
# RNVP normalising flows for computing molecular vibrational wave functions - Report

Sebastian Mendoza

August 2022

---

**Abstract**

This project deals with a widely applicable and quite ubiquitous problem in physics: solving Schrödinger's Equation for complex systems. While analytical solutions are scarce, many different methods have been created to find approximate solutions to the problem. One of the most current obstacles is the curse of dimensionality: the amount of data needed for converging approximations scales exponentially with the number of dimensions. Here, we try to alleviate this exponential dependency by applying modern Machine Learning methods (namely Normalizing flows) to the numerical methods for finding eigen-energies and eigenstates; we perform numerical approximations and mathematical proofs to show the efficacy of this approach.

---

# Contents

# 1 Background Knowledge and Introduction

## 1.1 Physics

The physical equation governing the setup of the problem is given by the Time Independent Schrödinger's Equation (**TISE**)

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle \tag{1}$$

where $\hat{H}$ describes a Hermitian Operator acting on state vectors $|\psi\rangle \in \mathcal{H}$, with real eigenvalues $E$.
For high dimensional systems, this equation can become arbitrarily difficult, with no existing analytical solution for most cases. While several approximations can be made to simplify the scenario itself, the solution can also be approximated numerically without simplifying the scenario.

In our case, we will attempt to approximate the groundstate $|\psi_0\rangle$ with a state $|\psi_\Theta\rangle$, and then descend as close as possible to the real state. One way to do this is by minimizing energy eigenvalue as, assuming the eigenstates of $\hat{H}$ form a basis for $\mathcal{H}$, we have that:

$$\boxed{\langle\psi_\Theta|\,\hat{H}\,|\psi_\Theta\rangle = \sum_{i,j=0}^{\infty} \bar{\alpha}_i\,\alpha_j\,\langle\psi_i|\,\hat{H}\,|\psi_j\rangle = \sum_{i=0}^{\infty} |\alpha_i|^2 E_i \geq \sum_{i=0}^{\infty} |\alpha_i|^2 E_0 = E_0} \tag{2}$$

We can therefore approximate the groundstate energy simply by minimizing the expectation value $\langle\hat{H}\rangle_{\psi_\Theta}$. This is often reffered to as the **Variational Principle**.
If we project to a hyperplane orthogonal to the corresponding approximated groundstate $|\widetilde{\psi}_0\rangle$, we have a similar scenario for the next smallest eigenvalue, and can therefore repeat the process to calculate an arbitrary number of eigenstates and eigen-energies.
In fact, given a basis for a problem $\{|\varphi_i\rangle\}_i$, we can define the (possibly infinite dimensional) matrices $\widetilde{H}$ and $S$ with $(\widetilde{H})_{ij} = \langle\varphi_i|\,\hat{H}\,|\varphi_j\rangle$ and $(S)_{ij} = \langle\varphi_i|\varphi_j\rangle$, the secular equations [4] give the following equation for the eigen-energies:

$$\det\left(\widetilde{H} - E\cdot S\right) \tag{3}$$

Which, when $\{|\varphi_i\rangle\}_i$ are orthonormal, have $S$ become the identity matrix and therefore equation (3) becomes analogous to finding the eigenvalues of $\widetilde{H}$. However, with an infinite basis these approaches are not in general solvable numerically. A possible solution

is to truncate the degree to which we consider the eigensates. That is, we define a truncation parameter $N_{\max}$ such that we redefine the approximated state as

$$|\psi_\Theta\rangle \coloneqq \sum_{k=0}^{N_{\max}} \alpha_k |\psi_k\rangle$$

Then the problem of finding the energies amount to finding eigenvalues of a finite dimensional matrices, which can be approximated numerically. The energies found through the eigenvalues are only approximations to the actual energies, with the errors of these approximations depending on $N_{\max}$.
It has been shown that, while the energy approximations converge as $N_{\max} \to \infty$, this solution suffers from the curse of dimensionality: the parameter $N_{\max}$ depends exponentially on the number of dimensions $d$ of the system for convergence. A possible solution to this problem rises from Machine Learning.

