Principal Component Flows

Edmond Cunningham [1 2] **Adam Cobb** [2] **Susmit Jha** [2]

Abstract

Normalizing flows map an independent set of latent variables to their samples using a bijective transformation. Despite the exact correspondence between samples and latent variables, their high level relationship is not well understood. In this paper we characterize the geometric structure of flows using principal manifolds and understand the relationship between latent variables and samples using contours. We introduce a novel class of normalizing flows, called principal component flows (PCF), whose contours are its principal manifolds, and a variant for injective flows (iPCF) that is more efficient to train than regular injective flows. PCFs can be constructed using any flow architecture, are trained with a regularized maximum likelihood objective and can perform density estimation on all of their principal manifolds. In our experiments we show that PCFs and iPCFs are able to learn the principal manifolds over a variety of datasets. Additionally, we show that PCFs can perform density estimation on data that lie on a manifold with variable dimensionality, which is not possible with existing normalizing flows.

1. Introduction

A normalizing flow is a generative model that generates a probability distribution by transforming a simple base distribution into a target distribution using a bijective function (Rezende & Mohamed, 2015; Papamakarios et al., 2019). Despite the fact that flows can compute the log likelihood of their samples exactly and associate a point in the data space with a unique point in the latent space, they are still poorly understood as generative models. This poor understanding stems from the unidentifiablity of their latent space - the latent space of a flow can be transformed into another valid latent space using any one of an infinite number of volume preserving transformations (Hyvärinen & Pajunen, 1999). Furthermore, methods that are rooted in mutual information (Alemi et al., 2018a; Higgins et al., 2017; Chen et al., 2016) that are used to understand other kinds of generative models break down when applied to flows because there is a deterministic mapping between the latent and data space (Ardizzone et al., 2020). To understand the generative process of flows, we need two items. The first is a set of informative structural properties of a probability distribution and the second is knowledge of how changes to the latent variables affect corresponding samples. We discuss the former using the concept of principal manifolds and the latter using contours.

The principal manifolds of a probability distribution can be understood as manifolds that span directions of maximum change (Gorban et al., 2008b). We locally define them using principal components which are the orthogonal directions of maximum variance around a data point, the same way that the principal components used in PCA (Jolliffe, 2011) are the orthogonal directions of maximum variance of a Gaussian approximation of a dataset. Principal manifolds capture the geometric structure of a probability distribution and are also an excellent fit for normalizing flows because it is possible to compute the principal components of a flow due to the bijective mapping between the latent and data spaces (see Definition 1).

The relationship between changes to latent variables and the effect on the corresponding sample in the data space can be understood through the contours of a flow. The contours of a flow are manifolds that trace the path that a sample can take when only some latent variables are changed. An important relationship we investigate is how the probability density on a contour relates to the probability density under the full model. This insight gives us a novel way to reason about how a flow assigns density to its samples.

We introduce a class of normalizing flows called principal component flows (PCFs) whose contours are its principal manifolds. We develop deep insights into the generative behavior of normalizing flows that help explain how flows assign density to data points. This directly leads to a novel test time algorithm for density estimation on manifolds that requires no assumptions about the underlying data dimen-

[1]University of Massachusetts [2]SRI International. Correspondence to: Edmond Cunningham <edmondcunnin@cs.umass.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

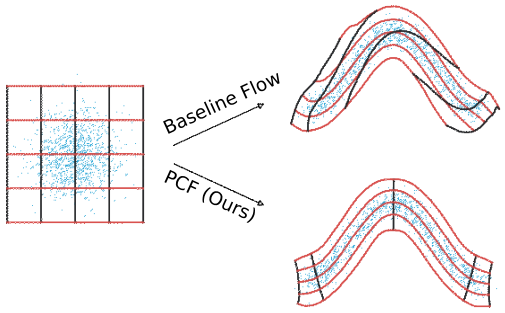


Figure 1. A principal component flow (PCF) has contours that are its principal manifolds while standard normalizing flows do not. The blue dots represent samples from the prior and from each model, on the left and right plots respectively. The red and black lines represent the contours that emerge when a latent variable is held constant and the others vary. Movement in the latent space of a PCF corresponds to movement along the principal manifolds (Theorem 1).

sionality. Furthermore, we develop two new algorithms that tackle separate important problems. The first is a learning algorithm to train PCFs using any flow architecture, even those that can be difficult to invert, at a comparable cost to standard maximum likelihood. The second is an algorithm to train injective PCFs that optimize a regularized maximum likelihood objective without needing to optimize a computationally expensive term found in the injective change of variables formula (Gemici et al., 2016). In our experiments we demonstrate the capabilities of PCFs by learning the principal manifolds of low dimensional data and high dimensional data that are embedded on a low dimensional manifold, and show that PCFs can learn the density of data that lies on a variable dimensional manifold - a task not possible using existing flow based methods. To summarize, our contributions are as follows:

1. We introduce a novel class of flows called PCFs whose contours are principal manifolds and propose an efficient learning algorithm.

