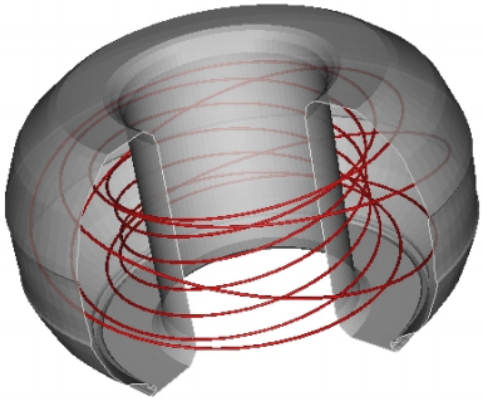
HènonNet: a symplectic neural network

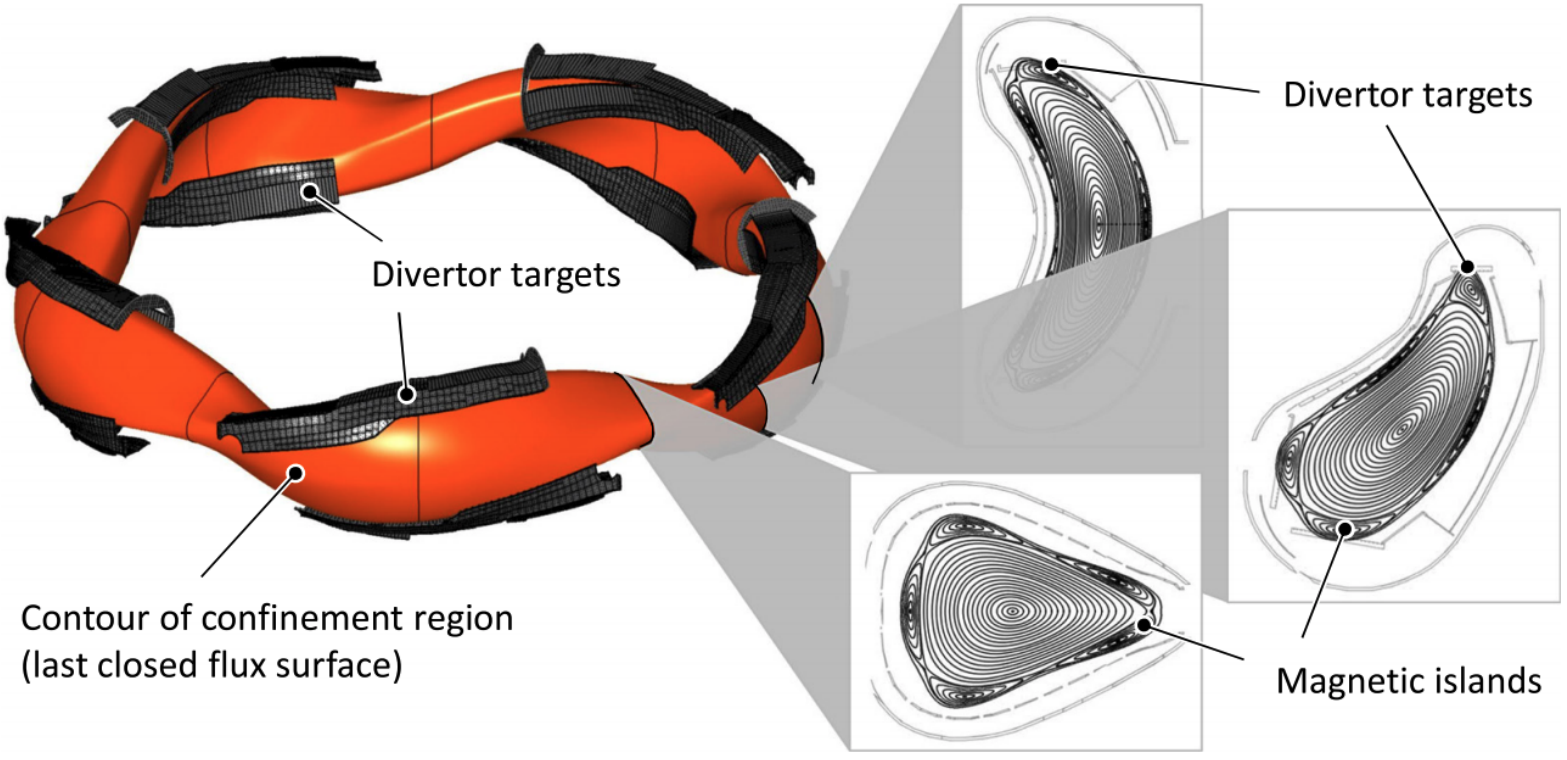
J. W. Burby (LANL), Q. Tang (LANL)

R. Maulik (ANL)

June 24, 2021
GAMP

Motivation: Poincare plots in fusion devices





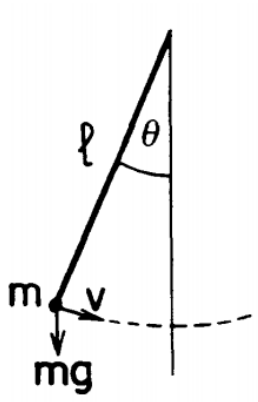
Hamiltonian system

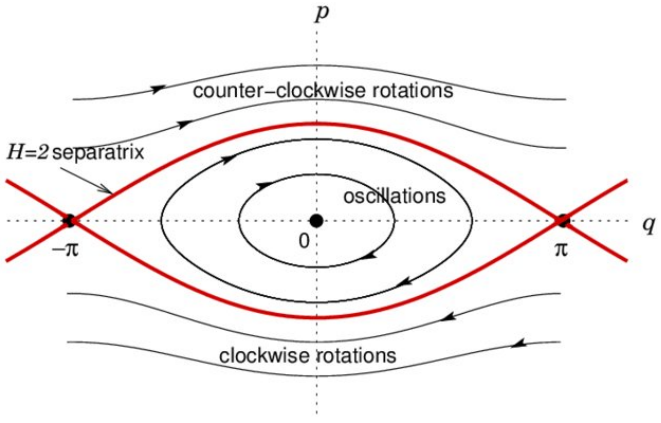
A Hamiltonian dynamical system is

$$\frac{d\mathbf{r}}{dt} = S_N \nabla H(\mathbf{r})$$

where $\mathbf{r} = (\mathbf{p}, \mathbf{q})$ is a 2N-dimensional vector of the canonical coordinate, H is the Hamiltonian,

$$S_N = \begin{bmatrix} 0 & I_N \\ -I_N & 0 \end{bmatrix}$$





Mathematical Pendulum

$$H(p, \theta) = \frac{p^2}{2ml^2} + mgl(1 - \cos\theta)$$

Symplectic maps

Definition (symplectic maps)

