Hamiltonian Mechanics Programming Lab (Sistemi Hamiltoniani 1 Laboratorio)

Niels Benedikter niels.benedikter@unimi.it,
Chiara Boccato chiara.boccato@unimi.it,
Università degli Studi di Milano

January 16, 2024

Contents

1 Introduction

1.1 Introduction to Linux

I recommend Linux Mint (`https://linuxmint.com/`), which is easy to use and to install. Read the instructions provided on the Linux mint page or just do an internet search for, e. g., "linux mint installation". You have 3 different options to choose from:

- Direct installation on your computer: download Linux Mint, create a bootable usb pen drive, boot from the USB drive, click "install", follow the instructions. You can keep your Windows/Mac operating system in parallel, so that the computer will ask which system to start when being switched on. *Attention: do not choose to overwrite your current system unless you really want to and have a backup of all your data!*
- If you are worried about interfering with your current system: install `https://www.virtualbox.org/` in your Windows or MacOS. This simulates a computer, in which you can then install Linux Mint.
- You can also install Linux on an external usb drive. You will need a bootable linux usb key (as in the installation instructions above) and an additional empty USB drive (minimum 4 gigabyte, better larger). You find the procedure explained here: `https://www.fosslinux.com/41285/install-complete-linux-mint-on-a-usb-drive.htm`

Get used to your new operating system before the first lab. At least figure out how to create a folder, how to create and edit a text file, and how to open a terminal. If you need additional software, you can install it for free from the "Software Manager" you find in the main menu.

Practical task: Set up a linux system and get used to it.

1.2 Programming in C

We use pure C of standard C99. No C++. Here is a very basic example reading input from the terminal and writing to the terminal. With `#include <stdio.h>` in the first line of the program we tell the compiler to include the standard library functions for input and output, in particular to and from the terminal.

```
#include <stdio.h>

float add(float x, float y)          // define a function
{
    float output;
    output = x + y;
    return output;
}
```

```c
int main()
{
    float n1, n2, n3;                // define variables

    printf("Enter a number: ");      // print message to terminal
    scanf("%f", &n1);                // read an integer, store in n1
    printf("Enter a number: ");
    scanf("%f", &n2);

    n3 = add(n1,n2);
    printf("The sum is %f\n", n3);
    return 0;                        // close the program
}
```

Note that `printf` uses directly the value of the variable (here `n3`) whereas `scanf` needs the address of the variable where to store the input. The address of a variable is obtained writing, e. g., `&n2`. The address of a variable is called a *pointer* in C because it points to the memory location of the variable. The codes for printing and reading from/to different kinds of variables, such as `%f` for floating point numbers (real numbers) depends on the kind of variable. Just search on the internet for scanf or printf to find tables. Note that printf and scanf may use different codes for the same type of variable.

Another example using a for loop to repeat some computation many times. The command `for(A;B;C)` contains three arbitary commands; command `A` is called when the loop is first entered; command `B` is checked after every iteration to decide if we can already terminate the loop; command `C` is applied at every iteration and usually used to increase some sort of counter.

```c
#include <stdio.h>

int main()
{
    float rate;
    float infected;
    infected = 1.0;

    printf("Enter the infection rate: ");
    scanf("%f", &rate);      // read rate of infections

    for (int i =1; i <= 30; i = i +1)
    {
        infected = infected * rate;
        printf("On the day %d: %f infected people.\n", i, infected);
    }
    return 0;
}
```

That's it for the moment. Save the code in a textfile with ending ".c" and in the next section we'll take a look at how to compile and execute your program.

Practical task: Refresh your knowledge of the C programming language. Search on the internet to identify your favorite reference webpages on programming in C.

1.2.1 The gcc compiler & makefiles

Open a terminal. Type

```
sudo apt install gcc
```

and hit return. You will be asked to enter your password, then the files will be automatically downloaded and installed (when asked for permission, hit "y"). Repeat with

```
sudo apt install libc6-dev
sudo apt install make
sudo apt install xterm
```

The first command installs libraries needed for programming (whatever those are, it doesn't matter to us). The second command installs a program called "make" which reduces the amount of typing of compiler commands. The last command installs a particular terminal program required by the scientific plotting library *Mizar* that we are going to use.

The gcc compiler A compiler is a program that turns your source code (a text file with ending .c) into a program that can be executed by the computer. We use the standard compiler gcc. A standard call to the compiler could be the following line on the terminal, executed in the folder where our sourcecode pandemic.c is saved:

```
gcc -std=c99 -Wall -Wextra -o euler euler.c -lm
```

The first word is the name of the program we execute (the compiler `gcc`). Then `-std=c99` indicates that we use the C99 standard of the programming language C (a fairly modern version of C). The options `-Wall` and `-Wextra` make sure we see all possible warning about possibly bad code and helps us avoid strange behavior of the resulting program. Next, `-o euler` tells the compiler to output the compiled program with the name "euler". Then we give the name of the source code `euler.c`. Finally `-lm` tells the compiler to include ("link") the functionality of the mathematics library of C.

The compiler will now either produced errors and warnings, or stay silent (if your code is correct) and create the file "euler". Your program is now ready to execute typing on the terminal the command

```
./euler
```

and hitting return.

Practical task: Make sure you can compile and execute the two simple programs given above.

makefiles To reduce the amount of compiler commands that one has to type (they can get very long and complicated) there is a tool called "make". Simply use a text editor to create a file called "makefile". You may copy and paste the following content.

```
# MIZAR
INCLUDE = -I ${MIZAR}
MIZCOM = ${MIZAR}/libmizarcom.a ${MIZAR}/libtek.a
MIZCOMGIF = ${MIZAR}/libmizarcom.a ${MIZAR}/libtek.a ${MIZAR}/libmizargif.a
MIZCOMPNG = ${MIZAR}/libmizarcom.a ${MIZAR}/libtek.a ${MIZAR}/libmizarpng.a
MIZCOMJPG = ${MIZAR}/libmizarcom.a ${MIZAR}/libtek.a ${MIZAR}/libmizarjpg.a
MIZINTTEK = ${MIZAR}/libmizarint.a ${MIZAR}/libtek.a
MIZFIL = ${MIZAR}/libmizarfil.a
MIZFILGIF = ${MIZAR}/libmizarfil.a ${MIZAR}/libmizargif.a

single_ho: single_ho.c
  gcc -Wall ${INCLUDE} -std=c99 -g -o single_ho single_ho.c ${MIZINTTEK} -lm -lgd

multi_ho: multi_ho.c
  gcc -Wall ${INCLUDE} -std=c99 -g -o multi_ho multi_ho.c ${MIZINTTEK} -lm -lgd

all: single_ho multi_ho

clean:
  rm -f single_ho
  rm -f multi_ho
```

Important: every make rule (such as `single_ho`) needs its command line (the line following it, with the gcc command) start with a *true tabulator* and not with a number of spaces! In your texteditor, you may need to go to the options or settings first and disable "tab as spaces".

Some further explanation on the content (marked in yellow – simply replace or add an extra make rule):

- MIZINTTEK will be interactive mode, after having compiled your program by typing `make single_ho` or `make all`, you may run the program now with

  ```
  xterm -t -e ./single_ho
  ```

  You do not need the `-lgd` option for the compiler in this case.

- MIZCOMPOS creates ps files, which is good for use for example in LaTeX documents and for print documents because it is a vector format. It can afterward also be converted to a pdf with `ps2pdf` on the commandline (very convenient for inclusion in LaTeX files).

- MIZCOMPNG and likewise MIZCOMGIF, MIZCOMJPG. These require linking the library with the `-lgd` at the end of the compiler instruction, otherwise you get a lot of error messages in trying to compile. Furthermore, if there is an error mentioning `deferr`, check that in the top lines,

you included the part `${MIZAR}/libtek.a`. (COM stands for combined mode, including interactivity, thich therefore requires the Tektronix terminal, so has to include also libtek.)

- Alternatively, you can also create gifs and other pictures using MIZFIL, which will create a file `grafic.dat`. Then on the commandline you run the following commands to convert the dat–file into a gif–file:

```
$MIZAR/mizargif grafic.dat
FIN
Y
```

Animations For animations, it is convenient to use the functions `pagin();` and `quadro();` of mizar.

Gif–files can be joined to an animation using for example the program gifsicle (see `Animazioni.txt` from the mizar documentation).

1.3 The Mizar library for scientific plotting

1.3.1 Option 1: Installing Mizar by Compiling its Source Code

1.3.2 Option 2: Installing Mizar using Binaries

If you do not want to bother with compiling Mizar or keep getting error messages (probably due to problems with unresolved dependencies), you can alternatively try to install the binaries we provide. Proceed via the following steps.

Install the following libraries Use `sudo apt install` on the terminal with all of the following:

```
libgd3
libgd-dev
libgd-tools
libpng16-16
libpng-dev
libpng-tools
libgif7
libgif-dev
libjpeg62
libjpeg62-dev
libjpeg8
libjpeg8-dev
libjpeg9
libjpeg9-dev
libjpeg-dev
libjpeg-progs
libjpeg-tools
```

Instead of doing this one by one, you can also type on the terminal `sudo apt install ...` writing for `...` the whole list at once.

Get the binaries Download the file `mizar_with_binaries.zip`. Unpack it anywhere you want. You find a folder called `mizar`.

Move Mizar to the right location This folder has to be moved to `/usr/local`, using the following instructions:

(i) Right click onto empty space in your file manager and do "Open as root". For all of the following procedure you should seean orange/red warning message at the top now.

(ii) Right click on the mizar folder and do "Copy".

(iii) Use repeatedly the up-arrow in the toolbar at the top of the file manager until you can identify a folder called `usr`.

(iv) Go into that folder, then go into the subfolder called `local`.

(v) Right click, select "paste" to place the `mizar` folder here.

(vi) Right click on the `mizar` folder, click on "properties", in the new window go to "permissions". From the 3 dropdown menus, make sure they say (in this order): Read & Write, Read & Write, Read Only.

You can close the file manager now.

Set environment variables Set the "variabili d'ambiente" with the same procedure as in `istruzioni.txt`, using the choice `/usr/local/`. This amounts to executing the following two lines on the terminal:

```
export MIZAR_ROOT="/usr/local/mizar"
export MIZAR="/usr/local/mizar/lib"
```

To have these settings stored permanently (otherwise you have to retype this commands every time after starting the computer), go to your home directory, press on your keyboard "Ctrl+h" to show hidden files, open the file `.bashrc` with a text editor, insert these two lines, save.

Test your installation Try to compile the examples that you can find in the file `mizar_base.tgz` using `make all` and execute them using the terminal command `xterm -t -e ./cursore`. Remember the latter command, this is the standard for running Mizar programmes.

Practical task: Install Mizar, either compiling the source code as in the official instructions or using the binaries as just explained. Look at the source code of the test examples and the make files to get an idea of how Mizar is practically used. Experiment with modifications to the Mizar commands in the source code.