2. To overcome the computational cost of computing the expensive Jacobian determinant for PCFs, we introduce iPCFs as an approach for extending PCFs to higher dimensional problems.

3. We introduce the first flow based solution to learning densities on manifolds with varying dimensionality.

2. Preliminaries

2.1. Normalizing flows

Let $f : \mathcal{Z} = \mathbb{R}^N \rightarrow \mathcal{X} = \mathbb{R}^N$ be a parametric bijective function from a latent variable, $\mathbf{z}$, to a data point $\mathbf{x} = f(\mathbf{z})$ with inverse $g(\mathbf{x}) = f^{-1}(\mathbf{x})$. The prior distribution over $\mathbf{z}$ will be denoted with $p_{\mathbf{z}}(\mathbf{z})$ and the Jacobian matrix of f and g will be denoted by $J = \frac{df(\mathbf{z})}{d\mathbf{z}}$ and $G = \frac{dg(\mathbf{x})}{d\mathbf{x}}$. The dependence of J and G on $\mathbf{z}$ or $\mathbf{x}$ is implied. A normalizing flow is a model that generates data by sampling $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ and then computing $\mathbf{x} = f(\mathbf{z})$ (Rezende & Mohamed, 2015; Papamakarios et al., 2019). The probability density of data points is computed using the change of variables formula:

$$\log p_{\mathbf{x}}(\mathbf{x}) = \log p_{\mathbf{z}}(g(\mathbf{x})) + \log |G| \tag{1}$$

Flows are typically comprised of a sequence of invertible functions $f = f_1 \cdots f_i \cdots f_K$ where each f_i has a Jacobian determinant that is easy to compute so that the overall Jacobian determinant, $\log |G| = \sum_{i=1}^{K} \log |G_i|$ is also easy to compute. As a result, flows can be trained for maximum likelihood using an unbiased objective.

Eq. (1) can be generalized to the case where f is an injective function that maps from a low dimensional $\mathbf{z}$ to a higher dimensional $\mathbf{x}$ (Gemici et al., 2016; Caterini et al., 2021). The change of variables formula in this case is written as:

$$\log p_{\mathbf{x}}(\mathbf{x}) = \log p_{\mathbf{z}}(\mathbf{z}) - \frac{1}{2} \log |J^T J|, \quad \mathbf{x} = f(\mathbf{z}) \tag{2}$$

This general change of variables formula is valid over the manifold defined by $f(\mathcal{Z})$. However, it is difficult to work with because the term $\log |J^T J|$ cannot be easily decomposed into a sequence of simple Jacobian determinants as in the case where $\dim(\mathbf{z}) = \dim(\mathbf{x})$. In the remainder of this paper, $\log p_{\mathbf{x}}(\mathbf{x})$ will refer to the definition given in Eq. (2) unless stated otherwise.

The requirement that f is bijective is a curse and a blessing. The constraint prohibits flows from learning probability distributions with topological properties that do not match that of the prior (Cornish et al., 2019). This constraint limits a flow's ability to learn the exact distribution of many real world datasets, including those that are thought to satisfy the manifold hypothesis (Fefferman et al., 2013). Nevertheless, invertibility makes it possible to compute the exact log likelihood under the model, associate any data point with a unique latent space vector, and affords access to geometric properties of the flow's distribution (Dombrowski et al., 2021). In the remainder of this paper we focus on the latter - the geometric properties of a flow's distribution through the use of principal manifolds and contours.

2.2. Principal components of a flow

The structure of a probability distribution that is generated by a normalizing flow can be locally defined by examining

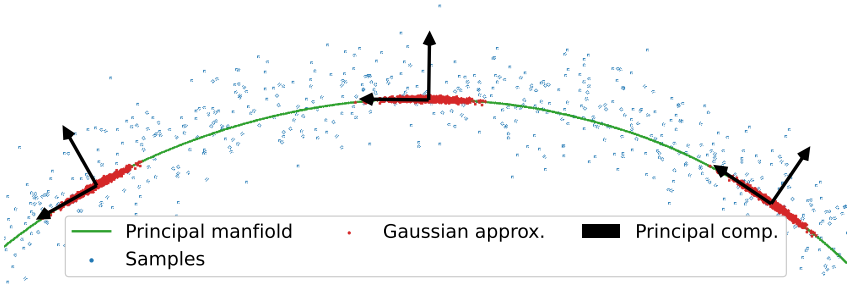


Figure 2. Illustration of principal components and a principal manifold of a flow. The blue dots are samples from a normalizing flow, red dots are samples from the Gaussian approximation defined in Definition 1, black arrows are the principal components and the green line is a principal manifold in Definition 2. Our main result in Theorem 1 states that the contours of principal component flows are its principal manifolds.

how samples from the model are distributed around a data point.