A differentiable map from $g : U \in \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ is called **symplectic** if its Jacoiban matrix satisfies

$$g'(\mathbf{p},\mathbf{q})^T \, S_N \, g'(\mathbf{p},\mathbf{q}) = S_N$$

Define a flow map ψ_t for $H(\mathbf{p},\mathbf{q})$ and time t such that

$$\psi_t(\mathbf{p}_0,\mathbf{q}_0) = (\mathbf{p}(t,\mathbf{p}_0,\mathbf{q}_0),\mathbf{q}(t,\mathbf{p}_0,\mathbf{q}_0))$$

Theorem

Let $H(\mathbf{p},\mathbf{q})$ be a twice continuously differentiable function on $U \in \mathbb{R}^{2N}$. Then, for each fixed t, the flow map ψ_t is a symplectic transformation wherever it is defined.

Numerical integrators should mimic this property!

Conventional approach: symplectic integrators

Definition (symplectic integrators)

A numerical integrator $(\mathbf{p}^{n+1}, \mathbf{q}^{n+1}) = g_h(\mathbf{p}^n, \mathbf{q}^n)$ is called symplectic if it satisfies symplecticity **exactly**, i.e.,

$$g_h'(\mathbf{p}^n, \mathbf{q}^n)^T \, S_N \, g_h'(\mathbf{p}^n, \mathbf{q}^n) = S_N$$

Pros: Many good properties such as area-preserving

Cons:

- typically at least semi-implicit or fully implicit
- hard to accelerate through a parallel-in-time integrator
- challenging to apply to PDEs (require some careful design)

How can we improve that?

We propose a symplectic neural network
to approximate the flow-map directly

Scientific ML applied to physics/math

Two types of SciML in math/physics:

- Build physics/math into loss functions: physics-informed neural networks (PINN), surrogate models, etc
- Modify network architecture: ResNet, ConvNet, autoencoder, recurrent neural network (RNN), etc

Cast them into something we are familiar with:

- Modify loss functions = change schemes in numerical PDEs
- Modify network architecture = change fundamental approximations

In this work, we will show how to achieve symplecticity through changing the neural network architecture

A symplectic neural network is a feed-forward NN trained to approximate a symplectic map

Definition (Symplectic Neural Network)

Given a flow map $\mathcal{P} : (x, y) \mapsto (\overline{x}, \overline{y})$, a **symplectic neural network** is a feed-forward neural network $N_{\boldsymbol{W}} : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ satisfying symplecticity such that $\mathcal{P} \approx N_{\boldsymbol{W}^*}$ for some $\boldsymbol{W}^* \in \boldsymbol{\mathcal{W}}$.

Two key questions:

- How to construct a symplectic NN?
- Can it approximate any symplectic diffeomorphism?

The key is a novel network architecture

The basic building block

Definition (Hénon layer)

Let $V_W : \mathbb{R}^N \to \mathbb{R}$, $W \in \mathcal{W}$, be a feed-forward neural network. The **Hénon layer** with potential V_W and bias $\eta \in \mathbb{R}^N$ is the layer $HL_{(W,\eta)} : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ given by

$$HL_{(W,\eta)} = H_{(W,\eta)} \circ H_{(W,\eta)} \circ H_{(W,\eta)} \circ H_{(W,\eta)},$$

where $H_{(W,\eta)} : (x, y) \mapsto (\overline{x}, \overline{y})$ is given by

$$\begin{aligned} \overline{x} &= y + \eta \\ \overline{y} &= -x + \nabla V_W(y). \end{aligned}$$

Some magic: $HL_{(W,\eta)}$ is a symplectic map for any (W, η)!

The key is a novel network architecture

The network architecture

Definition (Hénon network)

A Hénon network $HN_{\boldsymbol{W},\boldsymbol{\eta}}$ is a composition of Hénon layers, i.e.

$$HN_{(\boldsymbol{W},\boldsymbol{\eta})} = HL_{(W_N,\eta_N)} \circ \cdots \circ HL_{(W_1,\eta_1)}.$$

All Hénon networks are symplectic

The theoretical foundation is a theorem due to Turaev

Theorem (Symplectic universal approximation)

Given a C^r symplectic diffeomorphism $\mathcal{F} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$, a compact region $U \subset \mathbb{R}^n \times \mathbb{R}^n$, and $\epsilon > 0$, there exists

- *a smooth function $V : \mathbb{R}^n \to \mathbb{R}$*
- *a vector $\eta \in \mathbb{R}^n$*
- *a positive integer N*

such that $|H[V, \eta]^{4N} - \mathcal{F}|_{C^r(U)} < \epsilon$.

[Dmitry Turaev. Nonlinearity 16.1 (2002): 123.]