## 1.2 Machine Learning

To avoid increasing the parameter $N_{\max}$ exponentially in relation to $d$, one could consider adding more flexibility to the basis. We will consider coordinate space to achieve this ($|\psi\rangle \mathrel{\widehat{=}} \psi(x)$). Given a basis $\{\varphi_i\}_i$, we define the augmented basis elements using a coordinate transformation.

$$\varphi_i^A(x) = \varphi_i(g(x)) \cdot \sqrt{\det \frac{\mathrm{d}g}{\mathrm{d}x}} \tag{4}$$

where $\sqrt{\det \frac{\mathrm{d}g}{\mathrm{d}x}}$ is the square root of the determinant of the Jacobian of this coordinate transformation, which is only used to preserve normalization:

$$\begin{aligned}\langle \varphi_i^A | \varphi_j^A \rangle &= \int \varphi_i^A(x) \cdot \varphi_j^A(x) \mathrm{d}x = \int \varphi_i(g(x)) \cdot \varphi_j(g(x)) \left(\frac{\mathrm{d}g}{\mathrm{d}x}\right) \mathrm{d}x \\ &= \int \varphi_i(g) \cdot \varphi_j(g) \left(\frac{\mathrm{d}g}{\mathrm{d}x}\right) \cdot \frac{\mathrm{d}x}{\mathrm{d}g} \mathrm{d}g = \langle \varphi_i | \varphi_j \rangle = 1\end{aligned}$$

We can notice that defining a different basis through this form, instead of some, albeit more flexible, arbitrary neural network $\{\mathrm{NN}_i(x)\}_{i=0}^{\infty}$ ensures two things:

- The augmented basis elements $\{\varphi_i^A\}_i$ constitute a basis for the space $\mathcal{H}$ when considered without a truncation parameter (which is not given for any set $\{\mathrm{NN}_i\}_i$)
- Orthonormality is preserved between the augmented basis elements $\{\varphi_i^A\}_i$

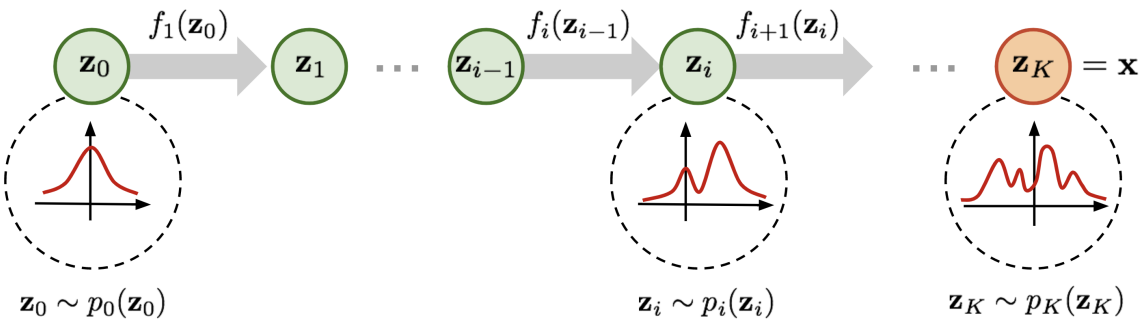


Figure 1: Normalizing Flows Standard Layout [5]

We also need to impose two further restrictions on the function $g$ for the previous manipulation to be valid:

- $g$ must have an inverse
- $g$ must be differentiable and have a differentiable inverse ($g$ must be a diffeomorphism)

The benefits given using this approach might be subtle: orthonormality preservation is needed to maintain the equivalence between the secular equations and diagonalizing $\widetilde{H}$, while having basis preservation is needed for the convergence of the approximated eigenvalues to the actual eigenvalues as $N_{\max} \to \infty$.

These requirements lead into a possible model for $g$: Normalizing Flows. This model is entirely based on finding diffeomorphisms $\{f_k\}_{k=1}^n$ and composing them (as diffeomorphisms are closed under composition) as shown in Fig. 1.

$$\boldsymbol{z} = (f_n \circ f_{n-1} \circ \cdots \circ f_1)(\boldsymbol{x})$$

We can then define different schemes for normalizing flows to test the method.