1.3.3 Alternatives to Mizar

2 Numerical solutions of Hamiltonian equations

2.1 The Forward Euler Algorithm

For $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ we consider

$$\dot{x} = f(x, t) \tag{2.1}$$

The approximation

$$\frac{x_{n+1} - x_n}{h} \simeq \dot{x} \simeq f(x_n, t_n) \tag{2.2}$$

where h represents a small time step, leads to

$$x_{n+1} = x_n + hf(x_n, t_n) \tag{2.3}$$
$$t_{n+1} = t_n + h \tag{2.4}$$

To approximate the solution we start with a certain initial data x_0, t_0 and iterate.

Euler is a first order algorithm. The local error (error per step) is proportional to h^2, while the global error (error at a fixed time) is proportional to h.

Example: the harmonic oscillator. Recall

$$H(p, q) = \frac{p^2}{2} + \frac{\omega^2}{2} q^2 \tag{2.5}$$

with $\omega > 0$. We have the system

$$\begin{aligned} \dot{q} &= p \\ \dot{p} &= -\omega^2 q \end{aligned} \tag{2.6}$$

The Euler step is

$$q_{n+1} = q_n + hp_n \tag{2.7}$$
$$p_{n+1} = p_n + h(-\omega^2 q_n) \tag{2.8}$$

Practical task: Compute numerically the solution of (2.6). For the beginning, choose $\omega = 2$, $h = 1/100$, the number of iterations $N = 1000$, $q_0 = 0.2$ and $p_0 = 0.3$. Then increase N and compare with the exact solution.

(i) Draw the Euler trajectories in the plane (p, q).

(ii) Draw the energy $H(p(t), q(t))$ along the Euler trajectories.

What do you find? Is the behavior of the energy and trajectories realistic?

2.2 The Runge-Kutta Algorithm

A better approximation can be obtained by taking into account of higher order derivatives. An example is the Runge-Kutta algorithm (see for example appendix A of Prof. A. Giorgilli's notes). If we stop at second order we obtain

$$\begin{aligned} k_1 &= hf(x_n, t_n), \\ x_{n+1} &= x_n + hf\big(x_n + \frac{k_1}{2}, t_n + \frac{h}{2}\big) \\ t_{n+1} &= t_n + h \end{aligned} \tag{2.9}$$

The local error here is order h^3.

Practical task: Draw the trajectories in the plane (p, q) obtained with the Runge-Kutta algorithm at second order (2.9) for the simple pendulum

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta)$$

(Remember here that $\theta \in S^1$, so θ needs to be taken $\mod 2\pi$)

Practical task: Do the same exercise for the **double** pendulum: for $\theta_1, \theta_2 \in S^1$,

$$\begin{cases} (m_1+m_2)\ell_1\ddot{\theta}_1 + m_2\ell_2\cos(\theta_1-\theta_2)\ddot{\theta}_2 \\ \qquad\qquad = -m_2\ell_2\sin(\theta_1-\theta_2)\dot{\theta}_2^2 - (m_1+m_2)g\sin(\theta_1) \\ \ell_2\ddot{\theta}_2 + \ell_1\cos(\theta_1-\theta_2)\ddot{\theta}_1 = \ell_1\sin(\theta_1-\theta_2)\dot{\theta}_1^2 - g\sin(\theta_2) \end{cases}$$

2.2.1 Appendix: create pictures with Mizar

(i) Compile with

```
MIZCOMTEK = ${MIZAR}/libmizarcom.a ${MIZAR}/libmizartek.a
```

i.e., for example

```
euler: euler.c
gcc -Wall ${INCLUDE} -std=c99 -g -o euler euler.c ${MIZCOMTEK} -lm
```

and run the program, as usual, with

```
xterm -t -e ./euler
```

Alternatively, you can compile it with

```
MIZFIL = ${MIZAR}/libmizarfil.a
```

and run it with

```
./euler
```

(in this case you don't open the tektronik terminal). In both cases you will produce a file called grafic.dat

(ii) Give the command

```
$MIZAR/mizarpng grafic.dat
```

press `FIN` and `Y`. In this way we create the file "grafico.png" (or a series of files if your program produces more images).

2.3 The Leapfrog Algorithm

We know that the Hamiltonian flow ϕ^t, defined by $\phi^t(q_0, p_0) = (q_t, p_t)$ is a canonical transformation, i.e., it has a symplectic Jacobian. The Leapfrog algorithm is a numerical integrator whose time step is a canonical transformation.

Consider a Hamiltonian of the form

$$H(q,p) = \frac{p^2}{2} + U(q);$$

then the Hamiltonian equations are

$$\begin{aligned} \dot{q} &= p \\ \dot{p} &= -\frac{\partial U}{\partial q}(q) =: F(q) \end{aligned} \tag{2.10}$$

with initial data q_0, p_0. The **Leapfrog time step** is

$$q_{n+1} = q_n + hp_{n+1/2} \tag{2.11}$$

$$p_{n+3/2} = p_{n+1/2} + hF(q_{n+1}) \tag{2.12}$$

where the first point $p_{1/2}$ has been obtained with a single half Euler step, once:

$$p_{1/2} := p_0 + \frac{h}{2}F(q_0). \tag{2.13}$$

This is the **Leapfrog initialization**.

How does Leapfrog compare to Euler? Leapfrog is a second order algorithm, while Euler is first order. However, the computational effort does not change. Here are some other advantages of Leapfrog:

(i) *Time reversibility.*

(ii) *Symplecticity.*

(iii) *Preservation of a (slightly modified) Hamiltonian.*

(iv) *Preservation of closed loops in the phase portrait (at least up to the machine precision).*

Practical task: Implement the Leapfrog algorithm for the simple pendulum

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta)$$

with θ taken $\mod 2\pi$. Draw the trajectories in the phase space.

Question: how can we draw the trajectories, since we only compute $p_{n+3/2}$ and q_{n+1}, but not p_{n+1}?
Answer: we do a backwards half Euler step, wich we obtain by the half Euler step $p_{n+3/2} = p_{n+1} + \frac{h}{2}F(q_{n+1})$ inverting for p_{n+1}

$$p_{n+1} = p_{n+3/2} - \frac{h}{2}F(q_{n+1}) \tag{2.14}$$

Attention: this formula should be used only for drawing the point and not for the algorithm!

Practical task: Compute the phase space plot of the harmonic oscillator with Leapfrog. Evolve to a fixed time T, subtract the analytical solution, and repeat this procedure for different time step resolutions h (i. e., analyze how the global error scales with h). Use a logarithmic plot to show that Euler is first order and Leapfrog second order (for logarithmic plots in mizar you can insert the line "coord(4);" in your code: it changes the axis into logarithmic axes). Verify that the energy is (up to an error that does **not** grow with time) conserved by Leapfrog.

Practical task: Implement the Leapfrog algorithm for the coupled harmonic oscillators, with Hamiltonian

$$H(q_1, q_2, p_1, p_2) = H_{\omega_1}(q_1, p_1) + H_{\omega_2}(q_2, p_2) + \frac{\omega_3^2}{2}(q_1 - q_2)^2 \tag{2.15}$$

where

$$H_\omega(q_1, p_1) = \frac{1}{2}(p^2 + \omega^2 q^2) \tag{2.16}$$

Compute a phase portrait in the plane (q_1, p_1) by Leapfrog. Plot the "left half" of the energy, $H_{\text{left}}(q_1, p_1) = \frac{1}{2}\left(p_1^2 + \omega^2 q_1^2\right)$ as a function of time. Choose the parameters as follows:

- $\omega_3 = 1/4$, $\omega_1 = \omega_2 = 1$
- $\omega_3 = 1/4$, $\omega_1 = 2\omega_2$, $\omega_2 = 1$

What do you observe?

2.4 Poincaré Sections for Linearly Coupled Harmonic Oscillators

Analytical solution of the linearly coupled harmonic oscillators As background knowledge and to compare with and check the numerical results we

recall the analytical solution, using the power of canonical transformations in the Hamiltonian formalism. So consider again the Hamilton function

$$H(q,p) = \frac{\omega_0}{2}(p_0^2+q_0^2) + \frac{\omega_1}{2}(p_1^2+q_1^2) + (q_0-q_1)^2 \; .$$

We start with the scaling transformations

$$\begin{pmatrix}\tilde{p}_0\\ \tilde{p}_1\end{pmatrix} = \begin{pmatrix}(\omega_0/\omega_1)^{1/4} & 0\\ 0 & (\omega_1/\omega_0)^{1/4}\end{pmatrix}\begin{pmatrix}p_0\\ p_1\end{pmatrix}$$

and

$$\begin{pmatrix}\tilde{q}_0\\ \tilde{q}_1\end{pmatrix} = \begin{pmatrix}(\omega_1/\omega_0)^{1/4} & 0\\ 0 & (\omega_0/\omega_1)^{1/4}\end{pmatrix}\begin{pmatrix}q_0\\ q_1\end{pmatrix} \; .$$

We obtain a Hamiltonian that is symmetric under exchange of the two oscillators:

$$H(\tilde{q},\tilde{p}) = \frac{\sqrt{\omega_0\omega_1}}{2}\left(\tilde{p}_0^2+\tilde{p}_1^2\right) + A\tilde{q}_0^2 + B\tilde{q}_1^2 + C\tilde{q}_0\tilde{q}_1$$

where

$$A = \frac{\omega_0^{3/2}}{2\omega_1^{1/2}} + \frac{\omega_0^{1/2}}{\omega_1^{1/2}} \; , \quad B = \frac{\omega_1^{3/2}}{2\omega_0^{1/2}} + \frac{\omega_1^{1/2}}{\omega_0^{1/2}} \; , \quad C = -2 \; .$$

Now note that the expression $\tilde{p}_0^2+\tilde{p}_1^2$, having the form of an ℓ^2–norm (squared), is invariant under all transformations $U \in SO(2)$ acting by

$$\begin{pmatrix}\tilde{\tilde{p}}_0\\ \tilde{\tilde{p}}_1\end{pmatrix} := U\begin{pmatrix}\tilde{p}_0\\ \tilde{p}_1\end{pmatrix} \; .$$

In order to obtain a canonical transformation we also transform at the same time

$$\begin{pmatrix}\tilde{\tilde{q}}_0\\ \tilde{\tilde{q}}_1\end{pmatrix} := U\begin{pmatrix}\tilde{q}_0\\ \tilde{q}_1\end{pmatrix} \; .$$

In matrix notation we have

$$A\tilde{q}_0^2 + B\tilde{q}_1^2 + C\tilde{q}_0\tilde{q}_1 = \begin{pmatrix}\tilde{q}_0 & \tilde{q}_1\end{pmatrix}\begin{pmatrix}A & C/2\\ C/2 & B\end{pmatrix}\begin{pmatrix}\tilde{q}_0\\ \tilde{q}_1\end{pmatrix} \; .$$

The matrix being symmetric, it can be diagonalized by a $U \in SO(2)$:

$$U^T\begin{pmatrix}A & C/2\\ C/2 & B\end{pmatrix}U = \begin{pmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{pmatrix} \; .$$

That way we just get two decoupled harmonic oscillators:

$$H(\tilde{\tilde{q}},\tilde{\tilde{p}}) = \frac{\omega_0\omega_1}{2}\left(\tilde{\tilde{p}}_0^2+\tilde{\tilde{p}}_1^2\right) + \lambda_1\tilde{\tilde{q}}_0^2 + \lambda_2\tilde{\tilde{q}}_1^2 \; .$$

The eigenvalues λ_1 and λ_2 are most easily obtained solving the system

$$\begin{aligned}\lambda_1\lambda_2 &= \det\begin{pmatrix}A & C/2\\ C/2 & B\end{pmatrix} = AB - \frac{C^2}{4}\\ \lambda_1+\lambda_2 &= \mathrm{tr}\begin{pmatrix}A & C/2\\ C/2 & B\end{pmatrix} = A+B \; .\end{aligned}$$

One finds

$$\lambda_{1,2} = \frac{A+B}{2} \pm \frac{1}{2}\sqrt{(A-B)^2 - C^2}\,.$$

Inverting the two transformations (the scaling and the linear transformation with U) we can compute the oscillator motions as a superposition of a low frequency and a high frenquency oscillation. (Attention, λ_1 and λ_2 are not directly the frequencies because there is an additional prefactor of the kinetic term.)

