Masterarbeit
Zur Erlangung des akademischen Grades Master in Physik

Simulating the Quantum Rotor with Normalizing Flows

eingereicht von: Fabian Dechent

Gutachter:	Prof. Agostino Patella Dr. Johannes Weber
Betreuer:	Dr. Pan Kessel

Eingereicht am Institut für Physik der Humboldt Universität zu Berlin am:

Abstract

Fabian Dechent

Simulating the Quantum Rotor with Normalizing Flows

Normalizing Flows as a class of generative models can be trained to sample from an arbitrary, high-dimensional probability density, which must only be evaluated up to normalization. Consequently, they have been used to estimate expectation values for discretized physical systems. While ultimately the goal is to improve upon Monte Carlo Markov Chain methods for elaborate theories like QCD, current research has focused on proof of concept studies and architecture proposals for adaptation to the manifolds of the field variables. In this work, Normalizing Flows are applied to the Topological Quantum Rotor. Albeit being a rather simple system, it serves as a fitting test bed through its non-trivial topology. The concept behind Normalizing Flows as well as the theory of the Rotor system are introduced. Various Rotor specific architecture adaptations to the flow model are analyzed and compared regarding susceptibility to mode-collapse and scalability with respect to lattice parameters. Training alterations, which are designed to improve the Normalizing Flow's mode collapse susceptibility, are introduced and finally, the results are set side by side with the performance of HMC simulations.

The code can be found on GitHub: https://github.com/quant-sic/nfqr

Contents

List of Figures

List of Tables

List of Symbols

$\mathscr{Z}$	Partition Function
χ_t	Topological Susceptibility
Q	Topological Charge
Q_{disc}	Discretized Topological Charge
S	Action
S_{disc}	Discretized Action
I	Moment of Inertia for the Rotor
M	Mass of the Rotor
β	Coupling of the Rotor
τ	Lattice Spacing
Λ	Lattice
Ω	Space of Lattice Variables
T	Total Time Extent of the Lattice
D	The System's Size $\lvert\Lambda\rvert$
γ_ρ	Residual Coefficient Scheduling Reduction Factor
γ_η	Reduction for Factor for ReduceOnPlateau Learning Rate Scheduling
γ_{Q^2}	Damping Coefficient for Q^2 in Scheduling
η	Learning Rate
η_β	Step Size for β–Scheduling
η_α	Step Size for Mixing Parameter α in Bidirectional Training
α	Mixing Parameter α in Bidirectional Training
ε_{Q^2}	Relative Error Threshold on Q^2
γ_β	Relative Weight Term in the Reverse KL divergence
m_{Q^2}	Slope of Time Series of logged $\overline{Q^2}$ Value
ESS_p	Effective Sampling Size estimated with target density
ESS_q	Effective Sampling Size estimated with surrogate density
$\tau_{int}^{Q^2}$	Integrated Autocorrelation Time for Markov Chain of Variable Q^2
a	Acceptance Rate of Markov Chain
z	Scaling Exponent for an exponential Fit

List of Abbreviations

QCD	Quantum Chromodynamics
CLP	Constant Line of Physics
NIS	Neural Importance Sampling
NMCMC	Neural Markov Chain Monte Carlo
MCMC	Markov Chain Monte Carlo
HMC	Hamiltonian or Hybrid Monte Carlo
Wolff	Wolff Cluster Algorithm
ML	Machine Learning
DL	Deep Learning
KL	Kullback-Leibler
ESS_q	Effective Sampling Size estimated with surrogate density
ESS_p	Effective Sampling Size estimated with target density
RQS	Rational Quadratic Spline
CS	Circular Spline
NCP	Non Compact Projection
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
RNN	Recurrent Neural Network

Chapter 1

Introduction

The regularization of Quantum Field Theories using a lattice provides the possibility to evade divergent expectation values for physical observables by dictating a minimum distance between space-time points and limiting the volume. Real physics can be recovered when the continuum limit for the lattice is taken. A particularly useful aspect of the lattice regulator is that a discretized theory allows for numerical methods that are otherwise not applicable. Especially, but not exclusively, Monte Carlo Markov Chain (MCMC) methods are being used.

Inherent to some widely used MCMC methods like the Hamiltonian Monte Carlo (HMC) [1], we encounter an unfavorable scaling in computational cost towards the continuum limit. This phenomenon of *critical slowing down* [2] becomes particularly obstructive for topological observables when periodic boundary conditions are used [3]. The local MCMC method gets stuck in the current topological sector, which is called *topological freezing*. Consequently, there is a continued search for alternative algorithms.

In the rapidly developing field of Machine Learning, neural networks have been applied to analyze physical problems with astounding success in comparison to previously known methods. The class of generative models, which effectively sample from a desired density, can similarly to MCMC methods be applied to study field theories on the lattice. For this purpose Normalizing Flows, proposed in [4], are among the most studied representatives.

Similar to the concept of a trivializing map suggested by [5], a Normalizing Flow approximately maps the highly correlated field variables of a theory onto decoupled ones. At this point, sampling efficiency can be drastically increased.

Crucial challenges for their application are the adaptation of the Machine Learning model to the manifolds of the field variables [6–8] and studying scaling [9] and performance on highly multi-modal target densities [10].

To this end in this thesis, we will work with the *Topological Quantum Rotor* system. Conceptually, it is quite an inelaborate model and can be solved analytically. This makes it possible to compute observables of interest exactly, and we can easily gain insight into the generative model's performance. What makes the Rotor interesting is its topological qualities. For periodic boundary conditions on the lattice and towards

the continuum limit, its density - given by the Boltzmann weight - becomes severely multi-modal. This behavior, which is shared with non-linear σ-models and gauge theories, results in the diverging computational cost for local MCMC methods. And thus it can be used as a useful test bed to investigate novel algorithms.

The beginning of this thesis is dedicated to deriving the action and studied observables of the Rotor system. Then, we will present the sampling-based approach of numerically calculating observables for the discretized system. Here, we will also briefly introduce MCMC methods, to which the generative model is compared later on. Afterward, we will introduce the architecture of Normalizing Flows and describe approaches to adapt the model to the Rotor. The results of the Normalizing Flow model will be evaluated and compared to the MCMC-based approach.

Chapter 2

Quantum Rotor

In the first chapter of this thesis, we will investigate the *Quantum Rotor*. To start, the system is discretized, which is an integral step for our numerical studies. Here, the discretized action as well as the discretized topological charge are derived. Afterward, we develop a closed-form expression for the topological susceptibility χ_t, which is the main observable estimated numerically.

2.1 Quantum Rotor System

The model itself is straightforward to introduce: For this, consider a mass M revolving on a circle with a radius R as depicted in Figure 2.1a. For this thesis, it is sufficient to restrict the analysis to the free case. Thus, there is no potential involved and the Lagrangian of the system is

$$\mathcal{L}(x(t), y(t)) = \frac{M}{2}\left(\dot{x}(t)^2 + \dot{y}(t)^2\right) = \frac{I}{2}\dot{\phi}^2(t) = \mathcal{L}(\phi(t)), \tag{2.1}$$

with a moment of inertia $I = MR^2$ and the angle on the circle ϕ. We work in natural units ($\hbar = 1, c = 1$). For a total time evolution extent $T \in \mathbb{R}$, the action is given as

$$S[\phi] = \int_0^T \mathcal{L}(\phi(t))\, dt = \frac{I}{2}\int_0^T \dot{\phi}^2(t)\, dt. \tag{2.2}$$

Keep in mind that in this case ϕ is a function $\phi : [0, T] \mapsto [0, 2\pi)$ describing a path on the circle $\mathbb{S}^1$, such that the enpoints 0 and 2π are identified. For numerical simulations, we need to discretize the temporal direction for this system. Corresponding to a fixed number of sampling points $D \in \mathbb{N}$ and thus a fixed increment in time $\tau = \frac{T}{D}$, we define a one-dimensional lattice $\Lambda = \{i\tau | i \in \{1, \ldots, D\}\}$ with $|\Lambda| = D$. A realization of angles on the lattice can then be represented by a vector $\boldsymbol{\phi} \in \Omega = [0, 2\pi)^D$, where for all $i \in \{1, \ldots, D\}$ component $\phi_i = \phi(i\tau)$. For a visualization of the lattice construction, see Figure 2.1b. We can impose periodic boundary conditions on the lattice in the

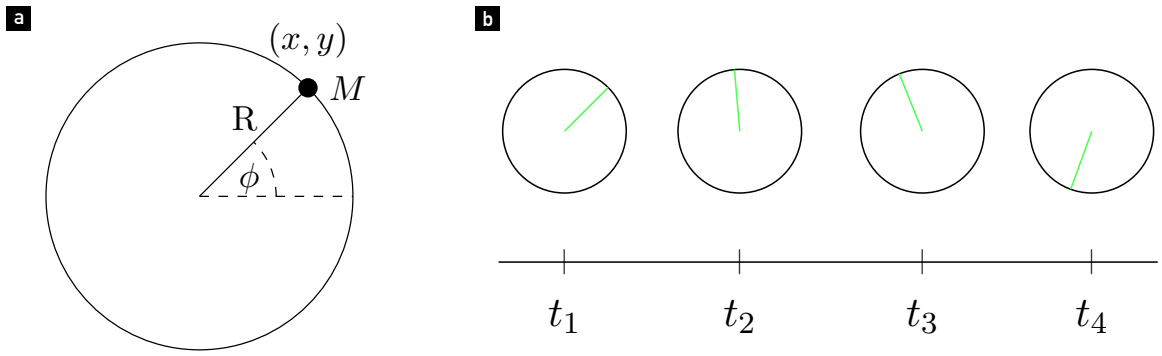


Figure 2.1: (a) Depiction of the Quantum Rotor system. The position $\phi = 0$ is chosen arbitrarily. (b) Depiction of a one-dimensional lattice of angles for the Quantum Rotor system.

sense that

$$\begin{aligned} \phi(\tau) = \phi_1 \overset{!}{=} \phi_{D+1} = \phi((D+1)\tau), \\ \phi_j = \phi_{\,\mathrm{mod}\,(j)_D}, \qquad j \in \mathbb{Z}. \end{aligned} \tag{2.3}$$

Ultimately we would like to calculate observables for the time-discretized Rotor. The path-integral formalism not only provides this possibility, but its derivation also elucidates the system's dependence on the boundary conditions of the circle. For this, we'll first need to determine the Hamiltonian and the Hilbert space of the system. The use of the Legendre transformation allows deriving the Hamiltonian H from the Lagrangian L, i.e.

$$H = \frac{L^2}{I} - \mathcal{L} = \frac{L^2}{2I}. \tag{2.4}$$

Canonical quantization proceeds by replacing the angular momentum $L = I\dot{\phi}$ with its corresponding operator $\hat{L} = -i\partial_\phi$ that satisfies the commutation relation

$$\left[\hat{\phi}, \hat{L}\right] = i, \tag{2.5}$$

where $\hat{\phi}$ is the position operator on $\mathbb{S}^1$. The eigenvectors of this Hamiltonian are equal to the eigenvectors of the angular momentum operator $\hat{L}$, i.e. $|q\rangle$ with $\hat{L}\,|q\rangle = q\,|q\rangle$ with $q \in \mathbb{R}$. The real space representation is

$$\langle \phi | q \rangle = \frac{e^{iq\phi}}{\sqrt{2\pi}}, \tag{2.6}$$

where $|\phi\rangle$ is an eigenstate of the position operator $\hat{\phi}$. Importantly, since the angle ϕ is defined on the finite interval $[0, 2\pi)$, where the endpoints are identified, boundary conditions for the wavefunctions need to be imposed. A full revolution around the

circle ought to result in an additionally acquired phase θ[1]. If $\theta = 0$, this corresponds to periodic boundary conditions. Therefore, possible values for q are restricted such that

$$\langle \phi + 2\pi | q \rangle \overset{!}{=} e^{i\theta} \langle \phi | q \rangle \Longleftrightarrow q = \frac{\theta}{2\pi} + k, \qquad k \in \mathbb{Z}. \tag{2.7}$$

We can equivalently introduce the θ dependence into the angular momentum operator, such that $\hat{L}_\theta = -i\partial_\phi + \frac{\theta}{2\pi}$ (see e.g. [11]). Consequently, we can introduce the set of eigenstates $|k\rangle$ with $k \in \mathbb{Z}$ for the operator $\hat{L}_0$. Explicitly introducing the θ-dependence in the Hamiltonian

$$\hat{H}(\theta) = \frac{1}{2I}\hat{L}_\theta^2 = \frac{1}{2I}\left(-i\partial_\phi + \frac{\theta}{2\pi}\right)^2, \tag{2.8}$$

we see that

$$\langle k | \hat{H}(\theta) | k \rangle = \langle q | \hat{U}^\dagger \hat{H}(\theta) \hat{U} | q \rangle = \langle q | \hat{H}(0) | q \rangle \tag{2.9}$$

for the unitary operator $\hat{U} = \exp\left(-i\frac{\theta}{2\pi}\hat{\phi}\right)$.

Using the spectral representation of this Hamiltonian, we can then calculate the transfer matrix element $\langle \phi_j | \hat{\mathcal{T}}(\theta) | \phi_{j-1} \rangle = \langle \phi_j | e^{-\tau \hat{H}(\theta)} | \phi_{j-1} \rangle$ corresponding to an evolution of step size τ from some angle ϕ_{j-1} at time $(j-1)\tau$ to ϕ_j at time $j\tau$:

$$\begin{aligned} \langle \phi_j | e^{-\tau \hat{H}(\theta)} | \phi_{j-1} \rangle &= \sum_{k_j \in \mathbb{Z}} \langle \phi_j | k_j \rangle \langle k_j | \phi_{j-1} \rangle e^{-\frac{\tau}{2I}(k_j + \frac{\theta}{2\pi})} \\ &\overset{(2.6)}{=} \sum_{k_j \in \mathbb{Z}} \frac{e^{ik_j(\phi_j - \phi_{j-1})}}{2\pi} e^{-\frac{\tau}{2I}\left(k_j + \frac{\theta}{2\pi}\right)^2} \\ &=: \sum_{k_j \in \mathbb{Z}} f(k_j). \end{aligned} \tag{2.10}$$

To get rid of the sum over the momenta, one can rewrite this by applying a Poisson resummation $\sum_{k\in\mathbb{Z}} f(k) = \sum_{n\in\mathbb{Z}} \mathcal{F}[f](n)$. We have defined $\mathcal{F}[f]$ as the Fourier transform of a function f

$$\begin{aligned} \hat{f}(k) := \mathcal{F}[f](k) &= \int_{-\infty}^{\infty} dx\, e^{-i2\pi kx} f(x), \\ f(x) = \mathcal{F}^{-1}\left[\hat{f}\right](x) &= \int_{-\infty}^{\infty} \frac{dk}{2\pi}\, e^{i2\pi kx} \hat{f}(k). \end{aligned} \tag{2.11}$$

[1] Vectors in the state space are well-defined up to a phase. An overall added phase still corresponds to the same unchanged state

So we need to calculate the Fourier-transformed function

$$\begin{aligned}\mathcal{F}[f](n_j) &= \int_{-\infty}^{\infty} \frac{dk_j}{2\pi}\ \exp\left[-i2\pi k_j n_j + ik_j(\phi_j - \phi_{j-1}) - \frac{\tau}{2I}\left(k_j + \frac{\theta}{2\pi}\right)^2\right] \\ &= \exp\left[i\frac{\theta}{2\pi}(\phi_j - \phi_{j-1} - 2\pi n_j)\right] \\ &\quad \times \int_{-\infty}^{\infty} \frac{dk_j}{2\pi}\ \exp\left[ik_j(\phi_j - \phi_{j-1} - 2\pi n_j) - \frac{\tau}{2I}{k_j}^2\right] \\ &= \sqrt{\frac{I}{2\pi\tau}} \exp\left[i\frac{\theta}{2\pi}(\phi_j - \phi_{j-1} - 2\pi n_j) - \frac{I[\phi_j - \phi_{j-1} - 2\pi n_j]^2}{2\tau}\right] \\ &=: g(\phi_j - \phi_{j-1} - 2\pi n_j).\end{aligned} \tag{2.12}$$

Let's extend this result to the evolution along the full time extent T between arbitrary angles ϕ_0 and ϕ_D. For this the transition probability

$$\langle\phi_D|\, e^{-T\hat{H}(\theta)}\, |\phi_0\rangle = \prod_{i=1}^{D-1}\left[\int_0^{2\pi} d\phi_i\right] \prod_{j=1}^{D} \langle\phi_j|\, e^{-\tau\hat{H}(\theta)}\, |\phi_{j-1}\rangle \tag{2.13}$$

ought to be evaluated. Here we can plug in the result given in Equation 2.12, such that the formula now reads

$$\langle\phi_D|\, e^{-T\hat{H}(\theta)}\, |\phi_0\rangle = \prod_{i=1}^{D-1}\left[\int_0^{2\pi} d\phi_i\right] \prod_{j=1}^{D} \sum_{n_j=-\infty}^{\infty} g(\phi_j - \phi_{j-1} - 2\pi n_j). \tag{2.14}$$

One can simplify this by exchanging the order between $\sum_{n_j}$ and $\prod_j$:

$$\langle\phi_D|\, e^{-T\hat{H}(\theta)}\, |\phi_0\rangle = \sum_{n_D=-\infty}^{\infty} \prod_{i=1}^{D-1}\left[\sum_{n_i=-\infty}^{\infty} \int_0^{2\pi} d\phi_i\right] \prod_{j=1}^{D} g(\phi_j - \phi_{j-1} - 2\pi n_j). \tag{2.15}$$

Notice how all angle differences $\phi_j - \phi_{j-1}$ are shifted by multiples of 2π, which results in the integration over the entire real line. This can be made manifest if one accumulates all shifts at the end of the path with the substitution

$$\phi'_j = \phi_j - 2\pi \sum_{k=1}^{j} n_k. \tag{2.16}$$

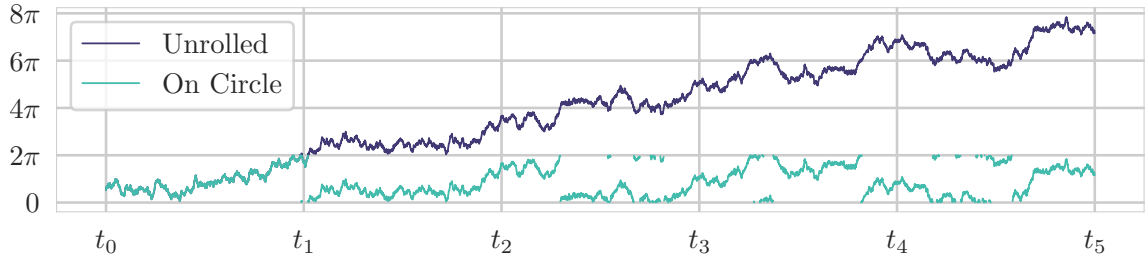


Figure 2.2: Depicts the revolutions around the circle unrolled onto the real line. The green line shows the angle on the circle, while the blue line shows the same angle evolution unrolled onto the real line. While it would be impossible to determine the number of revolutions for the green line just by knowing ϕ_0 and ϕ_5, for the blue trajectory this is possible.

Then using the linearity of the integral:

$$\begin{aligned}\langle\phi_D|\, e^{-T\hat{H}(\theta)}\,|\phi_0\rangle = &\sum_{n_D=-\infty}^{\infty}\sum_{n_1,\dots,n_{D-1}=-\infty}^{\infty}\prod_{i=1}^{D-1}\left[\int_{-2\pi\left(\sum_{k=2}^{i} n_k\right)}^{-2\pi\left(\sum_{k=2}^{i} n_k - 1\right)} d\phi_i'\right] \\ &\times \prod_{j=1}^{D} g\left(\phi_j' - \phi_{j-1}' - 2\pi\delta_{D,j}\sum_{k=1}^{j} n_k\right).\end{aligned} \tag{2.17}$$

Notice that since we do not integrate over the angles ϕ_D and ϕ_1, in the last factor the shifts do not cancel, which gives rise to the term $2\pi\delta_{D,j}\sum_{k=1}^{j} n_k$. Since all the series indices have infinite ranges, it is possible to reindex $n_i' = \sum_{k=1}^{i} n_k$ for $i \in \{1,\dots,D\}$ without having to re-evaluate the index ranges:

$$\begin{aligned}\langle\phi_D|\, e^{-T\hat{H}(\theta)}\,|\phi_0\rangle &= \sum_{n_D'=-\infty}^{\infty}\sum_{n_1',\dots,n_{D-1}'=-\infty}^{\infty}\prod_{i=1}^{D-1}\left[\int_{-2\pi n_i'}^{-2\pi\left(n_i'-1\right)} d\phi_i'\right] \\ &\quad\times \prod_{j=1}^{D} g\left(\phi_j' - \phi_{j-1}' - 2\pi\delta_{D,j} n_D'\right) \\ &= \sum_{M=-\infty}^{\infty}\prod_{i=1}^{D-1}\left[\int_{-\infty}^{\infty} d\phi_i\right]\prod_{j=1}^{D} g\left(\phi_j - \phi_{j-1} + 2\pi\delta_{D,j} M\right).\end{aligned} \tag{2.18}$$

In the last step, the summations were acted out for $k \in \{1,\dots,D-1\}$, which results in integrations over the entire real line. We also renamed the index n_D and changed the sign in front of the additional term in the function g, which is an arbitrary choice. The method of unrolling the revolutions of the Rotor around the circle onto the entire real line is visualized in Figure 2.2. At this point, we plug in the specific expression for

g, and we find the discretized action $S^{(M)}_{disc}$ and the discretized topological charge $Q^{(M)}_{disc}$:

$$\begin{aligned}\prod_{j=1}^{D} g\left(\phi_j - \phi_{j-1} - 2\pi\delta_{D,j}M\right) &\propto e^{\left[i\frac{\theta}{2\pi}\tau\sum_{j=1}^{D}\frac{\phi_j-\phi_{j-1}+2\pi\delta_{D,j}M}{\tau} - \frac{I}{2}\tau\sum_{j=1}^{D}\left(\frac{\phi_j-\phi_{j-1}+2\pi\delta_{D,j}M}{\tau}\right)^2\right]} \\ &= e^{\left[i\theta Q^{(M)}_{disc}(\boldsymbol{\phi}) - S^{(M)}_{disc}(\boldsymbol{\phi})\right]}.\end{aligned} \tag{2.19}$$

We have defined

$$\begin{aligned}Q^{(M)}_{disc}(\boldsymbol{\phi}) &= \frac{\tau}{2\pi}\sum_{j=1}^{D}\frac{\phi_j - \phi_{j-1} + 2\pi\delta_{D,j}M}{\tau}, \\ S^{(M)}_{disc}(\boldsymbol{\phi}) &= \frac{I}{2}\tau\sum_{j=1}^{D}\left(\frac{\phi_j - \phi_{j-1} + 2\pi\delta_{D,j}M}{\tau}\right)^2.\end{aligned} \tag{2.20}$$

For the sake of readability, we define the measure

$$\int \mathcal{D}\phi_j = \int_{-\infty}^{\infty} d\phi_j \left(\frac{I}{2\pi\tau}\right)^{\frac{1}{2}}. \tag{2.21}$$

Putting it all together, the final result with finite time steps with arbitrary ϕ_D and ϕ_0 is

$$\langle\phi_D|\, e^{-T\hat{H}(\theta)}\,|\phi_0\rangle = \sum_{M=-\infty}^{\infty}\prod_{j=1}^{D-1}\int \mathcal{D}\phi_j\ \exp\left[i\theta Q^{(M)}_{disc}(\boldsymbol{\phi}) - S^{(M)}_{disc}(\boldsymbol{\phi})\right]. \tag{2.22}$$

In the course of the calculation, we have arrived at time-discretized expressions for the topological charge and the action. We'll see that these expressions do match the continuum expressions in the limit $\tau \to 0$. There are however a few alterations to these formulas to make them more practical and for which the continuum expressions are still obtained for $\tau \to 0$. As a consequence, for a discrete lattice one deals with a different model.

Of course, within the scope of this thesis, it does not matter, which exact system is simulated, as long as the properties of the original model that render the simulation challenging are preserved.

In the case of the topological charge

$$Q^{(M)}_{disc}(\boldsymbol{\phi}) = \frac{\tau}{2\pi}\sum_{j=1}^{D}\frac{\phi_j - \phi_{j-1} + 2\pi\delta_{D,j}M}{\tau} = \frac{\phi_D - \phi_0 + 2\pi M}{2\pi} \tag{2.23}$$

was obtained in the path integral derivation. A more practical definition of a topological charge is the *geometric* or *classical* topological charge [11]

$$Q_{disc}(\boldsymbol{\phi}) = \frac{1}{2\pi}\sum_{j=1}^{D} \mathrm{mod}(\phi_j - \phi_{j-1})_{(-\pi,\pi]}. \tag{2.24}$$

This definition strips away the ambiguity of which paths are taken in between the sampling points and the interpolation is taken to be the shortest arc between the points [11]. Most importantly, in the continuum limit, this will match the topological charge.

For the discretized action

$$\begin{aligned} S^{(M)}_{disc}(\boldsymbol{\phi}) &= \frac{I}{2}\tau \sum_{j=1}^{D}\left(\frac{\phi_j - \phi_{j-1} + 2\pi\delta_{D,j}M}{\tau}\right)^2 \\ &= \frac{I}{2}\tau \sum_{j=1}^{D-1}\left(\frac{\phi_j - \phi_{j-1}}{\tau}\right)^2 + \frac{I}{2\tau}((\phi_D + 2\pi M) - \phi_{D-1})^2 \end{aligned} \tag{2.25}$$

was obtained. In the continuum limit $\exp\left[-S^{(M)}_{disc}(\boldsymbol{\phi})\right]$ will be suppressed for all M that deviate from the exact number of net full revolutions around the circle along the path $\boldsymbol{\phi}$, as this minimizes $((\phi_D + 2\pi M) - \phi_{D-1})^2$ with $\phi_D \in [0, 2\pi)$ and $\phi_{D-1} \in \mathbb{R}$. Alternatively, we define the *standard* action

$$\begin{aligned} S_{disc}(\boldsymbol{\phi}, \beta) &= \frac{I}{2}\tau \sum_{j=1}^{D}\left(\frac{\mathrm{mod}(\phi_j - \phi_{j-1})_{(-\pi,\pi]}}{\tau}\right)^2 \\ &= \beta \sum_{j=1}^{D}(1 - \cos(\phi_j - \phi_{j-1})) + \mathcal{O}\left(\mathrm{mod}(\phi_j - \phi_{j-1})_{(-\pi,\pi]}{}^4\right), \end{aligned} \tag{2.26}$$

where again the shortest arc is already encoded. Here the dimensionless coupling $\beta = \frac{I}{\tau}$ was introduced. Note that β is indeed dimensionless because in the used natural units, I has the unit of a time. With this discretization, an error $\mathcal{O}(\tau^2)$ compared to the continuous system is accepted:

$$\frac{1 - \cos(\phi(x) - \phi(x-\tau))}{\tau^2} = \frac{1 - \left(1 - \frac{1}{2}\left(\frac{d\phi}{dx}\right)^2\tau^2 + \mathcal{O}\left(\tau^4\right)\right)}{\tau^2} = \frac{1}{2}\left(\frac{d\phi}{dx}\right)^2 + \mathcal{O}\left(\tau^2\right). \tag{2.27}$$

Throughout the rest of the calculations in this chapter, we will use these modified expressions. Notice that for this action no terms in $\sum_M$ are exponentially suppressed anymore and the sum diverges. As it does not affect the redefined system, we will drop it. For later calculations, we would also like to derive an expression for the partition function. It can easily be obtained from Equation 2.22 by associating ϕ_0 with ϕ_D and

integrating over the free parameter:

$$\mathscr{Z}(\theta, D, \beta) = tr\left\{\mathcal{T}(\theta,\beta)^D\right\} = \int_0^{2\pi} d\phi_0 \prod_{j=1}^{D-1} \int \mathcal{D}\phi_j \exp\left[i\theta Q_{disc}(\boldsymbol{\phi}) - S_{disc}(\boldsymbol{\phi},\beta)\right] \tag{2.28}$$

Notice also that by redefining, we effectively turned Q_{disc} and S_{disc} into 2π periodic functions. The integrations over the real axis in $\mathcal{D}\phi_j$ therefore result in a divergence. We thus restrict the integrations to the circle. Furthermore, constants in the measure are dropped, as they are irrelevant for later calculations. We arrive at the partition function

$$\begin{aligned}\mathscr{Z}(\theta, D, \beta) &= \prod_{j=1}^{D} \int_{-\pi}^{\pi} d\phi_j \exp\left[i\theta Q_{disc}(\boldsymbol{\phi}) - S_{disc}(\boldsymbol{\phi},\beta)\right] \\ &= \int \mathcal{D}\phi \ \exp\left[i\theta Q_{disc}(\boldsymbol{\phi}) - S_{disc}(\boldsymbol{\phi},\beta)\right]\end{aligned} \tag{2.29}$$

The measure

$$\mathcal{D}\phi = \prod_{j=1}^{D} \int_{-\pi}^{\pi} d\phi_j \tag{2.30}$$

was defined, and we reintroduced the periodic boundary conditions on the lattice as defined in Equation 2.3.

2.2 Discretized topological Susceptibility of the Quantum Rotor

To validate the numerical simulations, we would like to obtain quantities from the system that can be compared with exact results. The expectation value of the topological charge can not be used, since it vanishes identically. While this will also be proven in this section, it intuitively makes sense as no direction of movement around the circle is favored. As an observable for a less trivial comparison, one can however look at the topological susceptibility:

$$\chi_t = \frac{1}{T}\left(\langle Q^2\rangle - \langle Q\rangle^2\right) = \frac{1}{T}\langle Q^2\rangle \tag{2.31}$$

We are able to give a closed-form formula for the topological susceptibility, which we can easily evaluate numerically.

