Graduate Course: Geometric Numerical Analysis and Deep Learning

Lecture notes, Lent 2025
University of Cambridge

Davide Murari
davidemurari.com

These lecture notes are for the Graduate Course "Geometric Numerical Analysis and Deep Learning", taught by Davide Murari in Lent 2025 at the University of Cambridge.

These notes are based on the following books

1. Ernst Hairer, Christian Lubich, Gerhard Wanner. Geometric Numerical Integration - Structure-Preserving Algorithms for Ordinary Differential Equations. Springer, 2nd Edition.
2. Ernst Hairer, Syvert Nørsett, Gerhard Wanner. Solving Ordinary Differential Equations I, Springer, 2nd Edition.
3. Ernst Hairer, Gerhard Wanner. Solving Ordinary Differential Equations II, Springer, 2nd Edition.
4. Arieh Iserles. A First Course in the Numerical Analysis of Differential Equations. Cambridge University Press, 2008

Some implementations of the presented methods can be found in the associated GitHub repository https://github.com/davidemurari/codePartIIICourse.

The sections marked in red are those involving neural networks. The others are more focused on the numerical analysis and dynamical systems aspects.

This document is in constant redevelopment and might contain typos or errors. I would very much appreciate if you report any mistakes found to dm2011@cam.ac.uk. Thanks!

Contents

1 Recap one-step numerical methods

In this course we consider initial value problems (IVPs) defined by autonomous Ordinary Differential Equations (ODEs) of the form

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t)) \in \mathbb{R}^d \\ \mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^d. \end{cases} \tag{1}$$

We recall that the non-autonomous case can be reduced to this setting by introducing the additional equation $\dot{t} = 1$. We will refer to the vector field $\mathcal{F}$ interchangeably as a function $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ and as a smooth vector field, denoting it as $\mathcal{F} \in \mathfrak{X}(\mathbb{R}^d)$. $\mathcal{F}$ is supposed to be Lipschitz continuous, so that we can guarantee the existence and uniqueness of the solution to (1).

Notation: In this notes, vectors are represented with bold symbols, like $\mathbf{x}$. Matrices are denoted with capital letters. For vector fields and sets we will use calligraphic letters, such as $\mathcal{F}$. To denote the exact flow map of a vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ we will use interchangeably $\phi^t_{\mathcal{F}} : \mathbb{R}^d \to \mathbb{R}^d$ and $\phi_{\mathcal{F}} : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$, where $\phi^t_{\mathcal{F}}(\mathbf{x}) := \phi_{\mathcal{F}}(t, \mathbf{x})$.

Let us consider the time domain $[0, T]$, $T > 0$, and introduce a uniform grid over it defined as $t_i = ih$, $i = 0, ..., N$, $h = T/N$. A one-step numerical method $\varphi^h_{\mathcal{F}} : \mathbb{R}^d \to \mathbb{R}^d$ aims to provide an approximation of the exact flow map $\phi^h_{\mathcal{F}} : \mathbb{R}^d \to \mathbb{R}^d$ of the vector field $\mathcal{F}$ to which it is applied. Whenever it will be clear from the context which vector field we are working with, we will omit the subscript $\mathcal{F}$, and write φ^h.

Definition 1 (Method of order p)**.** *A one-step numerical method $\varphi^h : \mathbb{R}^d \to \mathbb{R}^d$ has order p if, whenever applied to a smooth enough vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$, it satisfies*

$$\varphi^h_{\mathcal{F}} = \phi^h_{\mathcal{F}} + \mathcal{O}(h^{p+1}).$$

1.1 Runge–Kutta methods

The simplest numerical method one can consider is the explicit Euler method, defined as

$$\varphi^h_{\mathcal{F}}(\mathbf{x}) = \mathbf{x} + h\mathcal{F}(\mathbf{x}).$$

By Taylor expanding the exact solution at $t = 0$, we see that

$$\phi^h_{\mathcal{F}}(\mathbf{x}) = \mathbf{x} + h\mathcal{F}(\mathbf{x}) + \mathcal{O}(h^2),$$

hence telling us that the explicit Euler method is first-order accurate. A generalisation of this method is provided by the very popular family of Runge–Kutta methods, with which we will work quite a lot. We provide the definition of these methods for non-autonomous vector fields so it is presented in full generality.

Definition 2 (Runge–Kutta method)**.** *Let us consider the non-autonomous differential equation* $\dot{\mathbf{x}}(t) = \mathcal{F}(t, \mathbf{x}(t))$*. A Runge–Kutta method of* s *stages based on the tableau* $(A, \mathbf{b}, \mathbf{c})$*,* $A \in \mathbb{R}^{s\times s}$*,* $\mathbf{b}, \mathbf{c} \in \mathbb{R}^s$*, is a one-step method defined as*

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_{i=1}^{s} b_i \mathcal{F}(t_n + c_i h, \mathbf{k}_{n,i})$$
$$\mathbf{k}_{n,i} = \mathbf{x}_n + h\sum_{j=1}^{s} a_{ij} \mathcal{F}(t_n + c_j h, \mathbf{k}_{n,j}).$$

We call $\mathbf{k}_{n,i}$*,* $i = 1, ..., s$*, the hidden stages of the method.*

We will generally drop the subscript n in the stages. A Runge–Kutta method is explicit if the matrix A in the tableau is stricly lower triangular. In this case there is no need to solve a non-linear algebraic equation at every step of the method. If the method is implicit, one can approximate the solution of the non-linear algebraic equation associated to one step by using iterative methods like Newton, or quasi-Newton schemes.

The study of the order conditions of Runge–Kutta methods is very well developed, but it is out of the scope of this course. For the interested reader see [14, 28]. In practice, the order conditions will amount to restrictions over the tableau $(A, \mathbf{b}, \mathbf{c})$.

Even though the order conditions for a generic Runge–Kutta method are not simple to derive, we can do that for an important subfamily of these methods: collocation methods. We dedicate the last part of this introduction of one-step methods to collocation methods because they have an interesting interpretation, and they will be perfect examples of structure-preserving numerical methods.

1.2 Collocation methods

Let us now consider the differential equation $\dot{\mathbf{x}} = \mathcal{F}(\mathbf{x})$, and suppose that we have just obtained an approximate solution $\mathbf{x}_n \approx \mathbf{x}(t_n)$. We want to approximate the solution $\mathbf{x}(t)$ at time $t_{n+1} = t_n + h$, so after one time step. To do so, we make the assumption that in-between time $t = t_n$ and $t_{n+1} = t_n + h$ we can approximate the solution with a polynomial of degree s. Let us call $\widetilde{\mathbf{x}} \in \mathbb{P}^s(\mathbb{R})$ this polynomial.

To characterise it, we enforce that $\widetilde{\mathbf{x}}(t_n) = \mathbf{x}_n$, and that

$$\dot{\widetilde{\mathbf{x}}}(t_n + c_i h) = \mathcal{F}(\widetilde{\mathbf{x}}(t_n + c_i h)),\ i = 1, ..., s,$$

for a choice of s distinct numbers $0 \leq c_1 < c_2 < ... < c_s \leq 1$. These $s + 1$ conditions suffice to uniquely characterise a polynomial of degree s. We now write explicitly the form of $\dot{\widetilde{\mathbf{x}}}(t)$:

$$\dot{\widetilde{\mathbf{x}}}(t) = \sum_{i=1}^{s} \mathcal{F}(\widetilde{\mathbf{x}}(t_n + c_i h))\ell_i\left(\frac{t - t_n}{h}\right), \tag{2}$$

where $\ell_i(t) \in \mathbb{P}^{s-1}(\mathbb{R})$ are the elementary Lagrange polynomials such that

$$\ell_i(c_j) = \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

We recall that these polynomials are defined as

$$\ell_i(t) = \prod_{\substack{j=1 \\ j \neq i}}^{s} \frac{t - c_j}{c_i - c_j}.$$

We now integrate both sides of (2) over the interval $[t_n, t_n + c_i h]$ to get

$$\widetilde{\mathbf{x}}(t_n + c_i h) = \mathbf{x}_n + \sum_{j=1}^{s} \mathcal{F}(\widetilde{\mathbf{x}}(t_n + c_j h)) \int_{t_n}^{t_n + c_i h} \ell_j \left(\frac{t - t_n}{h} \right) dt.$$

The change of variables $t = t_n + sh$, allows us to rewrite the condition as

$$\widetilde{\mathbf{x}}(t_n + c_i h) = \mathbf{x}_n + h \sum_{j=1}^{s} \mathcal{F}(\widetilde{\mathbf{x}}(t_n + c_j h)) \int_{0}^{c_i} \ell_j(s) ds,$$

which can be rewritten as

$$\mathbf{k}_i = \mathbf{x}_n + h \sum_{j=1}^{s} a_{ij} \mathcal{F}(\mathbf{k}_j)$$

if we set

$$\mathbf{k}_i := \widetilde{\mathbf{x}}(t_n + c_i h), \ a_{ij} := \int_{0}^{c_i} \ell_j(s) ds.$$

Integrating (2) over $[t_n, t_n + h]$, we instead find

$$\widetilde{\mathbf{x}}(t_n + h) = \mathbf{x}_n + h \sum_{i=1}^{s} b_i \mathcal{F}(\mathbf{k}_i),$$

where we set $b_i := \int_0^1 \ell_i(s) ds$. We thus see that as long as we define $\mathbf{x}_1 = \varphi_{\mathcal{F}}^h(\mathbf{x}_0) := \widetilde{\mathbf{x}}(t_0 + h)$, we recover a one-step method of the Runge–Kutta type with a suitably constrained tableau $(A, \mathbf{b}, \mathbf{c})$.

An important and common choice for the coefficients $c_1, ..., c_s$ is the one provided by Gauss-Legendre quadrature nodes. We call these methods Gauss-Legendre collocation methods.

Before moving to the derivation of the order conditions, let us recall that a quadrature rule based on these collocation points writes

$$\int_{t_0}^{t_0 + h} f(t) dt = h \sum_{i=1}^{s} \omega_i f(a + c_i h) + \mathrm{err}(f),$$
$$|\mathrm{err}(f)| \leq C h^{2s+1} \max_{t \in [t_0, t_0 + h]} |f^{(2s)}(t)|,$$

where $c_1, ..., c_s \in [0,1]$ are the zeros of the $s-$th degree Legendre polynomial, and $\omega_1, ..., \omega_s$ are the weights of the quadrature rule. This quadrature rule is exact for polynomials of degree $2s-1$, i.e., it is of order $2s$. The first three polynomials of such a kind, defined over $[-1,1]$, are

$$P_0(x) = 1,\ P_1(x) = x,\ P_2(x) = \frac{1}{2}(3x^2 - 1),$$

and a way to express them in general is given by Rodigues' formula

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n.$$

Theorem 1 (Gauss-Legendre collocation methods)**.** *The Gauss-Legendre collocation methods based on s collocation nodes are of order $2s$ when applied to a smooth enough vector field $\mathcal{F}: \mathbb{R}^d \to \mathbb{R}^d$.*

To prove this theorem, we need a formula which is very useful also for other estimates. This is the Gröbner-Alekseev formula:

Proposition 1 (Gröbner-Alekseev Formula [14])**.** *Let us consider the two autonomous initial value problems*

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases}, \quad \begin{cases} \dot{\mathbf{y}}(t) = \mathcal{F}(\mathbf{y}(t)) + \mathcal{G}(\mathbf{y}(t)) \\ \mathbf{y}(0) = \mathbf{x}_0 \end{cases}$$

with $\mathcal{F} \in \mathcal{C}^1(\mathbb{R}^d, \mathbb{R}^d)$, and supposing they both admit a unique solution. Then

$$\mathbf{y}(t) - \mathbf{x}(t) = \int_0^t \left.\frac{\partial \phi_{\mathcal{F}}^{t-\tau}(\mathbf{z}_0)}{\partial \mathbf{z}_0}\right|_{\mathbf{z}_0 = \mathbf{y}(\tau)} \mathcal{G}(\mathbf{y}(\tau))d\tau \tag{3}$$

for every $t \geq 0$.

Proof. Since the polynomial approximation $\widetilde{\mathbf{x}}(t)$ is differentiable, it solves the initial value problem

$$\begin{cases} \dot{\widetilde{\mathbf{x}}}(t) = \mathcal{F}(\widetilde{\mathbf{x}}(t)) + \left(\dot{\widetilde{\mathbf{x}}}(t) - \mathcal{F}(\widetilde{\mathbf{x}}(t))\right) \\ \dot{\widetilde{\mathbf{x}}}(t_n) = \mathbf{x}_n \end{cases}$$

over the time interval $[t_n, t_{n+1}]$. Let us consider the initial value problem with the same initial condition $\mathbf{x}_n$ as above, but with the correct vector field $\mathcal{F}$. Call $\mathbf{x}$ the exact solution of this second problem. Using (3) with $\mathcal{G}(\widetilde{\mathbf{x}}(t)) := \dot{\widetilde{\mathbf{x}}}(t) - \mathcal{F}(\widetilde{\mathbf{x}}(t))$, we can say that

$$\begin{aligned} \widetilde{\mathbf{x}}(t_{n+1}) - \mathbf{x}(t_{n+1}) &= \int_{t_n}^{t_{n+1}} \left.\frac{\partial \phi_{\mathcal{F}}^{t_{n+1}-t}(\mathbf{z}_0)}{\partial \mathbf{z}_0}\right|_{\mathbf{z}_0 = \widetilde{\mathbf{x}}(t)} \left(\dot{\widetilde{\mathbf{x}}}(t) - \mathcal{F}(\widetilde{\mathbf{x}}(t))\right) dt \\ &= h\int_0^1 \left.\frac{\partial \phi_{\mathcal{F}}^{t_{n+1}-(t_n+sh)}(\mathbf{z}_0)}{\partial \mathbf{z}_0}\right|_{\mathbf{z}_0 = \widetilde{\mathbf{x}}(t_n+sh)} \left(\dot{\widetilde{\mathbf{x}}}(t_n+sh) - \mathcal{F}(\widetilde{\mathbf{x}}(t_n+sh))\right) ds. \end{aligned}$$

The integral on the right-hand side can be approximated with the Gauss-Legendre quadrature rule associated to the s nodes $c_1, ..., c_s$, so to get

$$\widetilde{\mathbf{x}}(t_{n+1}) - \mathbf{x}(t_{n+1}) = h \underbrace{\sum_{i=1}^{s} \omega_i \left. \frac{\partial \phi_{\mathcal{F}}^{t_{n+1}-t_{n,i}}(\mathbf{z}_0)}{\partial \mathbf{z}_0} \right|_{\mathbf{z}_0 = \widetilde{\mathbf{x}}(t_{n,i})} \mathcal{G}(\widetilde{\mathbf{x}}(t_{n,i}))}_{(\#)} + \mathcal{O}(h^{2s+1}),$$

where $\omega_1, ..., \omega_s$ are the Gauss-Legendre quadrature weights, and $t_{n,i} = t_n + c_i h$. By the characterisation of the collocation polynomial $\widetilde{\mathbf{x}}$ in (2), we see that the term (#) vanishes, and hence we can conclude the proof. □

An example of Gauss-Legendre collocation method is the implicit midpoint method, with tableau

$$A = 1/2, \ b = 1, \ c = 1/2,$$

and which writes

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathcal{F}\left(\frac{\mathbf{x}_n + \mathbf{x}_{n+1}}{2}\right).$$

See [19, Chapter 3] for more details about collocation methods.

1.3 A-stable Runge–Kutta methods

Apart for the order of a Runge–Kutta method, another important aspect to consider when to choose which method to implement is stability. There are several ways to characterise the stability of a method. We now briefly recall the notion of linear stability, or A-stability. In Section 6 we will also discuss the non-linear stability or B-stability property.

Lemma 1 (Stability function of a Runge–Kutta method)**.** *Let us consider the linear test equation $\dot{x}(t) = \lambda x(t)$, $\lambda \in \mathbb{C}$. If we apply the $s-$stage Runge–Kutta method φ^h of tableau $(A, \mathbf{b}, \mathbf{c})$ to it, we get the update $x_{n+1} = R(\lambda h)x_n$ where $R(z) = 1 + z\mathbf{b}^\top (I - zA)^{-1}\mathbf{1}$ is a rational function (quotient of two polynomials) called the stability function of the method.*

We remark that in the above lemma, $\mathbf{1} \in \mathbb{R}^s$ is a vector of all ones.

Proof. We apply the Runge–Kutta method to the test equation to get

$$k_i = x_n + h\lambda \sum_{j=1}^{s} a_{ij} k_j, \ x_{n+1} = x_n + h\lambda \sum_{i=1}^{s} b_i k_i.$$

Let us define

$$\mathbf{k} = \begin{bmatrix} k_1 & \cdots & k_s \end{bmatrix}^\top,$$

so that the conditions above turn into

$$\mathbf{k} = x_n \mathbf{1} + h\lambda A\mathbf{k}, \ x_{n+1} = x_n + h\lambda \mathbf{b}^\top \mathbf{k}. \tag{4}$$

This derivation allows us to conclude by replacing the expression for $\mathbf{k}$ in the second equation:

$$x_{n+1} = x_n + h\lambda x_n \mathbf{b}^\top (I - h\lambda A)^{-1}\mathbf{1} = (1 + h\lambda \mathbf{b}^\top (I - h\lambda A)^{-1}\mathbf{1})x_n =: R(h\lambda)x_n.$$

To show that $R(z)$ is a rational function, we notice that, based on the *Matrix determinant Lemma* [9, Lemma 1.1], $R(z)$ corresponds to the determinant of the rank-one perturbation of the identity

$$I + h\lambda(I - h\lambda A)^{-1}\mathbf{1}\mathbf{b}^\top. \tag{5}$$

The matrix in (5) can be rewritten as

$$(I - h\lambda A)^{-1}(I - h\lambda A + h\lambda \mathbf{1}\mathbf{b}^\top).$$

Thus, we can conclude $R(z)$ is a rational function since

$$R(z) = \frac{\det(I - zA + z\mathbf{1}\mathbf{b}^\top)}{\det(I - zA)}. \tag{6}$$

□

The rewriting (6) of $R(z)$ as a rational function allows us to easily see that for explicit methods $R(z)$ is a polynomial of degree s since $A \in \mathbb{R}^{s\times s}$ is strictly lower triangular and hence $\det(I - zA) = 1$.

A simple, but very important, consequence of Lemma 1, is that for a Runge–Kutta method of order p one has

$$e^z = R(z) + Cz^{p+1} + \mathcal{O}(z^{p+2}) \text{ for } z \to 0$$

since $x_1 = e^{h\lambda}x_0 \approx R(h\lambda)x_0$ and $x(t) = e^{\lambda t}x_0$ is the exact solution. The constant C is usually $\neq 0$. If not, we increase p until it becomes $\neq 0$. This also implies that $R(z)$ provides a rational approximation of the exponential function e^z of order p and error constant C. Further, we can say that for an $s-$stage explicit Runge–Kutta method of order p, one has

$$R(z) = \sum_{i=0}^{p} \frac{z^i}{i!}.$$

Exercise 1. *Show that when the Runge–Kutta method φ^h with tableau $(A, \mathbf{b}, \mathbf{c})$ is applied to the linear differential equation $\dot{\mathbf{x}}(t) = B\mathbf{x}(t)$, with $B \in \mathbb{R}^{d\times d}$, then the update writes $\mathbf{x}_{n+1} = \varphi^h(\mathbf{x}_n) = R(hB)\mathbf{x}_n$, where $R(z)$ is the stability function of the Runge–Kutta method. (Hint: The main variation from the previous proof is that to rewrite the scheme in as a single linear system as in (4), one needs to use Kronecker products.)*

Definition 3 (A-stable Runge–Kutta method)**.** *A Runge–Kutta method φ^h of tableau $(A, \mathbf{b}, \mathbf{c})$ is A-stable if*

$$\mathbb{C}^- := \{z \in \mathbb{C} : \ \mathrm{Re}(z) < 0\} \subseteq \{z \in \mathbb{C} : \ |R(z)| < 1\} =: S.$$

We remark that since the stability function of explicit Runge–Kutta methods is a polynomial, these methods can not be A-stable given that the stability region S must be bounded.

2 What is geometric integration and why do we need it?

Geometric numerical integration focuses on preserving the geometric properties of differential equations when approximating their solutions. These methods are particularly important in the simulation of systems where properties like energy, momentum, or phase-space volume need to be conserved for accurate long-term behaviour.

We have seen in the previous section that Runge–Kutta methods can in principle provide as accurate solutions as we need. In fact, we could either restrict the step size h or increase the order of the method as much as we need to get the desired approximation accuracy. Why do we hence need to look into particular classes of methods if we can get as close as we need to the target solution with "general purpose methods"?

In some applications, for example in molecular dynamics or astrophysics, where one has to make simulations for very long time intervals. In this case, it is not feasible to increase the cost of the time integrator by reducing its step size or increasing its order, since the simulations would become excessively expensive. In other situations, for example when we want to approximate the solutions of a chaotic Hamiltonian system, where trying to accurately follow the solutions for relatively long time intervals is essentially impossible due to the high sensitivity of the solutions to input perturbations. In all these situations, the focus of numerical methods is often moved to reproducing the qualitative properties of the target solution, so to gain long-term stability and interpretability.

To provide an analogy, it is also fair to say that the field of dynamical systems, as started by Poincaré, usually moves its focus to study the qualitative behaviour of the solutions, rather than a quantitative analysis of them. This is a similar change of perspective that distinguishes geometric numerical methods from general purpose numerical methods.

We now motivate even further this need for geometric integrators with a simple example. We consider the simple harmonic oscillator. This system has equations

$$\ddot{q}(t) = -q(t) \iff \begin{bmatrix} \dot{q}(t) \\ \dot{p}(t) \end{bmatrix} = \begin{bmatrix} p(t) \\ -q(t) \end{bmatrix}. \tag{7}$$

Such a system is Hamiltonian, and we will study this class of systems later in Section 5. In particular, this system has the quadratic conserved energy function

$$H(q,p) = \frac{1}{2}\left(q^2 + p^2\right) \tag{8}$$

since

$$\begin{aligned} \frac{d}{dt} H(q(t),p(t)) &= \partial_q H(q(t),p(t))\dot{q}(t) + \partial_p H(q(t),p(t))\dot{p}(t) \\ &= q(t)p(t) - p(t)q(t) = 0. \end{aligned}$$

Let us now consider the explicit Euler method, and analyse how the energy H varies after doing one step with this method. The step writes

$$q_{n+1} = q_n + hp_n,\ p_{n+1} = p_n - hq_n.$$

The updated energy thus takes the form

$$\begin{aligned} E(q_{n+1}, p_{n+1}) &= \frac{1}{2}\left(q_n^2 + p_n^2 + h^2(q_n^2 + p_n^2) + 2hq_np_n - 2hp_nq_n\right) \\ &= E(q_n, p_n) + h^2\, E(q_n, p_n) = (1 + h^2)E(q_n, p_n) > E(q_n, p_n). \end{aligned}$$

Exercise 2. *Repeat this calculation with the implicit Euler method, defined as*

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathcal{F}(\mathbf{x}_{n+1}).$$

In general, for a one-step method φ^h of order p applied to a vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ having a conserved energy $E : \mathbb{R}^d \to \mathbb{R}$ which is sufficiently regular, by simple Taylor expansion we have that

$$\begin{aligned} E(\varphi^h_{\mathcal{F}}(\mathbf{x}_n)) &= E(\phi^h_{\mathcal{F}}(\mathbf{x}_n) + \mathcal{O}(h^{p+1})) = E(\phi^h_{\mathcal{F}}(\mathbf{x}_n)) + \mathcal{O}(h^{p+1}) \\ &= E(\mathbf{x}_n) + \mathcal{O}(h^{p+1}). \end{aligned}$$

However, there are some particular classes of methods that exactly conserve quadratic energy functions like the one in (8). Just to anticipate a method that we will see more in detail in the upcoming sections, the implicit Midpoint method preserves exactly such a conserved energy. We can easily verify it for the simple harmonic oscillator, since it is a linear system. The implicit Midpoint applied to this equation writes

$$q_{n+1} = q_n + h(p_n + p_{n+1})/2,\ p_{n+1} = p_n - h(q_n + q_{n+1})/2.$$

We can rewrite the above update in the vector form

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}\mathbb{J}\mathbf{x}_n + \frac{h}{2}\mathbb{J}\mathbf{x}_{n+1},$$

where

$$\mathbb{J} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

is the canonical symplectic matrix that we will study more in detail in Section 5. This leads to

$$\left(I - \frac{h}{2}\mathbb{J}\right)\mathbf{x}_{n+1} = \left(I + \frac{h}{2}\mathbb{J}\right)\mathbf{x}_n$$

and hence

$$\mathbf{x}_{n+1} = \left(I - \frac{h}{2}\mathbb{J}\right)^{-1}\left(I + \frac{h}{2}\mathbb{J}\right)\mathbf{x}_n =: M\mathbf{x}_n.$$

We now have

$$\|\mathbf{x}_{n+1}\|_2^2 = \mathbf{x}_n^\top M^\top M\mathbf{x}_n$$

where

$$M^\top = \left(I + \frac{h}{2}\mathbb{J}\right)^\top \left(I - \frac{h}{2}\mathbb{J}\right)^{-T} =_{\mathbb{J}^\top = -\mathbb{J}} \left(I - \frac{h}{2}\mathbb{J}\right)\left(I + \frac{h}{2}\mathbb{J}\right)^{-1}.$$

This allows us to conclude that

$$\begin{aligned} M^\top M &= \left(I - \frac{h}{2}\mathbb{J}\right)\left(I + \frac{h}{2}\mathbb{J}\right)^{-1}\left(I - \frac{h}{2}\mathbb{J}\right)^{-1}\left(I + \frac{h}{2}\mathbb{J}\right) \\ &= \left(I - \frac{h}{2}\mathbb{J}\right)\left[\left(I - \frac{h}{2}\mathbb{J}\right)\left(I + \frac{h}{2}\mathbb{J}\right)\right]^{-1}\left(I + \frac{h}{2}\mathbb{J}\right) \\ &=_{\mathbb{J}^2 = -I} \frac{1}{1 + h^2/4}\left(I - \frac{h}{2}\mathbb{J}\right)\left(I + \frac{h}{2}\mathbb{J}\right) \\ &= \frac{1}{1 + h^2/4}\left(I - \frac{h}{2}\mathbb{J} + \frac{h}{2}\mathbb{J} + \frac{h^2}{4}I\right) = I. \end{aligned}$$

We will see that this is an example of a much broader family of methods having similar properties.