Poincaré sections In Poincaré sections we may observe the following behavior. Poincaré sections can be used to detect periodic orbits.

- If a trajectory produces a finite number of points in the Poincaré section, the motion is periodic (after a finite number of "rounds" the particle has returned to its initial phase-space coordinates).
- If the motion is quasiperiodic with incommensurable frequencies we see a curve, made up from a dense set of points in which the particle returns to the surface of the Poincaré section. This is caused by the frequencies of doing one round not being in a rational relation.
- Finally we may see completely filled areas: this is the signature of chaotic behavior. It does not happen for the linearly coupled oscillators, but we will later observe it in the Hénon–Heiles model.

Poincaré sections are particularly suited to "detecting" periodic orbits, for example in the Lotka–Volterra model [Tuc02]. For a further discussion of the qualitative behavior of Poincaré sections, I recommend [Gio20, Capitolo 8].

Practical task: Study the Poincaré section ($q_0 = 0$) of the linearly coupled harmonic oscillators. Implement a program that asks the user the value of the energy. Study different initial data (in Mizar you can use the function utcur). In particular, study the cases

- $\omega_0 = \omega_1 = 1$
- $\omega_1 = 1$, $\omega_1 = \frac{\sqrt{5}-1}{2}$

2.5 The Hénon–Heiles model

While at Princeton in 1962, Michel Hénon and Carl Heiles worked on the nonlinear motion of a star around a galactic center with the motion restricted to a plane. In 1964 they published an article titled "The applicability of the third integral of motion: Some numerical experiments". Their original idea was to find a third integral of motion in a galactic dynamics. For that purpose they took a simplified two-dimensional nonlinear axi-symmetric potential and found that the third integral existed only for a limited number of initial conditions. In the modern perspective the initial conditions that do not have the third integral of motion are called chaotic orbits (and we aim to see them in our numerical experiments as the main qualitative difference to the linearly coupled harmonic oscillators). [Wik22]

The Hénon–Heiles model consists of two harmonic oscillators with a non–linear coupling, i. e., the Hamilton function is not quadratic in position and momenta. In fact

$$H(q,p) = \frac{\omega_0}{2}\left(p_0^2 + q_0^2\right) + \frac{\omega_1}{2}\left(p_1^2 + q_1^2\right) + q_0^2 q_1 - \frac{1}{3}q_1^3 \ . \tag{2.17}$$

We observe that, if $\omega_0, \omega_1 > 0$, then if the initial energy $E_0 := H(q_0, p_0, q_1, p_1)$ is strictly less than the critical energy (or escaping energy)

$$E_{\text{crit}} := \min\left\{\frac{\omega_0^3}{24} + \frac{\omega_0^2\omega_1}{8}, \frac{\omega_1^3}{6}\right\} \ ,$$

then the Poincaré section remains inside the bounded domain in the (p_1, q_1)–plane defined by

$$\frac{\omega_1}{2}\left(p_1^2 + q_1^2\right) - \frac{1}{3}q_1^3 \leq E_0 \ . \tag{2.18}$$

For the following it will be convenient to always plot the contour of this domain to help with orientation in the numerical plots. It is also convenient to choose the initial energy as a fraction of the critical energy.

Practical task: Show (by pen and paper computation, without computer) that Eq. (2.18) indeed constraints the Poincaré sections to a bounded domain in the plane. Why so?

Poincaré sections for the Hénon–Heiles model We repeat the exercise done in the last section for the linearly coupled harmonic oscillators:

(i) fix an initial energy as a fraction of the critical energy, e. g., $E_0 = \frac{E_{\text{crit}}}{100}$, $E_0 = \frac{E_{\text{crit}}}{10}$, $E_0 = \frac{E_{\text{crit}}}{1000}$, $E_0 = 0.024E_{\text{crit}}$, $E_0 = 0.039E_{\text{crit}}$ (play with different and similar values);

(ii) fix a surface in phase space (call it Σ) as the plane defined by $q_0 = 0$;

(iii) choose the initial value of q_0 on the plane, i. e., $q_0 = 0$;

(iv) define q_1 and p_1 from the two coordinates of a mouse click (in Mizar, use the function `utcur`);

(v) determine p_0 by solving the equation $E_0 = H(0, p_0, q_1, p_1)$;

(vi) set the size of `utarea` so that the plot contains all the domain permitted by the energy condition Eq. (2.18);

(vii) follow a leapfrog trajectory to a very large time and draw a point at coordinates (q_1, p_1) every time the surface Σ is crossed in forward direction (say, whenever in the leapfrog step a change of sign from $u_0 \leq 0$ to $u_0 > 0$ occurs).

Practical task: Plot the Poincaré sections of the Hénon–Heiles model both for the resonant and the non–resonant case, experimenting with different fractions of the critical energy for E_0. Observe the emergence of chaotic behavior. What qualitative difference do you notice in the emergence of chaotic trajectories between the resonant and non–resonant case?

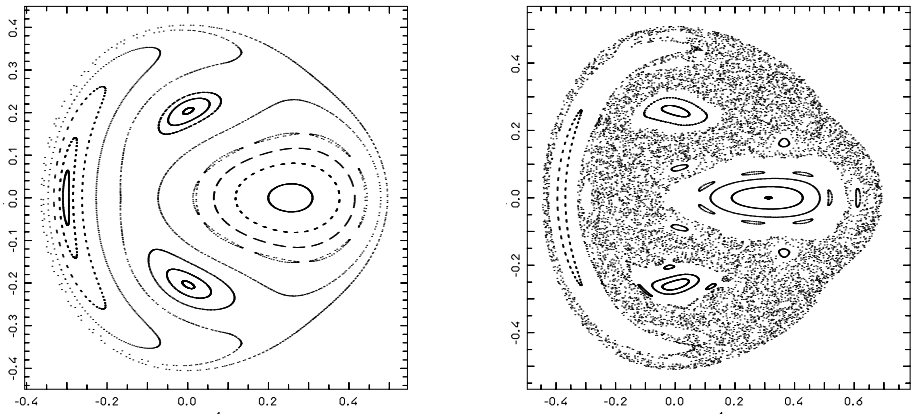

Figure 1: **Left:** Poincaré sections for the resonant (i. e., $\omega_0 = \omega_1 = 1$) Hénon–Heiles model with $E_0 = 0.5E_{\text{crit}}$ (where the critical energy is found to be $E_{\text{crit}} = 0.1667$). **Right:** Same with $E_0 = 0.8E_{\text{crit}}$; chaotic trajectories appear. (Try to zoom in to see details.)

In Fig. 1 we display the result for the resonant Hénon–Heiles model, in Fig. 2 for the non–resonant case. For a more extended discussion on the Hénon–Heiles model, please read section 6.2 of Prof. A. Giorgilli's notes:
`http://www.mat.unimi.it/users/antonio/metmod/Note_6.pdf`.

An improved algorithm for Poincaré sections [Hen82, Tuc02] Observe that our algorithm for finding the intersection of the trajectory with the surface Σ is not very precise: in fact we may have gone up to a distance h (length of a time step) beyond the surface, where we then detect the change of sign in q_0. So far we have simply taken the values (q_1, p_1) found there and plotted them on the $q_0 = 0$–plane. One way of improving this is by linearly interpolating between the (q_1, p_1) of the previous step and the current step to obtain a guess for the position on the $q_0 = 0$–plane. A further improvement is obtained by the algorithm we shall discuss now.

Consider a general system of first–order ordinary differential equations

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, x_2, \ldots, x_N) \\ &\vdots \\ \dot{x}_{N-1} &= f_{N-1}(x_1, x_2, \ldots, x_N) \\ \dot{x}_N &= f_N(x_1, x_2, \ldots, x_N) \, . \end{aligned}$$

Moreover we consider a surface Σ defined by an equation $S(x_1, x_2, \ldots x_N) = 0$; for simplicity $S(x_1, x_2, \ldots x_N) = x_N - a$ for some $a \in \mathbb{R}$ (which in our application may be specialized even further to $a = 0$).

We transform the differential equations such that x_N becomes the new independent variable, replacing the time variable t. To do so we compute

$$\frac{\mathrm{d}t}{\mathrm{d}x_N} = (\dot{x}_N)^{-1} = \frac{1}{f_N(x_1, \ldots)} \, .$$

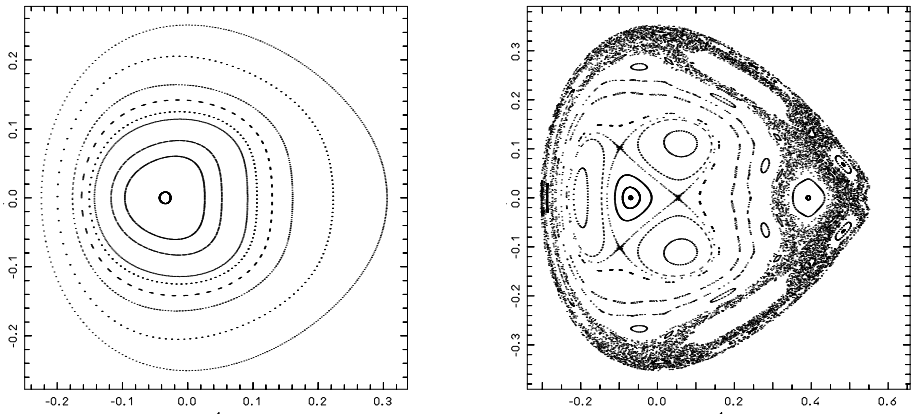

Figure 2: **Left:** Poincaré sections for the non–resonant (i. e., $\omega_0 = 1$ and $\omega_1 = \frac{\sqrt{5}-1}{2}$) Hénon–Heiles model with $E_0 = 0.5E_{\text{crit}}$. **Right:** Same with $E_0 = 0.99E_{\text{crit}}$ (notice the higher fraction of the critical energy).

If you implement this procedure, remember to check for $f_N(x_1, x_2, \ldots) = 0$; if this should by coincidence be encountered resort to linear interpolation or the naive algorithm (change of sign in q_0 as above) to find the intersection point with Σ.