From the definition of $\mathscr{Z}(\theta, D, \beta)$ in Equation 2.29, it follows that:

$$\begin{aligned}\chi_t &= \frac{1}{T}\langle Q^2\rangle = -\frac{1}{T}\partial_\theta^2 \log \mathscr{Z}(\theta, D, \beta)\Big|_{\theta=0} \\ &= \frac{1}{T}\left(\frac{\left(\partial_\theta Z(\theta, D, \beta)\right)^2}{\mathscr{Z}(\theta, D, \beta)^2}\Big|_{\theta=0} - \frac{\partial_\theta^2 \mathscr{Z}(\theta, D, \beta)}{\mathscr{Z}(\theta, D, \beta)}\Big|_{\theta=0}\right) \\ &= -\frac{1}{T}\frac{\partial_\theta^2 \mathscr{Z}(\theta, D, \beta)}{\mathscr{Z}(\theta, D, \beta)}\Big|_{\theta=0}.\end{aligned} \tag{2.32}$$

The last equality is valid, because

$$\partial_\theta \mathscr{Z}\Big|_{\theta=0} = \langle Q\rangle = 0. \tag{2.33}$$

There are several steps to take from this point, to concretize Equation 2.32. When the standard action (see Equation 2.26) and geometric topological charge (see Equation 2.24) are considered, a transfer matrix element takes the form

$$\begin{aligned}\langle\phi_j|\,\hat{\mathcal{T}}_S(\theta,\beta)\,|\phi_{j-1}\rangle &= \sqrt{\frac{\beta}{2\pi}}\exp\left[-\beta(1-\cos(\phi_j-\phi_{j-1})) + \frac{i\theta}{2\pi}\bmod(\phi_j-\phi_{j-1})_{2\pi}\right] \\ &= \sum_{k_j} e^{ik_j(\Delta\phi_j)}\sqrt{\frac{\beta}{2\pi}}e^{-\beta} \\ &\times \int_{-\pi}^{\pi}\frac{d(\Delta\phi_j)}{2\pi}\,\exp\left[\beta\cos(\Delta\phi_j) + i(\Delta\phi_j)\left(\frac{\theta}{2\pi}-k_j\right)\right].\end{aligned} \tag{2.34}$$

We have in the last step expanded the expression with a Fourier series and defined $\Delta\phi_j = \phi_j - \phi_{j-1}$. Here we can extract the momentum space eigenvalues of the transition matrix. This can be made apparent by applying the completeness relation $\sum_k |k\rangle\langle k| = \mathbb{I}$:

$$\begin{aligned}\langle\phi_j|\,\hat{\mathcal{T}}_S(\theta,\beta)\,|\phi_{j-1}\rangle &= \langle\phi_j|\,e^{-\tau\hat{H}_S(\theta,\beta)}\,|\phi_{j-1}\rangle = \sum_{k_j}\langle\phi_j|k_j\rangle\langle k_j|\phi_{j-1}\rangle\, e^{-\tau E_{k_j}(\theta,\beta)} \\ &= \sum_{k_j}\frac{e^{ik_j(\Delta\phi_j)}}{2\pi}e^{-\tau E_{k_j}(\theta,\beta)}.\end{aligned} \tag{2.35}$$

When the Equations 2.34 and 2.35 are combined, we find

$$e^{-\tau E_{k_j}(\theta,\beta)} = \sqrt{2\pi\beta}\frac{e^{-\beta}}{2\pi}\int_{-\pi}^{\pi} d(\Delta\phi_j)\,\exp\left[\beta\cos(\Delta\phi_j) + i(\Delta\phi_j)\left(\frac{\theta}{2\pi}-k_j\right)\right]. \tag{2.36}$$

Like in [12], we define the functions

$$\mathcal{I}_\nu(x) = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{i\nu\phi}e^{x\cos\phi}d\phi, \tag{2.37}$$

which for integer values reduce to modified Bessel functions of the first kind. With this definition, we obtain the identity

$$e^{-\tau E_{k_j}(\theta,\beta)} = \sqrt{2\pi\beta}e^{-\beta}\mathcal{I}_{\frac{\theta}{2\pi}-k_j}(\beta) \tag{2.38}$$

for the eigenvalues of the transfer matrix operator momentum space eigenstates. The partition function can be expressed in terms of these eigenvalues:

$$\mathscr{Z}(\theta,D,\beta) = tr\left\{\hat{\mathcal{T}}_S(\theta,\beta)^D\right\} = tr\left\{e^{-T\hat{H}_S(\theta,\beta)}\right\} = \sum_k e^{-TE_k(\theta,\beta)} = \sum_k \left(e^{-\tau E_k(\theta,\beta)}\right)^D. \tag{2.39}$$

Using this definition, we combine Equation 2.38 and Equation 2.32:

$$\begin{aligned}\chi_t = \frac{1}{T}\frac{\partial_\theta^2\mathscr{Z}(\theta,D,\beta)}{\mathscr{Z}(\theta,D,\beta)}\bigg|_{\theta=0} &= -\frac{D}{T}\sum_{k\in\mathbb{Z}}(D-1)\frac{e^{-TE_k(\theta,\beta)}}{\mathscr{Z}(\theta,D,\beta)}\left[\frac{\partial_\theta e^{-\tau E_k(\theta,\beta)}}{e^{-\tau E_k(\theta,\beta)}}\right]^2\bigg|_{\theta=0} \\ &+ \frac{e^{-TE_k(\theta,\beta)}}{\mathscr{Z}(\theta,D,\beta)}\left[\frac{\partial_\theta^2 e^{-\tau E_k(\theta,\beta)}}{e^{-\tau E_k(\theta,\beta)}}\right]\bigg|_{\theta=0}.\end{aligned} \tag{2.40}$$

It turns out to be convenient to define

$$w_k(\theta,D,\beta) = \frac{\mathcal{I}^D_{\frac{\theta}{2\pi}-k}(\beta)}{\mathscr{Z}(\theta,D,\beta)}, \tag{2.41}$$

such that Equation 2.40 is slightly altered:

$$\chi_t = \frac{1}{T}\frac{\partial_\theta^2\mathscr{Z}(\theta,D,\beta)}{\mathscr{Z}(\theta,D,\beta)}\bigg|_{\theta=0} = -\frac{D}{T}\sum_{k\in\mathbb{Z}}(D-1)w_k(0,D,\beta)\left[\frac{\mathcal{I}'_k(\beta)}{\mathcal{I}_k(\beta)}\right]^2 + w_k(0,D,\beta)\frac{\mathcal{I}''_k(\beta)}{\mathcal{I}_k(\beta)}. \tag{2.42}$$

It was used that the prefactors $\sqrt{2\pi\beta}e^{-\beta}$ cancel. The remaining derivatives are defined as

$$\begin{aligned}\mathcal{I}'_n(x) &:= \frac{1}{2\pi}\frac{\partial}{\partial\nu}\mathcal{I}_\nu(x)|_{\nu=n} = \frac{i}{2\pi}\int_{-\pi}^{\pi}\frac{\phi}{2\pi}e^{in\phi}e^{x\cos\phi}d\phi, \\ \mathcal{I}''_n(x) &:= \frac{1}{(2\pi)^2}\frac{\partial^2}{\partial\nu^2}\mathcal{I}_\nu(x)|_{\nu=n} = -\frac{1}{2\pi}\int_{-\pi}^{\pi}\left(\frac{\phi}{2\pi}\right)^2 e^{in\phi}e^{x\cos\phi}d\phi.\end{aligned} \tag{2.43}$$

Expression 2.42 can be easily computed numerically, as the main contribution in the sum is covered by only a few terms with small $|k|$. This is proven in Appendix A.

Finally let's prove $\langle Q \rangle = 0$: From Equation 2.38 and Equation 2.43 one can easily check that

$$\begin{aligned} e^{-\tau E_k(\theta,\beta)} &= e^{-\tau E_{-k}(-\theta)}, \\ \partial_\theta e^{-\tau E_k(\theta,\beta)}|_{\theta=0} &= -\partial_\theta e^{-\tau E_{-k}(\theta,\beta)}|_{\theta=0}. \end{aligned} \tag{2.44}$$

If we expand Equation 2.33:

$$\langle Q \rangle = \partial_\theta \sum_{k\in\mathbb{Z}} \left(e^{-\tau E_k(\theta,\beta)}\right)^D = \sum_{k\in\mathbb{Z}} D e^{-\tau E_k(\theta,\beta)(D-1)} \partial_\theta e^{-\tau E_k(\theta,\beta)} \Big|_{\theta=0}, \tag{2.45}$$

we see that the positive momenta terms will cancel the negative momenta terms, as expected, and the zeroth term vanishes identically.

Chapter 3

Estimating Observables

We have discretized the Rotor and identified the topological susceptibility χ_t as a fitting observable to evaluate numerically. Within the path integral formalism, the expectation value of an observable $f(\boldsymbol{\phi}) \in \mathcal{L}^1(\Omega)$ on the space of the lattice variables Ω is evaluated with

$$\begin{aligned} \langle f \rangle &= \int \mathcal{D}\phi \; f(\boldsymbol{\phi}) \frac{\exp[-S_{disc}(\boldsymbol{\phi}, \beta)]}{\mathscr{Z}(D, \beta)} \\ &= \int \mathcal{D}\phi \; f(\boldsymbol{\phi}) p(\boldsymbol{\phi}, D, \beta) \\ &\equiv \int \mathcal{D}\phi \; f(\boldsymbol{\phi}) p(\boldsymbol{\phi}). \end{aligned} \tag{3.1}$$

Here as well as in the sequel, the vacuum angle θ is set to zero. Further, note that for ease of notation, the dependencies of the expectation value are dropped and $\mathscr{Z}(D, \beta) \equiv \mathscr{Z}(0, D, \beta)$.

With the measure from Equation 2.30, this is equivalent to the expectation value

$$\mathbb{E}_\Phi[f(\Phi)], \tag{3.2}$$

with respect to the random variable $\Phi \in \Omega$. Its D-dimensional distribution $\mathbb{P}^X$ is defined by the density p. Therefore, throughout the rest of this thesis, we define as equivalent:

$$\mathbb{E}_\Phi[f(\Phi)] \equiv \mathbb{E}_{\Phi \sim p}[f(\Phi)]. \tag{3.3}$$

There are many strategies to numerically estimate this expectation value. In the following sections, we will introduce two approaches that rely on *Sampling*. For a more complete review, the reader is referred to [13, 14].

3.1 Integral Estimation through Sampling

Let us consider a sequence of independent identically distributed (*iid*) random variables $(\Phi_k)_{k \in \mathbb{N}}$ on Ω, which are drawn from p. We introduce the random variables $X_k = f(\Phi_k)$

with $X_k \in \mathbb{R}$. Let us define the sample average estimator

$$\overline{X}(N) = \frac{1}{N}\sum_{k=1}^{N} X_k, \tag{3.4}$$

which can be used to estimate the expectation value. As $(\Phi_k)_{k\in\mathbb{N}}$ are *iid*, so are $(X_k)_{k\in\mathbb{N}}$, because the function f is bounded on Ω. For the same reason, we can say X_k are in $\mathcal{L}^2(\mathbb{R})$ and $\sup_{k\in\mathbb{N}} \mathrm{Var}[X_k] < \infty$ and thus the strong law of large numbers applies: Given a random variable X that is identically distributed to X_k for all k

$$\mathbb{P}\left(\lim_{N\to\infty}\left|\frac{1}{N}\sum_{i=1}^{N} X_k - \mathbb{E}[X]\right| = 0\right) = 1. \tag{3.5}$$

So the correct integral is recovered with probability one. In real applications, the limit $N \to \infty$ can not be calculated, but a finite amount of samples is taken instead. If for a finite set of samples from p, Equation 3.4 is used to estimate the integral, an error is consequently introduced, which can be quantified using the central limit theorem. Prerequisites are that $(X_k)_{k\in\mathbb{N}}$ are *iid* and in $\mathcal{L}^2(\mathbb{R})$. The deviation between the estimate and the exact value of the expectation value

$$S_N^* = \overline{X}(N) - \mathbb{E}[X], \tag{3.6}$$

will be approximately distributed according to

$$\mathbb{P}^{S_N^*} \approx \mathcal{N}\left(0, \mathrm{Var}\left[X\right]/N\right) \tag{3.7}$$

for N large enough. $\mathcal{N}(\mu, \sigma^2)$ denotes the normal distribution with mean μ and standard deviation σ. The larger $\mathrm{Var}\left[X\right]$, the larger the expected error on the integral approximation and the more field configurations should be taken into account:

$$\sigma_{\overline{X}(N)} \sim \mathcal{N}\left(0, \mathrm{Var}\left[X\right]/N\right). \tag{3.8}$$

Of course, $\mathrm{Var}[X]$ needs to be estimated as well. For this, usually

$$\overline{\sigma}_X(N) = \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}\left(X_k - \overline{X}^{(N)}\right)^2} \tag{3.9}$$

is used, because it is unbiased if $(X_k)_{k\in\mathbb{N}}$ are *iid.*, such that

$$\mathbb{E}\left[\overline{\sigma}_X(N)\right] = \sqrt{\mathrm{Var}\left[X\right]} = \sigma_X \tag{3.10}$$

for all N. Finally the estimate for the error for *iid* random variables is therefore

$$\overline{\sigma}_{\overline{X}} = \frac{\overline{\sigma}_X(N)}{\sqrt{N}} \tag{3.11}$$

In the sequel, if the sample average estimator and its error are evaluated and the number of samples is clear from the context or not important for the calculations, $\overline{X}(N)$ will just be denoted by $\overline{X}$ and the error estimate as $\overline{\sigma}_{\overline{X}}$.

From many distributions of interest, we cannot directly sample. In the case of a physical system, this is often due to an intractable normalization $\mathscr{Z}$. Nevertheless, in such cases, there are ways to estimate 3.1.

3.2 Importance Sampling (IS)

Let's distinguish between the target density p, which is possibly normalized with an untractable partition function $\mathscr{Z}$, and it's unnormalized nominator $\tilde{p}$, such that

$$p = \frac{\exp[-S]}{\mathscr{Z}} = \frac{\tilde{p}}{\mathscr{Z}}. \tag{3.12}$$

Through an approach called *Importance Sampling*, we can calculate an expectation value with respect to the density p, over a surrogate density q, which satisfies $\text{supp}(p) \subseteq \text{supp}(q)$, instead. To see this, we rewrite the expectation value

$$\mathbb{E}_{\Phi\sim p}[f(\Phi)] = \mathbb{E}_{\Phi\sim q}\left[\frac{p(\Phi)}{q(\Phi)}f(\Phi)\right] = \mathbb{E}_{\Phi\sim q}\left[w(\Phi)f(\Phi)\right]. \tag{3.13}$$

We have defined the *importance weight* $w(\Phi) = \frac{p(\Phi)}{q(\Phi)}$. In order to evaluate w, we need p, which we might only be able to evaluate up to a multiplicative constant. So let's define the unnormalized importance weights

$$\tilde{w}(\Phi) = \frac{\tilde{p}(\Phi)}{q(\Phi)}. \tag{3.14}$$

It is possible to construct an expression for the partition function, which can in practice be estimated using the importance unnormalized importance weights themselves:

$$\begin{aligned}\mathbb{E}_{\Phi\sim q}\left[\frac{\tilde{p}(\Phi)}{q(\Phi)}\right] &= \mathbb{E}_{\Phi\sim q}\left[\frac{\mathscr{Z}p(\Phi)}{q(\Phi)}\right] = \mathscr{Z}\mathbb{E}_{\Phi\sim q}\left[\frac{p(\Phi)}{q(\Phi)}\right] \\ &= \mathscr{Z}\int d\boldsymbol{\phi}\; q(\boldsymbol{\phi})\frac{p(\boldsymbol{\phi})}{q(\boldsymbol{\phi})} = \mathscr{Z}\int d\boldsymbol{\phi}\; p(\boldsymbol{\phi}) = \mathscr{Z},\end{aligned} \tag{3.15}$$

such that the importance weights can be expressed as

$$w(\Phi) = \frac{\tilde{p}(\Phi)}{q(\Phi)\mathbb{E}_{\Phi\sim q}\left[\frac{\tilde{p}(\Phi)}{q(\Phi)}\right]}. \tag{3.16}$$

When the expectation value ought to be obtained numerically, we make use of estimators defined in Section 3.1. We proved that the error on the estimators depends on the

variance of the observable. The variance and thereby the error of the estimate changes when the density, with respect to which the variance is calculated changes. We will now give an approximation of the change of variance for importance sampling. It can be proven (see [13, 15]) that

$$\begin{aligned}\frac{\mathrm{Var}_{\Phi\sim p}\left[\overline{f(\Phi)}\right]}{\mathrm{Var}_{\Phi\sim q}\left[\overline{w(\Phi)f(\Phi)}\right]} &\approx \frac{1}{1+\mathrm{Var}_{\Phi\sim q}[w(\Phi)]} \\ &= \frac{1}{1+\mathbb{E}_{\Phi\sim q}\left[w^2(\Phi)\right]-\mathbb{E}^2_{\Phi\sim q}[w(\Phi)]} \\ &= \frac{1}{\mathbb{E}_{\Phi\sim q}[w^2(\Phi)]} \\ &= \frac{\mathbb{E}^2_{\Phi\sim q}[\tilde{w}(\Phi)]}{\mathbb{E}_{\Phi\sim q}[\tilde{w}^2(\Phi)]} \\ &=: \mathrm{ESS}_q\end{aligned} \tag{3.17}$$

This ratio is called the *Effective Sample Size* (ESS) estimated with q. When the expectation value is estimated, the accuracy of the estimate increases with the number of samples (see Equation 3.8). If importance sampling is used, the variance of the observable changes, which corresponds to an altered effective number of samples using the original density. In that sense, the *essq* can be used as an indicator of the deviation of q from p. The ESS_q is mainly useful because it does not depend on the observable function f itself. The sacrifice for this is that for a surrogate density q, which is not sufficiently close to p, this approximation can be very unreliable [13, 15].

An additional complication is introduced, when the ESS_q is merely estimated with a finite set of samples drawn from q. The deviation of q from p is then mostly checked for regions with high probability mass for q. And so to get an accurate measure, one would need to sample an increasingly large number of samples, if the approximation is off in the first place. If there is the possibility to sample from p itself, one can circumvent this and rewrite the effective sampling size using expectation values with respect to p. For this, we need yet another way to express the partition function

$$\mathbb{E}_{\Phi\sim p}\left[\tilde{w}^{-1}(\Phi)\right] = \int d\boldsymbol{\phi}\; p(\boldsymbol{\phi})\frac{q(\boldsymbol{\phi})}{\tilde{p}(\boldsymbol{\phi})} = \int d\boldsymbol{\phi}\; p(\boldsymbol{\phi})\frac{q(\boldsymbol{\phi})}{\mathscr{Z}p(\boldsymbol{\phi})} = \mathscr{Z}^{-1}. \tag{3.18}$$

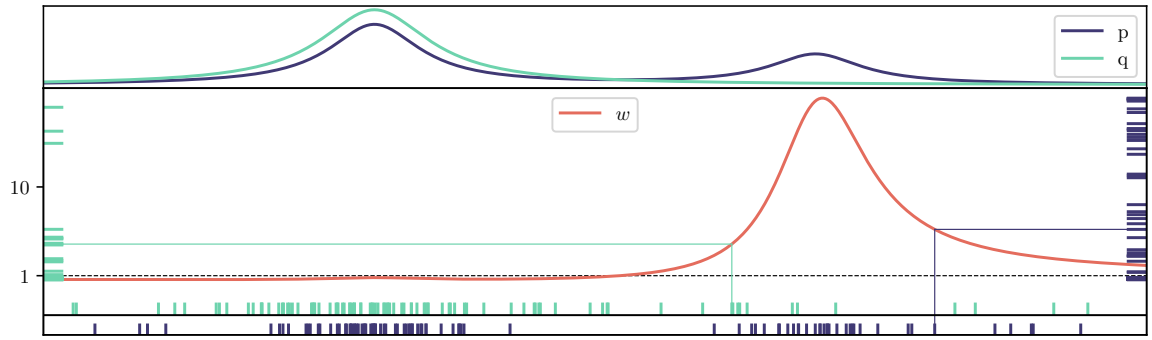


Figure 3.1: The upper plot shows a bimodal target density p and a surrogate density q, which are both properly normalized. The left mode of the target density is approximately modeled by the surrogate, whereas the right mode is completely neglected. The lower plot contains the resulting importance weights $w = \frac{p}{q}$ and the rugs on the bottom of the figure depict samples drawn from either p or q.

And with this

$$\begin{aligned}\frac{1}{\mathbb{E}_{\Phi\sim q}\left[w^2(\Phi)\right]} &= \left(\int d\boldsymbol{\phi}\; q(\boldsymbol{\phi})\frac{p^2(\boldsymbol{\phi})}{q^2(\boldsymbol{\phi})}\right)^{-1} \\ &= \left(\int d\boldsymbol{\phi}\; \frac{p(\boldsymbol{\phi})}{\mathscr{Z}}\frac{\tilde{p}(\boldsymbol{\phi})}{q(\boldsymbol{\phi})}\right)^{-1} = \frac{\mathscr{Z}}{\mathbb{E}_p[\tilde{w}]} \\ &= \frac{1}{\mathbb{E}_{\Phi\sim p}[\tilde{w}(\Phi)]\mathbb{E}_{\Phi\sim p}[\tilde{w}^{-1}(\Phi)]} \\ &=: \mathrm{ESS}_p\end{aligned} \tag{3.19}$$

For the sake of comparison, the mechanics of the ESS_q and ESS_p are visualized in Figure 3.1. For a surrogate density that misses areas of probability mass in the target density, a large error will be attached to the estimate of the ESS_q. This is because an area with large importance weight and thus large contribution to the *essq* is hardly sampled.

Ultimately, if there is a surrogate density sufficiently close to a target density p, which we can easily sample from, then importance sampling can be used to correct the bias in the samples and estimate an expectation value with respect to p.

3.3 Monte Carlo Markov Chains (MCMC)

In contrast to importance sampling, for *Monte Carlo Markov Chain (MCMC)* methods, we create a set of samples iteratively in such a way that asymptotically the sampled configurations follow a given target density p. Crucially, again we utilize algorithms that allow us to work with the unnormalized nominator $\tilde{p}$.

In this section, we first introduce the general framework of MCMC methods, before specific algorithmic variations are discussed in further detail. The notation and explanations are largely borrowed from [16], where an extensive introduction can be found. Furthermore [13, 14] have been insightful.

3.3.1 Markov Chains

The basis for MCMC methods is the concept of the *Markov Chain*. Given a target density p on Ω, it samples configurations from Ω, which are asymptotically distributed with respect to p. By design, configurations in a Markov Chain are produced consecutively, dependent only on their respective predecessor. Due to the sequential creation, a parallelized construction of a single Markov Chain is inherently impossible.

A Markov Process (Markov Chain) is characterized by a transition function $A(\boldsymbol{\phi}', \boldsymbol{\phi})$, which defines the probability to move from the configuration $\boldsymbol{\phi}$ to $\boldsymbol{\phi}'$. As such we require A to be non-negative and normalized:

$$0 \leq A(\boldsymbol{\phi}'|\boldsymbol{\phi}) \leq 1 \qquad \text{and} \qquad \int d\boldsymbol{\phi}' \; A(\boldsymbol{\phi}'|\boldsymbol{\phi}) = 1. \tag{3.20}$$

Note that for the normalization, the start configuration ϕ is included in the integral. So it is possible to repeat a configuration in a Markov Process. If the desired target density p ought to be reached, the transition function must leave it invariant

$$\int_\Omega d\boldsymbol{\phi}' A(\boldsymbol{\phi}, \boldsymbol{\phi}') p(\boldsymbol{\phi}') = p(\boldsymbol{\phi}), \tag{3.21}$$

which ensures that p is a fixed point of the process.

Since A is normalized, Equation 3.21 can be rewritten as

$$\int_\Omega d\boldsymbol{\phi}' A(\boldsymbol{\phi}, \boldsymbol{\phi}') p(\boldsymbol{\phi}') = \int_\Omega d\boldsymbol{\phi}' A(\boldsymbol{\phi}', \boldsymbol{\phi}) p(\boldsymbol{\phi}), \tag{3.22}$$

which is called the *balance equation*. This ensures that there are no probability sources or sinks and an equilibrium is reached. Eqs. 3.21 and 3.22 are hard to verify for a given transition kernel and target density. A condition that is easier to check, which is sufficient for Equation 3.22 is the *detailed balance equation*:

$$A(\boldsymbol{\phi}, \boldsymbol{\phi}') p(\boldsymbol{\phi}') = A(\boldsymbol{\phi}', \boldsymbol{\phi}) p(\boldsymbol{\phi}). \tag{3.23}$$

If there is a finite amount of steps n, such that the n step transition function

$$A^{(n)}(\boldsymbol{\phi}, \boldsymbol{\phi}') = \int_\Omega d\boldsymbol{\phi}_n \cdots \int_\Omega d\boldsymbol{\phi}_1 \; A(\boldsymbol{\phi}, \boldsymbol{\phi}_n) \cdots A(\boldsymbol{\phi}_1, \boldsymbol{\phi}') > 0, \tag{3.24}$$

for any start configuration ϕ' and any final configuration ϕ, the Markov Process is called *irreducible*. If additionally the chain is aperiodic and has an invariant density p

as defined in Equation 3.21, the process will become stationary at p [13].

We are free to initialize the Markov Chain with an arbitrary configuration $\boldsymbol{\phi}_0$. If drawn at random, one speaks of a *hot start*, whereas the start with a fixed vector is called a *cold start*. Since $\boldsymbol{\phi}_0$ is not drawn from p, we effectively create subsequent samples $\boldsymbol{\phi}^{(i)}$ at chain index $i \in \mathbb{N}$ from a distribution $p^{(i)}$, for which $\lim_{i\to\infty} p^{(i)} = p$ [17]. Under the assumption that after N_{burnin} steps, the error due to this deviation in distribution becomes negligible in comparison with the sampling error, this first part of the Markov Chain is discarded, and the rest is treated as samples from the correct distribution.

3.3.2 Autocorrelation

Ultimately the chain is utilized as a sampler in the course of estimating an expectation value as described in Section 3.1. Successive samples in a Markov Chain will generally be correlated and thus the error estimator 3.9, which was introduced for independent random variables, needs to be revisited. Let the Markov Chain be a sequence of configurations $(\Phi_k)_{k\in\mathbb{N}}$, then for observables $X_k = f(\Phi_k)$ we define for an offset $t \in \mathbb{Z}$ the *autocorrelation*

$$C_X(X_k, X_{k+t}) = \mathbb{E}[(X_k - \mathbb{E}[X_k])(X_{k+t} - \mathbb{E}[X_{k+t}])] = \mathbb{E}[X_k X_{k+t}] - \mathbb{E}[X_k]\mathbb{E}[X_{k+t}]. \quad (3.25)$$

If k is large enough and equilibrium has been reached in the Markov Process, the autocorrelation function only depends on the difference t. So

$$C_X(X_k, X_{k+t}) \equiv C_X(t), \quad (3.26)$$

which reduces to σ_X^2 for $t = 0$. In the following, we will use the normalized autocorrelation function $\Gamma_X(t) = \frac{C_X(t)}{C_X(0)}$,

The error $\sigma_{\overline{X}}$ we expect for the estimated expectation value $\overline{X}$ as defined in Equation 3.4 increases due to the correlation of subsequently sampled configurations compared to the independent case:

$$\sigma_{\overline{X}}^2 \approx 2\frac{\sigma_X^2}{N}\left(\frac{1}{2} + \sum_{t=1}^{N} \Gamma_X(t)\right) = 2\frac{\sigma_X^2}{N}\tau_{int}^X. \quad (3.27)$$

The error σ_X on X can be estimated by $\overline{\sigma}_X$as defined in Equation 3.9. We define the estimated error for the expectation value of the observable X using the Markov Chain (MC) as:

$$\sigma_{\overline{X}} \approx \overline{\sigma}_X\sqrt{\frac{2\tau_{int}^X}{N}} =: \overline{\sigma}_{\overline{X}}^{(MC)}. \quad (3.28)$$

For an explicit derivation, see [16]. τ_{int}^X is called the *integrated autocorrelation time* for the observable X. Its name originates from the observation that in many cases $\Gamma_X(t)$

decreases exponentially and

$$\tau_{exp}^X = \int_0^\infty dt\; e^{-t/\tau_{exp}^X} \approx \frac{1}{2} + \sum_{t=1}^{N} e^{-t/\tau_{exp}^X} = \tau_{int}^X. \tag{3.29}$$

Numerically for the experiments in this thesis, the estimation of the integrated autocorrelation time and thereby the error to the expectation value estimate was performed using the methods proposed in [18, 19].

The autocorrelation can be viewed as an indication of the needed chain length when a certain error magnitude is desired. [16] propose that the chain length should be at least $1000\tau_{int}^X$ and more than $10000\tau_{int}^X$ for an error less than 1%. The number of configurations $N_{brurnin}$ dropped at the beginning of the chain should be around $20\tau_{int}^X$. These estimates are very broad and these relationships should be studied regarding the used observable.

3.3.3 Metropolis-Hastings

The *Metropolis-Hastings* algorithm was originally introduced in [20] for any target density of the form

$$p(\boldsymbol{\phi}) = \mathscr{Z}^{-1} \exp(-S(\boldsymbol{\phi})), \tag{3.30}$$

where S is an arbitrary function on Ω. It provides a construction of the transition function that satisfies the *detailed balance* condition. Given a current configuration $\boldsymbol{\phi}$ in the Markov Chain, we choose a new proposal configuration $\boldsymbol{\phi}'$ according to a *selection function* $T(\boldsymbol{\phi}', \boldsymbol{\phi})$. The proposed sample is accepted and taken as the next configuration in the Markov Chain with the probability

$$T_{MH}(\boldsymbol{\phi}', \boldsymbol{\phi}) = \min\left(1, \frac{T(\boldsymbol{\phi}, \boldsymbol{\phi}')p(\boldsymbol{\phi}')}{T(\boldsymbol{\phi}', \boldsymbol{\phi})p(\boldsymbol{\phi})}\right). \tag{3.31}$$

If the proposal is rejected the current configuration is repeated as the next step in the chain. Note that the partition function $\mathscr{Z}$ cancels and the evaluation can be done with $\tilde{p}$ instead.

The only constraints on the selection function T is that $T(\boldsymbol{\phi}', \boldsymbol{\phi}) > 0$ if and only if $T(\boldsymbol{\phi}, \boldsymbol{\phi}') > 0$ [13] and that irreducibility of the chain is given. Its choice, however, heavily influences the acceptance probability and thereby the performance of the Markov Chain. A popular choice is to use a symmetric selection function, such that the factors $T(\boldsymbol{\phi}'|\boldsymbol{\phi})$ and $T(\boldsymbol{\phi}|\boldsymbol{\phi}')$ cancel each other. The detailed balance condition is fulfilled for the overall transition function $A = T_{MH}T$

$$A(\boldsymbol{\phi}', \boldsymbol{\phi})p(\boldsymbol{\phi}) = \min\left(T(\boldsymbol{\phi}', \boldsymbol{\phi})p(\boldsymbol{\phi}), T(\boldsymbol{\phi}, \boldsymbol{\phi}')p(\boldsymbol{\phi}')\right), \tag{3.32}$$

because it is symmetric in $\boldsymbol{\phi}'$and $\boldsymbol{\phi}$. For MCMC methods that involve a Metropolis step in the sense that proposed configurations are accepted with probability 3.31, we

define an acceptance rate a. If during the construction of a chain of length N, $N_{accepted}$ proposals have been accepted,

$$a = \frac{N_{accepted}}{N}. \tag{3.33}$$

3.3.4 Hybrid Monte Carlo Method (HMC)

When we design a transition function that contains a proposal and acceptance step, the proposal step is quite unconstrained. If we were to pick the next configuration randomly, for a target density, which is far from uniform, the acceptance rate would be dramatically low and thus the Markov Chain would not sample efficiently. A way to assert some control over the acceptance rate is to propose new configurations ϕ', for which the difference in action $\Delta S = S(\phi') - S(\phi)$ is small. In the *Hybrid* or *Hamiltonian Monte Carlo* method, we use this tactic to achieve a reasonably high acceptance rate, while the configurations are ideally only weakly correlated [16].

New proposals are created through molecular dynamics evolution [13, 14, 16]. Here we introduce a fictitious conjugate momentum $\boldsymbol{p}^{\phi}$to the variable ϕ and construct the *guide Hamiltonian*

$$H\left(\boldsymbol{\phi}, \boldsymbol{p}^{\phi}\right) = \frac{1}{2}\left(\boldsymbol{p}^{\phi}\right)^2 + S(\boldsymbol{\phi}). \tag{3.34}$$

The integral observation for this construction is that, if it is possible to sample

$$(\boldsymbol{\phi}, \boldsymbol{p}^{\phi}) \sim \mathscr{Z}^{-1} \exp[-H(\boldsymbol{\phi}, \boldsymbol{p}^{\phi})], \tag{3.35}$$

then marginally, we produce samples

$$\boldsymbol{\phi} \sim \mathscr{Z}^{-1} \exp[-S(\boldsymbol{\phi})]. \tag{3.36}$$

The evolution in time t of a state $(\boldsymbol{\phi}(t), \boldsymbol{p}^{\phi}(t))$, is described by the Hamiltonian equations

$$\begin{aligned} \dot{\boldsymbol{p}^{\phi}} &= -\frac{\partial H}{\partial \boldsymbol{\phi}} = -\frac{\partial S}{\partial \boldsymbol{\phi}} \\ \dot{\boldsymbol{\phi}} &= \frac{\partial H}{\partial \boldsymbol{p}^{\phi}} = \boldsymbol{p}^{\phi}. \end{aligned} \tag{3.37}$$

This evolution keeps the Hamiltonian 3.34 constant for any initial state if it is integrated exactly. Numerically it needs to be approximated. The choice of integration method is however constrained. It turns out that in order to achieve the detailed balance condition, one needs to require two properties. The proposal function T_{md} needs to be reversible:

$$T_{md}(\boldsymbol{p}^{\phi'}, \boldsymbol{\phi}' | \boldsymbol{p}^{\phi}, \boldsymbol{\phi}) = T_{md}(-\boldsymbol{p}^{\phi}, \boldsymbol{\phi} | -\boldsymbol{p}^{\phi'}, \boldsymbol{\phi}'), \tag{3.38}$$

and the evolution of the variables $(\boldsymbol{\phi}, \boldsymbol{p}^{\phi})$ is required to be volume preserving. This is equivalent to the condition

$$\det\left[\frac{\partial(\boldsymbol{\phi}_{i+1}, \boldsymbol{p}^{\phi}_{i+1})}{\partial(\boldsymbol{\phi}_i, \boldsymbol{p}^{\phi}_i)}\right] = 1, \tag{3.39}$$

for the determinant of the Jacobian for the evolution step.