### 1.2.1 RNVP model

The Real-valued Non-Volume-Preserving model is a type of normalizing flow in which each layer $f_k(x)$ is given by

$$g_k(x) = \mathcal{P}_k[x] + \mathcal{Q}_k[f_k(x)] \quad \text{with} \quad f_k(x) = e^{s_k(\mathcal{P}_k[x])} \odot x + t_k(\mathcal{P}_k[x]) \tag{5}$$

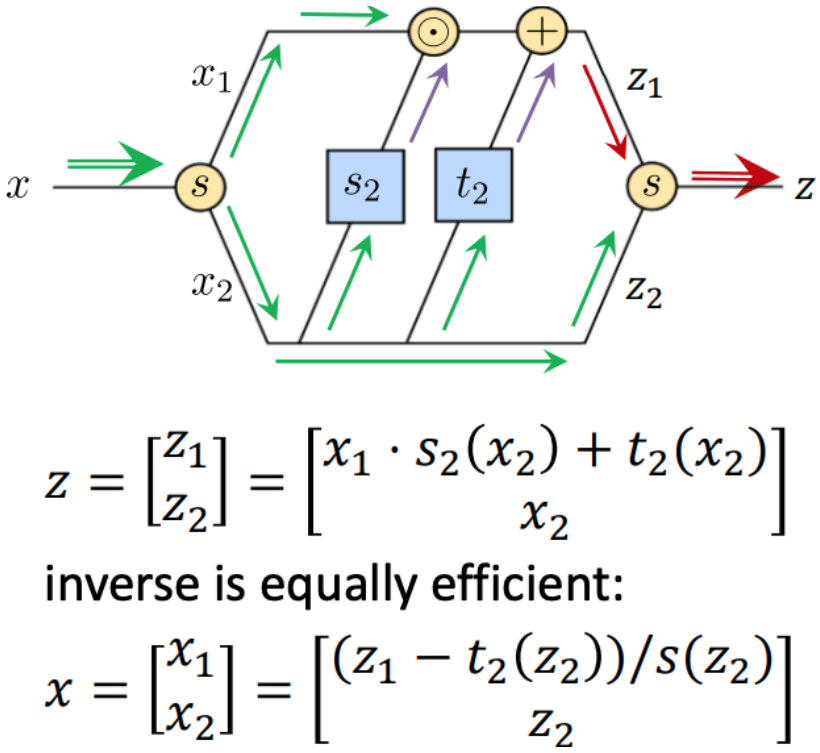


Figure 2: RNVP basic layer [3]

where $\odot$ denotes the Hadamard or elementwise product, $\mathcal{P}_k$ is a projection over $\lfloor \dim x/2 \rfloor$ of the components of $x$, $\mathcal{Q}_k \equiv \mathrm{id} - \mathcal{P}_k$ is an orthogonal projection over these components, and $t_k$ and $s_k$ are arbitrary differentiable functions with same input and output dimension (and with $s_k(x) \neq 0 \quad \forall x$). This function is invertible and differentiable, and a more conceptual illustration is provided in Fig. 2

## 1.3 Mathematics

One of the most important aspects of this approach is **convergence**. We need to assure that, at the limit of $N_{\max} \to \infty$, the approximation states and energies approach the true states and energies. The CMI Theory team is currently working on proving that a sufficient condition to guarantee convergence is to show that, if $g$ is the function used in the augmented basis, then the system converges to the real results if $g$ is a symbol for the Schwarz Space $\mathcal{S}$. That is:

$$f \circ g \in \mathcal{S} \quad \forall f \in \mathcal{S} \tag{6}$$

where the Schwarz space is defined as

$$\mathcal{S} \coloneqq \left\{ f \;\middle|\; \sup_{x \in \mathbb{R}^d} \left| x^\beta \frac{\partial^\alpha}{\partial x^\alpha} f \right| < \infty \quad \forall \alpha, \beta \in \mathbb{N}^d \right\} \tag{7}$$

Thus, to guarantee convergence using the augmented basis as $N_{\max} \to \infty$ with $g$ being the RNVP model, it is only needed to show that functions defined through the RNVP paradigm are symbols for the Schwarz space.