Let us assume that usually $f_N(x_1, \ldots) \neq 0$. Then we can proceed by the chain rule to get the new system of equations

$$\begin{aligned} \frac{\mathrm{d}x_1}{\mathrm{d}x_N} &= \frac{\mathrm{d}x_1}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}x_N} = \frac{f_1(x_1, x_2, \ldots)}{f_N(x_1, x_2, \ldots)} \\ &\vdots \\ \frac{\mathrm{d}x_{N-1}}{\mathrm{d}x_N} &= \frac{\mathrm{d}x_{N-1}}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}x_N} = \frac{f_{N-1}(x_1, x_2, \ldots)}{f_N(x_1, x_2, \ldots)} \,. \end{aligned} \tag{2.19}$$

Our new algorithm works as follows:

(i) As before, integrate the leapfrog trajectory until we detected a change of sign in $S(x_1, x_2, \ldots)$.

(ii) Save the last found value of $S(x_1, \ldots)$, calling it, e. g., S_{f}.

(iii) Make a copy of the last obtained values of the phase space variables.

(iv) To the copy of the phase space variables, apply a single step with the evolution Eq. (2.19) for a distance $\Delta x_N = -S_{\text{f}}$. This lands us exactly on the surface Σ. (Since this step is only executed once we do not worry about accumulation of errors, so this step can simply be a forward Euler step.)

(v) Plot the obtained values for (q_1, p_1), then revert to the leapfrog trajectory and continue the evolution.

Practical task: Implement the improved algorithm to plot again the Poincaré sections of the Hénon–Heiles model.

2.6 Symplectic Methods by the Splitting Construction

In the following we discuss the splitting construction, a general method to construct symplectic algorithms. As a particular case we will re–derive the Leapfrog algorithm. The best reference for this section is [LR01].

We write $x = (p, q) \in \mathbb{R}^{2n}$, and the Hamiltonian equations as

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \{x, H\} = L_H x \; .$$

Here $\{,\}$ denotes the Poisson bracket and the linear operator L_H is defined by its action on functions f on phase space, $L_H f := \{H, f\}$.

By abstract functional calculus, the solution of the Hamiltonian equations, i. e., the Hamiltonian flow can be written as

$$x(t) = e^{tL_H} x_0 = \sum_{n=0}^{\infty} \frac{t^n}{n!} L_H^n x_0 \; .$$

The representation by the exponential series is not very useful in applications. In the following we will therefore consider systems with Hamilton function

$$H(p, q) = A(p, q) + B(p, q)$$

with the assumption that A and B separately generate a Hamiltonian flow that can be computed analytically.

Consider now a time step $h \in \mathbb{R}$. To construct a splitting method, we try to find coefficients $c_1, d_1, \ldots, c_n, d_n \in \mathbb{R}$ such that

$$e^{hL_H} = e^{c_1 h L_A} e^{d_1 h L_B} \cdots c^{c_n h L_A} e^{d_n h L_B} + \mathcal{O}(h^k) \; ,$$

where the order k of the error term should be as high as possible.

As a composition of symplectic flows, the numerical time step

$$S(h) := e^{c_1 h L_A} e^{d_1 h L_B} \cdots c^{c_n h L_A} e^{d_n h L_B}$$

is automatically a symplectic flow (and this is exact, not just as an approximation).

> Dear friend Wollstein!
> If you receive these lines, we (three) have solved the problem in a different manner — in the manner of which you have constantly tried to dissuade us. The feeling of security that you have predicted for us once we would overcome the difficulties of the move, is still eluding us; on the contrary, Endenich may not even be the end!
> What has happened in recent months against the Jews evokes justified fear that they will not let us live to see a more bearable situation.
> (..)
> I am sorry that we cause you yet more effort beyond death, and I am convinced that you are doing what you can do (which perhaps is not very much). Forgive us our desertion! We wish you and all our friends to experience better times.
> Your truly devoted
> Felix Hausdorff
>
> *Farewell letter of Felix Hausdorff 1942*

Example: The Euler–Cromer (semi–implicit Euler) integrator (A further reference for this example is [DR05].) We consider a Hamilton function

$$H(p,q) = T(p) + V(q)$$

and split as

$$A := T\,, \qquad B := V\,.$$

(We may think of $T(p) = p^2/(2m)$ and $V(q)$ a potential, with the obvious generalization to more than one particle.) We try to write

$$e^{hL_H} = e^{c_1 hL_T} e^{d_1 hL_V} + \mathcal{O}(h^k)\,.$$

To compute the coefficients c_1 and d_1, we use the *Baker–Campbell–Hausdorff (BCH) formula*[1], generalizing the exponential law to non–commuting linear operators (you may also think of matrices) X and Y:

$$e^X e^Y = e^{X+Y+\frac{1}{2}[X,Y]+\frac{1}{12}[X,[X,Y]]-\frac{1}{12}[Y,[X,Y]]+\ldots}\,. \tag{2.20}$$

The exponent on the right hand side is a formal series which in general may not be convergent. Its explicit form is quite complicated, but in applications frequently only the first few terms (as given here) are needed.

Let's apply the BCH formula to our concrete case:

$$e^{c_1 hL_T} e^{d_1 hL_V} = e^{c_1 hL_T + d_1 hL_V + \frac{1}{2}c_1 d_1 h^2 [L_T, L_V] + \mathcal{O}(h^3)}\,.$$

We choose $c_1 = d_1 = 1$, so that

$$c_1 hL_T + d_1 hL_V = h(L_T + L_V) = hL_{T+V} = hL_H\,.$$

Moreover, acting on any phase space function f, by use of the Jacobi identity we have

$$\begin{aligned}[L_T, L_V]f &= L_T L_V f - L_V L_T f = \{T, \{V, f\}\} - \{V, \{T, f\}\} \\ &= \{\{T, V\}, f\} = L_{\{T,V\}} f\,.\end{aligned}$$

We conclude that

$$e^{c_1 hL_T} e^{d_1 hL_V} = e^{hL_H + \frac{h^2}{2} L_{\{T,V\}} + \mathcal{O}(h^3)}\,.$$

We are working on the re–derivation of an Euler algorithm, i. e., an integrator of first order (i. e., which to first order in h agrees with the exact flow). This is achieved having hL_H in the exponent, so we want to treat $\frac{h^2}{2} L_{\{T,V\}}$ as an error term, outside the exponent as in Eq. (2.20). To obtain this form, we may use the *dual BCH/Zassenhaus formula*:

$$e^{X+Y} = e^x e^Y e^{-\frac{1}{2}[X,Y]} \ldots$$

[1] Felix Hausdorff was professor at the University of Bonn. He is considered to be one of the founders of modern topology and who contributed significantly to set theory, measure theory, and functional analysis. Being Jewish, in 1942 he committed suicide to evade deportation.

Applied to separate the $\mathcal{O}(h^2)$–terms from the exponent, we get

$$\begin{aligned} e^{hL_H + \frac{h^2}{2} L_{\{T,V\}} + \mathcal{O}(h^3)} &= e^{hL_H} e^{\frac{h^2}{2} L_{\{T,V\}} + \mathcal{O}(h^3)} e^{-\frac{1}{2}[hL_H, \frac{h^2}{2} L_{\{T,V\}}]} \\ &= e^{hL_H} (1 + \frac{h^2}{2} L_{\{T,V\}} + \mathcal{O}(h^3))(1 + \mathcal{O}(h^3)) \\ &= e^{hL_H} + \mathcal{O}(h^2) \; . \end{aligned}$$

So we have shown that

$$e^{hL_H} = e^{hL_T} e^{hL_V} + \mathcal{O}(h^2) \; ,$$

as expected the obtained algorithm agrees with the exact flow to first order and has an additive error term of order h^2.

Let us complete the discussion by showing the explicit implementation of the algorithm. In fact, the flow generated by the Hamiltonian function V may be easily computed: just derive the Hamiltonian equations and observe that they can be explictly solved, corresponding to a "kick" to the momentum with the force derived from V. In fact, one finds the analytic solution (with initial data (q_n, p_n) moving by a time intervall h to the next "step")

$$e^{hL_V} \begin{pmatrix} q_n \\ p_n \end{pmatrix} = \begin{pmatrix} q_n \\ p_n - V'(q_n)h \end{pmatrix} =: \begin{pmatrix} q_n \\ p_{n+1} \end{pmatrix} \; .$$

Instead the Hamiltonian function $T(p) = p^2/(2m)$ generates the force-free movement of the particle, i. e.,

$$e^{hL_T} \begin{pmatrix} q_n \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} q_n + \frac{p_{n+1}}{m} h \\ p_{n+1} \end{pmatrix} =: \begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} \; .$$

The combination of these two steps is precisely the Euler–Cromer algorithm.

From the construction we easily understand that every splitting algorithm comes with a conserved energy that is a small deviation of the original Hamiltonian. In fact, we saw that

$$\begin{aligned} e^{hL_T} e^{hL_V} &= e^{hL_H + \frac{h^2}{2} L_{\{T,V\}} + \mathcal{O}(h^3)} \\ &= e^{hL_{H + \frac{h}{2}\{T,V\} + \mathcal{O}(h^3)}} \; . \end{aligned}$$

That is,

$$H' := H + \frac{h}{2}\{T, V\} + \mathcal{O}(h^3)$$

is the conserved Hamiltonian. (Attention: the expansion to higher orders may again only be a formal, non–convergent, series.) From this example we also see that a k–th order integrator has a conserved Hamiltonian H' differing up to k–th order from H (as in the Euler–Cromer case the correction is of order h.)

To be even more specific, the construction of the conserved Hamiltonian of the Euler–Cromer integrator can be written out for the harmonic oscillator $H(p, q) = p^2 + q^2$, splitting as $T(p) = p^2$ and $V(q) = q^2$:

$$\{T, V\} = \{p^2, q^2\} = \frac{\partial p^2}{\partial q} \frac{\partial q^2}{\partial p} - \frac{\partial q^2}{\partial q} \frac{\partial p^2}{\partial p} = -4pq \; .$$

Therefore

$$H' = p^2 + q^2 + \frac{h}{2}(-4pq) + \mathcal{O}(h^2)$$

is conserved under the application of the Euler–Cromer iteration.

Example: The Leapfrog (Verlet) integrator (A further reference for this example is [DR05].) The Verlet integrator is obtained using the splitting

$$e^{\frac{h}{2}L_V} e^{hL_T} e^{\frac{h}{2}L_V} \ . \tag{2.21}$$

Explictly, the successive execution of these three flows corresponds to the steps (where F is the force corresponding to the potential V)

$$\begin{aligned} p_{n+\frac{1}{2}} &= p_n + \frac{h}{2}F(q_n) \\ q_{n+1} &= q_n + hp_{n+\frac{1}{2}} \\ p_{n+1} &= p_{n+\frac{1}{2}} + \frac{h}{2}F(q_{n+1}) \ . \end{aligned}$$

If the step p_{n+1} is eliminated to directly go from $p_{n+\frac{1}{2}}$ to $p_{n+\frac{3}{2}}$, this is exactly the Leapfrog algorithm.