Both properties above are fulfilled by the *leapfrog* integration method [16, 21]. For a small increment Δt in the evolution time, the state $(\boldsymbol{\phi}_i, \boldsymbol{p}^{\phi}{}_i)$ is updated with

$$\begin{aligned} \boldsymbol{p}^{\phi}_{i+1/2} &= \boldsymbol{p}^{\phi}_i - \left.\frac{\partial S}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}_i} \frac{\Delta t}{2} \\ \boldsymbol{\phi}_{i+1} &= \boldsymbol{\phi}_{i+1} + \boldsymbol{p}^{\phi}_{i+1/2}\Delta t \\ \boldsymbol{p}^{\phi}_{i+1} &= \boldsymbol{p}^{\phi}_{i+1/2} - \left.\frac{\partial S}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}_i} \frac{\Delta t}{2}. \end{aligned} \tag{3.40}$$

In order to produce a new proposal configuration, a random momentum is drawn from a standard normal distribution and N such leapfrog integration steps are chained together.

In a Metropolis fashion, the proposal is accepted or rejected according to the probability

$$T_{MH}(\boldsymbol{p}^{\phi'}, \boldsymbol{\phi}' | \boldsymbol{p}^{\phi}, \boldsymbol{\phi}) = \min\left(1, \frac{\exp(-H(\boldsymbol{p}^{\phi'}, \boldsymbol{\phi}'))}{\exp(-H(\boldsymbol{p}^{\phi}, \boldsymbol{\phi}))}\right) \tag{3.41}$$

to ensure that the resulting Markov Chain fulfills the *detailed balance* condition [16]. Irreducibility is generally dependent on the potential S used in the HMC [14, 22]. In the case of the Rotor, it is given, since S_{disc} is bounded from above on the compact state space $[0, 2\pi]^D$ and $S_{disc} \in C^2([0, 2\pi]^D)$.

For the Rotor, we can also explicitly calculate the force used in the molecular dynamics evolution:

$$\begin{aligned} \nabla_{\boldsymbol{\phi}} S_{disc}(\boldsymbol{\phi}) &= \beta \nabla_{\boldsymbol{\phi}} \sum_{j\tau \in \Lambda} (1 - \cos(\phi_j - \phi_{j-1})) \\ &= \beta \sum_{j\tau \in \Lambda} [\sin(\phi_j - \phi_{j-1})\boldsymbol{e}_j - \sin(\phi_j - \phi_{j-1})\boldsymbol{e}_{j-1}] \\ &= \beta \sum_{j\tau \in \Lambda} [\sin(\phi_j - \phi_{j-1}) - \sin(\phi_{j+1} - \phi_j)]\, \boldsymbol{e}_j. \end{aligned} \tag{3.42}$$

Here the last equality holds for periodic boundary conditions and the vectors $\boldsymbol{e}_j$ with $j \in \{1, \ldots, D\}$ span the canonical base of the product space.

The HMC method is dependent on local dynamics. If a target density is highly peaked, such as for systems with non-trivial topology, where topological sectors are separated

by large energy barriers, the autocorrelation times of topological observables $\mathcal{O}$ on the Markov Chain increase drastically. In the continuum limit this separation of topological sectors becomes ever larger and the autocorrelation times are hypothesized [9, 23] to diverge as a function of the lattice spacing τ:

$$\tau_{int}^{\mathcal{O}} = \alpha_{\mathcal{O}} \tau^{-z_{\mathcal{O}}}. \tag{3.43}$$

$z_{\mathcal{O}}, \alpha_{\mathcal{O}} \in \mathbb{R}$ are system and HMC parameter dependent scaling parameters. This phenomenon is called *critical slowing down* or *topological freezing* in the case of topological observables.

3.3.5 Wolff Cluster Algorithm

Critical slowing down makes the use of HMC algorithms quite inefficient towards the continuum limit. The *Wolff Cluster* algorithm [24], which is applicable to spin systems with local interactions, does not suffer from such efficiency degradation. Here sites on the lattice are combined to form clusters, which are collectively altered. We assume that the simulated system can be described by the partition function

$$\begin{aligned} \mathscr{Z}(\beta, n) &\propto \prod_{i\tau \in \Lambda} \int_{\mathbb{S}^{n-1}} d\bar{\phi}_i \exp\left\{ \beta \sum_{\langle ij \rangle} \bar{\phi}_i \bar{\phi}_j \right\} \\ &= \prod_{i\tau \in \Lambda} \int_{\mathbb{S}^{n-1}} d\bar{\phi}_i \exp\left\{ \beta \sum_{\langle ij \rangle} \cos(\sphericalangle(\bar{\phi}_i, \bar{\phi}_j)) \right\}. \end{aligned} \tag{3.44}$$

$\bar{\phi}_i$ and $\bar{\phi}_j$ denote vectors in $\mathbb{S}^{n-1}$,$\sphericalangle(\bar{\phi}_i, \bar{\phi}_j)$ the angle between the two vectors and $0 \in \mathbb{R}^n$, Λ the lattice and $\langle ij \rangle$ the set of next neighbors on the lattice. For $n = 2$, Equation 3.44 is proportional to the partition function in Equation 2.29 of the Rotor system. For an arbitrary unit vector $\bar{\alpha} \in \mathbb{S}^{n-1}$, we define a reflection $R(\bar{\alpha})$ about the hyperplane perpendicular to $\bar{\alpha}$

$$R(\bar{\alpha})\bar{\phi}_i = \bar{\phi}_i - 2(\bar{\phi}_i \bar{\alpha})\bar{\alpha}. \tag{3.45}$$

Notice that applying R twice yields the original vector:

$$R^2(\bar{\alpha}) = \mathbb{I}. \tag{3.46}$$

An update step for the Markov Chain, constructed with the Wolff algorithm, then consists of the following stages:

1. From a uniform distribution choose $\bar{\alpha} \sim U\left(\mathbb{S}^{n-1}\right)$
2. Choose a location on the lattice i and flip its spin $\bar{\phi}_i \leftarrow R(\bar{\alpha})\bar{\phi}_i$

3. Visit all direct neighbors j of i and with probability

$$\begin{aligned} P(\bar{\phi}_i, \bar{\phi}_j) &= 1 - \exp(\min\{0, \beta\bar{\phi}_i[1 - R(\bar{\alpha})\bar{\phi}_j]\}) \\ &= 1 - \exp(\min\{0, 2\beta(\bar{\alpha}\bar{\phi}_i)(\bar{\alpha}\bar{\phi}_j)\}) \end{aligned} \tag{3.47}$$

connect to cluster c and flip.

4. Repeat step 3 with $i \leftarrow j$ if it was connected, otherwise, this branch of the cluster formation is terminated.

As desired, the Markov Chain constructed in such a manner is irreducible. This holds, since transition probability from any configuration $\boldsymbol{\phi}$ to any other configuration $\boldsymbol{\phi}'$ is strictly positive: There is a non-zero probability that for all $i\tau$ $in\Lambda$ a cluster of size 1 is built and flipped with an $\bar{\alpha}$, such that $\bar{\phi}'_i = R(\bar{\alpha})\bar{\phi}_i$. So then after $|\Lambda|$ update steps $\boldsymbol{\phi}'$ is obtained. The transition function $A(\boldsymbol{\phi}', \boldsymbol{\phi})$ also fulfills the detailed balance condition. To get to the new configuration a cluster c is flipped. Since probabilities inside the cluster c cancel, only the factors at the edge ∂c are taken into account:

$$\begin{aligned} \frac{A(\boldsymbol{\phi}', \boldsymbol{\phi})}{A(\boldsymbol{\phi}, \boldsymbol{\phi}')} &= \prod_{\langle ij\rangle \in \partial c} \frac{1 - P(R(\alpha)\bar{\phi}_i, \bar{\phi}_j)}{1 - P(R(\alpha)\bar{\phi}'_i, \bar{\phi}'_j)} \\ &= \prod_{\langle ij\rangle \in \partial c} \exp\left\{\min\left(0, \beta R\bar{\phi}_i[1 - R]\bar{\phi}_j\right) - \min\left(0, \beta R\bar{\phi}'_i[1 - R]\bar{\phi}'_j\right)\right\} \\ &= \prod_{\langle ij\rangle \in \partial c} \exp\left\{\min\left(0, \beta\bar{\phi}_i[R - 1]\bar{\phi}_j\right) - \min\left(0, -\beta\bar{\phi}_i[R - 1]\bar{\phi}_j\right)\right\} \\ &= \prod_{\langle ij\rangle \in \partial c} \exp\left\{\min\left(0, \beta\bar{\phi}_i[R - 1]\bar{\phi}_j\right) + \max\left(0, \beta\bar{\phi}_i[R - 1]\bar{\phi}_j\right)\right\} \\ &= \prod_{\langle ij\rangle \in \partial c} \exp\left\{\beta\bar{\phi}_i[R - 1]\bar{\phi}_j\right\} \\ &= \exp\left\{\sum_{\langle ij\rangle \in \partial c} \bar{\phi}_i[R - 1]\bar{\phi}_j\right\} = \exp\left\{\sum_{\langle ij\rangle \in \partial c} \bar{\phi}'_i\bar{\phi}'_j - \bar{\phi}_i\bar{\phi}_j\right\} \\ &= \frac{p(\boldsymbol{\phi})}{p(\boldsymbol{\phi}')}. \end{aligned} \tag{3.48}$$

Especially for small lattice spacings τ, the algorithm exhibits almost negligible integrated autocorrelation times for the Rotor system and was used to efficiently sample configurations $\boldsymbol{\phi} \sim p$. Reaching the efficiency of the cluster algorithm would be a hard task. However, it is unclear how to generalize such a method to systems that do not exactly fit the aforementioned specifications.

3.3.6 Metropolized Independence Sampling

Finally, we discuss a variant of MCMC algorithms that like importance sampling relies on a surrogate density q. Again, we require q to be reasonably close to the actual target density p up to some scaling constant $\mathscr{Z}$. We should also be able to sample from it and evaluate it rather efficiently. Like any proposal method, a function that is actually independent of the previous configuration can be used to create new configurations that are accepted or rejected by a Metropolis step. And thus we remove any bias from the surrogate density and construct a Markov Chain, which contains samples that are distributed with respect to the actual target density p asymptotically [15]. Notice also that we can deviate from the common choice of a symmetric proposal function because we can explicitly evaluate it.

The steps for one configuration update are fairly simple. First one draws a proposal configuration $\boldsymbol{\phi}'$ from q. Then the proposal is accepted with the probability

$$T_{MH}\left(\boldsymbol{\phi}', \boldsymbol{\phi}\right) = \min\left(1, \frac{q\left(\boldsymbol{\phi}\right) p\left(\boldsymbol{\phi}'\right)}{p\left(\boldsymbol{\phi}\right) q\left(\boldsymbol{\phi}'\right)}.\right) \tag{3.49}$$

A Markov Chain constructed with this scheme is irreducible if the support of the surrogate density is equal to the state space Ω. A proof of detailed balance is also quite straightforward and again relies on the fact that the transition function

$$A(\boldsymbol{\phi}', \boldsymbol{\phi})p(\boldsymbol{\phi}) = T_{MH}\left(\boldsymbol{\phi}', \boldsymbol{\phi}\right) q(\boldsymbol{\phi}')p(\boldsymbol{\phi}) = \min\left(p\left(\boldsymbol{\phi}\right) q\left(\boldsymbol{\phi}'\right), q\left(\boldsymbol{\phi}\right) p\left(\boldsymbol{\phi}'\right)\right) \tag{3.50}$$

is symmetric.

As [23] point out, for independent update proposals, one can set a lower bound for the autocorrelation function of the chain for all observables $\Gamma(t)$ using the acceptance rate a:

$$\Gamma(t) \geq (1-a)^t. \tag{3.51}$$

Therefore, it is necessary to increase a in order to achieve low autocorrelations. However, an acceptance rate close to one does not necessarily translate to a small autocorrelation time [23].

Chapter 4

MCMC Results

For both the Hamiltonian Monte Carlo and the Wolff cluster algorithm, we can now quantify the capability to estimate the topological susceptibility for the Quantum Rotor. As the HMC algorithm is generally applicable, this is the benchmark for any Normalizing Flow-based approach we devise. The Wolff algorithm is analyzed because it can be used to produce samples drawn from the target density p very efficiently.

If a method M is used for the estimation of the expectation value of the observable Q, the result is written as $\overline{Q}^{(M)}$. To estimate the expectation value of the topological susceptibility, we thus require

$$\overline{\left(Q - \overline{Q}^{(M)}\right)^2}^{(M)} := \mathrm{Var}^{(M)}\left[Q\right], \tag{4.1}$$

which is a *secondary* or *derived* observable. Estimating the error on the expectation value result thus requires further treatment, which is outlined e.g. in [18]. On the other hand $\mathbb{E}[Q] = 0$, and so we can analyze $\overline{Q^2}^{(M)}$ as a primary observable instead.

The number of configurations that are required in a Markov Chain that is used to estimate $\overline{Q^2}$ value is crucially dependent on $\tau_{int}^{Q^2}$. As described in Section 3.3.4, for the HMC, it is hypothesized that it scales exponentially with the lattice spacing (3.43) if we approach the continuum limit on a constant line of physics (CLP) [23]. This requires keeping the time extent $T = \tau D$ constant and varying τ and D accordingly. We quantify the scaling with the exponent z_{Q^2}.

The integrated autocorrelation times $\tau_{int}^{Q^2}$ for several points along the CLP of $T = 4$ and $I = 0.25$ are plotted in 4.1c. Here, we also add the exponential fit, which results in $z_{Q^2} = 6.11 \pm 0.11$. This is slightly lower than in [17] but overall consistent. For all HMC runs in this thesis, the rate of accepted configurations was set equal to $a = 80 \pm 3\%$ and the step size was adapted for 20 trajectory steps through bisection search initially.

The scaling behaviors dependent on solely the coupling β or the system size D are plotted in Figure 4.1a and 4.1b. Varying only the dimension with a fixed coupling, from $D \approx 40$, we see a nearly constant autocorrelation $\tau_{int}^{Q^2}$. This is because the Quantum Rotor has a mass gap ($(E_1 - E_0) = \Delta E_1 > 0$) and therefore the Euclidean correlators are exponentially suppressed [25], such that at large enough lattice size

the effects of a finite lattice size are virtually gone. For smaller lattice sizes, we see a decreased $\tau_{int}^{Q^2}$. In contrast, when D is fixed and β is increased, we see exponential scaling similar to Figure 4.1a. The exponential fit is added here as well to quantify the scaling ($z_{Q^2} = 5.84 \pm 0.07$).

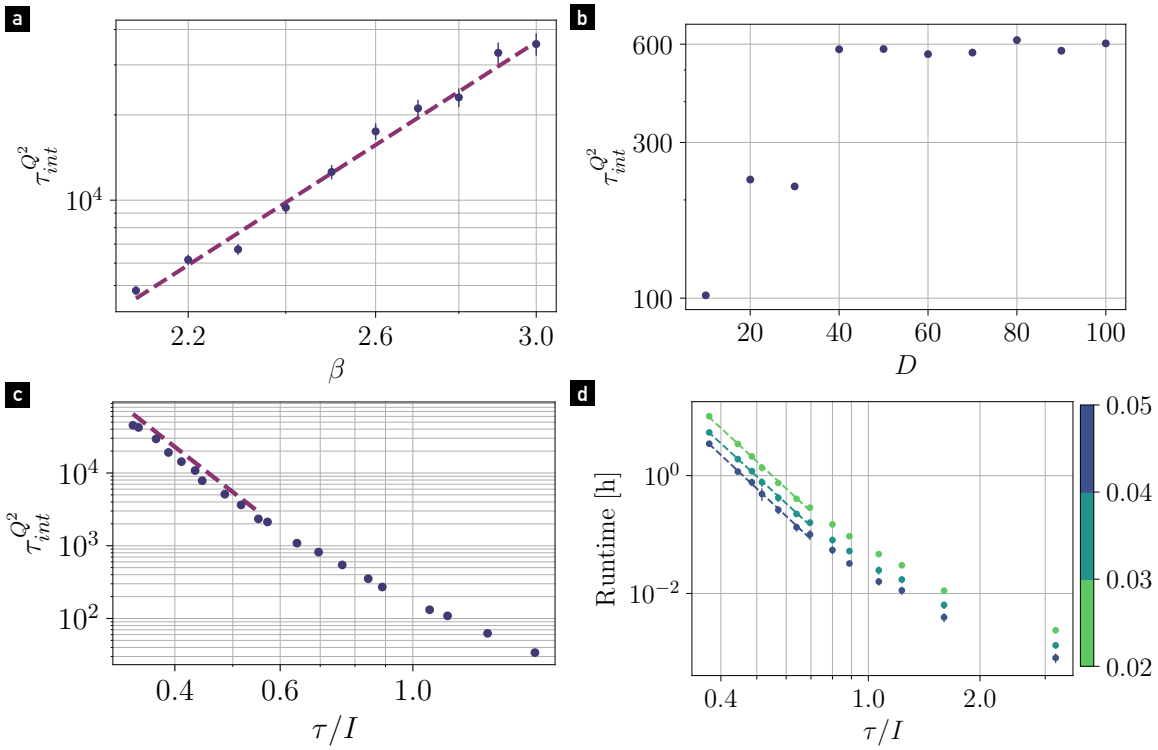


Figure 4.1: Integrated autocorrelation times for estimating Q^2 with the HMC. (a) $\tau_{int}^{Q^2}$ of $\overline{Q^2}^{(\mathrm{HMC})}$ Markov Chains for different coupling strengths β. D is fixed to 25. (b) $\tau_{int}^{Q^2}$ of $\overline{Q^2}^{(\mathrm{HMC})}$ Markov Chains for different lattice sizes D. β is fixed to 1.0. (c) $\tau_{int}^{Q^2}$ of $\overline{Q^2}^{(\mathrm{HMC})}$ Markov Chains with varying lattice spacing τ, while the volume $V = \tau D$ is fixed to 4.0. The moment of inertia $I = 0.25$. We slightly shift the fit upwards, such that it does not cover the data points. The Markov Chains are initialized drawing from a uniform distribution on the torus product space $\mathbb{T}^D$ and the number of burn-in steps was 10^6. We use 10 replicas for each lattice spacing and the analysis is done considering all replicas. (d) Runtime in hours, which is required to reach a certain relative error ε_{Q^2}. The error value is color-coded and can be extracted from the color bar. The time extent is fixed like in (c). In (a), (b), (c) the errors on $\tau_{int}^{Q^2}$ are included but very small. We additionally include exponential fits in (a), (c) and (d).

Intending to construct a metric, which is comparable to the Normalizing Flow method, we quantify the number of configurations needed to reach a desired relative error threshold on the topological susceptibility. So if the error $\overline{\sigma}^{(\mathrm{HMC})}_{\overline{Q^2}}$ and the mean $\overline{Q^2}^{(\mathrm{HMC})}$ are

given, for a relative error threshold ε_{Q^2}, we require:

$$\frac{\overline{\sigma}_{\overline{Q^2}}^{(\mathrm{HMC})}}{\overline{Q^2}^{(\mathrm{HMC})}} < \varepsilon_{Q^2}. \tag{4.2}$$

The HMC algorithm is an iterative method. And so we can quantify the total elapsed runtime C for N_{HMC} steps when the elapsed time Δt_{HMC} per step is known, as

$$C(N_{\mathrm{HMC}}, \Delta t_{\mathrm{HMC}}) = N_{\mathrm{HMC}} \Delta t_{\mathrm{HMC}}. \tag{4.3}$$

The hardware for the time measurements is listed in Appendix D.

For $\varepsilon_{Q^2} \in \{0.03, 0.04, 0.05\}$ the required runtimes are plotted in Figure 4.1d dependent on the lattice spacing. To be able to give an interval of error for the estimate of the number of configurations, 10 chains are run. Using the relation on the autocorrelation times, we only add every $\lfloor \tau_{int}^{Q^2}/2 \rfloor$ configuration to the Markov Chain during the HMC run. This facilitates error estimation drastically, and we do not discard more configurations than necessary, which would drive up the runtime artificially. The threshold is found by running the analysis on increasingly large consecutive subsets at the beginning of the Markov Chains until the relative error is smaller than required. The found runtimes are averaged over all analyzed chains, and we can assign an error by computing the standard deviation for independent samples. As the error is mostly governed by $\tau_{int}^{Q^2}$ (3.28), we see an exponential relationship between the runtime and the lattice spacing consistent with Figure 4.1c. The scaling coefficients are listed in Table 4.1. Note that for the runtimes, we do not include the number of burn-in steps.

ε_{Q^2}	z
0.05	5.90 ± 0.01
0.04	5.79 ± 0.01
0.03	5.84 ± 0.01

Table 4.1: Error dependent scaling exponents for the HMC.

The Wolff algorithm on the other hand performs drastically better and displays much less correlation between subsequently sampled configurations. Interestingly, for larger β, $\tau_{int}^{Q^2}$ actually becomes smaller. The dependence of $\tau_{int}^{Q^2}$ on the coupling β and the system size D are plotted in 4.2a and 4.2b. The comparatively small autocorrelation times allow for an effective generation of sets $\{x|x \sim p\}$. The specifications on the datasets can be found in Appendix C.3.

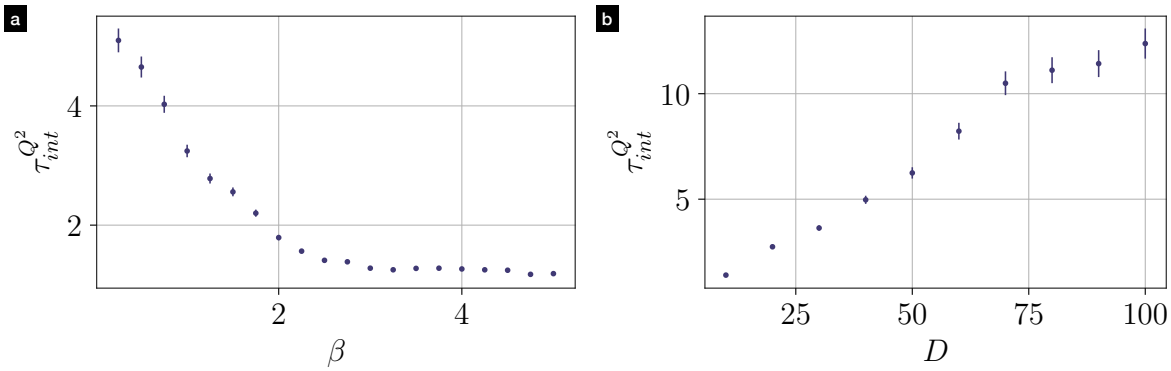


Figure 4.2: Integrated autocorrelation times for estimating Q^2 with the Wolff cluster algorithm. (a) $\tau_{int}^{Q^2}$ of $\overline{Q^2}^{(\text{Wolff})}$ Markov Chains for different coupling strengths β. D is fixed to 25. (b) $\tau_{int}^{Q^2}$ of $\overline{Q^2}^{(\text{Wolff})}$ Markov Chains for different lattice sizes D. β is fixed to 1.0.

Chapter 5

Normalizing Flows

As an alternative to methods like HMC and the Wolff algorithm, we are in the sequel going to introduce a tool from probabilistic Machine Learning called *Normalizing Flows*. Its high-level concept is to create a variational ansatz for the target density $p(\boldsymbol{x}) = \exp(-S(\boldsymbol{x}))/\mathscr{Z}$. Given a family of proper densities $q(\boldsymbol{x};\boldsymbol{\theta})$, we can optimize for the defining parameters $\boldsymbol{\theta}$ to find $\boldsymbol{\theta}^*$, such that $q(.;\boldsymbol{\theta}^*)$ describes p as well as possible with respect to a similarity measure yet to be defined. This strategy is called *Variational Inference* [26]. Once a reasonable approximation is obtained, we can efficiently sample from $q(.,\boldsymbol{\theta}^*)$ and utilize importance sampling and metropolized independence sampling to remove the bias.

This chapter contains the theory concerning variational inference with Normalizing Flows, their architectural design and various adaptations to the circle. As a prerequisite, a brief section on deep learning is prepended.

5.1 Introduction to Deep Learning

In *Machine Learning*, algorithms are designed to learn behavior based on seen data. As a further distinction, we are going to introduce *Deep Learning* methods, which are a class of machine learning algorithms that are based on artificial neural networks. Originally inspired by structures observed in biological neural matter, artificial neural networks are defined by a graph-like network structure that encodes a computational path. From a purely mathematical perspective, a neural network is a function f that maps an input to the corresponding desired output. The single chained steps that compound f are rendered flexible, because their behavior depends on a set of free parameters $\Theta = \{\boldsymbol{\theta}\}$. If we are interested in encoding a certain behavior into the network, the function must be differentiable with respect to the parameters that need to be altered. Frequently, neural networks are called *models*.

Usually, it is desired to include a high degree of nonlinearity in neural networks. This relies on the assumption that the problem to solve is non-linear because otherwise it could be solved optimally and much more efficiently without the need for neural networks.

To adapt the model $f(.;\boldsymbol{\theta})$ to a specific problem i.e. desired input-output relationship, one utilizes *gradient descent*. Given an input $\boldsymbol{z} \in \mathbb{R}^{D_1}$, let's suppose the desired output is $g(\boldsymbol{z}) \in \mathbb{R}^{D_2}$. A measure of distance between the output of the model $f(\boldsymbol{z};\boldsymbol{\theta}) \in \mathbb{R}^{D_2}$ and the desired output must additionally be defined by a differentiable *loss function* $L(f(\boldsymbol{z},\boldsymbol{\theta}), g(\boldsymbol{z})) \in \mathbb{R}$. L needs to have a global minimum at $f(\boldsymbol{z},\boldsymbol{\theta}) = g(\boldsymbol{z})$ and is often required to be convex in its arguments. For an update in the course of gradient descent, we then need to calculate the gradient of the loss function with respect to the model's parameters, $\nabla_{\boldsymbol{\theta}} L$. In an iterative process, for a single step, the model parameters are then updated through

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \langle L(f(\boldsymbol{z};\boldsymbol{\theta}), g(\boldsymbol{z})) \rangle_{\mathcal{Z}}. \tag{5.1}$$

$\eta > 0$ is a small, variable *hyperparameter* called *learning rate*. The mean over the set of all inputs $\mathcal{Z}$ is denoted by $\langle . \rangle_{\mathcal{Z}}$. In principle, this update step is repeated until a desired loss is achieved. In practice, many additional methods and alterations to this procedure are deployed to train a network (see e.g. [27, 28]). Importantly, the mean loss is usually only estimated for batches of the input samples. This adaptation is called *stochastic gradient descent*. The loss function energy landscape is usually also locally non-convex with respect to the model parameters, which is why it is possible to get stuck in a local minimum. To try to alleviate this and improve the convergence of models, many training strategies have been proposed. In this thesis, we will use optimizers that offer an adaptive learning rate such as [29] and various approaches for global learning rate scheduling.

A very efficient method for evaluating gradients with respect to all parameters of a neural network is the *backpropagation* algorithm. Today there are many frameworks for different programming languages that include functionalities like automatic differentiation, predefined network architectures proposed by literature and training tools, such as optimizers. Throughout this thesis, we will use *Pytorch* [30].

5.2 Normalizing Flow Design

Normalizing Flows are *generative models* in the sense that they can produce new instances of data that ought to fulfill certain properties. Specifically, this architecture allows drawing samples $\boldsymbol{x} \in \mathcal{X}$ from a probability density q_X. The space $\mathcal{X}$ is generally a topological manifold. Ultimately, the normalizing flow realizes a random variable X, which distribution is defined by the density q_X. Among generative models, normalizing flows are special, because due to the cleverly designed architecture, the exact density $q_X(\boldsymbol{x})$ can be determined analytically.

A flow consists of a base density q_Z and a transformation F, which acts on a vector of random variables $Z \sim q_Z$ on another topological manifold $\mathcal{Z}$. The components Z_i of the vector $Z = (Z_1, \ldots, Z_D)$ are usually independent, and generally, a base density is chosen, which allows sampling rapidly and efficiently evaluating the samples'

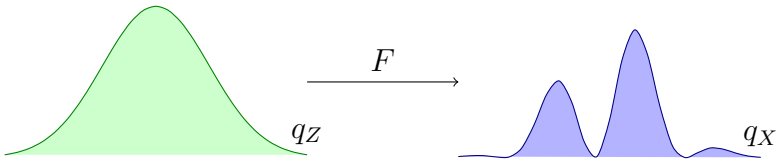


Figure 5.1: A Normalizing Flow transforms a simple Gaussian base density q_Z into a more complex multi-modal output distribution q_X by means of the transformation F.

probability. The samples $\boldsymbol{Z} \sim q_Z$ are then transformed by the model to produce $X = F(Z) \sim q_X$. The goal here is to effectively sample from a richer, more complex distribution q_X [31]. The idea is presented schematically in Figure 5.1. Given an arbitrary vector $\boldsymbol{x} \in \mathcal{X}$, we would like to evaluate the probability $q_X(\boldsymbol{x})$ in terms of F and q_Z. This is possible if we further add differentiable structures and a Riemannian metric g to $\mathcal{Z}$ and $\mathcal{X}$. Then $(\mathcal{Z}, g)$ and $(\mathcal{X}, g)$ are differentiable Riemannian manifolds. We require the manifolds $\mathcal{Z}$ and $\mathcal{X}$ to be homeomorph with equal dimension D, and the transformation F to be a *diffeomorphism*. That means it is a bijective mapping between the manifolds $\mathcal{Z}$ and $\mathcal{X}$, where the mapping itself as well as its inverse, are smooth. If $x : V \mapsto \mathbb{R}^n$ and $z : W \mapsto \mathbb{R}^n$ are coordinate systems on $\mathcal{X}$ and $\mathcal{Z}$ respectively, we can use the change of variables theorem [32]:

$$\begin{aligned} q_X(\boldsymbol{x}) &= q_Z\left(F^{-1}(\boldsymbol{x})\right) \left|\det d\left(zF^{-1}x^{-1}\right)_{x(\boldsymbol{x})}\right| \\ &= q_Z\left(F^{-1}(\boldsymbol{x})\right) \left|\det d\left(xFz^{-1}\right)_{z(F^{-1}(\boldsymbol{x}))}\right|^{-1}. \end{aligned} \tag{5.2}$$

Here d is the differential and $xFz^{-1} =: \hat{F}$ is the Matrix representation of the diffeomorphism F in local coordinates. $\det d\hat{F}$ is the determinant of the Jacobian of $\hat{F}$. Note that the determinant is not dependent on the choice of basis for the local coordinates [33]. Through this change of variables, we can evaluate q_X, if we have access to q_Z, which is true by construction. Figure 5.2 contains a schematic of the relationship between F, q_Z and q_X.

A quite useful property of diffeomorphisms is that they form a group and thus several compound transformations can be composed, and the overall transformation retains its desired behavior. Given N compound transformations

$$\begin{aligned} (F_N \circ \ldots \circ F_1)^{-1} &= F_1^{-1} \circ \ldots \circ F_N^{-1}, \\ \det d(\hat{F}_N \circ \ldots \circ \hat{F}_1)_{\boldsymbol{z}} &= \det d(\hat{F}_N)_{\hat{F}_{N-1} \circ \ldots \circ \hat{F}_1(\boldsymbol{z})} \cdots \det d(\hat{F}_1)_{\boldsymbol{z}}. \end{aligned} \tag{5.3}$$

Ultimately, we would like to leverage the properties of the Normalizing Flow to sample from a density q_X that is reasonably close to a target distribution p. At this point, we, therefore need to introduce a measure of similarity between densities. For general notational simplicity and the easier discussion of key flow construction aspects, we will in the following associate $\mathcal{X}$ and $\mathcal{Z}$ with $\mathbb{R}^D$ and thus $\hat{F} = F$.

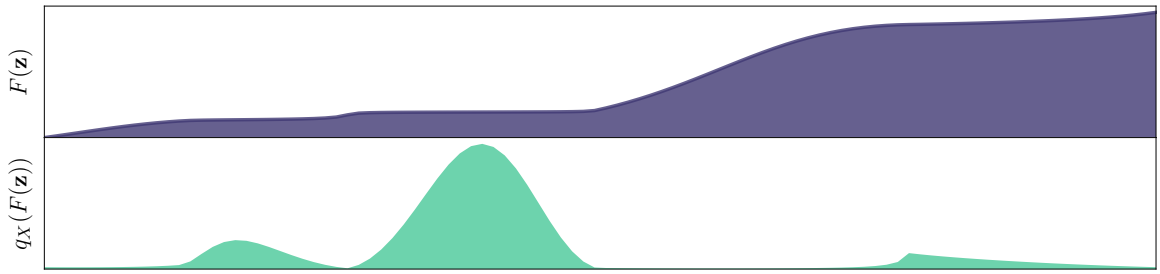


Figure 5.2: The transformation F determines the density q_X. Here the base density q_Z is set to be uniform and on a finite interval.