Lemma 1. *Let* $\mathbf{x}$ *be a data point,* f *be the invertible function of a flow and* $\sigma > 0$ *be a scalar. Consider samples that are generated by* $\mathbf{x}' = f(\mathbf{z} + \sigma\epsilon)$ *where* $\mathbf{z} = f^{-1}(\mathbf{x})$ *and* $\epsilon \sim N(0, I)$*. Then* $\frac{1}{\sigma}(\mathbf{x}' - \mathbf{x}) \xrightarrow{D} N(0, JJ^T)$ *as* $\sigma \to 0$*.*

The lemma is true as a direct consequence of the Delta method (Oehlert, 1992) which describes the distribution of a function of an asymptotically normal estimator. Lemma 1 says that points generated by a flow in a small region around a fixed point $\mathbf{x}$ will be approximately distributed as a Gaussian with mean $\mathbf{x}$ and covariance proportional to JJ^T. The principal components of data generated by a Gaussian distribution are the eigenvectors of the covariance matrix, so we can use the eigenvectors of JJ^T to define the principal components of a flow.

Definition 1 (Principal components of a flow at x)**.** *The principal components of a flow at* $\mathbf{x} = f(\mathbf{z})$ *are the eigenvectors of* JJ^T*,* $\hat{\mathbf{w}}$*, where* $J = \frac{df(\mathbf{z})}{d\mathbf{z}}$*. The principal components are ordered according to the eigenvalues of* JJ^T*.*

The concept of principal components is shown in Fig. 2. Blue dots represent samples from a flow, red dots are samples from the local approximations drawn according to Lemma 1 and black arrows represent the principal components computed using Definition 1. We see that the red dots are approximately distributed as a Gaussian and the black arrows span their principal directions. Furthermore, the principal components are oriented along the main structure of the data. The global structure of a flow, which we call the "principal manifolds", are found by integrating along the principal components.

Definition 2 (Principal manifold of a flow)**.** *The principal manifold of a flow is the path formed by integrating along principal components starting at* $\mathbf{x}_0$*. Let* $\mathbb{K}$ *be a subset of* $[1, \ldots, \dim(\mathbf{z})]$ *and* $t \in \mathbb{R}^{|\mathbb{K}|}$*. A principal manifold of dimension* $|\mathbb{K}|$ *is the solution to*

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{w}_{\mathbb{K}}(\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{3}$$

where $\mathbf{w}_{\mathbb{K}}(\mathbf{x}(t)) = \hat{\mathbf{w}}_{\mathbb{K}}\sqrt{\Lambda_{\mathbb{K}}}$ *are the principal components with indices in* $\mathbb{K}$ *at* $\mathbf{x}(t)$ *scaled by the square root of their corresponding eigenvalues.*

The green curve in Fig. 2 is one of an infinite number of principal manifolds of the distribution. It spans the main structure of the samples and has principal component tangents. Principal manifolds can be used to reason about the geometric structure of a flow, but can only be found via integration over the principal components. Furthermore, there is no clear way compute the probability density over the principal manifolds. This is crucial when a principal manifold is used as a low dimensional representation of data and we still want to perform density estimation. We will revisit principal manifolds in Section 3.

2.3. Contours of a normalizing flow

The tool we use to analyze the generative properties of flows are the contours that emerge when some latent variables are held constant while others vary.

