# A Generic Implementation of a Truncated Power Series Algebra in Julia for Beam Dynamics Modelling

Master Thesis

**Author:** Matthieu Melennec[1]

**Supervisor:** Dr. Andreas Adelmann[2]

[1] D-PHYS, ETH Zurich
[2] Paul Scherrer Institut

## Abstract

This project was aimed at implementing a beam dynamics model in Julia. The dynamics are described as a Hamiltonian system and the transfer maps are obtained form the Hamilton equations of motion. We represent these transfer maps as truncated power series, and implement symbolicaly using Julia symbolic computations packages. The symplecticity of maps is studied, and we implement Dragt-Finn factorisation to restore the symplecticity lost through finite order truncation.

Results obtained on systems like the harmonic oscillator, single beamline elements and a FODO beamline are presented. They allow us to study the performances of the package, in particular the runtime performance.

## Context

This document is the report to my master thesis, carried out during the 2022 Fall semester semester in the Accelerator Modelling and Advanced Simulations (AMAS) Group of the Laboratory for Simulation and Modelling (LSM) at the Paul Scherrer Institut (PSI) and under the supervision of Dr. Andreas Adelmann, towards the graduation of an MSc Physics at ETH Zürich.
Matthieu Melennec - 27.03.2023

# Contents

# Chapter 1

# Introduction

## 1.1 Mathematical Formalism

The Formulation of the mathematical background of the model heavily relies on Lie Algebras [1] and their properties. In this section, we define some concepts of Lie Algebras and introduce some interesting properties. Another key topic in this thesis was to ensure that the transfer maps of the system were symplectic [2] (see section 1.2). In this section we also define the symplectic group [3], symplectic maps [4], and present some relevant properties.

### 1.1.1 Lie Algebras

**Definition 1.1.1.** [1] A vector space $L$ over a field $F$ together with an operation

$$\begin{array}{rccc} [\cdot,\cdot]: & L\times L & \to & L \\ & (x,y) & \mapsto & [x,y] \end{array}$$

is a Lie Algebra if the operation $[\cdot,\cdot]$ satisfies the conditions

- **Bilinearity:**
  - $\forall\lambda \in F, \forall x,y \in L, [\lambda x, y] = [x, \lambda y] = \lambda[x,y]$
  - $\forall x_1, x_2, y \in L, [x_1 + x_2, y] = [x_1, y] + [x_2, y]$
  - $\forall x, y_1, y_2 \in L, [x, y_1 + y_2] = [x, y_1] + [x, y_2]$
- **Alternativity:** for any $x$ in $L$, $[x,x] = 0$.
- **Jacobi Identity:** for any $x,y,z$ in $L$,

$$[x,[y,z]] + [z,[x,y]] + [y,[z,x]] = 0 \tag{1.1}$$

The operation associated to a Lie Algebra is usually called its Lie bracket (sometimes commutator).

We introduce the Poisson bracket [3], usually used in Hamiltonian mechanics. For some $n \in \mathbb{N}$ we consider vectors in $\mathbb{R}^{2n}$ of the form $(q_1, p_1, \ldots, q_n, p_n)$, for $1 \le i \le n$. For any two functions $f, g$ in

$\mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right)$, we define the Poisson bracket as

$$\{f,g\} = \sum_{i=1}^{n} \frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial q_i} \tag{1.2}$$

**Proposition 1.1.1.** *[3] The set of infinitely differentiable functions $\mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right)$ equipped with the Poisson bracket is a Lie Algebra.*

*Proof.*

- **Bilinearity:** By linearity of the derivative, we have

$$\begin{aligned}\{kf,g\} &= \sum_i \frac{\partial kf}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial kf}{\partial p_i}\frac{\partial g}{\partial q_i}\\ &= k\left(\sum_i \frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial q_i}\right) = k\{f,g\}\\ &= \sum_i \frac{\partial f}{\partial q_i}\frac{\partial kg}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial kg}{\partial q_i} = \{f,kg\}\end{aligned}$$

- **Alternativity:** $\{f,f\} = \sum_i \frac{\partial f}{\partial q_i}\frac{\partial f}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial f}{\partial q_i} = 0$
- **Jacobi Identity:** We check that

$$\{f,\{g,h\}\} + \{h,\{f,g\}\} + \{g,\{h,f\}\}$$

First, let us expand the first term

$$\begin{aligned}\{f,\{g,h\}\} &= \left\{f, \sum_i \frac{\partial g}{\partial q_i}\frac{\partial h}{\partial p_i} - \frac{\partial g}{\partial p_i}\frac{\partial h}{\partial q_i}\right\}\\ &= \sum_{i,j} \frac{\partial f}{\partial q_j}\frac{\partial^2 g}{\partial p_j\partial q_i}\frac{\partial h}{\partial p_i} + \frac{\partial f}{\partial q_j}\frac{\partial g}{\partial q_i}\frac{\partial^2 h}{\partial p_j\partial p_i}\\ &\quad - \frac{\partial f}{\partial q_j}\frac{\partial^2 g}{\partial p_j\partial p_i}\frac{\partial h}{\partial q_i} - \frac{\partial f}{\partial q_j}\frac{\partial g}{\partial p_i}\frac{\partial^2 h}{\partial p_j\partial q_i}\end{aligned}$$

If we do the same with the terms $\{h,\{f,g\}\}$ and $\{g,\{h,f\}\}$ and play around with the indices $i,j$, each terms in the respective sums cancel one another and we recover $\{f,\{g,h\}\} + \{h,\{f,g\}\} + \{g,\{h,f\}\} = 0$.

□

As a consequence of this proposition, we introduce the *Lie operator* generated by a map $f \in \mathcal{C}^\infty(\mathbb{R}^{2n},\mathbb{R})$, defined as

$$\begin{array}{cccc} :f: \; : & \mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right) & \to & \mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right)\\ & g & \mapsto & \{f,g\}\end{array} \tag{1.3}$$

Let us denote by $\mathcal{P}$ the set of Lie operators generated by $\mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right)$. Observe that if we define the commutator

$$\begin{array}{cccc} [\cdot,\cdot]: & \mathcal{P}\times\mathcal{P} & \to & \mathcal{P}\\ & (:f:,:g:) & \mapsto & :f::g: - :g::f:\end{array}$$

the set $\mathcal{P}$ can be promoted to be a Lie Algebra. Indeed, the bilinearity of the Poisson bracket implies that of the commutator $[\cdot,\cdot]$ is itself bilinear. Then one observes that $[: f :, : f :] =: f :: f : - : f :: f := 0$. We finally verify the Jacobi Identity. Let $: f :, : g :, : h :$ be Lie operators. To make the following easier to read, we drop the colons and denote these operators as $f, g, h$ respectively.

$$\begin{aligned}[f,[g,h]] + [h,[f,g]] + [g,[h,f]] &= fgh - fhg - ghf + hgf \\ &+ hfg - hgf - fgh + gfh \\ &+ ghf - gfh - hfg + fhg \\ &= 0\end{aligned}$$

We now define the Lie transformation generated by $\mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right)$, which will find its use in section 1.5

**Definition 1.1.2.** Consider the $\mathcal{C}^\infty$ map $f : \mathbb{R}^{2n} \to \mathbb{R}$. The Lie transformation associated with $f$ is the series defined via

$$\exp(: f :) = \sum_{k=0}^{\infty} \frac{(: f :)^k}{k!} \tag{1.4}$$

where $: f :^k$ defines a compsition of Lie operators $: f :$ $k$ times:

$$: f :^k g = \{\underbrace{f, \ldots, \{f, \{f}_{k \text{ times}}, g\}\} \ldots\}$$

### 1.1.2 The Lie Exchange Formula

A key theorem when using Lie operators is the Lie exchange formula, given by theorem 1.1.1[2, 5].

**Theorem 1.1.1.** *Let $f, g$ be maps on $\mathbb{R}$. Then*

$$e^{:f:}e^{:g:} = e^{:g:}e^{:h:}$$

*with $h = e^{:g:}f$.*

Before we can prove this theorem, we must introduce some definitions [3] and lemmas [6, 2]

**Definition 1.1.3.** Let $f$ be a map in $\mathcal{C}^\infty\left(\mathbb{R}^{2n},\mathbb{R}\right)$. We denote the adjoint of the Lie operator $: f :$ as $\#f\#$. It acts on any Lie operator $: g :\in \mathcal{P}$ following the rule [7]

$$\#f\# : g := [: f :, : g :]$$

**Lemma 1.1.1.** *For any Lie operator $: f :\in \mathcal{P}, e^{:f:}e^{-:f:} = id_{2n}$.*

Note that it also holds that $- : f :=: -f :$.

*Proof.* Let $: f :$ be a Lie operator. By explicit computation [6]:

$$\begin{aligned}e^{-:f:}e^{:f:} &= \sum_{k,k'=0}^{\infty} \frac{(-1)^{k'}}{k!k'!} : f :^{k+k'} \\ &= \sum_{m=0}^{\infty} : f :^m \sum_{k=0}^{m} \frac{(-1)^k}{k!(m-k)!}\end{aligned}$$

We claim that for any $m \geq 1, \mathcal{S}_m := \sum_{k=0}^{m} \frac{(-1)^k}{k!(m-k)!} = 0$. First, one must note that for any $m \geq 1$, we have

$$\mathcal{S}_m = \sum_{k=0}^{m} \frac{(-1)^k}{m!} \binom{m}{k}$$

It therefore suffices to prove that for any $m \geq 1, s_m := \sum_{k=0}^{m} (-1)^k \binom{m}{k} = 0$. One proves this by induction on $m$. Indeed, for $m = 1, s_1 = (-1)^0 \binom{1}{0} + (-1)^1 \binom{1}{1} = 1 - 1 = 0$.
Let us therefore assume that it holds $s_m = 0$ for some $m \geq 1$:

$$\begin{aligned}
\sum_{k=0}^{m+1} (-1)^k \binom{m+1}{k} =& (-1)^{m+1} + (-1)^0 + \sum_{k=1}^{m} (-1)^k \binom{m+1}{k} \\
& \text{Using Pascal's Identity:} \\
=& (-1)^{m+1} + 1 + \sum_{k=1}^{m} (-1)^k \left[ \binom{m}{k-1} + \binom{m}{k} \right] \\
=& (-1)^{m+1} + \sum_{k=1}^{m} (-1)^k \binom{m}{k-1} \\
& \underbrace{+ 1 + \sum_{k=1}^{m} (-1)^k \binom{m}{k}}_{=0 \text{ by assumption}} \\
=& -(-1)^m - \sum_{k=0}^{m-1} (-1)^k \binom{m}{k} \\
=& -\sum_{k=0}^{m} (-1)^k \binom{m}{k} = 0
\end{aligned}$$

which proves our claim. We therefore have $e^{-:f:} e^{:f:} =: f :^0 = \mathrm{id}_{2n}$, concluding the proof. □

**Lemma 1.1.2.** *[3, 8] For any two maps $f, g$ on $\mathbb{R}, \# f \# : g :=: \{f, g\} :$.*

*Proof.* Let $h : \mathbb{R}^{2n} \to \mathbb{R}$. By the Jacobi Identity (1.1),

$$\begin{aligned}
[: f :, : g :] h =& : f :: g : h - : g :: f : h \\
=& \{f, \{g, h\}\} - \{g, \{f, h\}\} \\
=& -\{h, \{f, g\}\} = \{\{f, g\}\} \\
=& : \{f, g\} :
\end{aligned}$$

□

**Lemma 1.1.3.** *[3] For any $f, g : \mathbb{R}^{2n} \to \mathbb{R}$, we have $e^{\# f \#} : g :=: e^{:f:} g :$.*

*Proof.* First, for any integer $k$ and for any real maps $f, g$, we have

$$\begin{aligned}
\#f\#^k : g : &= \#f\#^{k-1} : \{f, g\} : \\
&= \#f\#^{k-2} : \{f, \{f, g\}\} : \\
&\qquad \dots \\
&=: \{\underbrace{f, \{f, \dots, \{f}_{k \text{ times}}, g\} \dots\}\} : \\
&=:: f :^k g : \qquad \text{by definition of } : f :^k
\end{aligned}$$

Hence

$$e^{\#f\#} : g := \left(\sum_{k=0}^{\infty} \frac{1}{k!} \#f\#^k\right) : g := \sum_{k=0}^{\infty} \frac{1}{k!} :: f :^k g :$$

By bilinearity of the Poisson Bracket,

$$e^{\#f\#} : g :=: \left(\sum_{k=0}^{\infty} \frac{1}{k!} : f :^k g\right) :=: e^{:f:} g :$$

□

We can now prove Theorem 1.1.1 [6, 2]

*Proof.* Let $f, g$ be two maps in $\mathcal{C}^\infty\left(\mathbb{R}^{2n}, \mathbb{R}\right)$. Then

$$\begin{aligned}
e^{:f:} e^{:g:} &= e^{:g:} e^{-:g:} e^{:f:} e^{:g:} \qquad \text{by Lemma 1.1.1} \\
&= e^{:g:} \sum_{k=0}^{\infty} \frac{1}{k!} e^{-:g:} : f :^k e^{:g:} \\
&= e^{:g:} \sum_{k=0}^{\infty} \left(\frac{1}{k!} e^{-:g:} : f : e^{:g:}\right)^k \\
&= e^{:g:} e^{:h:}
\end{aligned}$$

We claim that $h = e^{-:g:} f$.
Let $\tau \in \mathbb{R}$ be some parameter and let $C(\tau) = e^{\tau :g:} f e^{-\tau :g:}$. Then we obtain an ODE

$$\begin{gathered}
C(0) = f \\
\frac{dC}{d\tau} =: g : C - C : g := \{: g :, C\} = \#g\#C
\end{gathered}$$

Solving for $C(\tau)$, we obtain

$$C(\tau) = e^{\tau \#g\#} : f :$$

By lemma 1.1.2, $C(\tau) =: e^{\tau :g:} f :$, and finally, setting $\tau = -1$, we recover our claim and conclude the proof. □

### 1.1.3 The Symplectic Group

**Definition 1.1.4.** Let $n \in \mathbb{N}^*$ be a positive integer, and let $\mathbf{J} \in \mathbb{R}^{2n\times 2n}$ be a skew-symmetric matrix (i.e. $\mathbf{J}^T = -\mathbf{J}$) such that $\mathbf{J}^{-1} = -\mathbf{J}$. The Symplectic Group generated by $\mathbf{J}$ is the set $\mathrm{Sp}_{\mathbf{J}}(2n)$ of matrices $\mathbf{M} \in \mathbb{R}^{2n\times 2n}$ such that the *symplectic condition* is fulfilled:

$$\mathbf{M}^T\mathbf{J}\mathbf{M} = \mathbf{J}$$

Such matrices are called *symplectic matrices* [3].

**Proposition 1.1.2.** *[9] Let $n \in \mathbb{N}^*$ and $\mathbf{J}$ be the skew-symmetric matrix defining the symplectic group $Sp_{\mathbf{J}}(2n)$. The following hold*

- *The matrix $\mathbf{J}$ is symplectic: $\mathbf{J} \in Sp_{\mathbf{J}}(2n)$.*
- *Let $\mathbf{M} \in Sp_{\mathbf{J}}(2n)$. Then $\det(\mathbf{M}) = 1$.*
- *Let $\mathbf{M} \in Sp_{\mathbf{J}}(2n)$. Then $\mathbf{M}^T \in Sp_{\mathbf{J}}(2n)$.*
- *Let $\mathbf{M} \in Sp_{\mathbf{J}}(2n)$. Then $\mathbf{M}^{-1} = \mathbf{J}^{-1}\mathbf{M}^T\mathbf{J} \in Sp_{\mathbf{J}}(2n)$.*

*Proof.*

- We verify the symplectic condition: $\mathbf{J}^T\mathbf{J}\mathbf{J} = \mathbf{J}$ because $\mathbf{J}^{-1} = \mathbf{J}^T$.
- Observe first that since $\mathbf{J}$ is an even sized anti-symmetric matrix, it has non-zero Pfaffian $\mathrm{Pf}(\mathbf{J}) \neq 0$. Next, recall the Pfaffian identity: For a skew-symmetric $2n \times 2n$ matrix $\mathbf{A}$ and an arbitrary $2n \times 2n$ invertible matrix $\mathbf{B}$, we have $\mathrm{Pf}(\mathbf{B}^T\mathbf{A}\mathbf{B}) = \det(\mathbf{B})\mathrm{Pf}(\mathbf{A})$. Hence The Pfaffian Identity on the Symplectic condition gives

  $$\mathrm{Pf}\left(\mathbf{M}^T\mathbf{J}\mathbf{M}\right) = \det(\mathbf{M})\mathrm{Pf}(\mathbf{J}) = \mathrm{Pf}(\mathbf{J}) \quad \Rightarrow \quad \det(\mathbf{M}) = 1$$

  In particular, this implies that all symplectic matrices are invertible.
- First, observe that

  $$\left(\mathbf{M}^T\mathbf{J}\right)\left(\mathbf{M}\mathbf{J}\mathbf{M}^T\right)(\mathbf{J}\mathbf{M}) = \mathbf{J}\mathbf{J}\mathbf{J} = -\mathbf{J} = \mathbf{M}^T\mathbf{J}\mathbf{J}\mathbf{J}\mathbf{M} \tag{1.5}$$

  Since $\mathbf{M}$ is symplectic, it has determinant 1, and so does $\mathbf{M}^T$. Thus $\mathbf{M}^T\mathbf{J}$ and $\mathbf{J}\mathbf{M}$ are invertible. We therefore obtain, from the right-hand side of (1.5),

  $$\mathbf{M}\mathbf{J}\mathbf{M}^T = \left(\mathbf{M}^T\mathbf{J}\right)^{-1}\mathbf{M}^T\mathbf{J}\mathbf{J}\mathbf{J}\mathbf{M}\,(\mathbf{J}\mathbf{M})^{-1} = \mathbf{J}$$

  which proves that $\mathbf{M}^T$ fulfills the symplectyic condition and is therefore symplectic.
- We verify that $\left(\mathbf{J}^T\mathbf{M}^T\mathbf{J}\right)\mathbf{M} = \mathbf{J}^T\mathbf{J} = \mathbf{I}_{2n}$. Hence the inverse of a symplectic matrix $\mathbf{M}$ is given by $\mathbf{M}^{-1} = \mathbf{J}^T\mathbf{M}^T\mathbf{J}$. We verify the symplectic condition

  $$\left(\mathbf{J}^T\mathbf{M}^T\mathbf{J}\right)^T\mathbf{J}\left(\mathbf{J}^T\mathbf{M}^T\mathbf{J}\right) = \left(\mathbf{J}^T\mathbf{M}\mathbf{J}\right)\mathbf{J}\left(\mathbf{J}^T\mathbf{M}^T\mathbf{J}\right) = \mathbf{J}^T\mathbf{M}\mathbf{J}\mathbf{M}^T\mathbf{J} = \mathbf{J}$$

  because the transpose of a symplectic matrix is symplectic.

□

**Remark.** *Note that the above definition of $Sp_{\mathbf{J}}(2n)$ does indeed define a group: As a subset of $\in \mathbb{R}^{2n\times 2n}$, the product of two symplectic matrices in $Sp_{\mathbf{J}}(2n)$ is associative; $\mathbf{1}_{2n} \in Sp_{\mathbf{J}}(2n)$ is the identity; from proposition 1.1.2, symplectic matrices are invertible.*

Inspired by [4] and [10], we propose to define Symplectic maps (or symplectomorphism) via the Symplectic condition. First, some definitions, from [11].

**Definition 1.1.5.** Consider an infinitely differentiable manyfold $M$, and let $x \in M$. A tangent, or differential, at $x \in M$ is a linear map $D_x : \mathcal{C}^\infty(M) \to \mathbb{R}$ which follows the *Leibniz rule*:

$$\forall f, g \in \mathcal{C}^\infty(M), \quad D_x(fg) = g(x)D_x(f) + f(x)D_x(g)$$

The tangent space to $M$ at $x$ is the set of differentials at $x$. It is denoted $T_xM$.

**Definition 1.1.6.** Let $M, N$ be two manifolds. A symplectomorphism (or symplectic map) between $M$ and $N$ is a diffeomorphism $\varphi : M \to N$ such that for every $x \in M, d\varphi_x : T_xM \to T_{\varphi(x)}N$ is a symplectic matrix.

What this means in particular for us is that a symplectic map is a $\mathcal{C}^\infty$ map $\varphi : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ such that for any $\xi \in \mathbb{R}^{2n}$, the Jacobian $\mathrm{Jac}_\varphi(\xi)$ is a symplectic matrix.

## 1.2 Symplectic Formalism and the Liouville Theorem

From now on, we will express the phase space coordinates $\xi$ of a $n$ degrees of freedom system pairing together canonical coordinates and their associated momenta, i.e. writing $\xi = (q_1, p_1, \ldots, q_n, p_n)$. We then introduce the rearranging matrix $\mathbf{J}$. First, we introduce the antisymmetric matrix

$$\mathbf{j} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

One then builds the $2n \times 2n$ *rearranging matrix* as

$$\mathbf{J} = \begin{pmatrix} \mathbf{j} & 0 & \ldots & 0 \\ 0 & \mathbf{j} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{j} \end{pmatrix} \tag{1.6}$$

One may in particular notice that the rearranging matrix is an even sized skew-symmetric matrix, and that $\mathbf{J}^{-1} = -\mathbf{J}$. Hence the rearranging matrix generates the Symplectic group $\mathrm{Sp}_{\mathbf{J}}(2n)$.

In particular, one observes that the rearranging matrix allows us to write the Hamilton equations in the following form [6]

$$\xi' = \mathbf{J}\nabla H \tag{1.7}$$

usually called the *symplectic Hamilton equations* [9]. In particular, consider a canonical transformation $f : \xi \mapsto \zeta$, and let $K$ denote the new Hamiltonian. Since $f$ is a canonical transformation, $\zeta$ must fulfil the Hamilton Equations. For any index $i$,

$$\begin{aligned} \zeta_i' &= \frac{\partial \zeta_i}{\partial \xi_j}\xi_j' = (\mathrm{Jac}_f)_{ij}\,\xi_j' \\ \Rightarrow \qquad \zeta' &= \mathrm{Jac}_f \mathbf{J}\nabla H \end{aligned}$$

where $\mathrm{Jac}_f$ is the Jacobian matrix of $f$. Now, observe that

$$\begin{aligned} \frac{\partial H}{\partial \xi_i} &= \frac{\partial K}{\partial \zeta_j}\frac{\partial \zeta_j}{\partial \xi_i} = \mathrm{Jac}_f \frac{\partial K}{\partial \zeta_j} \\ \Rightarrow \qquad \nabla H &= \mathrm{Jac}_f^T \nabla K \end{aligned}$$

and therefore we obtain [6]

$$\zeta' = \mathbf{J}\nabla K = \mathrm{Jac}_f \mathbf{J} \mathrm{Jac}_f^T \nabla K \tag{1.8}$$

This implies that any canonical transformation is a symplectic map. This condition implies that Symplectic on a region $\Omega$ of phase-space is phase space volume preserving in that region. This is often refered to as the *Liouville Theorem* [8, 12].

*Proof.* Consider the symplectic map

$$\begin{array}{rclcl} \mathcal{M} & : & \Omega \subset \mathbb{R}^{2n} & \to & \Omega \\ & & \xi_i & \mapsto & \xi_f \end{array}$$

The final phase-space volume occupied by the beam is given by [13]

$$\int_\Omega d\xi_f = \int_\Omega |\mathrm{Jac}_\mathcal{M}|\, d\xi_i$$

The symplectic condition implies that $|\mathrm{Jac}_\mathcal{M}| = 1$, which concludes the proof. □

In particular, one can see that symplectic maps preserve the Hamilton equations as in (1.8), proving that any symplectic map is, in fact, a canonical transformation.

## 1.3 Hamiltonian Dynamics and Transfer Maps

The dynamics in particle accelerators are often modelled as Hamiltonian systems [14]. In this section, we build the Hamiltonian of a relativistic charged particle in an accelerator. We build the transfer map of the phase space coordinates throughout the accelerator beamline. The derivation from the Maxwell equations to the transfer map were studied in the Particle Accelerator Physics and Modelling I course [13] taught at ETH Zürich by Dr. Andreas Adelmann, which is where this project started from.