5.3 Kullback-Leibler Divergence

As a measure of similarity between the target density p and the flow output q_X, the *Kullback-Leibler* (KL) divergence may be used. Crucially for us, if we are effectively only interested in the gradient of the KL divergence, we can optimize with respect to the unnormalized target function $\tilde{p}$. Therefore, in the backpropagation step, it serves as a loss function, which is shifted by a constant factor. To our disadvantage, through this shift, we lose insight into whether a trained model has converged sufficiently. As the divergence is not symmetric, there are two forms, which will prove useful in different setups. The reverse KL divergence is defined as

$$\begin{aligned} \mathbb{KL}(q_X||p) &= \mathbb{E}_{X\sim q_X}\left[\log \frac{q_X(X)}{p(X)}\right] \\ &= \mathbb{E}_{Z\sim q_Z}\left[\log q_Z(Z) - \log\left|\det dF(Z)\right| - \log p(F(Z))\right]. \end{aligned} \tag{5.4}$$

The change of variables theorem from Eq. 5.2 was used to exchange the integration over $\mathcal{Z}$ and $\mathcal{X}$. Here we see that although the KL divergence provides a loss on $\mathcal{X}$, we only need to be able to calculate the expectation value with respect to q_Z. Taking the gradient of Eq. (5.4) with respect to the model parameters $\boldsymbol{\theta}$, the target probability term can be modified:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}\mathbb{E}_{Z\sim q_Z}[\log p(F(Z;\boldsymbol{\theta}))] &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{Z\sim q_Z}\left[\log \frac{\tilde{p}(F(Z;\boldsymbol{\theta}))}{\mathcal{Z}}\right] \\ &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{Z\sim q_Z}\left[\log \tilde{p}(F(Z;\boldsymbol{\theta})) - \log \mathcal{Z}\right] \\ &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{Z\sim q_Z}\left[\log \tilde{p}(F(Z;\boldsymbol{\theta}))\right]. \end{aligned} \tag{5.5}$$

This holds as $\mathcal{Z}$ is a constant with regard to $\boldsymbol{\theta}$. For the applicability of variational inference methods to target densities with intractable normalization, this property turns out to be integral. While the KL divergence as a loss function suffers from less than optimal behavior as we shall see, this requirement makes the search for alternatives

difficult. The modified term for the reverse KL divergence

$$\begin{aligned}\mathcal{F}_{q_X} &= \mathbb{E}_{X\sim q_X}[\log q_X(X)] - \mathbb{E}_{X\sim q_X}[\log \tilde{p}(X)] \\ &= -\mathbb{H}[q_X] + \mathbb{E}_{X\sim q_X}[S(X)]\end{aligned} \tag{5.6}$$

is called *variational free energy* or *Helmholtz free energy*. $\mathbb{H}[q_X]$ is the *differential entropy* of the distribution q_X. In the machine learning literature $\log \mathscr{Z}$ is referred to as the *evidence*. Thus, since $\mathbb{KL}(q_X||p) \geq 0$,

$$-\mathcal{F}(q_X) \leq \log \mathscr{Z} = \log \int_{\mathcal{X}} d\boldsymbol{x}\ \tilde{p}(\boldsymbol{x}) \tag{5.7}$$

is called the *evidence lower bound* (ELBO).

In the training case, where we only have access to the shifted Reverse KL divergence, one can estimate the KL divergence. In [34] it is proven that

$$\mathbb{KL}(q_X||p) = \frac{1}{2}\mathrm{Var}_{X\sim q_X}(S + \ln q_X(X)) + \mathcal{O}\left(\mathbb{E}_{X\sim q_X}\left[|\frac{p(X)}{q_X(X)} - 1|^3\right]\right). \tag{5.8}$$

The forward KL divergence is defined as

$$\begin{aligned}\mathbb{KL}(p||q_X) &= \mathbb{E}_{X\sim p}\left[\log \frac{p(X)}{q_X(X)}\right] = -\mathbb{E}_{X\sim p}[\log q_X] + C \\ &= -\mathbb{E}_{X\sim p}\left[\log q_Z\left(F^{-1}(X)\right) + \log\left|\det d\left(F^{-1}\right)(X)\right|\right] + C.\end{aligned} \tag{5.9}$$

C is a constant with respect to any changes in q_X, which is why we can neglect it for a training procedure that uses gradients. Optimizing for the forward KL divergence, i.e. minimizing Eq. 5.9 with respect to some parameters $\boldsymbol{\theta}$, is equivalent to a *maximum likelihood* approach:

$$\begin{aligned}\arg\max_{\boldsymbol{\theta}}\{\mathbb{E}_{X\sim p}[q_X(X;\boldsymbol{\theta})]\} &= \arg\max_{\boldsymbol{\theta}}\{\mathbb{E}_{X\sim p}[\log q_X(X;\boldsymbol{\theta})]\} \\ &= \arg\min_{\boldsymbol{\theta}}\{-\mathbb{E}_{X\sim p}[\log q_X(X;\boldsymbol{\theta})]\} \\ &= \mathbb{KL}(p||q_X).\end{aligned} \tag{5.10}$$

The idea is to maximize the likelihood $q_X(X;.) : \boldsymbol{\theta} \mapsto q_X(X;\boldsymbol{\theta})$ for the seen data X. Additionally, it was used that log is a strictly increasing function on $\mathbb{R}_{>0}$.

The two forms of the KL divergence display different tendencies when used as a loss function. Consider the reverse KL in Eq. 5.4. If in the course of a minimization procedure of q_X, $\boldsymbol{x} \in \mathrm{supp}(q_X)$, then $\boldsymbol{x} \in \mathrm{supp}(p)$ necessarily. Otherwise, the KL divergence would diverge. So $\mathrm{supp}(q) \subseteq \mathrm{supp}(p)$ and thus in practice the support of p will typically be underestimated, which is why the reverse KL divergence is said to be *zero-forcing* [28]. The other way around works for the forward KL divergence. It is said to be *zero-avoiding* [28]. If, in Eq. 5.9, $\boldsymbol{x} \in \mathrm{supp}(p)$, then $\boldsymbol{x} \in \mathrm{supp}(q)$ necessarily

and so the support of p will typically be overestimated.

During the optimization, as usual in stochastic gradient descent, the expectation values of the gradient need to be estimated by Monte Carlo integration. If $\{\boldsymbol{z}_i\}_{i=1}^N$ is a set of realizations of Z, then the gradient of the reverse KL divergence with respect to the model parameters $\boldsymbol{\theta}$ is approximated by

$$\nabla_{\boldsymbol{\theta}} \mathbb{KL}(q_X||p) \approx -\frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \log|\det dF(\boldsymbol{z}_i;\boldsymbol{\theta})| + \nabla_{\boldsymbol{\theta}} \log \tilde{p}(F(\boldsymbol{z}_i;\boldsymbol{\theta})). \tag{5.11}$$

Similarly, we estimate the expectation value for the gradient of the forward KL divergence. Given a set of realizations $\{\boldsymbol{x}_i\}_{i=1}^N$ of $X \sim p$,

$$\nabla_{\boldsymbol{\theta}} \mathbb{KL}(p||q_X) \approx \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \log q_Z\left(F^{-1}(\boldsymbol{x}_i;\boldsymbol{\theta})\right) + \nabla_{\boldsymbol{\theta}} \log\left|\det d\left(F^{-1}\right)(\boldsymbol{x}_i;\boldsymbol{\theta})\right|. \tag{5.12}$$

5.4 Normalizing Flow Construction

We have thus far introduced an abstract framework of Normalizing Flow models. Apart from the constraints discussed in Section 5.2, concrete model architectures can be freely invented. Regarding the computational cost of training and density evaluation, careful flow construction proves quite valuable. If we frequently want to evaluate the density $q_X(\boldsymbol{x})$, or the training is set up bi-directionally, an efficient inversion of F is desirable. Furthermore, for all training, it is beneficial to be able to quickly evaluate the Jacobian determinants $\det dF$ and possibly $\det d\left(F^{-1}\right)$. Computing the determinant of a $D \times D$ matrix is characterized by computational complexity $\mathcal{O}\left(D^3\right)$ [35], which scales quite poorly with the dimension but can be drastically improved dependent on the matrix form. As a result, many cleverly designed flow architectures have been proposed. For an extensive overview, we refer to [31, 36].

A simple example of how to achieve both of these features is to use a linear function F. For a vector $\boldsymbol{z} \in \mathbb{R}^D$, invertible $A \in \mathbb{R}^{D\times D}$ and $\boldsymbol{b} \in \mathbb{R}^D$, define

$$F(\boldsymbol{z}) = A\boldsymbol{z} + \boldsymbol{b}. \tag{5.13}$$

Then the inverse and Jacobian determinants are given by:

$$\begin{aligned} F^{-1}(\boldsymbol{x}) &= A^{-1} \cdot (\boldsymbol{x} - \boldsymbol{b}), \\ \det dF &= \det A. \end{aligned} \tag{5.14}$$

Depending on how A is defined, the inverse A^{-1} as well as $\det A$ can efficiently be evaluated. An example would be to constrain A to be diagonal or triangular.

A framework for F that is less restricting, relies on carefully separating the indices of the variables $\boldsymbol{z} = (z_1, \ldots, z_D)$ into three disjoint sets I_A,I_F and I_P, such that $I_A \cup I_F \cup I_P =$

$\{1, \ldots, D\}$. This defines *transformed* variables with indices in I_A, *frozen* variables that are not transformed and possibly serve as inputs to the transformation with indices in I_F and passive variables that are left unchanged and do not affect the transformation with indices in $I_P \subseteq \{1, \ldots, D\}$. To manifestly achieve this, we define the map F:

$$F_i(\boldsymbol{z}) = \begin{cases} \tau(z_i; C) & i \in I_A \\ z_i & i \in I_F \cup I_P. \end{cases} \tag{5.15}$$

We have introduced the *transformer* function τ, which transforms $\boldsymbol{z}$ elementwise. The additional parameters $C \in \mathbb{R}^{|I_A| \times m}$ with $m \in \mathbb{N}$ are an arbitrary matrix that is defined by the *conditioner* function c:

$$C = c\left(\{z_j\}_{j \in I_F}\right). \tag{5.16}$$

We therefore provide τ with m parameters for each transformed value z_j. Since the inverse of τ is known, we can evaluate the inverse of the entire map as

$$F^{-1}(\boldsymbol{x})_i = \begin{cases} \tau^{-1}(x_i;) & i \in I_A \\ x_i & i \in I_F \cup I_P. \end{cases} \tag{5.17}$$

If τ is easily invertible, F is as well. The partial derivatives

$$\frac{\partial F_i}{\partial z_j} = \begin{cases} \frac{\partial \tau}{\partial z_i}\delta_{i,j} =: A_{ij} & i \in I_A, j \in I_A \\ \delta_{i,j} & i \in I_F \cup I_P \\ \frac{\partial \tau}{\partial C_i}\frac{\partial C_i}{\partial z_j} =: B_{ij} & i \in I_A, j \in I_F \end{cases} \tag{5.18}$$

reveal that the Jacobian is a block matrix:

$$dF = \begin{pmatrix} A & 0 & 0 \\ B & \mathbb{I} & 0 \\ 0 & 0 & \mathbb{I} \end{pmatrix}. \tag{5.19}$$

So the Jacobian determinant is simply given by

$$\begin{aligned} \det dF = \det A &= \prod_{k \in I_A} \frac{\partial \tau}{\partial z_k}, \\ \log |\det dF| &= \sum_{k \in I_A} \log \left| \frac{\partial \tau}{\partial z_k} \right|. \end{aligned} \tag{5.20}$$

Note that the determinant is invariant under permutations of the variable indices. Thus, this result can always be achieved for any disjoint subdivision. In this construction, we see that the conditioner function c can be designed arbitrarily and an efficient

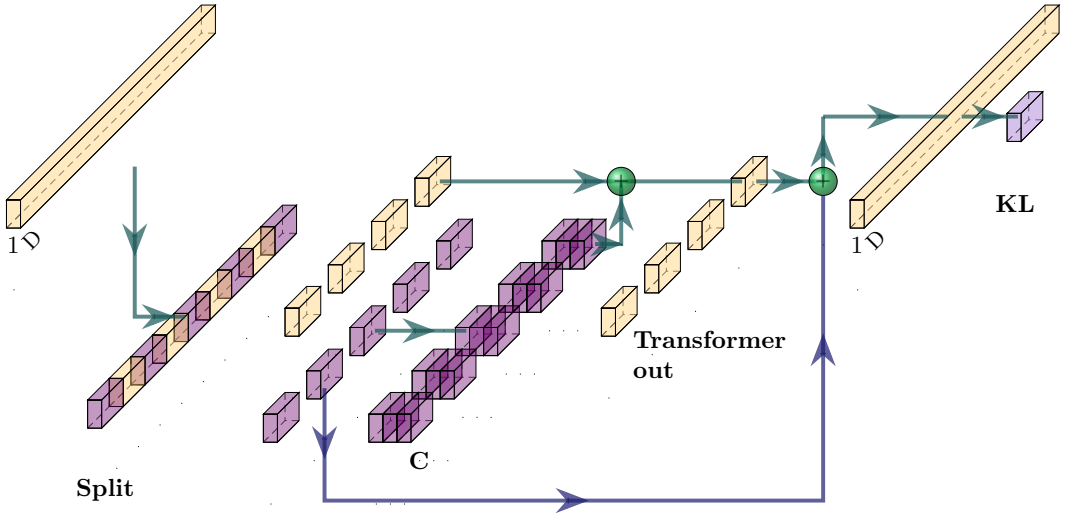


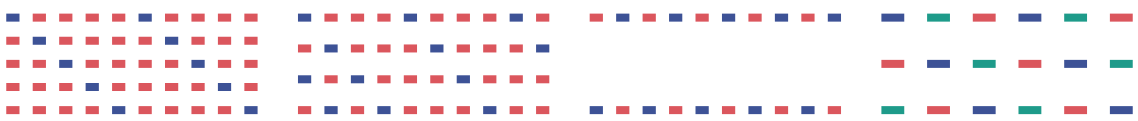
Figure 5.3: Depiction of a transformer-conditioner layer. In the depicted layer, the one-dimensional input is split into two parts in a checkerboard fashion. The ochre-colored lattice points are actively transformed. The purple points are mapped to the conditioner output C and used in the transformation. They remain however unchanged in this layer.

Jacobian evaluation is still possible with complexity $\mathcal{O}(D)$. In particular, one can employ a deep learning setup and utilize the expressive power of neural networks. We will in the following introduce two variants of this framework. An exemplary transformer-conditioner structure is sketched in Figure 5.3.

5.4.1 Coupling Layers

For *Coupling Layers*, all variables z_i with $i \in I_F$ serve as input for the conditioner. As discussed, inversion and calculating the Jacobian determinant can potentially be achieved very efficiently. That is, if c,τ and τ^{-1} can be evaluated quickly. To increase the expressiveness of the flow, several coupling layers can be stacked as proven in eq. (5.3). The choice of size and position of the elements within the sets I_A, I_F and I_P are completely arbitrary as long as they are disjoint, and its union contains all lattice point indices. It is however desirable to transform every point at least once. So the number of required layers is dependent on the choice of set size. Figure 5.4 contains several coupling-layer split realizations.

It is an unsolved problem, whether universality is given for several stacked coupling layers strictly less than D [31]. Universality, in this case, means that assuming F is expressive enough, it is possible to model any target density p arbitrarily well with q_X using any base distribution q_Z.



(a) (b) (c) (d)

Figure 5.4: In Coupling Layers the input is split into three sets: The transformed lattice variables (blue,I_A), the variables that are used as input to the conditioner (orange,I_F) and the variables that are used for neither and left unchanged (green,I_P). See Eqs. (5.16) and (5.15). To transform all variables, there are varying numbers of layers needed.

5.4.2 Autoregressive Layer

For *autoregressive* flows, we re-envision the dimension of the input as time-like. A complete map is effectively built of D stacked transformer-conditioner layers. At step $t \in \{1, \dots, D\}$, the conditioner is only dependent on $\boldsymbol{z}_{<t} = \{z_1, \dots, z_{t-1}\}$, z_t is transformed like

$$x_t = F_t(\boldsymbol{z}) = \tau(z_t; C_t) \tag{5.21}$$

and $\boldsymbol{z}_{>t}$ is left unchanged. Since at every step the used input $z_{\leq t}$ is disjointly separated as conditioner input and transformed variable, the Jacobian determinant can efficiently be evaluated. During the forward pass of F it is possible to transform all entries in parallel. On the contrary, the inversion of an autoregressive layer needs to be done sequentially and thus D backward passes are required. There have been suggestions to circumvent this drawback [31].

This construction is built around the fact that any probability density can be decomposed into a product of conditional densities using the chain rule:

$$p(\boldsymbol{x}) = \prod_{t=1}^{D} p(x_t | \boldsymbol{x}_{<t}). \tag{5.22}$$

And thereby universality for the autoregressive flow model can easily be proven [31, 36].

5.5 Normalizing Flows on Tori

Within the transformer-conditioner architecture framework, one is left with the remaining task of finding a suitable conditioner and transformer function. For $\mathcal{X} = \mathcal{Z} = \mathbb{R}^D$, several transformer functions have been proposed (see [37–40] among others). The space of variables that characterize the Quantum Rotor system with D lattice points, is a cartesian product of D circles $\mathbb{S}^1 \times \cdots \times \mathbb{S}^1$. This is homeomorph to the torus $\mathbb{T}^D$. So we are looking for a way to define a flow transformation $F : \mathbb{T}^D \mapsto \mathbb{T}^D$. Several proposed solutions for flows on Riemannian manifolds are discussed in [6, 31, 36].

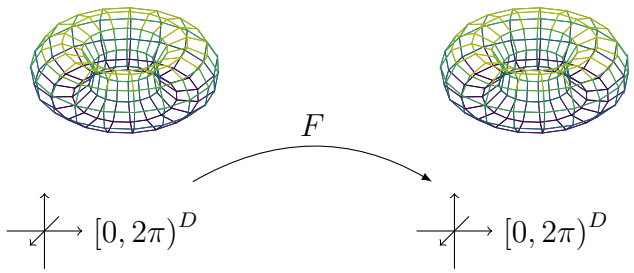


Figure 5.5: The flow acts the hypercube $[0, 2\pi)^D$, where opposing edges are associated.

A comparatively simple solution proposed by [6] is to use diffeomorphisms that are tailored to tori in the first place.

The idea here is to use the presentation of the torus as a hypercube $[0, 2\pi)^D$, where we associate opposite edges with each other. A local coordinate system is given by the identity. Hence, with respect to the torus $\hat{F} = F$. See Figure 5.5 for a visualization. The periodicity, which is crucial for a valid representation needs to be retained by the chosen transformations.

5.5.1 Transformations tailored to Tori

We require element-wise transformations of the input. As [6] propose it is sufficient to define transformations that are diffeomorphisms on the circle $\mathbb{S}^1$. Representing the circle as the interval $[0, 2\pi]$, a map F like that needs to obey periodic boundary conditions and ensure invertibility. For an arbitrary $\alpha \in \mathbb{S}^1$, periodicity on the circle translates to

$$F(0) = F(2\pi) = \alpha. \tag{5.23}$$

As the resulting flow density ought to be periodic as well, we additionally impose

$$\partial_\phi F(0) = \partial_\phi F(2\pi) \tag{5.24}$$

on the derivative to produce a continuous probability density q_X on the circle. Invertibility is obtained by requiring strict monotony, for example with

$$\partial_\phi F > 0. \tag{5.25}$$

In 5.3, the possibility to chain diffeomorphisms and still get a valid overall transformation was mentioned. To additionally increase the expressiveness of the flow, [6] suggest a convex combination of $K \in \mathbb{N}$ transformations F_i $i \in \{1, \dots, K\}$, such as

$$F(\phi) = \sum_i \rho_i F_i(\phi), \tag{5.26}$$

where $\sum_i \rho_i = 1$ and $\rho_i > 0$. This unfortunately renders the overall transformation not analytically invertible and thus numerical inversion must be used if the inverse is

required.

Circular Splines

Rational Quadratic Splines (RQS) were introduced by [40] as maps that operate on $\mathbb{R}$. Nevertheless, the construction can be restricted to a finite interval and include periodic boundary conditions in a straightforward manner [6]. Therefore, they can be adapted to fulfill the conditions (5.23), (5.24) and (5.25) to operate on the circle. The slightly modified map is called a *Circular Spline*(CS).

The idea behind the CS transformation is to build a flexible, expressive and analytically invertible function out of piece-wise defined splines. In the course of that, we split the finite input and output intervals $[0, 2\pi)$ into $K \in \mathbb{N}$ subdivisions by assigning a width w_k and a height h_k respectively with $k \in \{0, \dots, K-1\}$. The spline additionally requires values for the derivatives (d_k, d_{k+1}) at the boundaries of the intervals to be uniquely determined. Using the definition $\varepsilon_k(\phi) = (\phi - w_k)/(w^{(k+1)} - w_k)$ and $s_k = (h^{(k+1)} - h_k)/(w^{(k+1)} - w_k)$, the rational quadratic spline for the subinterval k is given by the formula:

$$F_{rqs}^{(k)}(\phi) = \frac{\alpha_k(\phi)}{\beta_k(\phi)} = h_k + \frac{(h_{k+1} - h_k)\left[s_k\varepsilon_k^2(\phi) + d_k\varepsilon_k(\phi)(1-\varepsilon_k(\phi))\right]}{s_k + [d_{k+1} + d_k - 2s_k]\varepsilon_k(\phi)(1-\varepsilon_k(\phi))^2}. \tag{5.27}$$

Since the overall piece-wise defined map ought to be at least in C^1, the derivatives of neighboring sub-intervals are fixed to match. Due to periodic boundary conditions on the circle, we also require $d_K = d_0$. A depiction of the resulting construction can be found in Figure 5.6a. For K divisions of the initial interval, $3K$ parameters have to be provided in total. The partial derivative of the spline with respect to the input angle is:

$$\frac{\partial F_{rqs}^{(k)}}{\partial \phi} = \frac{s_k^2[d_{k+1}\varepsilon^2 + 2s_k(1-\varepsilon_k(\phi)) + d_k(1-\varepsilon_k(\phi))^2]}{[s_k + [d_{k+1} + d_k - 2s_k]\varepsilon_k(\phi)(1-\varepsilon_k(\phi))]^2}. \tag{5.28}$$

Since F_{rqs} is inherently built from analytically invertible functions and the subdivisions of input and output intervals do not overlap, the overall map can be inverted analytically as well and the inverse is also differentiable [40]. We can increase the expressiveness of the transformation by choosing a larger number of subdivisions K, so additional strategies like a convex combination from Eq. (5.26) are not needed. Further Implementation details can be found in appendix C.1.

Moebius Transformations

Another map, suggested by [6], that directly operates on the Torus $\mathbb{T}^D$ is the *Moebius* transformation. Its definition is best understood if it is depicted on the circle as in Figure 5.6b. To apply the Moebius transformation $F_{moebius}$ to an angle ϕ, first, the latter has to be converted into a complex number - or equivalently a two-dimensional

vector $z(\phi)$ - with the unit norm. Given an auxiliary complex number or vector $\boldsymbol{w} \in \mathbb{R}^2$ with $\|\boldsymbol{w}\| < 1$, the Moebius transformation for an angle ϕ is defined as:

$$\tilde{F}_{moebius}(\phi; \boldsymbol{w}) = \frac{1 - \|\boldsymbol{w}\|^2}{\|z(\phi) - \boldsymbol{w}\|^2}(z(\phi) - \boldsymbol{w}) - \boldsymbol{w}. \tag{5.29}$$

The result of this transformation is that an input density is decreased close to $\boldsymbol{w}$ and increased for points on the circle farther away from w. Eq. 5.29 yields a vector in $\mathbb{R}^2$, which still needs to be converted to the output angle. Although all conditions (5.23), (5.24) and (5.25) are met, we can additionally encode $F_{moebius}(0) = \alpha \in (0, 2\pi]$ manifestly. The complete map is determined by the formula

$$F_{moebius}(\phi) = \mathrm{mod}\left[\mathrm{atan2}\left(\tilde{F}_{moebius}(\phi)\right) - \mathrm{atan2}\left(\tilde{F}_{moebius}(0)\right) + \alpha\right]_{[0,2\pi)}. \tag{5.30}$$

The atan2 [41] function maps a vector $\boldsymbol{x} = (x, y)$ to the angle between $\boldsymbol{x}$ and the positive x-axis. The derivative is given by

$$\begin{aligned} \frac{\partial F_{moebius}}{\partial \phi}(\phi) &= \frac{\partial}{\partial \phi}\mathrm{atan2}\left(\tilde{F}_{moebius}(z(\phi))\right) \\ &= d(\mathrm{atan2})\left(\tilde{F}_{moebius}\right) \cdot \mathrm{d}\tilde{F}_{moebius}(\boldsymbol{z}) \cdot dz(\phi) \end{aligned} \tag{5.31}$$

Here $\cdot$ denotes matrix multiplication. An explicit formula can be found in appendix C.1.

As the composition of Moebius transformations yields another Moebius transformation, an increased expressiveness can only be achieved using a convex combination of compound maps.

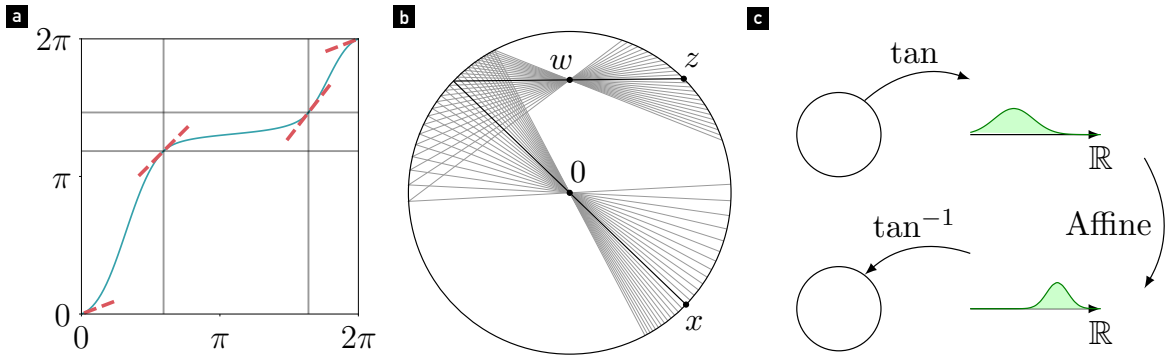


Figure 5.6: Depictions of transformations tailored to $\mathbb{S}^1$. (a) Circular Splines. (b) Moebius Transformation. (c) Non-compact projection (NCP).

Non-Compact Projections

In their paper, [6] have also included the idea of projecting the circle onto $\mathbb{R}$ first, apply an unconstrained transformation and map back onto the circle. An angle $\phi \in \mathbb{S}^1$

is projected onto the *non-compact* space $\mathbb{R}$ with $x = \tan\left(\frac{\phi}{2} - \frac{\pi}{2}\right)$. Given parameters $\alpha > 0$ and $\beta \in \mathbb{R}$, we apply an affine transformation $g(x) = \alpha x + \beta$ on $\mathbb{R}$ and map back onto the circle with arctan. A depiction of the described steps can be found in Figure 5.6c. Mathematically, the complete map is

$$F_{ncp}(\phi) = 2 \arctan\left(\alpha \tan\left(\frac{\phi}{2} - \frac{\pi}{2}\right) + \beta\right) + \pi \tag{5.32}$$

and the partial derivative of the NCP with respect to the input angle:

$$\frac{\partial F_{ncp}}{\partial \phi} = \left[\frac{1+\beta^2}{\alpha} \sin^2\left(\frac{\phi}{2}\right) + \alpha \cos^2\left(\frac{\phi}{2}\right) - \beta \sin\phi\right]^{-1}. \tag{5.33}$$

As [6] point out, F_{ncp} is not defined at the points 0 and 2π. We can however continue F_{ncp}, such that $F_{ncp}(0) = 0$ and $F_{ncp}(2\pi) = 2\pi$, since

$$\begin{aligned} \lim_{\phi \to 2\pi^-} F_{ncp}(\phi) &= 2\pi, \\ \lim_{\phi \to 0^+} F_{ncp}(\phi) &= 0. \end{aligned} \tag{5.34}$$

To increase numerical stability at the edges of the interval $[0, 2\pi]$, we may evaluate F_{ncp} through the Taylor approximations:

$$\begin{aligned} F(\theta)|_{\theta \approx 0} &= \frac{\theta}{\alpha} \\ F(\theta)|_{\theta \approx 2\pi} &= 2\pi + \frac{\theta - 2\pi}{\alpha} \end{aligned} \tag{5.35}$$

If numerical instability proves to be negligible, it is also possible to simply apply the modulus of 2π to the result.

As the non-compact projections form a group, the composition of several compound NCP functions does not increase the expressiveness of the overall transformation. Therefore, convex combinations are used. Although the construction works for affine maps, the conditions (5.23), (5.24) and (5.25) are not met for a general diffeomorphism on $\mathbb{R}$ [6]. Further implementation details are described in appendix C.1.

5.5.2 Residual Diffeomorphism

Neural network architectures have tended towards greater depth since often performance improves and models are capable of learning more complex features of the data. However, this leads to an increasingly difficult training process. Mainly due to the problem of vanishing gradients. Residual network layers proposed in [42] help mitigate this problem.

A neural network layer F is turned into a residual layer by constructing the layer as:

$$\boldsymbol{x} = F(\boldsymbol{z}) + \boldsymbol{z}. \tag{5.36}$$

In the same fashion, we can construct a residual coupling layer. [43] give conditions to construct a residual transformation on $\mathbb{R}$, for which the overall function remains invertible. On the compact manifold $\mathbb{T}^D$, we can as previously stated use a convex combination to achieve a similar result. Given a diffeomorphism F and parameters (ρ_1, ρ_2) with $\sum_i \rho_i = 1$:

$$\boldsymbol{x} = \rho_1 F(\boldsymbol{z}) + \rho_2 \boldsymbol{z}. \tag{5.37}$$

A reason why residual constructions might have proven to be useful is that especially in deep architectures the transformation of a single layer should not have to deviate far from the identity [42]. Like in a general computer vision task with convolutional networks, the coupling layer is capable of producing the identity map itself. I.e. the parameters needed for the identity map lie within the accepted space. However, it turns out that this manifest construction often improves training.

The combination parameters and their evolution along the training process can be designed arbitrarily. The implementation details can be found in Appendix C.1

5.6 Transformations without an analytical Inverse

We have previously introduced transformations, for which there is no analytical inverse. If the inverse F^{-1} is needed, it needs to be computed numerically and so there will be some error attached to the result. This is e.g. the case if a Normalizing Flow model is trained with the Revere KL divergence setup, and we would like to determine the probability of a given sample $\boldsymbol{x} \in \mathcal{X}$. Additionally, it might be necessary to differentiate through the numerical inverse if the Flow is trained bi-directionally.

Numerical inversion of a strictly monotonous function can efficiently be done using bisection search. For a desired maximum error on the exact inverse ε, the bisection algorithm has computational complexity $\mathcal{O}(-\log \epsilon)$. In practice, it is crucial to determine an accuracy ε that is small enough to support stable training. On the other hand, it should be large enough to make an inversion in a reasonable number of iterations possible. The value $\varepsilon = 10^{-4}$ has proven to be a good choice.

If Bisection search is used, it is prohibitively costly to evaluate the gradient of a numerically inverted function using tools like automatic differentiation implemented in ML libraries. Luckily, the calculation is possible if the gradient of the forward direction is known. These relationships shall be proven in the following.