Our idea:

Use $H[V, \eta]^{4N}$ as a model NN architecture

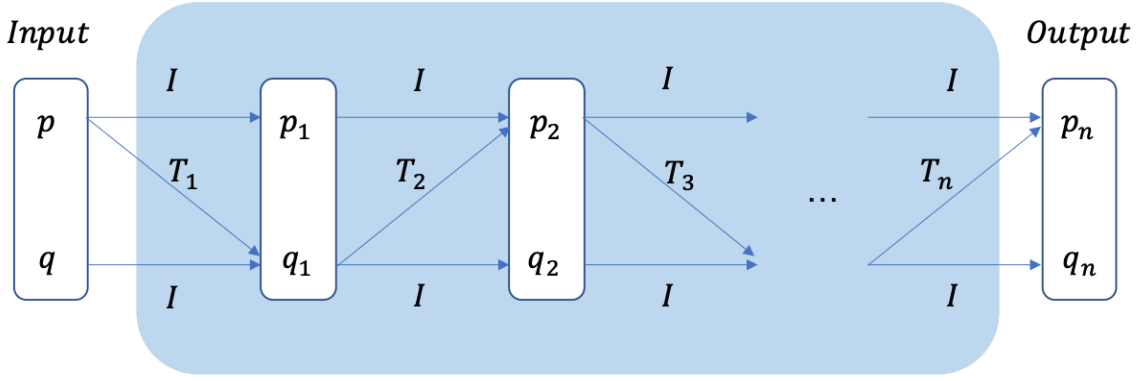
HénonNets have some good properties

- Collection of all HénonNets forms a group
- Closure of that set is the symplectomorphism group
- HénonNet is an invertible network. Inverse of a HénonNet is a HénonNet. Its inverse is easy to construct and fast to evaluate.
- Derivatives of a HénonNet are easy to compute using automatic differentiation (sensitivity analysis)
- HénonNet is a type of ResNet. We can prove each Hénon layer fits into the ResNet framework of

$$\bar{\mathbf{x}} = \mathbf{x} + \mathcal{F}(\mathbf{x}).$$

Other works in the literature

- SympNet [Jin, et al., Neural Networks 2020]:



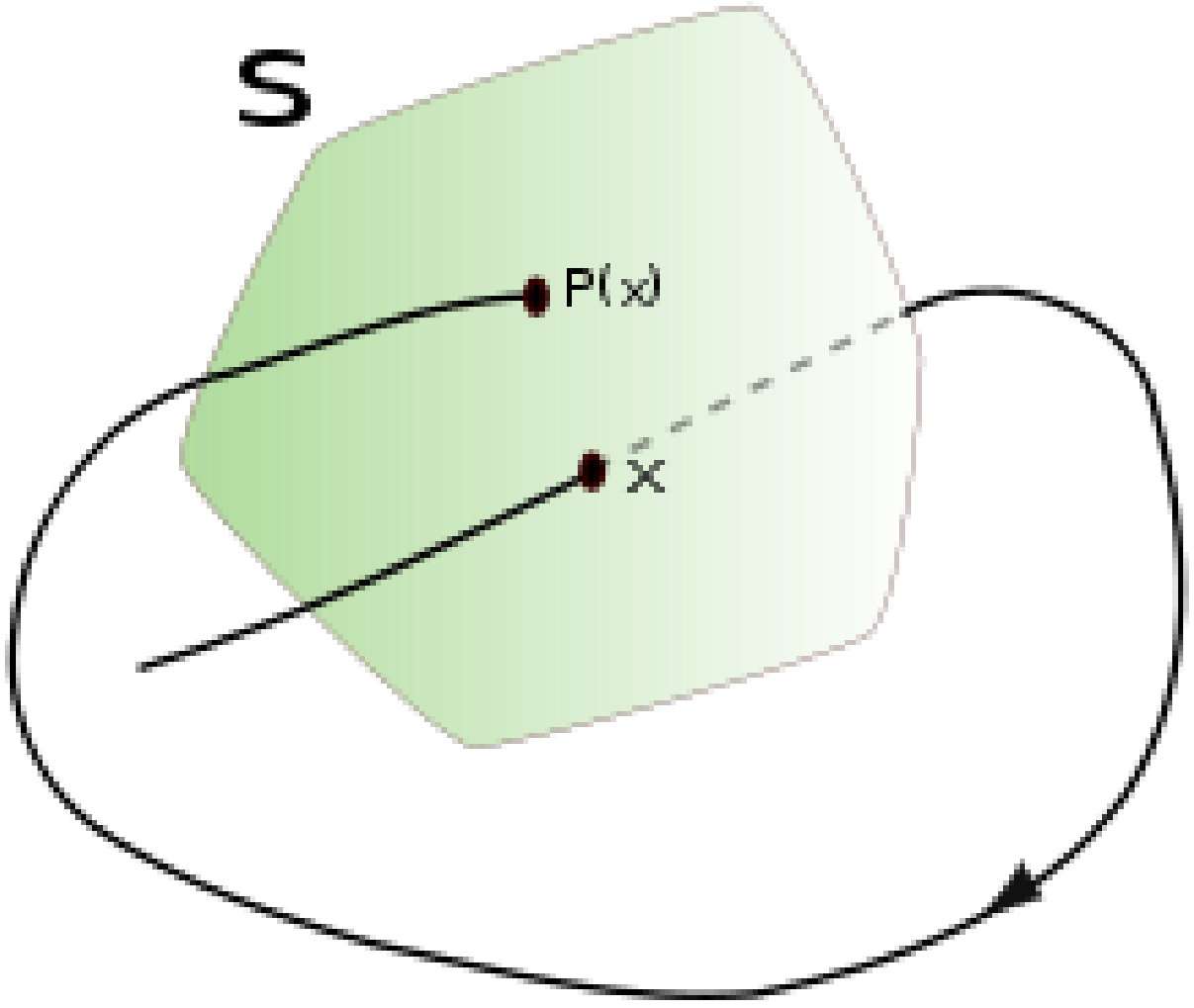
- Symplectic GP [Rath et al., Chaos, 2021]: generating function and implicit-defined Poincaré maps

$$\begin{pmatrix} \tilde{F}(\boldsymbol{q},\boldsymbol{P}) \\ \partial_{\boldsymbol{q}}\tilde{F}(\boldsymbol{q},\boldsymbol{P}) \\ \partial_{\boldsymbol{P}}\tilde{F}(\boldsymbol{q},\boldsymbol{P}) \end{pmatrix} \sim \mathcal{GP}(\boldsymbol{n}(\boldsymbol{q},\boldsymbol{P}), K(\boldsymbol{q},\boldsymbol{P},\boldsymbol{q}',\boldsymbol{P}'))$$

An important application

Use **HénonNet** to accelerate Poincaré maps faster than ever before

Today, field-line Poincaré plots require field-line tracing



Today, field-line Poincaré plots require field-line tracing

To get $\mathcal{P}(x, y)$, the current approach is:

