A Sample Code Manual for Computing Normal Forms for Poincaré Maps

nofo4maps project
Version 1.0
April 22, 2025

J. Gimeno, À. Jorba†, M. Jorba-Cuscó, M. Zou

Table of Contents

1 Introduction

The *nofo4maps* project has been formally documented in the manuscript GJJZ24 (`http://www.maia.ub.es/dsg/2024/2405gimeno.pdf`). This manual provides a basic code sample for computing normal forms on a Poincaré map, with the main goal of demonstrating the methodology for computing a normal form in the case of an elliptic fixed point.

As an illustrative model , we use the classical **Hénon-Heiles** system, a well-known Hamiltonian model in celestial mechanics. This system is one of the examples discussed in the manuscript. While the manuscript presents advanced methods, this manual serves as a complementary resource, offering a more accessible and practical guide to the methodology. For this particular example, we also compute the associated twist map around the elliptic fixed point.

The computation of normal forms does not require this assumption. However, the example is Hamiltonian and after computing the normal form, we proceed to compute the associated twist map around the elliptic fixed point.

Through this manual, we aim to highlight the key steps, provide insights into necessary adaptations for other systems, and enhance the reproducibility of the work. We are also happy to provide further assistance, discuss details, and even share the undocumented version of the manuscript upon request. Our goal is to offer a clearer understanding of the methodology itself through this simplified presentation.

1.1 The Discrete Hénon-Heiles System

The Hénon-Heiles (HH) system is defined by the Hamiltonian:

$$H(x, y, p_x, p_y) = \frac{1}{2}(p_x^2 + p_y^2) + \frac{1}{2}(x^2 + y^2) + x^2 y - \frac{1}{3}y^3$$

To study its dynamics in discrete time, we consider the Poincaré map $P\colon \{x = 0\} \to \{x = 0\}$, which describes the return of trajectories to the section $\{x = 0\}$.

By fixing this section and selecting a specific energy level, the four-dimensional flow defined by (x, y, p_x, p_y) is effectively reduced to a two-dimensional map in variables $z = (y, p_y)$. The discrete system is then written as:

$$\bar{z} = P(z)$$

Here, the map P incorporates both the fixed energy constraint and a flying time that depends on the initial condition z.

This manual documents code that explores various aspects of P, through the following steps:

1. Compute a fixed point z_0 of P; see [1-fixed-point], page 5.
2. Compute high-order derivatives of P at z_0; see [2-jet-fixed-point], page 6.
3. Compute the local normal form near z_0; see [3-normal-form], page 7.
4. Construct a twist map if z_0 is elliptic; see [4-twist], page 8.

2 Software Requirements

The code is written in C and has been successfully compiled using the GNU C Compiler (GCC) version 12.2.0. It is self-contained, requires no external libraries, and relies on providing appropriate initial guesses for correct execution.

2.1 Optional Software

Output files generated with the Taylor v2.2 (`https://github.com/joang/taylor2-dist`) package are already included in the `0-model` folder. These files define data structures such as `MY_JET`, which encapsulate multivariate polynomial manipulation. The folder also contains commented examples on how to regenerate these files using the `taylor` tool, which may be useful for extending the current examples or adapting them to other contexts.

Additionally, any standard linear algebra library (e.g., LAPACK) can be optionally used. In this sample project, we have taken advantage of the fact that the system reduces to a planar discrete map—even though the original Hénon-Heiles system resides in 4D phase space. Therefore, linear algebra routines such as solving linear systems and computing eigenvalues have been implemented specifically for 2-by-2 matrices.

For more general use cases, the following capabilities are typically required:

- A linear system solver
- An eigenvalue decomposition routine

These are standard operations supported by most linear algebra libraries. Note that depending on the context of your specific problem, it may be necessary to perform these computations using complex arithmetic.

3 Usage

To use the code, follow the folder structure in the specified order, adhering to the instructions in the accompanying `README` files or in the subsequent sections of this manual:

- `0-model`: Contains the model and outputs from `taylor` used in subsequent folders. Everything required for the sample is included here.
- `1-fixed-point`: Computes a fixed point of the Poincaré map for the Hénon-Heiles system, fixing an energy level and a spatial section at $x = 0$.
- `2-jet-fixed-point`: Computes the high-order derivatives of the Poincaré map at the fixed point computed in `1`.
- `3-normal-form`: Performs a change of coordinates on the jet obtained in `2` to derive its normal form.
- `4-twist`: Computes a twist map based on the output of `3`, leveraging the elliptic fixed point from `1` and the Hamiltonian nature of the system.