Let us show again (as already done in Section 2.3), that this is a second order integrator. We use the BCH formula twice to combine the three flows into one exponent:

$$\begin{aligned} & e^{\frac{h}{2}L_V} e^{hL_T} e^{\frac{h}{2}L_V} \\ &= \exp\left(\frac{h}{2}L_V + hL_T + \frac{1}{2}\left[\frac{h}{2}L_V, hL_T\right] + \mathcal{O}(h^3)\right) e^{\frac{h}{2}L_V} \\ &= \exp\left(\frac{h}{2}L_V + hL_T + \frac{h}{2}L_V + \frac{1}{2}\left[\frac{h}{2}L_V, hL_T\right] + \frac{1}{2}\left[\frac{h}{2}L_V + hL_T + \frac{1}{2}\left[\frac{h}{2}L_V, hL_T\right], \frac{h}{2}L_V\right] + \mathcal{O}(h^3)\right) \\ &= \exp\left(hL_H + \frac{h^2}{4}[L_V, L_T] + \frac{h^2}{4}[L_T, L_V] + \mathcal{O}(h^3)\right) \ , \end{aligned}$$

where we used $[L_V, L_V] = 0$. Since $[L_T, L_V] = -[L_V, L_T]$ the second order terms in the exponent cancel, and we find

$$e^{\frac{h}{2}L_V} e^{hL_T} e^{\frac{h}{2}L_V} = \exp\left(hL_H + \mathcal{O}(h^3)\right) \ .$$

So indeed Leapfrog is correct up to second order, i. e., there is no error term of order h^2. (The conversion of the $\mathcal{O}(h^3)$ error term in the exponent into an additive error outside the exponent is always easily obtained applying the first order Zassenhaus formula, in fact independent of the concrete operators X one finds $\exp(X + \mathcal{O}(h^k)) = \exp(X) + \mathcal{O}(h^k)$.)

By retaining all terms of order h^3 in the BCH unification of the three flows into one exponent, the reader may derive the order–h^2 correction to H yielding a conserved Hamiltonian H' for the Leapfrog algorithm.

2.6.1 The SABA3 Algorithm

Let $H = A+B$. As an additional requirement (already satisfied by the Leapfrog algorithm), in the following we consider only the construction of symmetric integrators (meaning that time reversal corresponds to an inverse step, $S(-h) = S(h)^{-1}$). This means that we use an ansatz of the form

$$S(h) = e^{d_1 h L_B} e^{c_2 h L_A} e^{d_2 h L_B} \cdots e^{d_n h L_B} e^{c_{n+1} h L_A} e^{d_n h L_B} \cdots e^{d_2 h L_B} e^{c_2 h L_A} e^{d_1 h L_B} \ .$$

To motivate the SABA3 algorithm, let us first attempt a naive push of the splitting construction to obtain an order–h^4 integrator. We take $n = 2$, use the BCH formula to write the product of flows as a single exponent, and then demand that $c_2, c_3, d_1, d_2 \in \mathbb{R}$ are chosen such that the order–h terms sum up to H while all terms of order h^2, h^3, and h^4 cancel. This leads to the following equations connecting the coefficients:

$$\begin{aligned} c_3 + 2c_2 &= 1 \\ d_1 + d_2 &= \frac{1}{2} \\ \frac{1}{12} - \frac{1}{2}c_2 + \frac{1}{2}c_2^2 + c_2 d_1 - c_2^2 d_1 &= 0 \\ -\frac{1}{24} + \frac{1}{4}c_2 - c_2 d_1 + c_2 d_1^2 &= 0 \ . \end{aligned}$$

Solutions to this system of equations can be found by numerical search. It turns out that the system has a unique real solution, namely

$$d_1 \simeq 0.675 \ , \quad c_2 \simeq 1.3512 \ , \quad d_2 \simeq -0.1756 \ , \quad c_3 \simeq -1.7024 \ . \tag{2.22}$$

So this is in principle a feasible approach. Unfortunately, this turns out to not work very well in applications: some time steps get negative, others compensate this by getting rather large (just look at $c_3 \simeq -1.7024$). With some steps getting large, in actual numerical computations often the Leapfrog method (second order) yields better results than this fourth–order method.

So is there any way we can avoid the emergence of negative steps? Since Eq. (2.22) is the unique solution, we have to modify the whole approach to permit more freedom in the choice of parameters. This is exactly the purpose of the *SABA3 integrator*, which we will discuss now.

The key idea is to think of a Hamiltonian of the form

$$H = A + \varepsilon B \ ,$$

where ε is to be thought of as a small parameter. Again by the BCH formula we obtain

$$S(h) = e^{d_1 h L_{\varepsilon B}} e^{c_2 h L_A} e^{d_2 h L_{\varepsilon B}} e^{c_3 h L_A} e^{d_2 h L_{\varepsilon B}} e^{c_2 h L_A} e^{d_1 h L_{\varepsilon B}} = e^{h L_K} \ ,$$

where

$$\begin{aligned} K = {} & k_{11} A + \varepsilon k_{12} B + h\varepsilon k_{21} \{A, B\} + h^2 \varepsilon k_{31} \{A, \{A, B\}\} \\ & + h^2 \varepsilon^2 k_{32} \{\{A, B\}, B\} + \mathcal{O}(h^3\varepsilon + h^3\varepsilon^2 + h^3\varepsilon^3 + h^4) \ . \end{aligned}$$

The coefficients k_{ij} are functions of the c– and d–parameters. If we demand cancellation up to second order in h^2, this yields the conditions

$$k_{21} = 0 \,, \qquad k_{31} = 0 \,, \qquad k_{32} = 0 \,.$$

To get a larger freedom in the choice of parameters, we decide to treat ε^2–terms as already so small that we may ignore them. So we demand only to cancel ε–terms up to the desired order in h, i. e., to second order in h we demand only

$$k_{21} = 0 \,, \qquad k_{31} = 0 \,, \tag{2.23}$$

and k_{32} remains free. It turns out that now one can indeed find a solution by positive parameters c_i and d_j. This way one obtains the *SABA3 integrator*:

inverted role of A, B

$$\begin{aligned} S(h) &= e^{c_1 h L_A} e^{d_1 h L_{\varepsilon B}} e^{c_2 h L_A} e^{d_2 h L_{\varepsilon B}} e^{c_2 h L_A} e^{d_1 h L_{\varepsilon B}} e^{c_1 h L_A} \\ c_1 &= \frac{5 - \sqrt{15}}{10} \,, \quad c_2 = \frac{\sqrt{15}}{10} \,, \quad d_1 = \frac{5}{18} \,, \quad d_2 = \frac{4}{9} \,. \end{aligned} \tag{2.24}$$

The error term is $\mathcal{O}(h^6\varepsilon + h^2\varepsilon^2)$. This algorithm works quite well for practical purposes.

Practical task: Consider the simple pendulum

$$H(p,q) = p^2 + \varepsilon(1 - \cos\theta) \,, \quad \theta \in [-\pi, \pi) \,.$$

Split as $H = T + V$, where $T = p^2$, and implement the SABA3 integrator to plot the phase space evolution for $\varepsilon = 1$ and $\varepsilon = \frac{1}{10}$. By playing with the parameters, can you provoke a visible difference in the plots compared to the Leapfrog method?

Practical task: Implement SABA3 for the harmonic oscillator $H = p^2 + q^2$, $T = p^2$, with fixed $\varepsilon = 1$. Evolve up to a fixed time t, subtract the analytical solution and make a logarithmic plot of the difference as a function of h. The logarithmic plot is well suited to read off the exponent of the error term. Show that the error of the SABA3 method, at fixed epsilon, is of second order in the step size h.
(This is the same task we did in Section 2.3 to verify that Leapfrog is a second order integrator.)

2.6.2 Correctors to Splitting Algorithms

We now discuss how to improve the order of the error for a splitting algorithm such as SABA4. The recommended reference for this is [LR01, Section 10].

Hypothesis: The Hamiltonian is of the form $H = A + \varepsilon B$, where A *is quadratic in the momenta* and independent of the positions (e. g., $A = T(p) = \frac{p^2}{2m}$) and B *depends only on the positions* (e. g., $B = V(q)$).

If this hypothesis is satisfied, we can compute the double Poisson bracket $\{\{A, B\}, B\}$ explicitly and find that it is a function that depends only on q.

Thus $\{\{A,B\},B\}$ may also be considered as a Hamiltonian that generates an explicitly known flow, namely a force just as the potential $V(q)$.

So let's recall that SABA3, written in a single exponent by using the BCH formula, is the flow

$$S = e^{hK} \qquad K = k_{11}A + \varepsilon k_{12}B + \cancel{h\varepsilon k_{21}\{A,B\}} + \cancel{h^2\varepsilon k_{31}\{A,\{A,B\}\}} + h^2\varepsilon^2 k_{32}\{\{A,B\},B\} + \mathcal{O}(h^3\varepsilon + h^3\varepsilon^2 + \ldots)$$

The cancelled terms on the first line are, by the choice of c– and d–coefficients as demanded in Eq. (2.23), exactly zero in the SABA3 method. The biggest non–vanishing error term is the term on the second line, i. e., $h^2\varepsilon^2 k_{32}\{\{A,B\},B\}$.

The corrector is an additional iteration step using the flow generated by $\{\{A,B\},B\}$ to cancel this term. In fact, we define the corrected time step as

$$S_{\text{corr}} := e^{-h^3\varepsilon^2\frac{c}{2}L_{\{\{A,B\},B\}}} S e^{-h^3\varepsilon^2\frac{c}{2}L_{\{\{A,B\},B\}}} \quad \text{with } c := k_{32}\,. \tag{2.25}$$

For SABA3 the explicit choice is $c = \frac{54-13\sqrt{15}}{648}$. In practical applications we often have $\varepsilon = 1$ instead of a really small perturbation parameter.

Practical task: Implement SABA3 with corrector for the harmonic oscillator. Make a logarithmic plot of the global error as a function of step size h. Compare the result to Leapfrog and SABA3+corrector: the error is now found to be of fourth order.

2.6.3 Alternative Splitting

Another possibility to improve the numerical method is to use a different splitting. In fact there is no deeper reason why to choose $H(p,q) = T(p) + V(q)$ except that both T and V generate a flow that has an explicit formula that we can implement. In general we can also split $H(p,q) = A(p,q) + B(p,q)$ as long as we have analytical expressions for the flows generated by L_A and L_B. The typical choice is of A as a quadratic Hamiltonian (i. e., a system of harmonic oscillators) and B a remaining interaction.

A natural application is the Hénon–Heiles model: we can write

$$\begin{aligned} H(p,q) &= A(p,q) + B(q) \\ A(p,q) &= \frac{\omega_0}{2}\left(q_0^2 + p_0^2\right) + \frac{\omega_1}{2}\left(q_1^2 + p_1^2\right) \\ B(q) &= q_0^2 q_1 - \frac{1}{3}q_1^3\,. \end{aligned}$$

The flow e^{hL_A} is just the analytical solution of two decoupled harmonic oscillators. The flow e^{hL_B} is just a force kick, where the partial force is to be computed from the potential $B(q_0, q_1)$.

Practical task: Implement the Leapfrog–like second order splitting

$$S = e^{\frac{h}{2}L_A} e^{hL_B} e^{\frac{h}{2}L_A}$$

with the new choice of A and B. Use this splitting to draw the Poincaré sections of the Hénon–Heiles model again. (In order to not mix up too many things in one program you may use simple linear interpolation for the Poincaré sections, not the advanced algorithm we discussed at the end of Section 2.5.)