1. integrate the equation

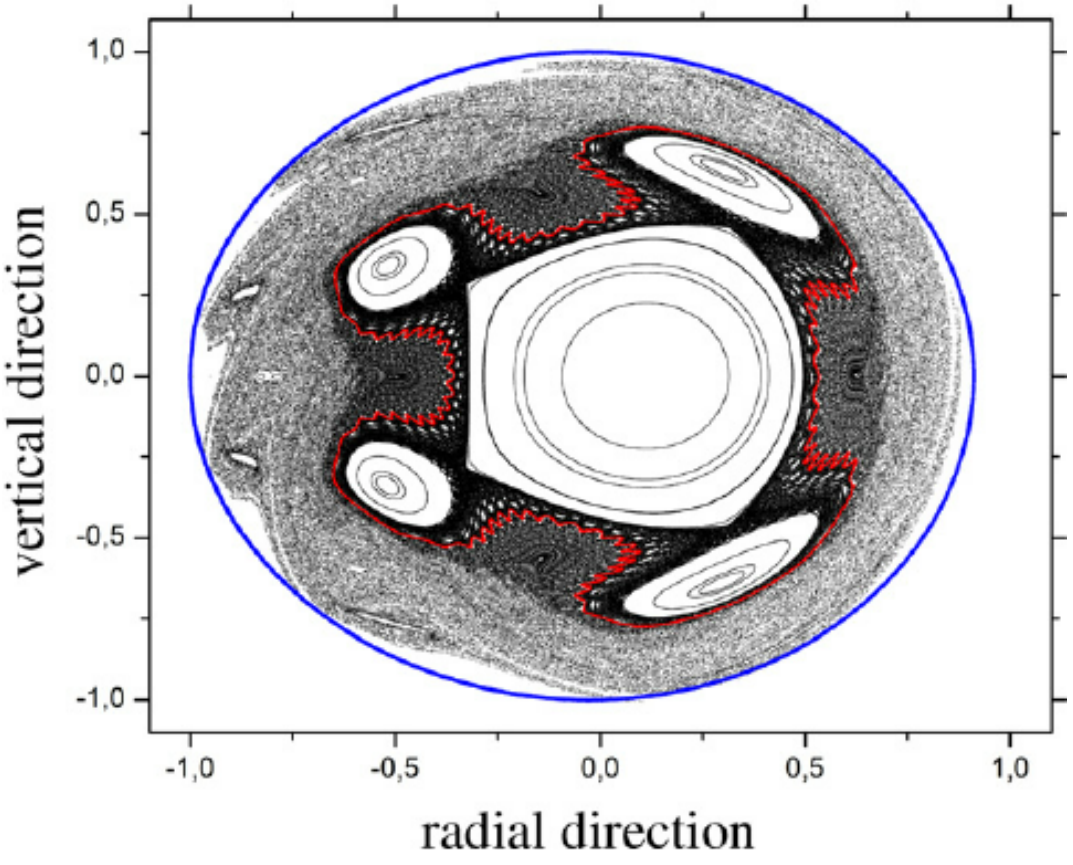
$$\frac{dx}{dt} = B^x(x, y, \phi)$$
$$\frac{dy}{dt} = B^y(x, y, \phi)$$
$$\frac{d\phi}{dt} = B^\phi(x, y, \phi)$$

   with initial condition $(x, y, 0)$ until $\phi(t^*) = 2\pi$.

2. Set $\mathcal{P}(x, y) = (x(t^*), y(t^*))$

That method has one major inefficiency

Generating plots like this



parallelizes over initial conditions but not in time!

We have developed a new method that pushes the inefficiency "offline"

Our new approach to generating Poincaré maps

1. Train a feed-forward neural network to approximate single iteration of Poincaré map $(x, y) \mapsto \mathcal{P}(x, y)$
2. Directly iterate the trained NN to generate Poincaré section

First step is slow, but **offline**

Second step is **online** and very fast

We have developed a new method that pushes the inefficiency "offline"

Time for one iteration of Poincaré map using NN

$T_{\text{one turn}} = t_{\text{one layer}} M_{\text{layers}}$

- $t_{\text{one layer}}$: time to evaluate a single NN layer
- M_{layers}: number of NN layers

$t_{\text{one layer}}$ is very short, especially on GPUs. Easily can be shorter than $t_{\text{one step}}$.

We have developed a new method that pushes the inefficiency "offline"

Time for one iteration of Poincaré map using NN

$T_{\text{one turn}} = t_{\text{one layer}} M_{\text{layers}}$

- $t_{\text{one layer}}$: time to evaluate a single NN layer
- M_{layers}: number of NN layers

Testing shows $M_{\text{layers}} \approx 10 - 20$, even for fields with chaos.

How do we train a Hénon network?

Formulation as optimization problem

HénonNet training as optimization

Let $\mathcal{P}_{\text{ode}}(x, y)$ be an approximation of the Poincaré map given by field-line integration. The HénonNet cost function is

$$S(\boldsymbol{W}, \boldsymbol{\eta}) = \frac{1}{|I|} \sum_{i \in I} |HN_{(\boldsymbol{W}, \boldsymbol{\eta})}(x_i, y_i) - \mathcal{P}_{\text{ode}}(x_i, y_i)|^2,$$

where (x_i, y_i) are samples of the region of interest in the Poincaré section.

Goal: find

$$(\boldsymbol{W}^*, \boldsymbol{\eta}^*) \in \arg\min S(\boldsymbol{W}, \boldsymbol{\eta}).$$

We used TensorFlow's Adam optimizer to find $(\boldsymbol{W}^*, \boldsymbol{\eta}^*)$.