Consider the function $F : \mathcal{Z} \mapsto \mathcal{X}$. We restrict it to act element-wise on the input $\boldsymbol{z}$. Thus, if $\widetilde{F^{-1}}$ is the numerical inverse of F, we need

$$\frac{\partial \widetilde{F^{-1}}(x;\theta)}{\partial x}, \tag{5.38}$$

where x is an arbitrary component of the vector $\boldsymbol{x} \in \mathcal{X}$ and $\theta \in \mathbb{R}$ is a parameter to the function. The inverse function theorem states that

$$\frac{\partial F^{-1}(x;\theta)}{\partial x} = \left[\frac{\partial F(z;\theta)}{\partial z}\right]^{-1}. \tag{5.39}$$

Therefore, we can express the gradient of the inverse with the gradient of the function itself. If we furthermore require the gradient of the inverse with respect to the parameter θ, this needs to be slightly adapted. We start similarly to the way we prove the inverse function theorem:

$$F\left(z;\theta\right) = F\left(F^{-1}(x;\theta);\theta\right) = x. \tag{5.40}$$

However, now we differentiate both sides with respect to θ:

$$\frac{\partial F\left(z;\theta\right)}{\partial \theta} + \frac{\partial F\left(z;\theta\right)}{\partial z}\frac{F^{-1}(x;\theta)}{\partial \theta} = 0. \tag{5.41}$$

Solving for $\frac{\partial F^{-1}(x;\theta)}{\partial \theta}$ yields

$$\frac{\partial F^{-1}(x;\theta)}{\partial \theta} = -\left[\frac{\partial F\left(z;\theta\right)}{\partial z}\right]^{-1} \cdot \frac{\partial F\left(z;\theta\right)}{\partial \theta}. \tag{5.42}$$

With a small addition, we again arrive at an expression that only depends on derivatives of the function itself. For a more extensive review of differentiation through numerically inverted functions, see [44].

5.7 Conditioner Network Architectures

The transformer-conditioner construction described in Section 5.4, frees up the conditioner to be any arbitrary function. As such it can also be designed to be a neural network. A simple model architecture that can be used is the *Dense* or *Linear* layer. Here every output x_i with $i \in \{1, \ldots, D\}$ is a linear function of every input z_i with $i \in \{1, \ldots, D\}$ composed with a non-linear *activation* function σ:

$$x_i = \sigma\left(\sum_j w_{ij} z_j + b_i\right). \tag{5.43}$$

The weights w_{ij} are elements of a matrix $W \in \mathbb{R}^{D\times D}$, the biases b_i of a vector in $\mathbb{R}^d$. The nonlinearity σ is applied to increase the expressiveness of the layer. Several such layers stacked on top of each other are called a *Multi Layer Perceptron* (MLP).

Another type of layer, which was also used in [10, 45] is a convolutional layer. Here a kernel is used to create an output from the input via a convolution operation. Suppose, we are dealing with a one-dimensional input and a kernel of size $K < D$. Then

$$x_i = \sigma\left(b_i + \sum_{j=-\lfloor K/2\rfloor}^{\lfloor K/2\rfloor} w_{i+j} z_{i+j}\right) \tag{5.44}$$

The number of kernels per layer is variable as well as kernel size and stride of the convolution and the behavior around the input edges.

The convolutional layer seems to be a reasonable candidate for a conditioner for a model that solves the Quantum Rotor since the action is calculated solely through local next-neighbor differences. It is widely used in computer vision for exactly the possibility to detect local features, for which the exact location within the given input is less important. On the edges, we can use periodic boundary conditions, as this fits the periodic boundary conditions on the lattice. Since the convolution operation reuses weights across the input and acts only locally, the layer may potentially be used to predict features for inputs of different lattice sizes [7].

We have stated in Section 5.5 that functions, which act on the hypercube representation of the torus $\mathbb{T}^D$, need to retain periodicity at the edges. As in general, the neural networks can receive any input representation, [6] suggest mapping the input back to the torus with

$$z_i \mapsto (\cos(z_i), \sin(z_i)), \tag{5.45}$$

before it is fed into the model.

5.8 Base Densities

Generally, any valid probability density is allowed as base density q_Z for the Normalizing Flow model. As such it should be normalized, and optimally, but not necessarily, we should be able to sample rapidly from it. Since we would like to operate on the n-dimensional torus, we need to use a distribution that is defined on $\mathbb{S}$. A quite simple choice is the uniform distribution

$$q_Z(z) = \frac{1}{2\pi}. \tag{5.46}$$

Another option is the Von-Mises distribution, which in contrast to the uniform distribution allows for a localized peak. With respect to a concentration parameter $\kappa > 0$

and the mean of the distribution $\mu \in \mathbb{R}$ around which the density is peaked on $\mathbb{S}$,

$$q_Z(z) = \frac{\exp(\kappa \cos(z-\mu))}{2\pi \mathcal{I}_0(\kappa)}. \tag{5.47}$$

Here $\mathcal{I}_0$ is the modified Bessel function of the first kind as defined in Eq. (2.37).

5.9 Equivariant Normalizing Flows

There are target densities, which are symmetric with respect to certain transformations. Especially, if the density defines a physical system, this is often the case and some theories are even designed in such a way that symmetries like local gauge invariance are respected.

For a density given by the normalized Boltzmann factor

$$p(\boldsymbol{\phi}) = \frac{\exp(S(\boldsymbol{\phi}))}{\mathscr{Z}}, \tag{5.48}$$

we require that a symmetry transformation leaves the action S and the measure $\mathcal{D}\phi$ unchanged. The Rotor has a global translational symmetry. Since its action only depends on the derivative of the angle, it is invariant under:

$$S_{disc}\left(\sum_{j\tau\in\Lambda} \mathrm{mod}(\phi_j + \alpha)_{[0,2\pi)} \boldsymbol{e}_j\right) = S_{disc}(\boldsymbol{\phi}), \qquad \alpha \in [0, 2\pi). \tag{5.49}$$

In addition, as we have seen in Section 2.2, the topological charge is a symmetric quantity.

We can manifestly incorporate symmetries into Normalizing Flows as well, in the sense that q_X is invariant under the included symmetry transformations. This has been shown e.g. in [8, 46, 47]. This reduces the necessity for model complexity and improves performance [45].

Given a symmetry transformation f on $\mathcal{X}$ and $\mathcal{Z}$, the model density q_X is symmetric, if the flow base density q_Z is invariant under f and the transformation F is equivariant, i.e. it commutes with f:

$$\begin{aligned} q_X(f(\boldsymbol{x})) &= q_Z((F^{-1}\circ f)(\boldsymbol{x}))\left|\det d(F^{-1}\circ f)_{\boldsymbol{x}}\right| \\ &= q_Z((f\circ F^{-1})(\boldsymbol{x}))\left|\det d(f\circ F^{-1})_{\boldsymbol{x}}\right| \\ &= q_Z(F^{-1}(\boldsymbol{x}))\left|\det df_{F^{-1}(\boldsymbol{x})}\cdot \det dF^{-1}_{\boldsymbol{x}}\right| \\ &= q_Z(F^{-1}(\boldsymbol{x}))\left|\det dF^{-1}_{\boldsymbol{x}}\right| \\ &= q_X(\boldsymbol{x}). \end{aligned} \tag{5.50}$$

It was used that $\left|\det df_{F^{-1}(\boldsymbol{x})}\right| = 1$, which is implied by the invariance of the metric. Usually, the flow transformation is a chain of several constituent transformations $F_i \in \{F_1, \ldots, F_N\}$. The entire transformation F acts equivariantly, if all constituent transformations act equivariantly, since

$$\begin{aligned} F(f(Z)) &= F_N \circ \ldots \circ F_1(f(Z)) = F_N \circ \ldots \circ f \circ F_1(Z) = f \circ F_N \circ \ldots \circ F_1(Z) \\ &= f(F(Z)). \end{aligned} \tag{5.51}$$

To incorporate global shift symmetry into the flow transformations, we can use the treatment discussed in [7, 8, 45]. For the Rotor, the lattice variables $e^{i\phi_i} \in U(1)$ commute. The treatment for non-abelian gauge theories can be found in [7, 8].

5.10 Normalizing Flows for Observable Estimation

We will now briefly recapitulate how a Normalizing Flow can be used to estimate expectation values.

5.10.1 Normalizing Flows as Sampler

After a Normalizing Flow has been trained, we can efficiently produce samples $\boldsymbol{x} \sim q_X$ with their corresponding probability density $q_X(\boldsymbol{x})$. This allows for metropolized independence sampling as discussed in Section 3.3.6. In this way, we can build a Markov Chain that provably contains samples distributed according to the exact target density p for a model q_X that sufficiently well approximates p. The better the approximation, the smaller the integrated autocorrelation time for the Markov Chain will be and thus the fewer samples one needs to produce to estimate the expectation value up to the desired accuracy. Note that it is less involved to calculate the acceptance rate for a Markov Chain than its autocorrelation and these measures are linked [23]. This approach of obtaining the expectation value is called *Neural Monte Carlo Markov Chain* (NMCMC) [23, 48] method.

The model can also be used to estimate an expectation value through importance sampling. Again it is crucial to approximate p through q sufficiently well, to achieve reasonable accuracy. The measure here will be the effective sampling size ESS. The approach is called *Neural Importance Sampling* (NIS)

5.10.2 Change of Variables

Assume, we have access to a function that takes highly correlated variables that follow a complex distribution and maps them to simpler distribution. Then as noted by [5]:

$$\begin{aligned}\langle \mathcal{O} \rangle &= \int \mathcal{D}\phi \; \mathcal{O}(\phi) \frac{e^{-S(\phi)}}{\mathscr{Z}} \\ &= \int \mathcal{D}\phi \; \mathcal{O}(F(\phi)) |\det dF_\phi| \frac{e^{-S(F(\phi))}}{\mathscr{Z}} \\ &= \int \mathcal{D}\phi \; \mathcal{O}(F(\phi)) \frac{e^{-\tilde{S}(\phi)}}{\mathscr{Z}} \\ &= \int \mathcal{D}\phi \; \mathcal{O}(F(\phi)) p'(\phi).\end{aligned} \tag{5.52}$$

We have defined a new action

$$S'(\phi) = S(F(\phi)) - \log |\det dF_\phi|. \tag{5.53}$$

The Normalizing Flow model is trained to achieve this and thus can be used for this change of variables. In order to evaluate the expectation value, we can use for example an HMC to sample configurations from p'. Afterward, F^{-1} is applied, and the expectation value is estimated as described in Chapter 3. This approach has been used e.g. in [49]. Albeit we expect deviating behavior in comparison to the approach in Section 5.10.1 the numerical analysis in this thesis exclusively focuses on using the Normalizing Flow as a Sampler.

Chapter 6

Normalizing Flow Training

We have thus far introduced the Topological Quantum Rotor system and determined the topological susceptibility χ_t as an observable, whose expectation value is particularly difficult to estimate through local MCMC methods due to topological freezing. Moreover, an introduction to Normalizing Flow models has been given, and we presented a way to learn the density of the Quantum Rotor. It was discussed how to use the Normalizing Flow as a sampler for NMCMC and NIS to produce statistically unbiased expectation value estimates.

In Chapter 5, several aspects of the Normalizing Flow setup were discussed, which can be freely varied up to the constraints given alongside. As usual in Deep Learning problems, we need to find an architecture through trial and error, for which the entire Normalizing Flow setup performs as well as possible.

To that extent, there are several desirable aspects. Importantly, the model should be able to approximate the target density sufficiently well. As measures of convergence, we will analyze the metrics introduced in Chapter 3 and their reliability. The better the model approximates the target density, the less computationally expensive will it be to correct the model's bias and extract expectation values. On the other hand, the training itself is computationally expensive, so we look for an optimum, which minimizes the overall cost. Furthermore, this optimum of convergence should be reached as quickly as possible, ideally with a small model, which helps keep the computational cost at a minimum.

Another property that is crucial in determining the viability of the flow model as an alternative to MCMC methods is scalability. This entails checking how the convergence properties listed above change as a response to varying the coupling β and the system size D. Ultimately, we aim to compare the flow model to the HMC algorithm. As the latter is conjectured to scale exponentially with decreasing lattice spacing for $\tau \cdot D = const$ in the integrated autocorrelation time (see Chapter 4), we analyze the behavior of the flow mostly for this experiment. Additionally, we look at separate dependencies on the coupling and the system size to better understand the limitations of the flow model.

To ensure the validity of the implemented methods, we perform various consistency tests, which are presented in Appendix B.

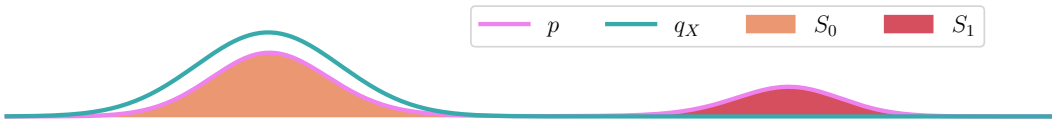


Figure 6.1: Bimodal target probability density p. S_0 is the larger sector compared to S_1 in the sense that $\mathbb{E}[\mathbb{1}_{S_0}] > \mathbb{E}[\mathbb{1}_{S_1}]$, where $\mathbb{1}$ is the indicator function. The model density q_X has collapsed onto the sector S_0, which contains the mode of p. Both q_X and p are shown as normalized functions.

6.1 Mode Collapse during the Training

To further determine the performance of a specific model architecture, we need to introduce the concept of *mode collapse*. This artifact turns out to be the most limiting factor for using Normalizing Flows with highly peaked densities such as the one of the Quantum Rotor.

Mode collapse describes the behavior of a generative model to place virtually all probability weight onto a subset of the target space. This subset is characterized by a comparatively large portion of the probability weight of the target distribution. In that sense, the model density q_X can be quite close to the target density p for this sector, but for other areas of the target space, the target density is dramatically underrepresented. This is schematically depicted in Figure 6.1.

We encounter mode collapse mainly for training the model with the Reverse KL divergence as a loss. As discussed in Chapter 5, this objective has a zero-forcing quality. Consider the situation in Figure 6.1. If during the training for a sampled point $\boldsymbol{x} \in S_1$, the current flow density $q_X(\boldsymbol{x})$ is far larger than the target $p(\boldsymbol{x})$, there will be a very large gradient update to mitigate this discrepancy and decrease q_X for a region close to, and containing, the sampled point. The more highly peaked the target density, the larger the fraction of points in sector S_1, which are overestimated initially. The stochastic gradient update step is an iterative process, which is not expected to precisely adapt to the target during a single step. Therefore, an overcorrection is likely to happen, and only very little probability weight will be placed on sector S_1 after several steps.

Once a sector of the target distribution is underrepresented, it is a slow process to rediscover it. This is because the Reverse KL divergence is estimated by Monte Carlo Sampling with a fixed number of sample configurations, which is equivalent to the batch size, at every update step. In the Reverse Kl training setup, the model itself serves as the sampler for the drawn configurations. Thus, if a sector is assigned comparatively little probability weight by the model, it is very unlikely that samples from this sector will contribute to the loss estimate.

Mode collapse is studied widely regarding generative models [50–53], but mainly with Generative Adversarial Networks (GAN). Solutions proposed include e.g. exchanging the loss [50]. This is unfortunately impossible since the Reverse KL divergence is needed

to be able to neglect the normalization of the target density, as discussed in Section 5.3. In [28] the mode-dropping susceptibility of the objective is well documented.

While Normalizing Flows, which are used as a sampler for NMCMC are guaranteed to be ergodic, one can speak of effective breaking of ergodicity [10], if a mode-collapsed generative model is used. This is because the density in underrepresented regions will typically be too low to be sampled for any reasonable number of samples.

In the training of the Normalizing Flow on the target density of the Quantum Rotor, mode collapse also manifests itself. In particular, the flow model concentrates disproportionately on the topological sectors $S_i := \{\boldsymbol{\phi} \in \mathbb{T}^D | Q_{disc}(x) = i\}$, which are characterized by smaller absolute values of i. This misbehavior becomes increasingly drastic for more highly peaked target densities until only the sector S_0 contains a practically relevant probability mass. Therefore, during training, inspecting the variance of the distribution of the topological charges $\mathrm{Var}[Q]$, of the configurations sampled by the model, gives a good indicator of mode collapse. For this thesis, the variance of sampled topological sectors is particularly useful, since we can compare the value with the exact susceptibility, but in general, the variance evolution along the training and a comparison against its value for less peaked target densities might still give a reasonable indication.

As the severity of mode collapse becomes a rather large issue, we will introduce the pathology using an exemplary training run, shown in Figure 6.2, before we go on to validate different architectures. As verified in Appendix B, the Normalizing Flow setup produces satisfying results for a regime with relatively uniform target density for relatively small coupling β and system size D. In Figure 6.2a, the flow is trained on target densities, differing by the used value of β. The larger β becomes, the more severely the model suffers from mode collapse in the sense that $\overline{Q^2}_{q_X}$ drops to very low values (compared to the theoretical value) during the training and recovers increasingly slowly for larger β. As shown in Figure 6.2b, this behavior is not recreated by increasing the number of lattice sites while keeping β fixed at a point, where for low D no mode collapse occurs. For the ensemble experiment of variable coupling, the distributions of the corresponding sampled topological charges $\overline{Q}_{q_X}$ along the training process are added in Figure 6.2e. For small coupling β, the distribution seems rather steady and is only slightly modified to match the target distribution. Naturally, the initial distribution of $\overline{Q}_{q_X}$ is dependent on the initialization of the model, i.e. the choice of parameters that the model is initialized with. In all our training runs, the initial weights are drawn randomly from appropriately sized uniform distributions [54–56]. For larger values of β, we can see a collapse of the distribution onto the sector S_0. Furthermore, the skewness of the distributions quantified by the third moment, which is expected to be zero, is added to this plot. Albeit there is no manifest reason for this, the model rather quickly settles into a virtually symmetric density q_X around the topological charge zero for all runs.

Mode collapse also distorts the metrics of convergence for NMCMC and NIS. Figure 6.2c contains estimates for the acceptance rate a and the integrated autocorrelation

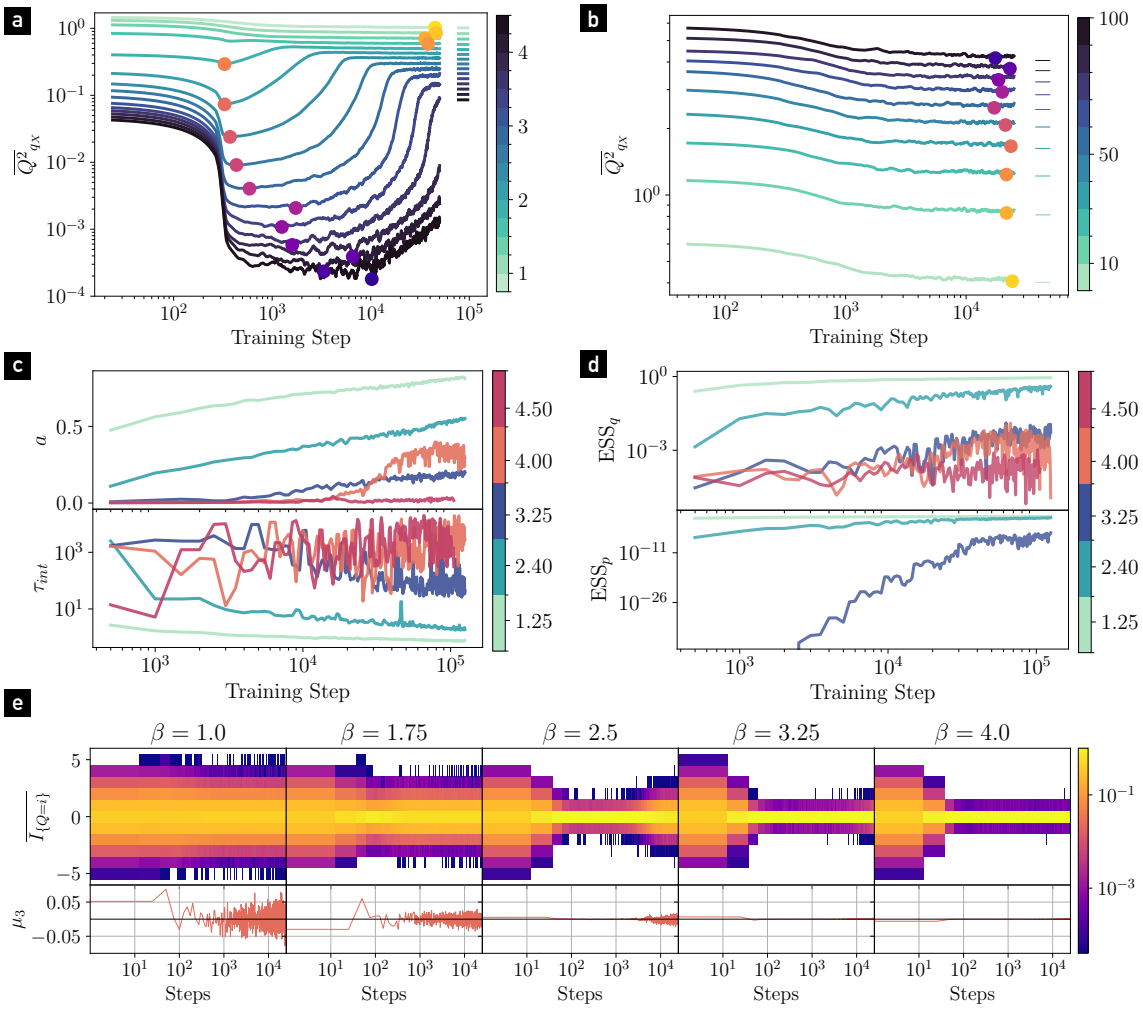


Figure 6.2: Training runs of an exemplary Normalizing Flow model. The Normalizing Flow consists of 5 coupling layers. The conditioners are MLPs with 3 hidden layers of size 100,125 and 150. The diffeomorphisms are NCPs with an expressiveness of $K = 8$ and a checkerboard-type split is used. The flow's base density is uniform. In plot (a), we depict an evaluation of $\overline{Q}_{q_X}$ along the model training. Different runs are set up with varying values of the coupling β, indicated by the color coding, whereas D is fixed to 25. In (b), the equivalent data is shown, however, different runs are set up with varying values of the system size D, indicated by the color coding, whereas β is fixed to 1.0. The theoretical values $T\chi_t$ are added on the right edge of each plot for comparison's sake. Additionally, the minima in the trajectories are marked. Figure (c) depicts the acceptance rate, as well as the integrated autocorrelation time $\tau_{int}^{Q^2}$ for NMCMC runs during the training. In (e) we show the effective sampling sizes ESS_p and ESS_p. In both (d) and (d) 250,000 configurations per run were used, and the color coding denotes the value of the coupling β. Figure (e) shows the evolution of topological charge distributions along the training. The upper row of plots visualizes the distributions of the topological charges of the configurations produced by the model along the training. The lower row shows the third moment μ_3 of said distributions. Each column contains data for a single training run with a single value of β. All scalar trajectories have been smoothed with a Savitzky–Golay [57] filter using a window size of 1% of the total number of logged values and a polynomial order of one.

time $\tau_{int}^{Q^2}$ for regular NMCMC runs during the training process. We plot multiple training runs with varying values of β. We can see that the metrics are only reliable up to a point of β. Afterward, both metrics indicate a better convergence than reached. This is because the metrics are calculated based on samples drawn from the model itself. So only sectors onto which the model has collapsed are taken into account, and the drastic underrepresentation in other sectors is ignored. In Figure 6.2d, the same plots are depicted for ESS_q (3.17) and ESS_p (3.19). Here, we can see the drawback of the ESS_q discussed in Chapter 3. Mode collapse leads to an unreliable effective sampling size estimate and, in addition, the estimate suffers from a very large variance. We can no longer reliably evaluate the model's performance. The root cause is the same as for the NMCMC metrics. In contrast, the ESS_p always reliably predicts the deviation of q_X from p. However, the values become extremely small for poorly converged models, so the metric is not shown for the largest coupling values. Therefore, a numerically stable calculation, which is discussed in Appendix C.2 is imperative. In both cases, the noise in the metric evolution can be reduced by increasing the number of configurations that are sampled by the flow model.

Unfortunately, it remains to be said that we have not introduced a metric, which can accurately predict the level of convergence in the general case. The variance of the loss, which meaningfulness was explained in Section 5.3, as well as the metrics for NMCMC and NIS, cannot reliably measure the approximation quality of the model when mode collapse has occurred. The only metric which is still useful in that case is the ESS_p. But it cannot be calculated if we don't have a way to produce configurations by sampling from the target density p.

To compare different model architectures regarding mode collapse, it is beneficial to define a metric that captures and quantifies the severity of the collapse. The spread of topological sectors S_i and therefore $\mathrm{Var}_{q_X}[Q]$ is, as discussed in this Section, a fitting measure. As the flow models reliably produce a symmetric distribution, we instead take $\overline{Q^2}_{q_X}$ for this purpose. The model initially samples from a density, which is close to the uniform one. This is an empirical and approximate observation and not an exact statement. We observe that $\overline{Q^2}_{q_X}$ thus starts larger than its target value. Mode collapse is characterized by a decrease in $\overline{Q^2}_{q_X}$ and thus the relative minimum

$$\min_n \left[\frac{\overline{Q^2}_{q_X,n}}{T\chi_t} \right] \tag{6.1}$$

$\overline{Q^2}_{q_X}$ reaches (relative to the exact susceptibility $T\chi_t$) along the training process. Here $\overline{Q^2}^{(iid)}_{q_X,n}$ denotes the estimate at step n.

Additionally, we quantify the number of training steps required to sample from a density q_X, where the spread in topological sectors is sufficiently close to the desired one. For

a tolerance $\varepsilon_{T\chi_t}$, we measure:

$$\min\left[\left\{n \in \{1,\ldots,N_{total}\} \middle| \frac{\left|\overline{Q^2}_{q_X,n} - T\chi_t\right|}{T\chi_t} < \varepsilon_{T\chi_t}\right\}\right]. \tag{6.2}$$

In this minimization, the training step index n runs the total number of training steps N_{total}. To ensure that we do not end up close to the target due to fluctuations as a result of the stochastic training process, we only consider a certain number of steps as a minimum if, for all later training steps, the required threshold is not violated anymore. This procedure is depicted in Figure 6.6a.

6.2 Scaling

An important factor of interest in simulating physical systems is the cost to run the algorithm. This cost depends on the accuracy of the estimates and the physical model parameters. As ultimately one would like to extrapolate the results to the continuum limit, the cost needs to be analyzed by varying the simulation parameters β and D. For this, the absolute cost at multiple parameter values is calculated and subsequently extrapolated.

In general, the Normalizing Flow approach consists of training and bias correction through NMCMC or NIS. Their respective costs are very different also depending on our definition of cost. One could e.g. quantify the computational cost through the number of $flops$, give the overall number of steps for each part of the algorithm or measure the elapsed time from start to finish. While the required flops are a crucial measure in their own right, the flow training can be parallelized by processing batches of configurations and additionally, training can be done on hardware, which is optimized for linear algebra operations. As a result, the flops do not necessarily estimate the viability of a method as opposed to the quite natural metric of the elapsed time.

In this section, we look at the scaling of the number of required training steps to reach a certain level of convergence as a measure to compare model architectures. Furthermore, we measure the elapsed time, which can be used to compare the Normalizing Flow approach to the HMC. For the following sections, when we analyze the scaling behavior, we make sure that we are either in a regime where mode collapse is negligible, or we use an alteration that circumvents mode collapse.

During the training of a Normalizing Flow model, we record the metrics for NMCMC and NIS in regular intervals by running their respective algorithms for 250,000 configurations. Note that to perform a more robust analysis, the frequency, in which metrics are recorded, is kept high at every tenth step. For the scaling analysis, we quantify the number of configurations or training steps, which are required to pass a certain threshold of fit quality determined by the recorded metrics. As the metric trajectories can be noisy due to the stochastic nature of the training, we smooth the curve first

by applying a Savitzky–Golay filter with a window size of 1% of the total number of logged values and a polynomial order of one, such that the trend is more prominent and random fluctuations do only contribute in a minor way to the result. We then consider the point in the training, after which no more values violate the threshold anymore. In this way, premature threshold crossings due to fluctuations are excluded. The mechanism for this criterion is sketched in 6.3a. Additionally, the finite resolution at which metrics are recorded, results in the artifact that, for low thresholds of convergence, the flow model very quickly in the training fits the target density well enough, such that the distances between different threshold crossings are not necessarily reliable. Therefore, we exclude threshold crossing, which happens early in the training (before the 50th training step).

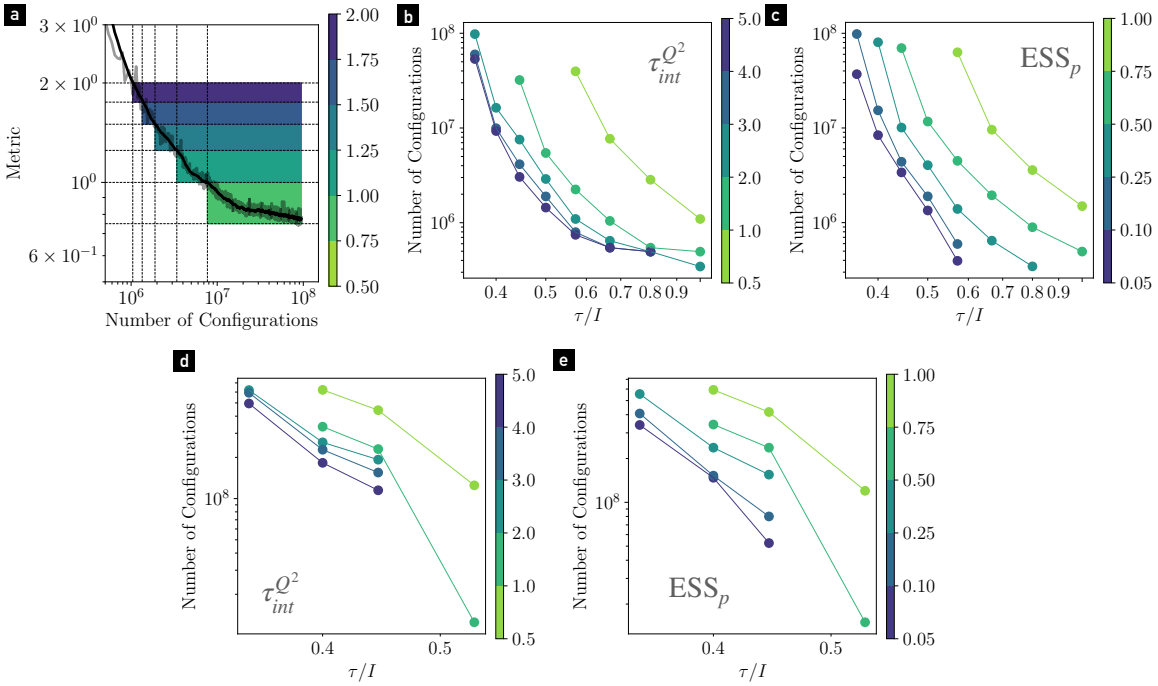


Figure 6.3: Depicts the scaling analysis for the required amount of training steps to reach a desired level of convergence. (a) This plot elaborates on the way, the required training steps are extracted. For a specific training run, we plot a given metric (e.g. $\tau_{int}^{Q^2}$) along the training process. The values are smoothed with a Savitzky-Golay filter. In (b) and (c) we plot for the CNN-type flow architecture several training runs on the CLP ($T = 4, I = 0.25$). During training, NMCMC/NIS runs are executed with the models at different training stages. With the procedure depicted in (a), we plot the number of required training steps, to reach a certain threshold in $\tau_{int}^{Q^2}$ during the NMCMC runs in (b) and ESS$_p$ during NIS runs in (c). Figures (d) and (e) contain the same plots but for the MLP-type architecture. Here (d) depicts the results for $\tau_{int}^{Q^2}$ and (e) for ESS$_p$. For all figures, we color code the thresholds and the values can be extracted from the color bar.

When measuring the elapsed time it takes to estimate the value of an observable using the Normalizing Flow approach, we now also take the bias correction step into account.

Note that we exclude the statistical error analysis for this experiment as it is typically of negligible cost.

The performance of NIS and NMCMC crucially depends on how well the density q_X has converged to p. In some sense, a badly converged sampler renders estimation in a reasonable amount of time almost impossible through the effective breaking of ergodicity. But generally, the number of configurations needed to perform an observable estimate within a certain accepted error decreases with improved approximation quality of the sampler. Another point of view is to shift the computational work required from the estimation to training the sampler. Furthermore, methods like NMCMC and NIS critically rely on ratios of probabilities, which for the Quantum Rotor are given as a normalized Boltzmann weight. The action is an extensive quantity, such that for increased system size D, even if the relative error of sampled configurations on the action $\frac{\Delta S}{S}$ does not increase, methods like NMCMC and NIS will perform worse and require a larger amount of configurations.