### 1.3.1 The Hamiltonian for a Charged Particle in an Electromagnetic field

Before we start, let us first recall the Maxwell equations

$$\text{Maxwell-Gauss law:} \quad \nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \tag{1.9}$$

$$\text{Maxwell-Faraday law:} \quad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{1.10}$$

$$\text{No mag. monopole:} \quad \nabla \cdot \mathbf{B} = 0 \tag{1.11}$$

$$\text{Maxwell-Ampère law:} \quad \nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} \tag{1.12}$$

From these, it is possible to define $\phi$ and $\mathbf{A}$, the scalar and vector potentials respectively, such that

$$\mathbf{E} = -\nabla\phi - \frac{\partial \mathbf{A}}{\partial t} \tag{1.13}$$

$$\mathbf{B} = \nabla \times \mathbf{A} \tag{1.14}$$

Indeed, $\nabla \times \mathbf{E} \neq 0$ implies that the electromagnetic potential cannot be a purely scalar potential. Using $\nabla \times \mathbf{B} = 0$, we conclude that there exists $\mathbf{A}$ such that (1.14) holds. From the Maxwell-Faraday law, we retrieve the expression (1.13).

We therefore obtain the expression for the Lorentz force

$$\mathbf{F} = q\left(-\nabla\phi - \frac{\partial \mathbf{A}}{\partial t} + \mathbf{v} \times \nabla \times \mathbf{A}\right) \tag{1.15}$$

from which one can derive the generalised potential $V = q\left(\phi - \mathbf{A}\cdot\mathbf{v}\right)$ [15]. Recalling that the kinetic energy $T$ of a relativistic particle reads $T = -m_0c^2\sqrt{1-\beta^2}$, we obtain the Lagrangian

$$\mathcal{L} = -m_0c^2\sqrt{1-\beta^2} - q\left(\phi - \mathbf{A}\cdot\mathbf{v}\right) \tag{1.16}$$

The generalised canonical momentum therefore is

$$\mathbf{P} = \nabla_{\dot{\mathbf{q}}}\mathcal{L} = \frac{m_0\mathbf{v}}{\sqrt{1-\beta^2}} + q\mathbf{A} = \gamma m_0\mathbf{v} + q\mathbf{A} = \mathbf{p} + q\mathbf{A} \tag{1.17}$$

where $\nabla_{\dot{\mathbf{q}}} \equiv (\partial_{\dot{x}}, \partial_{\dot{y}}, \partial_{\dot{z}})$. We then obtain the Hamiltonian via the Legendre transform

$$H(\mathbf{q},\mathbf{p}) = \dot{\mathbf{q}}\cdot\mathbf{P} - L(\mathbf{q}(t), \dot{\mathbf{q}}(t), t) = T + V \tag{1.18}$$

Inserting $\mathcal{L}$ and $\mathbf{P}$, one finds

$$c^2\left(\mathbf{P} - q\mathbf{A}\right)^2 - \left(H - q\phi\right)^2 = -m_0^2c^4$$

and finally obtains the generic Hamiltonian for relativistic particles in an EM field

$$H = \sqrt{c^2\left(\mathbf{P} - q\mathbf{A}\right)^2 + m^2c^4} + q\phi \tag{1.19}$$

We recall the Euler-Lagrange equation

$$\nabla_{\mathbf{q}}\mathcal{L} - \frac{d}{dt}\nabla_{\dot{\mathbf{q}}}\mathcal{L} = 0 \tag{1.20}$$

and finally, from equations (1.18) and (1.20), we recover the Hamilton equations

$$\nabla_{\mathbf{q}}H = -\frac{d\mathbf{P}}{dt} \tag{1.21}$$

$$\nabla_{\mathbf{P}}H = \frac{d\mathbf{q}}{dt} \tag{1.22}$$

which will act as our equations of motion.
Note however that in order to achieve this result, we have ignored collective effects and synchrotron radiation. We have also assumed that there was no beam collimation nor beam losses.

This formulation causes an issue to the accelerator modeller. Indeed, when designing an accelerator, the various components are positioned at given positions along the reference trajectory. As such, while we know at which longitudinal position a particle arrives at some lattice element, it is very hard to know at what time it reaches the said position. For this reason, we usually change the independent variable in time $t$ to path length $s$ [13].

Since the path length corresponds to the longitudinal coordinate, the Hamilton equations (1.21) and (1.22) are explicitly

$$\dot{x} = \frac{\partial x}{\partial t} = \frac{\partial H}{\partial P_x}, \quad \dot{P}_x = \frac{\partial P_x}{\partial t} = -\frac{\partial H}{\partial x} \tag{1.23}$$

$$\dot{y} = \frac{\partial y}{\partial t} = \frac{\partial H}{\partial P_y}, \quad \dot{P}_y = \frac{\partial P_y}{\partial t} = -\frac{\partial H}{\partial y} \tag{1.24}$$

$$\dot{s} = \frac{\partial s}{\partial t} = \frac{\partial H}{\partial P_s}, \quad \dot{P}_s = \frac{\partial P_s}{\partial t} = -\frac{\partial H}{\partial s} \tag{1.25}$$

Because the treatment of the horizontal and vertical axes in the transversal plane are identical in what follows, for the sake of simplicity, we drop the equations (1.24).
Inverting the first equation in (1.23), we get

$$x' = \frac{\partial x}{\partial s} = \frac{\dot{x}}{\dot{s}} = \frac{\partial H/\partial P_x}{\partial H/\partial P_s} \tag{1.26}$$

Since $H$ is constant (no explixit time dependency), we can write

$$\begin{aligned} 0 = dH &= \left.\frac{\partial H}{\partial P_x}\right|_{P_s} dP_x + \left.\frac{\partial H}{\partial P_s}\right|_{P_x} dP_s \\ \Rightarrow -\left.\frac{\partial P_s}{\partial P_x}\right|_{H} &= \frac{\partial H/\partial P_x}{\partial H/\partial P_s} \end{aligned}$$

which when combined with (1.26) gives

$$x' = -\frac{\partial P_s}{\partial P_x} \tag{1.27}$$

When following the same steps for $P_x$, we obtain

$$P_x' = \frac{\partial P_s}{\partial x} \tag{1.28}$$

Finally, inverting equations (1.25), we get

$$t' = \frac{\partial t}{\partial s} = \frac{\partial P_s}{\partial H}, \quad H' = -\frac{\partial P_s}{\partial t} \tag{1.29}$$

One therefore ses that $H_1 = -P_s$ is the Hamiltonian for the coordinate system $x, P_x, y, P_y, -t, H$. Rearranging equation (1.19) and recalling that the Hamiltonian is equal to the energy $E$, we therefore get

$$H_1 = -\sqrt{\frac{(E - q\phi)^2}{c^2} - m^2c^2 - (P_x - qA_x)^2 - (P_y - qA_y)^2} - qA_s \tag{1.30}$$

Note that by identifying the Hamiltonian $H$ to the total energy $E$, we have set $E$ as the canonical momentum conjugate to $-t$.

When working with infinite series expansions (like is the backbone of the model presented in this thesis), it is important to have small values, allowing lower order truncations and other approximation. As such, we introduce the *reference momentum* $P_0$ to normalise our values. While in general any choice of $P_0$ is valid, one usually choose it to be the nominal momentum of particles in the accelerator. We therefore do the substitutions

$$P_i \to \bar{P}_i = \frac{P_i}{P_0}$$

and simultaneously keep Hamilton's equations unchanged via the Hamiltonian substitution

$$H_1 \to \bar{H} = \frac{H_1}{P_0}$$

Finally, defining the normalised vector potential $\mathbf{a} = q\mathbf{A}/P_0$, we obtain the new Hamiltonian

$$\bar{H} = -\sqrt{\frac{(E-q\phi)^2}{P_0^2c^2} - \frac{m^2c^2}{P_0^2} - \left(\bar{P}_x - a_x\right)^2 - \left(\bar{P}_y - a_y\right)^2} - a_s \tag{1.31}$$

It now remains to scale down the longitudinal momentum $E/P_0$, which in general will be close to the speed of light. We therefore perform a canonical transformation using a generating function of the second kind:

$$F_2\left(x, \tilde{P}_x, y, \tilde{P}_y, -t, \delta; s\right) = x\tilde{P}_x + y\tilde{P}_y + \left(\frac{s}{\beta_0} - ct\right)\left(\frac{1}{\beta_0} + \delta\right)$$

with $\tilde{P}_x, \tilde{P}_y, \delta$ the new canonical momentum and $\beta_0 = v/c$ the normalised velocity of a particle with the reference momentum $P_0$. We then have

$$\bar{P}_i = \frac{\partial F_2}{\partial q_i}, \qquad Q_i = \frac{\partial F_2}{\partial \tilde{P}_i}, \qquad K = \tilde{H} + \frac{\partial F_2}{\partial s}$$

The transverse variables remain the same

$$\begin{aligned} \bar{P}_x &= \tilde{P}_x, \quad X = x \\ \bar{P}_y &= \tilde{P}_y, \quad Y = y \end{aligned}$$

and the longitudinal variables become

$$\delta = \frac{E}{P_0c} - \frac{1}{\beta_0}, \qquad Z = \frac{s}{\beta_0} - ct$$

Note that a relativistic particle with momentum $P_0$ will have $\delta = 0$. In general, $\delta$ is called the *energy deviation*. For the sake of simplicity, we will, from now on, update the notations as follows

$$K \to H, \qquad \tilde{P}_i \to p_i, \qquad Z \to z$$

Noticing that $mc/P_0 = 1/\gamma_0\beta_0$ where $\gamma_0 = 1/\sqrt{1-\beta^2}$, the new Hamiltonian reads

$$H = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta - \frac{q\phi}{P_0c}\right)^2 - (p_x - a_x)^2 - (p_y - a_y)^2 - \frac{1}{\left(\beta_0\gamma_0\right)^2}} - a_s \tag{1.32}$$

Untill now, we have assumed that the reference trajectory followed a linear path. This is of course not realistic, as apart from linear accelerators, most accelerators are in fact loops, hence requiring bends in the trajectory. Since we are following the path of a reference particle within these loops, it

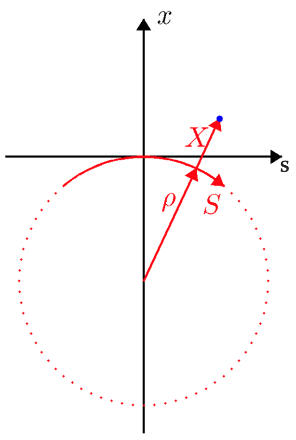


Figure 1.1: Coordinates of a particle (in blue) following a curved reference trajectory (in red). $\rho$ is the radius of curvature of the reference trajectory.

is more relevent to change from a cartesian coordinate system to the Frenet system, where we follow the bent trajectory of the reference particle [16]. Consider a particle in curved motion, as illustrated in figure 1.1.

The coordinates must be changed from $(x, y, s)$ to $(X, Y, S)$, with

$$x = (\rho + X)\cos\left(\frac{S}{\rho}\right) - \rho \tag{1.33}$$

$$y = Y \tag{1.34}$$

$$s = (\rho + X)\sin\left(\frac{S}{\rho}\right) \tag{1.35}$$

This change of coordinates can be obtained using a generating function of the third king

$$F_3(X, p_x, Y, p_y, S, \delta) = -\left[(\rho + X)\cos\left(\frac{S}{\rho}\right) - \rho\right] p_x - Y p_y - \left[(\rho + X)\sin\left(\frac{S}{\rho}\right)\right]\delta \tag{1.36}$$

Via $q_i = -\frac{\partial F_3}{\partial p_i}$, we recover the equations (1.33) to (1.35). The momenta are recovered with $P_i = -\frac{\partial F_3}{\partial Q_i}$, giving

$$P_X = p_x \cos\left(\frac{S}{\rho}\right) + p_s \sin\left(\frac{S}{\rho}\right)$$
$$P_Y = p_y$$
$$P_S = p_s\left(1 + \frac{X}{\rho}\right)\cos\left(\frac{S}{\rho}\right) - p_x\left(1 + \frac{X}{\rho}\right)\sin\left(\frac{S}{\rho}\right)$$

The vector potential also transforms to

$$A_X = A_x \cos\left(\frac{S}{\rho}\right) - A_s \sin\left(\frac{S}{\rho}\right)$$
$$A_Y = A_y$$
$$A_S = A_s \cos\left(\frac{S}{\rho}\right) + A_x \sin\left(\frac{S}{\rho}\right)$$

Thus the generic Hamiltonian becomes

$$H = \frac{\delta}{\beta_0} - (1+hx)\sqrt{\left(\frac{1}{\beta_0} + \delta - \frac{q\phi}{P_0 c}\right)^2 - (p_x - a_x)^2 - (p_y - a_y)^2 - \frac{1}{\beta_0^2\gamma_0^2}} + (1+hx)a_s \tag{1.37}$$

with $h = 1/\rho$ the "curvature" of the trajectory.

## 1.4 Hamiltonians of Common Beamline Elements

Let us change this generic Hamiltonian (1.37) to adapt it to some of the most common beamline elements, as they are presented in [13].

### 1.4.1 Drift

The drift is a magnetic field free region in the accelerator beamline. this implies in particular that $\nabla \times \mathbf{A} = 0$, and that $\mathbf{A}$ can be an arbitrarily gradient-free vector. For simplicity, we take $\mathbf{A} = 0$. What is more, the abscence of a magnetic field implies that the reference trajectory will be linear, giving $h = 0$. Therefore the drift Hamiltonian is

$$H_{\text{drift}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} \tag{1.38}$$

### 1.4.2 Multipole Fields

It is often hard to find solutions to the Maxwell equations, and is still ongoing work. As such, this study will be restricted to the case where the **B** field can be expressed as a multipole magnet via the *multipole magnet field* [17, 13].

$$B_y + iB_x = \sum_{m=1}^{\infty} (b_m + ia_m)\left(\frac{x+iy}{r_0}\right)^{m-1} \tag{1.39}$$

where $b_m$ and $a_m$ are arbitrary coefficents chosen to fit the problem, and $r_0$ is an arbitrary reference radius. we will restrict our study to *ideal multipoles*, i.e. multipoles such that in their multipole expansion, there exists a unique $m^*$ such that $(a_{m^*}, b_{m^*}) \neq (0,0)$. In particular, Maxwell's equation (1.11) implies that multipoles come in pairs, and therefore an ideal multipole with non-zero order $m^*$ is a $(2m^*)$-tupole. Lastly, a "normal multipole" is one such that for all $m, a_m = 0$, and a "skew multipole" is one such that for all $m, b_m = 0$. From this, we will build the magnetic fields of the studied multipole to follow.

### 1.4.3 Dipole

A dipole correponds to a magnetic field that is perpendicular to a plane. In the simpler cases this just gives a magnetic field $\mathbf{B} = (0, b_1, 0)$, as in figure 1.2. A well known results of such fields are

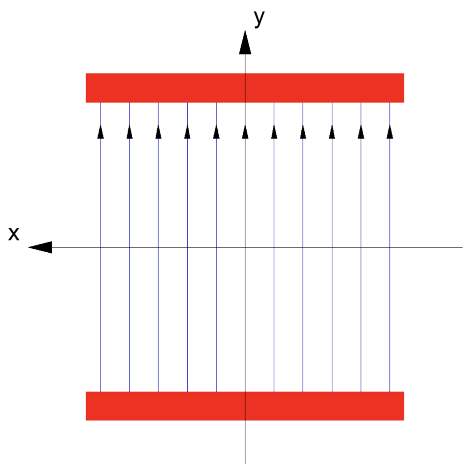


Figure 1.2: Normal dipole element, with $\mathbf{B} = (0, b_1, 0)$ (from [13]).

their bending properties on charged particle trajectories. As such dipoles are used in accelerator design to bend the trajectory and allow a turn to obtain a quasi-circular beamline. In curvilinear coordinates, the curl of the vector field $\mathbf{A}$ is given by

$$\begin{aligned}
[\nabla \times \mathbf{A}]_x &= \frac{\partial A_s}{\partial y} - \frac{1}{1+hx}\frac{\partial A_y}{\partial s} \\
[\nabla \times \mathbf{A}]_y &= \frac{1}{1+hx}\frac{\partial A_x}{\partial s} - \frac{h}{1+hx}A_s - \frac{\partial A_s}{\partial x} \\
[\nabla \times \mathbf{A}]_s &= \frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}
\end{aligned}$$

By identification with $\mathbf{B}$, we get

$$\mathbf{A} = \left(0, 0, -b_1\left(x - \frac{hx^2}{2(1+hx)}\right)\right) \tag{1.40}$$

and therefore the dipole Hamiltonian is

$$H_{\text{di}} = \frac{\delta}{\beta_0} - (1+hx)\sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} + (1+hx)k_0\left(x - \frac{hx^2}{2(1+hx)}\right) \tag{1.41}$$

with $k_0 = qb_1/P_0$ the dipole strength. In general, we align the reference trajectory on the dipole strength, so that the radius of curvature $\rho$ is that of a particle with reference momentum $P_0$. Hence usually $h = k_0$.

### 1.4.4 Quadrupole

A quadrupole is a collection of four magnets as in figure 1.3. It allows, as we will see to act as a focusing or defocusing element for particle beams, in a similar way to concave or convex lenses. The associated magnetic field reads $\mathbf{B} = (b_2 y/r_0, b_2 x/r_0, 0)$.

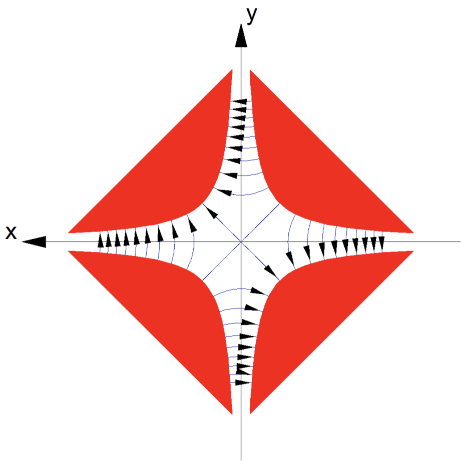


Figure 1.3: Normal quadrupole with $\mathbf{B} = \left(b_2 \frac{y}{r_0}, b_2 \frac{x}{r_0}, 0\right)$ (from [13]).

Observing that at the origin, there is no magnetic field, there is no curvature, and therefore taking the longitudinal axis as the reference trajectory, we can work in a straight coordinate system. Hence we get a simple potential

$$\mathbf{A} = \left(0, 0, -\frac{1}{2}\frac{b_2}{r_0}\left(x^2 - y^2\right)\right) \tag{1.42}$$

and therefore the quadrupole Hamiltonian is

$$H_{\text{quad}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} + k_2\left(x^2 - y^2\right) \tag{1.43}$$

with $k_2 = b_2 q / r_0 P_0$ the quadrupole strength. Observe that the $k_2\left(x^2 - y^2\right)$ term in the potential acts as focusing (resp. defocusing) potential in the $x$ (resp. $y$) direction if $k_2 > 0$ and vice-versa with $k_2 < 0$. If we look at the force

$$\mathbf{F} = q\mathbf{v} \times (\nabla \times \mathbf{A}) = \frac{P_0}{2m}\begin{pmatrix} -p_y k_2 \\ p_x k_2 \\ 0 \end{pmatrix} \tag{1.44}$$

we see that for $k_2 > 0, \mathbf{F}$ will be acting inwards in $x$ on outgoing particles and alog the $x$-momentum along $y$. This is why graphically, quadrupoles with $k_2 > 0$ are represented as converging lenses and those with $k_2 < 0$ as diverging lenses.

## 1.4.5 Sextupole

The sextupole consists of six magnets organised as in figure 1.4. As we will see, it induces a non-linearity in the system via third order terms in the Hamiltonian. This element induces a kick in momentum space, and will be studied here as a source of non-linearity in the system. The magnetic field it induces is

$$\mathbf{B} = \left(2b_3\frac{xy}{r_0^2}, b_3\frac{x^2 - y^2}{r_0^2}, 0\right) \tag{1.45}$$

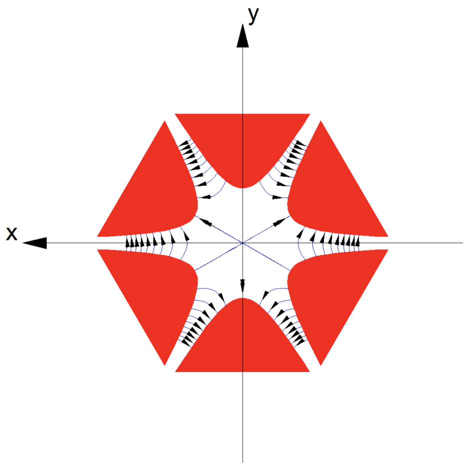


Figure 1.4: Normal sextupole with $\mathbf{B} = \left(2b_3\frac{xy}{r_0^2}, b_3\frac{x^2-y^2}{r_0^2}, 0\right)$ (from [13]).

As for the quadrupole we observe that $\mathbf{B}(0,0,s) = 0$. Hence we set the reference trajectory along $(x,y) = 0$ and can remain in a straight coordinate system. We then obtain the vector potential

$$\mathbf{A} = \left(0, 0, \frac{b_3}{3r_0^2}\left(3xy^2 - x^3\right)\right) \tag{1.46}$$

and finally the Hamiltonian

$$H_{\text{sext}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} + \frac{1}{6}k_2\left(x^3 - 3xy^2\right) \tag{1.47}$$

with $k_2 = \frac{q}{P_0}\frac{\partial^2 B_y}{\partial x^2} = \frac{2b_3 q}{P_0}$ the sextupole strength.

## 1.5 Building the Map $\mathcal{M}$

We are interested in evolution, with respect to the path length $s$, of some function $f(\xi, s)$ of $\xi = (q_1, p_1, \ldots, q_n, p_n)$, where $q_i$ and $p_i$ are the generalised coordinates and momenta, respectively. We follow the derivation in [9]. In particular, the derivative of $f$ reads

$$f' = \sum_i \frac{\partial f}{\partial q_i}q_i' + \frac{\partial f}{\partial p_i}p_i' \tag{1.48}$$

Using the Hamilton equations, (1.48) becomes

$$f' = \sum_i \frac{\partial f}{\partial q_i}\frac{\partial H}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial H}{\partial q_i}$$

using the Lie operator notation defined in section 1.1.1, we can rewrite (1.48) as

$$f' = - : H : f \tag{1.49}$$

We now apply Taylor's Theorem and observe that for any $s \in \mathbb{R}$,

$$\begin{aligned} f(s) &= \sum_{k=0}^{\infty} \frac{1}{k!} \left[ \left( \frac{d}{ds} \right)^k f \right]_{s=0} \\ &= \sum_{k=0}^{\infty} \frac{1}{k!} \left( -s : H : \right)^k \cdot f(s=0) \\ &= \exp\left( -s : H : \right) \cdot f(s=0) \end{aligned} \tag{1.50}$$

Hence we obtain the following expression for the transfer map $\mathcal{M}$ induced by $H$

$$\mathcal{M} = \exp\left( -s : H : \right) \tag{1.51}$$

Note that this makes $\mathcal{M}$ the Lie transformation associated with $-sH$.

We check that the transformation induced by $\mathcal{M}$ is indeed symplectic. First, define a map $f : \mathbb{R}^{2n} \to \mathbb{R}$ and let $\mathcal{F} := \exp(: f :)$. We will prove that $\mathcal{F}$ is symplectic, i.e. that any Lie transformation associated to a real map is symplectic [7]. Observe that for some $1 \leq i, j \leq 2n$, we have $\{\xi_i, \xi_j\} = \mathbf{S}_{i,j}$. Next we introduce the following lemmas, from [8, 18].