Summary of HénonNet approach

Offline training: Poincare map

1. Build the training data: using an accurate RK scheme to integrate from (x_i, y_i) to $\mathcal{P}(x_i, y_i)$
2. Build a HénonNet to approximate the Poincare map $\mathcal{P}$
3. Using the supervised learning to train the HénonNet

Online prediction: Poincare plot

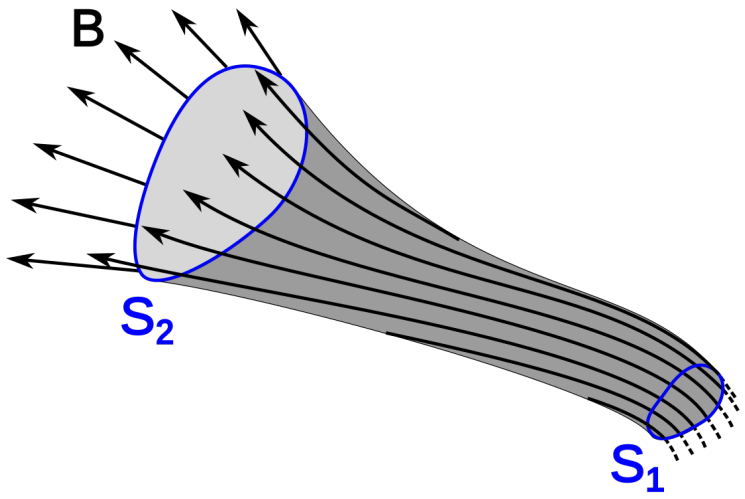
Select a few starting points (x_s, y_s) and use the trained HénonNet to predict many times: $\mathcal{P}(x_s, y_s)$, $\mathcal{P}(\mathcal{P}(x_s, y_s))$, ...

- Training is slow but prediction is very fast (10x on CPU)
- HénonNet can be viewed as an explicit parallel symplectic integrator (surrogate) for a very stiff system ($\Delta t = 2\pi$!)

Our approach also exactly preserves the primary physics constraint on Poincaré maps

Proposition

If $\mathcal{P}$ is the Poincaré map for a magnetic field $\boldsymbol{B}$ and U is a region in the Poincaré section then $\Gamma(U) = \Gamma(\mathcal{P}(U))$, where $\Gamma(U)$ is the magnetic flux through U.



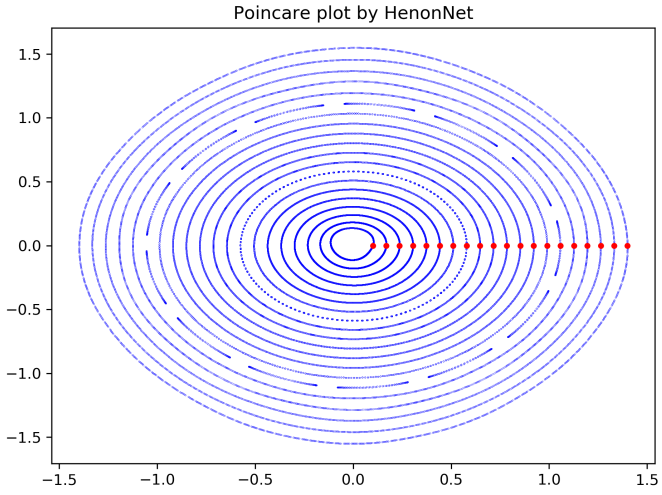
Proposition

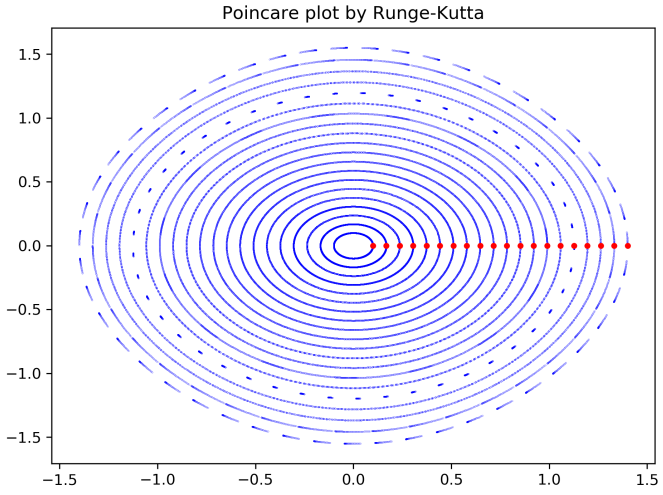
Our neural approximation for $\mathcal{P}$ has exactly the same property

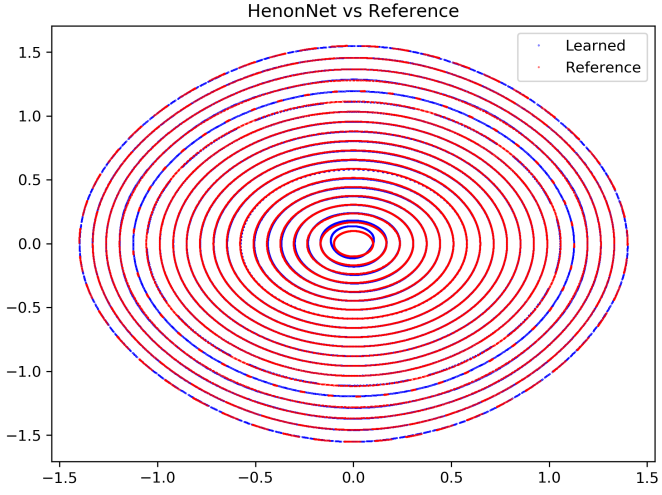
Examples

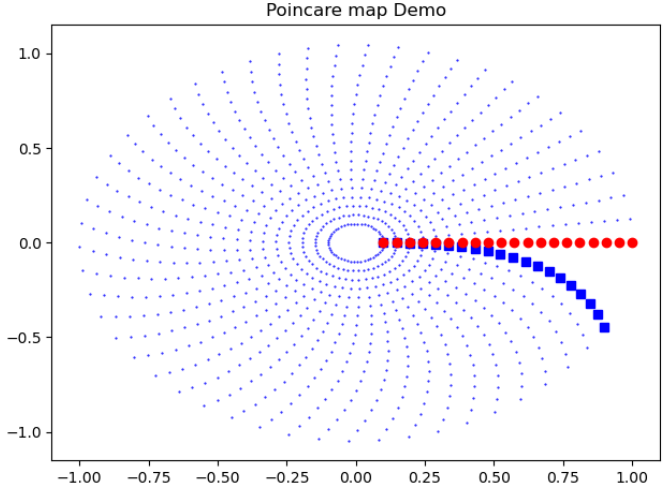
Pendulum Poincare plot predicted by HénonNet