# 2 Implementation

## 2.1 The RNVP model

The whole implementation was performed using Jax and Flax [1], with the models following the standard torch-like structure. In particular, the function of the normalizing flow itself is given as

```
class RNVP(nn.Module):
    """Implement a model of RNVP without inheriting from the nn.Module class
    """

    num_layers: int
    dim: int
    hidden_lyrs = [32, 32, 32]

    """Args:
        num_layers (int): number of layers (affine transformations) in the flow
        dim (int): dimension of the input states
        masks (ndarray): arrays to 'mask' the dimensions that remain invariant on each layer
        hidden_lyrs (list): list of hidden layers sizes
    """

    def setup(self) -> None:
        feats_num = self.hidden_lyrs + [self.dim]
        self.t_nets = [MLP(feats_num) for _ in range(self.num_layers)]
        self.s_nets = [MLP(feats_num) for _ in range(self.num_layers)]
        return None

```

```
    @nn.compact
    def __call__(self, x: jnp.ndarray, mode='Forward') -> tuple:
        """Compute a forward or backward pass through the normalizing flow

        Args:
            x (jnp.ndarray): input vector

        Returns:
            tuple: mapped input and the logarithm of the determinant of the jacobian of the entire transformation
        """
        param_scale, epsi = 1e-1, 0.05
        masks = create_masks(self.num_layers,
                             self.dim, jax.random.PRNGKey(0))
        z = x
        if mode.casefold() == 'forward':
            for i in reversed(range(self.num_layers)):

                z_ = masks[i] * z
                s = self.s_nets[i](z_) * (1-masks[i])
                _s = truncated_exp(s+epsi)
                t = param_scale*self.t_nets[i](z_) * (1-masks[i])
                z = (1 - masks[i]) * (z - t) /_s + z_

        elif mode.casefold() == 'backward':
            z = x
            for i in range(self.num_layers):
                z_ = z*masks[i]
                s = self.s_nets[i](z_)*(1 - masks[i])
                _s = truncated_exp(s+epsi)
                t = param_scale*self.t_nets[i](z_)*(1 - masks[i])
                z = z_ + (1 - masks[i]) * (z * _s + t)

        else:
            raise ValueError('mode must be either "Forward" or "Backward"')

        return z
```

Listing 1: RNVP Model

Two things that are worth noting:

1. We employ a function named 'truncated_exp' rather than 'exp'. This is for convergence issues, as using the 'exp' function instead of some polynomial that shares the same behaviour close to 0 would make the RNVP model not necessarily a symbol of $\mathcal{S}$. This truncated exp is given by e.g.,

```
def truncated_exp(x):
    return 1+x+x**2/2+x**3/6+x**4/24+x**5/120+x**6/720
```

Listing 2: truncated exp function

2. The 'MLP' model used is given by:

```
class MLP(nn.Module):
    """MLP to be used for 't' and 's' functions in RNVP model"""

    feats_num: Sequence[int]
    activ_func: Callable = nn.sigmoid

    @nn.compact
    def __call__(self, input) -> jnp.array:
        for idx, feat in enumerate(self.feats_num):
            input = nn.Dense(features=feat)(input)
            if idx != len(self.feats_num)-1:
                input = self.activ_func(input)
        return input
```

Listing 3: MLP model

where using the sigmoid function (or any other activation function with bounded derivatives) is crucial ensuring the model is a symbol for $\mathcal{S}$

## 2.2 Integration

In the case of H2S, we decided to use the harmonic oscillator basis as a primitive basis for the problem. We are then able to use a Gauss-Hermit Quadrature to integrate over functions:

$$\int_{-\infty}^{\infty} f(x)e^{-x^2}\mathrm{d}x \approx \sum_{i=0}^{n} w_i f(x_i) \tag{8}$$

where $n$ is the number of quadrature points, $x_k$ is the $k$-th root of the $n$-th Hermite polynomial $H_n$, and $w_k = \frac{2^{n-1}n!\sqrt{\pi}}{n^2[H_{n-1}(x_k)]^2}$.
This is relevant, as the quadrature points can be calculated only once (under the augmented coordinates $g(x)$) and then used for every epoch.

## 2.3 Optimization

Using the matrix described in the Secular Eqs. (3) and the Gauss-Hermite integration in Eq. 8, we can calculate the loss function as:

$$\mathcal{L} = \sum_{\zeta \in \sigma(\widetilde{H})} \zeta$$

with $\sigma(\cdot)$ being the spectrum of an operator (or a usable subset of it). Here we have to use Gauss-Hermite quadrature in $y = g(x)$ to calculate the components of $\widetilde{H} = \langle \varphi_i^A | \, \hat{H} \, | \varphi_j^A \rangle = \int \mathrm{d}y \hat{H}(y) \varphi_i^A(y) \varphi_j^A(y)$ (mapping the quadrature points to the original space using $g^{-1}$ and performing the appropriate change of variables). Note that this is not equivalent to talking the trace, as sometimes we are only interested in minimizing an initial subset of the spectrum of $\hat{H}$ (to find the lower energies of the operator).