3.1 Warnings

This code is a sample designed to illustrate the process of computing normal forms for Poincaré maps. Specifically, it addresses 2-by-2 systems, solving linear systems and computing eigenpairs.

Note that this sample does not include components such as parameter-dependent normal forms or torus visualizations, as these are outside the scope of the demonstration and might detract from clarity. You can find the details in the associated manuscript of the *nofo4maps* project.

3.2 Enhancements

You may consider the following improvements:

- Implement parameter-dependent normal forms.
- Develop the torus visualization for a radius within the validity range.
- Improve the linear algebra parts by incorporating custom libraries.
- Run the code with extended precision (this requires modifying the output files from `taylor` in the `0-model` folder).
- Use wrappers to run the code, particularly the part of `taylor`, in a different programming language, e.g., Python.

0-model folder

It contains the raw equations of the Hénon-Heiles system, those with extension `.eq`.

- `hh.eq` contains the ODE system and an expression to isolate the variable `py`, given an energy level (as an external variable) and assuming `x = 0`.
- `hh_vars.eq` is based on `hh.eq`, but includes an additional line to define jets with 2 symbols of degree 1.
- `hh_s2.eq` extends `hh_vars.eq` to allow jets of arbitrary degree.
- `hh_s1.eq` is similar to `hh_s2.eq`, but limited to a single symbol.

Each of these files contains commented lines indicating the corresponding command to run using Taylor v2.2 (`https://github.com/joang/taylor2-dist`). For example, the following commands:

```
taylor -o hh.h -name ode -expression -header hh.eq
taylor -o hh.c -name ode -expression -jet -step -headername ode.h hh.eq
```

were used to generate the output files `hh.c` and `hh.h` included in the [0-model], page 4, directory. These commands invoke the `taylor` library, which processes the model described in `hh.eq` written in plain text form. For a detailed explanation of the command-line arguments, refer to the Taylor library manual.

The generated output files includes two main data structures: `MY_FLOAT` and `MY_JET` declared in the `.h` files and implemented in `.c` files. `MY_JET` represents a multivariate polynomial whose coefficients are of type `MY_FLOAT`. In parts of this project, `MY_FLOAT` is configured to use complex numbers. Files with the suffix `_cmplx` indicate the use of complex arithmetic.

The folder will then contain a pool of models with small variations in the `MY_FLOAT` and `MY_JET` data structures. From other parts of the project, we establish symbolic links to the appropriate model depending on the specific purpose of each component. See the following sections of this manual for more details.

> **Observation:** Some models involving jets with varying numbers of symbols and degrees can be unified under a generic polynomial model with two symbols and arbitrary degree. In this approach, the specific computation or context determines which coefficients are actually needed at each step.
>
> Understanding the required number of symbols and degree for each computation not only serves as valuable training, but also allows one to take advantage of the different polynomial arithmetic implementations provided by `taylor`, helping to optimize both computational and memory efficiency.

1-fixed-point folder

This project computes the fixed point of a Poincaré map derived from the Hénon–Heiles system, specifically for the case where `x = 0` and at a fixed energy level `h_0`.

File Description

- `ode.eq`, `ode.c`, and `ode.h`: Symbolic links to the model located in the model's folder.
- `la2d.h`, `la2d.c`: Linear algebra routines for 2-by-2 matrices.
- `newton.c` and `newton.h`: Code for computing a fixed point of the Poincaré map, given a sufficiently accurate initial guess.
- `main.c`: Contains the main code that uses the provided initial guesses.

Since this folder focuses on computing fixed points of a Poincaré map, it is necessary to also obtain derivatives of that map. For this reason, the model must compute the first-order variational flow of the Hénon–Heiles system.

The symbolic links for `ode` were created with the following shell commands:

```
ln -sf ../0-model/hh_vars.eq ode.eq
ln -sf ../0-model/hh_vars.c ode.c
ln -sf ../0-model/hh_vars.h ode.h
```

Note that since the option `-headername ode.h` was used when generating `ode.c`, it is appropriate to link to `ode.h` rather than `hh_vars.h`.

Compiling and Execution

To compile and run the program, follow these steps:

1. Use the provided `Makefile` to compile the code by executing:

   ```
   make
   ```

2. Run the executable with an output filename, e.g. `pfix`:

   ```
   ./main.exe pfix
   ```

2-jet-fixed-point folder

This project computes high-order derivatives of the Poincaré map at the fixed point of the Hénon–Heiles system, for a fixed energy level `h_0` and spatial section `x = 0`.