Training data: 200K points generated with time from 0 to 2π
HénonNet: 10 layers with 10 internal size (310 trainable parameters)
Final loss: 6e-7; the network predicts 2000 times to produce the plot









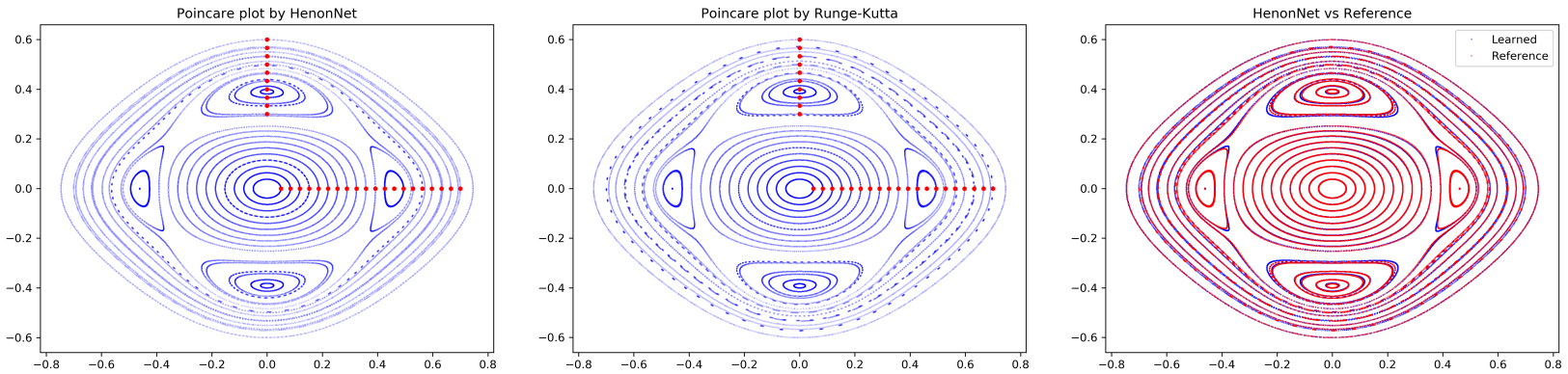
Perturbed pendulum Poincare plot predicted by HénonNet

Consider a perturbed pendulum

$$H_{\text{pp}}(x, y, \phi) = \frac{1}{2}y^2 - \omega_0^2 \cos x - \epsilon \left[0.3\, xy \sin(2\phi) + 0.7\, xy \sin(3\phi) \right],$$

with $\omega_0 = 0.5$ and $\epsilon = 0.5$.

Training data: 220K points generated with time from 0 to 2π
HénonNet: 10 layers with 10 internal size (310 trainable parameters)
Final loss: 1e-7; the network predicts 1000 times to produce the plot

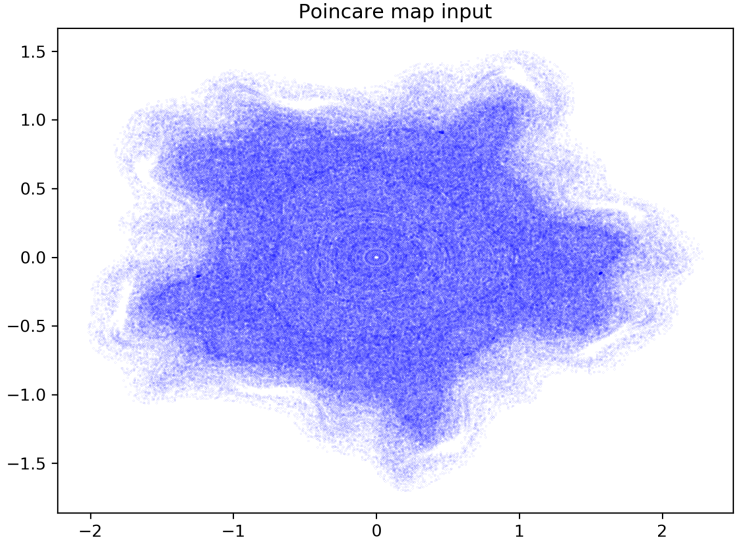


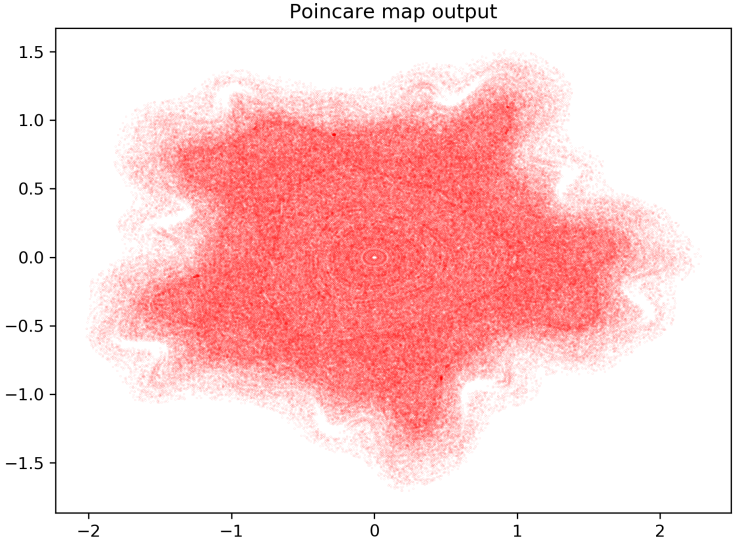
Resonant magnetic perturbation predicted by HénonNet

Consider a perturbed pendulum ($\epsilon = 0.25$)

$$H_{\text{RMP}}(x, y, \phi) = \frac{1}{2}y^2 + \frac{1}{4}x^2 + \frac{\epsilon}{4}\left(\tanh\left((x-y)x^2\cos(3\phi)\right) + \frac{1}{5}\tanh\left((x-y)x^2\sin(3\phi)\right) \right),$$