Figure 1 shows a comparison of different integrators applied to the equations in (7). As argued above, we see that only the implicit Midpoint method preserves for long time the conserved quantity (8).

3 Deep neural networks and dynamical systems

Neural networks (NNs) are a class of machine learning methods. They can be defined as parametric maps that we denote with $\mathcal{N}_\theta$, depending on a set of parameters θ belonging to some space Θ. The space Θ can be a linear space or a non-linear manifold. Throughout, we will always refer to $\mathcal{N}_\theta$ as a map between two linear spaces, i.e., $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^c$ for some $c, d \in \mathbb{N}$. NNs are generally expressed as the composition $\mathcal{N}_\theta = F_{\theta_L} \circ \cdots \circ F_{\theta_1}$, $\theta = (\theta_1, ..., \theta_L)$, of L parametric maps $F_{\theta_i} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, $i = 1, ..., L$, where $d_0 = d$ and $d_L = c$. L is the number of layers of the network, and it is usual to have the layers F_{θ_i} to be similarly parametrised. More explicitly, these layers tend to consist of linear maps after which a scalar function $\sigma : \mathbb{R} \to \mathbb{R}$, called the *activation function*, is applied to each of the input entries. Common examples of these activation functions are $\mathrm{ReLU}(x) = \max\{0, x\}$, $\mathrm{LeakyReLU}(x) = \max\{ax, x\}$ for $a \in (0, 1)$, the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$, and the hyperbolic tangent $\sigma(x) = \tanh(x)$.

Motivated by the resemblance of these (artificial) neural networks to biological ones, we call *neurons* the components of the vectors obtained while processing the input vector with the network layers. Furthermore, we call *neural network architecture* the parametrisation strategy adopted to design the L layers. Choosing the right architecture for a particular problem is crucial, as it defines the search space in which the approximate solution to the problem will be found. When the number of layers L is larger than two, we refer to $\mathcal{N}_\theta$ as a

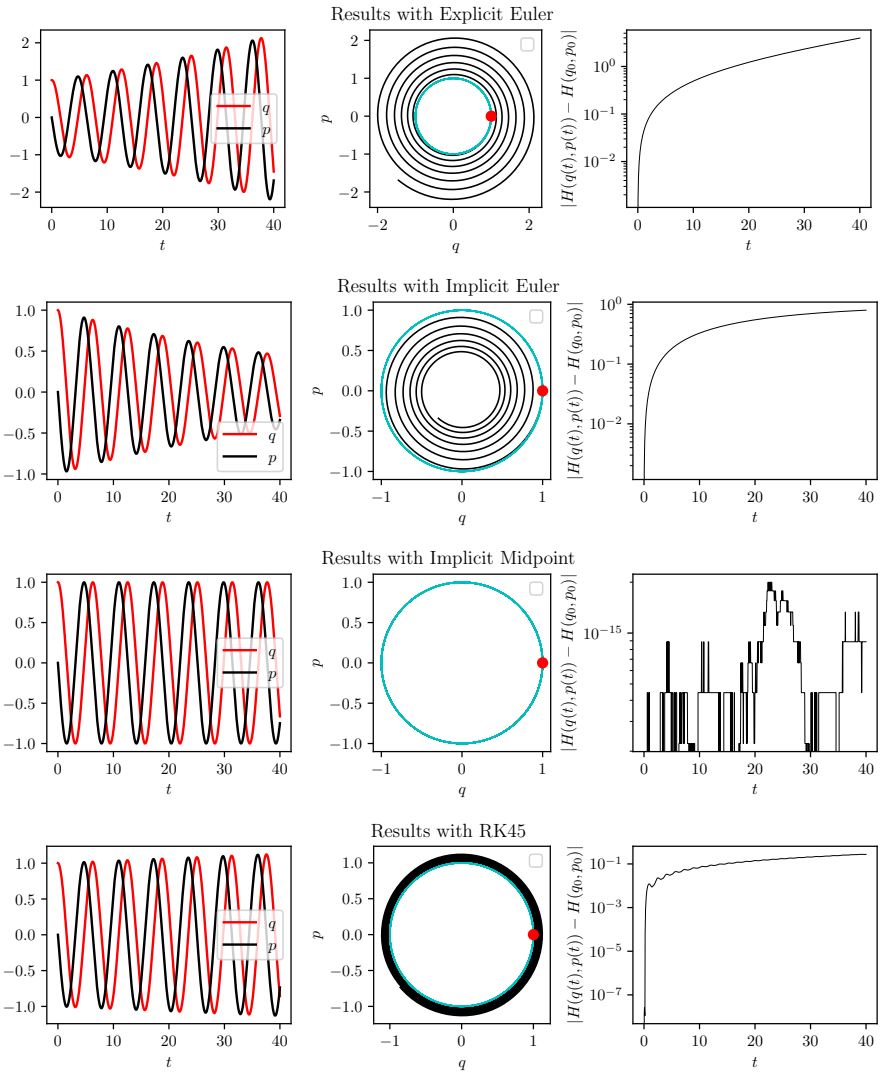

Figure 1: Comparison of four integrators applied to the equation in (7), evaluating if they preserve the conserved quantity (8). The considered initial condition is $\mathbf{x}_0 = (1, 0)$.

deep neural network. *Deep Learning* is the area of machine learning focused on deep networks.

Once a model $\mathcal{N}_\theta$ is chosen, i.e., an architecture is fixed, one needs to find a good set of parameters θ that allow for $\mathcal{N}_\theta$ to solve sufficiently accurately the task of interest. The selection of the parameters is generally the result of the approximate solution of an optimisation problem where a cost function, more commonly called *loss function*, is minimised. This phase is called *neural network training*.

The loss function can combine multiple terms that might or might not depend on data. If the loss depends on data, we refer to the learning task as *supervised learning*, otherwise as *semi-supervised* or *unsupervised learning*. We will see both supervised and unsupervised examples throughout this course. The loss function for each considered problem will be introduced when the problems are discussed. Just to show a quite common example, let us consider the setting in which we are interested in approximating an unknown target function $F : \mathbb{R}^d \to \mathbb{R}^c$ of which we only have available some samples. The samples can be organised into the training set $\{(\mathbf{x}_i, \mathbf{y}_i = F(\mathbf{x}_i))\}_{i=1,...,N}$. In this situation, we define a generic neural network $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^c$, without focusing on the architecture for now, and we introduce the mean-squared error (MSE) loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{N}_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2 .$$

Just to fix the ideas, to find a good set of weights $\theta^* \in \Theta \subset \mathbb{R}^p$, we can proceed with the simplest iterative method to minimise $\mathcal{L}(\theta)$, i.e. gradient descent. This method is based on the iteration

$$\theta_0 \leftarrow \text{Initial guess / Random initialisation}, \tag{9}$$

$$\theta_{k+1} \leftarrow \theta_k - \tau_k \nabla \mathcal{L}(\theta_k), \ k = 0, ..., N_{\text{epochs}}. \tag{10}$$

We set $\theta^* = \theta_{N_{\text{epochs}}}$, and the trained model $\mathcal{N}_{\theta^*}$ can then be used to approximate F for unseen inputs. The gradient descent algorithm is not used very often by practitioners, but more efficient variants like Stochastic Gradient Descent (SGD) or Adam are usually preferred. The problem

$$\min_{\theta \in \Theta} \mathcal{L}(\theta)$$

is generally a highly non-convex and high-dimensional optimisation problem to solve. For this reason, it is also quite hard to mathematically analyse the optimisation process, and for example determine how to initialise the weights or which method to use. Some researchers are using dynamical systems theory to get insights into this aspect of deep learning, thinking for example to the iterates in (10) as steps with the explicit Euler method applied to the negative gradient flow $\dot{\theta}(t) = -\nabla \mathcal{L}(\theta)$. This is the first connection we can highlight between deep learning and dynamical systems.

Another very relevant connection between dynamical systems and deep learning can be found in the process of designing new neural network architectures.

In fact, we will now see that the layers of some architectures can be associated to steps of a numerical method applied to a specific differential equation. We will also see how this intuition allows us to build neural networks with some desired structure, i.e., networks which by construction satisfy some desired property. More explicitly, we will see in Section 5 how to build symplectic neural networks, and in Section 6 how to constrain the Lipschitz constant of a neural network. Both these constrained networks will be inspired by dynamical systems and coming from the use of geometric numerical methods suitably used to approximate their solutions.

Among the broad range of NN architectures, we will focus on *Residual Neural Networks* (ResNets). ResNets are based on layers of the form

$$F_{\theta_i}(\mathbf{x}) = \mathbf{x} + \mathcal{F}_{\theta_i}(\mathbf{x}), \tag{11}$$

where $\mathcal{F}_{\theta_i}$ is a parametric function preserving the dimension of the input $\mathbf{x}$. In other words, ResNets rely on layers where the parameters come into play only in the residual term $\mathcal{F}_{\theta_i}(\mathbf{x}) = F_{\theta_i}(\mathbf{x}) - \mathbf{x}$. Due to this restriction on the updates, one layer of a ResNet has coinciding input and output dimensions. This way of processing the input has been introduced in [16] to overcome the drawbacks of feedforward neural networks, for which deeper networks generally tend to lead to worse performance, contrary to what one would expect.

ResNets can be closely connected with discretisations of initial value problems. Indeed, one can see the map in (11) as one step of the explicit Euler method with step size equal to 1 applied to the non-autonomous differential equation

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathcal{F}(\mathbf{y}(t), \theta(t)) \\ \mathbf{y}(t_i) = \mathbf{x} \end{cases}, \tag{12}$$

where $\mathcal{F}_{\theta_i}(\cdot) = \mathcal{F}(\cdot, \theta(t_i))$, and $\dot{\mathbf{y}}(t) := \frac{d}{dt}\mathbf{y}(t)$. This connection between differential equations and neural network architectures allows us to borrow from the fields of dynamical systems and numerical analysis to model neural networks behaving as desired. More explicitly, we can generalise this intuition and consider ResNet-like architectures defined as

$$\mathcal{N}_\theta = \varphi^{h_L}_{\mathcal{F}_{\theta_L}} \circ \cdots \circ \varphi^{h_1}_{\mathcal{F}_{\theta_1}},$$

where $\varphi^h_{\mathcal{F}}$ is a numerical method we can choose and the vector fields $\mathcal{F}_{\theta_1}, ..., \mathcal{F}_{\theta_L} : \mathbb{R}^d \to \mathbb{R}^d$ are parametric vector fields we can also choose.

Depending on the choice of the numerical method $\varphi^h_{\mathcal{F}} : \mathbb{R}^d \to \mathbb{R}^d$ and how $\mathcal{F}$ is modelled, it is possible to obtain a different neural network architecture. Furthermore, given that the dynamical system in (12) is only of interest for modelling purposes, one can assume without loss of generality that $\theta : \mathbb{R} \to \Theta$ is piecewise-constant in time. This assumption leads to a piecewise autonomous time-switching system.

Neural networks are parametric functions that aim to approximate an unknown target map $F : \mathbb{R}^d \to \mathbb{R}^c$. When a property is known to be satisfied

by F, one may be interested in ensuring that the neural network also satisfies that property. An example is a neural network approximating the solution of a differential equation which is known to conserve the linear energy function $I(\mathbf{x}) = \mathbf{1}^\top \mathbf{x}$, where $\mathbf{1} \in \mathbb{R}^d$ is a vector of ones. In that case, to increase the interpretability of the approximation, the network should reproduce this conservation property. We will see one way to get this in Section 4.

Furthermore, it might be that one does not know if the target function has a peculiar structure while still being interested in approximating it with a constrained map. This situation occurs, for example, when building neural networks that can classify images while being robust to input perturbations. In this case, it is not known how the perfect classifier behaves outside of the training set of input-output labelled pairs. However, in Section 6, we show that it is beneficial to model the neural network so that it has a small Lipschitz constant, i.e., a reduced sensitivity to input perturbations.

We now introduce a methodology based on geometric integration to impose a specific property over networks, i.e., to obtain structured neural networks. We suppose this property is closed under function compositions as, for example, when considering symplectic functions, functions from a manifold $\mathcal{M}$ onto itself, volume-preserving maps, or functions with a Lipschitz constant smaller than 1. The first step is to design a family of parametric vector fields $\mathcal{S}_\Theta$ whose solutions satisfy the target property. Here are a few examples:

1. For the preservation of a linear energy function $I : \mathbb{R}^d \to \mathbb{R}$, one can adopt the skew-gradient formulation

$$\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = \left(A_\theta\left(\mathbf{x}(t)\right) - A_\theta\left(\mathbf{x}(t)\right)^\top\right) \nabla I\left(\mathbf{x}(t)\right),$$

   see [24], where A_θ is a matrix-valued neural network, see Section 4.

2. For symplectic neural networks, one can work with parametric Hamiltonian systems like $\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = \mathbb{J}\nabla H_\theta\left(\mathbf{x}(t)\right) \in \mathbb{R}^{2d}$, for example by setting $H_\theta\left(\mathbf{x}\right) = \mathbf{c}^\top \sigma\left(A\mathbf{x} + \mathbf{b}\right)$, $A \in \mathbb{R}^{h \times 2d}$, $\mathbf{b} \in \mathbb{R}^h$, and $\mathbf{c} \in \mathbb{R}^h$, with $\theta = (A, \mathbf{b}, \mathbf{c})$, see Section 5.

3. For the 1−Lipschitz property, one can use contractive dynamical systems like $\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = -A^\top \sigma\left(A\mathbf{x}(t) + \mathbf{b}\right)$, $\theta = (A, \mathbf{b})$, see Section 6.

Properly designing the vector fields is not enough. Indeed, one also needs to choose a numerical method $\varphi^h_{\mathcal{F}_\theta}$ which reproduces the desired structure at a discrete level. We will see how the numerical integrators $\varphi^h_{\mathcal{F}_\theta}$ discussed in the course can be beneficial in this context.

The parametric set $\mathcal{S}_\Theta$ should be chosen so that its elements enable the design of a network that accurately solves the task at hand. While for more conventional neural networks many approximation theorems have been developed [7, 15, 18, 20, 23], less is known for more structured ones. There is ongoing research in this area, but it goes out of the scope of this course.

For illustration purposes, and also to mention the third connection between deep learning and dynamical systems, which is illustrated in Figure 2. This

figure shows the results of a data-driven modelling task. These tasks generally aim to approximate the flow map or the vector field of an unknown differential equation based on available (observed) trajectory data and possibly some additional knowledge on the dynamics. To motivate the study of structured neural networks, this experiment compares a symplectic neural network built following this dynamical systems-based approach to an unconstrained ResNet. We train the two networks, so they approximate the flow map $\phi_{\mathcal{F}}^{0.1}$ of the simple harmonic oscillator, a planar Hamiltonian system with Hamiltonian $H(\mathbf{x}) = \|\mathbf{x}\|_2^2/2$.

Such a dynamical system conserves the quadratic Hamiltonian function. The constrained network, being symplectic, almost replicates this conservation law, see Figure 2, leading to a much more stable long-term behaviour than with the unconstrained ResNet.

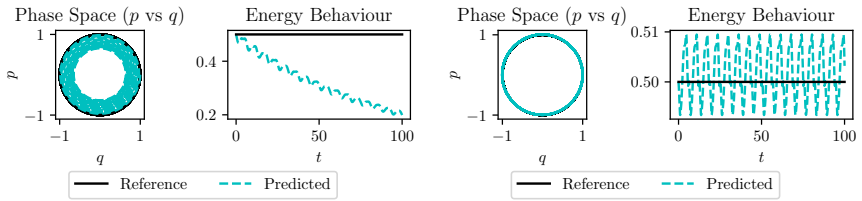

Figure 2: Comparison of the approximate solutions obtained with an unconstrained ResNet (left) and a symplectic neural network (right).

4 Methods conserving first integrals

We can say that there are two classes of one-step numerical methods preserving first integrals (conservation laws): methods that preserve a broad class of first integrals without having to know them, and methods that require the knowledge of the first integral to preserve.

We now briefly introduce what is a differential equation with a first integral. We then move to Runge–Kutta methods that preserve polynomial first integrals. After showing a negative result on the existence of Runge–Kutta methods preserving all first integrals of degree higher or equal than three, we will introduce two other classes of methods that are built to preserve any targeted first integral: projection and discrete gradient methods.

4.1 ODEs with a first integral

Definition 4 (First integral)**.** *A system of differential equations* $\dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t))$, $\mathcal{F} \in \mathfrak{X}(\mathbb{R}^d)$*, admits a first integral* $I : \mathbb{R}^d \to \mathbb{R}$ *if and only if the function* I *is constant along the solutions of the differential equation, i.e.,* $I(\phi_{\mathcal{F}}^t(\mathbf{x}_0)) = I(\mathbf{x}_0)$ *for any* $t \geq 0$ *and* $\mathbf{x}_0 \in \mathbb{R}^d$*. If* I *is continuously differentiable, we can*

equivalently characterise a first integral by the condition

$$\frac{d}{dt}I(\phi_{\mathcal{F}}^t(\mathbf{x}_0)) = \nabla I(\phi_{\mathcal{F}}^t(\mathbf{x}_0)) \cdot \frac{d}{dt}\phi_{\mathcal{F}}^t(\mathbf{x}_0) = \nabla I(\phi_{\mathcal{F}}^t(\mathbf{x}_0)) \cdot \mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{x}_0)) = 0$$

for every $t \geq 0$ *and* $\mathbf{x}_0$.

There are several systems admitting a first integral and they can be written in the form

$$\dot{\mathbf{x}}(t) = (A(\mathbf{x}(t)) - A(\mathbf{x}(t))^\top)\nabla I(\mathbf{x}(t)), \tag{13}$$

since

$$\mathcal{F}(\mathbf{x}(t)) = \frac{\mathcal{F}(\mathbf{x}(t))\nabla I(\mathbf{x}(t))^\top - \nabla I(\mathbf{x}(t))\mathcal{F}(\mathbf{x}(t))^\top}{\|\nabla I(\mathbf{x}(t))\|_2^2}\nabla I(\mathbf{x}(t)).$$

For these systems, the level sets $\mathcal{I}_c = \{\mathbf{x} \in \mathbb{R}^d : \ I(\mathbf{x}) = c\}$ are invariant with respect to the flow map $\phi_{\mathcal{F}}^t$, and it is sometimes desirable to have the same property also at a discrete level. A reason why this could be interesting is for stability/boundedness purposes, since if the level sets of I are compact and they are numerically preserved, the discrete solution will also remain bounded. We will also discuss in more detail a particular class of systems with a first integral, which are Hamiltonian systems, in Section 5.

4.2 Polynomial invariants

4.2.1 Linear invariants

Theorem 2 (Conservation of linear first integrals)**.** *All explicit and implicit Runge–Kutta methods conserve linear invariants.*

Proof. We focus on the differential equation $\dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t))$, and suppose it admits the linear first integral $I(\mathbf{x}) = \mathbf{v}^\top\mathbf{x}$, $\mathbf{v} \in \mathbb{R}^d$. In other words, we suppose that $\mathbf{v}^\top\mathcal{F}(\mathbf{x}) = 0$ for every $\mathbf{x} \in \mathbb{R}^d$.

Let us define the Runge–Kutta stages

$$\mathbf{k}_i = \mathbf{x}_0 + h\sum_{j=1}^{s} a_{ij}\mathcal{F}(\mathbf{k}_j),$$

and the associated one-step update

$$\mathbf{x}_1 = \mathbf{x}_0 + h\sum_{i=1}^{s} b_i\mathcal{F}(\mathbf{k}_i).$$

The proof can be concluded by the following derivation:

$$I(\mathbf{x}_1) = \mathbf{v}^\top\mathbf{x}_0 + h\sum_{i=1}^{s} b_i\mathbf{v}^\top\mathcal{F}(\mathbf{k}_i) = \mathbf{v}^\top\mathbf{x}_0 = I(\mathbf{x}_0).$$

□

Exercise 3. *Implement the Runge–Kutta method you prefer, and verify that when applied to the equations of the SIR model*

$$\begin{cases} \dot{S}(t) = -\beta I(t)S(t), \\ \dot{I}(t) = \beta I(t)S(t) - \gamma I(t), \\ \dot{R}(t) = \gamma I(t), \end{cases}$$

it preserves the linear invariant $E(S, I, R) = S + I + R$.

4.2.2 Quadratic invariants

We now focus on quadratic functions of the form $Q(\mathbf{x}) = \mathbf{x}^\top C\mathbf{x}$ with $C \in \mathbb{R}^{d\times d}$ symmetric. This is a generic enough quadratic function since if we manage to conserve it, we will also be able to conserve something like $\tilde{Q}(\mathbf{x}) = \mathbf{x}^\top C\mathbf{x}+\mathbf{v}^\top\mathbf{x}$ by linearity and Theorem 2.

Remark 1. *The theorem and proof we are going to see now are very important since we will use similar techniques and definitions also for the subsequent topics.*

Theorem 3 (Conservation of quadratic first integrals)**.** *Let us consider a Runge–Kutta method with tableau determined by the triple* $(A, \mathbf{b}, \mathbf{c})$. *Let us define the matrices* $B = \operatorname{diag}(\mathbf{b})$ *and* $M = BA + A^\top B - \mathbf{b}\mathbf{b}^\top$. *If* $M = 0_{s\times s}$ *then the Runge–Kutta method under consideration conserves quadratic first integrals.*

Before moving to the proof, we explicitly write down the entry m_{ij} of the matrix $M \in \mathbb{R}^{s\times s}$:

$$m_{ij} = b_i a_{ij} + a_{ji} b_j - b_i b_j.$$

Proof. Let us focus on the differential equation $\dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t))$ and suppose it admits the first integral $Q(\mathbf{x}) = \mathbf{x}^\top C\mathbf{x}$, i.e., that $\mathbf{x}^\top C\mathcal{F}(\mathbf{x}) = 0$ for every $\mathbf{x} \in \mathbb{R}^d$. The hidden stages of the Runge–Kutta method under consideration write

$$\mathbf{k}_i = \mathbf{x}_0 + h\sum_{j=1}^{s} a_{ij}\mathcal{F}(\mathbf{k}_j), \tag{14}$$

leading to the update $\mathbf{x}_1 = \mathbf{x}_0 + h\sum_{i=1}^{s} b_i\mathcal{F}(\mathbf{k}_i)$. We now expand $Q(\mathbf{x}_1)$ to get

$$Q(\mathbf{x}_1) = \mathbf{x}_1^\top C\mathbf{x}_1 = Q(\mathbf{x}_0) + h^2\sum_{i,j=1}^{s} b_i b_j \mathcal{F}(\mathbf{k}_i)^\top C\mathcal{F}(\mathbf{k}_j) + 2h\sum_{i=1}^{s} b_i \mathbf{x}_0^\top C\mathcal{F}(\mathbf{k}_i). \tag{15}$$

We now notice that, from (14), we can express $\mathbf{x}_0$ in s different ways as

$$\mathbf{x}_0 = \mathbf{k}_i - h\sum_{j=1}^{s} a_{ij}\mathcal{F}(\mathbf{k}_j).$$

We replace this i-dependent rewriting in the last term of (15), to get

$$
\begin{aligned}
Q(\mathbf{x}_1) = \mathbf{x}_1^\top C \mathbf{x}_1 &= Q(\mathbf{x}_0) + h^2 \sum_{i,j=1}^{s} b_i b_j \mathcal{F}(\mathbf{k}_i)^\top C \mathcal{F}(\mathbf{k}_j) \\
&+ \underbrace{2h \sum_{i=1}^{s} b_i \mathbf{k}_i^\top C \mathcal{F}(\mathbf{k}_i)}_{=0} - 2h^2 \sum_{i,j=1}^{s} b_i a_{ij} \mathcal{F}(\mathbf{k}_j)^\top C \mathcal{F}(\mathbf{k}_i) \\
&= Q(\mathbf{x}_0) + h^2 \sum_{i,j=1}^{s} b_i b_j \mathcal{F}(\mathbf{k}_i)^\top C \mathcal{F}(\mathbf{k}_j) - h^2 \sum_{i,j=1}^{s} b_i a_{ij} \mathcal{F}(\mathbf{k}_j)^\top C \mathcal{F}(\mathbf{k}_i) \\
&- h^2 \sum_{i,j=1}^{s} b_j a_{ji} \mathcal{F}(\mathbf{k}_i)^\top C \mathcal{F}(\mathbf{k}_j) \\
&= Q(\mathbf{x}_0) - h^2 \sum_{i,j=1}^{s} m_{ij} \mathcal{F}(\mathbf{k}_i)^\top C \mathcal{F}(\mathbf{k}_j) = Q(\mathbf{x}_0)
\end{aligned}
$$

if $m_{ij} = 0$ for every $i, j \in \{1, ..., s\}$. □

Example 1. *An example of a Runge–Kutta method which preserves quadratic first integrals is the implicit mid-point method, which has tableau*

$$A = 1/2, \quad b = 1, \quad c = 1/2,$$

and writes

$$\mathbf{x}_1 = \mathbf{x}_0 + h\mathcal{F}\left(\frac{\mathbf{x}_0 + \mathbf{x}_1}{2}\right).$$

In this case, in fact, one has

$$M = BA + A^\top B - \mathbf{b}\mathbf{b}^\top = 1/2 + 1/2 - 1 = 0.$$

The implicit mid-point is a particular case of a more general family of methods preserving quadratic first integrals, as we argue in the following proposition.