**Lemma 1.5.1.** *For any three maps $f, g, h \in \mathcal{C}^{\infty}\left(\mathbb{R}^{2n}, \mathbb{R}\right)$, we have*

$$: f : \{g, h\} = \{: f : g, h\} + \{g, : f : h\} \tag{1.52}$$

*Proof.* Observing that for any two maps $g, h$, $\{g, h\} = -\{h, g\}$, the lemma follows from the Jacobi identity. □

**Corollary 1.5.1.** *Through an induction on $k \in \mathbb{N}$, the above Lemma allows us to write*

$$: f :^k \{g, h\} = \sum_{m=0}^{k} \binom{k}{m} \left\{ : f :^m g, : f :^{k-m} h \right\} \tag{1.53}$$

**Lemma 1.5.2.** *For any three $\mathcal{C}^{\infty}$ maps $f, g, h : \mathbb{R}^{2n} \to \mathbb{R}$, we have*

$$\exp(: f :)\{g, h\} = \{\exp(: f :)g, \exp(: f :)h\} \tag{1.54}$$

*Proof.* From the Corollary 1.5.1, we have

$$\begin{aligned} \exp(: f :)\{g, h\} &= \sum_{k=0}^{\infty} \frac{: f :^k}{k!} \{g, h\} \\ &= \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{m=0}^{k} \binom{k}{m} \left\{ : f :^m g, : f :^{k-m} h \right\} \end{aligned}$$

From the bilinearity of the Lie bracket, this can be written

$$\begin{aligned}\exp(:f:)\{g,h\} &= \sum_{k=0}^{\infty}\sum_{m=0}^{k}\left\{\frac{:f:^m}{m!}g, \frac{:f:^{k-m}}{(k-m)!}h\right\} \\ &= \sum_{m=0}^{\infty}\left\{\frac{:f:^m}{m!}g, \sum_{k=0}^{\infty}\frac{:f:^k}{k!}h\right\} \\ &= \left\{\sum_{m=0}^{\infty}\frac{:f:^m}{m!}g, \sum_{k=0}^{\infty}\frac{:f:^k}{k!}h\right\}\end{aligned}$$

and we do recover $\exp(:f:)\{g,h\} = \{\exp(:f:)g, \exp(:f:)h\}$. □

This results allows us in particular to observe [7] that for any $i, j$,

$$\{\mathcal{F}\xi_i, \mathcal{F}\xi_j\} = \mathcal{F}\{\xi_i, \xi_j\} = \mathcal{F}\mathbf{S}_{i,j} = \mathbf{S}_{i,j} \tag{1.55}$$

On another side, observe that for any two functions $g, h$, we have, from [7]

$$\{g,h\} = \sum_{k,l}\frac{\partial g}{\partial \xi_k}\mathbf{S}_{k,l}\frac{\partial h}{\partial \xi_l} \tag{1.56}$$

Combining equations (1.55) and (1.56), we have in particular

$$\mathbf{S}_{i,j} = \{\mathcal{F}\xi_i, \mathcal{F}\xi_j\} = \sum_{k,l}\frac{\partial \mathcal{F}\xi_i}{\partial \xi_k}\mathbf{S}_{k,l}\frac{\partial \mathcal{F}\xi_j}{\partial \xi_l} = \left(\mathrm{Jac}_{\mathcal{F}}\mathbf{S}\mathrm{Jac}_{\mathcal{F}}^T\right)_{i,j} \tag{1.57}$$

which is the symplectic condition, proving that Lie transformations generated by real maps are symplectic maps.

# Chapter 2

# Implementation

## 2.1 Implementation Idea

In order to efficiently implement the map in equation (1.51), we need a model where functions $f$ are represented as objects $F$ in finite dimension instead of by their values [19]. In other words, we must build the transformation $T$ such that the diagrams in figure 2.1 commute.

$$\begin{tikzcd}
f, g \arrow[r, "T"] \arrow[d, "{\pm, \times, /}"'] & F, G \arrow[d, "{\oplus, \ominus, \otimes, \oslash}"] \\
f \pm, \times, / g \arrow[r, "T"] & F \oplus, \ominus, \otimes, \oslash G
\end{tikzcd}
\qquad
\begin{tikzcd}
f \arrow[r, "T"] \arrow[d, "\partial"'] & F \arrow[d, "\partial_\circ"] \\
\partial f \arrow[r, "T"] & \partial_\circ F
\end{tikzcd}$$

Figure 2.1: Commuting diagrams of the requirements on the implementable representation of functions (from [19]).

Hence we need a finite dimensional Differential Algebra that allows to accurately approximate functions $f$.

### 2.1.1 The ${}_nD_v$ Differential Algebra

We explore the Differential Algebraic structure ${}_nD_v$, fully presented in [19], which corresponds to our needs with respect to the diagrams in figure 2.1.

**Definition 2.1.1.** For some $n \in \mathbb{N}$, we define the equivalence relation $=_n$ such that for any $f, g \in \mathcal{C}^n(\mathbb{R}^v)$, $f =_n g$ if their Taylor Series expansions agree up to order $n$. The equivalence classes are denoted $[f]_n$.

**Definition 2.1.2.** For any $n, v \in \mathbb{N}$, we define

$${}_nD_v = \{[f]_n | f \in \mathcal{C}^n(\mathbb{R}^v)\}$$

For any $f, g \in \mathcal{C}^n(\mathbb{R}^v), t \in \mathbb{R}$ and $1 \leq k \leq v$, we define the following operations

$$[f]_n + [g]_n = [f + g]_n \tag{2.1}$$

$$t \times [f]_n = [t \times f]_n \tag{2.2}$$

$$[f]_n \times [g]_n = [f \times g]_n \tag{2.3}$$

$$\partial_k [f]_n = \left[ x_k \frac{\partial f}{\partial x_k} \right]_n \tag{2.4}$$

**Proposition 2.1.1.** *For any integers $n, v$, ${}_nD_v$ equipped with operations* (2.1) *to* (2.4) *is a Differential Algebra.*

Let $n, v$ be two integers and let us now have a look at the basis and dimension of ${}_nD_v$. Let us denote $d_k \overset{\text{def}}{=} [x_k]_n$ for some integer $1 \leq k \leq v$. Observe in particular that for any $f \in \mathcal{C}^n(\mathbb{R}^v)$,

$$\begin{aligned}[f]_n = [T_n(f)]_n &= \left[ \sum_{j_1 + \dots + j_v \leq n} c_{j_1, \dots, j_v} \cdot x_1^{j_1} \dots x_v^{j_v} \right]_n \\ &= \sum_{j_1 + \dots + j_v \leq n} c_{j_1, \dots, j_v} \cdot d_1^{j_1} \dots d_v^{j_v} \end{aligned} \tag{2.5}$$

where $T_n(f)$ is the truncated power series expansion (TPSE) of $f$ to order $n$ and with

$$c_{j_1, \dots, j_v} = \frac{1}{j_1! \dots j_v!} \cdot \left. \frac{\partial^{j_1 + \dots + j_v} f}{\partial x_1^{j_1} \dots \partial x_v^{j_v}} \right|_{\mathbf{x}=\mathbf{0}} \tag{2.6}$$

where $\cdot|_{\mathbf{x}=\mathbf{0}}$ denotes evaluation of the derivative at 0. We observe from (2.5) that the terms $d_1^{j_1} \dots d_v^{j_v}$ generate ${}_nD_v$. Furthermore, they are linearly independent, and therefore form a basis for ${}_nD_v$. Each of these can be represented as a unique sequence of length $n + v$

$$(\underbrace{1, 1, \dots, 1}_{j_1 \text{ times}}, 0, \underbrace{1, 1, \dots, 1}_{j_2 \text{ times}}, 0, \dots, 0, \underbrace{1, 1, \dots, 1}_{j_v \text{ times}}, 0, \underbrace{1, 1, \dots, 1}_{n - j_1 - \dots - j_v \text{ times}})$$

There are $\binom{n+v}{v}$ ways of building such sequences. We therefore conclude that ${}_nD_v$ has dimension $\frac{(n+v)!}{n!v!}$. In particular, we observe from table 2.1 that the dimension of ${}_nD_v$ blows up with $n$ and $v$, requiring high computing power, which is the main difficulty when using this structure.

Finally, we build the vectorial representation of a function $f$'s Truncated Power Series at order $n$. First, we order the basis elements $d_1^{j_1} \dots d_v^{j_v}$ using the following method. From an element $[f]_n$ of ${}_nD_v$ written as in (2.5), we look for the terms with lowest sum $j_1 + \dots + j_v$. Next, take the terms with highest exponent of $d_1$, from these, the ones with highest exponents of $d_2$ and so on. Eventually, we are left with a single term, which we place next in our ordered basis ${}_n\mathcal{B}_v$. We carry on until all terms have been ordered. This ordering method results in the basis [19]

$${}_n\mathcal{B}_v = \left( \mathbf{d}^{[0]}, \mathbf{d}^{[1]}, \dots, \mathbf{d}^{[n]} \right)$$

where $\mathbf{d} = (d_1, \dots, d_v)^T$ and $\mathbf{d}^{[i]}$ denotes the $i$-th Kronecker power defined recursively by

$$\mathbf{d}^{[i]} = \left( d_1^i \cdot (d_2, \dots, d_v)^{[0]}, d_1^{i-1} \cdot (d_2, \dots, d_v)^{[1]}, \dots, d_1^0 \cdot (d_2, \dots, d_v)^{[i]} \right)^T$$

$$(d_{v-1}, d_v)^{[k]} = \left( d_{v-1}^k d_v^0, d_{v-1}^{k-1} d_v^1, \dots, d_{v-1}^0 d_v^k \right)^T$$

| $v\backslash n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 | 66 |
| 3 | 4 | 10 | 20 | 35 | 56 | 84 | 120 | 165 | 220 | 286 |
| 4 | 5 | 15 | 35 | 70 | 126 | 210 | 330 | 495 | 715 | 1,001 |
| 5 | 6 | 21 | 56 | 126 | 252 | 462 | 792 | 1,287 | 2,002 | 3,003 |
| 6 | 7 | 28 | 84 | 210 | 462 | 924 | 1,716 | 3,003 | 5,005 | 8,008 |
| 7 | 8 | 36 | 120 | 330 | 792 | 1,716 | 3,432 | 6,435 | 11,440 | 19,448 |
| 8 | 9 | 45 | 165 | 495 | 1,287 | 3,003 | 6,435 | 12,870 | 24,310 | 43,758 |
| 9 | 10 | 55 | 220 | 715 | 2,002 | 5,005 | 11,440 | 24,310 | 48,620 | 92,378 |
| 10 | 11 | 66 | 286 | 1,001 | 3,003 | 8,008 | 19,448 | 43,758 | 92,378 | 184,756 |

Table 2.1: Dimension of ${}_nD_v$ with respect to $n, v$ (from [19]).

We now finally have the vector representation of a function $f \in \mathcal{C}^n(\mathbb{R}^v)$ with ${}_nD_v$ as

$$[f]_n = (c_{j_1,\dots,j_v})_{j_1+\dots+j_v\leq n} \tag{2.7}$$

with $j_i$'s the exponents of the multiplied basis ${}_n\mathcal{B}_v$ monomial element.

Let us look at a concrete example in dimensions $n = v = 3$. From the previous method of ordering, we obtain table 2.2 for the coefficients $j_k$.

This table is built recursively as follows. Iterating over values $m = \sum j_k$, we obtain coefficients $j_2, \dots, j_v$ by building the table for $n \to m$ and $v \to v-1$. We then compute the coefficient $j_1$ for each row through $j_1 = \sum j_k - \sum_{k\geq 2} j_k$. The base case, once $v \to 1$, is simply $j_v = [0, \dots, m]$.

Finally, one obtains the ordered coefficients $(c_{j_1,\dots,j_v})_{j_1+\dots+j_v\leq n}$ by iterating over the rows of the table obtained with the previous method and computing equation (2.6).

### 2.1.2 Order of the Truncated Power Series Expansion

In practice, the tools provided by the ${}_nD_v$ differential algebra are used to implement the Lie transfer map $\mathcal{M} = \exp(-s : H :)$ as follows. First, we define the Hamiltonian truncation order $n_1$ and truncate the Hamiltonian:

$$H = H|_{\leq n_1} + \mathcal{O}\left(|\xi|^{n_1+1}\right) \tag{2.8}$$

where $H|_{\leq n_1}$ is the power series representation of $H$, truncated at order $n_1$. From now on, we drop the higher order terms $\mathcal{O}\left(|\xi|^{n_1+1}\right)$, and only look at $H|_{\leq n_1}$.

The exponential operator is then itself also truncated at order $n_2$, and the Lie transfer map $\mathcal{M} = \exp(-s : H :)$ is implemented, in practice, as

$$\mathcal{M}(n_1, n_2) = \sum_{k=0}^{n2} \frac{(-s)^k}{k!} : H|_{\leq n_1} :^k \tag{2.9}$$

Observe that since $H|_{\leq n_1}$ is of finite order and so is the exponential expansion. Hence the polynomial maps obtained via $\mathcal{M}(n_1, n_2)\xi$ are of finite degree. In particular, we have the following lemma

| $\sum j_k$ | $j_1$ | $j_2$ | $j_3$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
|  | 0 | 1 | 0 |
|  | 0 | 0 | 1 |
| 2 | 2 | 0 | 0 |
|  | 1 | 1 | 0 |
|  | 1 | 0 | 1 |
|  | 0 | 2 | 0 |
|  | 0 | 1 | 1 |
|  | 0 | 0 | 2 |
| 3 | 3 | 0 | 0 |
|  | 2 | 1 | 0 |
|  | 2 | 0 | 1 |
|  | 1 | 2 | 0 |
|  | 1 | 1 | 1 |
|  | 1 | 0 | 2 |
|  | 0 | 3 | 0 |
|  | 0 | 2 | 1 |
|  | 0 | 1 | 2 |
|  | 0 | 0 | 3 |

Table 2.2: Coefficients $j_k$ used for the TPSE, as in (2.5), for $n = v = 3$.

**Lemma 2.1.1.** *Let $f, g : \mathbb{R}^{2n} \to \mathbb{R}$ be polynomials of finite non-zero degree. Then if the degree* $\deg(f)$*, i.e. the highest order of the non-zero term, is equal to* 1*, the polynomial $\sum_{k=0}^{m} \frac{:f:^k}{k!} g$ has degree at most* $\deg g$*. Else if* $\deg(f) \geq 2$*, $\sum_{k=0}^{m} \frac{:f:^k}{k!} g$ has degree at most* $m \deg(f) + \deg(g) - 2m$*.*

*Proof.*

- **Case $\mathbf{deg(f)} = \mathbf{1}$**
  We first prove that for any $k \in \mathbb{N}, \deg\left(: f :^k g\right) \leq \deg(g) - k$. We proceed by incudtion on $k$. First, $: f :^0 g = g \Rightarrow \deg\left(: f :^0 g\right) = \deg(g) - 0$. We also prove the case $k = 1$: We have $: f : g = \sum_i \frac{\partial f}{\partial q_i} \frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i} \frac{\partial g}{\partial q_i}$. Since $f$ is linear by assumption, its derivatives become constant factors. Now, recalling that derivatives lower the degree of a polynomial by at least 1, we conclude that for any $i, \deg\left(\frac{\partial f}{\partial q_i} \frac{\partial g}{\partial p_i}\right) \leq \deg\left(\frac{\partial g}{\partial p_i}\right) \leq \deg(g) - 1$, and similarly for terms $\frac{\partial f}{\partial p_i} \frac{\partial g}{\partial q_i}$. Since the degree of the sum of two polynomials is equal to the highest of either polynomials' degrees, we conclude that $\deg(: f : g) = \deg(g) - 1$. Hence we assume that the claim holds for some integer $k$. Then we have $: f :^{k+1} g =: f :^k (: f : g)$. From our claim, it has degree at most $\deg(: f : g) - k \leq \deg(g) - (k + 1)$, concluding the proof by induction. Summing all the terms in the polynomial $\sum_{k=0}^{m} \frac{:f:^k}{k!} g$, we conclude that it has degree at most $\max_{0 \leq k \leq m}(\deg(g) - k) = \deg(g)$.

- **Case $\mathbf{deg(f)} > \mathbf{1}$**
  We proceed as before and first prove by induction that $: f :^k g$ has degree at most $k \deg(f) + \deg(g) - k$. The initialisation is the same as the previous case. For $k = 1$, it is slightly more

complicated. For some $i$, we look at $\frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i}$. As stated before, we have $\deg\left(\frac{\partial g}{\partial p_i}\right) \leq \deg(g)-1$. However, we now also have $\deg\left(\frac{\partial f}{\partial q_i}\right) \leq \deg(f)-1$. Recalling that the degree of the product of two polynomials is equal to the sum of each polynomial's degrees, we get $\deg\left(\frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i}\right) \leq (\deg(f)-1)+(\deg(g)-1) = 1\deg(f)+\deg(g)-1$, proving the claim for $k=1$. Assuming now that it holds for some integer $k$, we have $: f^{k+1} g =: f :^k (: f : g)$, which, by assumption, has degree at most $k\deg(f)+\deg(: f : g)-k = (k+1)\deg(f)+\deg(g)-(k+1)$, proving our claim. Therefore $\sum_{k=0}^{m}\frac{:f:^k}{k!}g$ has degree at most $\max_{0\leq k\leq m}(k\deg(f)+\deg(g)-k) = m\deg(f)+\deg(g)-m$.

□

We conclude that the polynomials $\mathcal{M}(n_1,n_2)\xi$ have degree at most $n_2(n_1-1)+1$.

## 2.2 Ensuring Symplecticity of Maps

A problem that the Truncated Power Series approach causes is the loss of symplecticity of the Lie Transform induced by $-sH$. Indeed, the result obtained in equation (1.57) holds only at infinite order. While truncating the exponential at high enough order might make the non-symplecticity negligible for a single element, this demands expensive computations, with the higher order error adding up in accelerators which contain thousands of beamline elements, causing harmful divergences of the phase-space volume. We therefore need a "symplectification algorithm" that will, from a finite order polynomial transfer map, give a symplectic map. Note that the truncation order of the TPSE of $H$ is not a relevent issue with respect to the symplecticity of the transfer maps. Indeed, recall from section 1.5 that any map in the form $\exp(: H :)$ is sympletic. It however has an influence on the precision of the implementation, as we ignore the higher order terms of the Hamiltonian.

### 2.2.1 Dragt-Finn Factorisation

**Theorem 2.2.1.** ***Dragt-Finn Factorisation theorem*** *[7] Let $f : \mathbb{R}^{2n} \to \mathbb{R}$ and consider the symplectic Lie transformation $\mathcal{F} = \exp(: f :)$ associated to $f$. Then there exists a set gen($\mathcal{F}$) such that for every $k \in \mathbb{N}$, there exists a unique homogeneous polynomial $g_k \in$ gen($\mathcal{F}$) of degree $k$ and such that*

$$\mathcal{F} = \prod_{k=2}^{\infty} \exp(: g_k :) \tag{2.10}$$

The proof of this theorem is presented in [7]. We omitted here some extra steps which are not needed for our specific case. We first introduce some lemmas and their proofs, from [7, 8].

**Lemma 2.2.1.** *Let $h : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ a set of functions. For any $1 \leq i,j \leq 2n$, $h$ satisfies $\{\xi_i, h_j\} = \{\xi_j, h_i\}$ if, and only if there exists a function $g : \mathbb{R}^{2n} \to \mathbb{R}$ which satisfies $h_i = \{g, \xi_i\} =: g : \xi_i$.*

*Proof.* First, let $g : \mathbb{R}^{2n} \to \mathbb{R}$ such that, for any $1 \leq i,j \leq 2n$, $h_i =: g : \xi_i$. Then the Jacobi Identity gives

$$\{\xi_i, h_j\} - \{\xi_j, h_i\} = \{\xi_i, \{g, \xi_j\}\} - \{\xi_j, \{g, \xi_i\}\} = -\{g, \{\xi_i, \xi_j\}\} = -\{g, \mathbf{S}_{i,j}\} = 0$$

Conversely, let $h : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ such that for every $1 \leq i, j \leq 2n$, $\{\xi_i, h_j\} = \{\xi_j, h_i\}$. Let $\xi^* := \mathbf{S}\xi$, which can also be written $\xi = \mathbf{S}^T\xi^*$. Consider now a function $\varphi : \mathbb{R}^{2n} \to \mathbb{R}$. Then

$$\{\xi_i, \varphi\} = \sum_j \left(\frac{\partial \xi_i}{\partial \xi_j}\right) \mathbf{S}_{j,k} \left(\frac{\partial \varphi}{\partial \xi_k}\right) = \sum_k \mathbf{S}_{i,k} \frac{\partial \varphi}{\partial \xi_k} = \sum_k \frac{\partial \varphi}{\partial \xi_k} \frac{\partial \xi_k}{\partial \xi_i^*} = \frac{\partial \varphi}{\partial \xi_i^*} \tag{2.11}$$

because for each $i$, there exists a unique $k$ with $\frac{\partial \xi_k}{\partial \xi_i^*} \neq 0$. In particular, we can write the initial assumption on $h$ as $\frac{\partial h_i}{\partial \xi_j^*} = \frac{\partial h_j}{\partial \xi_i^*}$. Therefore $h_i \cdot d\xi^*$ is an exact differential. In particular, we can define the path integral

$$g := -\int_0^{\xi^*} h \cdot d\xi'^* = -\int_0^{\xi} h^T \mathbf{S} d\xi' \tag{2.12}$$

Then we do have (from (2.11)) $\{g, \xi_i\} = -\frac{\partial g}{\partial \xi_i^*} = h_i$ □

**Lemma 2.2.2.** *Any symplectic matrix* $\mathbf{N} \in Sp_{\mathbf{S}}(2n)$ *can be written as*

$$\mathbf{N} = \exp(\mathbf{SM}) \tag{2.13}$$

*with* $\mathbf{M} \in \mathbb{R}^{2n \times 2n}$ *a symmetric matrix.*

*Proof.* Consider a parameter $\tau \in \mathbb{R}$ and the parametrised symplectic matrices $\mathbf{N}(\tau)$ such that

- $\forall \tau_1, \tau_2 \in \mathbb{R}$,
$$\mathbf{N}(\tau_1 + \tau_2) = \mathbf{N}(\tau_1)\mathbf{N}(\tau_2) \tag{2.14}$$
- $\mathbf{N}(0) = \mathbf{I}_{2n}$
- $\mathbf{N}(1) = \mathbf{N}$

Differentiating the equation (2.14), we get

$$\left.\frac{\partial \mathbf{N}}{\partial \tau_1}\right|_{\substack{\tau_1 = 0 \\ \tau_2 = \tau}} = \mathbf{N}'(0)\mathbf{N}(\tau) = \mathbf{N}(\tau)$$

This solves $\mathbf{N}(\tau) = \exp(\tau \mathbf{N}'(0))$. We introduce an arbitrary matrix $\mathbf{M} \in \mathbb{R}^{2n \times 2n}$ such that $\mathbf{N}'(0) = \mathbf{SM}$. In the limit $\tau \to 0$, we in particular find $\mathbf{N}(\tau) = \exp(\tau\mathbf{SM}) \approx \mathbf{I}_{2n} + \tau\mathbf{SM}$. The symplectic condition therefore becomes $(\mathbf{I}_{2n} + \tau\mathbf{SM})\,\mathbf{S}\,(\mathbf{I}_{2n} + \tau\mathbf{SM})^T = \mathbf{S}$. Since this hold for any $\tau$ in small enough neighbourhood of 0, this gives $\mathbf{SM}^T\mathbf{S}^T + \mathbf{SMS} = 0 \Leftrightarrow \mathbf{M} = \mathbf{M}^T$. Taking $\tau = 1$, we find the wanted result. □

Hence if we look at some symplectic symplectic transfer map $\mathcal{F}$, we may write $\text{Jac}_{\mathcal{F}}(0) = \exp(\mathbf{SM})$ with $\mathbf{M} \in \mathbb{R}^{2n \times 2n}$ a symmetric matrix. Let us define

$$g_2 := -\sum_{i,k} \mathbf{M}_{i,k} \xi_i \xi_k = -\sum_{i,k} \mathbf{M}_{i,k} \left(\sum_j \mathbf{S}_{j,i} \xi_j^*\right) \xi_k \tag{2.15}$$

In particular, $: g_2 : \xi_i = -\frac{\partial g}{\partial \xi_i^*} = \sum_j (\mathbf{SM})_{i,j} \xi_j$. Exponentiating, we finally get

$$\exp(: g_2 :)\xi = \text{Jac}_{\mathcal{M}}(0)\xi \tag{2.16}$$

**Lemma 2.2.3.** *Let us write* $\bar{\xi} = \mathcal{F}\xi$. *Then we have* $\exp\left(- : g_2 :\right) \bar{\xi} = \xi + \mathcal{O}\left(|\xi|^2\right)$

*Proof.* Let $\sigma \in \mathbb{N}^{2n}$ denote a collection of exponents and let $\alpha(\sigma)$ denote the associated coefficient such that we have

$$\bar{\xi}_i = \sum_{|\sigma|>0} \alpha_i(\sigma)\xi^\sigma$$

where $\xi^\sigma = \prod_j \xi_j^{\sigma_j}$. Then we can write

$$\begin{aligned}\exp\left(- : g_2 :\right) \bar{\xi}_i &= \exp\left(- : g_2 :\right) \sum_{|\sigma|=1} \alpha_i(\sigma)\xi^\sigma + \mathcal{O}\left(|\xi|^2\right)\\ &= \exp\left(- : g_2 :\right) \sum_j \mathrm{Jac}_{\mathcal{M}}(0)_{i,j}\xi_j + \mathcal{O}\left(|\xi|^2\right)\end{aligned}$$

But from (2.16), this becomes

$$\begin{aligned}\exp\left(- : g_2 :\right) \bar{\xi} &= \mathrm{Jac}_{\mathcal{M}}(0)^{-1}\mathrm{Jac}_{\mathcal{M}}(0)\xi + \mathcal{O}\left(|\xi|^2\right)\\ &= \xi + \mathcal{O}\left(|\xi|^2\right)\end{aligned}$$

which concludes the proof. □

**Lemma 2.2.4.** *For all $k > 2$, there exists homogeneous polynomials $g_k$ of degree $k$ such that when successively applied on $\exp\left(- : g_2 :\right) \bar{\xi}$, the order of the remiander can be made arbitrarily big. In other words,*

$$\forall k > 2, \quad \left(\prod_{l=2}^{k} \exp\left(- : g_{k-l+2} :\right)\right) \bar{\xi} = \xi + \mathcal{O}\left(|\xi|^k\right) \tag{2.17}$$

Before prooving this lemma, we need an intermediary result.