3 Computer algebra with polynomials

Our next goal is to construct a first integral for the Hénon–Heiles model using the method of Poincaré. To this end, we need to implement operations with complex polynomials. We will first work with polynomials of a single complex variable, then generalize to polynomials of four variables (representing two momenta and two positions in the Hénon–Heiles model). We need functions to do (algebraically) the following operations with polynomials:

(i) output as a text string

(ii) addition of polynomials

(iii) multiplication of a polynomial with a complex number

(iv) multiplication of two polynomials

(v) derivatives of polynomials

(vi) evaluation of a polynomial in a given point *(use for example Horner's method)*

(vii) Poisson brackets, obtained as a combination of derivative, multiplication, and a difference (multiplication by -1 and sum)

Notice that Poisson brackets of polynomials have the property that

$$\{\text{cubic}, \text{quadratic}\} = \text{cubic} \ , \quad \{\text{cubic}, \text{cubic}\} = \text{quartic}$$

and so on, so the method of Poincaré we will discuss next week will lead to a growing degree of the polynomials. In our program we will fix (once and for all at the beginning, for example as `#define MAXDEG (20)`) some degree (say, 20 for each variable) that is sufficiently high for the number of Poincaré iterations we intend.

Practical task: Implement the operations (i)-(vi) above for polynomials of a single complex variable.

3.1 Examples of useful libraries and functions

- `#include <string.h>`
  - Here we have for example the function

    ```
    char *strcat(char *destination, const char *source);
    ```

    that concatenates the destination string and the source string, and the result is stored in the destination string.
- `#include <stdio.h>`
  - The function

    ```
    int sprintf(char *str, const char *format, arg1, arg2, ... );
    ```

where `str` is a character array on which data is written, `format` is the output string, `arg1,arg2,...` are the values to be converted (similar to `printf()`).

- `#include <complex.h>`

  This is the library managing operations with complex numbers.

  - Define a complex number as `double complex z = 1.0 + 2.0*I`
  - The functions `double creal(double complex z)` and `double cimag(double complex z)` return its real and imaginary part.
  - To define an array of complex numbers:

    ```
    complex double *poly;
    ```

- `#include <stdlib.h>`

  - The function

    ```
    void *calloc(size_t nelem, size_t elsize);
    ```

    allocates space for an array of `nelem` elements each with size in bytes `elsize`. The space will be initialized to all bits 0.
    Example:

```
complex double *poly=(complex double *)calloc((MAXDEG+1),sizeof(complex double));
```

3.2 Complex Numbers in C

string operations.

3.3 Arrays in C

treating as pointer!

3.4 ... or by recursion (advanced programming challenge, optional)

explain and show a little bit of code

4 The Method of Poincaré for the Construction of First Integrals

Consider a Hamilton function, with positions $x = (x_1, \ldots, x_n)$ and momenta $y = (y_1, \ldots, y_n)$, consisting of a quadratic part and an interaction (with a coupling constant $\varepsilon \in \mathbb{R}$ that we think of as small but will eventually take it as $\varepsilon = 1$):

$$H(x, y) = H_0(x, y) + \varepsilon H_1(x, y)$$

where

$$H_0(x,y) = \frac{1}{2}\sum_{l=1}^{n} \omega_l (x_l^2 + y_l^2) \ , \qquad \omega_l > 0 \ .$$

We assume that H_1 is a homogeneuous polynomial of degree 3 as in the Hénon–Heiles model, although this can be easily generalized. In the Hénon–Heiles model, the number of particles is $n = 2$ and they move in one–dimensional space $\mathbb{R}$.

Diagonalization of the Hamiltonian H_0 generating a linear evolution By transforming to complex coordinates

$$\left.\begin{array}{l} x_l = \dfrac{1}{\sqrt{2}}(\xi_l + i\eta_l) \\ y_l = \dfrac{i}{\sqrt{2}}(\xi_l - i\eta_l) \end{array}\right\} \quad \longleftrightarrow \quad \left\{\begin{array}{l} \xi_l = \dfrac{1}{\sqrt{2}}(x_l - iy_l) \\ \eta_l = \dfrac{-i}{\sqrt{2}}(x_l + iy_l) \end{array}\right. \tag{4.1}$$

we get

$$H_0 = \sum_{l=1}^{n} \omega_l I_l \ , \qquad I_l := i\xi_l\eta_l \ .$$

The linear Hamiltonian system defined by H_0 has n independent first integrals, namely

$$I_l = \frac{1}{2}(x_l^2 + y_l^2) = i\xi_l\eta_l \ , \qquad l \in \{1, 2, \ldots, n\} \ .$$

Poisson brackets are easy to compute also in complex coordinates:

Lemma 4.1 (Poisson bracket in complex coordinates)**.** *Given two differentiable functions f and g of the momenta we have*

$$\{f,g\} = \sum_{l=1}^{n} \left(\frac{\partial f}{\partial \xi_l}\frac{\partial g}{\partial \eta_l} - \frac{\partial f}{\partial \eta_l}\frac{\partial g}{\partial \xi_l} \right) \ .$$

Proof. Considering the case $n = 1$ for simplicity we find

$$\frac{\partial \xi}{\partial x} = \frac{1}{\sqrt{2}} = \frac{\partial \eta}{\partial y} \ , \qquad \frac{\partial \xi}{\partial y} = -\frac{i}{\sqrt{2}} = \frac{\partial \eta}{\partial x} \ .$$

So using the chain rule we get

$$\begin{aligned} \{f,g\} &= \frac{\partial f}{\partial q}\frac{\partial g}{\partial p} - \frac{\partial g}{\partial q}\frac{\partial f}{\partial p} = \frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y} \\ &= \frac{\partial f(\xi,\eta)}{\partial x}\frac{\partial g(\xi,\eta)}{\partial y} - \frac{\partial g(\xi,\eta)}{\partial x}\frac{\partial f(\xi,\eta)}{\partial y} \\ &= \left(\frac{\partial f}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial f}{\partial \eta}\frac{\partial \eta}{\partial x}\right)\left(\frac{\partial g}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial g}{\partial \eta}\frac{\partial \eta}{\partial y}\right) - \left(\frac{\partial g}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial g}{\partial \eta}\frac{\partial \eta}{\partial x}\right)\left(\frac{\partial f}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial f}{\partial \eta}\frac{\partial \eta}{\partial y}\right) \\ &= \frac{\partial f}{\partial \xi}\frac{\partial g}{\partial \eta} - \frac{\partial f}{\partial \eta}\frac{\partial g}{\partial \xi} \ . \end{aligned}$$

The general case is easily obtained reinstating the indices. □

Goal: construct a first integral of the interacting system H. We construct a formal power series starting as a perturbation of an I_l: the tentative first integral of the interacting system should take the form

$$\phi(x,y) = \phi_0(x,y) + \varepsilon\phi_1(x,y) + \varepsilon^2\phi_2(x,y) + \varepsilon^3\phi_3(x,y) + \dots$$

where, for some arbitrarily picked l,

$$\phi_0(x,y) = I_l(x,y) \ ,$$

and ϕ_s is a homogeneous polynomial of degree $s+2$.

Poincaré's construction We have to solve the Poisson bracket

$$\{\phi, H\} = 0$$

for ϕ, iteratively order by order in ε:

$$\{\phi_0 + \varepsilon\phi_1 + \varepsilon^2\phi_2 + \varepsilon^3\phi_3 + \dots, H_0 + \varepsilon H_1\} = 0 \ .$$

This yields the set of equations

$$\begin{aligned}
&\text{at order } \varepsilon^0 : && \{\phi_0, H_0\} = 0 \\
&\text{at order } \varepsilon^1 : && \{\phi_1, H_0\} + \{\phi_0, H_1\} = 0 \\
&\text{at order } \varepsilon^2 : && \{\phi_2, H_0\} + \{\phi_1, H_1\} = 0 \\
&\text{at order } \varepsilon^3 : && \{\phi_3, H_0\} + \{\phi_2, H_1\} = 0 \ .
\end{aligned}$$

Introducing the linear operator $\partial_{\boldsymbol{\omega}}$ (parametrized by $\boldsymbol{\omega} = (\omega_1, \omega_2, \dots)$ the frequencies appearing in H_0) acting on polynomials or functions by

$$\partial_{\boldsymbol{\omega}} f := \{f, H_0\}$$

we can write this system as

$$\partial_{\boldsymbol{\omega}}\phi_0 = 0 \ , \qquad \partial_{\boldsymbol{\omega}}\phi_s = \{H_1, \phi_{s-1}\} \quad \forall s \in \mathbb{N} \setminus \{0\} \ .$$

The first equation is satisfied by choosing $\phi_0 = I_l$ for some l. The second equation, called *homological equation*, shows us how to obtain iteratively the higher orders from the previously computed order (ϕ_{s-1} is known, solve for ϕ_s).

Note that in our case, the Hénon–Heiles model, all involved functions are polynomials, so that we can implement this construction as a symbolic computation on a computer with multi–variable polynomials.

Inversion of the Operator $\partial_{\boldsymbol{\omega}}$. Unfortunately the operator $\partial_{\boldsymbol{\omega}}$ has a non–trivial null space and is therefore not invertible (one can easily write down homogeneuous polynomials ψ such that $\partial_{\boldsymbol{\omega}}\psi = 0$). However, in some cases (such as the non–resonant Hénon–Heiles model), these do not appear, and on the relevant subspace the operator $\partial_{\boldsymbol{\omega}}$ can be inverted. Explicit inversion is obtained by diagonalizing it; this is achieved by using the complex coordinates as follows:

Lemma 4.2 (Diagonalization of $\partial_{\boldsymbol{\omega}}$)**.** *The linear operator $\partial_{\boldsymbol{\omega}}$ is diagonal over the basis*

$$\xi^{\boldsymbol{j}}\eta^{\boldsymbol{k}} := \prod_{l=1}^{n} \xi_l^{j_l}\eta_l^{k_l} \ ,$$

where $\boldsymbol{j} = (j_1, j_2, \ldots) \in \mathbb{N}^n$ and $\boldsymbol{k} = (k_1, k_2, \ldots) \in \mathbb{N}^n$ are multi–indices. One has

$$\partial_{\boldsymbol{\omega}}\xi^{\mathbf{j}}\eta^{\mathbf{k}} = i\langle\boldsymbol{\omega}, \mathbf{j} - \mathbf{k}\rangle\xi^{\mathbf{j}}\eta^{\mathbf{k}}$$

with the Euclidean scalar product in $\mathbb{R}^n$ as

$$\langle\boldsymbol{\omega}, \mathbf{j} - \mathbf{k}\rangle := \sum_{m=1}^{n} \omega_m(j_m - k_m) \ .$$

Proof. Using Lemma 4.1, check that for any differentable function f of the position and momenta we have

$$\partial_{\boldsymbol{\omega}} f = \sum_{m=1}^{n} i\omega_m \left(\xi_m \frac{\partial}{\partial\xi_m} - \eta_m \frac{\partial}{\partial\eta_m}\right) f \ .$$

One then computes

$$\begin{aligned} \partial_{\boldsymbol{\omega}}\xi_l^{j_l}\eta_l^{k_l} &= i\omega_l\xi_l \frac{\partial\xi_l^{j_l}}{\partial\xi_l}\eta_l^{k_l} - i\omega_l\eta_l \frac{\partial\eta_l^{k_l}}{\partial\eta_l}\xi_l^{j_l} \\ &= i\omega_l j_l\xi_l^{j_l}\eta_l^{k_l} - i\omega_l k_l\xi_l^{j_l}\eta_l^{k_l} \qquad = i\omega_l(j_l - k_l)\xi_l^{j_l}\eta_l^{k_l} \ . \end{aligned}$$

The multi–index case follows easily. □

Solving the Homological Equation We can therefore attempt to solve the equation $\partial_{\boldsymbol{\omega}}\phi = \psi$ for ϕ as follows.