Proposition 2. *All Gauss-Legendre collocation methods preserve quadratic first integrals.*

Proof. The proof does not aim to show that these methods satisfy the assumptions of Theorem 3, but we use the main properties of collocation methods.

Let $\mathbf{u}(t)$ be the polynomial approximation of the solution provided by the collocation method over the time interval $[t_n, t_{n+1} = t_n + h]$. Let us consider the function $q(t) := Q(\mathbf{u}(t)) = \mathbf{u}(t)^\top C \mathbf{u}(t)$, which has time derivative

$$\frac{d}{dt} q(t) = 2\dot{\mathbf{u}}(t)^\top C \mathbf{u}(t).$$

We thus have

$$q(t_{n+1}) = q(t_n) + \int_{t_n}^{t_n+h} \dot{q}(t)dt \tag{16}$$

$$= q(t_n) + 2h \int_0^1 \dot{\mathbf{u}}(t_n + sh)^\top C\mathbf{u}(t_n + sh)ds \tag{17}$$

Recalling that Gaussian quadrature rules based on s quadrature nodes are exact for polynomials up to degree $2s-1$, and that $\mathbf{u}(t)$ is of degree s, we conclude that the integral in (17) can be exactly computed by Gaussian quadrature since $\dot{\mathbf{u}}(t)^\top C\mathbf{u}(t) \in \mathbb{P}^{2s-1}(\mathbb{R})$. Thus, we get

$$\begin{aligned} q(t_{n+1}) - q(t_n) &= 2h \sum_{i=1}^{s} \dot{\mathbf{u}}(t_0 + c_i h)^\top C\mathbf{u}(t_0 + c_i h) \\ &= 2h \sum_{i=1}^{s} \mathcal{F}(\mathbf{u}(t_0 + c_i h))^\top C\mathbf{u}(t_0 + c_i h) = 0, \end{aligned}$$

which allows us to conclude the proof since $q(t_{n+1}) = Q(\mathbf{x}_{n+1})$ and $q(t_n) = Q(\mathbf{x}_n)$. □

Example 2 (Free rigid body)**.** *An interesting example in this case is provided by the equations of motion of a free rigid body, which can be written as*

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & \mathbf{x}_3/I_3 & -\mathbf{x}_2/I_2 \\ -\mathbf{x}_3/I_3 & 0 & \mathbf{x}_1/I_1 \\ \mathbf{x}_2/I_2 & -\mathbf{x}_1/I_1 & 0 \end{bmatrix} \mathbf{x}(t),$$

and hence have the quadratic first integral $Q(\mathbf{x}) = \|\mathbf{x}\|_2^2$. They also conserve another first integral which is still quadratic: $\tilde{Q}(\mathbf{x}) = \mathbf{x}^\top I^{-1}\mathbf{x}$ where $I = \mathrm{diag}(I_1, I_2, I_3) \in \mathbb{R}^{3\times 3}$.

Exercise: *Approximate the solution of a free rigid body by using the implicit Midpoint method and verify that the two first integrals are both preserved. What does this imply?*

4.3 Higher-degree polynomials

PROBLEM: We now show that, for $n \geq 3$, there is no Runge–Kutta method that can conserve all the polynomial invariants of degree n. We will thus need to look for other families of numerical methods to preserve them and also generic non-polynomial first integrals.

To prove this result, we need a few preliminary lemmas.

Lemma 2. *If* $\mathrm{trace}(B(Y)) = 0$ *for every* $Y \in \mathbb{R}^{d\times d}$*, then* $g(Y) = \det(Y)$ *is a first integral of the matrix differential equation* $\dot{Y} = B(Y)Y$*,* $B(Y), Y \in \mathbb{R}^{d\times d}$*.*

Proof. We first notice that, taken $h \ll 1$, it follows

$$\begin{aligned}\det(Y + hBY) &= \det((I + hB)Y) = \det(I + hB)\det(Y) \\ &= (1 + h\text{trace}(B) + \mathcal{O}(h^2))\det(Y).\end{aligned}$$

Thus,

$$\begin{aligned}\frac{d}{dt}g(Y(t)) &= \lim_{h\to 0} \frac{\det(Y(t+h)) - \det(Y(t))}{h} \\ &= \lim_{h\to 0} \frac{\det(Y(t) + hB(Y(t))Y(t)) - \det(Y(t))}{h} \\ &= \text{trace}(B(Y(t))\det(Y(t)),\end{aligned}$$

or, more synthetically, $g'(Y)(BY) = \text{trace}(B)\det(Y)$. This derivation implies that $g(Y) = \det(Y)$ is a first integral of the differential equation if $\text{trace}(B(Y)) = 0$ for all Y. □

We recall that for a matrix $Y \in \mathbb{R}^{d\times d}$, $g(Y) = \det(Y)$ is a polynomial function of degree d in the entries of Y. We will now show that no Runge–Kutta method can conserve such a first integral for $d \geq 3$.

Lemma 3. *Let $R(z)$ be a differentiable function defined in a neighbourhood of $z = 0$, and assume that $R(0) = 1$ and $R'(0) = 1$. Then, for $d \geq 3$, $\det(R(B)) = 1$ for all $B \in \mathbb{R}^{d\times d}$ with $\text{trace}(B) = 0$ if and only if $R(z) = e^z$.*

Proof. Let us first assume $R(z) = e^z$. Then $R(tB) = \exp(tB)$ is the exact time$-t$ solution of the initial value problem $\dot{Y}(t) = BY(t)$, $Y(0) = I$. Thus, if $\text{trace}(B) = 0$ it follows by the previous lemma that

$$\det(R(B)) = \det(Y(1)) = \det(Y(0)) = \det(I) = 1.$$

For the *only if* part, we consider matrices of the form

$$B = \text{diag}(\mu, \nu, -(\mu + \nu), 0, ..., 0),$$

with μ, ν close enough to zero. In this way, we have $\text{trace}(B) = 0$. Applying $R(z)$ we get

$$R(A) = \text{diag}(R(\mu), R(\nu), R(-(\mu + \nu)), R(0), ..., R(0)).$$

Assuming that $\det(R(B)) = 1$ implies that

$$R(\mu)R(\nu)R(-(\mu + \nu)) = 1$$

for all μ, ν close to zero. We also notice that $R(-\mu) = 1/R(\mu)$, simply by setting $\nu = 0$. Thus,

$$R(\mu)R(\nu) = R(\mu + \nu).$$

Let us now exploit this derivation to see that

$$\frac{R(\mu + h) - R(\mu)}{h} = \frac{R(\mu)R(h) - R(\mu)}{h} = R(\mu)\frac{R(h) - 1}{h}$$

for μ and h sufficiently close to zero. In conclusion, if we take the limit as $h \to 0$, we can conclude that

$$R'(\mu) = \lim_{h\to 0} \frac{R(\mu+h) - R(\mu)}{h} = R(\mu)R'(0) = R(\mu)$$

for every μ sufficiently close to zero. This implies $R(\mu) = e^{\mu}$ as desired. □

Lemma 4 (Solution to Exercise 1)**.** *Let us consider the linear ODE $\dot{\mathbf{x}}(t) = B\mathbf{x}(t)$, $B \in \mathbb{R}^{d\times d}$, and the s-stages Runge–Kutta method of tableau $(A, \mathbf{b}, \mathbf{c})$. Then, the update map $\varphi^h(\mathbf{x})$ is defined as $\varphi^h(\mathbf{x}) = R(hB)\mathbf{x}$, where R is the stability function of the method.*

Proof. We first write in vector form the update. We have

$$\mathbf{x}_{n+1} = \varphi^h(\mathbf{x}_n) = \mathbf{x}_n + h\sum_{i=1}^{s} b_i B\mathbf{k}_i,\ \ \mathbf{k}_i = \mathbf{x}_n + h\sum_{j=1}^{s} a_{ij} B\mathbf{k}_j.$$

We now introduce the vector

$$\mathbf{k} = \begin{bmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_s \end{bmatrix} \in \mathbb{R}^{ds},$$

which allows us to express the update map φ^h as follows

$$(I_{ds} - hA\otimes B)\mathbf{k} = \mathbf{1}_s \otimes \mathbf{x}_n \implies \mathbf{k} = (I_{ds} - hA\otimes B)^{-1}\mathbf{1}_s \otimes \mathbf{x}_n,$$
$$\mathbf{x}_{n+1} = \mathbf{x}_n + h(\mathbf{b}^\top \otimes B)\mathbf{k} = \mathbf{x}_n + h(\mathbf{b}^\top \otimes B)(I_{ds} - hA\otimes B)^{-1}\mathbf{1}_s \otimes \mathbf{x}_n$$

First of all, we notice that in the scalar case, we can also write

$$R(z) = 1 + (\mathbf{b}^\top \otimes z)(I_s - z\otimes A)^{-1}\mathbf{1}_s,$$

showing that the expression above can be seen as $R(hB)$. To show this extension makes sense, let us suppose $h\|A\otimes B\| < 1$ for some sub-multiplicative norm $\|\cdot\|$. We can thus simplify this expression by using the series expansion

$$(I_{ds} - hA\otimes B)^{-1} = \sum_{k\geq 0} h^k (A\otimes B)^k = \sum_{k\geq 0} h^k A^k \otimes B^k,$$

which, coupled with the property $(M_1\otimes N_1)(M_2\otimes N_2) = (M_1M_2)\otimes(N_1N_2)$, leads to

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \sum_{k\geq 0} h^{k+1} \underbrace{(\mathbf{b}^\top A^k \mathbf{1}_s)}_{\in\mathbb{R}} \otimes B^{k+1}\mathbf{x}_n = \left(1 + \sum_{k\geq 0} (\mathbf{b}^\top A^k \mathbf{1}_s) h^{k+1} B^{k+1}\right)\mathbf{x}_n.$$

We can now conclude by noticing that, in the scalar case, if $|z| < 1$, one has

$$R(z) = 1 + z\mathbf{b}^\top (I_s - zA)^{-1}\mathbf{1}_s = 1 + z\mathbf{b}^\top \sum_{k\geq 0} z^k A^k \mathbf{1}_s = 1 + \sum_{k\geq 0} (\mathbf{b}^\top A^k \mathbf{1}_s) z^{k+1},$$

and hence $\mathbf{x}_{n+1} = R(hB)\mathbf{x}_n$ as desired. □

Theorem 4. *For $d \geq 3$, no Runge–Kutta method can conserve all polynomial first integrals of degree d.*

Proof. Let us consider the linear matrix differential equation $\dot{Y} = BY$, $Y \in \mathbb{R}^{d\times d}$, where $\operatorname{trace}(B) = 0$. Based on Lemma 2, we know that $g(Y) = \det(Y)$ is a degree d polynomial first integral for the system. If we apply the Runge–Kutta method of tableau $(A, \mathbf{b}, \mathbf{c})$ to this system, we get

$$Y_1 = R(hB)Y_0$$

where $R(z) = 1 + z\mathbf{b}^\top (I - zA)^{-1}\mathbf{1}$. By Lemma 3 we conclude that it is not possible to have $\det(R(hB)) = 1$ for all B with $\operatorname{trace}(B) = 0$ because otherwise we would have $R(z) = e^z$, which is never true. Thus, there is a choice of B for which $\det(Y_1) \neq \det(Y_0)$. □

4.4 Methods preserving generic first integrals

The methods we are going to see now belong two classes: projection methods and discrete gradient methods. Both of these families of methods rely on the knowledge of the first integral to be preserved, and the method is built in such a way that we can preserve the level sets of this target function.

4.4.1 Projection methods

We can reformulate the problem of preserving the invariant function $I : \mathbb{R}^d \to \mathbb{R}$ into something much more generic. Conserving a first integral indeed means preserving its level sets. More precisely, let us suppose to start from the initial condition $\mathbf{x}_0$, then what we really want to do is ensure that the updated positions we get starting there remain on the (non-linear) manifold

$$\mathcal{M}_{\mathbf{x}_0} := \left\{\mathbf{x} \in \mathbb{R}^d : \ g(\mathbf{x}) := I(\mathbf{x}) - I(\mathbf{x}_0) = 0\right\} \subset \mathbb{R}^d,$$

which has dimension $d - 1$. We can thus interpret this desired result as a constraint over the numerical method.

After one step starting at $\mathbf{x}_0$, a generic one-step method will exit the manifold $\mathcal{M}_{\mathbf{x}_0}$, but we will not get "too far" from it. We can then project back the obtained point to the correct level set. This is the main idea behind projection methods, also illustrated in Figure 3. We now formalise the definition of these methods and provide an example for a simple conserved quantity.

Let us consider an arbitrary one-step method $\varphi^h : \mathbb{R}^d \to \mathbb{R}^d$ of order p, for example a Runge–Kutta method. We define a projection method based on φ^h as follows:

$$\widetilde{\mathbf{x}}_1 = \varphi^h(\mathbf{x}_0), \quad \mathbf{x}_1 := \Pi_{\mathcal{M}_{\mathbf{x}_0}}(\widetilde{\mathbf{x}}_1),$$

where

$$\Pi_{\mathcal{M}_{\mathbf{x}_0}}(\widetilde{\mathbf{x}}_1) := \arg\min_{\mathbf{y}\in\mathcal{M}_{\mathbf{x}_0}} \|\mathbf{y} - \widetilde{\mathbf{x}}_1\|_2 .$$

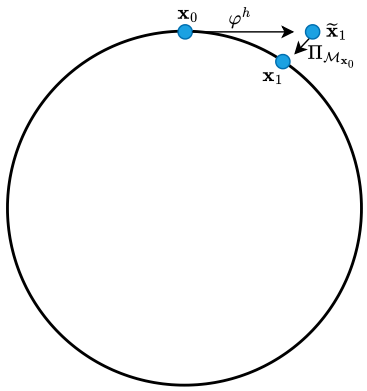

Figure 3: Illustration of one step of a projection method based on the one-step method $\varphi^h : \mathbb{R}^d \to \mathbb{R}^d$.

A practical way to compute these terms, is by defining

$$\mathbf{x}_1(\lambda) = \widetilde{\mathbf{x}}_1 + \lambda \nabla I(\widetilde{\mathbf{x}}_1),$$

and looking for a $\lambda_* \in \mathbb{R}$ such that $\mathbf{x}_1(\lambda_*) \in \mathcal{M}_{\mathbf{x}_0}$, i.e., $I(\mathbf{x}_1(\lambda_*)) = I(\mathbf{x}_0)$. We also remark that if I is smooth enough, one has

$$I\left(\widetilde{\mathbf{x}}_1\right) = I\left(\phi^h(\mathbf{x}_0) + \mathcal{O}(h^{p+1})\right) = I\left(\phi^h(\mathbf{x}_0)\right) + \mathcal{O}\left(h^{p+1}\right) = I\left(\mathbf{x}_0\right) + \mathcal{O}\left(h^{p+1}\right),$$

and hence $\lambda_0 = 0$ tends to be a good initial guess for the iterative method we have to use to solve the (non-linear) algebraic equation to satisfy the constraints.

Example 3. *For the case $I(\mathbf{x}) = \|\mathbf{x}\|_2^2/2$, we have $\nabla I(\mathbf{x}) = \mathbf{x}$, and hence $\mathbf{x}_1(\lambda) = (1+\lambda)\varphi^h(\mathbf{x}_0)$. The preservation of the first integral is then equivalent to*

$$(1+\lambda)^2 \left\|\varphi^h(\mathbf{x}_0)\right\|_2^2 = \|\mathbf{x}_0\|_2^2,$$

which implies

$$\lambda = -1 \pm \frac{\|\mathbf{x}_0\|_2}{\|\varphi^h(\mathbf{x}_0)\|_2}.$$

Since when $h = 0$ we would like to recover $\lambda = 0$, we choose the plus sign, hence getting the projection method

$$\mathbf{x}_1 = \frac{\varphi^h(\mathbf{x}_0)}{\|\varphi^h(\mathbf{x}_0)\|_2} \|\mathbf{x}_0\|_2 .$$

This approach easily extends to more than a single first integral. Let us suppose that the vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ admits $n < d$ functionally independent

first integrals $I = (I_1, ..., I_n) : \mathbb{R}^d \to \mathbb{R}^n$. This means that

$$I(\phi^t_{\mathcal{F}}(\mathbf{x}_0)) = I(\mathbf{x}_0),\ \forall t \geq 0,$$
$$\dim\left\{\mathbf{x} \in \mathbb{R}^d :\ I(\mathbf{x}) = \mathbf{c}\right\} = d - n\ \forall \mathbf{c} \in \text{range}(I),$$

or equivalently $\text{rank}(\partial_{\mathbf{x}} I(\mathbf{x})) = n$ for every $\mathbf{x}$ where $\partial_{\mathbf{x}} I(\mathbf{x})$ is well defined. In this case, we can define a projection method preserving the level sets of I, i.e., the joint level sets of $I_1, ..., I_n$, as follows:

$$\widetilde{\mathbf{x}}_1 = \varphi^h(\mathbf{x}_0),$$
$$\mathbf{x}_1(\boldsymbol{\lambda}) = \widetilde{\mathbf{x}}_1 + (\partial_{\widetilde{\mathbf{x}}_1} I(\widetilde{\mathbf{x}}_1))^\top \boldsymbol{\lambda},\ \boldsymbol{\lambda} \in \mathbb{R}^n.$$

Lemma 5 (Existence of a $\boldsymbol{\lambda}$)**.** *Let $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ be a vector field with n functionally independent first integrals collected in $I : \mathbb{R}^d \to \mathbb{R}^n$. Consider the projection method defined above based on the one-step method $\varphi^h : \mathbb{R}^d \to \mathbb{R}^d$. Then there exists an interval $[0, \bar{h}] \subset \mathbb{R}$ and a function $\boldsymbol{\lambda} : [0, \bar{h}] \to \mathbb{R}^n$ such that*

$$F(h, \boldsymbol{\lambda}(h)) := g(\mathbf{x}_1(\boldsymbol{\lambda}(h))) = I\left(\varphi^h(\mathbf{x}_0) + \partial_{\mathbf{x}} I(\varphi^h(\mathbf{x}_0))^\top \boldsymbol{\lambda}(h)\right) - I(\mathbf{x}_0) = 0 \quad (18)$$

for all $h \in [0, \bar{h}]$, and $\boldsymbol{\lambda}(0) = 0$.

Proof. The proof is based on the implicit function theorem. We first notice that $F(0, 0) = 0$ since $\mathbf{x}_1(\boldsymbol{\lambda}(0)) = \mathbf{x}_0$. Furthermore,

$$\partial_{\boldsymbol{\lambda}} F(h, \boldsymbol{\lambda})|_{(h,\boldsymbol{\lambda})=(0,0)} = \partial_{\mathbf{x}_0} I(\mathbf{x}_0) \partial_{\mathbf{x}_0} I(\mathbf{x}_0)^\top \in \mathbb{R}^{n \times n}, \quad (19)$$

which is invertible by the functional independency assumption. Thus, by the implicit function theorem, there is an interval $[0, \bar{h}]$ and a function $\boldsymbol{\lambda} = \boldsymbol{\lambda}(h)$ playing the desired role. □

Lemma 6 (Order of projection methods)**.** *Let $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ be a vector field admitting n functionally independent first integral collected in $I : \mathbb{R}^d \to \mathbb{R}^n$. Let $\varphi^h : \mathbb{R}^d \to \mathbb{R}^d$ be an order p one-step method, where h is a small enough step size so that Lemma 5 ensures the existence of $\boldsymbol{\lambda} = \boldsymbol{\lambda}(h)$. Then, the projection method we defined above still is an order p method.*

Proof. Let us expand F, defined as in (18), at $\boldsymbol{\lambda} = 0$, to get

$$F(h, \boldsymbol{\lambda}_*) = F(h, 0) + (\partial_{\boldsymbol{\lambda}} F(h, \boldsymbol{\lambda})|_{\boldsymbol{\lambda}=0})\, \boldsymbol{\lambda}_* + \mathcal{O}(\|\boldsymbol{\lambda}_*\|_2^2).$$

We can then notice that

$$F(h, 0) = I(\varphi^h(\mathbf{x}_0)) - I(\mathbf{x}_0) = \mathcal{O}(h^{p+1}),$$
$$\partial_{\boldsymbol{\lambda}} F(h, \boldsymbol{\lambda})|_{\boldsymbol{\lambda}=0} = \partial_{\boldsymbol{\lambda}} F(0, \boldsymbol{\lambda})|_{\boldsymbol{\lambda}=0} + \mathcal{O}(h).$$

Combining these results together with (19), we can conclude that

$$0 = F(h, \boldsymbol{\lambda}(h)) = \mathcal{O}(h^{p+1}) + \boldsymbol{\lambda}(h)\mathcal{O}(h) + \partial_{\mathbf{x}_0} I(\mathbf{x}_0) \partial_{\mathbf{x}_0} I(\mathbf{x}_0)^\top \boldsymbol{\lambda}(h) + \mathcal{O}(\|\boldsymbol{\lambda}(h)\|_2^2),$$

which implies that $\boldsymbol{\lambda}(h) \in \mathcal{O}(h^{p+1})$ and hence

$$\mathbf{x}_1(\boldsymbol{\lambda}(h)) = \phi^h_{\mathcal{F}}(\mathbf{x}_0) + \mathcal{O}(h^{p+1})$$

as desired. □

Typically projection methods are not so well seen in the community of geometric integration, because even though effective in preserving the conservation laws, they tend to break any other property the base method φ^h might have had (e.g., preserving linear invariants, being symplectic, or being volume preserving).

4.4.2 Discrete gradient methods

The last class of energy preserving methods we are going to see are discrete gradient methods. These are based on rewriting the vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ in a convenient way, which is the skew-gradient formulation we have already mentioned. More explicitly, we rewrite them as

$$\mathcal{F}(\mathbf{x}) = S(\mathbf{x})\nabla I(\mathbf{x}),\ \mathbb{R}^{d\times d} \ni S(\mathbf{x})^\top = -S(\mathbf{x}),$$

where $I : \mathbb{R}^d \to \mathbb{R}$ is known to be a first integral of $\mathcal{F}$. Discrete gradient methods are all implicit. They are based on the notion of discrete gradient, which we now define.

Definition 5 (Discrete gradient)**.** *Let $I : \mathbb{R}^d \to \mathbb{R}$ be a scalar valued continuously differentiable function. A discrete gradient of I is a function $\overline{\nabla} I : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ such that*

1. $\lim_{\mathbf{y}\to\mathbf{x}} \overline{\nabla} I(\mathbf{x},\mathbf{y}) = \nabla I(\mathbf{x})$, *for all* $\mathbf{x} \in \mathbb{R}^d$,
2. $\overline{\nabla} I(\mathbf{x},\mathbf{y})^\top(\mathbf{y}-\mathbf{x}) = I(\mathbf{y}) - I(\mathbf{x})$ *for all* $\mathbf{x},\mathbf{y} \in \mathbb{R}^d$.