**Lemma 2.2.5.** *Let $k, l > 1$ be some integers, and let $g, h : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ be sets of polynomials of minimal non-zero order $k$ and $l$ respectively. Then the polynomial $\{g, h\}$ is a polynomial with lowest non-zero order $kl - k - l + 1$.*
*If either $k$ or $l$ is equal to 1, say arbitrarily $l = 1$, then $\{g, h\}$ has minimal non-zero order $k - 1$. If $k = l = 1$, then $\{g, h\}$ is a constant, with no non-zero order term.*

*Proof.* For a polynomial $P$, let us denote by $\mu(P)$ its lowest non-zero order. Let $g, h$ be polynomials of lowest orders $\mu(g) = k > 1$ and $\mu(h) = l > 1$. We have

$$\mu\left(\{g, h\}\right) = \mu\left(\sum_i \frac{\partial g}{\partial q_i}\frac{\partial h}{\partial p_i} - \frac{\partial g}{\partial p_i}\frac{\partial h}{\partial q_i}\right)$$

Since derivation lowers the order of each term of a polynomial by 1 and multiplication adds them, we find

$$\mu\left(\sum_i \frac{\partial g}{\partial q_i}\frac{\partial h}{\partial p_i} - \frac{\partial g}{\partial p_i}\frac{\partial h}{\partial q_i}\right) \geq (k-1)(l-1) = kl - k - l + 1$$

If now $k > 1$ and $l = 1$, for any $i$, $\frac{\partial h}{\partial q_i}$ is a constant and we get

$$\mu\left(\sum_i \frac{\partial g}{\partial q_i}\frac{\partial h}{\partial p_i} - \frac{\partial g}{\partial p_i}\frac{\partial h}{\partial q_i}\right) \geq k - 1$$

Similarily, if $k = l = 1$, both derivated polynomials are constants with degree 0. □

**Remark.** *Note that this lemma implies that if we exponentiate, for any polynomials $g, h$ of respective non-zero minimal order $k, l$, we find the corresponding result for* $\exp(: g :)h$*:*

$$\begin{aligned}\exp(: g :)h &= \sum_{m=0}^{\infty} \frac{: g :^m}{m!} h \\ &= h + \sum_{m=1}^{\infty} \frac{: g :^m}{m!} h\end{aligned}$$

*Then one can prove by induction on $m \geq 1$ that the minimal non-zero order $\mu(: g :^m h)$ is equal to $(k-1)^m(l-1)$. Of course, if $k > 1$ and $l = 1$, this becomes $(k-1)^m$. In the particular case where $k = 1$, all terms $m > l$ vanish and we are left with a shift.*

We now proove lemma 2.2.4

*Proof.* We first prove this result for order 3. Let $h : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ a second order function such that $\exp(- : g_2)\bar{\xi} = \xi + h(\xi) + \mathcal{O}\left(|\xi|^3\right)$. Then we have in particular

$$\begin{aligned}\mathbf{S}_{i,j} &= \left\{\xi_i + h_i(\xi) + \mathcal{O}\left(|\xi|^3\right), \xi_j + h_j(\xi) + \mathcal{O}\left(|\xi|^3\right)\right\} \\ &= \mathbf{S}_{i,j} + \{\xi_i, h_j\} + \{h_i, \xi_j\} + \mathcal{O}\left(|\xi|^2\right)\end{aligned}$$

where all higher order terms were eaten up in $\mathcal{O}\left(|\xi|^2\right)$ thanks to lemma 2.2.5. Thus $\{\xi_i, h_j\} + \{h_i, \xi_j\} = 0$. This implies, from lemma 2.2.1, that there exists a map $g_3 : \mathbb{R}^{2n} \to \mathbb{R}$ which satisfies $h(\xi) =: g_3 : \xi$. Its explicit form is given by (2.12), which as an integral of an order 2 homogeneous polynomial is an order 3 homogeneous polynomial. Therefore $\exp\left(- : g_2 :\right)\bar{\xi} = \xi + : g_3 : \xi + \mathcal{O}\left(|\xi|^3\right)$, and finally $\exp\left(- : g_3 :\right)\exp\left(- : g_2 :\right)\bar{\xi} = \xi + \mathcal{O}\left(|\xi|^3\right)$. By proceding iteratively, we can increase the order of the remainder to infinitely big order. □

We are now equipped with all necessary tools to prove theorem 2.2.1.

*Proof.* [7] The idea of the proof of this theorem is to build, iteratively, the generators $g_k$. First, we obtain the 2nd order generator from lemma 2.2.2, by identifying the symplectic matrix $\mathrm{Jac}_{\mathcal{F}}(0)$ to the exponential $\exp\left(\mathbf{SM}\right)$, with $\mathbf{M} \in \mathbb{R}^{2n\times 2n}$ a symmetric matrix, and recovering $g_2$ such that $: g_2 := \mathbf{SM}$. Lemma 2.2.3 tells us that applying only $\exp(: g_2 :)$, we obtain a remainder of the order $\mathcal{O}\left(|\xi|^2\right)$ with respect to the exact map $\mathcal{F}$. Lemma 2.2.4 uses results from the proof of lemma 2.2.1 to build a homogeneous polynomial $g_3$ of degree 3 such that $\exp(: g_2 :)\exp(: g_3 :)$ has an error of $\mathcal{O}\left(|\xi|^3\right)$. Repeating that procedure, we can iteratively find generators such that $\prod_{l=2}^{k} \exp(: g_l :)$ has an error of $\mathcal{O}\left(|\xi|^k\right)$, which in a neighbourhood of zero, is increasingly small. In particular, repeating this procedure in the limit $k \to \infty$, we can write

$$\mathcal{F} = \prod_{k=0}^{\infty} \exp(: g_k :) \tag{2.18}$$

which finalises the proof of theorem 2.2.1. □

The principal interest of this decomposition is that each generator $g_k$ generates terms of order $\geq k-1$ (from lemma 2.2.5 and its corollary). Hence the terms induced by $\exp(: g_k :)$ correspond to the optical properties induced by the order $k$ terms of the Hamiltonian.

**Remark.** *Observe that $\mathcal{F}$ might contain constant, order 0 terms, corresponding to a shift, or constant "kick" in phase-space, independent of the coordinates $\xi$. These define the effect of linear terms in the Hamiltonian, such as the term $\delta/\beta_0$ in the generic Hamiltonian* (1.37). *If this is the case, it suffices to find the generators of $\mathcal{F}$ of order at least 2 and add that shift once done. (2.18) becomes $\mathcal{F} = \mathcal{F}|_0 + \prod_{g_k \in gen(\mathcal{F})} \exp(: g_k :)$.*

This theorem is central to our work. Indeed, it allows to ensure symplecticity of truncated symplectic polynomial maps. If we take the proof for lemma 2.2.4, we can build an iterative algorithm to find the homogeneous polynomials $g_k$.

1. Set the shift $\bar{\xi}_0 = \mathcal{F}|_0$
2. Define $\tilde{\mathcal{F}}_1 = \bar{\xi} - \bar{\xi}_0$.
3. Set $\exp(: g_2 :) := \mathrm{Jac}_{\tilde{\mathcal{F}}_1}(0)$
4. Define $\tilde{\mathcal{F}}_2 := \exp(- : g_2 :)\tilde{\mathcal{F}} = \mathrm{Jac}_{\tilde{\mathcal{F}}}(0)^{-1}\tilde{\mathcal{F}}$.
5. Iterating over $k > 2$, do
   - Take $h_k$, the $k$-th order terms $\tilde{f}_k$ of $\tilde{\mathcal{F}}_{k-1}$, and let $g_k = -\int_0^\xi \tilde{f}_k^T \mathbf{S} d\xi'$
   - Define $\tilde{\mathcal{F}}_k := \exp(- : g_k :)\tilde{\mathcal{F}}_{k-1}$
6. Don't forget to add the shift $\bar{\xi}_0$ to $\prod_{g_k \in \mathrm{gen}\mathcal{F}} \exp(: g_k :)$ once you have reached infinity and beyond!

What this means in practice is that if we decide to truncate the infinite degree polynomial $\mathcal{F}$ at order $m$, we have an algorithm to restore symplecticity. Indeed, a truncation at order $m$ means that we are ignoring the reminder $\mathcal{O}\left(|\xi|^{m+1}\right)$. In other words,

$$\mathcal{F} = \prod_{k=0}^{m} f_k + \mathcal{O}\left(|\xi|^{m+1}\right) = \mathcal{F}|_0 + \prod_{k=2}^{m} \exp(: g_k :) + \mathcal{O}\left(|\xi|^{m+1}\right) \tag{2.19}$$

which in a small enough neighbourhood of 0 gives

$$\mathcal{F} \approx \mathcal{F}|_0 + \prod_{k=2}^{m} \exp(: g_k :) \tag{2.20}$$

Since the Lie transformations $\exp(: g_k :)$ are symplectic, we have found an approximation of $\mathcal{F}$ that restores its symplecticity!

**Example 2.2.1.** We show an example of computation of Dragt-Finn factorisation generators for a low dimensional example, taken and adapted from [6]. Consider the 2D map

$$\begin{array}{cccc} \mathcal{M} & : & \mathbb{R}^2 & \to & \mathbb{R}^2 \\ & & (x,p) & \mapsto & \begin{pmatrix} 1 - x - 3p + x^2 + 2xp + p^2 + x^3 + x^2p + 2xp^2 + p^3 \\ 2 + x + 2p + x^2 - 2xp - p^2 + x^3 + 5x^2p + 3xp^2 - \frac{2}{3}p^3 \end{pmatrix} \end{array}$$

We first identify $\mathcal{M}|_0 = (1, 2)$ and correct the shift to get $\tilde{\mathcal{M}}_1 = \mathcal{M} - \mathcal{M}|_0$. Next, compute $\exp(: g_2 :) \equiv \mathrm{Jac}_{\tilde{\mathcal{M}}_1}(0)$ and find that

$$\exp(: g_2 :) \begin{pmatrix} x \\ p \end{pmatrix} = \begin{pmatrix} -x - 3p \\ x + 2p \end{pmatrix} \tag{2.21}$$

Correcting the participation of exp(: $g_2$ :), we obtain

$$\begin{array}{cccc} \tilde{\mathcal{M}}_2 & : & \mathbb{R}^2 & \rightarrow & \mathbb{R}^2 \\ & & (x,p) & \mapsto & \begin{pmatrix} x + x^2 + 2xp + p^2 + x^3 + x^2p + 2xp^2 + p^3 \\ p + x^2 - 2xp - p^2 + x^3 + 5x^2p + 3xp^2 - \frac{2}{3}p^3 \end{pmatrix} \end{array}$$

We now look for the 3rd order generator $g_3 = h_{3,0}x^3 + h_{2,1}x^2p + h_{1,2}xp^2 + h_{0,3}p^3$. In practice, we don't integrate and identify the 2nd order terms of $\tilde{\mathcal{M}}_2$, as in the recipe above, but do the equivalent differentiation of $g_3$ and identify the terms with $\tilde{\mathcal{M}}_2$. At first order in the expansion of exp(: $g_3$ :), we have

$$\begin{aligned} x - \frac{\partial g_3}{\partial p} &= x - h_{2,1}x^2 - 2h_{1,2}xp - 3h_{0,3}p^2 = x + x^2 + 2xp + p^2 \\ p + \frac{\partial g_3}{\partial x} &= p + 3h_{3,0}x^2 + 2h_{2,1}xp + h_{1,2}p^2 = p + x^2 - 2xp - p^2 \end{aligned}$$

and finally find that

$$g_3 = \frac{1}{3}x^3 - x^2p - xp^2 - \frac{1}{3}p^3 \tag{2.22}$$

Next, we correct the higher order terms of exp(: $g_3$ :) to get $\tilde{\mathcal{M}}_3 = \exp(- : g_3 :)\tilde{\mathcal{M}}_2$, given explicitly as

$$\begin{array}{cccc} \tilde{\mathcal{M}}_3 & : & \mathbb{R}^2 & \rightarrow & \mathbb{R}^2 \\ & & (x,p) & \mapsto & \begin{pmatrix} x - x^3 - x^2p + 2xp^2 + p^3 \\ p + x^3 + 3x^2p + xp^2 - \frac{2}{3}p^3 \end{pmatrix} \end{array}$$

For $g_4$, we do exactly as for $g_3$ but at the 3rd order of $\tilde{\mathcal{M}}_3$, and find

$$g_4 = \frac{1}{4}x^4 + x^3p + \frac{1}{2}x^2p^2 - \frac{2}{3}xp^3 - \frac{1}{4}p^4 \tag{2.23}$$

Observe that the map $\mathcal{M}|_0 + \exp(: g_2 :)\exp(: g_3 :)\exp(: g_4 :)$ contains all terms of $\mathcal{M}$, as well as higher order terms ensuring symplecticity. It is therefore not necessary to continue looking for generators $g_{k\geq 5}$.

It is sometimes easier to factorise the map $\mathcal{F}$ in decreasing order of generators, i.e.

$$\mathcal{F} = \mathcal{F}|_0 + \prod_{k=0}^{m} \exp(: g^*_{m-k} :) \tag{2.24}$$

This can be achieved via the Lie exchange formula (see section 1.1.2).

### 2.2.2 Symplecticity of the Drift

A particular result for drift regions is that even after a truncation at an order $n_2 \geq 1$ of the expansion of the exponential $\mathcal{M} = \exp(-L : H :)$, with $L$ the length of the drift, the induced map will be fully simplectic. Recall the Drift region Hamiltonian

$$H_{\text{drift}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} \tag{2.25}$$

Note that it does not have a physical space dependence. This implies in particular that the higher order terms in the exponential expansion of the Lie transform it generates are all 0. Indeed, we have

$$-:H:\xi = \begin{pmatrix} \frac{\partial H}{\partial p_x} \\ 0 \\ \frac{\partial H}{\partial p_y} \\ 0 \\ \frac{\partial H}{\partial \delta} \\ 0 \end{pmatrix}, \qquad :H:^2 \xi = -\sum_i \frac{\partial H}{\partial p_i}\frac{\partial}{\partial q_i} \begin{pmatrix} \frac{\partial H}{\partial p_x} \\ 0 \\ \frac{\partial H}{\partial p_y} \\ 0 \\ \frac{\partial H}{\partial \delta} \\ 0 \end{pmatrix} = \mathbf{0} \tag{2.26}$$

Hence the Lie transform associated to a drift of length $L$ is explicitly given by

$$\mathcal{M}\xi = \xi + L \begin{pmatrix} \frac{\partial H}{\partial p_x} \\ 0 \\ \frac{\partial H}{\partial p_y} \\ 0 \\ \frac{\partial H}{\partial \delta} \\ 0 \end{pmatrix} \tag{2.27}$$

This means in particular that if we were to truncate the series representation of the exponential $\mathcal{M} = \exp -L:H:$ at any order greater than 1, all terms would appear and the map would be fully simplectic.

## 2.3 First Version of the Package

### 2.3.1 Context

The first implementation of the model was done in `Julia 1.8`. The choice of this language was motivated by the claims on `Julia`'s performance being comparable to that of `C++`, while offering the ease of use of `Python` [20]. Furthermore, `Julia` offers the possibility to parallelise and use GPUs [21], which is desirable when dealing with big vector spaces such as ${}_nD_v$. While `PyTorch` and `TensorFlow` have similar promises and tools, we chose to use `Julia` mainly for the very complete (with respect to our needs) and simple-to-use Computer Algebra System module `Symbolics` [22], which appeared as more adapted than `SymPy`. In particular, we relied on the very simple variable definition and symbolic differentiation capabilities of `Symbolics`.

### 2.3.2 Algorithm Structure

The algorithmic workflow of our code, named `JuliAccel`, is illustrated in figure 2.2. The algorithm is fed with the symbolic expression for the Hamiltonian $H$ and the orders $n_1$ and $n_2$ for the respective truncated power series representations of $H$ and $\mathcal{M} = \exp(-\tau:H:) = \sum_{k=0}^{n2} \frac{:H|_{\leq n_1}:^k}{k!}$ in ${}_{n_{1,2}}D_v$.

Let us have a look at the operator structure mentioned in figure 2.2. When applying $\mathcal{M}$ on a bunch, we want to avoid computing $\exp(-s:H:)\xi$ from scratch every time. Instead we want an operator that is precompiled for $H$, i.e. that acts as a function only taking $\xi$ as argument. To this end, we expand the `Operator` structure provided in the `Julia Symbolics` package. In the source code, `Operators` are an abstract subtype of `Functions` on which special operations are defined. It is in particular defined for the differential operators $\frac{\partial}{\partial x}$, with composition and exponentiating operations. For our code, we however need a little more. We defined more specific operators, which we may

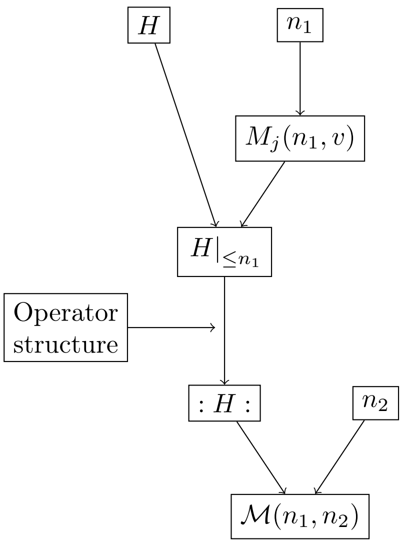


Figure 2.2: Algorithmic workflow of the code. $M_j(n_1, v) = \left(j^l_k\right)^{1\le l\le \dim({}_{n_1}D_v)}_{1\le k\le v}$ contains the coefficients (as in table 2.2) used to obtain $H|_{\le n_1}$ as in (2.5); $T_{n_1}$ is the TPS expansion of $H$ at order $n_1$; $\mathcal{P}(H) =: H :$ is the Lie operator; $\mathcal{M} = \sum_{k=0}^{n2} \frac{:H|_{\le n_1}:^k}{k!}$ is the transfer map induced by $H|_{\le n_1}$ up to order $n_2$.

classify in two categories: "basic" operators and "coupled" operators. Basic operators are defined to act on Symbolic expressions in a particular way. In particular,

- **Identity operator:** simple Identity operator, with a possibility to multiply by a factor as optional argument.

```
struct Id <: Operator end
(O::Id)(x::Num)::Num = x
(O::Id)(v::Vector{Num})::Vector{Num} = v
```

Coupled operators define operations on two operators. In particular, we have

- **Operations operator:** Defines the operator obtained after addition / subtraction / division of two operators

```
struct Add{T1<:Operator, T2<:Operator} <: Operator
    O1::T1
    O2::T2
end
(O::Add)(x::Num)::Num = O.O1(x) + O.O2(x)
(O::Add)(v::Vector{Num})::Vector{Num} = O.O1(v) .+ O.O2(v)
```

- **Inner Product operator:** Defines operator multiplication, which is a composition of operators.

```
struct InnerProd{T1<:Operator, T2<:Operator} <: Operator
    O1::T1
    O2::T2
end
(O::InnerProd)(x::Num)::Num = O.O1(O.O2(x))
(O::InnerProd)(v::Vector{Num})::Vector{Num} = O.O1(O.O2(v))
```

- **External Multiplication operator:** Defines external multiplication of an operator, allowing to define operators such as $\frac{\partial f}{\partial q} \cdot \frac{\partial}{\partial p}$.

```
struct ExternalMul{T1<:Real, T2<:Operator} <: Operator
    r::T1
    O::T2
end
(O::ExternalMul)(x::Num)::Num = O.r * O.O(x)
(O::ExternalMul)(v::Vector{Num})::Vector{Num} = O.r .* O.O(v)
```

This first version provided excellent precision results. As expected, we do observe that the maps generated with the raw model were not symplectic (i.e. $\text{Jac}_{\mathcal{M}} \neq 1$), and need us to do a Dragt-Finn factorisation of the generated polynomials. We used the very useful tree representation of symbolic expressions of `Symbolics` to do so. It works by building a tree representation of symbolic operations. When difning a symbolic expression, the package separates the expression into pairs of operands and their linking operation. A tree is then built where the nodes correspond to the concerned operation and the bracnches link to the operands, which are either other symbolic operations, in which case we start again, or our variable symbols, in which case they are leaves. As a simple example, consider the operation

$$\left(\sqrt{3x} - (x-y)^3\right) \times (x+y)$$

Its tree representation is illustrated in figure 2.3 Through TPSE representation (and possible addi-

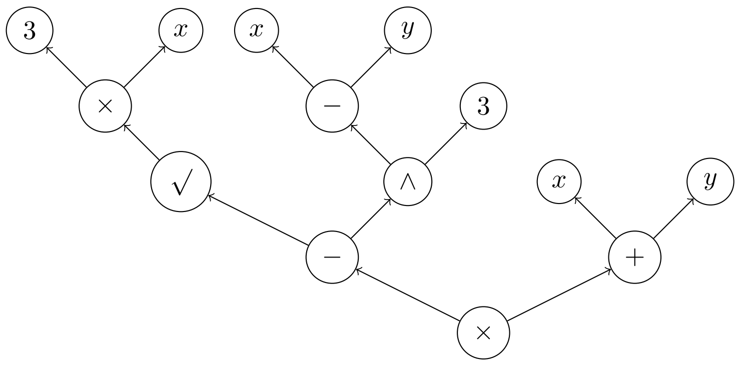


Figure 2.3: Tree representartion of $\left(\sqrt{3x} - (x-y)^3\right) \times (x+y)$

tional expanding/simplifying) the expression, we are left with a set of two leaved trees representing each monomial of the TPS expansion. This allows in particular to easily compute the map $g_k = -\int_0^\xi h_k^T \mathbf{S} d\xi'$ as defined in equation (2.12) to obtain the order $k$ generator in the Dragt-Finn factorisation procedure. The algorithm to compute $g_k$ from a flattened tree representation of $\tilde{\mathcal{F}}_{k-1}$ is as follows [6]

1. Generate a homogeneous polynomial $L_k = \sum_{|\sigma|=k} \lambda_k(\sigma)\xi^\sigma$, with $\xi^\sigma = \prod_i \xi_i^{\sigma_i}$, of arbitrary coefficients $\lambda_k(\sigma)$ to be determined.
2. Generate the polynomial maps : $L_k : \xi$.
3. iterating over the monomials $\lambda_k(\sigma) : \xi^\sigma : \xi$, identify the coefficients $\lambda_k(\sigma)$ with the lead coefficient form the node of $\tilde{\mathcal{F}}_{k-1}$ corresponding to the monomial : $\xi^\sigma : \xi$.

4. Once the $\lambda_k(\sigma)$s have been identified, correct for higher order terms: $\tilde{\mathcal{F}}_k = \exp\left(- : L_k :\right)\tilde{\mathcal{F}}_{k-1}$.

One issue with that method is that it requires that the tree representation of $\tilde{\mathcal{F}}_{k-1}$ be flat. This requires that we expand, amplify and flatten the polynomial at every iteration, which for higher orders of truncation becomes quickly tideous. In particular, a big problem arises from the differentiation of expressions. Indeed, the derivation is implemented as an operator, with operands the variable with respect to which the differentiation is done, and the expression to be differentiated. Recalling that tree traversal has time complexity $\mathcal{O}(N)$ with $N$ the number of nodes, this means that the numerical evaluaton of a symbolic expressioin becomes exponentially large. While this could be dealt with and optimised with a thorough study of the trees generated thoroughout the code, we unforttunattely did not have the oportunity to efficiently correct for this complexity problem. Do note however that this initial verison of the code gave very promissing results in terms of precsion, making it a promising imnplementation approach.

### 2.3.3 Quantum Simulation

With this version of the code, we implemented the possibility to use complex coordinates to model quantum behaviour. As such, we tried to reproduce the results presented in [23]. We first try to solve the time dependent Schrödinger equation (TDSE)

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = \mathcal{H}\left|\psi(t)\right\rangle \tag{2.28}$$

Since the computer cannot store infinite data, we choose to restrict our study to a finite number $M$ of energy levels:

$$\mathcal{H} = \sum_{j=0}^{M-1}\left|\phi_j\right\rangle\left\langle\phi_j\right| E_j \tag{2.29}$$

A known problem with the Schrödinger equation (2.28) is that it cannot be solved using standard integrators (e.g. Runge-Kutta), as such integrators are based on truncations of the truncated power series expansions of the function, which is not norm conserving. For quantum mechanical problems, this is not possible. Instead use the time translation operator

$$U(t) = e^{-it\mathcal{H}} \tag{2.30}$$

This matrix being too big to store, we focus on computing the matrix element [23]

$$\left\langle\psi\left|e^{-it\mathcal{H}}\right|\psi'\right\rangle = \sum_{j=0}^{M-1}\left\langle\psi|\phi_j\right\rangle\left\langle\phi_j|\psi'\right\rangle e^{-itE_j} \tag{2.31}$$

with $\{E_j\}_j$ the Eigenvalues of $\mathcal{H}$ and $\{|\phi_j\rangle\}_j$ the associated Eigenvectors. However, it might be hard to diagonalise the whole matrix $\mathcal{H}$. Hence we use the Lie-Trotter formula to decompose the Hamiltonian into easy to diagonalise terms.