As the training routine implemented for this thesis includes many additional monitoring processes, we recover the minimum required time by extrapolating single training and bias correction steps. This means if the model is trained for N_{train} steps, which take a time of Δt_{train} each and evaluated using N_{eval} steps, which take a time of Δt_{eval}, the entire time cost is calculated as

$$C(N_{train}, N_{eval}; \Delta t_{train}, \Delta t_{eval}) = N_{train}\Delta t_{train} + N_{eval}\Delta t_{eval}. \tag{6.3}$$

For simplicity, the discussion is limited to NMCMC as the method to correct the bias. If we want to determine the minimum cost it takes to reach a relative error ε_{Q^2} for the expectation value estimate of Q^2, we need to find

$$C_{min}\left(\varepsilon_{Q^2}\right) = \min_{N_{train}, N_{eval}} \left\{ C \,\middle|\, \frac{\overline{\sigma}^{\left(\mathrm{NMCMC}\left(\mathrm{q_{X,N_{train}}}\right)\right)}_{\overline{Q^2}}}{\overline{Q^2}^{\left(\mathrm{NMCMC}\left(\mathrm{q_{X,N_{train}}}\right)\right)}} < \varepsilon_{Q^2} \right\}. \tag{6.4}$$

$\mathrm{NMCMC}\left(q_{X,N_{train}}\right)$ denotes the NMCMC method applied to the sampling distribution $q_{X,N_{train}}$ at step N_{train}. Practically, this optimization can be done through iterating N_{train} and N_{eval}. During training, we save the current model in intervals with increasing length. Specifically, for the i-th saved model M_i, we determine the saving step S_i by:

$$\frac{S_i - S_{i-1}}{i} = r. \tag{6.5}$$

In our runs $r = 0.75$. In this way, we create increasing waiting periods, such that early in the training the saving frequency is high, whereas it is tempered off in the later stage of the training. This saving schedule improves the scaling results as it provides the possibility to take very quickly converging models into account.

The NMCMC analysis, which is run on the saved models for several numbers of drawn configurations is repeated three times and the relative errors are averaged, such that

some stochastic variations are removed and a more pronounced optimum is found. In Figure 6.4a, we show this experiment for a single exemplary run, i.e. a single parameter set (β, D). Figure 6.4b combines multiple such parameter combinations for a CLP. For every parameter combination the minimum is recorded and the resulting runtime and minimum amount of training steps are plotted in Figure 6.4c.

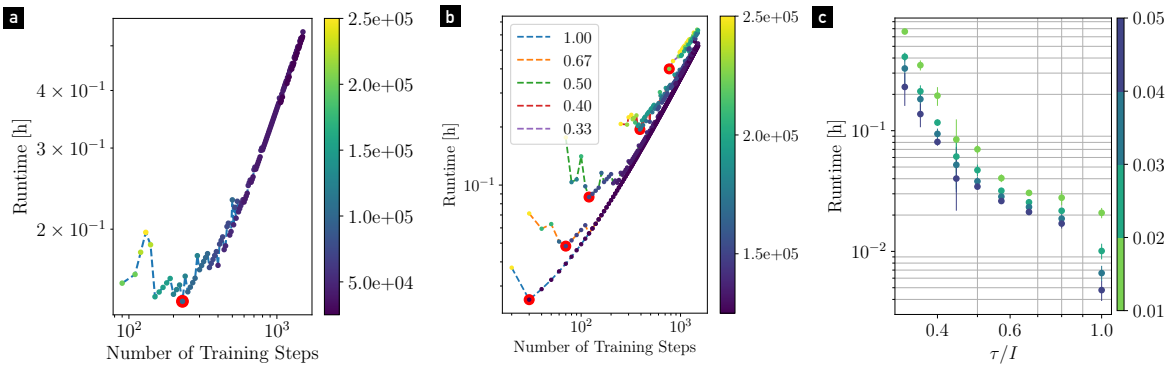


Figure 6.4: Time cost minimization for the Normalizing Flow set up. (a) For a single training run models are saved in regular small intervals, which are being used to run NMCMC for multiple numbers of produced configurations. In this Figure, the number of NMCMC configurations is color coded and the values can be extracted from the color bars. Each number of configurations is run three times and the error for this setup is averaged across the runs. For every training step, which corresponds to a saved model, we plot the minimum time cost it takes in total to achieve a relative error on $\overline{Q^2}^{(\mathrm{NMCMC})}$ of less than 2%. The minimum total cost along the training process is marked with a red circle. (b) Several runs for different β, D on a CLP with $T = 4, I = 0.25$ are plotted to be able to compare the evolution of the optimal cost. The legend specifies τ/I for the individual runs. In (c) the optima (red spots) for several analyzed parameter configurations β, D and attached relative error thresholds are plotted. The CNN-type architecture is used, and the results are averaged over 5 independent runs with differing random seeds. The error thresholds are color-coded and can be extracted from the color bar.

The multiplicative time cost factors Δt_{train} and Δt_{eval} per step are, of course, dependent on the used hardware, since e.g. the training can be drastically accelerated using a GPU. The hardware the elapsed times have been measured on are given in appendix D.

6.3 Hyperparameter Search — Determining the best performing Architecture

Any parameter that specifies the Machine Learning architecture and the training process is called a *hyperparameter*. As is often the case, in the Normalizing Flow setup for the Quantum Rotor, there is an infinite amount of variability in the hyperparameters. As long as the design limitations introduced in Chapter 5 are met, one is left to guess

and search for the best-performing choice. Albeit several principles have been explored in the literature (see e.g. [9, 45]) – especially regarding well-performing conditioner architectures – usually, these are tailored to distinct datasets and problems.

As a metric of performance regarding the hyperparameter search, we analyze the mode collapse susceptibility first. It is less computationally expensive to quantify than the scaling behavior because no bias correction steps need to be run. Additionally, a model that has suffered mode collapse to some degree will either produce unreliable NIS/NMCMC metrics (see Section 6.1) or scale very poorly. The resulting best-performing architectures regarding mode collapse are subsequently compared for scaling behavior.

Generally, the model needs to be expressive enough to accurately approximate the target density. It is not possible to reliably distinguish, whether this is given when looking at an isolated training run. During the analysis, one can only move from large networks and small β, D to smaller networks and larger β, D and check, whether the network size is the limiting factor. Optimally, the model should be as small as possible. However, usually, network sizes are strictly controlled if there is a possibility to overfit. In the Reverse KL setup, which is the only setup, which enables us to exactly learn the target density and is part of all of our approaches, overfitting is no issue.

It becomes apparent that indeed the model performance drastically depends on the choice of architecture. There are design choices, which are necessary for success. For example, a network should be able to transform every input variable at least once. Other design choices are less obvious and there is no other possibility than to search for the best-performing configuration. Additionally, the performance is starkly influenced by hyperparameters that control the training. To this set belong the learning rate, and its evolution along the training process or the batch size, which were briefly introduced in 5. Not every aspect is varied within the subsequent sections. As base distribution to the flow, we have only tested the uniform density.

Since there are many architectures to probe, an exhaustive search is hardly possible. We rather vary a single parameter and analyze the difference in performance. This is of course not strict evidence for the relation that is extracted since another area of the high dimensional space of hyperparameters could exhibit a different relation concerning the varied parameter. There are modern frameworks for hyperparameter optimization using e.g. Bayesian statistics [58]. They have not been utilized as a part of this thesis.

6.3.1 Diffeomorphisms

The diffeomorphisms presented in Section 5.5.1 show distinctly different behavior. For comparison, we train a model with equal specifications besides the used diffeomorphism on multiple increasingly peaked densities. Only flows with a single kind of coupling layer have been tested. The results are shown in Figure 6.6. As also pointed out by [6], the RQS transformation produces inferior results compared to NCP and Moebius when the target density is highly peaked. Furthermore, the NCP outperforms the Moebius

transformation. On top of the plain diffeomorphisms, the plot shows an experiment with an explicitly residual NCP diffeomorphism as introduced in Section 6.3.2. The residual model is slightly better able to avoid mode collapse initially and then also crosses the threshold of relative deviation more quickly than all non-residual counterparts. The scheduling procedure is described in Appendix C.

Due to the superior performance, mostly NCP layers have been used when varying the other parameters. Whether the NCP was adjusted with a Taylor approximation close to the interval boundaries or whether a simple modulo operation was added to map back to the interval has no significant influence on the performance. We found that an exponential function works best at constraining the conditioner input to the NCP transformation. For a comprehension of constraining maps regarding the diffeomorphism inputs, consult appendix C.1.

The expressiveness is kept low, i.e. $K \in \{2, 3\}$. This leads to better results since the conditioner needs to output fewer parameters and can learn more quickly, while the expressiveness of the transformation seems to suffice for the Quantum Rotor target density.

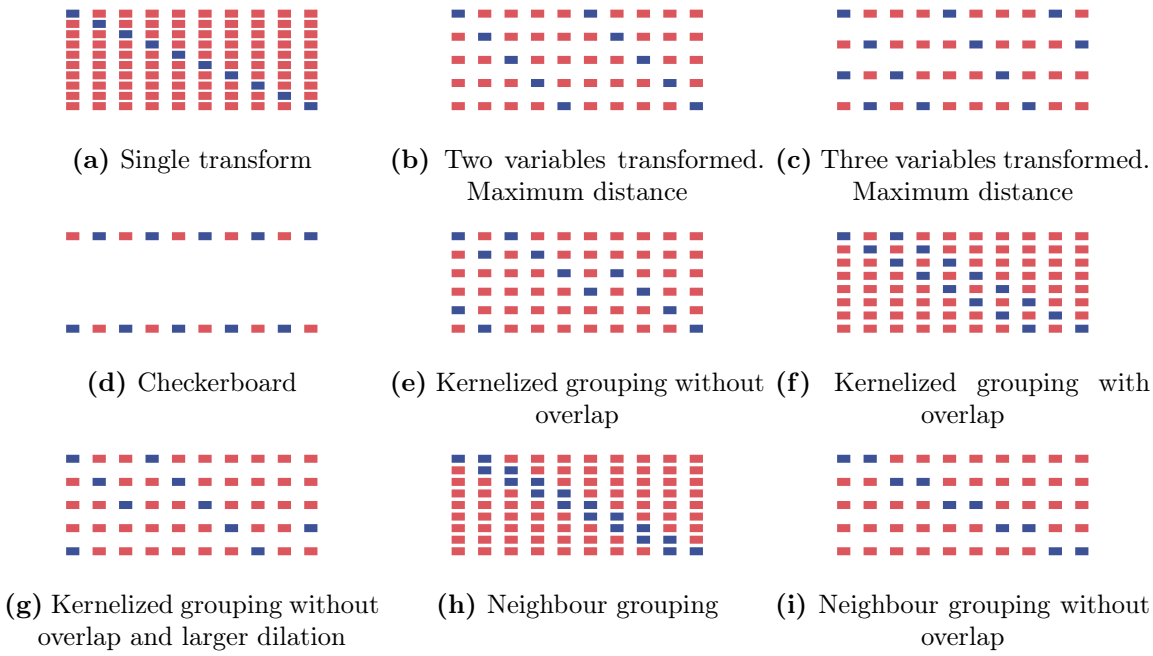
6.3.2 Layers

In Chapter 5, we have introduced two types of layers. The autoregressive layer, as well as the coupling layer. Up to different ways of implementing the conditioner, the autoregressive architecture is comparatively fixed. Network parameters can be shared across conditioners for example using a recurrent neural network (RNN) and the networks can even be boiled down to a single architecture using masking, which asserts that the autoregressive structure is preserved by the network. Advanced autoregressive conditioner architectures have not been implemented for this thesis, as using a separate MLP decoder for each transformation has produced inferior results to the coupling layer structure. While mode collapse seems to be less of a problem for autoregressive models, they do not converge well to the target density and so considering our preliminary results, we have only used coupling layers.

As described in Chapter 5 concerning coupling layers, there are many choices regarding the set of variables that are fed into the conditioner and the set of variables that are transformed. This is because the only condition is that the sets are disjoint. Leaving variables out of both sets has not been investigated. Instead, all untransformed variables are given to the conditioner to maximize the available information. All tested splits are visualized in Figure 6.5. We found that the recurring theme in machine learning that deeper architectures perform better is also applicable to the number of coupling layers. For the experiments with a checkerboard split, we use 16 coupling layers. In any case, we use at least as many layers as required to have transformed every input variable at least once.

Note that only the MLP conditioner architecture was analyzed for different splits. Combined transformation of neighbors as depicted in Figure 6.5 performs very poorly.

This is to be expected since next-neighbor relations should influence the transformation the most and so the conditioner should at least get the next neighbors of each variable that ought to be transformed. Looking at kernelized splits, it is even better to leave out two variables between transformed ones. Additionally, across all split types we see that the fewer variables are transformed per layer, the less susceptible a flow is to mode collapse. This is evident from the plot in Figure 6.6b, where an MLP architecture that outputs parameters for a single variable is compared to a checkerboard split. An obvious disadvantage of this result is that this increases the number of required layers with D quite unfavorably. Ultimately an excessive amount of coupling layers became numerically unstable in our experiments. Note that, as will be discussed in the next



(a) Single transform

(b) Two variables transformed. Maximum distance

(c) Three variables transformed. Maximum distance

(d) Checkerboard

(e) Kernelized grouping without overlap

(f) Kernelized grouping with overlap

(g) Kernelized grouping without overlap and larger dilation

(h) Neighbour grouping

(i) Neighbour grouping without overlap

Figure 6.5: Tested Coupling layer splits. Each image depicts a unique manner of splitting the lattice into variables that are fed into the conditioner and variables that are transformed. Each row represents a layer. The red blocks denote variables that are fed into the conditioner. Blue blocks denote transformed variables. Each splitting depicts as many layers as are needed to transform each variable at least once.

section, this behavior might to a large extent be a result of the nets' limited capability to reliably output parameters. The influence of the used splits is also discussed in [45], for the sake of constructing an equivariant flow layer.

If a CNN-type conditioner is used without a global network stacked on top like an MLP, we can in principle still use any type of split. However, due to the local nature of the CNN, which we optimally would like to retain, a checkerboard makes the most sense. And thus the architecture is designed in a way, which requires that the sets I_A and I_P are equal in size. This implies that we can only transform an even number of lattice points. This is of course not a limitation as we can still analyze arbitrarily large even numbers of lattice points.

6.3.3 Conditioner Architecture

Another determining factor in the Normalizing Flow setup is the choice of conditioner. To this end, we analyze the MLP and CNN listed in Section 5.7. As the nonlinearity σ,

$$Mish(x) = x \tanh(\text{Softplus}(x)), \tag{6.6}$$

introduced in [59] was used. The Softplus function is given in Equation C.2.

For the densely connected MLP, the hidden layer sizes as well as the number of hidden layers greatly influence the performance of the flow model. We see accelerated learning for up to three hidden layers, while afterward the performance plateaus. The number of neurons per hidden layer also had a drastic effect on performance. Larger MLPs tend to escape mode collapse quicker and converge faster. The size of the layers should also be adapted to the number of variables, which are given as input to the conditioner and the number of parameters produced as input to the transform. We cannot reproduce the claim of [9] that a single hidden layer of size $|\Lambda|$ produces a sufficiently expressive network.

We claimed in Section 6.3.2 that the network size is a limiting factor in the number of variables that can be transformed within a coupling layer. Interestingly, we find that if a separate conditioner is used for every variable, the performance almost reaches the same level as if the coupling layer only transformed a single variable in the sense that it almost equally quickly escapes mode collapse. The left difference in performance is most likely due to the simplified training if the variables are transformed one after another and thereby can exert influence on later transformation parameters.

The mode collapse susceptibility of the CNN and the MLP are set side by side in Figure 6.6b. The CNN performance is vastly better than the checkerboard MLP. It does perform worse than the MLP, which transforms only single variables, however, the CNN network is built with many fewer trainable parameters.

6.3.4 Training Specifications

The Normalizing Flow training is done in 32-bit precision. When the flow is used as a sampler for bias correction this can lead to sampling *NaNs* and infinities, especially since often the importance weight w or its inverse is required. To mitigate this issue, the model weights are converted to 64-bit precision for the sampling process.

As in any Machine Learning setup, the training depends on the optimization choices. Throughout all experiments, the optimizer *Adam* [29] is used. Additionally, the base learning rates η have been scheduled in a manner that reduces them by a factor γ_η if the estimated Reverse KL divergence plateaus. This reduction was frozen for 5000 training steps after an alteration occurred. The minimum allowed learning rate was set to 10^{-5}. Another promising approach would be to dynamically adapt the learning rate to control the fluctuations of $\overline{Q^2}_{qX}$ around a linear trend. The smaller the learning rate, the smaller the change in network parameters can potentially be. And since the network

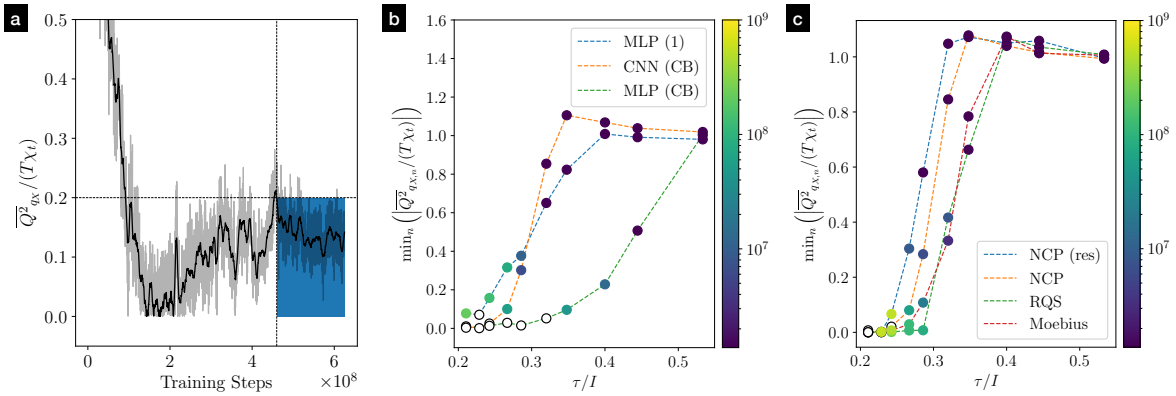


Figure 6.6: Mode Collapse Susceptibility for different Normalizing Flow architectures. (a) For an exemplary single training run. To quantify the mode collapse susceptibility, we plot $\frac{|\overline{Q^2}_{q_X} - T\chi_t|}{T\chi_t}$, which is smoothed with a Savitzky-Golay filter of window size of 1% of total steps and polynomial order one. To determine the training step at which the model samples configurations, for which this relative deviation is lower than an arbitrary threshold (here we depict 0.2), we take the point after which the threshold is not violated anymore in the course of the rest of the training. (b) The mode collapse susceptibility analysis is done for different conditioner architectures. We compare an MLP architecture with a split that transforms a single variable (MLP(1)) with another MLP conditioner, where a checkerboard split is used. Both MLPs are constructed with three hidden layers with 100, 125 and 150 neurons respectively. Additionally, a CNN conditioner (with three hidden layers of size 16, 32 and 32) in conjunction with a checkerboard split is shown. As diffeomorphism, NCP is used with $K = 3$. For the checkerboard splits, 16 coupling layers were used. The MLP(1) run, uses as many coupling layers as required to transform every input variable exactly once. So the number of layers is equal to the system size $D = |\Lambda|$. (c) We compare the mode collapse susceptibility for the diffeomorphisms introduced in Section 5.5.1. NCP(res) denotes a residual diffeomorphism, where the NCP is used as the base. We used an exponential manual scheduling reduction rate $\gamma_\rho = 0.99$ and a patience of 100 steps, during which no reduction takes place. The flow architecture is equal to the CNN flow in (b). For both (b) and (c) the threshold from discussion (a) is set to 0.5 and the number of resulting configurations is color coded and can be extracted from the color bar. The lattice spacing value indicates the point on the CLP ($T = 4, I = 0.25$).

is a continuous function, the fluctuations must also decrease, and we potentially end up with a more stable training process.

The magnitude of the base learning rate does not influence mode collapse. However, especially for a highly peaked target distribution, at the end of the training a very small learning rate is required to converge properly. At the beginning of the training, a higher base learning rate is beneficial to quickly overcome mode collapse.

Gradient Clipping was introduced to stabilize training for highly peaked target distributions. This means rescaling the gradient, such that its $\mathcal{L}^2$ norm is smaller than a given value. This was set to 1.0. Clipping all gradient components individually had no beneficial effect.

The batch size is another crucial parameter. As described in Section 6.1 we need sufficiently many samples to estimate the KL divergence accurately. It is possible to accumulate batches, so the effective batch size can theoretically become arbitrarily large. In practice, there is a trade-off between run time and effective batch size, because the batches are processed sequentially. There is also a point of saturation, after which increasing the batch size will not accelerate the training process, since the KL divergence is estimated sufficiently well already. For an increasingly peaked target distribution and a larger lattice, a larger batch size results in faster convergence. It does however not stop the initial mode collapse. For the listed runs in this chapter, a batch size of 10,000 is used, when not explicitly stated otherwise.

6.3.5 Final Model Specifications and Scaling Behavior

As a result of the hyperparameter search, for further analysis, we use a CNN-type architecture, which specifications are listed in Table 6.1. We do not apply the residual diffeomorphism, as additional scheduling would have to be done, which would further complicate the qualitative analysis. For the MLP, the single transforms variant is not used, because of the aforementioned numerical instability. Instead, regarding scaling, we compare the scaling of the CNN-type model with the checkerboard-type MLP, whose specifications are listed in Table 6.1 as well.

For both models, we do the scaling analysis outlined in Section 6.2 and the results are given in Figure 6.3b and 6.3c for the CNN-type and in Figures 6.3d and 6.3e for the MLP-type. For each type, the scaling behavior is extracted for the integrated autocorrelation $\tau_{int}^{Q^2}$ and the forward effective sampling size ESS_p, which provide the most reliable metrics. A regime for the coupling β and the system size D was chosen, for which mode collapse can be largely neglected. It is evident that the CNN vastly outperforms the MLP-type architecture. For both models, the same metric thresholds are used. Thus comparing the plots, we see that the MLP-type model requires more training steps to reach the same threshold for $\tau_{int}^{Q^2}$, as well as for ESS_p. Through the log-log scale of the plots, it also becomes apparent that the scaling behavior is over-exponential. The shown metric is also used in [9] for the ϕ^4 theory. In contrast to their result, we cannot reproduce exponential scaling.

	CNN-type	**MLP-type**
Diffeomorphism	NCP	NCP
Expressiveness	2	2
Number of Coupling Layers	16	16
Conditioner Architecture	CNN	MLP
Number of Hidden Layers in Conditioner	3	3
Hidden Layer sizes	16,32,32	100,125,150
Nonlinearity	Mish	Mish
Added Batchnorm before Nonlinearity	Yes	No
Split Type	checkerboard	checkerboard

Table 6.1: Normalizing Flow model architectures used for the final analysis

The cost minimization analysis is done only for the CNN-type model. The resulting runtimes are plotted in Figure 6.4c for relative error thresholds $\varepsilon_{Q^2} \in \{0.05, 0.04, 0.03, 0.02\}$. We see over-exponential scaling behavior, which is to be expected, considering the analyzed scaling of $\tau_{int}^{Q^2}$.

6.4 Circumventing Mode Collapse

We have determined a model architecture within the choices we have analyzed that performs as well as possible regarding mode collapse. To truly pose as an alternative to the HMC method, we do however need a way to circumvent the problem of mode collapse entirely. This is because once a model has collapsed, the amount of time it takes for the architecture to rediscover lost sectors is so large that the algorithm becomes infeasible.

6.4.1 Scheduling of β

One alteration to previously specified training behavior we would like to explore is to slowly move from a rather easy, flat target density p to a highly peaked one during the training process. We thereby eliminate the drastic overcompensation of the model in the initial stage of the training and rather slowly nudge its density q_X towards the desired one.

For local conditioners like the CNN, we might for this purpose vary the coupling as well as the system size. As we cannot change D for a fully connected architecture in a straightforward manner, only β was modified. The idea is visualized in Figure 6.7a. While one point of view is the mechanism of gradually bringing the flow distribution close to the final target distribution without introducing large model parameter

updates. Another point of view can be made clear if we decompose the reverse KL divergence like in Equation 5.6. It becomes apparent then that by decreasing β, the entropy term of the objective gains in relative weight. We could equivalently keep β fixed, introduce a factor $\gamma_\beta > 1$ in the entropy term, such that

$$\mathcal{F}_{q_X} = -\gamma_\beta \mathbb{H}[q_X] + \mathbb{E}_{X\sim q_X}[S(X)] \tag{6.7}$$

and step by step reduce it until $\gamma_\beta = 1$. As the entropy is maximized for a uniform density, this view is equivalent to starting with a small value for β and then incrementally increasing it.

Scheduling β can in principle be done arbitrarily. However, it is important to not advance too quickly to stay clear of mode-dropping. As we have already identified that the variance in sampled topological sectors gives insight into the state of mode collapse, we can adapt the scheduling dependent on the linear development within a window of size N_{window} of recently logged values of $\overline{Q^2}_{q_X}$. Given the slope m_{Q^2} of its time series, an absolute change rate η_β and factor γ_{Q^2}, we apply

$$\beta \leftarrow \beta + \eta_\beta e^{-\gamma_{Q^2} m_{Q^2}}. \tag{6.8}$$

Furthermore, a maximum accepted linear change rate in $\overline{Q^2}_{q_X}$ is set. If the change exceeds this limit, no change in β is made. After every alteration, β is frozen for a given amount of steps to give the model the chance to adapt to the change in target density. This period is called *patience*.

For a target β_{target} and an initial value $\beta_{initial}$, the scheduling process ought to traverse $\Delta\beta = (\beta_{target} - \beta_{initial})$. For a patience of $N_{patience}$, neglecting the exponential damping (i.e. setting $\gamma_{Q^2} = 0$), we thus require a minimum of

$$\frac{\Delta\beta}{\eta_\beta} N_{patience} \tag{6.9}$$

training steps to reach the ultimate target density. This scales linearly in β and gives a lower bound on the scaling behavior for the β-scheduling experiment.

Generally, the workable rate of change in β - i.e. without mode collapse - is dependent on the flexibility of the model as well as hyperparameters like training step size, batch size and gradient clipping. And also very importantly, on the behavior of the target distribution under varied β. Note that during β-scheduling, the scheduling of the base learning rate η is paused.

With this experiment, we achieve a drastic improvement in the mode collapse susceptibility metric (6.1). This is shown in Figure 6.7b. It proves to be very beneficial to apply a local conditioner architecture rather than a global one. For an equal scheduling setup and parameter combination (β, D), the MLP-type model suffers from mode-dropping to a greater extent than the CNN-type model. For the CNN architecture, we also compare different scheduling step sizes η_β. Here, a smaller rate of change improves the training slightly.

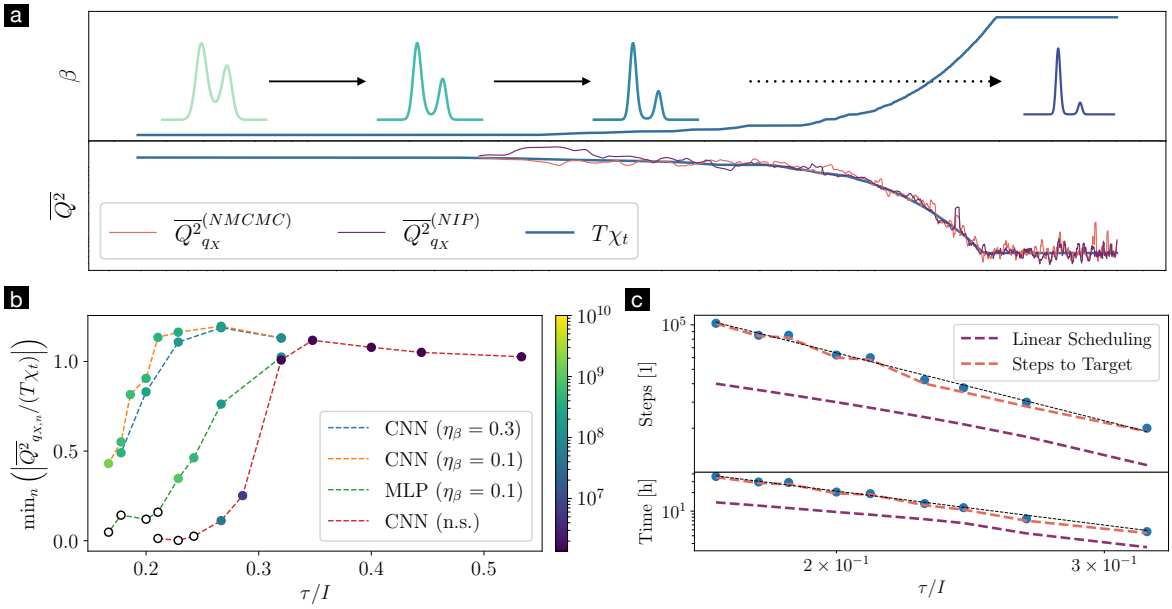


Figure 6.7: Introduction to the method of β scheduling and depiction of the results obtained. (a) An exemplary scheduling run is plotted. In the upper part, we show the evolution of β along the training. Toy densities in 1D underline the concept of moving from a less starkly peaked initial target to a highly peaked target. The lower part shows the according evolution of the topological susceptibility χ_t, as well as the results of NMCMC and NIS runs, along the training process. In (b) the results of the mode collapse analysis for the β scheduling method are shown. As a baseline, the CNN-type runs without scheduling (CNN (n.s.)) are added. We further decrease the lattice spacing for the scheduled experiments to probe the limit of the approach. The CNN-type model is trained with scheduling for two absolute change rates η_β. Moreover, concerning the scheduling, a maximum allowed change rate of 0.05 over a window of $N_{window} = 1000$ training steps, is used. The exponential damping is applied with $\gamma_{Q^2} = 50$. Between scheduling changes, the value for β is frozen for $N_{patience} = 1000$ steps. Initially β is set to $\beta_{initial} = 2.0$. For further comparison, the scheduling experiment is repeated for the MLP-type model. In Figure (c) the cost scaling outlined in Section 6.2 is analyzed for $\varepsilon_{Q^2} = 5\%$ using the CNN-type architecture. The training specifications are equal to the CNN ($\eta_\beta = 0.1$) run from (b). We additionally plot the minimum required amount of steps from Equation 6.9 (Linear Scheduling) and the required amount of steps to reach the target density due to the exponential damping and maximum allowed change rate (Steps to Target).

Regarding the scaling analysis of the experiment, a modification to the method introduced in Section 6.2 needs to be considered. Scheduling the target density, in principle provides the possibility to extract the entire evolution of expectation values while increasing β. In this way, one might only consider the extra work needed to increase β further in between two measured parameter points. Instead, we have chosen to quantify the entire work needed to reach the target density for each estimated parameter tuple (β, D).

To achieve this we repeat the analysis outlined in Section 6.2, with the alteration that the cost optimization (6.4) is started only once the ultimate target density is reached in the scheduling process. The result is shown in Figure 6.7c. A quite interesting observation is that essentially the number of steps or amount of time required to reach a certain error threshold is mostly dictated by the scheduling. We see that once the target is reached, the desired relative error ε_{Q^2} can be reached with NMCMC, and the elapsed time for this bias correction hardly has a negligible influence on the total runtime.

We are successfully able to reduce mode collapse and achieve a scaling behavior, which makes further extending the regime, which can be sampled for the Quantum Rotor, possible. Regardless, we see some collapsing behavior for small τ/I. This can likely be overcome by more appropriately scheduling β in an adaptive approach, where step size and patience are dependent on the current value of β. We can in general use a larger β step size and learning rate for smaller absolute values of β with a smaller batch size. While the opposite for all hyperparameters is true when β is large. This is because the target density is more peaked and thus the estimation of the KL divergence needs to be more precise. An adaptive approach will likely be able to improve cost scaling and possibly mitigate mode collapse to an extent, for which it is negligible.

6.4.2 Bidirectional Training

Another way to mitigate mode collapse is to train the flow bidirectionally. While so far we have only used the Reverse KL divergence, we will now mix in the Forward KL divergence. Through the work done in Chapter 5, we can train the Normalizing Flow model bidirectionally regardless of the used diffeomorphism.

It is not possible to train on the actual target density with solely the Forward KL divergence. As for this, we would need samples from p, which in our case can be produced, but are in the general case unavailable. What is however possible, is to produce samples from all topological sectors, with nearly correct distribution for each of these sectors individually. For this, we can initialize an HMC with a classical solution to the Quantum Rotor, which belongs to a given sector.