We now provide three examples of discrete gradients:

1. Average Vector Field (AVF) :

$$\overline{\nabla} I(\mathbf{x},\mathbf{y}) := \int_0^1 \nabla I\left((1-s)\mathbf{x} + s\mathbf{y}\right) ds, \tag{20}$$

2. Gonzalez:

$$\overline{\nabla} I(\mathbf{x},\mathbf{y}) := \nabla I\left(\frac{\mathbf{x}+\mathbf{y}}{2}\right) + \frac{I(\mathbf{y}) - I(\mathbf{x}) - (\mathbf{y}-\mathbf{x})^\top \nabla I\left(\frac{\mathbf{x}+\mathbf{y}}{2}\right)}{\|\mathbf{y}-\mathbf{x}\|_2^2}(\mathbf{y}-\mathbf{x}), \tag{21}$$

3. Itoh-Abe:

$$\overline{\nabla} I(\mathbf{x},\mathbf{y}) := \begin{bmatrix} \frac{I(y_1,x_2,\ldots,x_d) - I(\mathbf{x})}{y_1 - x_1} \\ \frac{I(y_1,y_2,x_3,\ldots,x_d) - I(y_1,x_2,x_3,\ldots,x_d)}{y_2 - x_2} \\ \vdots \\ \frac{I(\mathbf{y}) - I(y_1,\ldots,y_{d-1},x_d)}{y_d - x_d} \end{bmatrix}. \tag{22}$$

Proposition 3 (AVF is a discrete gradient)**.** *Let $I : \mathbb{R}^d \to \mathbb{R}$ be a smooth function. Then the AVF map (20) is a well-defined discrete gradient.*

Proof. First, let us notice that

$$\begin{aligned} I(\mathbf{y}) - I(\mathbf{x}) &= \int_0^1 \frac{d}{ds} I((1-s)\mathbf{x} + s\mathbf{y}) ds \\ &= \int_0^1 \nabla I((1-s)\mathbf{x} + s\mathbf{y})^\top (\mathbf{y} - \mathbf{x}) ds = \overline{\nabla} I(\mathbf{x}, \mathbf{y})^\top (\mathbf{y} - \mathbf{x}) \end{aligned}$$

as desired. Furthermore,

$$\lim_{\mathbf{y} \to \mathbf{x}} \overline{\nabla} I(\mathbf{x}, \mathbf{y}) = \int_0^1 \lim_{\mathbf{y} \to \mathbf{x}} \nabla I((1-s)\mathbf{x} + s\mathbf{y}) ds = \nabla I(\mathbf{x}).$$

□

Proposition 4 (Gonzalez is a discrete gradient)**.** *Let $I : \mathbb{R}^d \to \mathbb{R}$ be a smooth function. Then the Gonzalez map (21) is a well-defined discrete gradient.*

Proof. By direct calculation we see that

$$\begin{aligned} \overline{\nabla} I(\mathbf{x}, \mathbf{y})^\top (\mathbf{y} - \mathbf{x}) &= \nabla I \left(\frac{\mathbf{x} + \mathbf{y}}{2} \right)^\top (\mathbf{y} - \mathbf{x}) \\ &\quad + \frac{I(\mathbf{y}) - I(\mathbf{x}) - (\mathbf{y} - \mathbf{x})^\top \nabla I \left(\frac{\mathbf{x}+\mathbf{y}}{2} \right)}{\|\mathbf{y} - \mathbf{x}\|_2^2} \|\mathbf{y} - \mathbf{x}\|_2^2 \\ &= \nabla I \left(\frac{\mathbf{x} + \mathbf{y}}{2} \right)^\top (\mathbf{y} - \mathbf{x}) \\ &\quad + I(\mathbf{y}) - I(\mathbf{x}) - (\mathbf{y} - \mathbf{x})^\top \nabla I \left(\frac{\mathbf{x} + \mathbf{y}}{2} \right) = I(\mathbf{y}) - I(\mathbf{x}). \end{aligned}$$

To check the consistency condition, we notice that

$$I(\mathbf{y}) = I(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^\top \nabla I(\mathbf{x}) + \mathcal{O}(\|\mathbf{y} - \mathbf{x}\|_2^2)$$

and hence

$$\left| \frac{I(\mathbf{y}) - I(\mathbf{x}) - (\mathbf{y} - \mathbf{x})^\top \nabla I(\mathbf{x})}{\|\mathbf{y} - \mathbf{x}\|_2^2} \right| \leq c$$

for some $c > 0$ as $\|\mathbf{y} - \mathbf{x}\|_2 \to 0$. This implies that

$$\lim_{\|\mathbf{y} - \mathbf{x}\|_2 \to 0} \overline{\nabla} I(\mathbf{x}, \mathbf{y}) = \nabla I(\mathbf{x})$$

as desired. □

Proposition 5 (Itoh-Abe is a discrete gradient)**.** *Let $I : \mathbb{R}^d \to \mathbb{R}$ be a smooth function. Then the Itoh-Abe map (22) is a well-defined discrete gradient.*

Proof. We start by calculating

$$(\mathbf{y}-\mathbf{x})^\top \overline{\nabla} I(\mathbf{x},\mathbf{y}) = (\mathbf{y}-\mathbf{x})^\top \begin{bmatrix} \frac{I(y_1,x_2,\dots,x_d)-I(\mathbf{x})}{y_1-x_1} \\ \frac{I(y_1,y_2,x_3,\dots,x_d)-I(y_1,x_2,x_3,\dots,x_d)}{y_2-x_2} \\ \vdots \\ \frac{I(\mathbf{y})-I(y_1,\dots,y_{d-1},x_d)}{y_d-x_d} \end{bmatrix}$$
$$= (I(y_1,x_2,\dots,x_d) - I(\mathbf{x}))$$
$$+ (I(y_1,y_2,x_3,\dots,x_d) - I(y_1,x_2,x_3,\dots,x_d))$$
$$+ \dots$$
$$+ (I(\mathbf{y}) - I(y_1,\dots,y_{d-1},x_d)) = I(\mathbf{y}) - I(\mathbf{x}),$$

since the mixed terms cancel out, as pointed out for example for the pair in red.

The consistency condition is easy to verify since, if we focus for example on the $i-$th entry we have

$$\lim_{h_i\to 0} \frac{I(x_1,\dots,x_{i-1},x_i+h_i,x_{i+1},\dots,x_d) - I(\mathbf{x})}{h_i} = \partial_{x_i} I(\mathbf{x}),$$

where we wrote $y_i = x_i + h_i$. Thus, the full vector converges to $\nabla I(\mathbf{x})$ given that its components converge to the correct partial derivatives. □

Once we choose a discrete gradient $\overline{\nabla} I$, we can define the associated discrete gradient method

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\overline{S}(\mathbf{x}_n,\mathbf{x}_{n+1})\overline{\nabla} I(\mathbf{x}_n,\mathbf{x}_{n+1}), \tag{23}$$

where $\overline{S}:\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}^{d\times d}$ is such that $\overline{S}(\mathbf{x},\mathbf{x}) = S(\mathbf{x})$ and for every $\mathbf{x},\mathbf{y}\in\mathbb{R}^d$, one has $\overline{S}(\mathbf{x},\mathbf{y})^\top = -\overline{S}(\mathbf{x},\mathbf{y})$.

Proposition 6. *The scheme in (23) conserves the first integral $I:\mathbb{R}^d\to\mathbb{R}$ of $\mathcal{F}:\mathbb{R}^d\to\mathbb{R}^d$.*

Proof. The proof follows by direct calculation. First, by one of the properties of discrete gradients we see that

$$I(\mathbf{x}_{n+1}) - I(\mathbf{x}_n) = \overline{\nabla} I(\mathbf{x}_n,\mathbf{x}_{n+1})^\top (\mathbf{x}_{n+1}-\mathbf{x}_n).$$

Replacing the expression for $\mathbf{x}_{n+1}-\mathbf{x}_n$ defined by (23), we can see that

$$\frac{I(\mathbf{x}_{n+1}) - I(\mathbf{x}_n)}{h} = \overline{\nabla} I(\mathbf{x}_n,\mathbf{x}_{n+1})^\top \overline{S}(\mathbf{x}_n,\mathbf{x}_{n+1})\overline{\nabla} I(\mathbf{x}_n,\mathbf{x}_{n+1}) = 0$$

by the skew-symmetry of $\overline{S}(\mathbf{x}_n,\mathbf{x}_{n+1})$. □

Let us focus on a particularly interesting class of systems, i.e., polynomial differential equations with a polynomial first integral. Fix a skew-symmetric matrix $S\in\mathbb{R}^{d\times d}$ and a polynomial function $I:\mathbb{R}^d\to\mathbb{R}$. Consider then the dynamics

$$\dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t)) := S\nabla I(\mathbf{x}(t)). \tag{24}$$

By construction, I is a first integral of the system.

Proposition 7. *Let $b_1, ..., b_s$ and $c_1, ..., c_s$ define a quadrature formula of polynomial order $m-1$. Consider the equation (24) with I a polynomial of degree m. Then the $s-$stages Runge–Kutta method*

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \sum_{i=1}^{s} b_i \mathcal{F}(\mathbf{x}_n + c_i(\mathbf{x}_{n+1} - \mathbf{x}_n))$$

exactly conserves I.

Proof. Since I is a polynomial of degree m, the vector field $\mathcal{F}(\mathbf{x}) = S\nabla I(\mathbf{x})$, being based on its gradient, will have components which are polynomials of degree $m-1$. Thus,

$$\begin{aligned} h \sum_{i=1}^{s} b_i \mathcal{F}(\mathbf{x}_n + c_i(\mathbf{x}_{n+1} - \mathbf{x}_n)) &= h \sum_{i=1}^{s} b_i \mathcal{F}((1-c_i)\mathbf{x}_n + c_i \mathbf{x}_{n+1}) \\ &= \int_0^1 \mathcal{F}((1-s)\mathbf{x}_n + s\mathbf{x}_{n+1}) ds. \end{aligned}$$

We can thus rewrite the Runge–Kutta method as

$$\mathbf{x}_{n+1} = \mathbf{x}_n + hS\overline{\nabla} I(\mathbf{x}_n, \mathbf{x}_{n+1})$$

where $\overline{\nabla} I$ is the AVF discrete gradient of I. This allows to conclude the proof. □

In this section we have shown how discrete gradient methods can be used to preserve first integrals. With minor changes, we can also apply these methods to dissipate some target function along the numerical solution. This could be useful when minimising a loss function, for example in the training procedure of a neural network. Let us focus, for simplicity, on the differential equation leading to the gradient descent updates, i.e., the negative gradient flow

$$\dot{\mathbf{x}}(t) = -\nabla V(\mathbf{x}(t)),$$

where $V : \mathbb{R}^d \to \mathbb{R}$ is an $L-$smooth convex and continuously differentiable function. For this class of equations, (23) is slightly changed to

$$\mathbf{x}_{n+1} = \mathbf{x}_n - h\overline{\nabla} V(\mathbf{x}_n, \mathbf{x}_{n+1}),$$

since instead of conserving the function $V : \mathbb{R}^d \to \mathbb{R}$ we want to dissipate V. We can in fact immediately verify that

$$V(\mathbf{x}_{n+1}) - V(\mathbf{x}_n) = \overline{\nabla} V(\mathbf{x}_n, \mathbf{x}_{n+1})^\top (\mathbf{x}_{n+1} - \mathbf{x}_n) = -h \left\| \overline{\nabla} V(\mathbf{x}_n, \mathbf{x}_{n+1}) \right\|_2^2 \leq 0,$$

and hence the target loss function is guaranteed to be decreased at every step of the update.

4.5 Neural networks conserving linear and quadratic first integrals

A relatively simple property to impose over a ResNet is mass preservation. By mass preservation, we refer to the conservation of the sum of the components of a vector (see [3]). This property is typical of semi-discretisations of mass-preserving PDEs, models for chemical reactions, population dynamics, and ecology (see, e.g. [17, 6, 25]). More explicitly, one could be interested in imposing such a structure if the goal is to approximate a function $F : \mathbb{R}^d \to \mathbb{R}^c$ that is known to satisfy $T_d\mathbf{x} := \sum_{i=1}^d x_i = T_cF(\mathbf{x}) := \sum_{j=1}^c F(\mathbf{x})_j$. A simple way to impose such property is by approximating the target function $F : \mathbb{R}^d \to \mathbb{R}^c$ as

$$F(\mathbf{x}) \approx \frac{\sum_{i=1}^d x_i}{\sum_{j=1}^c \tilde{F}(\mathbf{x})_j}\tilde{F}(\mathbf{x})$$

where $\tilde{F} : \mathbb{R}^d \to \mathbb{R}^c$ is any sufficiently expressive neural network. However, this choice might lead to hard training procedures because of the denominator. Imposing this structure at the level of network layers is not so intuitive in general. Hence, we rely again on a suitable ODE formulation. A vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ whose flow map preserves the sum of the components of the state vector is one having a linear first integral $I(\mathbf{x}) = \mathbf{1}^\top\mathbf{x} = \sum_{i=1}^d x_i$. Thus, we can design vector fields of the form

$$\dot{\mathbf{x}}(t) = \left(A(\mathbf{x}) - A(\mathbf{x})^\top\right)\mathbf{1}, \;\; A : \mathbb{R}^d \to \mathbb{R}^{d\times d}, \tag{25}$$

having exact flow map $\phi_{\mathcal{F}}^t$ for which $I(\phi_{\mathcal{F}}^t(\mathbf{x}_0)) = I(\mathbf{x}_0)$ for any $\mathbf{x}_0 \in \mathbb{R}^d$ and $t \geq 0$. To model these vector fields, we can work with parametric functions like $B_2\sigma(B_1\mathbf{x} + \mathbf{b}_1) \in \mathbb{R}^{d(d-1)/2}$ and use them to build the upper triangular matrix-valued function A in (25). An alternative parametrisation strategy could be

$$\mathcal{F}_\theta(\mathbf{x}) = \widehat{\mathcal{F}}_\theta(\mathbf{x}) - \frac{1}{d}\left(\mathbf{1}^\top\widehat{\mathcal{F}}_\theta(\mathbf{x})\right)\mathbf{1},$$

where $\widehat{\mathcal{F}}_\theta : \mathbb{R}^d \to \mathbb{R}^d$ can be any parametric vector field, e.g. any neural network.

This mass conservation could also be extended to weighted-mass conservation, and we would have to replace $\mathbf{1}$ with a vector of weights $\boldsymbol{\alpha}$. However, this extension does not allow the dimensionality to change from one layer to the next as easily. Imposing this property at a discrete level is also fairly simple since every Runge–Kutta method preserves linear first integrals without time-step restrictions. Thus, a possible strategy to model mass-preserving neural networks is based on combining layers of the following types:

1. Lifting layers: $L : \mathbb{R}^k \to \mathbb{R}^{k+s}$, $L(x_1, \ldots, x_k) = (x_1, \ldots, x_k, 0, 0, \ldots, 0)$,
2. Projection layers: $P : \mathbb{R}^{k+s} \to \mathbb{R}^k$, $P(x_1, \ldots, x_k, x_{k+1}, \ldots, x_{k+s}) = (x_1 + o, \ldots, x_k + o)$, with $o = \sum_{i=1}^s x_{k+i}/s$,
3. Dynamical blocks: single steps of the explicit Euler method applied to (25).

To test this neural network architecture, we focus on the approximation of the flow map of the SIR model

$$\dot{\mathbf{x}} = \begin{bmatrix} -x_1 x_2 & x_1 x_2 - x_2 & x_2 \end{bmatrix}^\top = \mathcal{F}(\mathbf{x})^\top. \tag{26}$$

This experiment relates to the research area of data-driven modelling, which has attracted a high amount of interest in recent years, especially through the tools provided by machine learning. We model the neural network as discussed above. We approximate the $1-$flow map of (26) working with pairs of the form $\{(\mathbf{x}_0^i, \mathbf{x}_1^i = \varphi_{\mathcal{F}}^1(\mathbf{x}_0^i))\}_{i=1,\dots,N}$[1]. In this context, we suppose it is not possible to integrate in time the system of ODEs because we have no access to it. What is provided is just a set of observed trajectories. The plots in Figure 4 represent the first two components of the solution for the SIR model. All the line segments connect the components of the initial conditions with those of the time-1 updates. The considerable benefit of mass preservation as a constraint is that it allows interpretable outputs. Indeed, in this case, the components of $\mathbf{x}$ represent the percentages of three species in the total population, and the network we train still allows us to get this interpretation to be mass-preserving.

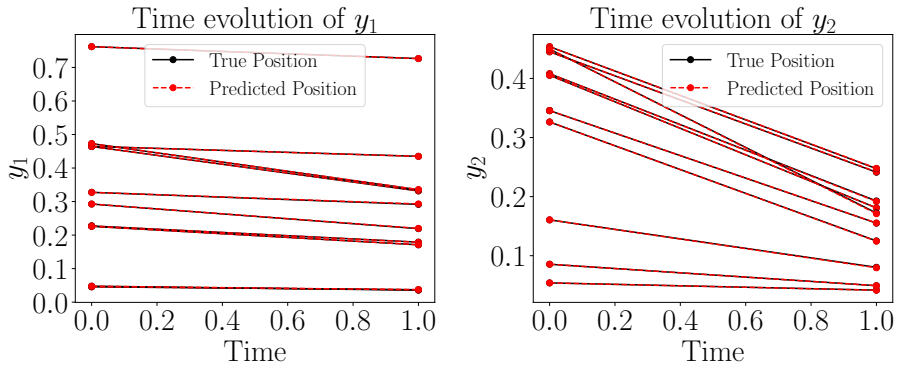

Figure 4: Plots of the approximation of the time $1-$flow map of the SIR model (26) for 10 test initial conditions. We report the first two components of the solutions. Each point represents either an initial condition (at time 0), or a time-1 update.

In a similar fashion, we could also build ResNets based on dynamical systems that preserve a quadratic first integral. We can for example use the projection method we derived in Section 4. Following the same procedure seen above, we can define a parametrised vector field $\mathcal{F}_\theta : \mathbb{R}^d \to \mathbb{R}^d$ which is tangent to Euclidean spheres, meaning that it conserves the quadratic energy function $E(\mathbf{x}) = \|\mathbf{x}\|_2^2/2$. To do so, an option is to model it as

$$\mathcal{F}_\theta(\mathbf{x}) = \widehat{\mathcal{F}}_\theta(\mathbf{x}) - \frac{\widehat{\mathcal{F}}_\theta(\mathbf{x})^\top \mathbf{x}}{\mathbf{1}^\top \mathbf{x}} \mathbf{1}, \tag{27}$$

[1]Here with $\varphi_{\mathcal{F}}^1(\mathbf{x}_0^i)$ we refer to an accurate approximation of the time-1 flow map of $\mathcal{F}$ applied to $\mathbf{x}_0^i$

for an unconstrained parametric vector field $\widehat{\mathcal{F}}_\theta : \mathbb{R}^d \to \mathbb{R}^d$. In fact, with the parametrisation in (27), we have

$$\mathbf{x}^\top \mathcal{F}_\theta(\mathbf{x}) = \mathbf{x}^\top \widehat{\mathcal{F}}_\theta(\mathbf{x}) - \frac{\widehat{\mathcal{F}}_\theta(\mathbf{x})^\top \mathbf{x}}{\mathbf{1}^\top \mathbf{x}} \mathbf{1}^\top \mathbf{x} = 0.$$

We could then define a layer of a ResNet by performing one step with a projection method applied to $\mathcal{F}_{\theta_i}$, getting

$$F_{\theta_i}(\mathbf{x}) = \frac{\varphi^h_{\mathcal{F}_{\theta_i}}(\mathbf{x})}{\|\varphi^h_{\mathcal{F}_{\theta_i}}(\mathbf{x})\|_2} \|\mathbf{x}\|_2,$$

where for example we could choose $\varphi^h_{\mathcal{F}_{\theta_i}}(\mathbf{x}) = \mathbf{x} + h\mathcal{F}_{\theta_i}(\mathbf{x})$.

5 Symplectic methods

5.1 Introduction to Hamiltonian systems

Especially in physics, many systems can be described through the Lagrangian and Hamiltonian formalisms. We briefly present these two formalisms in the case of systems defined on a linear space.

The Lagrangian description of a conservative system is based on a Lagrangian function $L : T\mathbb{R}^d \to \mathbb{R}$, where $T\mathbb{R}^d \simeq \mathbb{R}^{2d}$ is the tangent bundle of $\mathbb{R}^d$. A generic point of $T\mathbb{R}^d$ is represented by its generalised coordinates $(\mathbf{q}, \dot{\mathbf{q}})$. The equations of motion of Lagrangian systems are defined by the second-order system

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}})\right) - \frac{\partial L}{\partial \mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}) = 0,$$

which can be obtained following a variational principle. One can then introduce the Legendre transform

$$(\mathbf{q}, \mathbf{p}) := \mathbb{F}L(\mathbf{q}, \dot{\mathbf{q}}) = \left(\mathbf{q}, \frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}})\right),$$

which we now suppose is a diffeomorphism, and provide a change of variables between the tangent bundle $T\mathbb{R}^d \simeq \mathbb{R}^{2d}$ and the cotangent bundle $T^*\mathbb{R}^d \simeq \mathbb{R}^{2d}$, where the pair $(\mathbf{q}, \mathbf{p})$ belongs. This change of variables allows one to define the conjugate momentum $\mathbf{p}$ and introduce the Hamiltonian formalism based on the Hamiltonian energy

$$H(\mathbf{q}, \mathbf{p}) = (\mathbf{p} \cdot \dot{\mathbf{q}} - L(\mathbf{q}, \dot{\mathbf{q}}))\Big|_{(\mathbf{q}, \dot{\mathbf{q}}) = (\mathbb{F}L)^{-1}(\mathbf{q}, \mathbf{p})}.$$

Several systems admit a Lagrangian function that takes the form

$$L(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top M(\mathbf{q})\dot{\mathbf{q}} - U(\mathbf{q}),$$

where $M(\mathbf{q})$ defines a metric on $\mathbb{R}^d$. For these systems, the Hamiltonian reads

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^\top M^{-1}(\mathbf{q})\mathbf{p} + U(\mathbf{q}).$$

The Hamiltonian H is separable if M is independent of $\mathbf{q}$. More generally, a Hamiltonian function $H : T^*\mathbb{R}^d \to \mathbb{R}$ is called separable if it is the sum of two terms depending only on one of the two variables, i.e., either on $\mathbf{q}$ or on $\mathbf{p}$. We will exploit this structure to build some explicit symplectic methods in the upcoming part of this section.

The Hamiltonian equations of motion with respect to the canonical symplectic structure of $\mathbb{R}^{2d}$ are

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{bmatrix} = \mathbb{J}\nabla H(\mathbf{q},\mathbf{p}) =: X_H(\mathbf{q},\mathbf{p}), \tag{28}$$

where

$$\mathbb{J} = \begin{bmatrix} 0 & I \\ -I & 0\end{bmatrix} \in \mathbb{R}^{2d\times 2d}$$

is the canonical symplectic matrix of $\mathbb{R}^{2d}$, and $I, 0 \in \mathbb{R}^{d\times d}$ are the identity and the zero matrices respectively. The equations in (28) could be more abstractly introduced using the formalism of differential forms. These equations provide the coordinate representation of those more general equations, and in this section we will only use this coordinate representation which is convenient since we are working with a linear configuration manifold, i.e., $\mathbb{R}^d$.

Hamiltonian systems have several interesting geometric and dynamic properties. First of all they are in the skew-gradient form (13), hence they conserve the energy function H. But even more importantly, they preserve a skew-symmetric bilinear form called the canonical symplectic form $\Omega : \mathbb{R}^{2d} \times \mathbb{R}^{2d} \to \mathbb{R}$ defined as

$$\Omega(\mathbf{v},\mathbf{w}) := \mathbf{v}^\top \mathbb{J}\mathbf{w}.$$

A map preserving Ω is said to be symplectic. A matrix $A \in \mathbb{R}^{2d\times 2d}$ is symplectic if and only if it satisfies $A^\top \mathbb{J} A = \mathbb{J}$ and similarly we say the linear map $L(x) = Ax$ symplectic. Instead, we say a non-linear differentiable map $F : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ to be symplectic if for every $\mathbf{x} \in \mathbb{R}^{2d}$ it holds

$$F'(\mathbf{x})^\top \mathbb{J} F'(\mathbf{x}) = \mathbb{J}, \tag{29}$$

where $F'(\mathbf{x}) \in \mathbb{R}^{2d\times 2d}$ is the Jacobian matrix of F evaluated at $\mathbf{x}$. Equivalently, F is symplectic if it infinitesimally preserves Ω since (29) is equivalent to say

$$\Omega(F'(\mathbf{x})\mathbf{v}, F'(\mathbf{x})\mathbf{w}) = \Omega(\mathbf{v},\mathbf{w}),\ \ \forall \mathbf{x},\mathbf{v},\mathbf{w} \in \mathbb{R}^{2d}.$$

Lemma 7 (Volume preservation)**.** *Let $F : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ be a symplectic diffeomorphism. Then F also preserves the canonical volume form of $\mathbb{R}^{2d}$, i.e., for any $\Omega \subset \mathbb{R}^{2d}$ open we have*

$$\mathrm{vol}(F(\Omega)) = \int_{F(\Omega)} dx_1 ... dx_{2d} = \int_\Omega dx_1 ... dx_{2d} = \mathrm{vol}(\Omega).$$

Proof. The proof follows directly from the fact that

$$\int_{F(\Omega)} dx_1 ... dx_{2d} = \int_{\Omega} |\det(F'(\mathbf{x}))| \, dx_1 ... dx_{2d}$$

and hence F is volume preserving if and only if $|\det(F'(\mathbf{x}))| = 1$ for every $\mathbf{x} \in \mathbb{R}^{2d}$. By the symplectic property, we have

$$F'(\mathbf{x})^\top \mathbb{J} F'(\mathbf{x}) = \mathbb{J} \implies \det(F'(\mathbf{x}))^2 \det(\mathbb{J}) = \det(\mathbb{J}),$$

and hence $|\det(F'(\mathbf{x}))| \equiv 1$ as desired. □

Proposition 8. *Let H be a twice continuously differentiable function on $U \subset \mathbb{R}^{2d}$. Then, for each fixed $t \in \mathbb{R}$, the time$-t$ flow map $\phi^t_{X_H}$ is symplectic whenever it is defined.*

We could prove this quickly using the more abstract formulation based on differential forms and the Cartan's magic formula, but we now see the typical proof provided in numerical analysis books.