Training data: 400K points generated with time from 0 to 2π
HénonNet: 50 layers with 5 internal size (800 trainable parameters)
Final loss: 3e-5; the network predicts 1000 times to produce the plot

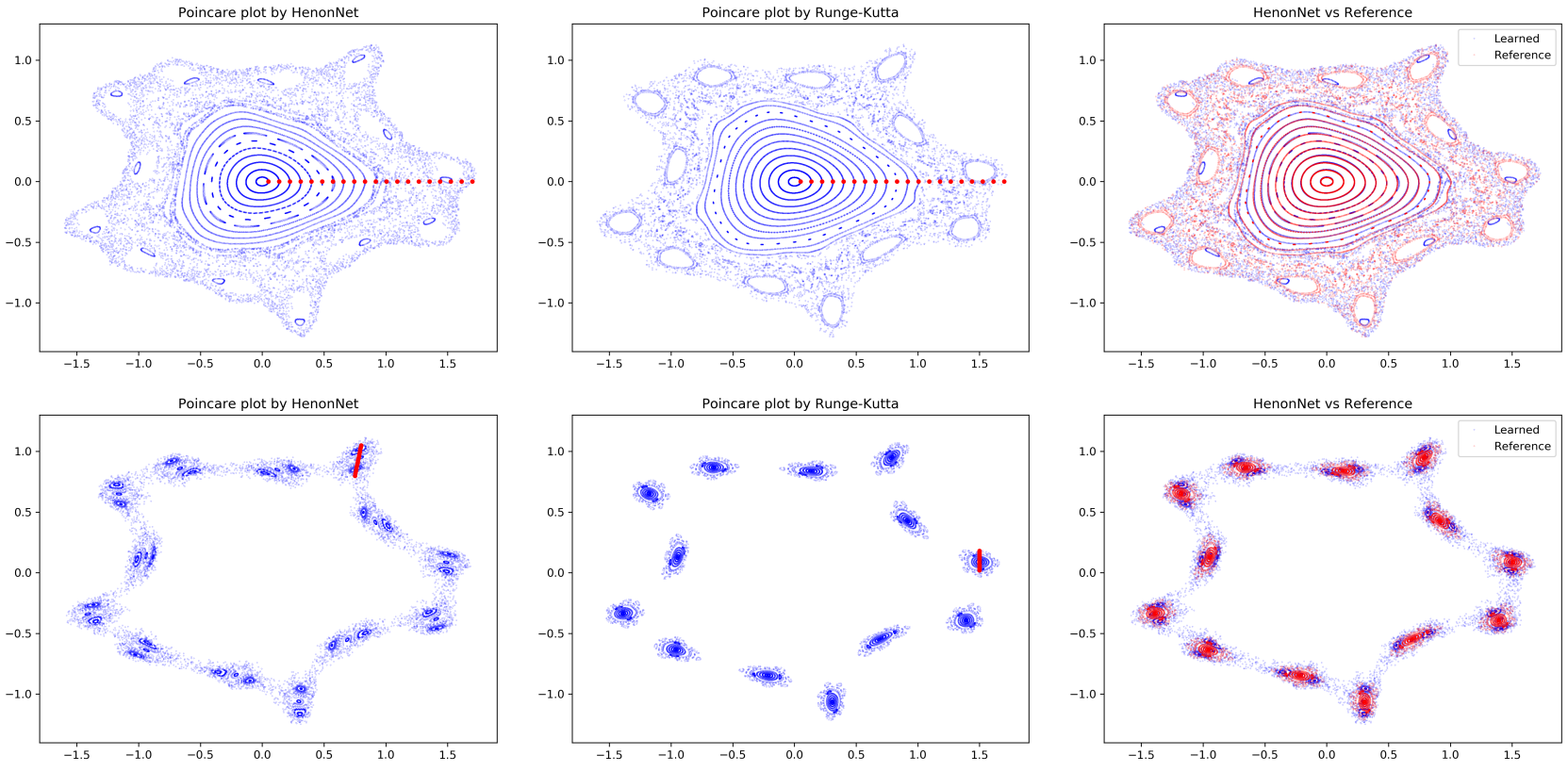




Resonant magnetic perturbation predicted by HénonNet

Consider a perturbed pendulum ($\epsilon = 0.25$)

$$H_{\mathrm{RMP}}(x, y, \phi) = \frac{1}{2}y^2 + \frac{1}{4}x^2 + \frac{\epsilon}{4}\left(\tanh\left((x-y)x^2\cos(3\phi)\right) + \frac{1}{5}\tanh\left((x-y)x^2\sin(3\phi)\right) \right),$$