**Theorem 2.3.1.** ***Lie-Trotter formula*** *[23] Let $A, B$ be two non-commuting operators and let $x \in \mathbb{R}$. Then we have*

$$e^{(A+B)x} = e^{Ax}e^{Bx} - \frac{1}{2}x^2\left[A, B\right] + \mathcal{O}\left(x^3\right) \tag{2.32}$$

In particular, the Lie-Trotter formula can be reformulated for Quantum modelling as the Lie-Trotter-Suzuki (LTS) decomposition.

**Theorem 2.3.2.** ***Lie-Trotter-Suzuki decomposition*** *[24] Let $m \in \mathbb{N}$ such that there exists $\{\mathcal{H}_j\}_{1\leq j\leq m}$ with $\mathcal{H} = \sum_{j=1}^{m} \mathcal{H}_j$, and such that $\max_j \|H_j\| = h \leq \infty$. The translation operator reads, for any $k \in \mathbb{R}$*

$$U(t) = \exp\left(-i\mathcal{H}t\right) = \left(\prod_{i=1}^{m} \exp\left(-i\frac{\mathcal{H}_j t}{k}\right)\right)^k + \mathcal{O}\left(\frac{m^2h^2t^2}{k}\right) \tag{2.33}$$

*Proof.* Let us consider a single time slice $t/k$

$$\begin{aligned} U\left(\frac{t}{k}\right) &= \exp\left(-i\frac{t}{k}\sum_{j=1}^{m}\mathcal{H}_j\right) \\ &= e^{-i\frac{t}{k}\mathcal{H}_1}\cdot \exp\left(-i\frac{t}{k}\sum_{j=2}^{m}\mathcal{H}_j\right) + \mathcal{O}\left(\frac{t^2}{k^2}\left\|\left[H_1, \sum_{j=2}^{m}\mathcal{H}_j\right]\right\|\right) \end{aligned}$$

But we have using the tiangle inequality

$$\begin{aligned} \left\|\left[\mathcal{H}_1, \sum_{j=2}^{m}\mathcal{H}_j\right]\right\| &\leq \max_j \|[\mathcal{H}_1, \mathcal{H}_j]\|\,(m-1) = \max_j \|\mathcal{H}_1\mathcal{H}_J - \mathcal{H}_J\mathcal{H}_1\| \\ &\leq 2\max_j \|\mathcal{H}_1\mathcal{H}_j\|\,(m-1) \\ &\leq 2h^2\,(m-1) \end{aligned}$$

Then, by induction on $j$,

$$\begin{aligned} U\left(\frac{t}{k}\right) &= e^{-i\frac{t}{k}\mathcal{H}_1}\cdot e^{-i\frac{t}{k}\mathcal{H}_2}\cdot \exp\left(-i\frac{t}{k}\sum_{j=2}^{m}\mathcal{H}_j\right) + \mathcal{O}\left(\frac{t^2}{k^2}\left((m-1)h^2 + (m-2)h^2\right)\right) \\ &= \dots \\ &= \prod_{j=1}^{m} e^{-i\frac{t}{k}\mathcal{H}_j} + \mathcal{O}\left(\frac{t^2h^2}{k^2}\left(1+\cdots+(m-1)\right)\right) \end{aligned}$$

Concatenating $k$ times, we finally get

$$U(t) = \left(\prod_{j=1}^{m} e^{-i\frac{t}{k}\mathcal{H}_j}\right)^k + \mathcal{O}\left(\frac{t^2h^2m^2}{k}\right) \tag{2.34}$$

and conclude the proof. □

First, we looked at simple leaky harmonicoscillator. A leaky harmonic oscillator is a simple harmonic oscillator with non diagonal terms allowing a change in energy levels. The Hamiltonian with finite energy levels reads

$$\mathcal{H}_{\text{leaky}} = \sum_{n=0}^{\infty}\left[\hbar\omega\left(n+\frac{1}{2}\right)|n\rangle\langle n| + a|n\rangle\langle n+1| + b|n+1\rangle\langle n|\right] \tag{2.35}$$

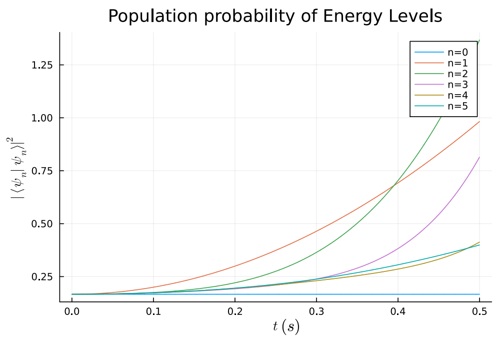
where the coefficients $a$ and $b$ allow us to ponderate the non-diagonal terms. For the sake of simplicity and to limit the divergence perspectives, we decided to perform our study on 6 energy levels, and setting $b = 0$. Hence the Hamiltonian simplifies to

$$\mathcal{H}_{\text{leaky}} = \sum_{n=0}^{5} \left[ \hbar\omega \left( n + \frac{1}{2} \right) |n\rangle\langle n| + a|n\rangle\langle n+1| \right] \tag{2.36}$$

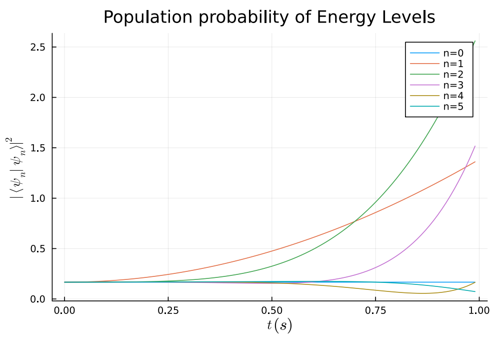
It is possible to do a LTS decomposition of the Hamiltonian as

$$\mathcal{H}_{\text{leaky}} = \mathcal{H}_{\text{diag}} + \sum_{n=0}^{5} \mathcal{H}_{\text{nd}}^{(n)} \tag{2.37}$$

Where $\mathcal{H}_{\text{diag}}$ corresponds to the diagonal part of the Hamiltonian, and $\mathcal{H}_{\text{nd}}^{(n)} = a|n\rangle\langle n+1|$ to the $n$-th non-diagonal ladder operator. Running a simiulation to study the time evolution of the energy levels population, we obtained the results in figure 2.4.



(a) $a = 4, b = 0$



(b) $a = 4, b = 0$, Trotterised

Figure 2.4: Simulation of the time evolution for a leaky harmonic oscillator, with truncation order $n_2 = 5$.

These plots obviously show a big problem. Indeed, if we loook at the population probabilities of the energy levels, we obtain probabilities greater than one. Increasing the truncation orders for the exponential did not correct for these divergences. This reveals an underlying issue with the implementation of quantum modelling that we decided should be kept for after the code was fully developped. This choice was also motivated by the lack complex symbols support from `Symbolics`, requiring that we implement it ourselves, which would have been simpler once the core of the code was finalised.

Another quantum physics implementation that we tried was the double slit experiment [23]. We study charged-particle interferometry by studying charged, spin-less non-relativistic particles in an external, static magnetic field $\mathbf{B} = \nabla \times \mathbf{A}$. This gives a Hamiltonian

$$\mathcal{H} = \frac{1}{2m^*} (\mathbf{p} - e\mathbf{A})^2 + V$$

with $m^*$ the effective mass of a particle with charge $e$, $\mathbf{p} = -i\hbar\nabla$ the momentum operator and $\mathbf{A}$ and $V$ the vector and scalar potentials (respectively). We restrict our study to the $x$-$y$ plane, imposing $\mathbf{B} = (0, 0, B(x, y))$, and $V = V(x, y)$. We hence choose $\mathbf{A} = (A_x(x, y), 0, 0)$, with

$$A_x(x, y) = -\int_0^y B(x, y)dy$$

We also set the wave function outside the simulation box to 0.

We normalise lengths by $\lambda$, the unit of wave-vectors is $k = \frac{2\pi}{\lambda}$, energies in $E = \frac{\hbar^2 k^2}{2m^*}$, time in $\frac{\hbar}{E}$ and vector potential in $\frac{e\lambda}{\hbar}$. The Hamiltonian becomes

$$\mathcal{H} = -\frac{1}{4\pi^2}\left(\left[\frac{\partial}{\partial x} - iA_x(x, y)\right]^2 + \frac{\partial^2}{\partial y^2}\right) + V(x, y)$$

For simplicity, we first assume no magnetic field and set $\mathbf{A} = 0$ and $V = 0$

We look for solutions $|\Psi(\mathbf{x}, t)\rangle$ of the TDSE

$$i\hbar\frac{\partial}{\partial t}|\Psi(\mathbf{x}, t)\rangle = \mathcal{H}|\Psi(\mathbf{x}, t)\rangle$$

These solutions are given by

$$|\Psi(\mathbf{x}, t)\rangle = \exp\left(-it\frac{\mathcal{H}}{\hbar}\right)|\psi(\mathbf{x})\rangle$$

with $|\psi(\mathbf{x})\rangle := |\Psi(\mathbf{x}, t = 0)\rangle$.

In particular, for the interference experiment, we take

$$|\psi(\mathbf{x})\rangle = \frac{1}{\sqrt{2}}\left(|\psi_1(\mathbf{x})\rangle + |\psi_2(\mathbf{x})\rangle\right)$$

Where $\psi_i(\mathbf{x}) = e^{i(|\mathbf{x}-\mathbf{r}_i|)}$ is the plane wave sourced at slit $i = 1, 2$ at position $\mathbf{r}_i$.

A big issue that arose is that applying $e^{-it\frac{\mathcal{H}}{\hbar}}$ to $|\psi_i(\vec{x})\rangle$ is unreasonably inefficient: hours to low order truncation (e.g. $n_2 = 4$) with low resolution. We concluded that we should push the upgrade to a complex implementation to once the $\mathbb{R}$ version was optimised.

## 2.4 Second Version of the Package

### 2.4.1 Context

Faced to the efficiency issue observed with the initial verison of the code, we decided to change the `Symbolics.jl` dependency to `TaylorSeries.jl` [25]. It is a dependency that, when given a symbolic expresison, will compute its TPS expansion up to a chosen order and represent it as a multinomial of that order using the `MultivariatePolynomials.jl` package, which is optimised for manipulation of finite degree multivariate polynomials.

With this package, the code gets heavily smplified, in the first place because all the procedure to build a TPSE from an expression is done by the `TaylorSeries.jl` package. After having defined, for instance

$$H = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta - \frac{q\phi}{P_0 c}\right)^2 - (p_x - a_x)^2 - (p_y - a_y)^2 - \frac{1}{(\beta_0\gamma_0)^2}} - a_z \qquad (2.38)$$

the package will automatically represent it as an ordered list of lead coefficients $c_{j_1,\dots,j_6}$ such that $H|_{\leq n_1} = \sum_{j_1+\dots+j_6\leq n_1} c_{j_1,\dots,j_6}\xi_1^{j_1}\dots\xi_6^{j_6}$, as in equation (2.5).

Furthermore, the differentiation is done via automatic differentiation, taking as arguments a TPSE and a variable with respect to which it should differentiate. This differes from the previous verison since with the `Symbolics` package, the `differentiate` function was in fact an instance of a subtype `Operator` of functions. Hence all the Operator structure described in section 2.3 is no longer needed, and the Lie operators and Lie transformation are no longer compositions of operators. They are functions taking two arguments $f$ and $g$, and compute $: f : g$ and $\exp(: f :)g$ respectively. This differs from the previous version of the code. Before, we would compute the differentials of the generator of the transformation and generating the operators, and then apply them to the phase-space coordinates. In this newer version, we evaluate the the application of the Lie transforms at the same time as we compute the differentials of the generator of the Lie transformation.

### 2.4.2 Implementation

However, this new dependency adds another complexity which we will turn to our advantage, as explained in the results section 3.4. Indeed, as stated above, the truncation order for all `TaylorSeries` generated TPSEs should be stated as a global constant of the package, before it can generate TPSEs. One would of course be tented to define $n_1$ as that global truncation order, hence defining $H$ would return $H|_{\leq n_1}$. However, this causes a severe issue. As stated in lemma 2.1.1, the degree of $\mathcal{M}(n_1, n_2)\xi$ can reach $n_2(n_1 - 1) + 1 > n_1$ for any $n_2 > 1$. This means that if we set the `TaylorSeries` truncation order to $n_1$, all higher order terms will be neglected. Since we are already truncating both $H$ and $\exp(-s : H :)$, this aggravates the problem and should be avoided. We therefore introduce a third setting $n_{\max} > n_1$, which is the overall truncation order. Of course, one should in theory define $n_{\max} = n_2(n_1 - 1) + 1$, but this causes some issues. Indeed, as we will see in section 3.4, it is not necessary to define such a high truncaton order $n_{\max}$ for good precison, avoiding some computation time (see table 2.1 and the exponentially large dimension of the ${}_nD_v$ algebra.) Hence the map $\mathcal{M}$ is now represented as

$$\mathcal{M}(n_1, n_2, n_{\max}) = \left.\left(\sum_{k=0}^{n_2} \frac{: H|_{\leq n_1} :^k}{k!}\right)\right|_{\leq n_{\max}} \tag{2.39}$$

where we reused the notation $\cdot|_{\leq n_{\max}}$, meaning truncation at order $n_{\max}$.

Let us look at the details of how the `TaylorSeries.jl` dependency represents TPSEs. Note that we are interested here in multivariate series, but the dependency provides a code optimised for single variable TPSEs. For multivariate polynomials, `TaylorSeries.jl` generates an exponent table like in table 2.2. Form that the `TaylorSeries.jl` package introduces the `HomogeneousPolynomial` structure, representing homogeneous polynomials of an order $k$. It is represented as an ordered list of coefficients associated to each of the order $k$ monomials. These ceofficients are the $c_{j_1,\dots,j_{2n}}$ leading the monomials generated from the part $\sum j_l = k$ of table 2.2. In particular, it is possible to set the type of coefficients, via `HomogeneousPolynomial{T}` where `T` is the said type. The final TPSE is then implemented as the `TaylorN` structure, taking a list of `HomogeneousPolynomial` for each order of the expansion. We have

$$\begin{aligned}\texttt{TaylorN\{T\}} &\equiv \texttt{Vector\{HomogeneousPolynomial\{T\}\}}\\ &\equiv \texttt{Vector\{Tuple\{Vector\{T\}, Integer\}\}}\end{aligned}$$

$$\begin{tikzcd}
 & \boxed{H} \arrow[dd] & & \boxed{n_{\max}} \arrow[ll] \\
\boxed{n_1} \arrow[dr] & & & \\
 & \boxed{H|_{\leq n_1}} \arrow[dd] & \boxed{n_2} \arrow[ddl] & \\
 & & & \boxed{\xi} \arrow[dll] \\
 & \boxed{\mathcal{M}(n_1, n_2, n_{\max})\xi} & &
\end{tikzcd}$$

Figure 2.5: Algorithmic workflow of the code. We implement $H$ in `TaylorSeries.jl` at order $n_{\max}$. The Hamiltonian is then truncated to order $n_1$. We then compute, up to order $n_2$ the application : $f :^k \xi$.

This structure build is very usefull to implement a Dragt-Finn factorisation algorithm. From the corrected map up to order $k-1, \tilde{\mathcal{F}}_{k-1}$, We proceed as follows

1. Define $L_k = 0$.
2. Iterate over the $2n$ `TaylorNs` of the transfer maps $\mathcal{M}_i$.
3. Let $l_i^{(k)}$ be the $k$-th order `HomogeneousPolynomial` of $\mathcal{M}_i$. We integrate $l_i^{(k)}$ with respect to $(\mathbf{S}\xi)_i$ and add it to $L_k$.

**Remark.** *Doing this causes an error in the coefficients for $L_k$. Let us use a toy model with one degree of freedom, i.e. $\xi = (q, p)$. Consider the order $k = 3$ factorisation. Let*

$$\tilde{\mathcal{F}}_2\xi = \begin{pmatrix} q - q^2 - 6p^2 + 2qp + \mathcal{O}\left(|\xi|^3\right) \\ p + 6q^2 - p^2 + 2qp + \mathcal{O}\left(|\xi|^3\right) \end{pmatrix}$$

*The map $L_3$ that generates this map at order 2 is $L_3 = 2q^3 + q^2p - qp^2 + 2p^3$. However, the method described before gives $\tilde{L_3} = 2q^3 + 2q^2p - 2qp^2 + 2p^3$. This is because the mixed terms that have both variables will be integrated as many times as there are different variables in them. A solution is to correct by dividing the leading coefficients of monomials by the number of variables it contains.*

This adds a next step to the Dragt-finn factorisation algorithm:

4. Iterating over the monomials of the `HomogeneousPolynomial`, let $\sigma$ be the exponent vector. We divide the leading coefficient by the number of non-zero exponents $\sigma_i \neq 0$.
5. Correct for higher order terms and let $\tilde{\mathcal{F}}_k = \exp(- : L_k :)\tilde{\mathcal{F}}_{k-1}$.

This concludes the Dragt-Finn factorisation algorithm in the second version of the code. There however is a problem due to the overall truncation order $n_{\max}$. Indeed, if we were to follow the Dragt-Finn factorisationrecipe, this means that we would compute the generators of $\mathcal{M}(n_1, n_2, n_{\max})$ up to order $n_{\max}$. Note that the recipe says we should go to order $n_{\max} + 1$, but $g_{n_{\max}+1}$ cannot be constructed as it is a homogeneous polynomial of degree $n_{\max} + 1$. This causes however a big issue, which is that the higher order generators will have most of their terms truncated at $n_{\max}$. This implies a loss of symplecticity, which is exponentially amplified by concatenating the Lie transformations associated to the higher order generators. As such, we want to limit the order of the generators, without modifying the base problem too much. But recall that the generator $g_k$ corresponds to the terms induced by the $k$-th order terms of the Hamiltonian. Hence we decide to concatenate the Lie transforms associated to the generators of $\mathcal{M}(n_1, n_2, n_{\max})$ up to the $n_1$-th order generator, giving the map $\tilde{\mathcal{M}}(n_1, n_2, n_{\max})$, defined as

$$\tilde{\mathcal{M}}(n_1, n_2, n_{\max}) = \mathcal{M}|_0 + \left[ \prod_{k=2}^{n_1} \left( \sum_{l=0}^{n_2} \frac{:g_k:^l}{l!} \right) \right] \Bigg|_{\leq n_{\max}} \tag{2.40}$$

where $\mathcal{M}|_0$ is the shift induced by $\mathcal{M}(n_1, n_2, n_{\max})$. We will see in the results section 3.4 that we recover [26] that the Dragt-Finn factorised map is more symplectic than the original map, i.e. $|\det(\text{Jac}_{\tilde{\mathcal{M}}}(\xi)) - 1| \ll |\det(\text{Jac}_{\mathcal{M}}(\xi)) - 1|$.

## 2.5 User Interface

An important part of the project was to develop this code keeping in mind that it was aimed to become a registered Julia package. As such, big efforts were put into making the code as user friendly and simple of use as possible. In this regard, three main axes where followed

### 2.5.1 MAD-X and Data Input

The Methodic Accelerator Design, or MAD, is a code that was developped in CERN to design and simulate particle accelerators [27]. As such, our code uses the same standards as MAD-X (current version of the MAD code) for beamline input and code setup by the user. First, let us talk about the formatting of beamline inputs that can be understood by the code. A more complete user guide is contained in the JuliAccel user manual, in appendix A. Let us have a look at the .mad file for a FODO beamline (see section 3.3), consisting of multiple cells of a focusing quadrupole, a drift, a defocusing quadrupole and another drift.

Listing 2.1: .mad file example for a FODO cell

```
!
!----- table of elements ----------------------------------------------------
!
d    : drift, l = 1.0;
!
qdf  : quadrupole, l = 0.2, k1 = 3.28;
qddf : quadrupole, l = 0.2, k1 = -3.31;
!
!
!----- table of segments ----------------------------------------------------
!
foc   : line = (qdf, d);
defoc : line = (qddf, d);
fodo  : line = (foc, defoc);
total : line = (100*fodo);
!
use, sequence = total;
!
!
!----- beam data ------------------------------------------------------------
!
beam, particle = electron, energy = 999.489e6;
!
```

In this file, we have three main sections: one defines the individual elements composing the beamline; the second one defines segments of the beamline, either made of multiple single elements or other segments; the last section gives the information on the beam itself. Note that a '`!`' at the beginning of a line means that this line should be ignored.

In the Table of Elements section, the user defines the names of the individual elements, along with all the information needed to generate the associatewd Lie transformation. For instance, for the drift, we only need its length `l`, but the quadrupole also needs the quadrupole strength `k1`, making it either positive (for the focusing quadrupole `qdf`) or negative (for the defocusing quadrupole `qddf`). The Table of Segments allows the user to name and compose the beamline. In particular, observe how for periodic segments, the parser allows integer multiplication such as '`total : line = (100*fodo);`'. The '`use`' command tells the parser which segment should eventually be used. Finally, the beam data is entered via the '`beam`' command, specifying the particle type and the energy of the beam. The energy of the beam is the energy of a particle following the reference trajectory with momentum $P_0$.

The package exports the `parse_madx` function which, given the path to a MAD-X data file, will parse the said file and extract all needed information about the beamline, and return an instance of the `MAD_X` structure:

```
struct MAD_X
    """Beam data"""
    beam::Dict{String, Any}
    """List of beamline elements"""
    lattice::Vector
end
```

where the `beam` field is a dictonary

```
Dict("particle"=>"<the_particle_type>", "energy"=><E>)
```

with $E$ the energy, in eV, of a particle with the reference momentum; The field `lattice` is simply a list of dictionaries containing all the data for each element. Note that it is the flattened sequence to be used, i.e. a vector containing an explicit list of the elements in the beamline, and has the same length as the number of single elements composing the beamline. Note however that the specific way in which these fields are formatted are ruled by how the parsing function works, and not meant to be easy of use for the user (which is why the .mad format and the parsing function exist in the first place!)

Along with the beamline data file, the package has a parser to recover the problem specific data such as the variable names and Hamiltonian formulas, which should be manipulable and variable for a same beamline depending on what the user wants to study. This is stored in a .japd (JuliAccel Problem Description) format file. We provide an example:

Listing 2.2: .japd file example for a FODO cell

```
!
!----- table of model settings ---------------------------------------------------
!
variables = (x, px, y, py, z, δ);
!
orders, trunc_order = 3, exp_order = 3, order = 8;
!
!
!----- table of Hamiltonians ------------------------------------------------
!
hamiltonian, type = drift, args = (), expr = δ/β0 - sqrt((1/β0+δ)^2 - px^2 - py^2 - 1/(β0*γ0)^2);
```

```
hamiltonian, type = quadrupole, args = (k1), expr = δ/β0 - sqrt((1/β0+δ)^2 - px^2 - py^2 - 1/(β0*γ0)^
    ↪ 2) + (k1/2)*(x^2 - y^2);
!
```

As one can see, this format is very closely inpired by the .mad format. First, the user defines in the Table of Variables the phase-space coordinates symbol, as they will appear in the Hamiltonians. In this table are also defined the truncation orders, with `trunc_order` the truncation order $n_1$ for the Hamiltonian TPSE, `exp_order` the exponential truncation order $n_2$, and `order` the overall truncation order $n_{\max}$.

The Table of Hamiltonians contains the formulas that should be used for the Hamiltonians. Observe that they take the `type` of the element described by that Hamiltonian, along with any other variable needed to define it, such as, in the above case, the quadrupole strength `k1`. The latter are stored in the `args` argument.

### 2.5.2 Variable Scope Communication

One of the big difficulties that were faced during the developpement of the package was to allow communication between the Main, user variable scope and the package global variable scope. Indeed, from the Julia performace guide, it is clearly expressed that any global variable should not change thorughout the code, and that it is best to set it as a global constant, which cannot be changed later down the line. This is a big issue here, since the constants that should be defined as global constants in our case are the variables symbols and the truncation orders. Of course, we want the user to be able to define these himself, in function of what problem he may be looking at. As such, we need a way to define global variables of the package outside of that package, and have all package functions run correctly. Enter Julia macros. Unfortunately, Julia being a "young" programming language, there is a lack of clear and unified metaprogramming documentation. In this context, we would like to discuss the solution we opted for.