Proof. We recall that the flow map $\phi^t_{X_H} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ satisfies

$$\frac{d}{dt} \phi^t_{X_H}(\mathbf{x}_0) = X_H \left(\phi^t_{X_H}(\mathbf{x}_0) \right)$$

for every $t \geq 0$. Differentiating both sides with respect to $\mathbf{x}_0$, we get

$$\partial_{\mathbf{x}_0} \frac{d}{dt} \phi^t_{X_H}(\mathbf{x}_0) = \mathbb{J} \nabla^2 H(\phi^t_{X_H}(\mathbf{x}_0)) \, \partial_{\mathbf{x}_0} \phi^t_{X_H}(\mathbf{x}_0),$$

where $\nabla^2 H$ is the Hessian matrix of H. Changing the differentiation order on the left, and calling $S_{\mathbf{x}_0}(t) = \partial_{\mathbf{x}_0} \phi^t_{X_H}(\mathbf{x}_0)$, we see that

$$\frac{d}{dt} S_{\mathbf{x}_0}(t) = \mathbb{J} \nabla^2 H(\phi^t_{X_H}(\mathbf{x}_0)) S_{\mathbf{x}_0}(t), \tag{30}$$

which is the variational equation for the Hamiltonian system of ODEs. We can then compute

$$\begin{aligned} \frac{d}{dt} \left(S_{\mathbf{x}_0}(t)^\top \mathbb{J} S_{\mathbf{x}_0}(t) \right) &= \left(\frac{d}{dt} S_{\mathbf{x}_0}(t) \right)^\top \mathbb{J} S_{\mathbf{x}_0}(t) + S_{\mathbf{x}_0}(t)^\top \mathbb{J} \left(\frac{d}{dt} S_{\mathbf{x}_0}(t) \right) \\ &\overset{(30)}{=} \left(S_{\mathbf{x}_0}(t)^\top \nabla^2 H(\phi^t_{X_H}(\mathbf{x}_0)) \mathbb{J}^\top \right) \mathbb{J} S_{\mathbf{x}_0}(t) + S_{x_0}(t)^\top \mathbb{J} \left(\mathbb{J} \nabla^2 H(\phi^t_{X_H}(\mathbf{x}_0)) S_{x_0}(t) \right). \end{aligned}$$

Since $\mathbb{J}^\top \mathbb{J} = I_{2d}$ and $\mathbb{J}^2 = -I_{2d}$ we conclude that the quantity above is 0 and hence $S_{\mathbf{x}_0}(t)^\top \mathbb{J} S_{\mathbf{x}_0}(t) = S_{\mathbf{x}_0}(0)^\top \mathbb{J} S_{\mathbf{x}_0}(0)$. At time 0, we recall that

$$S_{\mathbf{x}_0}(0) = \partial_{\mathbf{x}_0} \phi^0_{X_H}(\mathbf{x}_0) = \partial_{\mathbf{x}_0} \mathbf{x}_0 = I_{2d}$$

which allows to conclude $S_{\mathbf{x}_0}(t)^\top \mathbb{J} S_{\mathbf{x}_0}(t) = \mathbb{J}$ for every $t \geq 0$ as desired. □

Before moving to numerical methods, let us also see that on $\mathbb{R}^{2d}$ the flow map of a vector field is symplectic if and only if the vector field is Hamiltonian.

Lemma 8. *Let $\mathcal{F} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ be a continuously differentiable vector field, and $\phi^t_{\mathcal{F}}$ its time−t flow map. $\phi^t_{\mathcal{F}} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is symplectic for every t for which it is defined if and only if there exists a twice continuously differentiable function $H : \mathbb{R}^{2d} \to \mathbb{R}$ such that $\mathcal{F}(\mathbf{x}) = \mathbb{J}\nabla H(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^{2d}$.*

Proof. One direction follows from Proposition 8. Let us now suppose that $\phi^t_{\mathcal{F}}$ is symplectic, and call $S_{\mathcal{F}}$ its sensitivity matrix, which we recall solves

$$\dot{S}_{\mathcal{F}}(t) = \mathcal{F}'(\mathbf{x}(t))S_{\mathcal{F}}(t).$$

Then, since $\phi^t_{\mathcal{F}}$ is symplectic, it follows that

$$\begin{aligned} 0 &= \frac{d}{dt}\left(S^\top_{\mathcal{F}}(t)\mathbb{J}S_{\mathcal{F}}(t)\right) = \dot{S}^\top_{\mathcal{F}}(t)\mathbb{J}S_{\mathcal{F}}(t) + S^\top_{\mathcal{F}}(t)\mathbb{J}\dot{S}_{\mathcal{F}}(t) \\ &= S^\top_{\mathcal{F}}(t)\left(\left(\mathcal{F}'(\mathbf{x}(t))\right)^\top\mathbb{J} + \mathbb{J}\mathcal{F}'(\mathbf{x}(t))\right)S_{\mathcal{F}}(t). \end{aligned}$$

Recalling that $\mathbb{J}^\top = -\mathbb{J}$, we thus notice that the Jacobian matrix

$$\frac{\partial(\mathbb{J}\mathcal{F}(\mathbf{x}_0))}{\partial \mathbf{x}_0} \in \mathbb{R}^{2d\times 2d}$$

is symmetric for every $\mathbf{x}_0 \in \mathbb{R}^{2d}$. Thus, being $\mathbb{R}^{2d}$ simply connected, Lemma 9 ensures that there is a globally defined function $H : \mathbb{R}^{2d} \to \mathbb{R}$ such that

$$\mathbb{J}\mathcal{F}(\mathbf{x}) \equiv \nabla H(\mathbf{x}),$$

which allows to conclude the proof. □

Lemma 9 (Integrability lemma)**.** *Let $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$ be continuously differentiable and with symmetric Jacobian matrix $\mathcal{F}'(\mathbf{x}) \in \mathbb{R}^{n\times n}$. Then, there exists a twice-continuously differentiable function $H : \mathbb{R}^n \to \mathbb{R}$ such that $\mathcal{F}(\mathbf{x}) = \nabla H(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^n$.*

Proof of the integrability lemma. Let us consider the function

$$H(\mathbf{x}) = \int_0^1 \mathbf{x}^\top \mathcal{F}(t\mathbf{x})dt.$$

The gradient of this function has k−th component given by

$$\begin{aligned} \partial_{x_k} H(\mathbf{x}) &= \partial_{x_k}\left(\int_0^1 \sum_{i=1}^n x_i F_i(t\mathbf{x})dt\right) = \int_0^1 \left(F_k(t\mathbf{x}) + t\sum_{i=1}^n x_i \left.\frac{\partial F_i(\mathbf{y})}{\partial y_k}\right|_{\mathbf{y}=t\mathbf{x}}\right)dt \\ &= \int_0^1 \left(F_k(t\mathbf{x}) + t\mathbf{x}^\top\mathcal{F}'(t\mathbf{x})\mathbf{e}_k\right)dt = \int_0^1 \left(F_k(t\mathbf{x}) + t\mathbf{x}^\top\mathcal{F}'(t\mathbf{x})^\top\mathbf{e}_k\right)dt \\ &= \int_0^1 \left(F_k(t\mathbf{x}) + t\mathbf{x}^\top\nabla F_k(t\mathbf{x})\right)dt = \int_0^1 \frac{d}{dt}\left(tF_k(t\mathbf{x})\right)dt = F_k(\mathbf{x}), \end{aligned}$$

where $\mathcal{F}(\mathbf{x}) = \begin{bmatrix} F_1(\mathbf{x}) & \dots & F_n(\mathbf{x}) \end{bmatrix}^\top$, and the red expression is obtained by the symmetry property $\mathcal{F}'(\mathbf{x})^\top = \mathcal{F}'(\mathbf{x})$. This derivation allows to conclude the proof since we get $\nabla H(\mathbf{x}) = \mathcal{F}(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^n$. □

We have seen that the flow map of a Hamiltonian system is a symplectic map. We would thus like to preserve this symplectic property also when approximating it numerically. There are several ways to get symplectic methods, but we will focus on those obtained via splitting methods and on Runge–Kutta methods which are symplectic.

Definition 6 (Symplectic one-step method)**.** *A one-step method $\varphi^h : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is symplectic if and only if when applied to a Hamiltonian system the map φ^h is symplectic, i.e.,*

$$\left(\frac{\partial \varphi^h(\mathbf{x})}{\mathbf{x}}\right)^\top \mathbb{J} \left(\frac{\partial \varphi^h(\mathbf{x})}{\mathbf{x}}\right) = \mathbb{J}$$

for every $\mathbf{x} \in \mathbb{R}^{2d}$.

5.2 Symplectic splitting methods

In this section we aim to exploit the fact that the flow map of a Hamiltonian system is symplectic, and approximate the solution of (28) by composing the exact flows of simpler Hamiltonian systems.

Splitting methods are a class of methods based on writing the target differential equation, say $\dot{\mathbf{x}} = \mathcal{F}(\mathbf{x})$, as the sum of simpler terms, for example as $\mathcal{F}(\mathbf{x}) = \mathcal{F}_1(\mathbf{x}) + \mathcal{F}_2(\mathbf{x})$, supposing we are able to find the exact solution of the two differential equations $\dot{\mathbf{x}} = \mathcal{F}_1(\mathbf{x})$ and $\dot{\mathbf{x}} = \mathcal{F}_2(\mathbf{x})$. Unfortunately, in general we have

$$\phi^t_{\mathcal{F}} \neq \phi^t_{\mathcal{F}_1} \circ \phi^t_{\mathcal{F}_2}, \ \phi^t_{\mathcal{F}} \neq \phi^t_{\mathcal{F}_2} \circ \phi^t_{\mathcal{F}_1}, \ \phi^t_{\mathcal{F}_1} \circ \phi^t_{\mathcal{F}_2} \neq \phi^t_{\mathcal{F}_2} \circ \phi^t_{\mathcal{F}_1}. \tag{31}$$

A simple example to show that (31) is true, can be found by considering the Hamiltonian vector field

$$\mathcal{F}(q,p) = \begin{bmatrix} p \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -q \end{bmatrix} =: \mathcal{F}_1(q,p) + \mathcal{F}_2(q,p), \ q,p \in \mathbb{R}. \tag{32}$$

In fact, in this case we have

$$\phi^t_{\mathcal{F}_1}(q,p) = \begin{bmatrix} q + tp \\ p \end{bmatrix}, \ \phi^t_{\mathcal{F}_2}(q,p) = \begin{bmatrix} q \\ p - tq \end{bmatrix},$$

while the equation $\dot{\mathbf{x}} = \mathcal{F}(\mathbf{x})$ can be rewritten as the second order differential equation $\ddot{q} = -q$, which has a trigonometric solution.

We recall that we are not aiming for an exact representation of $\phi^t_{\mathcal{F}}$ but for us it would be sufficient to approximate it after a time step $t = h > 0$. This

is by far a more achievable goal. In fact, we can show that as long as $h > 0$ is small enough, one has

$$\phi^h_{\mathcal{F}} = \phi^h_{\mathcal{F}_1} \circ \phi^h_{\mathcal{F}_2} + \mathcal{O}(h^2), \qquad \phi^h_{\mathcal{F}} = \phi^h_{\mathcal{F}_2} \circ \phi^h_{\mathcal{F}_1} + \mathcal{O}(h^2), \tag{33}$$

$$\phi^h_{\mathcal{F}} = \phi^{h/2}_{\mathcal{F}_1} \circ \phi^h_{\mathcal{F}_2} \circ \phi^{h/2}_{\mathcal{F}_1} + \mathcal{O}(h^3), \qquad \phi^h_{\mathcal{F}} = \phi^{h/2}_{\mathcal{F}_2} \circ \phi^h_{\mathcal{F}_1} \circ \phi^{h/2}_{\mathcal{F}_2} + \mathcal{O}(h^3). \tag{34}$$

We call (33) the Lie-Trotter splitting method, and (34) the Strang splitting method.

Proposition 9. *The Lie-Trotter splitting method is first-order accurate.*

Proof. Supposing enough regularity of $\mathcal{F}, \mathcal{F}_1, \mathcal{F}_2$, we can Taylor expand around $h = 0$, and write

$$\phi^h_{\mathcal{F}}(\mathbf{x}) = \mathbf{x} + h\mathcal{F}(\mathbf{x}) + \mathcal{O}(h^2),$$

and also

$$\phi^h_{\mathcal{F}_2}\left(\phi^h_{\mathcal{F}_1}(\mathbf{x})\right) = \phi^h_{\mathcal{F}_2}\left(\mathbf{x} + h\mathcal{F}_1(\mathbf{x}) + \mathcal{O}(h^2)\right) = \mathbf{x} + h\mathcal{F}_1(\mathbf{x}) + h\mathcal{F}_2(\mathbf{x}) + \mathcal{O}(h^2).$$

This implies the desired result, since the local error is proportional to h^2. □

Exercise 4. *Repeat the reasoning in the proof above and prove that, assuming enough regularity of the vector fields, the Strang splitting method is second-order accurate.*

Coming back to Hamiltonian systems, let us consider Hamiltonian functions of the following type:

$$H(\mathbf{q}, \mathbf{p}) = K(\mathbf{p}) + U(\mathbf{q}), \ \mathbf{q}, \mathbf{p} \in \mathbb{R}^d. \tag{35}$$

A Hamiltonian as in (35) is said to be separable. The system of Hamiltonian equations associated to (28) write

$$\begin{cases} \dot{\mathbf{q}} = \partial_{\mathbf{p}} K(\mathbf{p}) \\ \dot{\mathbf{p}} = -\partial_{\mathbf{q}} U(\mathbf{q}), \end{cases}$$

and hence we can split the vector field similarly to what we did in (32), i.e., as

$$X_H(\mathbf{q}, \mathbf{p}) = \begin{bmatrix} \partial_{\mathbf{p}} K(\mathbf{p}) \\ -\partial_{\mathbf{q}} U(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} \partial_{\mathbf{p}} K(\mathbf{p}) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -\partial_{\mathbf{q}} U(\mathbf{q}) \end{bmatrix} =: X_K(\mathbf{q}, \mathbf{p}) + X_U(\mathbf{q}, \mathbf{p}).$$

The interesting aspect here is that we can solve exactly the Hamiltonian equations associated to the vector fields X_K and X_H. In fact, for X_K, one has

$$\begin{cases} \dot{\mathbf{q}} = \partial_{\mathbf{p}} K(\mathbf{p}) \\ \dot{\mathbf{p}} = 0 \end{cases}$$

leading to

$$\mathbf{p}(t) = \mathbf{p}_0, \ \mathbf{q}(t) = \mathbf{q}_0 + \int_0^t \partial_{\mathbf{p}} K(\mathbf{p}(s)) ds = \mathbf{q}_0 + t \ \partial_{\mathbf{p}} K(\mathbf{p})|_{\mathbf{p}=\mathbf{p}_0}.$$

Being the exact flow map of a Hamiltonian system symplectic, we can obtain first and second-order accurate symplectic methods just as in (33) and (34), where we set $\mathcal{F}_1 = X_K$ and $\mathcal{F}_2 = X_U$. In the context of symplectic integration, the first order method is called **Symplectic Euler**, and the second order one is called **Störmer-Verlet** or **Leapfrog**.

We remark that the separable expression in (35) is not completely artificial, since the Hamiltonian of mechanical systems written in cartesian coordinates generally takes the form

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^\top M^{-1}\mathbf{p} + U(\mathbf{q}),\ M \in \mathbb{R}^{d\times d},\ M^\top = M,\ M > 0,$$

which is exactly as in (35).

5.3 Symplectic Runge–Kutta methods

Some Runge–Kutta methods are symplectic. To understand what kind of condition we need over the tableau defining the method, we need to work with the variational equation associated to (28). We recall that the variational equation is a differential equation describing the dynamics of the sensitivity matrix of the solution, and writes

$$\frac{d}{dt}S_{\mathbf{x}_0}(t) = \mathbb{J}\nabla^2 H(\mathbf{x}(t))S_{\mathbf{x}_0}(t) \in \mathbb{R}^{2d\times 2d}, \tag{36}$$

where $\mathbf{x}(0) = \mathbf{x}_0$, $S_{\mathbf{x}_0}(t) = \partial_{\mathbf{x}_0}\phi^t_{X_H}(\mathbf{x}_0)$ and hence $S_{\mathbf{x}_0}(0) = I_{2d}$. We also recall that to prove that $\phi^t_{X_H}$ is a symplectic map, we studied the solution of (36). More explicitly, the map $\phi^t_{X_H}$ is symplectic because (36) has the quadratic conserved energies described by

$$S_{\mathbf{x}_0}(t) \in \left\{A \in \mathbb{R}^{2d\times 2d} :\ A^\top \mathbb{J} A = \mathbb{J}\right\},\ t \geq 0.$$

This connection between quadratic energy functions and symplectic maps leads to the following theorem.

Theorem 5 (Symplectic Runge–Kutta methods)**.** *A Runge–Kutta method with tableau* $(A, \mathbf{b}, \mathbf{c})$ *is symplectic if it preserves quadratic first integrals.*

We recall that Runge–Kutta methods preserve quadratic first integrals if $M = BA + A^\top B - \mathbf{b}\mathbf{b}^\top = 0$, where $B = \mathrm{diag}(\mathbf{b})$. To prove Theorem 5 we need the following result.

Proposition 10. *Let* $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$ *be a smooth vector field. For a Runge–Kutta*

method $\varphi^h : \mathbb{R}^n \to \mathbb{R}^n$ *the following diagram commutes:*

$$\begin{array}{rlcrlll}
\dot{\mathbf{x}} & = \mathcal{F}(\mathbf{x}) & \xrightarrow{\textit{differentiation w.r.t. } \mathbf{x}_0} & \dot{\mathbf{x}} & = \mathcal{F}(\mathbf{x}), & \mathbf{x}(0) & = \mathbf{x}_0, \\
\mathbf{x}(0) & = \mathbf{x}_0 & & \dot{S} & = \mathcal{F}'(\mathbf{x})S, & S(0) & = I_n \\
\varphi^h \downarrow & & & & & \downarrow \varphi^h & \\
\mathbf{x}_1 = \varphi^h_{\mathcal{F}}(\mathbf{x}_0) & & \xrightarrow{\textit{differentiation w.r.t. } \mathbf{x}_0} & \mathbf{x}_1 & = \varphi^h_{\mathcal{F}}(\mathbf{x}_0) & & \\
& & & S_1 & = \varphi^h_{\mathcal{F}'}(S_0). & &
\end{array}$$

Proof of Proposition 10. Let us first write down one step of the Runge–Kutta method of tableau $(A, \mathbf{b}, \mathbf{c})$ applied to $\mathcal{F}$:

$$\mathbf{k}_i = \mathbf{x}_0 + h\sum_{j=1}^{s} a_{ij}\mathcal{F}(\mathbf{k}_j), \;\; \mathbf{x}_1 = \mathbf{x}_0 + h\sum_{i=1}^{s} b_i\mathcal{F}(\mathbf{k}_i).$$

Let us now differentiate both terms with respect to $\mathbf{x}_0$:

$$\begin{aligned}
\frac{\partial \mathbf{k}_i}{\partial \mathbf{x}_0} &= I_n + h\sum_{j=1}^{s} a_{ij}\frac{\partial \mathcal{F}(\mathbf{k}_j)}{\partial \mathbf{x}_0} = I_n + h\sum_{j=1}^{s} a_{ij}\mathcal{F}'(\mathbf{k}_j)\frac{\partial \mathbf{k}_j}{\partial \mathbf{x}_0} \\
\frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0} &= I_n + h\sum_{i=1}^{s} b_i\frac{\partial \mathcal{F}(\mathbf{k}_i)}{\partial \mathbf{x}_0} = I_n + h\sum_{i=1}^{s} b_i\mathcal{F}'(\mathbf{k}_i)\frac{\partial \mathbf{k}_i}{\partial \mathbf{x}_0}.
\end{aligned} \tag{37}$$

This is result we get following the blue path. We now need to check if we get the same one following the red one. To verify this, we apply the same Runge–Kutta method to the variational equation coupled with the original ODE

$$\begin{cases} \dot{\mathbf{x}} = \mathcal{F}(\mathbf{x}) \\ \dot{S} = \mathcal{F}'(\mathbf{x})S, \end{cases}$$

and get

$$\begin{aligned}
K_i &= S_0 + h\sum_{j=1}^{s} a_{ij}\mathcal{F}'\left(\mathbf{k}_j\right)K_j \\
S_1 &= S_0 + h\sum_{i=1}^{s} b_i\mathcal{F}'\left(\mathbf{k}_i\right)K_i.
\end{aligned}$$

Recalling that $S_0 = I_n$, we see that

$$\begin{aligned}
K_i &= I_n + h\sum_{j=1}^{s} a_{ij}\mathcal{F}'\left(\mathbf{k}_j\right)K_j \\
S_1 &= I_n + h\sum_{i=1}^{s} b_i\mathcal{F}'\left(\mathbf{k}_i\right)K_i.
\end{aligned} \tag{38}$$

We notice that (37) and (38) coincide, and hence we conclude that the diagram commutes. □

Proof of Theorem 5. The proof is an immediate consequence of Proposition 10. Indeed, applying a Runge–Kutta method that preserves quadratic invariants, by the commutativity of the diagram, we immediately have that

$$S_1^\top \mathbb{J} S_1 = \mathbb{J}$$

and, equivalently, that

$$\left(\frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}\right)^\top \mathbb{J} \left(\frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}\right) = \mathbb{J}.$$

□

5.4 Energy preservation and long-term simulations

Theorem 6. *Let $\dot{\mathbf{x}} = \mathbb{J}\nabla H(\mathbf{x})$ be a Hamiltonian system with Hamiltonian H and with no other conserved quantities than H. Let φ^h be a symplectic and energy-preserving method for the Hamiltonian system, then φ^h reproduces the exact solution up to a time re-parametrisation.*

An interpretation of this result is that it is very hard to build a numerical method which is both symplectic and preserves the Hamiltonian energy H. A proof of this Theorem can be found in [29].

One class of problems where it is possible to simultaneously preserve the symplectic structure of the phase space and the energy are linear Hamiltonian systems. Let us consider a quadratic Hamiltonian energy

$$H(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top C \mathbf{x}.$$

Then the associated Hamiltonian system is linear and writes

$$\dot{\mathbf{x}}(t) = \mathbb{J} C \mathbf{x}.$$

In this case, we know that the Runge–Kutta methods that preserve quadratic first integrals as H, also preserve the symplectic form. Applying these methods, like the implicit midpoint method, will lead to approximate solutions that coincide with the exact solutions up to a time reparametrisation. In other words, the numerical solution will live on the correct phase-space orbit, but such a curve will be covered not at the exact speed.

Even though exact energy conservation is unlikely to be obtained in general, preserving the symplectic form might be enough to do quite well in terms of energy conservation. In fact, symplectic methods exactly conserve a modified Hamiltonian energy, and almost conserve the correct one for exponentially long times.