They can be found using the principle of least action:

$$\begin{aligned}\delta S &= \frac{I}{2}\int_0^T 2\dot{\phi}(t)\,\frac{d}{dt}\delta\phi\,dt \\ &= -I\int_0^T \ddot{\phi}(t)\delta\phi dt \\ &\overset{!}{=} 0.\end{aligned} \tag{6.10}$$

The periodic boundary conditions were used in this calculation. Equation 6.10 is solved by $\ddot{\phi}(t) = 0$ and thus $\phi(t) = \mod(at+b)_{[0,2\pi)}$ with $a, b \in \mathbb{R}$. Due to periodic boundary conditions on a finite lattice, $a = \frac{k2\pi}{T}$ with $k \in \{-D, \dots, D\}$, and b can be restricted to the interval $[0, 2\pi)$ as the Rotor revolves on a circle. So the classical solutions are linear functions with a constant angular velocity. When Q_{disc} is evaluated, since the shortest possible connection between lattice points is considered, through the modulo operation the topological charges are reduced to $k \in \{-\lfloor\frac{D}{2}\rfloor, \dots, \lfloor\frac{D}{2}\rfloor\}$.

Since the HMC relies on local update steps, after the initialization in a classical solution, it will wander around the sector. Once a proposed configuration does not belong to the sector, it is either rejected and subsequent samples are drawn until another configuration fits the sector again, or the HMC is just re-initialized in a random classical solution corresponding to the same topological charge. The configurations from different sectors are mixed according to an arbitrary predefined distribution, which would optimally be close to the target density's distribution of topological charges. The latter is unfortunately in general unknown. So we end up with different target densities for the two directions of training, which is visualized in Figure 6.8a.

In [10] samples drawn from p have been used to train the flow. The method of letting a local method like the HMC explore tails if the target density, which samples could then be used in the Forward KL training, is also suggested in [9].

Specifically, the idea is to train on the loss-mixture

$$L = \alpha\mathbb{KL}(q_X||p) + (1-\alpha)\mathbb{KL}(p||q_X). \tag{6.11}$$

$\nabla_{\boldsymbol{\theta}} L$ can then be estimated using (5.11) and (5.12), where the forward KL divergence is estimated on samples drawn from the created set of configurations $\{\boldsymbol{\phi}_n\}_{n=1}^{n=N}$.

The Forward KL loss is calculated for configurations, which are not drawn from p, and so ultimately for any $0 < \alpha < 1$, we cannot converge to the exact target density. Thus, α needs to be annealed and must reach $\alpha = 1$, after which point we only train using the reverse KL divergence. The scheduling is done equivalently to the β scheduling introduced in Section 6.4.1. The experiment evolution for α and the state of the flow is plotted in Figure 6.8b for an exemplary run.

The tests, which are done to ensure that the Forward KL training yields correct results are outlined in Appendix B.

Ultimately, we have not been able to train a flow model in this experimental setup,

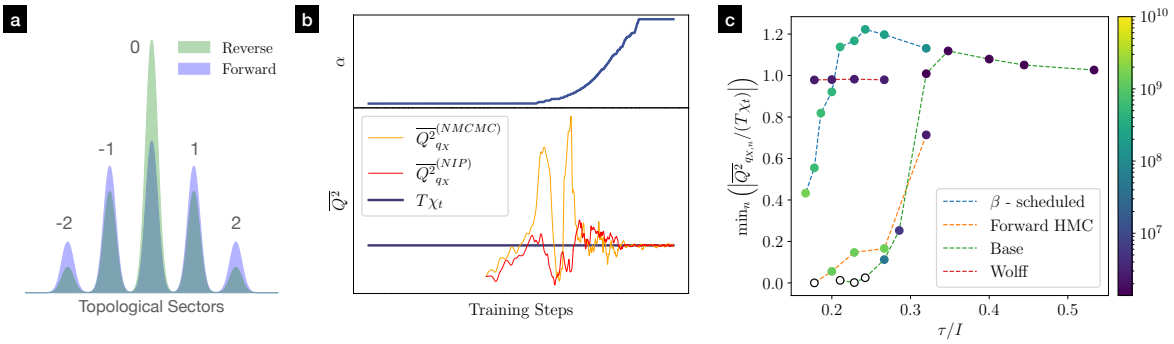


Figure 6.8: Depiction and Example for preventing mode collapse using a combination of Forward and Reverse KL training. (a) Illustrates the choice of distribution for the Forward KL part. Optimally we choose a distribution across topological sectors as close as possible to the one of the target density. In any case, we want to cover all sectors with it. (b) Exemplary experiment run. We schedule α (upper plot) in such a way that we slowly move to a purely reverse training, optimally without introducing large gradients, which might lead to mode collapse. (c) Mode collapse susceptibility result for the CNN type model. We include the baseline CNN, as well as the β-scheduled run for comparison. Here for scheduling α, we use $\eta_\alpha = 0.01$,$\gamma_{Q^2} = 50$ a maximum allowed change rate of 0.008 for $\overline{Q^2}_{q_X}$ over the course of $N_{window} = 1000$ with a patience of $N_{patience} = 1000$. Initially, we set $\alpha = 0.5$. In this Figure, we also add the mode collapse susceptibility for pure Forward KL training with datasets created with the Wolff algorithm (Wolff). The dataset size is $5 \cdot 10^5$, where every third configuration of a Markov Chain is included, which is created by the Wolff method. 25000 burn-in steps are excluded at the beginning of the chain.

such that the transition to reverse training was possible without suffering mode collapse. A possible reason for this might be that the dataset which was constructed with HMC and used during the forward-directed training did not sufficiently closely capture the true target density. This result is shown in Figure 6.8c. The distribution of configurations over topological sectors per batch for the Forward KL estimate is given in Table 6.2. Note that we used far fewer topological sectors than are theoretically possible, given the system sizes in Figure 6.8c. This relies on the observation that the sampled distributions for smaller system sizes show that sectors characterized by larger topological charges are negligible, and this probability mass distribution inequality is amplified for even larger system sizes.

Topological Sector	-3	-2	-1	0	1	2	3
Relative Weight	0.05	0.1	0.2	0.3	0.2	0.1	0.05

Table 6.2: Distribution of topological sectors for batches used in the estimation of the Forward KL divergence.

Figure 6.8c also includes the mode collapse susceptibility for a run, where the mode was trained purely on the Forward KL divergence with a dataset created by the Wolff cluster method. We see no sign of mode collapse. This underlines, that the mode collapse problem is restricted to the Reverse KL divergence.

The method of combining both training directions produces inferior results as we are so far not able to circumvent mode collapse in a way, which is possible with β scheduling described in Section 6.4.1. An additional drawback is that we introduce more hyperparameters such as the distribution of the topological charges.

6.5 Comparison to HMC

After the analysis in this chapter, we can now compare the performance of the Normalizing Flow setup with the results for the HMC method presented in Chapter 4. The runtime of estimating Q^2 and therefore the topological susceptibility χ_t for a given relative error is put side by side for the two approaches.

We combine the runtime results from previous sections in Figure 6.9. For a relative error $\varepsilon_{Q^2} \in \{0.03, 0.04, 0.05\}$, we plot the results of the HMC, the unaltered Normalizing Flow training as well as the β-scheduled training for the CNN-type model. The bias correction for the Normalizing Flow models is done using the NMCMC method. The results for the HMC (see Chapter 4) and for the unaltered flow training (see Section 6.3.5) are unchanged. For the β-scheduling, we plot the runtime required to reach the target density since the additional bias correction is negligible. Here, as well as for the HMC, we additionally add an exponential fit, which is a very good fit at least in the probed regime.

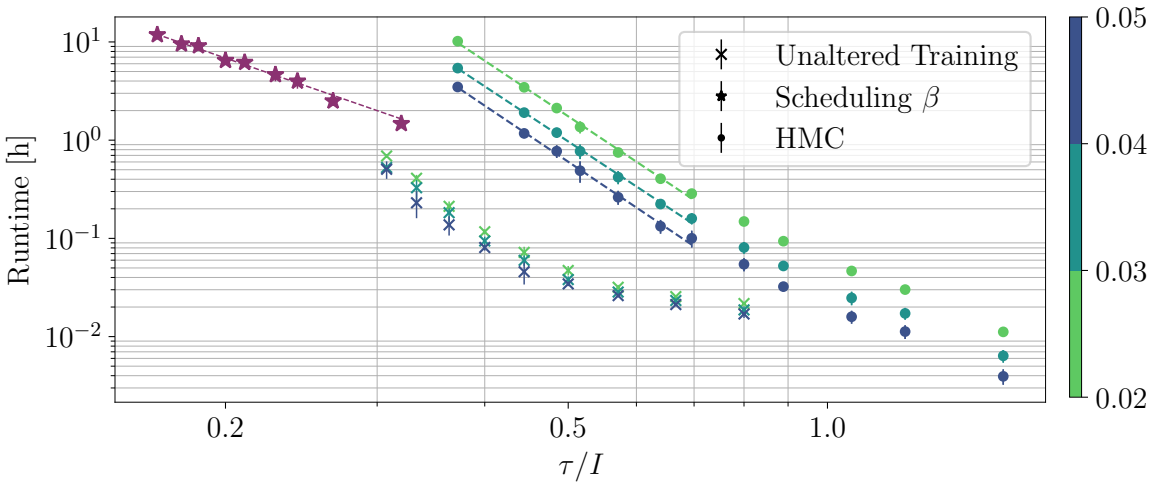


Figure 6.9: Comparison of Runtimes for the HMC algorithm, the Normalizing Flow training without β-scheduling and with β-scheduling on a CLP ($I = 0.25, T = 4$). The relative error thresholds ε_{Q^2} for the HMC and the unaltered Flow training are color-coded and can be extracted from the color bar. As Normalizing Flow architecture, the CNN-type model is trained. For the β-scheduling an absolute step size $\eta_\beta = 0.1$, a damping factor of $\gamma_{Q^2} = 50$ and a patience of 1000 steps are used. The flow training is done with a batch size of 10000, and a learning rate $\eta = 0.001$. For the unaltered training, five independent differently seeded runs are evaluated. Three in the case of the β-scheduling. The HMC is run with 10 independent chains, while the step size is fixed, such that an acceptance rate of 80% is achieved for 10 integration steps during the molecular dynamics evolution.

Before mode collapse, the Normalizing Flow model outperforms the HMC method quite considerably, although we see a larger offset in the runtime, which makes HMC more cost-efficient for large lattice spacings. After the initial phase, the Normalizing Flow and the HMC appear to show comparable scaling behavior. Another interesting point is the scaling behavior regarding the relative error threshold ε_{Q^2}. From Equation 3.28, we see that the runtime cost C scales approximately as

$$C(\varepsilon_{Q^2}) \propto \varepsilon_{Q^2}^{-2}, \tag{6.12}$$

This exponential scaling is plausible in Figure 6.9. The Normalizing Flow setup runtime on the other hand is a combination of the training cost, which results in a certain autocorrelation (see Section 6.3.5) and the bias correction cost, which is the Markov Chain error scaling for the NMCMC method.

Of course, the Normalizing Flow then for smaller values of the lattice spacing, suffers from mode-dropping, which is why at this point the comparison breaks down and no exponential fit was added to the unaltered training data. In this regime, we evaluate the β-scheduling method. Here, the HMC can not be compared anymore due to the exponential increase in $\tau_{int}^{Q^2}$. Regardless, the runtime of the HMC is larger even for larger lattice spacings. As discussed in Section 6.4.1, the bias correction cost is negligible compared to the training cost, and since the runtime optimization is only started once the target density is reached, all plotted error thresholds can be combined into the single data point given. The scaling behavior with respect to the lattice spacing is improved compared to the HMC. However, it is still, at least for the tested regime and simulation hyperparameters, an approximately exponential scaling ($z = -3.04 \pm 0.02$). Nevertheless, β-scheduling scaling can likely be drastically improved, with an adaptive change rate, which also includes scheduling of the learning rate and batch size. Moreover, the runtime offset for the method is needlessly large, since the scheduling is initialized at $\tau/I = 0.5$, which is much larger than the point at which the unaltered training starts to show mode collapse.

In summary, we find that the Normalizing Flow architecture slightly outperforms the HMC algorithm, and we expect the results to further improve if the β-scheduling approach is finetuned.

Chapter 7

Conclusion

In this thesis, we worked with the Topological Quantum Rotor. The model was introduced, and the discretized theory was derived. Through this derivation, we also explicitly acquired a formula for the topological charge of the system.

Moreover, we identified the topological susceptibility χ_t as an observable, which is known to produce diverging integrated autocorrelation times towards the continuum limit when an HMC method is used for its estimation. Based on [12], we derived a closed-form expression for χ_t, which can be easily be evaluated numerically up to a negligible error (see Appendix A). Thus, we were able to compare any estimate for χ_t with the exact value.

Subsequently, the process of estimating expectation values using sampling approaches was introduced. In this context, importance sampling and multiple Monte Carlo Markov Chain methods, such as the Hamiltonian Monte Carlo and the Wolff cluster algorithm were investigated. With the effective sampling sizes ESS_q and ESS_p for the importance sampling and the acceptance rate a and the integrated autocorrelation τ_{int} for a Markov Chain, we derived metrics that can be used to quantify the error on the estimate. The also introduced metropolized independence sampling, as well as the importance sampling, can be used with a sampling algorithm, which is only required to have larger support than p in the importance sampling case and be ergodic in the metropolized independent sampling case.

To verify the choice of observable, we next analyzed the capability of the HMC and Wolff algorithm to estimate the topological susceptibility through $\overline{Q^2}$. We see that indeed the HMC produces exponentially diverging autocorrelation times and runtimes, for a given relative error threshold. The scaling analysis is done for the coupling β of the Quantum Rotor and the lattice dimension $D = |\Lambda|$ as well as for a constant line of physics, where we keep the total time extent T constant and vary β and D accordingly. The Wolff algorithm performed much better and showed only comparatively little correlation between subsequently sampled configurations, and we thus confirmed that it can easily be used to create datasets, where samples can be seen as approximately drawn from the Quantum Rotor target density p.

Then, the idea behind using the Normalizing Flow framework for the expectation value estimation was explained. In principle, its construction allows us to learn any target

density, which can be evaluated up to normalization. To achieve this, we are however locked into using the Reverse KL divergence as training loss. The flow relies on basic design principles, to which the modular components ought to adhere. Ultimately we can choose from multiple proposed designs for most of the modules of the flow. The construction is in particular capable of learning a probability density on any differentiable manifold. Thus, we can adapt the model to the D-dimensional torus, which is the manifold of the lattice variables. The choice of adaptation, in the end, was to follow [6] and use diffeomorphisms, which are explicitly tailored to the circle and additionally map the input to the conditioner onto the circle. We introduced all required parts of the flow architecture and listed respective design choices concerning an adaptation to the Quantum Rotor. At this point it was also noted, how the trained Normalizing Flow model can be used as a sampler for expectation value estimation using the importance sampling and metropolized independence sampling called NIS and NMCMC respectively.

Subsequently, the training results of the Normalizing Flow setup are discussed. They are governed by the condition of mode collapse. In this situation the flow model samples from a density, which does practically neglect comparatively small modes of the target density. A collapsed model can effectively not be used as a sampler and one speaks of an effective breaking of ergodicity. Besides the mode collapse susceptibility, we introduced scaling as a metric for the performance of the Normalizing Flow model. Here we measure how the metrics for NMCMC and NIS, which quantify how well the flow density fits the target density, evolve along the training process. And how the required amount of steps scales for a given metric threshold. Moreover, the minimum required elapsed time, for the entire algorithm consisting of training and bias correction to reach a given relative error on the expectation value estimate was identified as a measure, which is comparable to the HMC. For both measures, due to the stochastic nature of the training process, an error estimate must be obtained by repeating the experiment multiple times with a unique random seed.

Having set up the metrics of performance, we went on to compare multiple design choices for the Normalizing Flow model regarding their mode collapse susceptibility and scaling behavior. Ultimately, the CNN-based conditioner in conjunction with the NCP diffeomorphism performed best. We also concluded that the expressiveness for the diffeomorphism is best kept small since the Quantum Rotor target density can still be modeled well, while the resulting smaller network learns quicker. Since for any tested architecture, we see mode collapse at some point if the coupling is increased, we introduced alterations to the training, designed to mitigate this problem. The β-scheduling method slowly increases the coupling β to the target value and so the initial large flow update due to the stark difference between model density and target density is removed. The scheduling is dependent on the linear evolution of the variance in the topological charge. The coupling β is only changed if its linear development is small enough and additionally, it is exponentially damped by the magnitude of the linear change in the recently logged values. As an alternative, we mixed Reverse KL training and Forward KL training. The idea relies on the zero-avoiding property of the Forward

KL divergence. During the training, one would then slowly evolve the mixing such that ultimately a pure Reverse KL training results. This evolution is dependent on $\overline{Q^2}_{qX}$ sampled topological sectors in an equal fashion to the scheduling of β. In our results, the β-scheduling method drastically outperformed the mixing method concerning the mode collapse susceptibility. Moreover, it requires less arbitrary hyperparameters, such as the distribution of topological sectors for batches in the Forward KL training process. With the β-scheduling method, we have been successfully able to reduce the model's susceptibility to mode collapse. We hypothesize that the scaling behavior of the β-scheduling can be improved by using an adaptive method that modifies the step size, learning rate and batch size during the scheduling process.

Importantly, with the β-scheduling method a generally applicable way to circumvent mode collapse has been analyzed, which is not bound to the Quantum Rotor as a target system. Instead, the evolution of the spread of sampled sectors is utilized as a generally available metric.

Here we note that there are more ways to improve or circumvent the problem of mode collapse for the Normalizing Flow setup. Another method might be the gradient boosting approach [60], which enables us to design an ensemble of separate flows, which are trained on different topological sectors. Subsequently, the ensemble flows are assigned relative weights, and we would hypothetically sample from a density close to the target. An intriguing architecture for future study is The Embedded Flow model introduced in [61]. Here probabilistic programming Normalizing Flow layers could be designed to manifestly include the multi-modality of the target density. Most importantly, the current focus of many publications is to include the target systems' symmetry in the flow through equivariant layers as we discussed in Section 5.9. However, designing equivariant layers for physical theories like QCD requires further research.

In the final section of our training analysis, the cost for the Normalizing Flow was then compared to the cost of the HMC for a given relative error on the estimate of Q^2. For the same regime in which the HMC scaling could be analyzed, we see that the Normalizing Flow setup outperforms the HMC. This can only be extended to the point where mode collapse cannot be neglected anymore. At this point the β-scheduling is compared. Here, as mentioned, we are not able to quantify an error for the HMC due to the exponentially increasing computational cost. However, the local scaling of the Normalizing Flow method is smaller than for the HMC.

Still, we encounter a scaling behavior, which is locally in good agreement with an exponential relation. Albeit the scaling is smaller than the HMC's, we are not able to achieve a logarithmic scaling reported e.g. in [17], and the tested system sizes are far smaller as well. Increasing the system size is a task, we hypothesize, that should be solvable for the Normalizing Flow training. This is because a local conditioner architecture like the CNN is used, and the batch size can be increased since a single training update step is highly parallelizable.

To conclude, it should be said that the results obtained for Normalizing Flows in this thesis do not justify their use in more elaborate theories. Albeit, we have found a

way to circumvent the mode collapse of the model, we have obtained scaling behavior, which is only slightly better than the one of the HMC algorithm. Consequently, to represent an improvement upon existing MCMC methods, the Normalizing Flow setup likely requires more elaborate alterations mentioned above.

Appendix A

Error Bound for Exact Susceptibility

In Section 2.2, we have obtained a closed form expression for the topological susceptibility of the Rotor:

$$\begin{aligned}\chi_t &= \frac{1}{T}\frac{\partial_\theta^2 \mathscr{Z}(\theta, D, \beta)}{\mathscr{Z}(\theta, D, \beta)}\bigg|_{\theta=0} = -\frac{D}{T}\sum_{k\in\mathbb{Z}}(D-1)w_k(0,D,\beta)\left[\frac{\mathcal{I}_k'(\beta)}{\mathcal{I}_k(\beta)}\right]^2 + w_k(0,D,\beta)\frac{\mathcal{I}_k''(\beta)}{\mathcal{I}_k(\beta)} \\ &:= -\frac{D}{T}\sum_{k\in\mathbb{Z}} S_k.\end{aligned} \tag{A.1}$$

This expression obviously only makes sense if the series converges. So we should prove that. Realistically, only a finite amount of terms can be computed when χ_t is evaluated, and so one needs to introduce a cutoff $|k'|$. At that point it is important to be aware of the error, which is introduced by the cutoff. In the following, we are going to provide an upper bound on said error.

The first observation is that $S_k = S_{-k}$, since $\mathcal{I}_k = \mathcal{I}_{-k}$, $\mathcal{I}'_k = -\mathcal{I}_{-k}$ and $\mathcal{I}''_k = \mathcal{I}''_{-k}$. Thereby we can restructure Eq. A.1:

$$\chi_t = -\frac{D}{T}\left(S_0 + 2\sum_{k\in\mathbb{N}} S_k\right). \tag{A.2}$$

Effectively, we thus need an upper bound on $S_k \leq |S_k|\ \forall k \in \mathbb{N}$. Since according to [62, 63], $\frac{\mathcal{I}_{\nu+1}(x)}{\mathcal{I}_\nu(x)} > 0$ for $\nu > \frac{1}{2}$,

$$|w_k| = \left|\frac{\mathcal{I}_k^D}{\sum_{k\in\mathbb{Z}}\mathcal{I}_k^D}\right| = \left|\frac{\left(\frac{\mathcal{I}_k}{\mathcal{I}_0}\right)^D}{\sum_{k\in\mathbb{Z}}\left(\frac{\mathcal{I}_k}{\mathcal{I}_0}\right)^D}\right| \leq \left(\frac{\mathcal{I}_k}{\mathcal{I}_0}\right)^D. \tag{A.3}$$

Notice that for notational simplicity we drop the variable dependencies. With this bound on the absolute weights, we can rewrite

$$|w_k|\left|\frac{\mathcal{I}_k'}{\mathcal{I}_k}\right|^2 \leq \left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2}\left|\frac{\mathcal{I}_k'}{\mathcal{I}_0}\right|^2 \leq \left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2}\left(\frac{\int_{-\pi}^{\pi} d\phi\ \frac{|\phi|}{2\pi}|e^{ik\phi}|e^{x\cos(\phi)}}{\int_{-\pi}^{\pi} d\phi\ e^{x\cos(\phi)}}\right)^2 \leq \frac{1}{4}\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2}, \quad \text{(A.4)}$$

as well as

$$|w_k|\left|\frac{\mathcal{I}_k''}{\mathcal{I}_k}\right| \leq \left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-1}\left|\frac{\mathcal{I}_k''}{\mathcal{I}_0}\right| \leq \left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-1}\frac{\int_{-\pi}^{\pi} d\phi\ \left(\frac{\phi}{2\pi}\right)^2|e^{ik\phi}|e^{x\cos(\phi)}}{\int_{-\pi}^{\pi} d\phi\ e^{x\cos(\phi)}} \leq \frac{1}{4}\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-1} \quad \text{(A.5)}$$

for $D \geq 2$ and $D \geq 1$ respectively.

Next, we use additional properties of the Bessel Functions of the first kind described in [62, 63]. For $\nu \geq \frac{1}{2}$, $\mathcal{I}_\nu > \mathcal{I}_{\nu+1}$ and the ratio

$$\frac{\mathcal{I}_{\nu+1}(x)}{\mathcal{I}_\nu(x)} \leq \exp\left(-\alpha_0\frac{\nu+\frac{1}{2}}{x}\right) < 1, \quad \text{(A.6)}$$

where $\alpha_0 = -\log\left(\sqrt{2}-1\right)$. Through this identity, we can give an upper bound for any ratio

$$\begin{aligned}\frac{\mathcal{I}_{\nu+n}(x)}{\mathcal{I}_\nu(x)} &= \prod_{i=0}^{n-1}\frac{\mathcal{I}_{\nu+i+1}(x)}{\mathcal{I}_{\nu+i}(x)} \leq \prod_{i=0}^{n-1}\exp\left(-\alpha_0\frac{\nu+i+\frac{1}{2}}{x}\right)\\ &= \exp\left(-\frac{\alpha_0}{x}\sum_{i=0}^{n-1}\nu+i+\frac{1}{2}\right) = \exp\left(-\frac{\alpha_0}{x}\left(n\nu+\frac{n}{2}+\frac{n(n+1)}{2}-n\right)\right)\\ &= \exp\left(-\frac{\alpha_0}{x}\left(n\nu+\frac{n^2}{2}\right)\right).\end{aligned} \quad \text{(A.7)}$$

In terms of all estimates above, the absolute series terms $|S_k|$ can be drastically simplified:

$$\begin{aligned}
|S_k| &= \left|(D-1)w_k(0,D,\beta)\left[\frac{\mathcal{I}_k'(\beta)}{\mathcal{I}_k(\beta)}\right]^2 + w_k(0,D,\beta)\frac{\mathcal{I}_k''(\beta)}{\mathcal{I}_k(\beta)}\right| \\
&\leq (D-1)\left|w_k(0,D,\beta)\left[\frac{\mathcal{I}_k'(\beta)}{\mathcal{I}_k(\beta)}\right]^2\right| + \left|w_k(0,D,\beta)\frac{\mathcal{I}_k''(\beta)}{\mathcal{I}_k(\beta)}\right| \\
&\leq \frac{D-1}{4}\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2} + \frac{1}{4}\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-1} \\
&\leq \frac{D-1}{4}\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2} + \frac{1}{4}\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2} \\
&= D\left|\frac{\mathcal{I}_k}{\mathcal{I}_0}\right|^{D-2} \\
&\overset{A.7}{\leq} D\exp\left(-\frac{\alpha_0(D-2)k^2}{2x}\right).
\end{aligned} \tag{A.8}$$

For the first inequality is justified by the triangle inequality. Since the terms $|S_k|$ vanish exponentially fast and in particular faster than $1/k$, through the direct comparison test the series converges absolutely and therefore also without taking the absolute values. The error that a cutoff introduces can be estimated as

$$\begin{aligned}
R_{k'} &= \int_{k'}^{\infty} dk\ \frac{2D^2}{T}S_k \leq \frac{2D^2}{T}\int_{k'}^{\infty} dk\ |S_k| \\
&\leq \frac{2D^2}{T}\int_{k'}^{\infty} dk\ \exp\left(-\frac{\alpha_0(D-2)k^2}{2x}\right) \\
&= \frac{D^2\sqrt{\pi}}{T\sqrt{\frac{\alpha_0(D-2)}{2x}}}\operatorname{erfc}\left(\sqrt{\frac{\alpha_0(D-2)}{2x}}k'\right).
\end{aligned} \tag{A.9}$$

This means that for a tolerable error $\varepsilon > 0$,

$$k' \geq \left\lceil \sqrt{\frac{2x}{\alpha_0(D-2)}}\operatorname{erfc}^{-1}\left(\frac{\varepsilon T\sqrt{\frac{\alpha_0(D-2)}{2x}}}{D^2\sqrt{\pi}}\right)\right\rceil. \tag{A.10}$$

In Figure A.1 k' is plotted for $\varepsilon = 10^{-16}$. Since the error vanishes so quickly, for all calculations, in this thesis k' was set to match double precision and any error on χ_t was taken to be negligible.

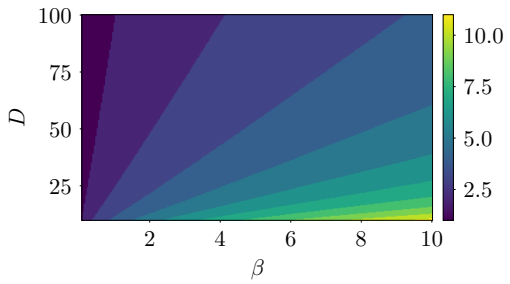


Figure A.1: Minimum required k' from Equation A.10 such that the error on χ_t is smaller than double precision 10^{-16}.

Appendix B

Validation and Tests

To ensure that the methods for numerical simulation studied in this thesis are implemented correctly, unit, as well as integration tests are designed. The action and observables are unit-tested.

This includes checking the returned results for handpicked inputs, as well as checking mathematical properties. The discretized action as well as the discretized topological charge should not be affected by global shifts of the lattice variables as outlined in Equation 5.49. While the action is symmetric regarding the manipulation $\boldsymbol{\phi} \leftarrow -\boldsymbol{\phi}$, the topological charge takes up a sign. The latter also only ought to only assume integer values. Moreover, since we have implemented the action and observables in `C++` in addition to python, the different implementations are checked for consistency.

A straightforward integration test can be constructed by comparing the estimated expectation value for the topological susceptibility calculated in Section 2.2 with the output of the simulation method to be tested. If a method M estimates the expectation value $T\chi_t$ to be $\overline{Q^2}^{(M)}$ with an error of $\overline{\sigma}^{(M)}_{\overline{Q^2}}$, we require that at least

$$\left| T\chi_t - \overline{Q^2}^{(M)} \right| < n\overline{\sigma}^{(M)}_{\overline{Q^2}} \tag{B.1}$$

ought to be reached. n is an integer generally required to be small. This is done for a two-dimensional grid of configurations $\{(\beta, D) | \beta \in \mathbb{R}_{>0}, D \in \mathbb{N}\}$, where the values are chosen to be in a regime, in which the simulation method produces results without much computational effort or method specific complications. For the Quantum Rotor system, this regime corresponds to small values for β, as this leaves the target density $\tilde{p}$ relatively flat. Varying D has a less grave impact, however in general the computational effort increases with the dimension of the lattice.

B.1 Monte Carlo

The *Hybrid Monte Carlo* method suffers from a drastically increased autocorrelation time for increased β. In order to produce the necessary amount of configurations in a comparatively small amount of time, the HMC was implemented in `C++` and made

callable form Python using *PyBind11* [64]. While it is not strictly necessary to use redundant implementations of the Rotor action and observables in the different used languages, it still is acceptable, if the implementations are tested for consistency.

A crucial factor for the evaluation using Markov Chains is to produce a chain that is sufficiently long. In Figure B.1a, for a single pair of values β, D different chain lengths are used to estimate the topological susceptibility χ_t through Q^2/T. In the lower plot, it becomes apparent, that for a mean value close to the exact one, one needs a certain amount of configurations. The error on the estimate is however also not reliable if we use too few configurations, as we can see in the upper plot, which depicts the estimated autocorrelation times $\tau_{\text{int}Q^2}$ for the different chain lengths. For a decent estimation of the expectation value and the error associated with the estimation, it is therefore integral to use a sufficient amount of configurations. According to [16], 1000 τ_{int} configurations are desired for food statistical accuracy and typically more than $10000\tau_{\text{int}}$ for less than 1% error. Considering Figure B.1a this seems to be a valid order of magnitude. Additionally, $20\tau_{\text{int}}$ should be discarded at the beginning of the chain to ensure that equilibrium has been reached. In terms of validating and testing the implementation, the estimate must tend towards the exact expectation value and the error on the estimate decreases, which is what we theoretically expect from the method.

The Cluster algorithm from Section 3.3.5 produces configurations, whose topological charges are almost uncorrelated. So the integrated autocorrelation time is comparatively small. Better yet, it decreases for increasing β. So only comparatively few samples need to be produced to achieve the desired accuracy. This is why an implementation in Python suffices.

Both MCMC algorithms have been tested for a subset of combinations of the input parameters β and D, where it was checked that we get sufficiently close to the theoretical topological susceptibility using the configurations in the produced Markov Chain in the sense of B.1. This is shown in Figures B.1.

B.2 Normalizing Flows

The Normalizing Flow was implemented in Python using *PyTorch*. There are ultimately infinitely many flow configurations to be tested since most aspects of the architecture introduced in Chapter 5 can be varied freely, and it is up to the user to determine a realization of an architecture that most effectively solves the problem at hand. Testing all different possible architectures is therefore impossible. To test a given complete flow model with the Reverse KL training setup works in the same way as validating the MCMC. First, the model is trained on a Rotor action with given parameters β and D up to a point of sufficient convergence. In a regime of small β, one can use metrics like the ESS_q or acceptance rate during NMCMC to determine how close the model's target density q_X is to p. If these metrics increase and get close to 1 during training, this means that the model learns the target density.