Definition 3 (Contours of a flow)**.** *Let* $\mathbb{K}$ *be a subset of* $[1, \ldots, \dim(\mathbf{z})]$ *and* $\mathbf{z}_{\mathbb{K}}$ *be the latent variables with indices in* $\mathbb{K}$*. Then, the contour obtained by varying* $\mathbf{z}_{\mathbb{K}}$ *and fixing all other variables is denoted as* $f_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})$*.*

We assume that there is a partition over the indices of the latent space, $\mathcal{P}$, so that every set of indices that we use to form contours is an element of the partition: $\mathbb{K} \in \mathcal{P}$. Additionally, we assume that the prior over $\mathbf{z}$ can be factored into independent components in order to isolate a prior for each contour: $p_{\mathbf{z}}(\mathbf{z}) = \prod_{\mathbb{K} \in \mathcal{P}} p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})$. This is not a limiting assumption as most flow architectures typically use a fully factorized prior such as a unit Gaussian prior (Papamakarios et al., 2019). The curved red and black lines in the right

side plots of Fig. 1 are examples of contours. A red line on the left side plot is created by varying $\mathbf{z}_1$ and fixing $\mathbf{z}_2$ and becomes the contour $f_1(\mathbf{z}_1)$ after it is passed through the flow. Similarly, a black line on the left side plot is formed by varying $\mathbf{z}_2$ and fixing $\mathbf{z}_1$ and becomes the contour $f_2(\mathbf{z}_2)$ when it is transformed by the flow. The Jacobian matrix of $f_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})$ is denoted by $J_{\mathbb{K}}$ and is equal the matrix containing the columns of J with indices in $\mathbb{K}$. The log likelihood of a contour is denoted by $\mathcal{L}_{\mathbb{K}}$ and is computed using the change of variables formula on manifolds in Eq. (2):

$$\mathcal{L}_{\mathbb{K}} \triangleq \log p(f_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})) = \log p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}}) - \frac{1}{2}\log|J_{\mathbb{K}}^T J_{\mathbb{K}}| \quad (4)$$

A single flow can assign many different log likelihoods to a given data point that are not given by Eq. (1). Each of the $|\mathcal{P}|$ contours that intersect at a data point can assign a density given by Eq. (4). Furthermore, the contours formed by grouping multiple $\mathbf{z}_{\mathbb{K}}$ will assign other densities. In the next section we will relate all of these different densities.

2.4. Pointwise mutual information between contours

Consider two disjoint subsets of a latent variable, $\mathbf{z}_{\mathbb{S}}$ and $\mathbf{z}_{\mathbb{T}}$, and their union $\mathbf{z}_{\mathbb{S}+\mathbb{T}}$. The densities of each contour can be related to the densities of their union using pointwise mutual information.

Definition 4 (Pointwise mutual information between disjoint contours)**.** *Let $\mathbf{z}_{\mathbb{S}}$ and $\mathbf{z}_{\mathbb{T}}$ be disjoint subsets of z. The pointwise mutual information between the contours for $\mathbf{z}_{\mathbb{S}}$ and $\mathbf{z}_{\mathbb{T}}$ is defined as*

$$\mathcal{I}_{\mathbb{S},\mathbb{T}} \triangleq \log \frac{p(f_{\mathbb{S}+\mathbb{T}}(\mathbf{z}_{\mathbb{S}+\mathbb{T}}))}{p(f_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}}))p(f_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}}))} = \mathcal{L}_{\mathbb{S}+\mathbb{T}} - \mathcal{L}_{\mathbb{S}} - \mathcal{L}_{\mathbb{T}} \quad (5)$$

$\mathcal{I}_{\mathbb{S},\mathbb{T}}$ plays an important role in describing the behavior of normalizing flows. We list a few important facts about $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ below to help build intuition (more facts and proofs can be found in Appendix B):

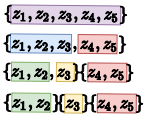
Facts about $\mathcal{I}_{\mathbb{S},\mathbb{T}}$

1. $\mathcal{I}_{\mathbb{S},\mathbb{T}} \geq 0$
2. $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$ iff $f_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})$ and $f_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})$ intersect orthogonally.
3. $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2}\log|J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| + \frac{1}{2}\log|J_{\mathbb{S}}^T J_{\mathbb{S}}| + \frac{1}{2}\log|J_{\mathbb{T}}^T J_{\mathbb{T}}|$

$\mathcal{I}_{\mathbb{S},\mathbb{T}}$ is a non-negative value that achieves its minimum of 0 when the contours intersect orthogonally. An equivalent condition is that the columns of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ are mutually orthogonal. The third fact shows that $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ has a closed form value that only depends on f and not on the priors over $\mathbf{z}_{\mathbb{S}}$ or $\mathbf{z}_{\mathbb{T}}$. Another way to think about $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ is as the difference between the contour log likelihoods and that of their union:

$$\mathcal{L}_{\mathbb{S}+\mathbb{T}} = \mathcal{L}_{\mathbb{S}} + \mathcal{L}_{\mathbb{T}} + \mathcal{I}_{\mathbb{S},\mathbb{T}} \quad (6)$$

If $\mathbb{S}+\mathbb{T} = [1,\ldots,\dim(\mathbf{z})]$, then we can decompose the change of variables formula into the sum of contour log likelihoods and $\mathcal{I}_{\mathbb{S},\mathbb{T}}$. Next, we show that this decomposition can be extended to any partition of the latent space.