These results come from the so-called Backward Error Analysis (BEA) of numerical methods. BEA is very useful when the qualitative behaviour of numerical methods is of interest, and when statements over very long time intervals are needed. In contrast to BEA, we recall that a forward error analysis relies on

the study of the local truncation error $\|\varphi_{\mathcal{F}}^h(\mathbf{x}_0) - \phi_{\mathcal{F}}^h(\mathbf{x}_0)\|$, and the global error obtained after n steps with such a method. BEA, instead, looks for a *modified differential equation* $\dot{\mathbf{y}}(t) = \mathcal{F}_h(\mathbf{y}(t))$, such that

$$\mathbf{y}(nh) = \left(\varphi_{\mathcal{F}}^h\right)^h(\mathbf{x}_0) = \underbrace{\varphi_{\mathcal{F}}^h \circ ... \circ \varphi_{\mathcal{F}}^h}_{n \text{ times}}(\mathbf{x}_0),$$

if $\mathbf{y}(0) = \mathbf{x}_0$. The modified vector field $\mathcal{F}_h : \mathbb{R}^d \to \mathbb{R}^d$ is typically expressed in terms of a formal power series as

$$\mathcal{F}_h(\mathbf{y}) = \mathcal{F}(\mathbf{y}) + h\mathcal{F}_2(\mathbf{y}) + h^2\mathcal{F}_3(\mathbf{y}) + ...,$$

meaning that it might not converge, and hence has to be properly truncated. The goal then becomes to study the difference between the two vector fields $\mathcal{F}$ and $\mathcal{F}_h$, which can give much insight into the qualitative behaviour of the numerical solution and the global error.

To find the terms $\mathcal{F}_2, \mathcal{F}_3, ...$, we can Taylor expand the exact solution $\mathbf{y}(t+h)$ of the modified equation around $h = 0$ as follows

$$\begin{aligned}\mathbf{y}(t+h) = \mathbf{y}(t) &+ h\left(\mathcal{F}(\mathbf{y}) + h\mathcal{F}_2(\mathbf{y}) + h^2\mathcal{F}_3(\mathbf{y}) + ...\right) \\ &+ \frac{h^2}{2}\left(\mathcal{F}'(\mathbf{y}) + h\mathcal{F}_2'(\mathbf{y}) + h^2\mathcal{F}_3'(\mathbf{y}) + ...\right) + \mathcal{O}(h^3).\end{aligned}$$

We would like $\mathbf{y}(t+h)$ to coincide with $\varphi_{\mathcal{F}}^h(\mathbf{y}(t))$, which can typically be expressed as

$$\varphi_{\mathcal{F}}^h(\mathbf{y}(t)) = \phi_{\mathcal{F}}^h(\mathbf{y}(t)) + \mathcal{O}(h^{p+1}).$$

Since the Taylor expansion of $\phi_{\mathcal{F}}^h$ only involves $\mathcal{F}$, we see that all the other terms multiplied by h^k, with $k < p$, have to be zero in order for $\varphi_{\mathcal{F}}^h(\mathbf{y}(t)) = \mathbf{y}(t+h)$. This implies that the modified equation must be of the form

$$\mathcal{F}_h(\mathbf{y}) = \mathcal{F}(\mathbf{y}) + h^p\mathcal{F}_{p+1}(\mathbf{y}) + h^{p+1}\mathcal{F}_{p+2}(\mathbf{y}) + \mathcal{O}(h^{p+2}).$$

This analysis immediately allows us to have, via the Gröbner-Alekseev formula that

$$\mathbf{y}(\mathbf{x}) - \mathbf{x}(t) = h^p e_p(t) + h^{p+1}e_{p+1}(t) + \mathcal{O}(h^{p+2}),$$

if $\dot{\mathbf{x}} = \mathcal{F}(\mathbf{x})$, which is the first result we can find out of BEA.

Let us now go back to Hamiltonian systems and symplectic methods, to see what BEA allows us to understand.

Theorem 7 (Theorem 3.1 in Chapter IX.3 [13])**.** *Let φ^h be a symplectic method of order p applied to the Hamiltonian system $\dot{\mathbf{x}} = \mathbb{J}\nabla H(\mathbf{x})$ for a smooth Hamiltonian $H : \mathbb{R}^{2d} \to \mathbb{R}$. Then the modified equation is also Hamiltonian, i.e., there are smooth functions $H_{p+1}, H_{p+2}, ... : \mathbb{R}^{2d} \to \mathbb{R}$ such that*

$$\mathcal{F}_h(\mathbf{y}) = \mathbb{J}\nabla\left(H(\mathbf{y}) + h^p H_{p+1}(\mathbf{y}) + h^{p+1}H_{p+2}(\mathbf{y}) + ...\right).$$

Proof. The proof proceeds by induction. Assume $\mathcal{F}_i(\mathbf{y}) = \mathbb{J}\nabla H_i(\mathbf{y})$ for $i = p+1, ..., r$. We now want to prove the existence of H_{r+1}. Let us consider the truncated modified equation

$$\mathcal{F}_{h,r}(\mathbf{y}) = \mathcal{F}(\mathbf{y}) + h^p \mathcal{F}_{p+1}(\mathbf{y}) + ... + h^{r-1}\mathcal{F}_r(\mathbf{y}).$$

This is known to be a Hamiltonian vector field, with Hamiltonian function $H(\mathbf{y}) + h^p H_{p+1}(\mathbf{y}) + ... + h^{r-1}H_r(\mathbf{y})$. We call ϕ_r^t its flow map, and we notice that

$$\phi^h_{\mathcal{F}_h}(\mathbf{y}_0) = \phi^h_r(\mathbf{y}_0) + h^{r+1}\mathcal{F}_{r+1}(\mathbf{y}_0) + \mathcal{O}(h^{r+2}).$$

The Jacobian of this flow is

$$(\phi^h_{\mathcal{F}_h})'(\mathbf{y}_0) = (\phi^h_r)'(\mathbf{y}_0) + h^{r+1}\mathcal{F}'_{r+1}(\mathbf{y}_0) + \mathcal{O}(h^{r+2}). \tag{39}$$

By induction assumption and symplecticity of the scheme, the red matrices are symplectic. We also know that

$$\phi^h_r(\mathbf{y}_0) = \mathbf{y}_0 + h\mathcal{F}_{h,r}(\mathbf{y}) \implies (\phi^h_r)'(\mathbf{y}_0) = I + \mathcal{O}(h).$$

Let us now check what has to happen for the right-hand side in (39) to be symplectic:

$$\mathbb{J} = (\phi^h_{\mathcal{F}_h})'(\mathbf{y}_0)^\top \mathbb{J} (\phi^h_{\mathcal{F}_h})'(\mathbf{y}_0) = \mathbb{J} + h^{r+1}\left(\mathcal{F}'_{r+1}(\mathbf{y}_0)^\top \mathbb{J} + \mathbb{J}\mathcal{F}'_{r+1}(\mathbf{y}_0)\right) + \mathcal{O}(h^{r+1}).$$

This derivation implies that necessarily

$$\mathbb{J}\mathcal{F}'_{r+1}(\mathbf{y}_0) = -\mathcal{F}'_{r+1}(\mathbf{y}_0)^\top \mathbb{J} = \left(\mathbb{J}\mathcal{F}'_{r+1}(\mathbf{y}_0)\right)^\top,$$

and hence the matrix $\mathbb{J}\mathcal{F}'_{r+1}(\mathbf{y}_0)$ is symmetric. On $\mathbb{R}^{2d}$, the only way vector field has symmetric Jacobian, is that it is a gradient. Thus, Lemma 9 ensures that there exists a smooth function $H_{r+1} : \mathbb{R}^{2d} \to \mathbb{R}$ such that

$$\mathbb{J}\mathcal{F}_{r+1}(\mathbf{y}) = -\nabla H_{r+1}(\mathbf{y})$$

for every $\mathbf{y} \in \mathbb{R}^{2d}$. We conclude that $\mathcal{F}_{r+1}$ is a Hamiltonian vector field since, by multiplying by $\mathbb{J}^\top$ on both sides, we get

$$\mathcal{F}_{r+1}(\mathbf{y}) = \mathbb{J}\nabla H_{r+1}(\mathbf{y}).$$

□

The series expansion for the modified Hamiltonian $\widetilde{H}$ can also be divergent. It is thus meaningful to consider a truncated version

$$\widetilde{H}_N(\mathbf{y}) = H(\mathbf{y}) + h^p H_{p+1}(\mathbf{y}) + ... + h^{N-1}H_N(\mathbf{y}). \tag{40}$$

The following theorem tells us how truncated modified Hamiltonians as the one in (40) behave along the numerical solution. We do not consider the proof of this result, since it would require a few more theorems. The proof can be found in the cited book.

Theorem 8 (Theorem 8.1 in Chapter IX.8 [13]). *Consider a Hamiltonian system with analytic $H : D \to \mathbb{R}$, $D \subset \mathbb{R}^{2d}$, and apply a symplectic method φ^h with step size $h > 0$. If the numerical solution stays in the compact set $K \subset D$, then there exist $h_0 > 0$ and $N = N(h)$ such that*

$$\widetilde{H}_N(\mathbf{y}_n) = \widetilde{H}_N(\mathbf{x}_0) + \mathcal{O}\left(e^{-h_0/2h}\right)$$
$$H(\mathbf{y}_n) = H(\mathbf{x}_0) + \mathcal{O}(h^p)$$

over exponentially long time intervals $nh \leq e^{h_0/2h}$.

In the theorem above, N is the largest integer satisfying $hN \leq h_0$.

5.5 Vanishing gradients and symplectic networks

In this section we discuss the stability of the training procedure of a neural network. We will use the same notation as in Section 3. We focus on a supervised learning task, where the goal is to minimise the mean-squared error loss

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \|\mathcal{N}_\theta(\mathbf{x}_n) - \mathbf{y}_n\|_2^2,$$

given a dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1,\dots,N} \subset \mathbb{R}^d \times \mathbb{R}^c$. The process of training the neural network $\mathcal{N}_\theta$ involves the computation of the gradients of the loss function $\mathcal{L}$ with respect to the network weights θ. To allow for more explicit calculations, let us suppose that the network $\mathcal{N}_\theta$ has $L-$layers and is defined as $\mathcal{N}_\theta = F_{\theta_L} \circ ... \circ F_{\theta_1}$, with $\theta = (\theta_1, ..., \theta_L)$.

We now arrange the weights $\theta_1, ..., \theta_L$ as vectors, and denote with θ_{ij} the $i-$th component of the $j-$th weight θ_j. If the parameters are trained by gradient descent, then the ith component of the parameter vector of the jth layer θ_j is updated as

$$\theta_{ij}^{k+1} = \theta_{ij}^k - \tau_k \partial_{\theta_{ij}} \mathcal{L}(\theta) = \theta_{ij}^k - \frac{\tau_k}{N}\sum_{n=1}^{N} \partial_{\theta_{ij}} \mathcal{L}_n(\theta) \tag{41}$$

where we used the notation $\mathcal{L}_n(\theta) = \|\mathcal{N}_\theta(\mathbf{x}_n) - \mathbf{y}_n\|_2^2$ and τ_k is a step-size, also called learning rate in this context.

We now denote as

$$\mathbf{x}^{j+1} = F_{\theta_j}(\mathbf{x}^j), \quad j = 1, 2, \dots, L,$$

and $\mathbf{x}^1 = \mathbf{x}$. One may notice that especially for the case L is large, the compositional nature of $\mathcal{N}_\theta$ can lead to vanishing gradients. Indeed, by the chain rule, we see that for any n

$$\partial_{\theta_{ij}} \mathcal{L}_n = \langle \partial_{\mathbf{x}_n^{j+1}} \mathcal{L}_n, \partial_{\theta_{ij}} \mathbf{x}_n^{j+1} \rangle = \Big\langle \Big(\prod_{l=j+1}^{L} \partial_{\mathbf{x}_n^l} \mathbf{x}_n^{l+1} \Big) \partial_{\mathbf{x}_n^{L+1}} \mathcal{L}_n, \partial_{\theta_{ij}} \mathbf{x}_n^{j+1} \Big\rangle,$$

where $\langle \mathbf{x}, \mathbf{y} \rangle$ is the Euclidean inner product between two vectors $\mathbf{x}, \mathbf{y}$. Together with the inequality

$$\left\| \prod_{l=j+1}^{L} \partial_{\mathbf{x}_n^l} \mathbf{x}_n^{l+1} \right\|_2 \leq \prod_{l=j+1}^{L} \left\| \partial_{\mathbf{x}_n^l} \mathbf{x}_n^{l+1} \right\|_2, \tag{42}$$

imply that if L is large and the norms on the right of (42) are smaller than 1, then the gradient $\nabla_{\theta_{ij}} \mathcal{L}_n$ will be very small (or converge to zero for $L \to \infty$), hence leading to the impossibility of updating the weights in a meaningful way using gradient information as in (41). This is illustrated in Figure 5 where the vanishing gradient phenomenon leads to a poor classification result. The layers of the network leading to these results are of the form $F_{\theta_j}(\mathbf{x}) = B_j^\top \sigma(A_j \mathbf{x} + \mathbf{b}_j)$, where the weight θ_j is constituted by the entries of A_j and $\mathbf{b}_j$.

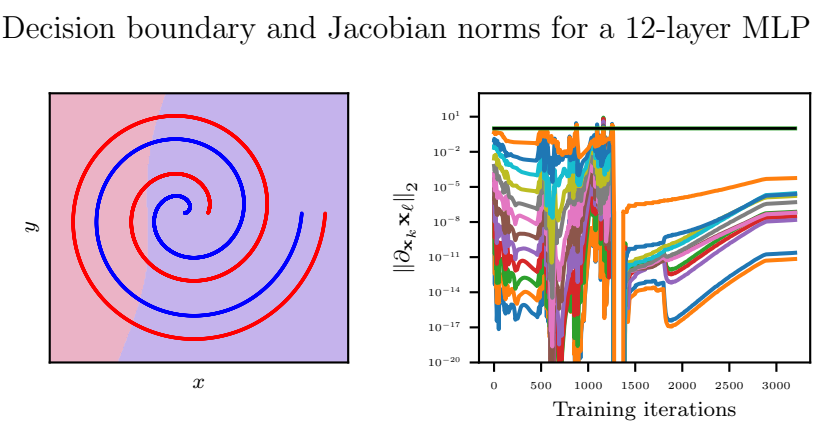

Figure 5: MLP network with 12 layers trained to distinguish red from blue points. On the left, the learned decision boundary cannot accurately separate the points, resulting in a test accuracy of 51%. On the right, the norms of the Jacobians through the training iterates for a fixed data point, showing a severe attenuation of information as we progress through the network, an issue known as the vanishing gradient problem.

Because of this fundamental issue, it is important to suitably design the layers $F_{\theta_1}, \ldots, F_{\theta_L}$, so that $\|\partial_{\mathbf{x}_n^j} \mathbf{x}_n^{j+1}\|_2$ is of moderate size. To do so, we adopt the dynamical systems-based design procedure introduced in Section 3. Let us recall that a non-linear continuously differentiable map $F_{\theta_i} : \mathbb{R}^{d_j} \to \mathbb{R}^{d_{j+1}}$ is symplectic if $d_j = d_{j+1} = 2d$, for some $d \in \mathbb{N}$, and

$$\partial_{\mathbf{x}}(F_{\theta_j}(\mathbf{x}))^\top \mathbb{J} \partial_{\mathbf{x}}(F_{\theta_j}(\mathbf{x})) = \mathbb{J} \tag{43}$$

for every $\mathbf{x} \in \mathbb{R}^{2d}$. We adopt a little abuse of notation here by using the letter d, even though originally $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^c$, so to remain consistent with the notation

used for Hamiltonian systems in this section. So, now, the network will go from $\mathbb{R}^{2d}$ to itself. As an immediate consequence of (43), we have that

$$\|\mathbb{J}\|_2 \leq \left\|\partial_{\mathbf{x}}(F_{\theta_j}(\mathbf{x}))\right\|_2^2 \|\mathbb{J}\|_2 \implies \left\|\partial_{\mathbf{x}}(F_{\theta_j}(\mathbf{x}))\right\|_2 \geq 1 \tag{44}$$

for every $\mathbf{x} \in \mathbb{R}^{2d}$. Using the notation introduced above, we thus can say that if the layers in a network are all symplectic, then it holds true that

$$\left\|\partial_{\mathbf{x}_n^j} \mathbf{x}_n^{j+1}\right\|_2 = \left\|\partial_{\mathbf{x}_n^j}(F_{\theta_j}(\mathbf{x}_n^j))\right\|_2 \geq 1,$$

and hence the vanishing gradient issue is not present anymore in this case.

Now that we have seen the benefits of having symplectic layers to prevent the vanishing gradient issue, we have to understand what are good ways to enforce the symplectic condition over a neural network layer. In principle, we could approach this design problem very generally, trying to find parametrisations of non-linear symplectic maps. However, a much more natural direction to take is exploiting the fact that the flow map of a Hamiltonian system is symplectic. More explicitly, we can design $F_{\theta_j} = \phi_{X_{H_j}}^{h_j}$ for some step size h_j and parametric Hamiltonian function H_j. Given that symplectic methods preserve the symplectic property of the flow, we do not even have to work with the exact flow, but we could replace with a single step of a symplectic integrator.

We define a *symplectic neural network* $\mathcal{N}_\theta$ as a network with j-th layer defined via a single step of a symplectic method ψ^h applied to a parametrised Hamiltonian system with Hamiltonian function H_j. This construction removes the vanishing gradient problem since $\mathcal{N}_\theta$, being the composition of symplectic maps, satisfies (42). We now conclude this section by providing an explicit example of an symplectic neural network, and show that this alternative design strategy leads to considerably improved results on the task in Figure 5.

Theoretically, there is no constraint on how the parametric Hamiltonian functions should be defined. However, some choices might restrict how expressive the network is or lead to network architectures completely different from the ones people are used to. A choice for H_j that allows to recover expressive and commonly used architectures is

$$H_j(\mathbf{x}) = \langle \mathbf{1}, \gamma(A_j \mathbf{x} + \mathbf{a}_j) \rangle, \; A_j \in \mathbb{R}^{2d \times 2d}, \, \mathbf{a}_j \in \mathbb{R}^{2d}, \tag{45}$$

where $\gamma : \mathbb{R} \to \mathbb{R}$ is a differentiable function applied to the entries of its input vector, and $\mathbf{1} \in \mathbb{R}^{2d}$ is a vector of all ones. This parametrisation allows us to get

$$\mathbb{J}\nabla H_j(\mathbf{x}) = \mathbb{J} A_j^\top \sigma(A_j \mathbf{x} + \mathbf{a}_j) \tag{46}$$

where $\gamma' = \sigma$ becomes the activation function of the neural network. After having defined this parametric vector-valued function, one simple option is to define the network layers F_{θ_j} as explicit Euler steps applied to vector fields as in (46) to get

$$F_{\theta_j}(\mathbf{x}) = \mathbf{x} + h \mathbb{J} A_j^\top \sigma(A_j \mathbf{x} + \mathbf{a}_j), \tag{47}$$

which has a similar structure to common ResNets. For example, to get $\sigma = \tanh$, one could set $\gamma = \log \circ \cosh$. The potential issue with defining the layer maps as in (47) is that the explicit Euler method is not symplectic and hence (44) is not guaranteed. Thus, there might still be a vanishing gradient problem. A solution to this issue is provided, for example, by the symplectic Euler method. Let us consider a splitting of the variable $\mathbf{x} \in \mathbb{R}^{2d}$ as $\mathbf{x} = (\mathbf{q}, \mathbf{p})$, $\mathbf{q}, \mathbf{p} \in \mathbb{R}^d$. If the Hamiltonian function H is separable, meaning that $H : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is defined based on two functions $K, U : \mathbb{R}^d \to \mathbb{R}$ as $H(\mathbf{q}, \mathbf{p}) = K(\mathbf{p}) + U(\mathbf{q})$, then the Hamiltonian dynamical system associated to H writes

$$\begin{cases} \dot{\mathbf{q}}(t) = \nabla K(\mathbf{p}(t)) \\ \dot{\mathbf{p}}(t) = -\nabla U(\mathbf{q}(t)) \\ \mathbf{q}(0) = \mathbf{q}_0,\ \mathbf{p}(0) = \mathbf{p}_0. \end{cases}$$

The symplectic Euler method for this problem is explicit and writes

$$\psi^h(\mathbf{q}, \mathbf{p}) = \begin{bmatrix} \widehat{\mathbf{q}} \\ \mathbf{p} - h\nabla U(\widehat{\mathbf{q}}) \end{bmatrix},\ \widehat{\mathbf{q}} = \mathbf{q} + h\nabla K(\mathbf{p}). \tag{48}$$

In order to make the parametric Hamiltonian in (45) separable, we can assume $A_j \in \mathbb{R}^{2d \times 2d}$ has a block structure as

$$A_j = \begin{bmatrix} 0 & B_j \\ C_j & 0 \end{bmatrix},\ B_j, C_j \in \mathbb{R}^{d \times d},$$

and we also write $\mathbf{a}_j = \begin{bmatrix} \mathbf{b}_j^\top & \mathbf{c}_j^\top \end{bmatrix}^\top$, $\mathbf{b}_j, \mathbf{c}_j \in \mathbb{R}^d$. In this way, using the same partitioning $\mathbf{x} = (\mathbf{q}, \mathbf{p})$ as before, we get

$$H_j(\mathbf{q}, \mathbf{p}) = \langle \mathbf{1}, \gamma(B_j\mathbf{p} + \mathbf{b}_j) \rangle + \langle \mathbf{1}, \gamma(C_j\mathbf{q} + \mathbf{c}_j) \rangle =: K_j(\mathbf{p}) + U_j(\mathbf{q}),$$

where $\mathbf{1} \in \mathbb{R}^d$ is the vector with all components equal to 1. To conclude, we can then get an explicitly defined symplectic neural network with j-th layer

$$\psi_j(\mathbf{x}) = \begin{bmatrix} \widehat{\mathbf{q}} \\ \mathbf{p} - hC_j^\top \sigma(C_j\widehat{\mathbf{q}} + \mathbf{c}_j) \end{bmatrix},\ \widehat{\mathbf{q}} := \mathbf{q} + hB_j^\top \sigma(B_j\mathbf{p} + \mathbf{b}_j), \tag{49}$$

which does not suffer from vanishing gradient problems.

In the remaining part of this section, we provide a numerical experiment testing out the architectures we have derived and showing the improvements in terms of vanishing gradient issues provided by using a symplectic network. We consider the problem of classifying into two classes the points in the 2D "Swiss roll" dataset, which can be seen in the top row of Figure 6. The red and blue colours in the figure represent the two classes. We test different network architectures. The symplectic network is denoted as HNN, standing for Hamiltonian Neural Network, a terminology introduced in [10], where these types of networks were introduced. The first is the HNN with layers defined as in (49), the second is a ResNet with layers based on the explicit Euler method and of

the form $F_{\theta_j}(\mathbf{x}) = \mathbf{x} + hB_j^\top \sigma(A_j\mathbf{x} + \mathbf{b}_j)$, and the third is an MLP with layers defined as $F_{\theta_j}(\mathbf{x}) = B_j^\top \sigma(A_j\mathbf{x} + \mathbf{b}_j)$. We consider the HNN and ResNet with $L = 12$ hidden layers of the form above (as we did for the MLP in Figure 5), composed with a final linear layer to adapt the network to the output dimensionality which, in this case, is two. The MLP is also considered for the case of $L = 2$ hidden layers. The dataset is embedded in a higher dimensional space of dimension four in the following way $(x_1, x_2) \mapsto (x_1, 0, x_2, 0)$. The network layers then preserve this intermediate fixed dimension.

In Figure 6, we can see that the ResNet and HNN models both perform accurately on this simple task, leading to a 100% classification accuracy over a test set. Instead, the MLP with 12 layers does not train appropriately, as we saw in Figure 5, leading to a classification that is only slightly better than chance. On the other hand, the MLP with two layers trains slightly better, leading to around 80% accuracy. These four models have been chosen to illustrate the issue of having vanishing gradients and, consequently, not being able to train the network. In the bottom row of Figure 6 we plot the norms of the Jacobian matrices of the last hidden layer with respect to the previous ones throughout the training iterations. For each of the four models, a fixed test data point has been used to evaluate these Jacobian matrices. We see that the ResNet and HNN models lead to well-behaved Jacobians. On the other hand, the MLP model has vanishing gradient issues, that lead to the impossibility of training the model with 12 layers, whereas these issues do not arise when training a network with just two hidden layers. While the HNN is built so that the norm of the Jacobian is never smaller than one, as can be seen in the plot, the skip connections in the ResNet naturally lead to stable behaviour. This is not surprising since residual connections were introduced precisely to allow the training of deeper networks.

6 Non-expansive numerical methods

6.1 Non-expansive dynamical systems

In this section we will work with the ℓ^2 norm, but the theory of non-expansive dynamical systems extends to other norms, see [4]. We say that a vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ is non-expansive if its flow map $\phi_{\mathcal{F}}^t : \mathbb{R}^d \to \mathbb{R}^d$ is non-expansive for every time $t \geq 0$, i.e., $\|\phi_{\mathcal{F}}^t(\mathbf{x}) - \phi_{\mathcal{F}}^t(\mathbf{y})\|_2 \leq \|\mathbf{x} - \mathbf{y}\|_2$.