First, we shoud explain why we decided to use macros instead of functions. Macros allows one to manipulate Julia Expressions, i.e. code as manipulable objects. In particular, this allows for very simple of use, but also flexible calls (such as in the number of arguments), enhancing usability and clarity of the end user's code.

The second argument, which was also the most constraining one, was that for a simple manipulation of symbols, we need the variables to "communicate" between the scopes, that is to be able to call a global variable of the package in the Main scope. This is not possible for variables that are defined after the initialisation of the package, since one cannot export a package variable before it was defined. Let us reformulate: when calling `import JuliAccel`, all the exports specified within the JuliAccel source code are exported from the package into the global scope. It is however impossible to export a variable that is not yet defined in the JuliAccel scope. But we would like to export a JuliAccel variable that is defined by calling a JuliAccel function, i.e. that can only be defined after having imported the package. The solution lies within the escaping capabilities of macros. A Julia macro returns and executes an Expression, i.e. Julia code, as if it were defined in a .jl file that got run. When returning this Expression, macros can escape the Expression, via the `esc()` function, into the scope where the macro has been called. As an example, say we import a module defining a macro `foo` which defines and evaluates an Expression `expr = :(x=3)`. When called, `@foo()` will define variable `x` in the module. If instead `@foo` defines and returns the *escaped* expression `expr` (i.e. returns `esc(expr)`), `x` will be defined in the scope where `@foo` was evaluated.

The macro that allows to define the phase space coordinates symbols and truncation orders in JuliAccel is written below:

```
macro set_problem(Γ::String, trunc_order::Int64, exp_order::Int64, order::Int64)

    @assert trunc_order ≤ order

    Γexpr = :(const Γcoord::Vector{TaylorN{Float64}} = set_variables($Γ, order=$(order+1)))
    eval(Γexpr)

    Γstr = replace(Γ, " "=>", ") * " = set_variables("*'"'*"$Γ"*'"'*", order=$(order+1))"

    eval(:(const trunc_order::Int64 = $trunc_order))
    eval(:(const order::Int64 = $order))
    eval(:(const exp_order::Int64 = $exp_order))
    eval(:(const Γdim::Int64 = length($Γcoord)))

    return esc(Meta.parse(Γstr))
end
```

This macro takes as argument Γ, a string representation of the variables (e.g. `"x px y py z δ"`), as well as the various truncation orders $n_1, n_2$ and $n_{\max}$. Let us look, line by line at what the macro does:

- `@assert trunc_order ≤ order`; This line checks that the truncation order $n_1$ does not exceed the overall truncation order, $n_{\max}$.
- `Γexpr = ...`; This line defines the Expression which, when evaluated in the following line, defines the variables Γ as the variables with respect to which the TPSEs will be computed.
- `eval(Γexpr)`; Evaluates `Γexpr`. This defines the symbols contained in Γ as constant global variables of the JuliAccel module scope.
- `Γstr = ...`; This function defines the string representation of the expression defining the variables. once parsed into an Expression, it is the Expression which shall be escaped to be evaluated in the global scope. Note that passing through a string representation in metaprogramming is a very innelegant solution. It has however the merit of being very easy to understand, and is the only solution that would work correctly whn evaluated in the Main scope.
- `eval(:(const trunc_order ...))` → `eval(:(const Γdim ...))`; These lines define the truncation orders and number of degrees of freedom as global constants in the JuliAccel scope.
- `return esc(...)`; The returned Expression is escaped and evaluated in the Main scope, defining the variables $x, px, y, py, z, \delta$ as variables in that scope.

This macro is of capital importance to the usability of the code. It is also a promising example of the little improvements that could be brought to the package and make it a very easy-of-use elegant, modern and fast code if continued.

### 2.5.3 The `gen_maps` Function

This function is the main function of the whole code, and is what should be used to obtian the Lie transformations as a user. It is a function that, with all needed user inputs, will automatically generate the Lie transformations associated to the beamline properties defined thanks to the .mad and .japd files. Its aim is to implement the steps to building the Lie transform $\mathcal{M}(n_1, n_2, n_{\max})$ automatically. Let us have a look at its documentation:

`JuliAccel.gen_maps` - Function

Provided well formated .mad and .japd files (as shown above in listings 2.5.1 and 2.5.1, respectively), computes the transfer polynomials for each element of the beamline.

`gen_map(mad, prob_def; symplectic=true)`
Arguments:

- `mad::String`: Path to the .mad file containing the beamline information;
- `prob_def::String`: Path to the .japd file containing the problem vairables and Hamiltonian formulae;
- `symplectic::Bool`: If set to `true`, the Lie transformations for the bamline elements will be symplectified via Dragt-Finn factorisation;

Returns `maps::Array{Vector{TaylorN{Float64}}}`, an ordered list of transfer polynomials $\mathcal{M}_k : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ giving the respective transfer maps of all elements of the beamline provided in `mad`.

Let us now do a runthrough of what the function does. The first step is to define the problem and define all variables. Hence it starts with a parsing of the `mad` and `prob_def` files and defining all variables defined within these files (variable names, beamline elements instances, beam energy...). Next, for every element in the beamline:

1. Identify the corresponding Hamiltonian and truncate its TPSE at order $n_1$.
2. Apply the Lie transformation generated by $-lH$ on $\xi$ using the function `lie_transform (-l*H, ξ)`. This function computes, for every order $k \leq n_2$ of the exponential expansion, the Lie bracket $\{-lH, \xi\}$ applied $k$ times, and sums all resulting polynomials, truncted at order $n_{\max}$.
3. If `symplectic == true`, the trasfer map is symplectified with Dragt-Finn factorisation. All Lie transformations $\exp(: g_k :), g_k \in \text{gen}(\mathcal{M})$ are concatenated together to give a single set of $2n$ polynomials.
4. The obtained $2n$ polynomials are added to `maps`, and finally `maps` is retruned.

# Chapter 3

# Results

## 3.1 Classical Harmonic Oscillator

The most basic physical model one can think of is of course the 1D Harmonic Oscillator. The phase space for this problem is given as generalised coordinates $\xi = (q, p)$. The Hamiltonian is then

$$H_{\text{osc}} = \frac{p^2}{2} + \frac{k}{2}q^2$$

with $k$ some system constant (e.g. the spring constant of a mass-spring oscillator). If we discretise the time $T$ of observation as $T = n\tau$ for some $n \in \mathbb{N}$, the transfer map $\mathcal{M}_{\text{osc}}(T)$ from $t = 0 \to T$ reads

$$\mathcal{M}_{\text{osc}}(T) = \mathcal{M}_{\text{osc}}(n\tau) = \exp(-n\tau : H_{\text{osc}} :) = \mathcal{M}^n_{\text{osc}}(\tau) \tag{3.1}$$

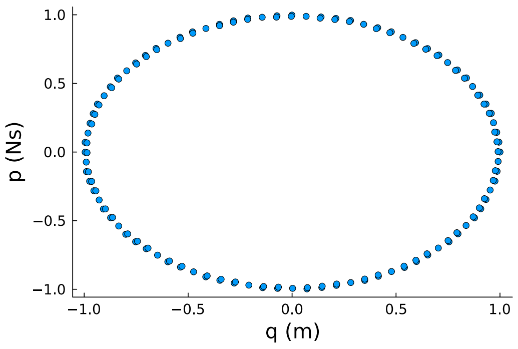
Note that thanks to the low dimensionality of the ${}_nD_v$ algebra for this case, we set $n_{\max}$, the truncation order of the TPSEs, to $n_2 + 1$ (see discussion in section 2.4).

Applying the transfer map to the phase coordinate for normalised constants, we obtain the phase space in figure 3.1a. This result is as expected. The comparison to the analytical result

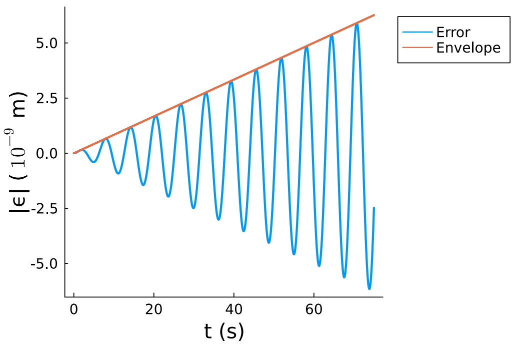
$$q(t) = \cos(\omega t), \quad p(t) = -m\omega \sin(\omega t) \qquad \text{with } \omega = \sqrt{\frac{k}{m}}$$

yields the error in Fig. 3.1b. Comparing the power series form of the analytical solution and the TPSE $\mathcal{M}_{\text{osc}}$, there only remains the higher order of the power series expansion of a siusoidal wave with frequency $\omega$, explaining the oscillating error. The linear enveloppe is explained by the picking up of almost constant errors $\mathcal{O}(\tau^{n_2})$ at every application of the transfer map.

If we compare the effect of the order on the error, we observe in Fig. 3.2 an exponential decrease of the error for increasing orders, up to a certain order at least. This is is because we reach machine precision errors, impeaching any further improvements.



(a) Phase space for order $n_2 = 4$ with normalised constant $k = 1$



(b) Error at order $n_2 = 4$ with normalised constant $k = 1$

Figure 3.1: Effect of the transfer map $\mathcal{M}_{\text{osc}}(2, 4, n_2 + 1)$

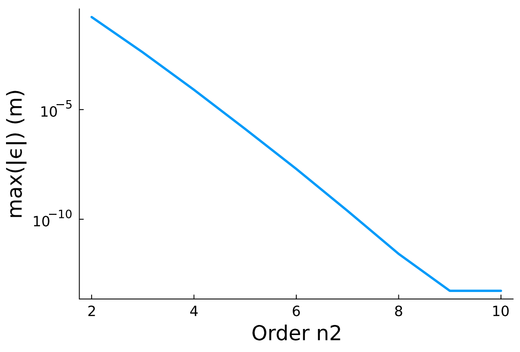


Figure 3.2: Errors at different exponential truncation orders $n_2$

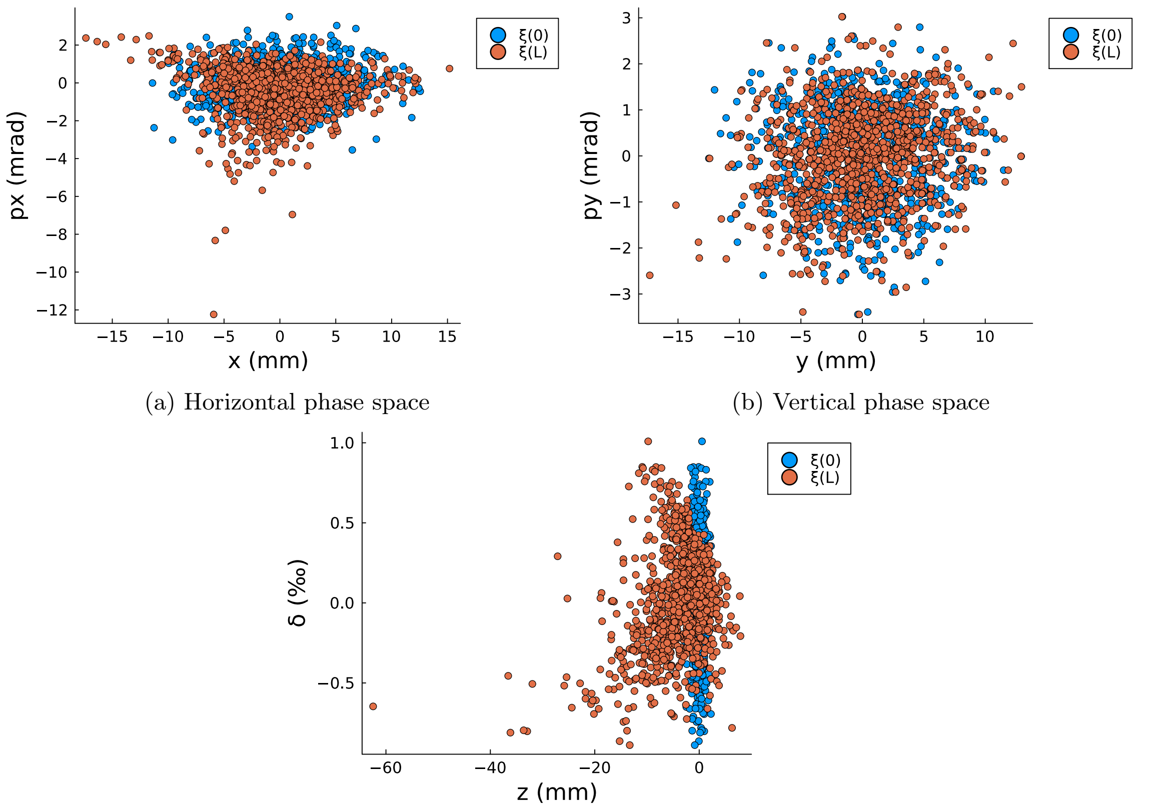
## 3.2 Magnetic Dipole

Recall that the dipole Hamiltonian is

$$H_{\text{di}} = \frac{\delta}{\beta_0} - (1 + hx)\sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} + (1 + hx)k_0\left(x - \frac{hx^2}{2(1 + hx)}\right) \tag{3.2}$$

with $k_0 = qB_0/P_0$ the dipole strength. In general, we align the reference trajectory on the dipole strength, so that the radius of curvature $\rho$ is that of a particle with reference momentum $P_0$. Hence $h = k_0$. Letting $L = 1.0$ m be the length of the quadrupole, the transfer map is given by $\mathcal{M}_{\text{di}}\xi = \exp(-L : H_{\text{di}} :)\xi$. We obtain, for 1000 particles, the phase space coordinates in figure 3.3.

These results are very encouraging, as we observe the expected effects of the dipole on phase-space [9]. In particular, we see the trailing part of the horizontal phase-space due to the fact that higher energy particles do not follow the reference trajectory bending and fall behind. On this plot, we also observe the coupling of the horizontal and vertical phase-spaces due to the factor $(1 + hx)$ in front of the square root in $H_{\text{di}}$ with the bunch stretching to negative $p_x$'s around $x = 0$. In the vertical phase space, we observe no particular effect, which is to be expected from the bending induced by



(a) Horizontal phase space

(b) Vertical phase space

(c) Longitudinal phase space

Figure 3.3: Effect on phase-space of a dipole on a Gaussian bunch of 1000 protons with energy $\mathcal{E}_{\text{kin}} = 100$ MeV, at expansion orders $n_1 = 3$, $n_2 = 6$.

the dipole only in the horizontal plane. Finally, in the longitudinal phase-space, we observe the effect of the coupling term $(1 + hx)k_0 \left( x - \frac{hx^2}{2(1+hx)} \right)$.

As a benchmark for the precision of our model, we use the `DifferentialEquations.jl` package [28], that provides integrators to solve ordinary differential equations. We use it to solve the Hamilton equations (1.21), (1.22). The integrator we use is Tsitouras' 5th-order Runge-Kutta method with 4th order interpolants. As expected, we obtain an exponential decrease in the error as $n_2$ increases. Since the non-linear phenomena appear in the horizontal plane of phase space, it is most interesting to look at the error in that plane, shown in figure 3.4.

## 3.3 The FODO Cell

We introduce the FODO cell [29], a set of drifts and quadrupoles allowing to store a particle beam in a periodic beamline. The FODO cell corresponds to one period, and is composed of focussing and defocusing quadrupoles and drifts. As such, the FODO cell is often compared to collections of optical elements, as in their representation in Fig. 3.5. First, recall that a drift is simply a zone of

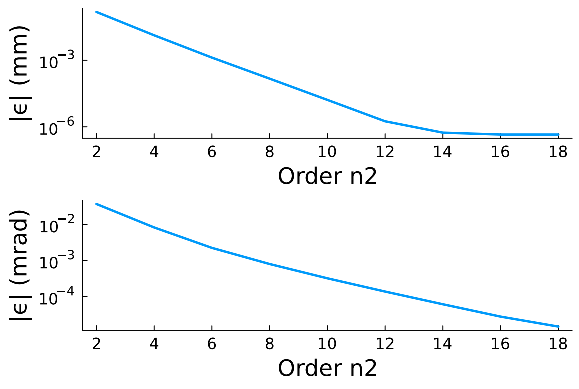


Figure 3.4: Average dipole error $x$ (top) and $p_x$ (bottom) coordinates for 1000 particles at truncation order $n_1 = 3$.

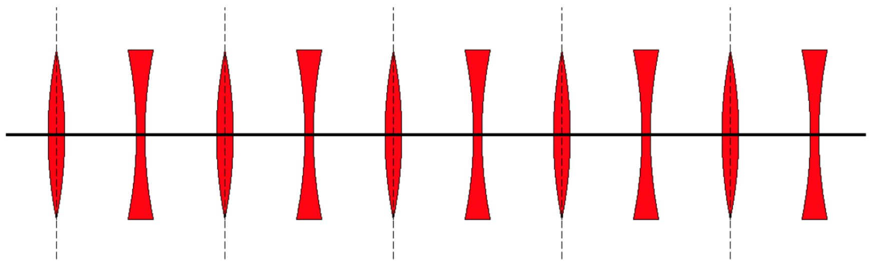


Figure 3.5: Optical representaiton of a FODO cell beamline. The broken lines delimit individual cells. Note that since the beamline is periodic, the cell can be shifted to simplify implementation. From [13]

the beamline with no external magnetic foeld $\mathbf{B} = 0$. Hence the Hamiltonian is simply

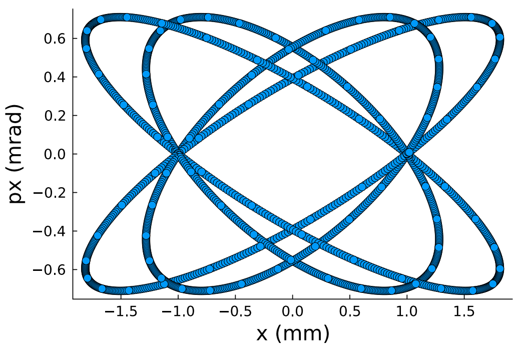
$$H_{\text{drift}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} \tag{3.3}$$

For a quadrupole, the Hamiltonian is

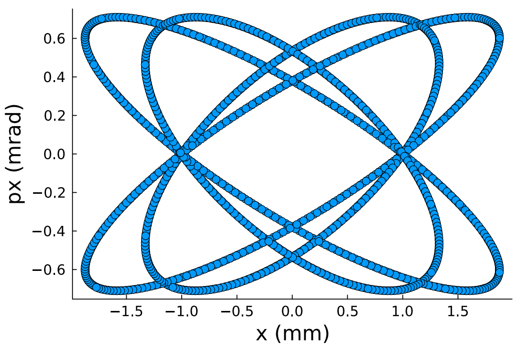
$$H_{\text{quad}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} + k_2\left(x^2 - y^2\right) \tag{3.4}$$

with $k_2 = \nabla B q/P_0$ the quadrupole strength. It sets in particular whether the quadrupole is focusing or de-focusing. Similarily to the harmonic oscillator, the term $k_2\left(x^2 - y^2\right)$ is either a focusing potential in $x$ and defocusing in $y$ and vice versa. If we simply look at the Lorentz force $\mathbf{F} = q(\mathbf{v}\times\mathbf{B})$, we easily recover this result. The FODO cell is a very relevent beramline element for us to study, because it is a simple example that allows to apply the model to a whole beamline and test how it behaves.

We track a particle through a beamline composed of 100 FODO cells and look at the resulting phase space, measured at the output of every element of the beamline. They appear in figure 3.6a.



(a) Phase space for the non-symplectic Lie Transform



(b) Phase space for the Dragt-Finn factorised Lie Transform

Figure 3.6: Horizontal phase space through 500 FODO cells at orders $n_1 = 3, n_2 = 4, n_{\max} = 9$.

As per expected, we see a periodic motion. In particular, we observe multiple orbits, each correxponding to an eliptic displacement in phase space, induced by each element. This makes the FODO cell a very interesting object in our study. Indeed, we use it as a sanity check for the Dragt-Finn factorisation algorithm. Indeed, while not necessary on a small FODO storage ring at sufficicently big truncation orders, we can use this to verify that after Dragt-Finn factorisation, we recover the eliptic motion in phase-space. This is what we observe in figure 3.6b, which corresponds to the phase space coordinates using Dragt-Finn factorised transfer maps. However, because Dragt-Finn factorisation modifies the transfer maps, it slightly changes the elipses appearing in figure 3.6, as we can see in figure 3.7.

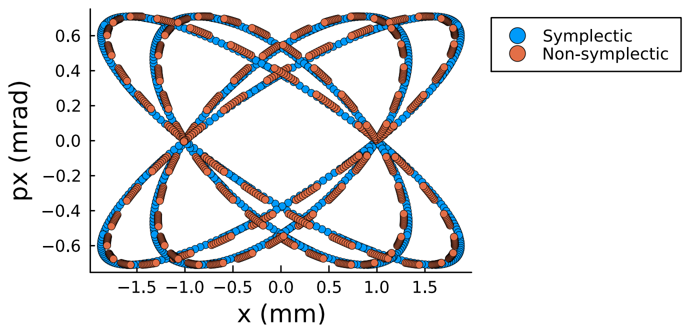


Figure 3.7: Raw and symplectic phase space through 500 FODO cells at orders $n_1 = 3, n_2 = 4, n_{\max} = 9$.

## 3.3.1 Dragt-Finn simplectification error: Twiss parameters

We introduce Twiss parameters and the Courant-Snyder inveriant, as they were presented in [13]. In what follows, we assume that the transfer maps are linear. Here, since we are using quadrupoles and drifts, this is a safe assumtion. Hence we look at the matrix representation of the transfer maps.

