendulum problem

```
from sympy import cos
def makePendulum():
    # Choose names of coordinates
    q, p = sympy.symbols(['q', 'p'])
    qvars= [q]; pvars  = [p]
    # Define the potential terms
    V = -cos(q)
    # Code for DP solver
    kimoki.makeModules('Pendulum',
                V, qvars, pvars)
```

Note that the `sympy` versions all the functions which occur in V must be known to the code generating program, since it must compute (higher order) symbolic derivatives of V. This is why one has to import the `cos`-module from `sympy` in the snippet above.
Further, the numerical `numpy` and/or `mpmath` version of all these functions, and their generated derivatives, must be known to the solver module (with the same names as used by `sympy`). For this reason the solver module(s) always import most of the elementary functions[1], using the appropriate names (see the top of `Pendulum.py` in this example). If more advanced functions (f.i. Bessel functions) are required, they must be added by hand to the imports at the top of the solver module.

- The call of

  `kimoki.makeModules('Pendulum',...)`

  will generate a solver module, `Pendulum.py`, and `runPendulum.py` as a demonstration driver module.

[1]I.e., sqrt, log, exp, sin, cos, tan, sinh, cosh, tanh, asin, acos, atan, atan2, asinh, acosh, and atanh.

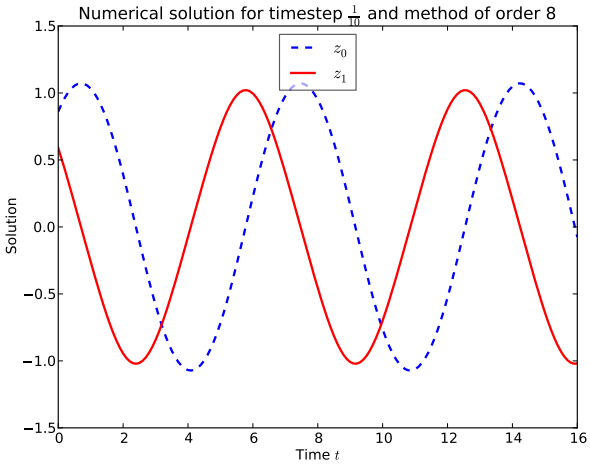

Fig. 1. Numerical solution of pendulum problem

  - `Pendulum.py`, the solver, contains all necessary code which is essential for our proposed higher order method. The user does not need to do anything with this file at the moment. But as an editable text file it is of course available for modification.
  - `runPendulum.py`, an example driver module, is made to test the solver module, and provide a starting point for real applications. It provides solutions of the Hamilton equations for schemes of various orders, using random initial conditions and parameters, and generate plots of the results. The drivers modules are added in HOMsPy for the user convenience. They are ready to be executed, but user can easily be modified. This is sometimes even necessary.

- One may now run the program `runPendulum.py`. Two separate tasks are executed by this program:
  First the subroutine `plotSingelSolution()` is called. This routine plots all components of a solution, generated with random initial values and parameters. The routine first checks if a solution has already been generated and saved to a `pickle` file, `#Pendulum_SolnTau100Ord8.pkl`. If the file does not exist, the subroutine `computeSingleSolution(order)` is called in order to generate a file with the solution. The solution is now read in and plotted, and the plot is saved to the file `Pendulum_Soln.png`; it will look similar to the plot in Fig. 1.
  `NB!` New execution of the `makeExample.py` can be result in overwriting the existing driver module.
  Second the subroutine `plotEnergyErrors()` is called. This routine plots how well energy is preserved by the solver, for different values of the timestep τ and order N of the solution. The energy error is expect to scale like τ^N with τ. The routine checks if `#Pendulum_EgyErrTau<T>Ord<N>.pkl` (with `<T>` = 500, 1000, and 2000, and `<N>` = 2, 4, 6, 8), has already been generated. If not, the subroutine `computeEnergyErrors()` is called in order to generate the required file(s), before the data is

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

read in and plotted. The plots are saved to the file `Pendulum_EgyErr.png`; it will look similar to the plot in Fig. 2.

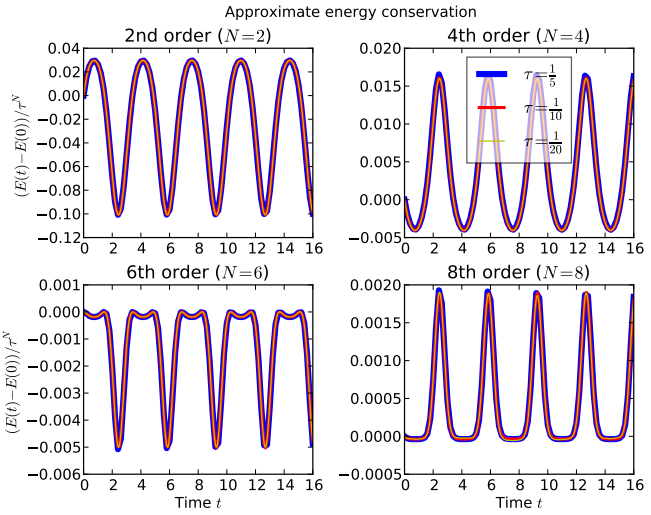

Fig. 2. Scaled energy errors for different values of the timestep τ and order N of the integrator. This figure verifies that the error scales like τ^N. The periodicity of the solution is reflected in the periodic variation of the energy error.