Since the flow map is not usually accessible, this definition is not so practical. However, supposing $\mathcal{F}$ is sufficiently smooth, we can get a much more practical characterisation of non-expansive dynamical systems by Taylor expansion. Indeed, let us fix a small enough scalar h and consider

$$\phi_{\mathcal{F}}^{t+h}(\mathbf{x}) = \phi_{\mathcal{F}}^t(\mathbf{x}) + h\mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{x})) + \mathcal{O}(h^2),$$
$$\phi_{\mathcal{F}}^{t+h}(\mathbf{y}) = \phi_{\mathcal{F}}^t(\mathbf{y}) + h\mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{y})) + \mathcal{O}(h^2),$$

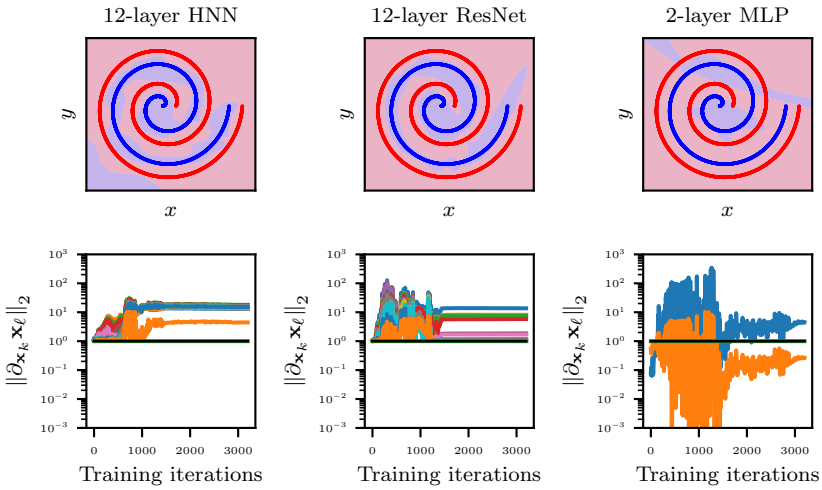

Figure 6: A comparison of a 12-layer HNN, a 12-layer ResNet and a 2-layer MLP on the "Swiss roll" dataset, as previously considered for a 12-layer MLP in Figure 5. Both the HNN and ResNet attain a test accuracy of 100%, while the 2-layer MLP has a test accuracy of 79.23%. Note that the Jacobian norms behave much less extremely than they did for the 12-layer MLP in Figure 5, resulting in networks that train better.

for an arbitrary pair of points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Then, we see that

$$\begin{aligned}\left\|\phi_{\mathcal{F}}^{t+h}(\mathbf{y}) - \phi_{\mathcal{F}}^{t+h}(\mathbf{x})\right\|_2^2 &= \left\|\phi_{\mathcal{F}}^{t}(\mathbf{y}) - \phi_{\mathcal{F}}^{t}(\mathbf{x})\right\|_2^2 \\ &\quad + 2h\langle \mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{y})) - \mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{x})), \phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{x})\rangle + \mathcal{O}(h^2),\end{aligned}$$

and hence

$$\begin{aligned}\frac{d}{dt}\left\|\phi_{\mathcal{F}}^{t}(\mathbf{y}) - \phi_{\mathcal{F}}^{t}(\mathbf{y})\right\|_2^2 &= \lim_{h\to 0} \frac{\left\|\phi_{\mathcal{F}}^{t+h}(\mathbf{y}) - \phi_{\mathcal{F}}^{t+h}(\mathbf{y})\right\|_2^2 - \|\phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{x})\|_2^2}{h} \\ &= 2\langle \mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{y})) - \mathcal{F}(\phi_{\mathcal{F}}^t(\mathbf{x})), \phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{x})\rangle.\end{aligned}$$

This derivation implies that if for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ one has

$$\langle \mathcal{F}(\mathbf{y}) - \mathcal{F}(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle \leq \nu \|\mathbf{y} - \mathbf{x}\|_2^2, \tag{50}$$

then it follows

$$\frac{d}{dt}\left\|\phi_{\mathcal{F}}^{t}(\mathbf{y}) - \phi_{\mathcal{F}}^{t}(\mathbf{y})\right\|_2^2 \leq 2\nu \|\phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{x})\|_2^2. \tag{51}$$

If we define $g(t) := \|\phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{y})\|_2^2$ and multiply both sides of (51) by the positive scalar $e^{-2\nu t}$, we see that

$$\frac{d}{dt}\left(e^{-2\nu t} g(t)\right) = e^{-2\nu t}\dot{g}(t) - 2\nu e^{-2\nu t} g(t) \leq 0.$$

We can thus conclude that $e^{-2\nu t}g(t)$ is monotonically non-increasing, so that $e^{-2\nu t}g(t) \leq g(0)$, and hence we have

$$\left\|\phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{x})\right\|_2 \leq e^{\nu t} \left\|\mathbf{y} - \mathbf{x}\right\|_2 \tag{52}$$

for every $t \geq 0$, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. We remark that the distance between any pair $\mathbf{x}$ and $\mathbf{y}$ is not expanded by the flow map $\phi_{\mathcal{F}}^t$ for $t \geq 0$ whenever $\nu \leq 0$. This analysis motivates the introduction of the following definition.

Definition 7 (One-sided Lipschitz inequality)**.** *The vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ is one-sided Lipschitz continuous if it satisfies (50) for a scalar $\nu \in \mathbb{R}$ and any pair $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. $\mathcal{F}$ is a non-expansive vector field if (50) holds for a $\nu \leq 0$. If ν can be taken strictly negative it is contractive.*

Before moving on, we remark that contractivity can be a pretty restrictive assumption on the dynamics. For example, one can see that a contractive dynamical system has to have a unique asymptotically stable equilibrium point. To verify this behaviour, let $\phi_{\mathcal{F}}^1 : \mathbb{R}^d \to \mathbb{R}^d$ be the time-1 flow of the contractive vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$. Banach's fixed point theorem guarantees that $\phi_{\mathcal{F}}^1$ admits a unique fixed point $\mathbf{x}^* \in \mathbb{R}^d$ such that $\phi_{\mathcal{F}}^1(\mathbf{x}^*) = \mathbf{x}^*$. In case this is an equilibrium point of $\mathcal{F}$, it has to be asymptotically stable since for any $\mathbf{x} \in \mathbb{R}^d$ we have

$$\lim_{t\to+\infty} \left\|\phi_{\mathcal{F}}^t(\mathbf{x}) - \mathbf{x}^*\right\|_2 = \lim_{t\to+\infty} \left\|\phi_{\mathcal{F}}^t(\mathbf{x}) - \phi_{\mathcal{F}}^t(\mathbf{x}^*)\right\|_2 \leq \lim_{t\to+\infty} e^{\nu t}\|\mathbf{x} - \mathbf{x}^*\|_2 = 0.$$

If $\mathbf{x}^*$ is not an equilibrium point, then it has to be part of a periodic orbit of period 1. This is impossible since the existence of such a periodic orbit would lead to infinitely many fixed points for ϕ_X^1, allowing us to conclude that in fact $\mathbf{x}^*$ must be an equilibrium point.

Even though the condition in (50) is more practical than where we started from, it can sometimes be hard to verify. For continuously differentiable vector fields, one can simplify the condition to an equivalent characterisation based on the Jacobian matrix $\partial_{\mathbf{x}}\mathcal{F}(\mathbf{x}) \in \mathbb{R}^{d\times d}$. Indeed, by the mean value theorem, for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, there is $\mathbf{z} = s\mathbf{x} + (1-s)\mathbf{y}$, for some $s \in (0,1)$, such that

$$\mathcal{F}(\mathbf{y}) - \mathcal{F}(\mathbf{x}) = \partial_{\mathbf{x}}\mathcal{F}(\mathbf{z})(\mathbf{y} - \mathbf{x}).$$

Thus, (50) can be formulated as an equivalent condition

$$\sup_{\mathbf{x}\in\mathbb{R}^d, \mathbf{v}\in\mathbb{R}^d\setminus\{\mathbf{0}\}} \frac{\langle \partial_{\mathbf{x}}\mathcal{F}(\mathbf{x})\mathbf{v}, \mathbf{v}\rangle}{\|\mathbf{v}\|_2^2} \leq \nu,$$

or, equivalently, as

$$\sup_{\mathbf{x}\in\mathbb{R}^d} \lambda_{\max}\left(\frac{\partial_{\mathbf{x}}\mathcal{F}(\mathbf{x})^\top + \partial_{\mathbf{x}}\mathcal{F}(\mathbf{x})}{2}\right) \leq \nu, \tag{53}$$

where $\lambda_{\max}(A)$ is the maximum eigenvalue of some matrix A.

To get some more familiarity with one-sided Lipschitz continuous functions, we now focus on a few results that connect it with the more popular notion of Lipschitz continuity.

We can start with scalar vector fields, so $\mathcal{F} : \mathbb{R} \to \mathbb{R}$. Let us start with the one-dimensional case. A continuously differentiable function (vector field) $\mathcal{F} : \mathbb{R} \to \mathbb{R}$ is one-sided Lipschitz continuous with constant $\leq \nu$ if $\mathcal{F}'(x) \leq \nu$ for every $x \in \mathbb{R}$. In particular, it defines a non-expansive dynamics if $\mathcal{F}$ is non-increasing.

Lemma 10. *An $L-$Lipschitz continuous function $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ is also one-sided Lipschitz continuous with constant greater or equal than L.*

Proof. The proof is a simple consequence of Cauchy-Schwartz inequality. Let us consider two arbitrary points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. By Lipschitz continuity we have

$$\langle \mathcal{F}(\mathbf{y}) - \mathcal{F}(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \leq \|\mathcal{F}(\mathbf{y}) - \mathcal{F}(\mathbf{x})\|_2 \|\mathbf{y} - \mathbf{x}\|_2 \leq L \|\mathbf{y} - \mathbf{x}\|_2^2.$$

which concludes the proof. □

Lemma 11. *Not all the one-sided Lipschitz continuous functions are Lipschitz continuous.*

Proof. Thinking to the one-dimensional case, we just need to find a function $\mathcal{F} : \mathbb{R} \to \mathbb{R}$ which is non-increasing but which is not Lipschitz continuous. Examples could be $\mathcal{F}(x) = -x^{2n+1}$, $n \in \mathbb{N}$, or $\mathcal{F}(x) = e^{-x^2}$. □

Before moving to numerical methods, we focus on a particular class of $d-$dimensional vector fields that lead to non-expansive and potentially contractive dynamics. We will use these systems to build Lipschitz-constrained neural networks in the last part of the course. These are negative gradient flows of convex potentials. By results in convex analysis, it is relatively immediate to verify the properties we have just derived for these systems.

Theorem 9 (Theorem 2.1.3 in [22])**.** *A continuously differentiable function $V : \mathbb{R}^d \to \mathbb{R}$ is convex on $\mathbb{R}^d$ if and only if for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ we have*

$$\langle \nabla V(\mathbf{y}) - \nabla V(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \geq 0.$$

This theorem implies that $\mathcal{F}(\mathbf{x}) = -\nabla V(\mathbf{x})$, for $V : \mathbb{R}^d \to \mathbb{R}$ a convex and continuously differentiable function, is a non-expansive vector field. Under the assumption that V is twice continuously differentiable, one could equivalently verify this property using (53), since the Hessian of a convex function is symmetric positive semi-definite (see [22, Theorem 2.1.4]), and hence $\lambda_{\max}(\partial_{\mathbf{x}} \mathcal{F}(\mathbf{x})) = \lambda_{\max}(-\partial^2_{\mathbf{xx}} V(\mathbf{x})) \leq 0$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$.

Definition 8 (L-smooth function)**.** *A convex and continuously differentiable function $V : \mathbb{R}^d \to \mathbb{R}$ is $L-$smooth if its gradient is $L-$Lipschitz continuous, i.e.,*

$$\|\nabla V(\mathbf{y}) - \nabla V(\mathbf{x})\|_2 \leq L \|\mathbf{y} - \mathbf{x}\|_2, \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d.$$

The concept of L-smoothness from convex analysis is of particular importance to us for the applications to deep learning, since this is what will allow us to derive step size constraints for numerical discretisations of non-expansive flows. An important result in convex analysis, the so-called *Baillon–Haddad theorem*, tells us that this holds if and only if the following inequality holds for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

$$\langle \nabla V(\mathbf{x}) - \nabla V(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \frac{1}{L} \|\nabla V(\mathbf{x}) - V(\mathbf{y})\|_2^2. \tag{54}$$

For other equivalent characterisations of $L-$smoothness, see [22, Theorem 2.1.5]. These dynamics based on negative gradient flows of convex continuously differentiable functions are non-expansive. We can similarly obtain contractive dynamics, by restricting even further the potential function V to continuously differentiable and strongly convex functions.

Definition 9 ($\mu-$strongly convex functions)**.** *A continuously differentiable function $V : \mathbb{R}^d \to \mathbb{R}$ is called $\mu-$strongly convex on $\mathbb{R}^d$ if there exists a constant $\mu > 0$ such that for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ we have*

$$\mathcal{F}(\mathbf{y}) \geq \mathcal{F}(\mathbf{x}) + \langle \nabla \mathcal{F}(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\mu}{2} \|\mathbf{y} - \mathbf{x}\|_2^2 .$$

Theorem 10 (Theorem 2.1.9 [22])**.** *The continuously differentiable function $V : \mathbb{R}^d \to \mathbb{R}$ is $\mu-$strongly convex if and only if*

$$\langle \nabla V(\mathbf{y}) - \nabla V(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \geq \mu \|\mathbf{y} - \mathbf{x}\|_2^2 \tag{55}$$

for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$.

(55) thus guarantees that for any continuously differentiable $\mu-$strongly convex potential $V : \mathbb{R}^d \to \mathbb{R}$, the vector field $\mathcal{F}(\mathbf{x}) = -\nabla V(\mathbf{x})$ is contractive since $\mathcal{F}$ is one-sided Lipschitz continuous with constant $\nu = -\mu < 0$.

6.2 Unconditionally non-linearly stable methods

The non-expansivity property of a numerical method is a property one can use to ensure the non-linear stability of the method. This notion provides an alternative stability analysis than the A-stability we saw in Section 1. We also recall that no explicit Runge–Kutta method is A-stable, since for those methods the function R is a polynomial, and hence the stability region S is bounded.

There have been several other notions of numerical stability introduced in the literature. The one that refers to non-expansive/contractive vector fields in a norm $\|\cdot\|$ generated by an inner product $\langle \cdot, \cdot \rangle$ is called B-stability.

Definition 10 (B-stable method)**.** *A numerical method $\varphi^h : \mathbb{R}^d \to \mathbb{R}^d$ is B-stable if when applied to any vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ which satisfies*

$$\left\|\phi_{\mathcal{F}}^t(\mathbf{y}) - \phi_{\mathcal{F}}^t(\mathbf{x})\right\| \leq \|\mathbf{y} - \mathbf{x}\|, \ \forall t \geq 0, \tag{56}$$

for a norm $\|\cdot\|$ generated by an inner product $\langle \cdot, \cdot \rangle$, we have

$$\left\|\varphi_{\mathcal{F}}^h(\mathbf{y}) - \varphi_{\mathcal{F}}^h(\mathbf{x})\right\| \leq \|\mathbf{y} - \mathbf{x}\|, \ \forall h > 0.$$

This definition tells us that a method φ^h is B-stable if it preserves the non-expansive nature of the dynamics at a discrete level.

Restricting to inner product norms allows us to have no barriers on the order of the methods we declare to be stable or non-expansive. If we were to include also norms like ℓ^1 or ℓ^∞, then there would be a maximal reachable order of 1, see [27].

The conditions we have used to define B-stability are not so practical. We now provide a very operative procedure to decide if a Runge–Kutta method is B-stable or not.

Proposition 11 (B-stable Runge–Kutta methods)**.** *A Runge–Kutta method with tableau $(A, \mathbf{b}, \mathbf{c})$ is B-stable if and only if said $B = \mathrm{diag}(\mathbf{b})$, and $M = BA + A^\top B - \mathbf{b}\mathbf{b}^\top$, one has $B \geq 0$ and $M \geq 0$.*

Proof. Let us consider $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ that satisfies

$$\langle \mathcal{F}(\mathbf{y}) - \mathcal{F}(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \leq 0$$

for a fixed inner product $\langle \cdot, \cdot \rangle : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. We consider two generic initial conditions $\mathbf{x}_0, \mathbf{y}_0 \in \mathbb{R}^d$, and compute one update with the Runge–Kutta method having tableau $(A, \mathbf{b}, \mathbf{c})$ and step $h > 0$:

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x}_0 + h \sum_{i=1}^s b_i \mathcal{F}(\mathbf{k}_i), \;\; \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathcal{F}(\mathbf{h}_i), \\
\mathbf{k}_i &= \mathbf{x}_0 + h \sum_{j=1}^s a_{ij} \mathcal{F}(\mathbf{k}_j), \;\; \mathbf{h}_i = \mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathcal{F}(\mathbf{h}_j).
\end{aligned}$$

We then introduce the notation

$$\begin{aligned}
\delta \mathbf{x}_r &:= \mathbf{y}_r - \mathbf{x}_r, \;\; r \in \{0, 1\}, \\
\delta \mathcal{F}_i &:= \mathcal{F}(\mathbf{k}_i) - \mathcal{F}(\mathbf{h}_i), \;\; i = 1, ..., s, \\
\delta \mathbf{k}_i. &= \mathbf{k}_i - \mathbf{h}_i, \;\; i = 1, ..., s.
\end{aligned}$$

We now expand the norm of the difference after the first update to compare it to the initial norm:

$$\begin{aligned}
\|\delta \mathbf{x}_1\|^2 = \langle \delta \mathbf{x}_1, \delta \mathbf{x}_1 \rangle = \|\delta \mathbf{x}_0\|^2 + h^2 \sum_{i,j=1}^s b_i b_j \langle \delta \mathcal{F}_i, \delta \mathcal{F}_j \rangle \\
+ 2h \sum_{i=1}^s b_i \langle \delta \mathbf{x}_0, \delta \mathcal{F}_i \rangle. \qquad (57)
\end{aligned}$$

As in the proof for Runge–Kutta methods preserving quadratic first integrals, we compute $\delta \mathbf{x}_0$ in s different ways as

$$\delta \mathbf{x}_0 = \delta \mathbf{k}_i - h \sum_{j=1}^s a_{ij} \delta \mathcal{F}_j, \;\; i = 1, ..., s. \qquad (58)$$

Replacing (58) into (57), we get

$$\|\delta\mathbf{x}_1\|^2 = \langle\delta\mathbf{x}_1, \delta\mathbf{x}_1\rangle = \|\delta\mathbf{x}_0\|^2 + h^2 \sum_{i,j=1}^{s} b_i b_j \langle\delta\mathcal{F}_i, \delta\mathcal{F}_j\rangle$$
$$+ \underbrace{2h \sum_{i=1}^{s} b_i \langle\delta\mathbf{k}_i, \delta\mathcal{F}_i\rangle}_{\leq 0, \text{ using } b_i \geq 0} - 2h^2 \sum_{i,j=1}^{s} b_i a_{ij} \langle\delta\mathcal{F}_i, \delta\mathcal{F}_j\rangle.$$

Similarly to the proof for quadratic first integrals, we can rewrite the last term in the previous equation by simmetry of inner products, and end up with

$$\|\delta\mathbf{x}_1\|^2 - \|\delta\mathbf{x}_0\|^2 = -h^2 \sum_{i,j=1}^{s} m_{ij} \langle\delta\mathcal{F}_i, \delta\mathcal{F}_j\rangle, \tag{59}$$

where m_{ij} is an entry of $M \in \mathbb{R}^{s\times s}$. From this relation we can conclude by using the positive semi-definiteness of M. A way to formalise this is to define

$$\delta\mathcal{F} := \begin{bmatrix} \delta\mathcal{F}_1 \\ \delta\mathcal{F}_2 \\ \vdots \\ \delta\mathcal{F}_s \end{bmatrix}, \quad \widetilde{M} = M \otimes I_d,$$

with $\otimes$ which is the Kronecker product, and rewrite the right-hand side of (59) as

$$-h^2 \delta X^\top \widetilde{M} \delta X,$$

which is non-positive since if $M \geq 0$, so is $\widetilde{M}$. □

Remark 2. *B-stability implies A-stability since one can consider the system*

$$\dot{\mathbf{x}} = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \mathbf{x},$$

equivalent to $\dot{x} = \lambda x$, $\lambda = \alpha + i\beta$, where if $\alpha = \mathrm{Re}(\lambda) < 0$ we have a non-expansive vector field. Thus, if a method φ^h is non-expansive, or non-linearly stable, it is also A-stable, or linearly stable. It follows that explicit methods can not be B-stable.

Similarly to what we have seen in Section 4, Gauss-Legendre collocation methods are B-stable. We state this result and prove it without checking the condition above, but just by using the definition of collocation methods.

Lemma 12. *All Gauss-Legendre collocation methods are B-stable.*

Proof. Let $\mathbf{u}(t)$ and $\widetilde{\mathbf{u}}(t)$ be the polynomial approximations of the solutions provided by the collocation method over the time interval $[t_n, t_{n+1} = t_n + h]$ with $\mathbf{u}(t_n) = \mathbf{x}_n$ and $\widetilde{\mathbf{u}}(t_n) = \mathbf{y}_n$. We suppose the collocation method is of s stages.

Let us consider the squared distance function $m(t) = \|\mathbf{u}(t) - \widetilde{\mathbf{u}}(t)\|^2 \in \mathbb{P}^{2s}(\mathbb{R})$. At the collocation nodes $s_i = t_n + c_i h$ we have

$$m'(s_i) = 2\langle \mathcal{F}(\mathbf{u}(s_i)) - \mathcal{F}(\widetilde{\mathbf{u}}(s_i)), \mathbf{u}(s_i) - \widetilde{\mathbf{u}}(s_i)\rangle \leq 0.$$

We can then conclude with the following derivation:

$$\begin{aligned}\|\mathbf{x}_{n+1} - \mathbf{y}_{n+1}\|^2 = m(t_{n+1}) &= m(t_n) + \int_{t_n}^{t_{n+1}} m'(t)dt \\ &= \|\mathbf{x}_n - \mathbf{y}_n\|^2 + h\sum_{i=1}^{s} b_i m'(s_i) \leq \|\mathbf{x}_n - \mathbf{y}_n\|^2\end{aligned}$$

since $m'(t) \in \mathbb{P}^{2s-1}(\mathbb{R})$ is exactly integrated by the Gauss-Legendre quadrature with s nodes.

□

6.3 Conditionally non-expansive methods

In the previous subsection, we derived conditions for a Runge–Kutta method to be non-linearly stable, i.e., non-expansive, regardless of the choice of the time step $h > 0$. This, however, led to the exclusion of several numerical methods, for example all the explicit ones. To motivate this further discussion, let us consider a simple example. This is the linear negative gradient flow of the potential function $V(\mathbf{x}) = \|\mathbf{x}\|_2^2/2$, which has equations

$$\dot{\mathbf{x}}(t) = -\mathbf{x}(t).$$

The analytical solution of this problem, is $\mathbf{x}(t) = e^{-t}\mathbf{x}(0)$, which is clearly non-expansive, since

$$\|\mathbf{y}(t) - \mathbf{x}(t)\|_2 = \|e^{-t}(\mathbf{y}(0) - \mathbf{x}(0))\|_2 = e^{-t}\|\mathbf{y}(0) - \mathbf{x}(0)\|_2 < \|\mathbf{y}(0) - \mathbf{x}(0)\|_2.$$

Let us focus on the simplest Runge–Kutta method, explicit Euler, which leads to the update

$$\mathbf{x}_{n+1} \ = \mathbf{x}_n - h\mathbf{x}_n = (1-h)\mathbf{x}_n.$$

We do not expect this map to be $1-$Lipschitz for every step $h > 0$, since we know the method is not B-stable. However, we notice that

$$\|\mathbf{y}_{n+1} - \mathbf{x}_{n+1}\|_2 = \|(1-h)(\mathbf{y}_n - \mathbf{x}_n)\|_2 = |1-h| \cdot \|\mathbf{y}_n - \mathbf{x}_n\|_2,$$

which is not greater than $\|\mathbf{y}_n - \mathbf{x}_n\|$ when $0 \leq h \leq 2$. This suggests that there might be methods which, despite not being B-stable, can be non-expansive for small enough time steps.

The theory of circle-contractivity developed in [8] provides a generalisation of the analysis done in the simple example above. We will not go in the details of this theory in this course, but will focus on a particular class of dynamical systems: negative gradient flows.