We are looking at a periodic lattice of path length period $C$. We write the "one-turn map", i.e. the transfer map through the whole beamline, from a point $s \in \mathbb{R}$ as $\mathbf{M}_{\text{OTM}}(s)$. Thanks to the lack of coupling of the horizontal plane with the others, we only consider the motion along the $x$ axis, and $\mathbf{M}_{\text{OTM}}$ becomes a $2 \times 2$ matrix. In particular, if $\mathbf{M}_{\text{OTM}}$ can be diagonalised, we write it as

$$\mathbf{M}_{\text{OTM}} = \mathbf{P} \underbrace{\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}}_{=:\mathbf{D}} \mathbf{P}^{-1} \tag{3.5}$$

Since the Eigenvalues $\lambda_{1,2}$ are not uniquely defined, we choose to set $P$ such that $\det(\mathbf{P}) = -i$. Since $\mathbf{M}_{\text{OTM}}$ is a symplectic matrix, we have $\lambda_1\lambda_2 = 1$. In particular, $\mathbf{M}_{\text{OTM}}$ being a real matrix, its eigenvalues are a complex conjugate pair, and there exists $\mu$ such that $\lambda_1 = e^{i\mu}$ and $\lambda_2 = e^{-i\mu}$. It is then convenient to do a rotation factorisation of $\mathbf{M}_{\text{OTM}}$:

$$\mathbf{M}_{\text{OTM}} = \bar{\mathbf{P}} \underbrace{\begin{pmatrix} \cos\mu & \sin\mu \\ -\sin\mu & \cos\mu \end{pmatrix}}_{=:\mathbf{R}(\mu)} \bar{\mathbf{P}}^{-1}$$

In particular, we can connect $\mathbf{R}(\mu)$ and $\mathbf{D}$ via the matrix $\mathbf{S}$, defined such that

$$\underbrace{\begin{pmatrix} \cos\mu & \sin\mu \\ -\sin\mu & \cos\mu \end{pmatrix}}_{\mathbf{R}(\mu)} = \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix}}_{\mathbf{S}^{-1}} \underbrace{\begin{pmatrix} e^{i\mu} & 0 \\ 0 & e^{-i\mu} \end{pmatrix}}_{\mathbf{D}(\mu)} \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{pmatrix}}_{\mathbf{S}} \tag{3.6}$$

Note that (3.6) implies that $\det(\bar{\mathbf{P}}) = 1$ and $\mathbf{R}^m(\mu) = \mathbf{R}(m\mu)$. This implies in particular that

$$\mathbf{M}^m_{\text{OTM}} = \bar{\mathbf{P}}\mathbf{R}(m\mu)\bar{\mathbf{P}}^{-1} \tag{3.7}$$

Next, if we write $\mathbf{R}(\mu) = \mathbf{I}_2 \cos\mu + \mathbf{J}_2 \sin\mu$, where $\mathbf{J}_2$ is the $2 \times 2$ rearranging matrix, as defined in (1.6), we obtain the *Twiss*-factorisation of $\mathbf{M}_{\text{OTM}}$

$$\mathbf{M}_{\text{OTM}} = \underbrace{\bar{\mathbf{P}}\mathbf{I}_2\bar{\mathbf{P}}^{-1}}_{\mathbf{I}_2} \cos\mu + \underbrace{\bar{\mathbf{P}}\mathbf{J}_2\bar{\mathbf{P}}^{-1}}_{\mathbf{\Omega}} \sin\mu \tag{3.8}$$

where direct computation show that $\det(\mathbf{\Omega}) = 1$, $\mathbf{\Omega}_{1,1} = \mathbf{\Omega}_{2,2}$ and $\mathbf{\Omega}_{1,2} > 0$. This gives the parametrisation

$$\mathbf{\Omega} = \begin{pmatrix} \alpha & \overbrace{\beta}^{>0} \\ -\underbrace{\frac{1+\alpha^2}{\beta}}_{=:\gamma>0} & -\alpha \end{pmatrix} \tag{3.9}$$

The parameters $\alpha, \beta$ and $\gamma$ are called the Twiss parameters. From (3.7) and (3.8), we se in particular that

$$\mathbf{M}^m_{\text{OTM}} = \mathbf{I}_2 \cos(m\mu) + \mathbf{\Omega} \sin(m\mu)$$

and conclude that the Twiss parameters are periodic. We also find

$$\bar{\mathbf{P}} = \begin{pmatrix} \sqrt{\beta} & 0 \\ -\frac{\alpha}{\sqrt{\beta}} & \frac{1}{\sqrt{\beta}} \end{pmatrix} \quad \Rightarrow \quad \mathbf{P} = \bar{\mathbf{P}}\mathbf{S}^{-1} = \begin{pmatrix} \sqrt{\frac{\beta}{2}} & \sqrt{\frac{\beta}{2}} \\ \frac{-\alpha+i}{\sqrt{2\beta}} & \frac{-\alpha-i}{\sqrt{2\beta}} \end{pmatrix}$$

Let us now look at how we can track the twiss parameters throughout the beamline. We wish to get the twiss parameters at path lengths $s_0$ and $s_1 + C$, where $C$ is the period of the beamline. There are two equivalent approaches to this:

$$\begin{tikzcd} s_0 \arrow[r] & s_1 \arrow[r] & s_1 + C \\ s_0 \arrow[r] & s_0 + C \arrow[r] & s_1 + C \end{tikzcd}$$

where in the top approach, we first apply $\mathbf{M}(s_0 \to s_1)$ then concatenate with the one-turn map at $s_1$, and in the second, we apply $\mathbf{M}_{\text{OTM}}(s_0)$ and then move to $s_1$. Mathematically, this gives

$$\mathbf{M}_{\text{OTM}}(s_1)\mathbf{M} = \mathbf{M}\mathbf{M}_{\text{OTM}}(s_1) \quad \Rightarrow \quad \mathbf{M}_{\text{OTM}}(s_1) = \mathbf{M}\mathbf{M}_{\text{OTM}}(s_0)\mathbf{M}^{-1}$$

This implies in particular that the Twiss matrix $\mathbf{\Omega}$ transforms as

$$\mathbf{\Omega}(s_1) = \mathbf{M}\mathbf{\Omega}(s_0)\mathbf{M}^{-1} \tag{3.10}$$

From this we build the *Courant-Snyder* invariant $J_{\text{CS}} = \xi^T \mathbf{J}_2 \mathbf{\Omega}^{-1}\xi$. Indeed, observe that

$$\xi^T(s_1)\mathbf{J}_2\mathbf{\Omega}^{-1}(s_1)\xi(s_1) = \xi^T(s_0)M^T\left(M\mathbf{\Omega}\mathbf{J}_2^{-1}M^T\right)^{-1}M\xi(s_0) = \xi^T(s_0)\mathbf{J}_2\mathbf{\Omega}^{-1}(s_0)\xi(s_0)$$

Explicitly, this can be writen

$$J_{\text{CS}} = \gamma x^2 + 2\alpha x p_x + \beta p_x^2 \tag{3.11}$$

This is the equation for an elipse, and explains the elliptical trajectories in figure 3.6. Each elipse corresponds to the elipse defined with the Twiss parameters at the entry of each 4 beamline elements in the FODO cell.

Most importantly, this gives us a tool to quantify the change in the system induced by the Dragt-Finn factorisation of the transfer maps. Indeed, if we compute what the Courant-Snyder invariant should be for each element, we can simply evaluate the error on the real value of $\gamma x^2 + 2\alpha x p_x + \beta p_x^2$, i.e. compute the error

$$\epsilon = \left|J_{\text{CS}} - (\gamma x^2 + 2\alpha x p_x + \beta p_x^2)\right|$$

In particular, we compute the Courant-Snyder invariant from the initial coordinates, and evaluate the error when running through the beamline. The error for a single FODO cell is shown in figure 3.8 As expected, we see in that figure first that increasing $n_2$ lowers the error, but also that the error on the symplectic maps is orders of magnitudes bigger ($\sim 10^{-2}$). Note that while in figure 3.8 the errors on the symplectic maps look identical, this is only because the lowering of the error due to an increase in $n_2$ is negligible before the error caused by Dragt-Finn factorisation.

## 3.4 The Sextupole, Divergences and Symplectification

Let us now look at the sextupole. Recall that the sextupole Hamiltonian is given by

$$H_{\text{sext}} = \frac{\delta}{\beta_0} - \sqrt{\left(\frac{1}{\beta_0} + \delta\right)^2 - p_x^2 - p_y^2 - \frac{1}{\beta_0^2\gamma_0^2}} + \frac{1}{6}k_2\left(x^3 - 3xy^2\right) \tag{3.12}$$

The term $\frac{1}{6}k_2\left(x^3 - 3xy^2\right)$ will participate in the transfer map with higher order terms. The Lie transform it generates modifies the phase space as it appears in figure 3.9.

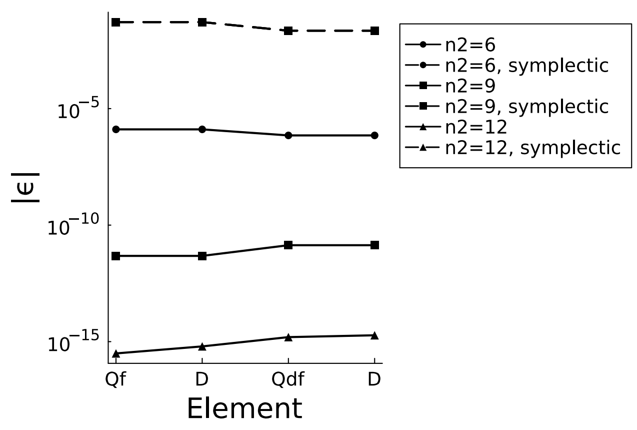


Figure 3.8: Relative error on the Courant-Snyder invariant through a FODO cell (focus. quad. $\rightarrow$ drift $\rightarrow$ defoc. quad. $\rightarrow$ drift). We look at order $n_1 = 3$ and $n_{\max} = 16$.

As for the magnetic dipole, we benchmark the performance of our model using Tsitouras' 5th-order Runge-Kutta method with 4th order interpolants. As before, we have an exponential decrease in teh error with the truncation ordewr $n_2$. The results are shown in figure 3.10

As we can see in figure 3.9, the sextupole induces non-linearities, prone to divergences. As such, we use it to add non-linearities in the FODO cell. The sequence becomes

$$Q_f \rightarrow D/2 \rightarrow S \rightarrow D/2 \rightarrow Q_d \rightarrow D$$

with $Q_{f,d}$ the focusing and defocusing quadrupoles respectively, $D$ the drifts and $S$ the sextupole. This non-linearity breaks the eliptic motion, as seen in figure 3.11. As we can see in that same figure, the Dragt-Finn factorisation allows to recover the eliptic motion in phase-space.

It is therefore interesting to study and quantify the effect of the Dragt-Finn factorisation theorem. We evaluate the symplecticity of a transfer map via the determinent of its Jacobian. We truncated the Hamiltonian at order 5, and for various truncation orders $n_{\max}$, we evaluate the evolution of the Jacobian with respect to the exponential truncation order. The resulting data is shown in figure 3.12 As we can see, we recover that the Dragt-Finn factorisation allows to correct for the non-symplecticity of the truncation of the Lie transforms. Note that the Dragt-Finn factorised maps are also not fuly symplectic, since they are also truncated. However the truncation by eliminating the Lie transforms of higher order generators still gives much better results, decreasing the error on the Jacobian determinant by orders of magnitude. These results also show that the possibility to choose the overall truncation order $n_{\max}$ can greatly decrease runtime without affecting the symplecticity of the maps. Indeed, observe that while the degrees of the Lie transformations for $n_2 \geq 3$ are greater than 10, ew see a limited decrease in non-symplecticity from $n_{\max} = 7$ to 9 in figure 3.12. As such, we can do a trade-off and make the computation much faster by decreasing $n_{\max}$ for a limited cost in non-symplecticity. In particular, if we aim for machine precision, decreasing $n_{\max}$ might not even have an effect on the Jacobian determinant.

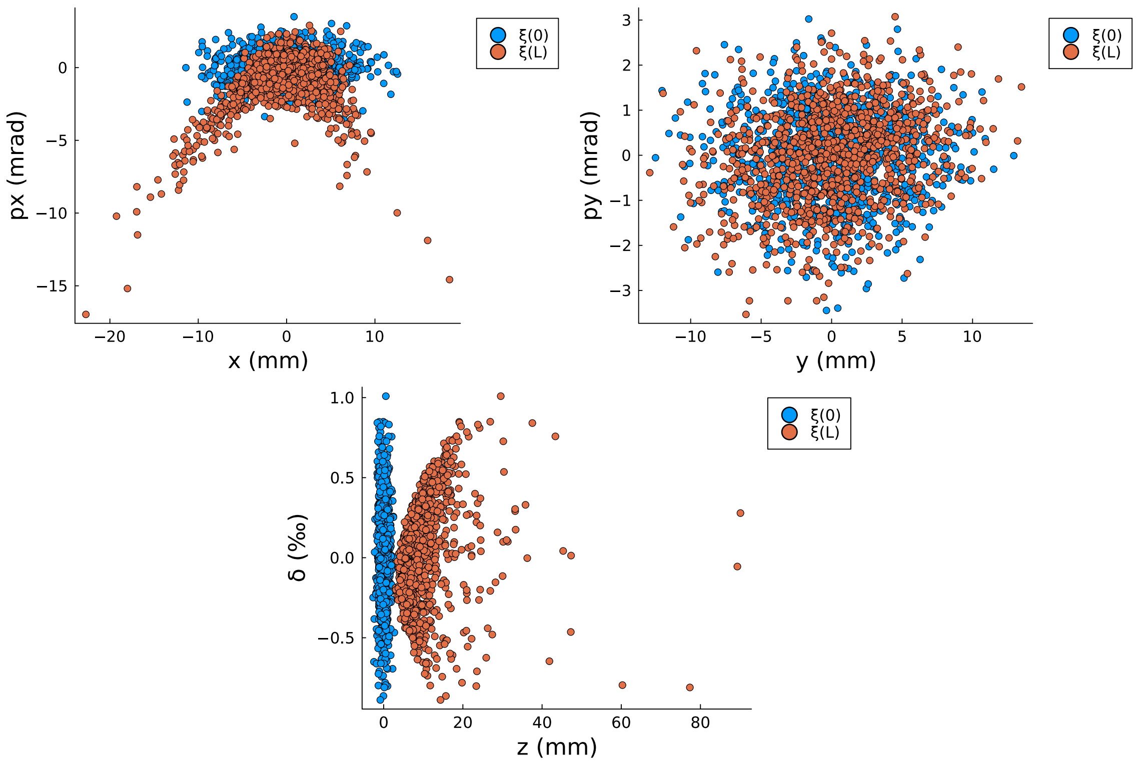


Figure 3.9: Effect on phase-space of a sextupole of length $L = 1$ m, with strength $k_2 = 0.134$ m$^{-3}$, on a Gaussian bunch of 1000 protons with energy $\mathcal{E}_{\text{kin}} = 100$ MeV, at expansion orders $n_1 = 3$, $n_2 = 6$

## 3.5 Runtime of the Functions

One problem that must be addressed is the runtime of the algorithm. We expect the runtime to compute the Lie transforms to increase significantly with the truncation order (see section 2.1.1). What is more, the Dragt-Finn factorisation step should slow down the algorithm, due to the need to compute and concatenate Lie transforms associated to the generators of the map. The results obtained when studying the runtime are illustrated in figure 3.13, and confirm our expectations. Note that the higher order expansions implying more terms, the runtime for the numerical evaluation of polynomials is also increasing with order, as in figure 3.14.

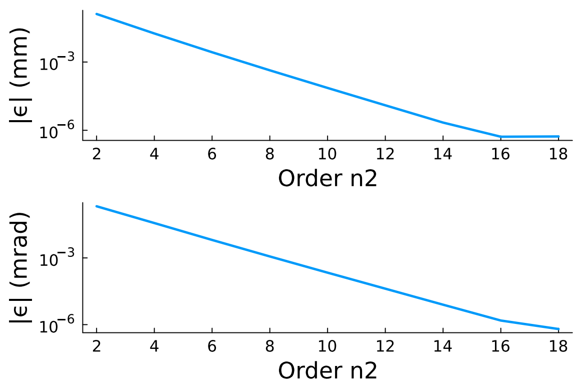


Figure 3.10: Average sextupole error $x$ (top) and $p_x$ (bottom) coordinates for 1000 particles at truncation order $n_1 = 3$.

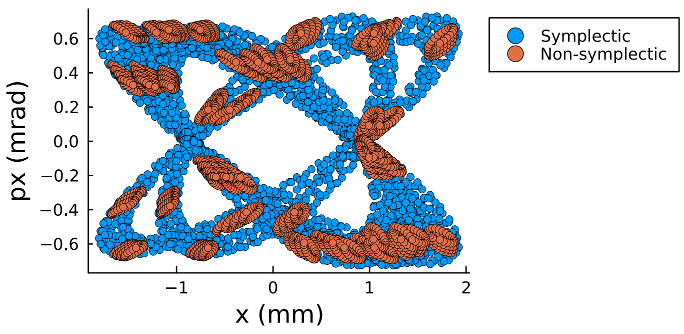


Figure 3.11: Horizontal phase space throught 500 sextupole FODO cells with $n_1 = 3$, $n_2 = 4$ and $n_{\max} = 9$.

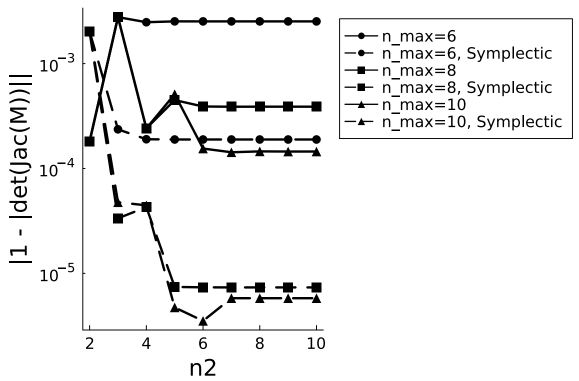


Figure 3.12: Order study of the symplecticity of a sextupole map at truncation order $n_1 = 4$.

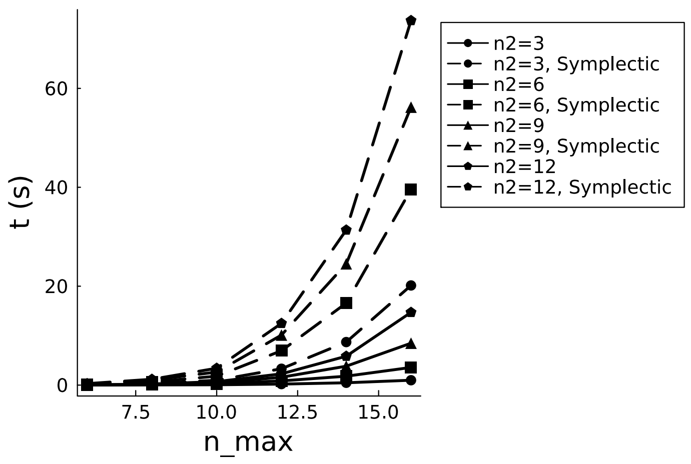


Figure 3.13: Average sextupole maps generation runtime for different orders, with $n_1 = 5$.

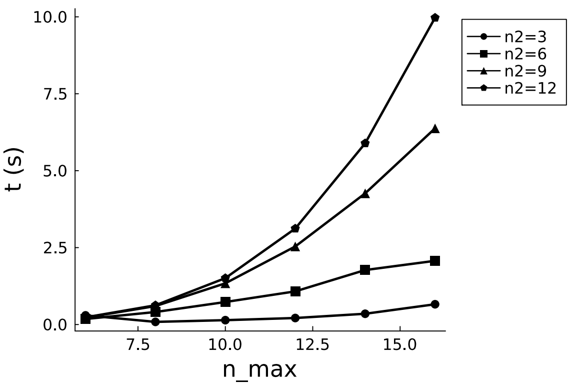


Figure 3.14: Average sextupole maps evaluation runtime on 1000 particles for different orders, with $n_1 = 5$.

# Conclusion and Discussion

In this thesis, we implemented a truncated power series algebra in Julia to moodel beam dynamics. We recovereed the equations of motion and built the transfer maps through a beamline. We defined and presented some properties of symplectic maps, and introduced the Dragt-Finn factorisation to restore the symplecticity of polynomial maps truncated at finite order.

We presented some results obtained on simple systems like the harmonic oscillator, single beamline elements and a FODO lattice. We discussed on the runtime and performance of the model for symplectic tracking through particle accelerators.

This sudy provides a whole package that can be used for dynamical system tracking, e.g. particle accelerator beamlines. This package offers the posisbility to restore the symplecticity of truncated maps, allowing to ensure phase-space conservation even at lower truncation orders. As discussed in chapters 2 and 3, this approach offers easy to use code, giving good results. It could still be optimised to further lower runtimes.

With further optimising and work on the project, it will be possible to develop a complete package for efficient and fully symplectic tracking of any dynamical system with known Hamiltonian. While developped for beamline modelling in particular, this package has the potential to become the backbone of any code modelling clasiscal dynamics, in any field of phyisics and applications.

With minor improvements, the model for classical particles can also be updated to complex dynamical variables and allow for quantum modelling. This would allow us to incorporate quantum phenomena to the model, to study quantum states within beams.

# Appendix A

# User Manual

## A.1 Package Activation

The `JuliAccel.jl` package can be activated and used via the package manager of your Julia REPL. First, it is important to open the REPL in the directory containing the JuliAccel repo. You can check which directory you are in in the Julia REPL via the command `pwd()`.

```
julia> pwd()
"path/to/repo"

julia> readdir()
1-element Vector{String}:
 "juliaccel.jl"
```

You can then activate the JuliAccel environment and use JuliAccel as a published Julia package

```
(@v1.8) pkg> activate JuliAccel.jl

julia> using JuliAccel
```

Should you need to activate the environment within a Julia code, it can be done by adding the following lines at at the top of said code

```
using Pkg
Pkg.activate("path/to/repo/juliaccel.jl")
using JuliAccel
```

## A.2 Basic Use

### Model Data Definition

The code builds transfer maps $\mathcal{M} = \exp\left(-L : H :\right)$ as truncated power series

$$\mathcal{M}(n_1, n_2, n_{\max}) = \left.\left(\sum_{k=0}^{n_2} \frac{: H|_{\leq n_1} :^k}{k!}\right)\right|_{\leq n_{\max}} \tag{A.1}$$

As such, when initialising the code, one must define the variables symbols and truncation orders $n_1, n_2, n_{\max}$. This is done using the `@set_problem` macro:

```
n1, n2, n_max = 3, 4, 10
Γ = eval(:(@set_problem("x px y py z δ", $n1, $n2, $n_max)))
```

It is then possible to explicitely define functions of these variables:

```
julia> f = sqrt(1 + x^2 - px) + exp(y+py) - z*δ
2.0 - 0.5 px + 1.0 y + 1.0 py + 0.5 x² + [...] + 𝒪(‖x‖¹¹)
```

Note that here, we turn the call of the `@set_problem` macro into an Expression which we evaluate. This is done in order to have as arguments the symbols of the values $n_1, n_2, n_{\max}$ via interpolation. Otherwise, we would need to enter their numerical values.

TPSEs defined like this can easily be evaluated by calling them like functions:

```
julia> f = sqrt(1 + x^2 - px) + exp(y+py) - z*δ
2.0 - 0.5 px + 1.0 y + 1.0 py + 0.5 x² + [...] + 𝒪(‖x‖¹¹)

julia> f(1,1,0,3,0,2)
20.80785435267857
```

It is also possible to use a vector of length corresponding to the number of variables as an argument. The order of the arguments is the same as the one in which the variables symbols are defined in `@set_problem`.

## Map Generation

The truncation at order $n_1$ of any function can be obtained with the `tps_expansion!()` function, which sets the coefficients associated to monomials of order greater than $n_1$ to 0.

```
julia> @set_variables("q p", 3, 5, 10)

julia> f = q + q*p - q*p^3 + 2q^2*p^3
1.0 q + 1.0 q p - 1.0 q p³ + 2.0 q² p³  + 𝒪(‖x‖¹¹)

julia> tps_expansion!(f)

julia> f
1.0 q + 1.0 q p - 1.0 q p³ + 𝒪(‖x‖¹¹)
```

The Lie operator and Lie transforms associated to some function $f$ can then be evaluated on a function $g$ using, respectively,

```
lie_bracket(f, g)
lie_transform(f, g)
```

If $g$ is an array of functions $g_i$, calling `lie_transform(f,g)` will return an array of polynomials $h$ such that `h_i == lie_transform(f,g_i)`.

It is possible, given a set of polnomials $\mathcal{M} = \exp(:\ f\ :)g$, to find, up to order $n_1$, the generators $f_k$ of the Dragt-Finn factorised expression $\tilde{\mathcal{M}} = \exp(:\ f_2\ :)\ldots\exp(:\ f_{n_1}\ :)g$ using the `dragt_finn_factorisation` function. Here is an example applying the Harmonic oscillator Lie transformation on coordinates $\Gamma = (q,p)$

```
julia> H = (q^2 + p^2) / 2
0.5 q² + 0.5 p² + 𝒪(‖x‖¹¹)

julia> lie_transform(H, Γ) |> dragt_finn_factorisation
2-element Vector{TaylorN{Float64}}:
 0.6728716563786007 q - 0.7368827160493827 p + 𝒪(‖x‖¹¹)
 0.7368827160493827 q + 0.6728716563786007 p + 𝒪(‖x‖¹¹)
```

Note that the `dragt_finn_factorisation` function returns a set of polynomial maps. That is because the Lie transforms associated to the generators are concatenated before returning the polynomial maps.

The symplecticity of a set of polynomial maps can be verified by computing the Jacobian of this set of map, and in particular its determinant. As such, we have the function `jacobian_det`, which takes transfer maps $\mathcal{M}$ and coordinates $\xi$, and evaluates the determinant of the Jacobian of $\mathcal{M}$ at $\xi$:

```
jacobian_det(M::Vector{TaylorN{Float64}}, coord::Vector{Flaot64})
```

If no coordinates are given, the Jacobian will be evaluated at 0. The coordinates can also be given as a matrix whose columns are phase-space coordinates. Then the value returned will be the average of the evaluation of the jacobian determinant over these columns.

## A.3 Accelerator Modelling Features

Since JuliAccel was developped for accelerator modelling, it contains some features usefull to that end.

### Data Input

#### Bemline Data

There is a possibility to input a beamline sequence and the associated beam data to the program inspired by the MAD X package [27] of CERN. Let us look at a sample MAD X input file for a FODO cell

Listing A.1: .mad file example for a FODO cell

```
!
!----- table of elements ----------------------------------------------------
!
d    : drift, l = 1.0;
!
qdf  : quadrupole, l = 0.2, k1 = 3.28;
qddf : quadrupole, l = 0.2, k1 = -3.31;
!
!
!----- table of segments ----------------------------------------------------
!
foc   : line = (qdf, d);
defoc : line = (qddf, d);
fodo  : line = (foc, defoc);
total : line = (100*fodo);
!
use, sequence = total;
!
!
!----- beam data ------------------------------------------------------------
!
beam, particle = electron, energy = 999.489e6;
!
```

Before anything, note that all lines that should not be parsed start with a “ ! ”, marking comment lines. At the beginning of the document, we precise a table of elements, giving the length and other properties of the individual elements of the beamline. The elements that have been implemented yet are

- **Drift** space data in the format `label: drift, l=real;` with `l` the length of the drift.
- **Quadrupole** data in the format `label: quadrupole, l=real, k1=real;` with `l` the length and `k1` the quadrupole strength.
- **Bending magnet** data in the format

  ```
  label: sbend, l=real, angle=real, k1=real, e1=real, e2=real;
  ```

  with `l` the length of the dipole, `angle` the bend angle, `k1` the dipole strength, `e1` the rotation angle for the entrance pole face and `e2` the rotation angle for the exit pole face. What $e_1$ and $e_2$ actually represent is illustrated in figure A.1.