Additionally, a modified version of the RNVP model was used for $g$, were instead of simply using an RNVP function, a different paradigm was made:

$$g = L_{\text{fixed}}^{-1} \circ \tanh \circ \text{RNVP} \circ \tanh^{-1} \circ L_{\text{fixed}} \circ L_{\text{learn}}$$

where $L_{\text{fixed}}$ is a fixed (no optimizable parameters) linear operation which scales the input vectors to $[-1, 1]$, and $L_{\text{learn}}$ is a learnable linear operation which is intended to best fit the input values to the primitive basis.

# 3 Results

## 3.1 H2S Simulation

The code was implemented in an H2S molecule, where the parameter $N_{\max}$ was varied over a range of numbers and the convergence of the energy approximations was measured. The RNVP model was compared to a model using only $L_{\text{learn}}$ and no neural network (labeled 'LIN'), and to a different paradigm named Invertible ResNet (labeled 'LIB'). The results for a fixed angle are shown in Fig. 3, and they clearly show that both IResNet and RNVP outperform the convergence rate of 'LIN' for the first 10 energy levels.

A different numerical test was applied to the entire vibrational system for H2S (without a fixed angle). The results are shown in Fig. 4. Here we can observe, to a larger extent

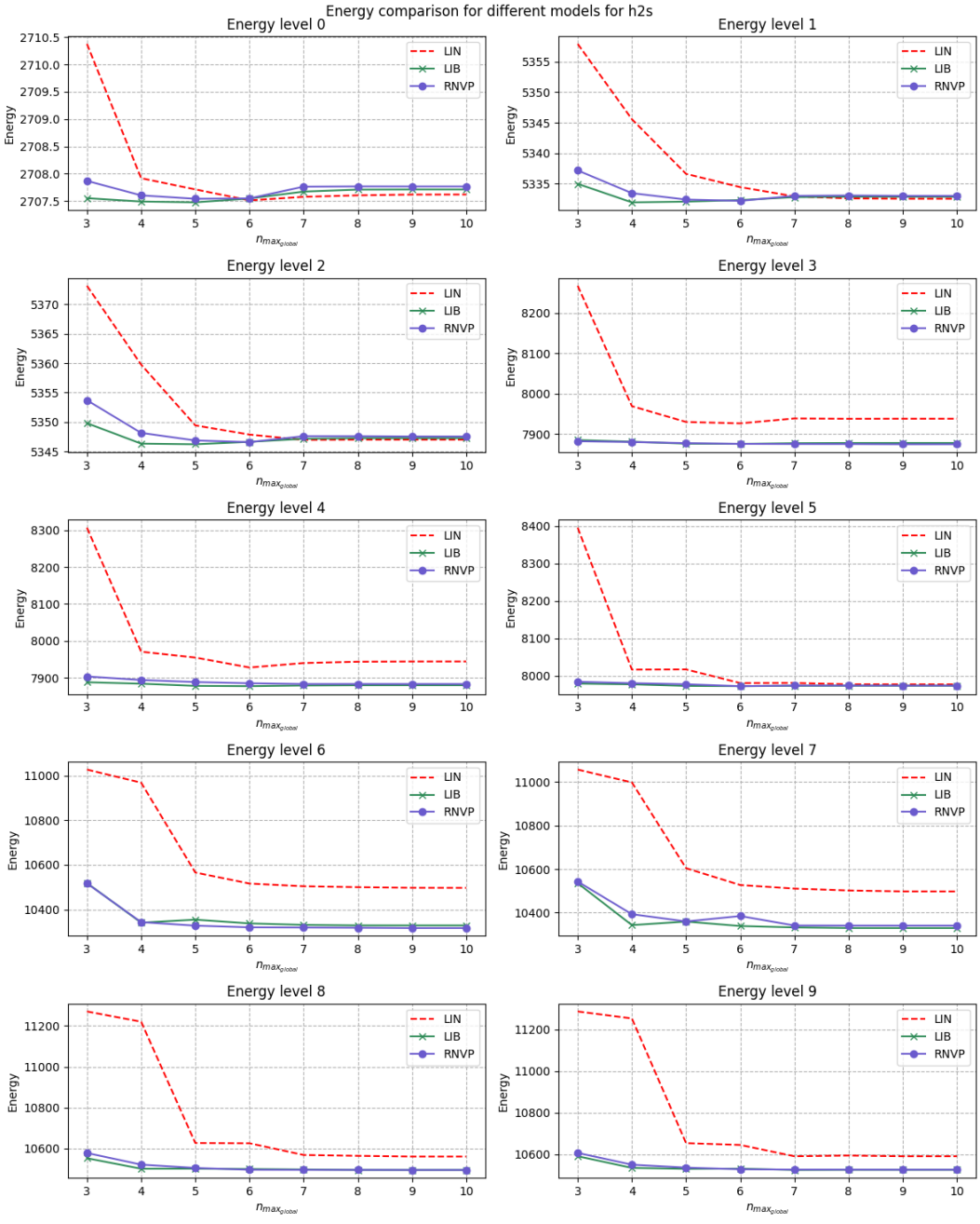


Figure 3: H2S energy levels for stretching system

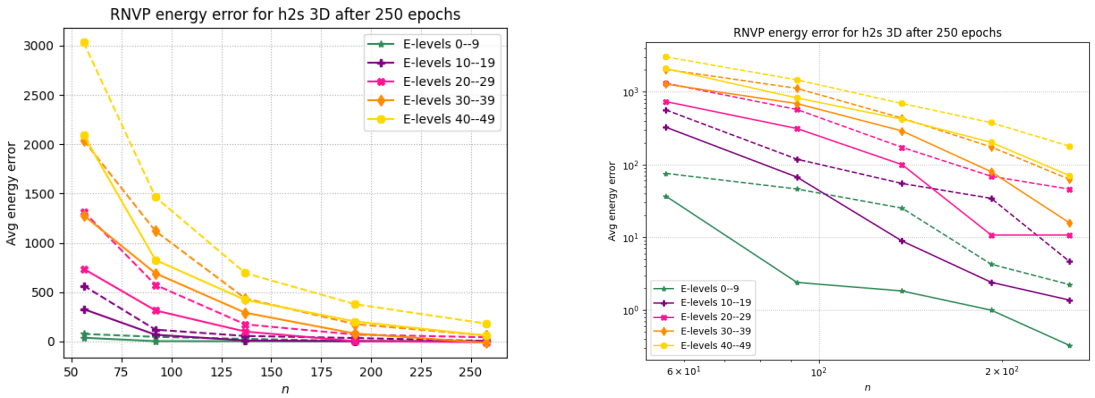


Figure 4: Energy error for H2S vibrational spectra (RNVP:— and LIN:- -)

(50 energy levels), that the optimization brought by RNVP outperforms LIN.