- transform the polynomial ψ into complex coordinates, i. e., expand in the basis $\xi^{\mathbf{j}}\eta^{\mathbf{k}}$:

$$\psi = \sum_{\mathbf{j},\mathbf{k}\in\mathbb{N}^n} \psi_{\mathbf{j},\mathbf{k}}\xi^{\mathbf{j}}\eta^{\mathbf{k}} \ .$$

- $\psi = \partial_{\boldsymbol{\omega}}\phi$ becomes, with $\phi_{\mathbf{j},\mathbf{k}}$ to be determined:

$$\sum_{\mathbf{j},\mathbf{k}} \psi_{\mathbf{j},\mathbf{k}}\xi^{\mathbf{j}}\eta^{\mathbf{k}} = \partial_{\boldsymbol{\omega}} \sum_{\mathbf{j},\mathbf{k}\in\mathbb{N}^n} \phi_{\mathbf{j},\mathbf{k}}\xi^{\mathbf{j}}\eta^{\mathbf{k}} \tag{4.2}$$

$$= \sum_{\mathbf{j},\mathbf{k}\in\mathbb{N}^n} \phi_{\mathbf{j},\mathbf{k}} i\langle\boldsymbol{\omega}, \mathbf{j} - \mathbf{k}\rangle\xi^{\mathbf{j}}\eta^{\mathbf{k}} \ . \tag{4.3}$$

- If all factors $i\langle\boldsymbol{\omega}, \mathbf{j} - \mathbf{k}\rangle \neq 0$ (we say that there are *no resonances*) then the equation is solved by setting

$$\phi_{\mathbf{j},\mathbf{k}} := \frac{-i}{\langle\boldsymbol{\omega}, \mathbf{j} - \mathbf{k}\rangle}\psi_{\mathbf{j},\mathbf{k}} \ .$$

  In the following we are going to discuss the absence of resonances in the non–resonant Hénon–Heiles model, so that this procedure in fact works.

Absence of Resonances in non–resonant Hénon–Heiles. We say that the pair $(\mathbf{j}, \mathbf{k})$ is a *resonance* if

$$\langle \boldsymbol{\omega}, \mathbf{j} - \mathbf{k} \rangle = 0 \, .$$

Note that $\mathbf{j} - \mathbf{k} \in \mathbb{Z}^n$. Furthermore $n = 2$ in the Hénon–Heiles model. So if $\omega_1 \in \mathbb{Q}$ and $\omega_2 \in \mathbb{R} \setminus \mathbb{Q}$, then the only solution $\mathbf{j} - \mathbf{k} \in \mathbb{Z}^n$ of $\langle \boldsymbol{\omega}, \mathbf{j} - \mathbf{k} \rangle = 0$ is

$$\mathbf{j} - \mathbf{k} = 0 \qquad \Leftrightarrow \qquad \mathbf{j} = \mathbf{k} \, .$$

One may verify that this does not happen in the non–resonant Hénon–Heiles model (theory to be discussed separately).

4.1 Implementation for the non–resonant Hénon–Heiles model

The goal is to implement the operations in terms of polynomials usign computer algebra. The relevant formulas are, according to Eq. (2.17), as follows:

$$\begin{aligned} H &:= H_0 + H_1 \qquad (\varepsilon = 1) \\ H_0 &:= \frac{\omega_1}{2}(y_1^2 + x_1^2) + \frac{\omega_2}{2}(y_2^2 + x_2^2) \, , \qquad \omega_1 := 1 \, , \; \omega_2 := \frac{\sqrt{5} - 1}{2} \\ H_1 &:= x_1^2 x_2 - \frac{1}{3} x_2^3 \, . \end{aligned}$$

We pick as the starting point of our construction

$$\phi_0 = I_1 \, .$$

With the complex coordinates as defined in Eq. (4.1) the model takes the form

$$\begin{aligned} H_0 &= \omega_1 I_1 + \omega_2 I_2 \, , \quad I_1 = i\xi_1\eta_1 \, , \quad I_2 = i\xi_2\eta_2 \\ H_1 &= \frac{1}{2^{3/2}} \Big(\xi_1^2\xi_2 + \xi_1^2 i\eta_2 - \eta_1^2\xi_2 - i\eta_1^2\eta_2 + 2i\xi_1\eta_1\xi_2 - 2\xi_1\eta_1\eta_2 \\ &\qquad - \frac{1}{3}\xi_2^3 - i\xi_2^2\eta_2 + \frac{1}{3} i\eta_2^3 + \xi_2\eta_2^2 \Big) \end{aligned}$$

For the computation by hand the following two lemmata are useful. (Instead, for the implementation on the computer it is easier to use Lemma 4.1 to compute Poisson brackets!).

Lemma 4.3 (Poisson bracket expansion)**.** *For functions A, B, C we have*

$$\{AB, C\} = A\{B, C\} + \{A, C\}B \, .$$

Proof. trivial computation □

Lemma 4.4 (Fundamental Poisson brackets in complex coordinates)**.**

$$\{\xi, \xi\} = 0 = \{\eta, \eta\} \, , \qquad \{\eta, \xi\} = -1 \, , \qquad \{\xi, \eta\} = 1 \, .$$

Proof. trivial computation □

Repeatedly expanding the Poisson brackets until the fundamental ones can be used, one may also compute by hand that

$$\{H_1, i\xi_1\eta_1\} = \frac{1}{\sqrt{2}} \left(i\xi_1^2\xi_2 - \eta_2\xi_1^2 + i\xi_2\eta_1^2 - \eta_2\eta_1^2 \right) \, . \tag{4.4}$$

Check that you get this right!

Comment: One can in principle also use this combinatorial approach to do the computer algebra of Poisson brackets by repeated expansion using Lemma 4.3 and Lemma 4.4. However, it is easier to not implement Poisson brackets by expansion to fundamental Poisson brackets, but compute them using Lemma 4.1.

Solving the homological equation. We need to solve $\partial_{\boldsymbol{\omega}}\phi = \psi$ for ϕ, with the given $\psi = \{H_1, i\xi_1\eta_1\}$ that we just computed. The exponents are read of as follows for the four summands in Eq. (4.4):

$$\begin{aligned}
&\xi_1^2\xi_2 = \xi_1^{j_1}\xi_2^{j_2}\eta_1^{k_1}\eta_2^{k_2} && \Rightarrow \quad \mathbf{j} = (2,1)\,, \quad \mathbf{k} = (0,0)\\
&\xi_1^2\eta_2 = \cdots && \Rightarrow \quad \mathbf{j} = (2,0)\,, \quad \mathbf{k} = (0,1)\\
&\xi_2\eta_1^2 && \Rightarrow \quad \mathbf{j} = (0,1)\,, \quad \mathbf{k} = (2,0)\\
&\eta_2\eta_1^2 && \Rightarrow \quad \mathbf{j} = (0,0)\,, \quad \mathbf{k} = (2,1)\,.
\end{aligned}$$

Note that the resonant case $\mathbf{j} = \mathbf{k}$ never appears, so there is no problem.

We make the ansatz

$$\phi = a\xi_1^2\xi_2 + b\eta_2\xi_1^2 + c\xi_2\eta_1^2 + d\eta_2\eta_1^2\,,$$

with $a, b, c, d \in \mathbb{C}$ to be determined, and plug it into the homological equation. On the left hand side we find

$$\partial_{\boldsymbol{\omega}}\phi = i\langle\boldsymbol{\omega}, \begin{pmatrix}2\\1\end{pmatrix} - \begin{pmatrix}0\\0\end{pmatrix}\rangle\; a\;\xi_1^2\xi_2 + \ldots$$

and on the right hand side

$$\psi = \frac{i}{\sqrt{2}} \qquad\qquad \xi_1^2\xi_2 + \ldots$$

Comparing term by term we get

$$a = \frac{1}{\sqrt{2}}\frac{1}{\langle\boldsymbol{\omega}, \begin{pmatrix}2\\1\end{pmatrix}\rangle}\,, \qquad b = \frac{i}{\sqrt{2}}\frac{1}{\langle\boldsymbol{\omega}, \begin{pmatrix}2\\-1\end{pmatrix}\rangle}\,, \qquad \ldots$$

The total first order correction is then

$$\begin{aligned}
\phi_1 = \frac{1}{\sqrt{2}}\Bigg(&\langle\boldsymbol{\omega}, \begin{pmatrix}2\\1\end{pmatrix}\rangle^{-1}\xi_1^2\xi_2 + i\langle\boldsymbol{\omega}, \begin{pmatrix}2\\-1\end{pmatrix}\rangle^{-1}\eta_2\xi_1^2\\
&+ \langle\boldsymbol{\omega}, \begin{pmatrix}-2\\1\end{pmatrix}\rangle^{-1}\xi_2\eta_1^2 + i\langle\boldsymbol{\omega}, \begin{pmatrix}-2\\-1\end{pmatrix}\rangle^{-1}\eta_2\eta_1^2\Bigg)\\
= \frac{1}{\sqrt{2}}\Bigg(&\frac{1}{2\omega_1+\omega_2}\xi_1^2\xi_2 + i\frac{1}{2\omega_1-\omega_2}\eta_2\xi_1^2\\
&+ \frac{1}{-2\omega_1+\omega_2}\xi_2\eta_1^2 + i\frac{1}{-2\omega_1-\omega_2}\eta_2\eta_1^2\Bigg)\,.
\end{aligned}$$

Again, check that you get this right both by hand and in your program.