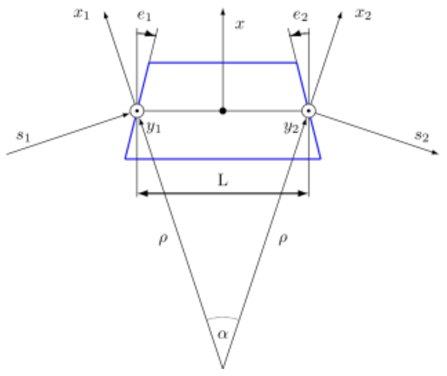


Figure A.1: Illustration of bending magnet data (from [27]).

- **Sextupole** data in the format

  ```
  label: sextupole, l=real, k2=real, tilt=real;
  ```

  with `l` the sextupole length, `k2` the sextupole strength and `tilt` the roll angle about the longitudinal axis (0 rad is a normal sextupole and a tilt of $\pi/6$ turns a normal sextupole into a skew quadrupole, and vice versa).

The second section of a .mad file is the table of segments, defining the order of the elements within the beamline. We do so by defining segments, which are ordered lists of either individual elemnts or other segments. Segments are defined with the format

```
label: line=(...);
```

where the parenthesis contain the comma separated list of elements composing the defined segment. Note that a segment containing $n$ elements `elem` can be expressed as `label: line=(n*elem);`. We finally say which segment is to be used with the line

```
use, sequence=label;
```

where `label` is the label of the segment to be used.

The last section of a .mad file is the beam data. It is simply defined with the sequence

```
beam, particle=name, energy=real;
```

precising the beam particle type name, either `proton` or `electron`, for now, and beam energy, in eV.

Files in the format .mad are parsed using the function `parse_madx`, which takes as argument the string containing the path to the .mad file to be parsed. The value returned is an instance of the `MAD_X` type, which has the field `beam`, which is a dictionary

```
Dict("particle"=>"<the_particle_type>", "energy"=><E>)
```

and the field `lattice`, which is a vector of beamline elements data, ordered as in the beamline.

### Model Data

The data for the model implementation is defined in a .japd type file, greatly inspired by the .mad file format. A sample file is provided here:

Listing A.2: .japd file example for a FODO cell

```
!
!----- table of model settings ---------------------------------------------------
!
variables = (x, px, y, py, z, δ);
!
orders, trunc_order = 3, exp_order = 3, order = 8;
!
!
!----- table of Hamiltonians -------------------------------------------------
!
hamiltonian, type = drift, args = (), expr = δ/β0 - sqrt((1/β0+δ)^2 - px^2 - py^2 - 1/(β0*γ0)^2);
hamiltonian, type = quadrupole, args = (k1), expr = δ/β0 - sqrt((1/β0+δ)^2 - px^2 - py^2 - 1/(β0*γ0)^
    ↪ 2) + (k1/2)*(x^2 - y^2);
!
```

Just like in the .mad file fomat, lines starting with a “!” are ignored by the parser. First, we have a table of model settings. They precise the variables names with

```
variables=(...);
```

with the parentheses enclosing a comma separated list of variables names. The various truncation orders $n_1, n_2$ and $n_{\max}$ are provided with the format

```
orders, trunc_order=integer, exp_order=integer, order=integer;
```

where the three orders correspond respectively to $n_1, n_2$ and $n_{\max}$. The other section in the problem definition file is the table of Hamiltonians, which gives the Hamiltonian formulae to be used for each element. Each input is formated as

```
hamiltonian, type=element_type, args=(...), expr=formula;
```

with the `type` precising the type of element corresponding to this Hamiltonian, `args` a list of input data used in the formula. They ususally correspond to the entries in the .mad file for the concerned type. Finally, expr is the Hamiltonian formula. Observe that in field expr, one can call variables that are defined within your code, as for example $\beta_0$ in listing A.2 above.

The problem data files are parsed using the function extract_model_data. It takes as argument a string containing the path to the .japd file, and returns an instance of the ModelData type, containing fields

- variables::String, which is the string of variables to be used in @set_problem.
- truncation_orders:Dict{String, Integer}, a dictionnary mapping the truncation orders to their values: Dict("truncation_order"=>n1, "exp_order"=>n2, "order"=>n_max)
- Hamiltonians::Dict{String, String}, a dictionnary mapping the type of beamline element to a string representation of its hamiltonian.

### The gen_maps function

The gen_maps function generates the transfer maps for a beamline with symplectic possibilities. It takes arguments

- mad: a string containing the path to a .mad file.
- prob_def: a string containing the path to a .japd file.
- symplectic: a boolean defining whether or not the maps should be Dragt-Finn factorised.

This function returns an array of transfer maps corresponding to the Lie transforms generated for each lement of the beamline provided in the MAD X data file.

## Physical Tools

### Particles

The package implements a Particle structure, with attributes the particle mass, its charge and its name (pname). In particular, it provides predefined instances for Protons and Electrons, with their masses and charges predefined, and with pnames "proton" and "electron" respectively.

### Lorentz Factors

It is also possible to compute the Lorentz $\gamma$ factor for a particle beam from its nominal kinetic energy and the particle mass. The function used is get_gamma(ekin, mass), returning the values of $\gamma$ for said particle beam. From this $\gamma$, we can compute the associated velovity ratio $\beta = v/c$ with the get_beta($\gamma$) function.

It is also possible to obtain these factors from a parsed MAD X data file, i.e. an instance of the MAD_X type, with the function get_lorentz_factors(mad::MAD_X).

### Distributions

To model particle beams, the package has the tools to build statistical distributions of particle phase space coordinates. The code provides a Distrib type, which is an abstract type for special distributions. In particular, we have defined the `Gaussian` subtype, which has fields $\mu$ and $\Sigma$, respectively the mean values for each coordinate and the covariance matrix.

The coordinates associated to a distribution are obtained with the `gen_distribution` function, taking as arguments a `Distrib` object and an integer $N$ of how many particles are considered. The returned value is a matrix whose columns are the phase-space coordinates of each of the $N$ particles.

## A.4 API

- `MAD_X` - Struct
  Outputs of a MAD X file parsing.
  **Fields**
  - `beam`: Beam data.
  - `lattice`: List of beamline elements.
- `parse_madx` - Function
  `parse_madx(file::String)`
  Parses the precised MAD X file and returns a `MAD_X` object.
- `ModelData` - Struct
  Output of the model data file parsing.
  **Fields**
  - `variables`: Variables symbol.
  - `truncation_orders`: Truncation orders.
  - `hamiltonians`: Hamiltonian formulas.
- `extract_model_data` - Function
  `extract_model_data(file::String)`
  Parses the precised map generation data, i.e. vaiables, truncation orders and Hamiltonian formulas. Returns a `ModelData` instance.
  **Examples**

  ```
  julia> file = "path/to/model/data.japd"

  julia> extract_model_data(hamiltonians)
  ```

- `@set_problem` - Macro
  **Examples**

  ```
  eval(:(@set_problem("q p", $n1, $n2, $n_max)))
  H = (q^2 + p^2) / 2
  ```

  In the above code snipet
  - `n1` is the truncation order used by `tps_expansion!`
  - `n2` is the truncation order for exponential expansions used by `lie_transform`.

  - `n_max` is the truncation order for any symbolic expression.

- `tps_expression!` - Function
  `tps_expression!(f::TaylorN{Float64})`
  Computes the truncated power series expansion of `f` at order `n1` defined via the `@set_problem` macro. Modifies the input function `f`.
  **Examples**

```
    julia> @set_variables("q p", 3, 5, 10)
    julia> f = q + q*p - q*p^3 + 2q^2*p^3
     1.0 q + 1.0 q p - 1.0 q p³ + 2.0 q² p³  + 𝒪(||x||¹¹)
    julia> tps_expansion!(f)
    julia> f
     1.0 q + 1.0 q p - 1.0 q p³ + 𝒪(||x||¹¹)
```

- `Proton` - Struct
  `Proton()`
  Proton instance.
  **Fileds**
  - `mass`: Particle mass.
  - `charge`: Particle charge.
  - `pname`: Particle name.

- `Electron` - Struct
  `Electron()`
  Electron instance.
  **Fileds**
  - `mass`: Particle mass.
  - `charge`: Particle charge.
  - `pname`: Particle name.

- `get_gamma` - Function
  `get_gamma(ekin::Float64, mass::Float64)`
  Computes the Lorentz factor $\gamma$ from the kinetic energy `ekin` of the beam and particle mass `mass`.

- `get_beta` - Function
  `get_beta(`$\gamma$`::Float64)`
  Computes the velocity ratio $\beta$ from the Lorentz factor $\gamma$.

- `get_lorentz_factors` - Function
  `get_lorentz_factors(mad::MAD_X)`
  Computes the $\gamma$ and $\beta$ factors from the beam data contained in a `MAD_X` instance obtained from parsing a MAD X data file.

- `Distribution` - Abstract type
  Abstract type for distributions.

- `Gaussian` - Struct
  `Gaussian(`$\mu$`::Vector{Float64},` $\Sigma$`::Matrix{Float64})`
  Gaussian Distribution instance.
  **Fields**
  - $\mu$: Mean vector.
  - $\Sigma$: Covariance matrix.
- `gen_distribution` - Function
  `gen_distribution(D::Distribution, N::Int64)`
  Generates `N` datapoints following distribution `D`.
- `lie_bracket` - Function
  `lie_bracket(f::TaylorN{Float64}, g::TaylorN{Float64})`
  Computes the Lie bracket of two `TaylorN{Float64}` instances.
- `lie_transform` - Function
  `lie_transform(f::TaylorN{Float64}, g::TaylorN{Float64}; n::Int64=exp_order)`
  Computes the action of the Lie transform associated to the first argument `f`, truncated at order `n`, on `g`.
  If `g` is a vector, the trnansform is broadcast over the entries of the vector.
  The truncation order `n` takes default value `n2` for the exponential, defined using the `@set_problem` macro.
- `dragt_finn_factorisation` - Function
  `dragt_finn_factorisation(M::Vector{TaylorN{Float64}})`
  This function computes the generators of the a set of polynomials `M`, and concatenates their associated Lie transforms.
  It returns a set of polynomials corresponding to the Lie transformations obtained with Dragt-Finn factorisation of `M`.
- `jacobian_det` - Function
  `jacobian_det(M::Vector{TaylorN{Float64}}, coords::Vector{Float64})`
  `jacobian_det(M::Vector{TaylorN{Float64}})`
  `jacobian_det(M::Vector{TaylorN{Float64}}, coords::Matrix{Float64})`
  Computes the Jacobian of the set of multivariate polynomials `M` and computes its value at coordinates `coords`.
  If no coordinates `coords` are given, the Jacobian will be evaluated at 0.
  If `coords` is a Matrix, the value of the Jacobian will be averaged over the columns of `coords`.
- `concatenate` - Function
  `concatenate(M::Vector{TaylorN{Float64}}, N::Vector{TaylorN{Float64}})`
  This function concatenates Lie trnasfer maps: `M`$\circ$`N`.
- `gen_maps` - Function
  `gen_maps(mad::String, prob_def::String; symplectic=true)`
  Provided well formated MAD X and model data files, `mad` and `prob_def` respectively, computes the transfer polynomials for each element of the beamline and returns an ordered array containing these. The argument `symplectic` allows the user to explicitly Dragt-Finn factorise each map (`symplectic=true`) or not (`symplectic=false`).

Appendix B

# Sample Codes: the FODO Cell

```julia
# ========================================================================

# IMPORTS

print("Importing packages... ")

include("path/to/git/repo/juliaccel.jl/src/JuliAccel.jl")
using .JuliAccel

using Plots
using ProgressBars

using Dates

println("Done!")


# ========================================================================

# VARIABLES

# Initial coordinates
ξi = [1.0, 0.0, 1.0, 0.0, 1.0, 0.0] .* 10^(-3)


# Dragt-Finn factorise?
symp = false


# Beamline data files
mad  = "path/to/fodo.mad"
prob = "path/to/fodo.japd"


# Saving paths
path = "path/to/save/data/"
```

```
# ======================================================================

# DATA SAVING

# Make saving Directory
time = now()
dir = path * "data_$time/"
mkdir(dir)


# Copy .japd files
cp(prob, dir*"fodo.japd")


# Save symplecticity
symp_log = dir*"symplectic.csv"
open(symp_log, "w") do io
    print(io, symp)
end


# Create coordinates path
coord = dir * "coordinates.csv"

init_str = "0,"
for i in 1:6
    global init_str *= "$(ξi[i]),"
end
chop(init_str, head=0, tail=1)

open(coord, "w") do io
    println(io, init_str)
end


# ======================================================================

# PROBLEM DEFINITION

print("Defining problem... ")

# Compute maps
Ms = gen_maps(mad, prob, symplectic=symp)

println("Done!")


# ======================================================================

# SYSTEM EVOLUTION

println("Dragging particle through...")

N = length(Ms)

ξ = Matrix{Float64}(undef, 6, length(Ms))
ξ[:, 1] = copy(ξi)

for n in ProgressBar(1:N-1)

    ξt = Ms[n](ξ[:, n])
    ξ[:, n+1] = copy(ξt)

    str = "0,"
    for i in 1:6
        str *= "$(ξt[i]),"
    end
    chop(init_str, head=0, tail=1)
```

```
open(coord, "a") do io
    println(io, str)
end
end


# ========================================================================

# PLOT

print("Plotting data... ")

...

println("Done!")
```

# Appendix C

# Sample Codes: the Dipole

It is also possible to define the Hamiltonian directly in the code.

```
# ========================================================================

# IMPORTS

print("Importing packages... ")

...

println("Done!")


# ========================================================================

# VARIABLES

# Truncation orders
n1    = 3
n2    = 6
n_max = n2 * (n1-1) + 1


# Dipole data
L = 1.0                 # Dipole length
B = 0.5                 # Dipole magnetic field


# Beam data

# Phase space distribution
μ  = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Σ  = [
    16.0  0.0   0.0  0.0  0.0    0.0    ;
     0.0  1.0   0.0  0.0  0.0    0.0    ;
     0.0  0.0  16.0  0.0  0.0    0.0    ;
     0.0  0.0   0.0  1.0  0.0    0.0    ;
     0.0  0.0   0.0  0.0  0.709  0.0    ;
     0.0  0.0   0.0  0.0  0.0    0.0981 ;
]

# Particles
particle = Proton()             # Particle type
N  = 1000                       # Number of particles

ekin = 100e6                    # Beam energy (eV)
```

```
# Saving paths
path = "path/to/save/data/"


# ========================================================================

# DATA SAVING

# Make saving Directory
time = now()
dir = path * "data_$time/"
mkdir(dir)


# Save truncation orders
trunks = dir * "truncation_orders.csv"
orders = [
    "n1,$n1",
    "n2,$n2",
    "n_max,$n_max"
]

open(trunks, "w") do io
    for str in orders
        println(io, str)
    end
end


# Initial coordinates path
initial_coord = dir * "initial_coordinates.csv"

open(initial_coord, "w") do io
    print(io,"")
end


# Final coordinates path
final_coord = dir * "final_coordinates.csv"

open(final_coord, "w") do io
    print(io,"")
end


# ========================================================================

# PROBLEM DEFINITION

print("Generating Beam... ")

# Generate beam
D  = Gaussian(μ, Σ)
ξi = gen_distribution(D, N)

print("Saving initial phase-space... ")

for i in 1:N
    open(initial_coord, "a") do io
        for j in 1:5
            print(io, "$(ξi[j,i]),")
        end
        println(io, ξi[6,i])
    end
end

println("Done!")
```

```
print("Defining problem... ")

# Hamiltonian
γ0 = get_gamma(ekin, particle.mass)
β0 = get_beta(γ0)

P0 = (γ0*β0*particle.mass) / Constants["clight"]      # Ref momentum
k0 = particle.charge * B / P0                           # Dipole strength

Γ = eval(:(@set_problem("x px y py z δ", $n1, $n2, $n_max)))  # Define problem

hc = 1 + k0*x                                         # Curvature term

# Define Hamiltonian
H = δ / β0 + hc*k0*(x - (k0*x^2)/(2*hc)) - hc*sqrt(
    (1/β0 + δ)^2 - px^2 - py^2 - 1/(β0*γ0)^2
)
tps_expansion!(H)

println("Done!")


print("Computing transfer map... ")
M = lie_transform(-L*H, Γ)                    # Transfer map
println("Done!")


# ========================================================================

# SYSTEM EVOLUTION

println("Computing evolution... ")

ξf = Matrix{Float64}(undef, 6, N)

for i in ProgressBar(1:N)

    ξt = M(ξi[:,i])

    open(final_coord, "a") do io
        for j in 1:5
            print(io, "$(ξt[j]),")
        end
        println(io, ξt[6])
    end

    ξf[:,i] = copy(ξt)
end

println("Done!")


# ========================================================================

# PLOT

print("Plotting... ")

...

println("Done!")
```

# Bibliography

[1] J. E. Humphreys, "Introduction to lie algebras and representation theory," 2012, ISBN: 978-0-387-90052-0.

[2] A. J. Dragt and J. M. Finn, "Lie series and invariant functions for analytic symplectic maps," *Journal of Mathematical Physics*, vol. 17, no. 12, pp. 2215–2227, 1976.

[3] A. J. Dragt, "Lectures on nonlinear orbit dynamics," in *AIP conference proceedings*, vol. 87, pp. 147–313, American Institute of Physics, 1982.

[4] J. D. Meiss, "Symplectic maps," *Encyclopedia of Nonlinear Science, Ed. A. Scott, New York, Rutledge*, 2008.

[5] L. M. Healy and A. J. Dragt, "Concatenation of lie algebraic maps," in *Lie Methods in Optics II: Proceedings of the Second Workshop Held at Cocoyoc, Mexico July 19–22, 1988*, pp. 67–95, Springer, 1989.

[6] F. Willeke, "Modern tools for particle tracking," tech. rep., SCAN-9502027, 1994.

[7] A. J. Dragt and J. M. Finn, "Lie series and invariant functions for analytic symplectic maps," *Journal of Mathematical Physics*, vol. 2215, no. 17, 1976.

[8] A. J. Dragt, *Lie methods for nonlinear dynamics with applications to accelerator physics.* University of Maryland, available at: https://www.physics.umd.edu/dsat/, 2011.

[9] A. Adelmann and S. Li, "Particle accelerator physics and modelling II." Class lecture, ETHZ, Spring semester 2022.

[10] D. McDuff and D. Salamon, *Introduction to symplectic topology*, vol. 27. Oxford University Press, 2017.

[11] J. M. Lee, *Manifolds and Differential Geometry*, vol. 107 of *Graduate Studies in Mathematics.* Providence, RI: American Mathematical Society, 2009.

[12] V. Arnold, *Mathematical Methods of Classical Mechanics*, vol. 60 of *Graduate Texts in Mathematics.* New York, NY: Springer-Verlag, 2nd edition ed., 1989, ISBN: 978-0-387-96890-2, DOI: https://doi.org/10.1007/978-1-4757-1693-1.

[13] A. Adelmann and S. Li, "Particle accelerator physics and modelling I." Class lecture, ETHZ, Fall semester 2021.

[14] K. R. Symon, "Derivation of hamiltonians for accelerators," tech. rep., Argonne National Lab.(ANL), Argonne, IL (United States), 1997.

[15] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics.* Pearson Education Ltd, 3rd edition ed., 2002.

[16] A. C. Kabel, "Maxwell-lorentz equations in general frenet-serret coordinates," in *Proceedings of the 2003 Particle Accelerator Conference*, vol. 4, pp. 2252–2254, IEEE, 2003.

[17] A. Wolski, *Introduction to Beam Dynamics in High-Energy Electron Storage Rings.* Morgan & Claypool Publishers, June 2018, ISBN: 978-1-6817-4989-1, DOI: 10.1088/978-1-6817-4989-1.

[18] A. Deprit, "Canonical transformations depending on a small parameter," *Celestial mechanics*, vol. 1, no. 1, pp. 12–30, 1969.

[19] M. Berz, *Modern Map Methods in Particle Beam Physics*, vol. 108 of *Advances in Imaging and Electron Physics.* Academic Press, 1999, doi: 10.1016/S1076-5670(08)70227-1.

[20] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[21] T. Besard, C. Foket, and B. De Sutter, "Effective extensible programming: Unleashing Julia on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[22] S. Gowda, Y. Ma, A. Cheli, M. Gwóźzdź, V. B. Shah, A. Edelman, and C. Rackauckas, "High-performance symbolic-numerics via multiple dispatch," *ACM Commun. Comput. Algebra*, vol. 55, pp. 92–96, jan 2022.

[23] H. D. Raedt, "Quantum dynamics in nanoscale devices," in *Computational Physics*, pp. 209–224, Springer, 1996.

[24] P. Kammerlander, "Quantum information processing I." Class lecture, ETHZ, Spring semester 2022.

[25] L. Benet and D. P. Sanders, "Taylorseries.jl: Taylor expansions in one and several variables in julia.," *Journal of Open Source Software*, vol. 4(36), pp. 1–4, 2019.

[26] P. V. Koseleff, "Comparison between deprit and dragt-finn perturbation methods," *Celestial Mechanics and Dynamical Astronomy*, vol. 58, pp. 17–36, 1994.

[27] H. Grote and F. Schmidt, "MAD-X: An upgrade from MAD8," *Conf. Proc. C*, vol. 030512, p. 3497, 2003.

[28] C. Rackauckas and Q. Nie, "Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia," *Journal of Open Research Software*, vol. 5, no. 1, p. 15, 2017.

[29] B. Holzer, "Lattice design in high-energy particle accelerators," *CERN Yellow Report CERN-2014-009, pp.61-100*, 2014, arXiv:1601.04913 [physics.acc-ph].

[30] S. Li, "A numerical model for the computation of a quantum free electron laser," Master's thesis, ETH Zürich, Mar. 2018.

[31] I. Lobach, S. Nagaitsev, A. Romanov, and G. Stancari, "Single electron in a storage ring: a probe into the fundamental properties of synchrotron radiation and a powerful diagnostic tool," *Journal of Instrumentation*, vol. 17, p. P02014, feb 2022.

[32] M. Berz and K. Makino, "Verified integration of odes and flows using differential algebraic methods on high-order taylor models," *Reliable computing*, vol. 4, no. 4, pp. 361–369, 1998.

[33] S.-Y. Lee, *Accelerator physics.* World Scientific Publishing Company, 2018.

[34] W. Herr and E. Forest, "Non-linear dynamics in accelerators," *Landolt Börnstein*, vol. 21, p. 38, 2013.

[35] J. R. Rees, "Symplecticity in beam dynamics: An introduction," tech. rep., SLAC National Accelerator Lab., Menlo Park, CA (United States), 2003.

[36] A. Ivanov and I. Agapov, "Physics-based deep neural networks for beam dynamics in charged particle accelerators," *Phys. Rev. Accel. Beams*, vol. 23, p. 074601, Jul 2020.

[37] A. Wolski, "Maxwell's equations for magnets," *CERN-2010-004, pp. 1-38*, June 2009, arXiv:1103.0713 [physics.acc-ph].

[38] F. C. Iselin, "Fast methods for generating lie algebraic maps of arbitrary degree for combined thick multipoles," tech. rep., CERN, 2000.

[39] K. Pakrouski, "Computational quantum physics." Class lecture, ETHZ, spring semester 2022.

[40] B. P. Palka, *An introduction to complex function theory.* Springer Science & Business Media, 1991.

[41] G. Rangarajan, "Symplectic integration of hamiltonian systems using polynomial maps," *Physics Letters A*, vol. 286, no. 2-3, pp. 141–147, 2001.

[42] A. Van-Brunt and M. Visser, "Special-case closed form of the baker–campbell–hausdorff formula," *Journal of Physics A: Mathematical and Theoretical*, vol. 48, p. 225207, may 2015.

## Acknowledgements

I would like to thank Dr. Andreas Adelmann for giving me the opportunity to do my master thesis in the AMAS group at PSI, as well as for supervising it. I also would like to thank him for the careful reading of this paper and his very instructive comments on better scientific writing. lastly, I would like to thank the entire AMAS group for the very valuable advice they shared with me and for the very constructive discussions we had, which were key to my better understanding of problems and to solve the multiple difficulties I came across.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

A Generic Implementation of a Truncated Power Series Algebra in Julia for Beam Dynamics Modeling

**Authored by** (in block letters):
*For papers written by groups the names of all authors are required.*

| **Name(s):** | **First name(s):** |
|---|---|
| MELENNEC | Matthieu Martin |

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| **Place, date** | **Signature(s)** |
|---|---|
| Villigen, 27.03.2023 | [illegible] |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*