### 3.2 Mathematical Results

On a more theoretical level, it was also shown that augmented basis for the RNVP paradigm has the desirable convergence properties. More specifically, it was shown that the RNVP model, with its neural networks being Multi-Layer Perceptron models using sigmoid activation functions, and its scaling function being a positive polynomial of this neural network, is a symbol for $\mathcal{S}$.
With this, defining the augmented basis using the RNVP model guarantess convergence to the real eigenstate as $N_{\max} \to \infty$.

## 4 Conclusion

Using the augmented model with RNVP-paradigm functions was theoretically shown to yield results that converge to the actual eigenfunctions corresponding to a hamiltonian $\hat{H}$, and the simulations suggest that it outperforms using no neural networks (at least in H2S case). Further tests in higher-dimensional systems should be performed to analyse convergence in dependence of dimension and extrapolate its behaviour.

# References

# References

[1] The JAX authors. *Jax reference documentation*. 2020. URL: https://jax.readthedocs.io/en/latest/.

[2] Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: (2019). DOI: 10.1109/TPAMI.2020.2992934. eprint: arXiv:1908.09257.

[3] Ullrich Köthe. *Introduction to Normalizing Flows*. Mar. 2021.

[4] Libretexts. *7.2: Linear variational method and the secular determinant*. July 2022.

[5] Phillip Lippe. *Tutorial 11: Normalizing flows for image modeling*. 2022. URL: https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial11/NF_image_modeling.html.