- Some comments:
  The code in the driver module is organized to first generate the solution and additional data, and save these to files, before the plots are made. For small demonstrations this is superfluous (by default the pickle files are removed immediately after they have been read), but such a separation is advantageous when fine-tuning plots (which may require many iterations) from data taking long time to generate.
  If the driver module crashes or is aborted, some pickle files (with names starting with the symbol `#`) may not have been removed.
  For high-quality figures one may want to generate the plots in `.pdf`-format, with axes labels and legends by use of LaTeX. This is possible in `matplotlib`, provided a working TeX-installation is detected. The driver module contains code for this (commented out to avoid unnecessary errors). In fact, most of the code in the driver module is related to *plotting* solution and data, very little to *generating* it.
- Fig. 3 illustrates use of the code generator.When the optional parameter MP=True a multiprecision version of the solver module and runfile is generated. The solver module consists of various functions and variables. Its most important function is `kiMoKi(z)`, which updates the solution z through one full timestep. The function `energy(z)` evaluates the Hamiltonian at the phase space point z. Many other functions are also defined.

V. Multi-precision (MP) version of HOMsPy

In this section, we will present the second important aspect of our program by an illustrative example which we have discussed using double-precision (DP) version in ref. [3]. Higher order methods and multi-precision container

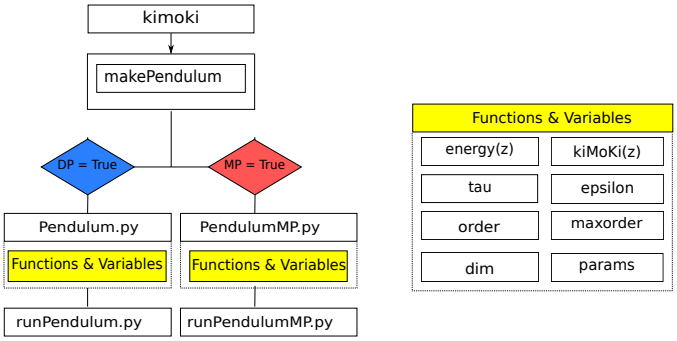

Fig. 3. Graphical representation of the procedure

solvers play very important rule to achieve high accuracy. Often some situations arise when minute information are required in many field of engineering, mathematics and physics. To tackle these situation multi-precision solvers can be important. In higher order accuracy for small values of time-steps rounding errors exceed the numerical truncation errors, multi-precision calculation is one of the way to fix this problem. In our paper [10], we discussed very-high-precision about the solutions of a class of Schrödinger type equations.

A. One parametric family of quartic anharmonic oscillator: procedural explanation

For the demonstration, we choose a problem of which the exact solution is known. In this example, we also demonstrate how parameter(s) (as α is a parameter given in equation (5)) can be included. Consider the non-linear anharmonic oscillator defined by the Hamiltonian

$$H = \frac{1}{2}p^2 + \frac{\alpha}{2}q^2 + \frac{1}{4}q^4. \tag{5}$$

Exact solution to this problem can be expressed in terms of the Jacobi elliptic functions [9],

$$q(t) = q_0\,\mathrm{cn}(\nu t|\mathrm{k}), \tag{6a}$$
$$p(t) = -q_0\nu\,\mathrm{sn}(\nu t|\mathrm{k})\,\mathrm{dn}(\nu t|\mathrm{k}). \tag{6b}$$

Here the initial conditions are $q(0) = q_0$, and $p(0) = p_0$, which implies that q_0 is either a maximum or a minimum of $q(t)$. The parameters and energy of the solution are given by

$$\nu = \left(\alpha + q_0^2\right)^{1/2},$$
$$k = 2^{-1/2}\,q_0/\nu,$$
$$E = \frac{\alpha}{2}\,q_0^2 + \frac{1}{4}q_0^4.$$

For detailed description of this problem see ref. [3]. A code snippet to generating multi-precision solver and driver modules is the following:

Solving anharmonic oscillators with multi-precision (MP)

```
def makeAnharmonicOscillator():
    # Choose names
    q, p, alpha = sympy.symbols(['q',
                  'p', 'alpha'])
    qvars = [q]; pvars = [p]
```

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
    params = [alpha]
    # Define potential
    V = alpha*q**2/2 + q**4/4
    # Code for MP computations
    kimoki.makeModules('Anharmonic', V,
   qvars,pvars, DP= True, MP= True,
   VERBOSE= True)
```

The code in line 10 shows that the `makeModules` function may take optional inputs: If the `MP` keyword is set to `True` then two additional files are generated: In this case the files `AnharmonicMP.py`, which is a solver module using multi-precision arithmetic, and `runAnharmonicMP.py`, which is a driver module. When the `VERBOSE` keyword is set to `True` some information from the code generating process will be written to screen, mainly information about the time used to process the various stages.

For further demonstration, we will consider the *global error* at a fixed endpoint by using different time-steps. Here we have used the same parameter as discussed in [3]. In our driver module, we wrote the following routine:

Evaluation of the exact position and velocity of the initial value problem