Resonant magnetic perturbation: sensitivity analysis

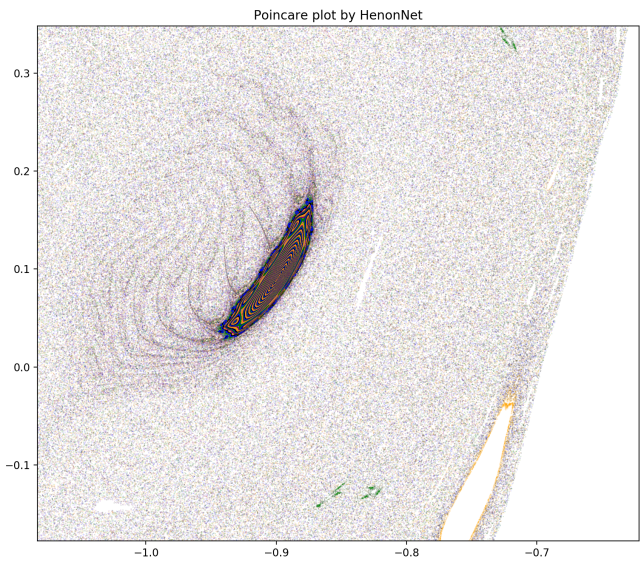
Given a discrete-time dynamic system

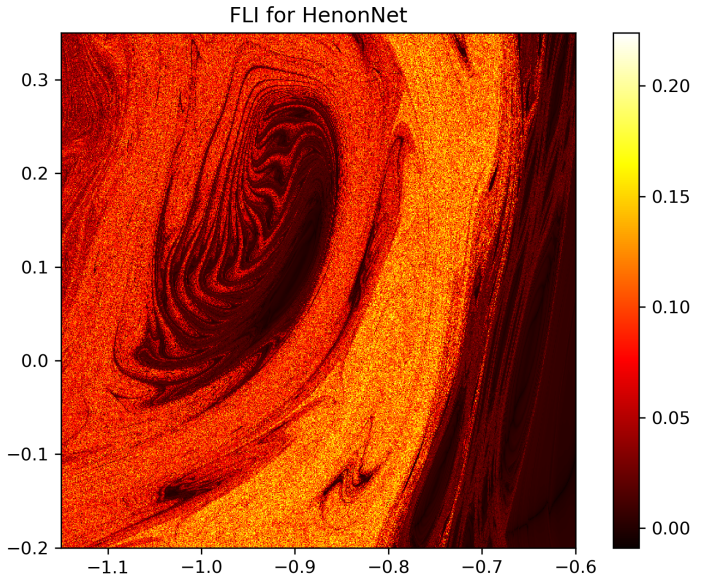
$$\begin{cases} x(t+1) = \mathcal{P}(x(t)) \\ v(t+1) = \frac{\partial \mathcal{P}}{\partial x} v(t) \end{cases}$$

and the time-t Fast Lyapunov Indicator (FLI) is defined as

$$FLI(x(0), v(0)) = \log \frac{\|v(t)\|}{\|v(0)\|}$$

FLI can be easily computed through HénonNet (< 20 lines of code)





Can we approximate more than one magnetic field?

We propose parameterized HenonNets to approximate a family of flow maps

Parameterized case

We generalized and proved:

Theorem (Parameterized symplectic universal approximation)

Given a C^r symplectic diffeomorphism $\mathcal{F}_\mu : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$ with smooth μ-dependence, a compact region $U \subset \mathbb{R}^n \times \mathbb{R}^n$, and $\epsilon > 0$, there exists

- *a smooth function $V_\mu : \mathbb{R}^n \to \mathbb{R}$*
- *a vector $\eta_\mu \in \mathbb{R}^n$*
- *a positive integer N*

such that $|H[V_\mu, \eta_\mu]^{4N} - \mathcal{F}_\mu|_{C^r(U)} < \epsilon$ uniformly in μ.

Test case: standard map

Parameterized, **symplectic**, **periodic** maps:

$$p_{n+1} = p_n + K \sin(\theta_n) \mod 2\pi$$
$$\theta_{n+1} = \theta_n + p_{n+1} \mod 2\pi$$

To generate training data, we select 11 K points in the range of [0, 1]. For each given K, a random number of 400 points is picked.

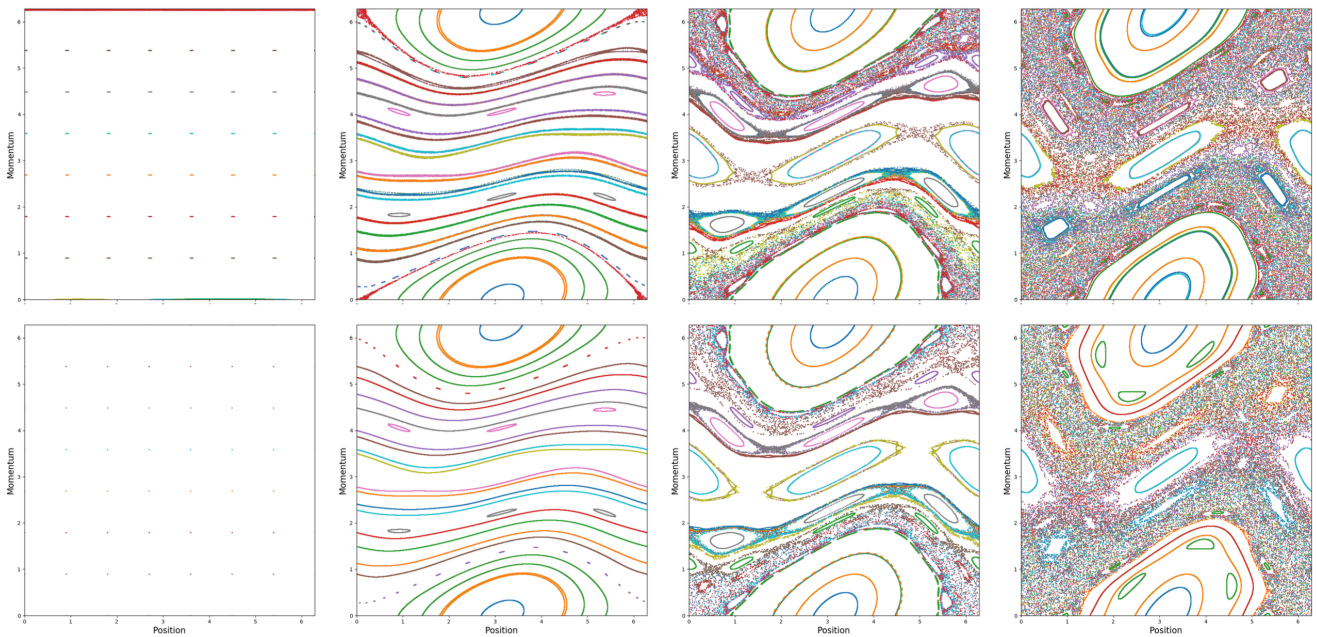


Figure: P-HenonNet (top row) vs ground truth (bottom row). From Left to Right: K=0, 0.5, 0.97 and 1.3 (extrapolation).

Conclusion and future works

In this work, we

- design and build a novel symplectic neural network
- apply it to construct a symplectic surrogate for Poincare maps
- show that the online stage to generate Poincare plot is about 10x faster than any conventional schemes
- are considering parameterized symplectic approximation
- **Structure-preserving NNs is a promising direction** (divergence-free, hyperbolicity, positivity, etc)
- Future works: dissipative system, PDEs, surrogates, etc

Reference:

J. W. Burby, Q. Tang, and R. Maulik. "Fast neural Poincaré maps for toroidal magnetic fields." Plasma Physics and Controlled Fusion 63.2 (2020): 024001.