File Description

- `ode.eq`, `ode.c`, and `ode.h`: Symbolic links to the model located in the model's folder. These files set the maximum degree for the Taylor expansion.
- `poinca.c` and `poinca.h`: Code for constructing the high-order derivatives of the Poincaré map. These are model-dependent.
- `main.c`: Contains the main code. It uses the fixed point (`pfix`) obtained from folder [1-fixed-point], page 5.

The symbolic links for `ode` and the input file were created using the following shell commands:

```
ln -sf ../0-model/hh_s2.eq ode.eq
ln -sf ../0-model/hh_s2.c ode.c
ln -sf ../0-model/hh_s2.h ode.h
ln -sf ../1-fixed-point/pfix
```

The jet expansion around the fixed point of the Poincaré map is performed using 2 symbols (corresponding to the dimension of the discrete system defined by the map) and a fixed maximum degree, which is set to 15 in `hh_s2.eq`.

Compiling and Execution

To compile and run the program, follow these steps:

1. Use the provided `Makefile` to compile the code by executing:

   ```
   make
   ```

2. Run the executable with an output filename, e.g. `jet_pfix`:

   ```
   ./main.exe pfix jet_pfix
   ```

3-normal-form folder

This project computes the normal form of the local approximation obtained in [2-jet-fixed-point], page 6.

File Descriptions

- `ode.eq`, `ode.c`, and `ode.h`: Symbolic links to the model located in the model's folder. These files define the maximum degree for the expansion, with arithmetic using complex numbers.
- `la2d.h`, `la2d.c`: Implementations for linear algebra operations on 2-by-2 matrices.
- `nofo.c` and `nofo.h`: Provide functions for computing the normal form.
- `util_jet.h`, `util_jet.c`: Introduce a method to estimate the validity range of a jet.
- `main.c`: Contains the main program logic. It uses the `jet_pfix` from [2-jet-fixed-point], page 6.

The symbolic links on this folder have created using the following shell commands:

```
ln -sf ../0-model/hh_s2.eq ode.eq
ln -sf ../0-model/hh_s2_cmplx.c ode.c
ln -sf ../0-model/hh_s2_cmplx.h ode.h
ln -sf ../1-fixed-point/la2d.c
ln -sf ../1-fixed-point/la2d.h
ln -sf ../2-jet-fixed-point/jet_pfix
```

Compiling and Execution

To compile and run the program, follow these steps:

1. Use the provided `Makefile` to compile the code by executing:

   ```
   make
   ```

2. Run the executable with an output filename, e.g. `nofo`:

   ```
   ./main.exe jet_pfix nofo
   ```

   Warnings:

   Some parts are model-dependent, such as the function `is_unavoidable_resonant` found in the `nofo` files. The function `is_resonant_monomial` only detects potential unavoidable resonances for a given log10 tolerance, and may require adaptation for different scenarios.

4-twist folder

This project computes a twist map from the normal form obtained in [3-normal-form], page 7.

File Descriptions

- `ode.eq`, `ode.c`, and `ode.h`: Symbolic links to the model located in the model's folder. These files define the maximum degree for the expansion, using complex arithmetic. It is sufficient to consider half the number of symbols and (floor of) half the degree used in [3-normal-form], page 7.
- `twist.c` and `twist.h`: Provide functions for computing the twist map from the normal form.
- `util_jet.h`, `util_jet.c`: Introduce a method to estimate the validity range of a jet.
- `main.c`: Contains the main program logic. It uses the `nofo` function from [3-normal-form], page 7.

The symbolic links on this folder have created using the following shell commands:

```
ln -sf ../0-model/hh_s1.eq ode.eq
ln -sf ../0-model/hh_s1_cmplx.c ode.c
ln -sf ../0-model/hh_s1_cmplx.h ode.h
ln -sf ../3-normal-form/util_jet.c
ln -sf ../3-normal-form/util_jet.h
ln -sf ../3-nofo/nofo
```

Compiling and Execution

To compile and run the program, follow these steps:

1. Use the provided `Makefile` to compile the code by executing:

   ```
   make
   ```

2. Run the executable with an output filename, e.g. `twist`:

   ```
   ./main.exe nofo twist
   ```

   Warnings:

   The twist computation may be sensitive to index selection, as only part of the normal form is used. This affects how the twist is obtained. The selection is user-dependent and should be carefully studied for each specific problem and research interest.

Acknowledgments

We extend the acknowledgments in this manual to those included in the associated manuscript. Additionally, we are grateful to the anonymous referees whose feedback encouraged us to produce a more complete and accessible documentation of the original code sample.