We have already seen this family of dynamical systems at the introduction of this section, but we now want to focus on how to restrict the step size in order to make the explicit Euler method non-expansive. Let us consider an $L-$smooth function $V : \mathbb{R}^d \to \mathbb{R}$, defining the dynamics

$$\dot{\mathbf{x}}(t) = -\nabla V(\mathbf{x}(t)) =: \mathcal{F}(\mathbf{x}(t)). \tag{60}$$

We have shown before that the vector field $\mathcal{F}$ is non-expansive, since, for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$-\langle \nabla V(\mathbf{y}) - \nabla V(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \leq -\frac{1}{L} \|\nabla V(\mathbf{y}) - \nabla V(\mathbf{x})\|_2^2 \leq 0.$$

Let us now compare the updates we get with the explicit Euler method integrating (60) starting from two initial points $\mathbf{x}_n, \mathbf{y}_n \in \mathbb{R}^d$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - h\nabla V(\mathbf{x}_n), \ \mathbf{y}_{n+1} = \mathbf{y}_n - h\nabla V(\mathbf{y}_n).$$

We see that

$$\begin{aligned} \|\mathbf{y}_{n+1} - \mathbf{x}_{n+1}\|_2^2 &= \|\mathbf{y}_n - \mathbf{x}_n\|_2^2 - 2h\langle \nabla V(\mathbf{y}_n) - \nabla V(\mathbf{x}_n), \mathbf{y}_n - \mathbf{x}_n \rangle \\ &\quad + h^2 \|\nabla V(\mathbf{y}_n) - \nabla V(\mathbf{x}_n)\|_2^2 \\ &\leq \|\mathbf{y}_n - \mathbf{x}_n\|_2^2 + \left(h^2 - \frac{2h}{L} \right) \|\nabla V(\mathbf{y}_n) - \nabla V(\mathbf{x}_n)\|_2^2 \\ &\leq \|\mathbf{y}_n - \mathbf{x}_n\|_2^2 \end{aligned}$$

if $h \leq 2/L$. Thus, the explicit Euler method applied to (60) is non-expansive if the time step h belongs to the interval $[0, 2/L]$, where L is the Lipschitz constant of $\mathcal{F}(\mathbf{x}) = -\nabla V(\mathbf{x})$. We remark that this is the exact same restriction we obtained for the specific case $V(\mathbf{x}) = \|\mathbf{x}\|_2^2/2$ in the example above, where $L = 1$ and we got $h \in [0, 2]$.

This approach to find non-expansive methods, which could be generalised to other dynamical systems and Runge–Kutta methods, is very appealing when designing neural networks based on dynamical systems. In fact, in the machine learning context, one typically has to work with very high-dimensional problems and using implicit solvers to define the ResNet layers can become prohibitively expensive. For this reason, a condition like the one above, which is cheap to evaluate, proves useful when building networks with Lipschitz constant upper bounded by 1, as we will see in the next section.

6.4 Lipschitz-constrained neural networks

Despite the great successes of deep learning in all areas of science and technology, most off-the-shelf neural networks show instabilities: tiny perturbations of the input lead to dramatic consequences in the output. These instabilities may be exploited in adversarial attacks [11] and are particularly problematic in high-risk applications like medical imaging [1]. In this section we will study stable neural network architectures based on dynamical systems.

The simplest notion of stability is *Lipschitz continuity*. We call a neural network $\mathcal{N}_\theta$ *stable* if it is ℓ-Lipschitz continuous, i.e., there exists a constant $\ell \geq 0$ such that

$$\|\mathcal{N}_\theta(\mathbf{x}) - \mathcal{N}_\theta(\mathbf{y})\|_2 \leq \ell \|\mathbf{x} - \mathbf{y}\|_2. \tag{61}$$

Due to the layered structure of deep neural networks, we can relate the Lipschitz constant ℓ to the Lipschitz constants of the individual layers ℓ_i as $\ell \leq \Pi_{i=1}^{\ell} \ell_i$ [12]. It was argued in [5] that such an estimate is pessimistic and hinders practical usefulness.

Stability of neural networks is desirable in many contexts such as the stable solution to inverse problems, classification that is robust to errors, efficient training of deep neural networks. It also used in the context of generative models and frequently used for (Wasserstein) GANs to regularise the discriminator [2], for instance via spectral normalisation [21].

6.4.1 The problem of robust classification

We now consider the problem of classifying the points in a compact set $\mathcal{X} \subset \mathbb{R}^d$ into some number $c \in \mathbb{N}$ of classes. A standard approach to this problem is to model the classifier using a neural network $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^c$ and predict the class for an input $\mathbf{x}$ as $\operatorname{argmax}_{k \in \{1,\dots,c\}} \mathcal{N}_\theta(\mathbf{x})_k$. The outputs of $\mathcal{N}_\theta$ may be interpreted as *logits*, meaning that $\exp(\mathcal{N}_\theta(\mathbf{x})_k) / \sum_{i=1}^{c} \exp(\mathcal{N}_\theta(\mathbf{x})_i)$ is treated as a probability that $\mathbf{x}$ is of class k. Regardless of the interpretation, Lipschitz continuity of $\mathcal{N}_\theta$ can be used to *certify the robustness* of the predictions.

Associated with $\mathcal{N}_\theta$, we can define the predicted class $\hat{k} : \mathcal{X} \to \{1, \dots, c\}$ as $\hat{k}(\mathbf{x}) = \operatorname{argmax}_{k \in \{1,\dots,c\}} \mathcal{N}_\theta(\mathbf{x})_k$ and the *margin* $m : \mathbb{R}^d \to \mathbb{R}$ as

$$m(\mathbf{x}) = \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \max_{k \in \{1,\dots,c\} \setminus \{\hat{k}(\mathbf{x})\}} \mathcal{N}_\theta(\mathbf{x})_k.$$

One can think of the margin as some wiggle room in the accuracy of the prediction. Even if the predicted value shrinks up to the margin, the prediction of the model is still the same. If the classifier is stable, then this means that there can be errors in the data which do not alter the classification result.

Proposition 12. *Let us consider an ℓ-Lipschitz continuous neural network $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^c$. Then, for every $\mathbf{y} \in \mathbb{R}^d$ with*

$$\|\mathbf{x} - \mathbf{y}\|_2 < \frac{m(\mathbf{x})}{\sqrt{2}\ell},$$

the prediction provided by the network would be of the same class as $\mathbf{x}$, i.e., $\hat{k}(\mathbf{x}) = \hat{k}(\mathbf{y})$.

This result can be used to give robustness guarantees for existing classifiers, but can also be used to motivate the training of robust classifiers, as long as we can upper bound the Lipschitz constant of a neural network.

To prove this proposition, we first need the following lemma.

Lemma 13. *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^c$ be two vectors. Then, it holds*

$$\left| \max_{k \neq \hat{k}} x_k - \max_{k \neq \hat{k}} y_k \right| \leq \max_{k \neq \hat{k}} |x_k - y_k| .$$

Proof. Without loss of generality, we can assume $\max_{k \neq \hat{k}} x_k \geq \max_{k \neq \hat{k}} y_k$. We define $j = \operatorname{argmax}_{k \neq \hat{k}} x_k$. It then follows

$$\begin{aligned}
\left| \max_{k \neq \hat{k}} x_k - \max_{k \neq \hat{k}} y_k \right| &= \max_{k \neq \hat{k}} x_k - \max_{k \neq \hat{k}} y_k \\
&= x_j - \max_{k \neq \hat{k}} y_k \\
&\leq x_j - y_j \text{ since } y_j \leq \max_{k \neq \hat{k}} y_k \\
&\leq \max_{k \neq \hat{k}} |x_k - y_k| .
\end{aligned}$$

□

Proof of Proposition 12. We start by considering the target margin, and show it is non-negative:

$$\begin{aligned}
&\mathcal{N}_\theta(\mathbf{y})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{y})_k \\
&= \mathcal{N}_\theta(\mathbf{y})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{y})_k + \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} \\
&+ \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k \\
&= \mathcal{N}_\theta(\mathbf{y})_{\hat{k}(\mathbf{x})} - \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} + \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k \\
&- \left(\max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{y})_k - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k \right) \\
&\geq - \left| \mathcal{N}_\theta(\mathbf{y})_{\hat{k}(\mathbf{x})} - \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} \right| + \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k \\
&- \max_{k \neq \hat{k}(\mathbf{x})} |\mathcal{N}_\theta(\mathbf{y})_k - \mathcal{N}_\theta(\mathbf{x})_k| \\
&\geq \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k - \max_{a,b \in \mathbb{R}} \left\{ |a| + |b| : \ \sqrt{a^2 + b^2} \leq \ell \|\mathbf{y} - \mathbf{x}\|_2 \right\} \\
&> \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k - \max_{a,b \in \mathbb{R}} \left\{ |a| + |b| : \ \sqrt{a^2 + b^2} \leq \frac{m(\mathbf{x})}{\sqrt{2}} \right\} \\
&\geq \mathcal{N}_\theta(\mathbf{x})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{x})_k - m(\mathbf{x}) = 0,
\end{aligned}$$

which allows to conclude that $\mathcal{N}_\theta(\mathbf{y})_{\hat{k}(\mathbf{x})} - \max_{k \neq \hat{k}(\mathbf{x})} \mathcal{N}_\theta(\mathbf{y})_k > 0$, and hence $\mathbf{y}$ gets assigned to the same class as $\mathbf{x}$. □

6.4.2 Building the neural network

In subsection 6.3, we discussed how one can restrict the time step of the explicit Euler method to get non-expansive maps approximating the time$-h$ flow of some negative gradient flows. This is the reasoning resented in [26]. We now use this theory to develop $1-$Lipschitz neural networks based on dynamical systems. Before doing so, however, we want to motivate the need for "cleverly" designed neural networks in order to force them to be $1-$Lipschitz. Let us consider the standard ResNet layer based on the explicit Euler method, which writes

$$\mathbf{x} \mapsto \mathbf{x} + hB_i^\top \sigma(A_i\mathbf{x} + \mathbf{b}_i) = F_{\theta_i}(\mathbf{x}).$$

We can upper bound the Lipschitz constant of F_{θ_i} as

$$\begin{aligned} \|F_{\theta_i}(\mathbf{y}) - F_{\theta_i}(\mathbf{x})\|_2 &= \left\|\mathbf{y} - \mathbf{x} + hB_i^\top \left(\sigma(A_i\mathbf{y} + \mathbf{b}_i) - \sigma(A_i\mathbf{x} + \mathbf{b}_i)\right)\right\|_2 \\ &\leq (1 + h\|A_i\|_2\|B_i\|_2 \mathrm{Lip}(\sigma))\, \|\mathbf{y} - \mathbf{x}\|_2, \end{aligned} \tag{62}$$

where $\mathrm{Lip}(\sigma)$ is the Lipschitz constant of σ, which for typical activation functions it is $\mathrm{Lip}(\sigma) = 1$. The derivation in (62) tells us that, for generic ResNets, it can be hard to have a layer with Lipschitz constant upper bounded by 1. To get this, we will modify the parametric update, relying on a parametrised negative gradient flow.

Let us consider the parametric scalar valued function

$$V(\mathbf{x}) = \mathbf{1}^\top \gamma(A\mathbf{x} + \mathbf{b}), \tag{63}$$

where

$$\gamma(\mathbf{x})_i = \begin{cases} \frac{1}{2}x_i^2 & \text{if} \quad x_i > 0, \\ \frac{\alpha}{2}x_i^2 & \text{else}, \end{cases}$$

$\alpha \in (0, 1)$, $\mathbf{1} \in \mathbb{R}^H$ is a vector of ones, and $A \in \mathbb{R}^{H\times d}, \mathbf{b} \in \mathbb{R}^H$ are free parameters. An Euler step with $\mathcal{F}(\mathbf{x}) = -\nabla V(\mathbf{x})$ leads to the layer

$$\mathbf{x} \mapsto \psi_{\mathcal{F}}^h(\mathbf{x}) := \mathbf{x} - hA^\top \sigma(A\mathbf{x} + \mathbf{b}), \tag{64}$$

where $\sigma(\mathbf{x})_i = \max\{\alpha x_i, x_i\}$ is the Leaky ReLU activation function.

Lemma 14. *The function in* (63) *is continuously differentiable, convex, and L-smooth.*

Proof. **Continuously differentiable:** V is the composition of continuously differentiable functions, and hence it is continuously differentiable.

Convex: γ is convex, thus all the components of the function $\gamma(A\mathbf{x} + \mathbf{b})$ are convex scalar valued functions since they are the composition of a convex function with a linear function. Finally, the $\mathbf{1}^\top$ takes a linear combination of the components of a convex function with positive weights, hence leading to a convex scalar valued function.

L-smooth: Being $L-$smooth means that the gradient

$$\nabla V(\mathbf{x}) = A^\top \sigma(A\mathbf{x} + \mathbf{b})$$

is $L-$Lipschitz continuous. Following the same argument done above, we see that $\text{Lip}(\nabla V) \leq \|A\|_2^2$, which allows us to conclude the proof. □

Based on this analysis, and the derivations in subsection 6.3, we can conclude that the Euler step in (64) is 1-Lipschitz if

$$h \leq 2/\|A\|_2^2. \tag{65}$$

As we will see in subsection 6.4.3, we can easily satisfy this constraint during training, using the power method to keep track of $\|A\|_2$. Of course, since non-expansive maps compose to give non-expansive maps, we can now compose any number of layers as in (64) to get a non-expansive neural network.

6.4.3 Testing it on a simple classification task

We implement a non-expansive neural network following the principles presented above. Each network layer corresponds to an explicit Euler step of a suitable vector field. More explicitly, the layers are based on the expression in (64). The implementation for this experiment can be found in the Jupyter notebook titled `adversarial_robustness.ipynb`.

A non-expansive neural network can be obtained by composing several of gradient steps we built before. To enforce the non-expansivity of the Euler step, we need to implement a suitable step size constraint, which requires us to estimate the spectral norms of the linear layers. This is done with the power method. Let us consider a matrix $A \in \mathbb{R}^{d\times c}$ defining the linear layer of interest. The power method is implemented as

$$\begin{aligned} &\mathbf{x}_0 \in \mathbb{R}^c \\ &\mathbf{x}_{i+1} = \frac{A^\top A u_i}{\|A^\top A \mathbf{x}_i\|_2} \in \mathbb{R}^c, \;\; i = 0, \ldots, k. \end{aligned} \tag{66}$$

The vector $\mathbf{x}_0$ could either be an initial estimate of the first right singular vector of A or a random vector. If $\mathbf{x}_0$ is not orthogonal to the target right singular vector, this iteration computes $\mathbf{x}_k$ which usually approximates the first right singular vector of A, and $\sqrt{\|A^\top A \mathbf{x}_k\|_2}$ converges to $\|A\|_2$ as $k \to \infty$.

Before training, we run many iterations of the power method and save the resulting estimates of the top right singular vectors of the linear layers. Much of the usual training loop for neural networks remains the same for networks built using non-expansive blocks: we load a minibatch of data and pass it through the network, we evaluate the network's predictions using the loss function, and we backpropagate and perform a gradient update. Before passing to the next mini-batch, however, we update our estimates of the spectral norms of the weights using the power method. Since we have a good estimate of the top right singular vector, we use it to warm-start the power method, making it possible to use just a single iteration of the power method. After this, `n_steps` of the Euler method are computed, where `n_steps` is the smallest integer such that

$h = T/\texttt{n\_steps}$ (with T the total integration time) satisfies the step size constraint, i.e., it is smaller than $2/\|A\|_2^2$. That is to say, we adapt the step size as necessary to preserve non-expansiveness when the Lipschitz constant of the vector field grows.

We work with the Fashion MNIST dataset, consisting of images of items from Zalando, along with a label denoting one of ten possible classes. It is based on a training set of 60,000 images and a test set of 10,000 images. Each is a greyscale image of size 28×28. Figure 7 shows five images in the training set with their associated labels.

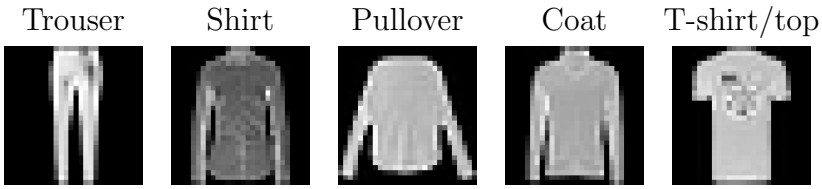

Figure 7: 5 example images from the Fashion MNIST dataset.

Once the network is trained, we can test its robustness to adversarial attacks. We consider the ℓ^2-PGD attack, standing for Projected Gradient Descent based on the ℓ^2-norm. The algorithm defining this attack is implemented in the notebook, but let us describe the mechanics of the attack in some more detail here. This attack aims to maximise the loss function `loss_fn`, which we provide as input, by perturbing the input image `image`. The correct label for the input image is `target`, and the perturbation of the input image we allow has ℓ^2-norm smaller than `epsilon`. To build this perturbation, we perform `n_iter` iterations of the following procedure. Let us consider the function

$$F(\texttt{delta}) := \texttt{loss\_fn}(\texttt{image} + \texttt{delta}, \texttt{target}).$$

Each of the `n_iter` iterations consists of one step of size `step_size` in the direction of $\partial_{\texttt{delta}} F(\texttt{delta}) / \|\partial_{\texttt{delta}} F(\texttt{delta})\|_2$ followed by a projection over the ℓ^2-ball of radius `epsilon` centered at the origin. Finally, `delta` is added to `image` to get the perturbed image.

In Figure 8 we show an example of an image attacked with the ℓ^2-PGD attack with 100 iterations. The attack is displayed in different magnitudes, and one can see that the image looks increasingly distinct from the first one on the left, i.e., the clean image. The network we attack to obtain these perturbations is a ResNet trained to classify the test images with around 89% accuracy.

We have now discussed all the necessary methods to evaluate the robustness of a non-expansive network architecture and compare it to that of an unconstrained ResNet. This comparison relies on two steps: training both networks on clean images and testing their accuracy on adversarial images built for the specific weights obtained after training. To have a code that takes five to ten minutes to train locally, we restrict the training and test sets to 30,000 and

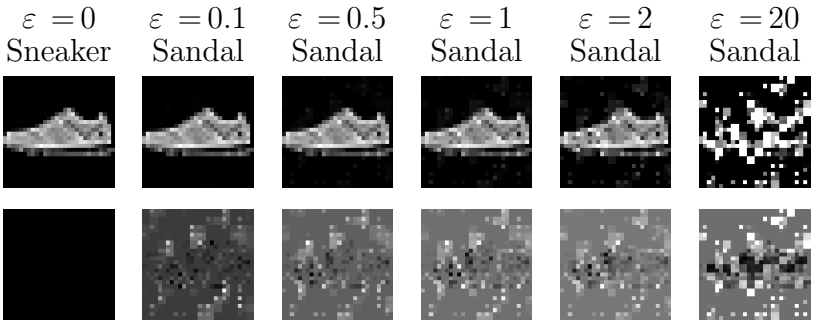

Figure 8: The first row displays the attacked images with increasing perturbation magnitudes. The second row displays the difference between the attacked and clean images. The titles specify the norm of the perturbation ε and the ResNet prediction when given that image as an input.

1,000 images, respectively. The non-expansive network and the ResNet reach a similar test accuracy of around $88 - 89\%$. We train both models for 30 epochs, again to benefit in terms of speed. When the training is completed, we freeze their parameters and build adversarial examples. The examples are obtained with 100 iterations of the ℓ^2-PGD attack, and we generate them for different perturbation magnitudes. We consider eight attack magnitudes smaller than one and compare them with the clean accuracy corresponding to $\varepsilon = 0$. Generating the attack for the 1,000 images takes around five to ten minutes. We plot the results obtained following this procedure in Figure 9. We see a very small drop in performance for this relatively simple dataset when constraining the Euler steps to be 1-Lipschitz. At the same time, robust accuracy improves over that of unconstrained layers. The gain in robustness is also expected for other datasets, while typically, the clean accuracy tends to decrease a bit more compared with the unconstrained model.

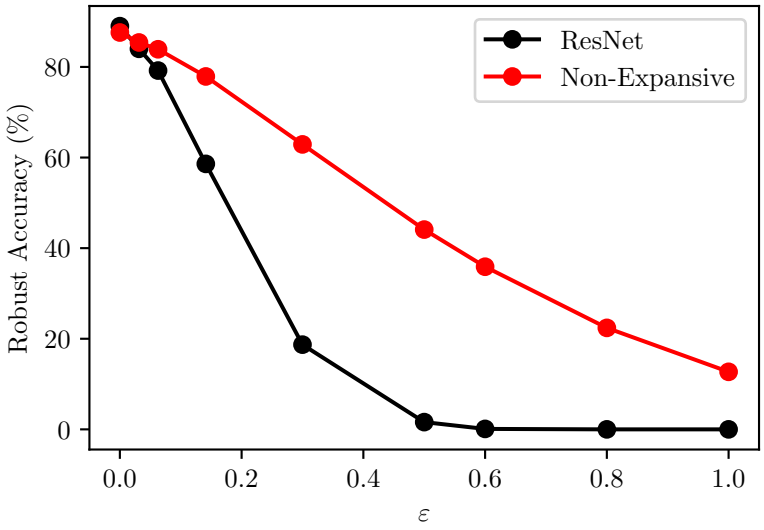

Figure 9: Comparison of the classification accuracy of a non-expansive network and a ResNet trained on 30,000 training images of the Fashion MNIST dataset. We then attack 1,000 of the test images with 100 iterations of ℓ^2-PGD of varying intensity ε. The attack magnitude is represented on the horizontal axis, with $\varepsilon = 0$ corresponding to the clean images. The vertical axis displays the classification accuracy obtained with the attacked images.

References

[1] Vegard Antun, Francesco Renna, Clarice Poon, Ben Adcock, and Anders C Hansen. On instabilities of deep learning in image reconstruction and the potential costs of AI. *Proceedings of the National Academy of Sciences*, 117(48):30088–30095, 2020. 56

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017. 57

[3] Sergio Blanes, Arieh Iserles, and Shev Macnamara. Positivity-preserving methods for population models. *arXiv preprint arXiv:2102.08242*, 2021. 31

[4] F. Bullo. *Contraction Theory for Dynamical Systems*. Kindle Direct Publishing, 1.2 edition, 2024. 48

[5] Leon Bungert, René Raab, Tim Roith, Leo Schwinn, and Daniel Tenbrinck. CLIP: Cheap Lipschitz training of neural networks. In *International Con-*

ference on Scale Space and Variational Methods in Computer Vision, pages 307–319. Springer, 2021. 57

[6] Hans Burchard, Eric Deleersnijder, and Andreas Meister. A high-order conservative Patankar-type discretisation for stiff systems of production–destruction equations. *Applied Numerical Mathematics*, 47(1):1–30, 2003. 31

[7] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989. 16

[8] Germund G Dahlquist. Generalized disks of contractivity for explicit and implicit Runge-Kutta methods. Technical report, CM-P00069451, 1979. 55

[9] Jiu Ding and Aihui Zhou. Eigenvalues of rank-one updated matrices with some applications. *Applied Mathematics Letters*, 20(12):1223–1226, 2007. 9

[10] Clara Lucía Galimberti, Luca Furieri, Liang Xu, and Giancarlo Ferrari-Trecate. Hamiltonian Deep Neural Networks Guaranteeing Nonvanishing Gradients by Design. *IEEE Transactions on Automatic Control*, 68(5):3155–3162, 2023. 47

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 56

[12] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing Lipschitz continuity. *Machine Learning*, 110:393–416, 2021. 57

[13] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2006. 42, 44

[14] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*, volume 8. Springer, 1993. 5, 7

[15] Juncai He, Lin Li, and Jinchao Xu. Approximation Properties of Deep ReLU CNNs. *Research in the Mathematical Sciences*, 9(3):38, 2022. 16

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 15

[17] Inga Hense and Aike Beckmann. The representation of cyanobacteria life cycle processes in aquatic ecosystem models. *Ecological Modelling*, 221(19):2330–2338, 2010. 31

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. 16

[19] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Number 44. Cambridge university press, 2009. 8

[20] Qianxiao Li, Ting Lin, and Zuowei Shen. Deep learning via dynamical systems: An approximation perspective. *Journal of the European Mathematical Society*, 2022. 16

[21] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations*, 2018. 57

[22] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87. Springer Science & Business Media, 2013. 51, 52

[23] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, and Michael Unser. Approximation of Lipschitz Functions Using Deep Spline Neural Networks. *SIAM Journal on Mathematics of Data Science*, 5(2):306–322, 2023. 16

[24] G Reinout W Quispel and Grant S Turner. Discrete gradient methods for solving ODEs numerically while preserving a first integral. *Journal of Physics A: Mathematical and General*, 29(13):L341, 1996. 16

[25] HH Robertson. The Solution of a Set of Reaction Rate Equations. *Numerical analysis: an introduction*, 178182, 1966. 31

[26] Ferdia Sherry, Elena Celledoni, Matthias J Ehrhardt, Davide Murari, Brynjulf Owren, and Carola-Bibiane Schönlieb. Designing stable neural networks using convex analysis and ODEs. *Physica D: Nonlinear Phenomena*, 463:134159, 2024. 59

[27] MN Spijker. Contractivity in the numerical solution of initial value problems. *Numerische Mathematik*, 42:271–290, 1983. 53

[28] Gerhard Wanner and Ernst Hairer. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, volume 375. Springer Berlin Heidelberg New York, 1996. 5

[29] Ge Zhong and Jerrold E Marsden. Lie-Poisson Hamilton-Jacobi theory and Lie-Poisson integrators. *Physics Letters A*, 133(3):134–139, 1988. 41