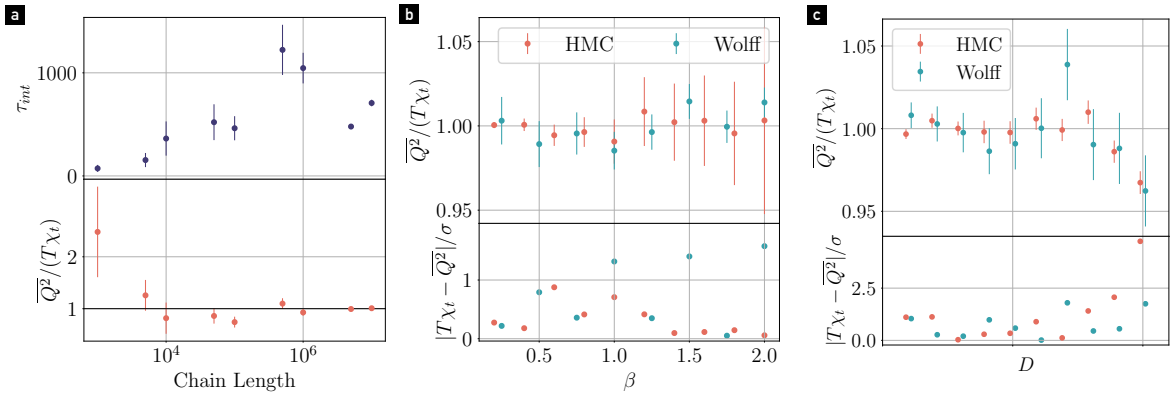


Figure B.1: Validation of the validity of the implemented MCMC methods HMC and Wolff. (a) $\overline{Q^2}^{(\mathrm{HMC})}$ evaluation for a variable number of configurations produced. The Rotor parameters are set as $D = 25$ and $\beta = 1.5$. The upper plot displays the integrated autocorrelation times and the lower plot the expectation value estimate. (b) MCMC results for different values of β, while $D = 25$. (c) MCMC results for different values of D, while $\beta = 1.0$. In (a) and (b) the HMC is run $5 \cdot 10^6$ steps, with 10^5 burn-in steps. The Wolff cluster algorithm is run with 10^5 steps and 25000 burn-in steps. The molecular dynamics trajectory in between Metropolis steps for the HMC is for all runs set to be 20 steps with a step size that is tuned to produce an acceptance rate of $80 \pm 3\%$ via bisection search. In Figures (b) and (c) the x-values in the plotted data are slightly offset for better readability.

The trained model is then used to sample from q_X. With NIS and NMCMC, we correct the bias in the flow's density. Again a sufficient number of configurations is needed to provide good statistics. How many configurations are needed, depends on the convergence of the model. The better it is trained, the fewer are needed. In Figure B.2a, for a fixed β and D, NIS and NMCMC are used with a varying amount of configurations to estimate the expectation value of Q^2. We see that with an increasing number of sampled configurations the error on the estimate decreases, and the estimate tends towards the theoretical value, which is what we expect. Figures B.2b and B.2c show - for a subset of combinations of the parameters β and D - that the flow model achieves a sufficient estimate in the sense of B.1 with both NIS and NMCMC.

The Forward KL training setup is validated in two ways. Consider the mixing method introduced in 6.4.2. We can dictate an arbitrary topological charge distribution for the samples within a batch shown in the model. After training the topological charges of configurations produced by the model should be distributed close to the dictated input distribution.

Another method involves training a model on a dataset created by the Wolff algorithm. Here we expect to learn a density q_X, which should be reasonably close to the Quantum Rotor density for the used parameters β and D. And so the estimate $\overline{Q^2}_{q_X}$ should be close to the theoretical value $T\chi_t$.

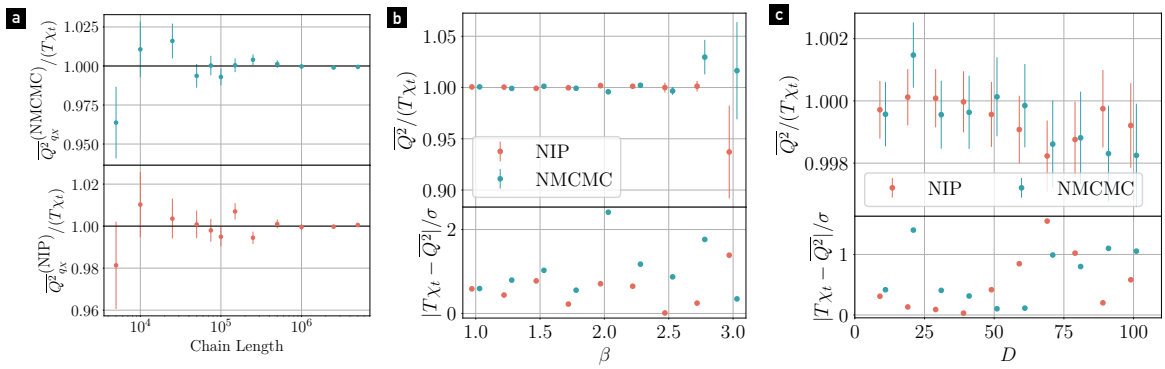


Figure B.2: Normalizing Flow Validation. We use an exemplary model with an MLP conditioner architecture with three hidden layers of size 50 each, an NCP diffeomorphism with expressiveness $K = 8$ and 5 layers, and 5 coupling layers with checkerboard type split. The model is trained for 50000 steps with a batch size of 10000. (a) $\overline{Q^2}$ evaluation using NIS and NMCMC for a variable number of configurations produced. $D = 25$ and $\beta = 1.5$. In Figures (b) and (c) and the upper plot displays the estimate rescaled by the theoretical expectation value. The lower plot displays the relative error in the sense of B.1. (b) $\overline{Q^2}$ evaluation for the Normalizing Flow setup for different values of β, where $D = 25$. (c) $\overline{Q^2}$ evaluation for the Normalizing Flow setup for different values of D, where $\beta = 1.0$.

Appendix C

Implementation details

This chapter includes implementation details concerning the experiments in this thesis.

C.1 Implementation Details for Torus Diffeomorphisms

Several functions defined for the task of training Normalizing Flows on a manifold like a torus require constrained input parameters. That means a parameter γ might be restricted to a sub-interval $\Omega_\gamma \subset \mathbb{R}$. Possible restrictions are e.g. $\Omega = \mathbb{R}_+, \Omega = \mathbb{R}_-$ or $\Omega = [a, b]$, where $a, b \in \mathbb{R}$. Additionally, there can be interdependence of several input parameters. For example, we might require a set of parameters $\{\gamma_0, \ldots, \gamma_N\}$ to be on the simplex such that $\sum_i^N \gamma_i = 1$ or the sphere such that $\sum_i^N \gamma_i^2 = 1$.

In the course of training the Normalizing Flow model, we update free model parameters using gradient descent. Restricting the gradient-based optimization to a predefined range can be achieved by having an unconstrained parameter, which is subject to training updates and is mapped to its constrained range at every evaluation step. If the right mapping function is used, we thus guarantee the correct parameter range for evaluation.

Note additionally, that a Deep Learning architecture does not necessarily produce unconstrained outputs. It is very common to insert an activation function between linear layers, which can constrain the output space. All models in this thesis have thus been designed without a non-linear activation at the output layer.

C.1.1 Constraining Maps for Diffeoorphisms

The residual layer weights as well as the parameters for the convex combinations for the Moebius and NCP transformations and the widths and heights of the segments in the CS transformation are constrained to the simplex with the *Softmax* function. Given

unconstrained weights $\{\rho'_1, \ldots, \rho'_N\}$, we obtain weights constrained to the simplex with

$$\rho_i = \frac{\exp(\rho'_i)}{\sum_{i=1}^{N} \exp(\rho'_i)} \tag{C.1}$$

in a smooth and differentiable way.

For limiting a parameter to either the positive or negative half of the real axis, as required for the NCP and CS, one can for example use the Softplus function

$$\gamma = \log(1 + \exp(\gamma')) \tag{C.2}$$

or the exponential function.

To create a two-dimensional parameter, whose L^2-norm is less than one, i.e. is it strictly contained inside a circle of radius one, we can use

$$(w_1, w_2) = 0.99 \frac{(w_1, w_2)}{1 + \|(w'_1, w'_2)\|}, \tag{C.3}$$

where $(w'_1, w'_2) \in \mathbb{R}^2$. Here, we limit the norm to 0.99 by choice. This map is required for the Moebius transformation.

C.1.2 Implementation Details

The derivative of the Moebius transformation $F_{moebius}$ defined in Section 5.5.1 with respect to the input angle ϕ is

$$\begin{aligned}
\frac{\partial F_{moebius}(\phi; \boldsymbol{w})}{\partial \phi} &= \frac{\partial}{\partial \phi} \operatorname{atan2}\left(\tilde{F}_{moebius}(z(\phi), \boldsymbol{w})\right) \\
&= \frac{\begin{pmatrix} -h_y & h_x \end{pmatrix}}{h_x^2 + h_y^2} \cdot \begin{pmatrix} \beta - (z_x - w_x)^2 & -(z_y - w_y)(z_x - w_x) \\ -(z_x - w_x)(z_y - w_y) & \beta - (z_y - w_y)^2 \end{pmatrix} \\
&\quad \cdot \frac{2\beta}{\|\boldsymbol{z} - \boldsymbol{w}\|^2} \cdot \begin{pmatrix} \frac{dz_x}{d\phi} \\ \frac{dz_y}{d\phi} \end{pmatrix}
\end{aligned} \tag{C.4}$$

We have defined $\beta = \frac{1 - \|\boldsymbol{w}\|^2}{\|\boldsymbol{z} - \boldsymbol{w}\|^2}$ and $\boldsymbol{h} = \tilde{F}_{moebius}(z(\phi), \boldsymbol{w})$.

The NCP map can be numerically unstable at the interval boundaries. The linearized computation discussed in Section 5.5.1 is done for $\operatorname{mod}|\phi - 2\pi|_{2\pi} < 10^{-3}$. Moreover, to further increase numerical stability, 10^{-3} is added to the constrained parameter α.

The CS transformation is the only analyzed map, which is explicitly invertible for

increased arbitrary complexity. Formulae for the inverse can be found in [40]. To this end, we have used code provided in [65].

The residual diffeomorphism combination parameters can be designed arbitrarily. One might use fixed values or schedule them during the training process. Since this can be done with a gradient descent-based approach, one needs to introduce unconstrained parameters (ρ_1', ρ_2'), which are mapped onto the simplex $\{(\rho_1, \rho_2) \in [0,1]^2 | \rho_1 + \rho_1 = 1\}$ using the Softmax transform (C.1) at every step they are needed. The initial value ρ_1^{initial} can be set to a specific value x for an arbitrary value $y \in \mathbb{R}$ through:

$$\begin{aligned} \rho_1' &= e^y \\ \rho_2' &= e^{y + \log((1-x)/x)}. \end{aligned} \tag{C.5}$$

Alternatively, the residual connection can be scheduled manually. For the flow training in this thesis, we use an exponentially decreasing evolution after the initial training stage, such that we reach a plain diffeomorphism in a controlled manner later on. For this, we also act on unconstrained values. This is not strictly necessary though but serves the purpose of removing code redundancy. In order to reduce ρ_1 by a factor γ_ρ, we apply

$$\begin{aligned} x &= \frac{1}{2}\left(\log\left(\frac{\gamma_\rho e^{\rho_1'}}{e^{\rho_2'} + (1-\gamma_\rho)e^{\rho_1'}}\right) + \rho_2' - \rho_1'\right) \\ \rho_1' &\leftarrow \rho_1' + x \\ \rho_2' &\leftarrow \rho_2' - x. \end{aligned} \tag{C.6}$$

Whenever a diffeomorphism can only be inverted numerically, bisection search is used. For an implementation, consult e.g. [45]. For all inversions, a precision of 10^{-4} as tolerance was used.

C.2 Implementation of Metrics

For the evaluation of metrics with the Normalizing Flow as the sampler, the model is used to produce configurations with 64-bit double precision. This is necessary to improve the accuracy of the metrics. Additionally, it mitigates overflow in the weight ratios q/p or q/p. This can happen if the flow model drastically over or underestimates the target density.

The acceptance rate a, as well as, the ESS_q are numerically stable and straightforward to obtain.

As shown in Section 6.1 the ESS_p becomes extremely small for a badly converged Normalizing Flow. So it is important to use a numerically stabilized manner of evaluation. Generally, we evaluate the importance weights w in log space $\log w = \log q - \log p$. Then for $\log w$ as well as $-\log w = \log w^{-1}$, the maximum is subtracted. This helps

prevent overflow in the summation of the exponentials $w = \log(\exp(w))$. The summation itself is done with Pytorch's numerically stabilized *logsumexp* function [66]. The pulled-out maxima, as well as normalization constants, are re-introduced at the end. Ultimately, we take the exponential of the result. This has turned out to be the most stable way of calculating ESS_p.

The effective sampling size ESS_p can usually not be evaluated, since there are no configurations from the target distribution at hand. As described in 3.3.5, we are for the Quantum Rotor however able to produce configurations from p using the Wolff Cluster Algorithm. Estimating the expectation value requires a set of configurations that adequately represents sampling from p. Since the Wolff algorithm does not suffer from an increased autocorrelation time for larger coupling β and only slightly increased integrated autocorrelation time for larger system size D, we can fix a size for a set of configurations, on which ESS_p is evaluated. For all measured configurations no integrated autocorrelation time $\tau_{int}^{Q^2}$ above 15 was measured. This can, of course, be removed by introducing $\approx 2\tau_{int}^{Q^2}$ update steps without recording a configuration in between sampling. All sets for which ESS_p was measured were set to a size of 250000 independent configurations.

C.3 Forward KL training

The Forward KL training is based on datasets. As it would take too long (depending on the used algorithm) to produce these in parallel during training, they are saved to disk. For this, we store the configurations in an LMDB dataset [67] with 32-bit precision. The key-based approach makes retrieval of single configurations very fast compared to other alternatives. Thus, we can efficiently draw random batches from the dataset during training.

Appendix D

Hardware

While for the thesis multiple different hardware configurations were used, in Table D.1 we give the hardware on which the times were specified.

Task	**Hardware**
HMC elapsed time per configuration	M1 Pro (2021)
Training Time plain flow	Tesla P100-PCIE-12GB
Training Time β scheduled	NVIDIA A100-PCIE-40GB

Table D.1: Hardware devices used for timing experiments.

Bibliography

[1] Simon Duane et al. "Hybrid Monte Carlo". In: *Physics Letters B* 195.2 (Sept. 3, 1987), pp. 216–222. ISSN: 0370-2693. DOI: 10.1016/0370-2693(87)91197-X. URL: https://www.sciencedirect.com/science/article/pii/037026938791197X (visited on 01/17/2023).

[2] Stefan Schaefer, Rainer Sommer, and Francesco Virotta. "Critical Slowing down and Error Analysis in Lattice QCD Simulations". In: *Nuclear Physics B* 845.1 (Apr. 2011), pp. 93–119. ISSN: 05503213. DOI: 10.1016/j.nuclphysb.2010.11.020. arXiv: 1009.5228. URL: http://arxiv.org/abs/1009.5228 (visited on 11/13/2020).

[3] Luigi Del Debbio, Gian Mario Manca, and Ettore Vicari. "Critical Slowing down of Topological Modes". In: *Physics Letters B* 594.3-4 (Aug. 2004), pp. 315–323. ISSN: 03702693. DOI: 10.1016/j.physletb.2004.05.038. arXiv: hep-lat/0403001. URL: http://arxiv.org/abs/hep-lat/0403001 (visited on 01/18/2023).

[4] Danilo Jimenez Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". June 14, 2016. arXiv: 1505.05770 [cs, stat]. URL: http://arxiv.org/abs/1505.05770 (visited on 01/07/2021).

[5] Martin Lüscher. "Trivializing Maps, the Wilson Flow and the HMC Algorithm". In: *Communications in Mathematical Physics* 293.3 (Feb. 2010), pp. 899–919. ISSN: 0010-3616, 1432-0916. DOI: 10.1007/s00220-009-0953-7. arXiv: 0907.5491 [hep-lat]. URL: http://arxiv.org/abs/0907.5491 (visited on 09/05/2022).

[6] Danilo Jimenez Rezende et al. "Normalizing Flows on Tori and Spheres". July 1, 2020. arXiv: 2002.02428 [cs, stat]. URL: http://arxiv.org/abs/2002.02428 (visited on 01/07/2021).

[7] Denis Boyda et al. "Sampling Using $SU(N)$ Gauge Equivariant Flows". In: *Physical Review D* 103.7 (Apr. 20, 2021), p. 074504. ISSN: 2470-0010, 2470-0029. DOI: 10.1103/PhysRevD.103.074504. arXiv: 2008.05456 [hep-lat, stat]. URL: http://arxiv.org/abs/2008.05456 (visited on 08/12/2022).

[8] Gurtej Kanwar et al. "Equivariant Flow-Based Sampling for Lattice Gauge Theory". In: *Physical Review Letters* 125.12 (Sept. 15, 2020), p. 121601. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.125.121601. arXiv: 2003.06413 [cond-mat, physics:hep-lat]. URL: http://arxiv.org/abs/2003.06413 (visited on 08/10/2022).

[9] Luigi Del Debbio, Joe Marsh Rossney, and Michael Wilson. "Efficient Modelling of Trivializing Maps for Lattice $\phi4$ Theory Using Normalizing Flows: A First Look at Scalability". In: *Physical Review D* 104.9 (Nov. 15, 2021), p. 094507. ISSN: 2470-0010, 2470-0029. DOI: 10.1103/PhysRevD.104.094507. arXiv: 2105.12481 [hep-lat]. URL: http://arxiv.org/abs/2105.12481 (visited on 07/18/2022).

[10] Daniel C. Hackett et al. *Flow-Based Sampling for Multimodal Distributions in Lattice Field Theory*. July 1, 2021. arXiv: 2107.00734 [cond-mat, physics:hep-lat]. URL: http://arxiv.org/abs/2107.00734 (visited on 07/27/2022).

[11] W. Bietenholz et al. "Perfect Lattice Topology: The Quantum Rotor as a Test Case". In: *Physics Letters B* 407.3-4 (Sept. 1997), pp. 283–289. ISSN: 03702693. DOI: 10.1016/S0370-2693(97)00742-9. arXiv: hep-lat/9704015. URL: http://arxiv.org/abs/hep-lat/9704015 (visited on 02/11/2022).

[12] Claudio Bonati and Paolo Rossi. "Topological Susceptibility of Two-Dimensional U (N) Gauge Theories". In: *Physical Review D* 99.5 (Mar. 18, 2019), p. 054503. ISSN: 2470-0010, 2470-0029. DOI: 10.1103/PhysRevD.99.054503. URL: https://link.aps.org/doi/10.1103/PhysRevD.99.054503 (visited on 03/23/2022).

[13] Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. New York, NY: Springer New York, 2004. ISBN: 978-0-387-76369-9 978-0-387-76371-2. DOI: 10.1007/978-0-387-76371-2. URL: http://link.springer.com/10.1007/978-0-387-76371-2 (visited on 08/25/2022).

[14] Eric Cancès, Frédéric Legoll, and Gabriel Stoltz. "Theoretical and Numerical Comparison of Some Sampling Methods for Molecular Dynamics". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 41.2 (Mar. 2007), pp. 351–389. ISSN: 0764-583X, 1290-3841. DOI: 10.1051/m2an:2007014. URL: http://www.esaim-m2an.org/10.1051/m2an:2007014 (visited on 08/25/2022).

[15] Jun S. Liu. "Metropolized Independent Sampling with Comparisons to Rejection Sampling and Importance Sampling". In: *Statistics and Computing* 6.2 (June 1996), pp. 113–119. ISSN: 0960-3174, 1573-1375. DOI: 10.1007/BF00162521. URL: http://link.springer.com/10.1007/BF00162521 (visited on 03/04/2022).

[16] Christof Gattringer and Christian B. Lang. *Quantum Chromodynamics on the Lattice*. Vol. 788. Lecture Notes in Physics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-01849-7 978-3-642-01850-3. DOI: 10.1007/978-3-642-01850-3. URL: http://link.springer.com/10.1007/978-3-642-01850-3 (visited on 02/11/2022).

[17] Karl Jansen, Eike Hermann Müller, and Robert Scheichl. "Multilevel Monte Carlo for Quantum Mechanics on a Lattice". Aug. 7, 2020. arXiv: 2008.03090 [hep-lat, physics:physics]. URL: http://arxiv.org/abs/2008.03090 (visited on 11/02/2020).

[18] Ulli Wolff. "Monte Carlo Errors with Less Errors". In: *Computer Physics Communications* 156.2 (Jan. 2004), pp. 143–153. ISSN: 00104655. DOI: 10.1016/S0010-4655(03)00467-3. URL: https://linkinghub.elsevier.com/retrieve/pii/S0010465503004673 (visited on 03/17/2022).

[19] Barbara De Palma et al. "A Python Program for the Implementation of the {\Gamma}-Method for Monte Carlo Simulations". In: *Computer Physics Communications* 234 (Jan. 2019), pp. 294–301. ISSN: 00104655. DOI: 10.1016/j.cpc.2018.07.004. arXiv: 1703.02766. URL: http://arxiv.org/abs/1703.02766 (visited on 03/18/2022).

[20] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (June 1953), pp. 1087–1092. ISSN: 0021-9606, 1089-7690. DOI: 10.1063/1.1699114. URL: http://aip.scitation.org/doi/10.1063/1.1699114 (visited on 03/18/2022).

[21] *Leapfrog Integration*. In: *Wikipedia*. Jan. 29, 2022. URL: https://en.wikipedia.org/w/index.php?title=Leapfrog_integration&oldid=1068682147 (visited on 03/19/2022).

[22] Alain Durmus, Éric Moulines, and Eero Saksman. "Irreducibility and Geometric Ergodicity of Hamiltonian Monte Carlo". In: *The Annals of Statistics* 48.6 (Dec. 1, 2020). ISSN: 0090-5364. DOI: 10.1214/19-AOS1941. URL: https://projecteuclid.org/journals/annals-of-statistics/volume-48/issue-6/Irreducibility-and-geometric-ergodicity-of-Hamiltonian-Monte-Carlo/10.1214/19-AOS1941.full (visited on 08/25/2022).

[23] M. S. Albergo, G. Kanwar, and P. E. Shanahan. "Flow-Based Generative Models for Markov Chain Monte Carlo in Lattice Field Theory". In: *Physical Review D* 100.3 (Aug. 22, 2019), p. 034515. ISSN: 2470-0010, 2470-0029. DOI: 10.1103/PhysRevD.100.034515. arXiv: 1904.12072. URL: http://arxiv.org/abs/1904.12072 (visited on 03/21/2021).

[24] Ulli Wolff. "Collective Monte Carlo Updating for Spin Systems". In: *Physical Review Letters* 62.4 (Jan. 23, 1989), pp. 361–364. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.62.361. URL: https://link.aps.org/doi/10.1103/PhysRevLett.62.361 (visited on 03/04/2022).

[25] A. Ammon et al. "On the Efficient Numerical Solution of Lattice Systems with Low-Order Couplings". In: *Computer Physics Communications* 198 (Jan. 2016), pp. 71–81. ISSN: 00104655. DOI: 10.1016/j.cpc.2015.09.004. arXiv: 1503.05088. URL: http://arxiv.org/abs/1503.05088 (visited on 10/21/2020).

[26] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112.518 (Apr. 3, 2017), pp. 859–877. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.2017.1285773. arXiv: 1601.00670. URL: http://arxiv.org/abs/1601.00670 (visited on 01/07/2021).

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[28] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. Cambridge, MA: MIT Press, 2012. 1067 pp. ISBN: 978-0-262-01802-9.

[29] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". Jan. 29, 2017. arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 03/26/2022).

[30] *PyTorch*. URL: https://www.pytorch.org (visited on 03/26/2022).

[31] George Papamakarios et al. "Normalizing Flows for Probabilistic Modeling and Inference". Dec. 5, 2019. arXiv: 1912.02762 [cs, stat]. URL: http://arxiv.org/abs/1912.02762 (visited on 01/07/2021).

[32] Leonor Godinho and José Natário. *An Introduction to Riemannian Geometry: With Applications to Mechanics and Relativity*. Universitext. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-08665-1 978-3-319-08666-8. DOI: 10.1007/978-3-319-08666-8. URL: http://link.springer.com/10.1007/978-3-319-08666-8 (visited on 09/16/2022).

[33] Dr Frederic P Schuller. "Lectures on the Geometric Anatomy of Theoretical Physics". In: (), p. 218.

[34] Kim A. Nicoli et al. "On Estimation of Thermodynamic Observables in Lattice Field Theories with Deep Generative Models". July 14, 2020. arXiv: 2007.07115 [hep-lat, physics:physics]. URL: http://arxiv.org/abs/2007.07115 (visited on 10/21/2020).

[35] *Computational Complexity of Mathematical Operations*. In: *Wikipedia*. Sept. 6, 2022. URL: https://en.wikipedia.org/w/index.php?title=Computational_complexity_of_mathematical_operations&oldid=1108780149 (visited on 09/17/2022).

[36] Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.2992934. arXiv: 1908.09257. URL: http://arxiv.org/abs/1908.09257 (visited on 03/24/2021).

[37] Diederik P. Kingma and Prafulla Dhariwal. "Glow: Generative Flow with Invertible 1x1 Convolutions". July 10, 2018. arXiv: 1807.03039 [cs, stat]. URL: http://arxiv.org/abs/1807.03039 (visited on 09/19/2022).

[38] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation". Apr. 10, 2015. arXiv: 1410.8516 [cs]. URL: http://arxiv.org/abs/1410.8516 (visited on 01/07/2021).

[39] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation Using Real NVP". Feb. 27, 2017. arXiv: 1605.08803 [cs, stat]. URL: http://arxiv.org/abs/1605.08803 (visited on 01/07/2021).

[40] Conor Durkan et al. "Neural Spline Flows". Dec. 2, 2019. arXiv: 1906.04032 [cs, stat]. URL: http://arxiv.org/abs/1906.04032 (visited on 01/07/2021).

[41] *Atan2*. In: *Wikipedia*. Mar. 13, 2022. URL: https://en.wikipedia.org/w/index.php?title=Atan2&oldid=1076804922 (visited on 04/14/2022).

[42] Kaiming He et al. *Deep Residual Learning for Image Recognition*. Dec. 10, 2015. arXiv: 1512.03385 [cs]. URL: http://arxiv.org/abs/1512.03385 (visited on 07/06/2022).

[43] Jens Behrmann et al. *Invertible Residual Networks*. May 18, 2019. arXiv: 1811.00995 [cs, stat]. URL: http://arxiv.org/abs/1811.00995 (visited on 08/24/2022).

[44] Jonas Köhler, Andreas Krämer, and Frank Noé. "Smooth Normalizing Flows". Nov. 30, 2021. arXiv: 2110.00351 [physics, stat]. URL: http://arxiv.org/abs/2110.00351 (visited on 03/28/2022).

[45] Michael S. Albergo et al. *Introduction to Normalizing Flows for Lattice Field Theory*. Aug. 6, 2021. arXiv: 2101.08176 [cond-mat, physics:hep-lat]. URL: http://arxiv.org/abs/2101.08176 (visited on 08/05/2022).

[46] Jonas Köhler, Leon Klein, and Frank Noé. *Equivariant Flows: Exact Likelihood Generative Learning for Symmetric Densities*. Oct. 26, 2020. arXiv: 2006.02425 [physics, stat]. URL: http://arxiv.org/abs/2006.02425 (visited on 08/12/2022).

[47] Danilo Jimenez Rezende et al. *Equivariant Hamiltonian Flows*. Sept. 30, 2019. arXiv: 1909.13739 [cs, stat]. URL: http://arxiv.org/abs/1909.13739 (visited on 08/12/2022).

[48] Kim A. Nicoli et al. "Asymptotically Unbiased Estimation of Physical Observables with Neural Samplers". In: *Physical Review E* 101.2 (Feb. 10, 2020), p. 023304. ISSN: 2470-0045, 2470-0053. DOI: 10.1103/PhysRevE.101.023304. arXiv: 1910.13496. URL: http://arxiv.org/abs/1910.13496 (visited on 03/19/2022).

[49] David Albandea, Pilar Hernández, and Alberto Ramos. "Luigi Del Debbio Richard Kenway Joe Marsh Rossney". In: *Phys. Rev. D* (2019), p. 24.

[50] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. Dec. 6, 2017. arXiv: 1701.07875 [cs, stat]. URL: http://arxiv.org/abs/1701.07875 (visited on 08/02/2022).

[51] Aditya Grover, Manik Dhar, and Stefano Ermon. *Flow-GAN: Combining Maximum Likelihood and Adversarial Learning in Generative Models*. Jan. 3, 2018. arXiv: 1705.08868 [cs, stat]. URL: http://arxiv.org/abs/1705.08868 (visited on 01/22/2023).

[52] Kanglin Liu et al. *Spectral Regularization for Combating Mode Collapse in GANs*. Oct. 12, 2019. arXiv: 1908.10999 [cs, stat]. URL: http://arxiv.org/abs/1908.10999 (visited on 01/22/2023).

[53] Sam Bond-Taylor et al. "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11 (Nov. 1, 2022), pp. 7327–7347. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2021.3116668. arXiv: 2103.04922 [cs, stat]. URL: http://arxiv.org/abs/2103.04922 (visited on 01/22/2023).

[54] *Linear — PyTorch 1.13 Documentation*. URL: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html (visited on 01/22/2023).

[55] *Conv1d — PyTorch 1.13 Documentation*. URL: https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html (visited on 01/22/2023).

[56] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, Mar. 31, 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html (visited on 01/05/2023).

[57] *Savitzky–Golay Filter*. In: *Wikipedia*. Jan. 7, 2023. URL: https://en.wikipedia.org/w/index.php?title=Savitzky%E2%80%93Golay_filter&oldid=1132105472 (visited on 01/09/2023).

[58] *Optuna - A Hyperparameter Optimization Framework*. Optuna. URL: https://optuna.org/ (visited on 01/10/2023).

[59] Diganta Misra. *Mish: A Self Regularized Non-Monotonic Activation Function*. Aug. 13, 2020. arXiv: 1908.08681 [cs, stat]. URL: http://arxiv.org/abs/1908.08681 (visited on 08/05/2022).

[60] Robert Giaquinto and Arindam Banerjee. "Gradient Boosted Normalizing Flows". Oct. 17, 2020. arXiv: 2002.11896 [cs, stat]. URL: http://arxiv.org/abs/2002.11896 (visited on 03/20/2021).

[61] Gianluigi Silvestri et al. *Embedded-Model Flows: Combining the Inductive Biases of Model-Free Deep Learning and Explicit Probabilistic Modeling*. Mar. 15, 2022. arXiv: 2110.06021 [cs, stat]. URL: http://arxiv.org/abs/2110.06021 (visited on 11/22/2022).

[62] Árpád Baricz. "Bounds for Modified Bessel Functions of the First and Second Kinds". In: *Proceedings of the Edinburgh Mathematical Society* 53.3 (Oct. 2010), pp. 575–599. ISSN: 0013-0915, 1464-3839. DOI: 10.1017/S0013091508001016. URL: https://www.cambridge.org/core/product/identifier/S0013091508001016/type/journal_article (visited on 05/16/2022).

[63] Prakash Balachandran, Weston Viles, and Eric D Kolaczyk. "Exponential-Type Inequalities Involving Ratios of the Modified Bessel Function of the First Kind and Their Applications". In: (), p. 18.

[64] *Pybind11 Documentation*. URL: https://pybind11.readthedocs.io/en/stable/index.html (visited on 01/14/2023).

[65] *Neural Spline Flows*. Bayesiains, Nov. 19, 2022. URL: https://github.com/bayesiains/nsf (visited on 12/19/2022).

[66] *Torch.Logsumexp — PyTorch 1.13 Documentation*. URL: https://pytorch.org/docs/stable/generated/torch.logsumexp.html (visited on 12/06/2022).

[67] *Lmdb — Lmdb 1.3.0 Documentation*. URL: https://lmdb.readthedocs.io/en/release/ (visited on 12/06/2022).

Declaration of Authorship

I hereby declare that my thesis is the result of my own work which has been written only with the help of the indicated sources.

I especially confirm that without any exception, I have fully cited all sources referring to direct quotations of other authors' statements or tables, graphs, quotations as well as all indirect quotations or modified tables, graphics and citations from the internet.

I am aware that any breach against these rules is considered as plagiarism and will be punished according to the general university regulations *(Allgemeine Satzung zur Regelung von Zulassung, Studium und Prüfung Humboldt-Universität zu Berlin, ZSP-HU)*.

This thesis, in same or similar form, has not been submitted to any institution yet.

Signed: 

Date: Berlin, 24.01.23