Solution Your program should produce the following result for the first few orders (don't forget the prefactor $1/\sqrt{2}$ to get the numerical coefficients right, and stay attentive not to permute coordinates accidentally):

$$\begin{aligned}
\phi_1 &= (0.27)\,\xi_1^2\xi_2 + (0.51i)\,\xi_1^2\eta_2 + (-0.51)\,\eta_1^2\xi_2 + (-0.27i)\,\eta_1^2\eta_2 \\
\phi_2 &= (0.02)\,\xi_1^4 + (0.09i)\,\xi_1^3\eta_1 + (-0.14)\,\xi_1^2\xi_2^2 + (-0.64i)\,\xi_1^2\xi_2\eta_2 + (1.06)\,\xi_1^2\eta_2^2 \\
&\quad + (-0.09i)\,\xi_1\eta_1^3 + (0.28i)\,\xi_1\eta_1\xi_2^2 + (-0.28i)\,\xi_1\eta_1\eta_2^2 + (0.02)\,\eta_1^4 + (1.06)\,\eta_1^2\xi_2^2 \\
&\quad + (0.64i)\,\eta_1^2\xi_2\eta_2 + (-0.14)\,\eta_1^2\eta_2^2 \\
\phi_3 &= (-0.03)\xi_1^4\xi_2 + (-0.15i)\xi_1^4\eta_2 + (-0.24i)\xi_1^3\eta_1\xi_2 + (0.75)\xi_1^3\eta_1\eta_2 + (-0.01)\xi_1^2\eta_1^2\xi_2 \\
&\quad + (-0.01i)\xi_1^2\eta_1^2\eta_2 + (0.14)\xi_1^2\xi_2^3 + (0.79i)\xi_1^2\xi_2^2\eta_2 + (-2.72)\xi_1^2\xi_2\eta_2^2 + (-15.19i)\xi_1^2\eta_2^3 \\
&\quad + (0.75i)\xi_1\eta_1^3\xi_2 + (-0.24)\xi_1\eta_1^3\eta_2 + (-0.59i)\xi_1\eta_1\xi_2^3 + (1.78)\xi_1\eta_1\xi_2^2\eta_2 \\
&\quad + (1.78i)\xi_1\eta_1\xi_2\eta_2^2 + (-0.59)\xi_1\eta_1\eta_2^3 + (-0.15)\eta_1^4\xi_2 + (-0.03i)\eta_1^4\eta_2 + (-15.19)\eta_1^2\xi_2^3 \\
&\quad + (-2.72i)\eta_1^2\xi_2^2\eta_2 + (0.79)\eta_1^2\xi_2\eta_2^2 + (0.14i)\eta_1^2\eta_2^3
\end{aligned}$$

Practical task: Check that with your program you get the correct result for ϕ_1, ϕ_2, and ϕ_3.

4.2 Analysis of truncated first integrals along leapfrog trajectories

We now sum up the first constructed ϕ_i to get an approximate constant of motion. We evaluate the resulting polynomial along a Leapfrog trajectory of the non-resonant Hénon–Heiles model (alternatively, a SABA3 trajectory, or a Leapfrog trajectory combined with analytical solution).

Practical task: Plot the value of the constant of motion $\phi_0 + \varepsilon\phi_1 + \varepsilon^2\phi_2 + \varepsilon^3\phi_3 + \ldots$ along a Leapfrog trajectory of Hénon–Heiles as a function of time. Increase the order of Poincaré's construction and compare the results.

- In the beginning, you can choose the initial data $q_0, q_1, p_1 = 0$, fix the energy to be half of E_{crit} and compute the corresponding p_0.
- Run the Leapfrog eqvolution. Convert the Leapfrog coordinates to the complex coordinates and plug them in the first integral. Every 100 Leapfrog steps plot the real part of the first integral (the imaginary part is zero by construction). Check the conservation of the first integral. (Don't evaluate the integral at each step, it would take too much time.)
- As a first check of your code, plot $\phi_0 = I_0$ (or I_1), i.e., the energy of the left (or the right) oscillator. Continue adding the next ϕ_i and compare with the previous results.
- Try at least three different energies. Change the initial data too.

Welford's online algorithm for the variance. To analyze the conservation of the first integral is a quantitative way we can compute the variance:

$$\frac{1}{N}\sum_{t=0}^{N}(\phi(t)-\bar{\phi})^2 \tag{4.5}$$

where N is the number of points calculated for the plot and $\bar{\phi}$ is the time average. In general, given a random variable X, its variance is given by

$$Var(X) = \mathbb{E}\big(X-\mathbb{E}(X)\big)^2 = \mathbb{E}(X^2) - \big(\mathbb{E}(X)\big)^2$$

To avoid computing the points and the sums two times, we can use Welford's online algorithm. We call the average

$$\bar{x}_N = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

the sum of squares

$$M_{2,N} = \sum_{i=1}^{N}(x_i-\bar{x}_N)^2$$

and the variance

$$\sigma_N^2 = \frac{M_{2,N}}{N}$$

Welford's iterative algorithm is

$$\bar{x}_N = \bar{x}_{N-1} + \frac{x_N-\bar{x}_{N-1}}{N}$$
$$M_{2,N} = M_{2,N-1} + (x_N-\bar{x}_{N-1})(x_N-\bar{x}_N)$$

Practical task: (Optional) Compute the variance (4.5).

4.3 Poincaré sections vs. contour lines of first integrals

We now compare the level sets of the constructed approximate constants of motion to the Poincaré sections we obtained in Lab 5.

Practical task: Write an algorithm that samples sufficiently many points to produce a good plot of the level sets.

1. Fix an energy E_0, for example 1/3 of the critical energy.
2. Fix $q_0 = 0$, and two values q_1 and p_1 (for the choice of q_1 and p_1 see (4.6) below).
3. Solve analytically the equation $E_0 = H(0, p_0, q_1, p_1)$ for p_0 as a function of E, q_1 and p_1 (as we did when programming Poincaré sections).
4. Pick your first integral ϕ, constructed started from I_1 (not from I_0).
5. Set $\tilde{\phi} = \phi(0, p_0, q_1, p_1)$, with p_0 determined as in point 3.
6. The function $\phi\big(0, p_0(E, q_1, p_1), q_1, p_1\big)$ with $p_0(q_1, p_1)$ determined as in 3. is a function of two variables. Therefore we can set it equal to $\tilde{\phi}$ and solve (by implicit function theorem) for $p_1 = p_1(q_1)$. This defines the level lines.
7. The solution $p_1 = p_1(q_1)$ via implicit function theorem is of course not analytically computable. We take a numerical approach to plot the level lines: put a square lattice over the graph spanned by p_1 and q_1, loop over all edges, and if you detect a change of sign of $\phi(0, p_0(E, x, y), x, y) - \tilde{\phi}$, determine x (or y) along the edge (here you can also use linear interpolation). Draw a point there.

Example of q_1 and p_1:

$$p_1 = 0, \qquad q_1 \in \{-0.18, -0.1, -0.07, ..., 0.06\} \tag{4.6}$$

Practical task: Draw now the Poincaré section $q_0 = 0$, with the same energy $E = E_{\text{crit}}/3$ and using the same set of initial data (4.6). Compare with the corresponding level sets. What do you observe? Test different energies.

5 Possible Calendar for a Semester of 12 Weeks

Week 1 Introduction to Linux. Installing Mizar. Crash course in C.
Homework: Prepare your computer. Do a simple exercise in C.

Week 2 Makefiles. Programming with Mizar. Forward Euler method.
Homework: Program Euler method for simple pendulum, double pendulum, coupled harmonic oscillators, Lotka-Volterra aka Predator-Prey model. Phase space and energy plot for mechanical models.

Week 3 Leapfrog method.
Homework: Compute phase space plot of harmonic oscillator with leapfrog. Evolve to a fixed time T, subtract the analytical solution, and repeat this procedure for different time step resolutions h (i. e., analyze how the global error scales with h), use a logarithmic plot to show that Euler is first order and Leapfrog second order.

Verify that the energy is (up to an error that does **not** grow with time) conserved by Leapfrog.

Coupled harmonic oscillators: compute a phase portrait in the plane (q_1, p_1) by Leapfrog. Plot the "left half" of the energy, $H_{\text{left}}(q_1, p_1) = \frac{1}{2}\left(p_1^2 + \omega^2 q_1^2\right)$ as a function of time. Observations?

Week 4 Analytic solution for linearly coupled harmonic oscillators. Hénon–Heiles model. Hénon's algorithm for computing Poincaré sections.

Homework: Use Leapfrog to compute Poincaré sections for the linearly coupled oscillators and Hénon–Heiles.

Week 5 Construction of splitting methods. SABA3 method.

Homework: SABA3 for the simple pendulum $H = T + V$, where $T = p^2$, $V = \varepsilon(1 - \cos(\theta))$, taking $\varepsilon = 1$.

SABA3 for the harmonic oscillator $H = p^2 + q^2$. Evolve up to a fixed time T and make a log plot of the error as a function of h.

Week 6 Improved splitting methods: correctors and alternative splittings.

Homework: Analyze the global error of SABA3 + corrector in a logarithmic plot. Use the splitting into quadratic Hamiltonian + interaction to plot Poincaré sections of the Hénon–Heiles model.

Week 7 Computer algebra with polynomials in one complex variable and complex coefficients.

Homework: Program the following functions:

- Sum of two polynomials.
- Multiplication of a polynomial with a complex number.
- Product of two polynomials.
- Derivative (algebraically) of a polynomial.
- Evaluation of a polynomial in a given point.
- Creating a human-readable string from a polynomial.

Week 8 Computer algebra with polynomials of four complex variables.

Homework: Implement the same functions as for single-variable polynomials, where the derivative should be partial derivatives with respect to the four different variables.

Week 9 Using our "library" of functions for polynomials to implement Poincaré's perturbative construction.

Homework: Implement a function that computes the Poisson bracket of two polynomials (where two of the four variables represent the momenta, and two variables represent the positions). Implement a function that solves the homological equation for the non–resonant case. (Your program should produce an error message if it encounters division by zero and then terminate in a controlled way.)

Week 10 We implement Poincaré's iterative method.

Homework: Compute by hand the complex representation of the interaction Hamiltonian H_1. Insert this data into your program as a polynomial. Run Poincaré's method to obtain at least ϕ_3. Compare to the result given above and make sure you get all coefficients right. (This will probably take some time.)

Week 11 We check how well the constructed polynomials are conserved along the time evolution.

Homework: Sum up the first constructed order ϕ_i to get an approximate constant of motion. Evaluate the polynomial along a leapfrog trajectory of the full Hénon–Heiles model. Plot the value as a function of time and compare how constant the result is as you increase the order of Poincaré's construction. Compute the standard deviation using Welford's online algorithm (you will have to take a subset of time steps otherwise this will become too slow – but make sure to not pick only special points).

Week 12 Compare the level sets of the constructed approximate constants of motion to the Poincaré sections we obtained in the earlier weeks.

Homework: Write an algorithm that samples sufficiently many points to produce a good plot of the level sets.

Final task Write a summary of the lab course: about a third to half a page of text for every week. Explain in *few* sentences what you did, what you learned, and show and discuss some screenshots.

References

[DR05] Denis Donnelly and Edwin Rogers. Symplectic integrators: An introduction. *American Journal of Physics*, 73(10):938–945, October 2005.

[Gio20] Antonio Giorgilli. Metodi e Modelli Matematici per le applicazioni. http://www.mat.unimi.it/users/antonio/metmod/metmod.html, 2020.

[Hen82] M. Henon. On the numerical computation of Poincaré maps. *Physica D: Nonlinear Phenomena*, 5(2):412–414, September 1982.

[LR01] Jacques Laskar and Philippe Robutel. High order symplectic integrators for perturbed Hamiltonian systems. *Celestial Mechanics and Dynamical Astronomy*, 80(1):39–62, May 2001.

[Tuc02] Warwick Tucker. Computing accurate Poincaré maps. *Physica D: Nonlinear Phenomena*, 171(3):127–137, October 2002.

[Wik22] Wikipedia. Hénon–Heiles system. *Wikipedia*, February 2022.