```
def ze_(t, alpha, q0):
# Positive energy solution;
# q(t) varies between q0 and -q0.
  if alpha + q0*q0/2 >= 0:
    nu = numpy.sqrt(alpha + q0*q0)
    k  = q0/(nu*numpy.sqrt(2))
    q  = q0*mpmath.ellipfun('cn',
         nu*t, k=k)
    p  = -nu*q0*mpmath.ellipfun('sn',
        nu*t,k=k)*mpmath.ellipfun('dn',
          nu*t,k=k)
    return q, p
ze = numpy.vectorize(ze_)
```

In similar way, other two `if-else` conditions can be defined for the negative energy solutions where $q(t)$ varies between q_0 and q_1 and also varies between q_0 and $q_1 > q_0$. Here `ze` is the exact solution written in vectorized form and `ellipfun()` is defined in `mpmath` for the calculations of elliptic functions.

In Fig. 4, the L_2 error is calculated by using DP and MP at fix time t. At $\tau = \frac{1}{10}$ and $\frac{1}{20}$, DP and MP are behaving identically, at $\tau = \frac{1}{40}$ and afterwards DP started to deviate and MP continues with the right behaviour (as expected). It is not due to failure of the 8th order method but it is due to the limited accuracy of DP calculations. By increasing the accuracy till 35 decimal places in MP, trend continues for very small time-steps $\tau = \frac{1}{40}, \ldots, \frac{1}{320}$.

We may conclude that if one wants to use the full strength of higher order methods then the limits of double precision calculations must be superseded and opts for multi-precision calculations.

VI. Concluding Remarks

In this paper we have demonstrated that the proposed extensions of the standard Störmer-Verlet symplectic integration scheme can be implemented numerically, and that

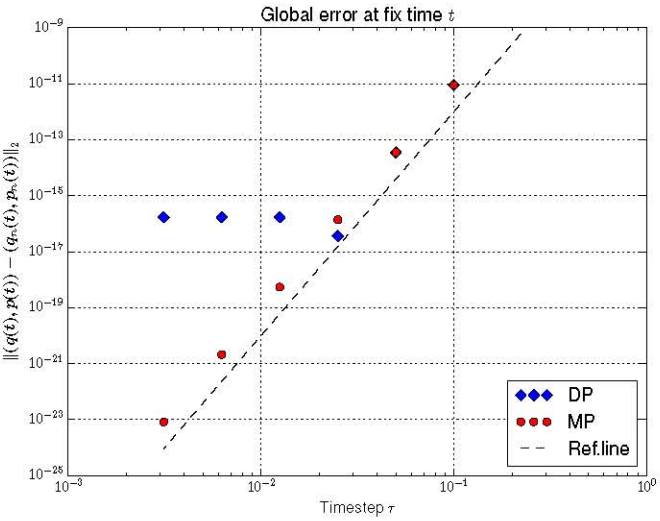

Fig. 4. Convergence rates of double-precision (DP) and multi-precision (MP) solver by using 8th order method at fix time $t = 10$. We used $\tau = \frac{1}{10}, \frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}, \frac{1}{320}$, $q_0 = 0.54$ and $\alpha = 0.13$.

the implemented code behave as expected with respect to accuracy.

The core module in HOMsPy is the `kimoki` module, which uses symbolic algebra to generate a solver module for each specific Hamiltonian, together with a driver module example. Different examples are presented with the use of double- and multi-precision solvers.

Acknowledgment

A. Mushtaq would like to thank prof. Trond Kvamsdal for his valuable support and also would like acknowledge department of mathematical sciences, NTNU for the conference support.

References

[1] A. Mushtaq, A. Kværnø, K. Olaussen, *Systematic Improvement of Splitting Methods for the Hamilton Equations*, Proceedings for the World Congress on Engineering, London July 4–6, **Vol I**, 247-251, (2012).

[2] 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.

[3] A. Mushtaq, K. Olaussen, *Automatic code generator for higher order integrators*, Computer Physics Communication **185** (2014), 1461-1472.

[4] *SymPy Developement Team*, `http://sympy.org/`

[5] *NumPy Developers*, `http://numpy.org/`

[6] F. Johansson *et. al.*, *Python library for arbitrary–precision floating-point arithmetic*, `http://code.google.code/p/mpmath/` (2010)

[7] J. D. Hunter, *Matplotlib: A 2D graphics environment*, Computing in Science & Engineering **9**, 90–95 (2007)

[8] C. Führer, J. E. Solem, O. Verdier, *Computation with Python*, Pearson 2013.

[9] M. Abramowitz and I.S. Segun, *Handbook of Mathematical Functions*, Ch. 16, Dover Publications (1968)

[10] A. Mushtaq, A. Noreen, K. Olaussen, I. Øverbø, *Very-high-precision solutions of a class of Schrödinger type equations*, Computer Physics Communications **182**, 1810–1813 (2011)