(a) Example partitions of the latent space.

(b) Decomposition of log likelihood.

Figure 3. The general change of variables decomposition depends on the binary tree partition used to generate the latent space partition. Partitions are generated by recursively dividing existing partitions into two parts. The last row in Fig. 3a shows a partition of the latent space with 3 sets. Fig. 3b shows the corresponding log likelihood decomposition. Each parent node in the binary tree contributes an $\mathcal{I}$ term and each leaf contributes a $\mathcal{L}$ term. See Eq. (7) for the full formula.

2.5. Change of variables formula decomposition

Eq. (6) can be recursively applied to itself to yield a decomposition over any partition of the latent space. Consider a partition of the indices, $\mathcal{P}$, like in Fig. 3a. $\mathcal{P}$ can be constructed as the leaves of a binary tree, $\mathcal{T}$, where each node is a subset of indices and each parent node is the union of its children. The corresponding decomposition in Fig. 3b is found by recursively applying Eq. (6) and tracking the leftover $\mathcal{I}$ terms. This construction lets us decompose the change of variables formula into the sum of contours log likelihoods and pointwise mutual information terms:

$$\log p_{\mathbf{x}}(f(\mathbf{z})) = \sum_{\mathbb{K}\in\mathcal{P}} \mathcal{L}_{\mathbb{K}} + \underbrace{\sum_{\mathbb{P}\in\text{parents}(\mathcal{T})} \mathcal{I}_{\mathrm{L}(\mathbb{P}),\mathrm{R}(\mathbb{P})}}_{\mathcal{I}_{\mathcal{P}}} \quad (7)$$

where $\mathrm{L}(p)$ and $\mathrm{R}(p)$ are the left and right children of $\mathbb{P}$, respectively. See Fig. 3 for a visual description. Notice that the sum of the various $\mathcal{I}_{\mathrm{L}(\mathbb{P}),\mathrm{R}(\mathbb{P})}$ is independent of the choice of $\mathcal{T}$ because any $\mathcal{T}$ with the same leaves will have the same value of $\log p_{\mathbf{x}}(f(\mathbf{z}))$ and $\sum_{\mathbb{K}\in\mathcal{P}} \mathcal{L}_{\mathbb{K}}$. This non-negative quantity is useful to know as it is the difference between the full log likelihood and sum of log likelihoods of the contours.

Definition 5 (Pointwise mutual information of a partition)**.** *Let $\mathcal{P}$ be a partition of $[1,\ldots,\dim(\mathbf{z})]$. The pointwise mutual information of the flow whose latent space is partitioned*

by $\mathcal{P}$ is

$$\mathcal{I}_{\mathcal{P}} \triangleq \log p_{\mathbf{x}}(f(\mathbf{z})) - \sum_{\mathbb{K}\in\mathcal{P}} \mathcal{L}_{\mathbb{K}} \tag{8}$$

Eq. (7) gives insight into how normalizing flows assign density to data. The log likelihood of a data point under a normalizing flow is the sum of the log likelihoods under its contours, and a non-negative term that roughly measures the orthogonality of the contours. If during training Eq. (7) is maximized, as is the case in maximum likelihood learning, then $\mathcal{I}_{\mathcal{P}}$ will surely not achieve its minimum value of 0. Therefore changes to different latent variables of a normalizing flow trained with maximum likelihood will likely produce similar changes in the data space.

2.6. Orthogonality condition using g

We have seen that $\mathcal{I}_{\mathcal{P}}$ is a non-negative term that achieves it minimum of 0 when the contours of the flow are orthogonal. However obtaining its value requires computing columns of the Jacobian matrix of $f(\mathbf{z})$. Many expressive normalizing flows layers (Huang et al., 2021; Chen et al., 2019; van den Berg et al., 2018) are constructed so that only $g(\mathbf{x})$ is easy to evaluate while the inverse $f(\mathbf{z})$ requires an expensive algorithm that can be difficult to differentiate. We introduce a novel alternate formulation of $\mathcal{L}_{\mathbb{K}}$, $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ and $\mathcal{I}_{\mathcal{P}}$ that can be computed with $g(\mathbf{x})$ to mitigate this issue:

$$\widehat{\mathcal{L}}_{\mathbb{K}} \triangleq \log p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}}) + \frac{1}{2}\log|G_{\mathbb{K}}G_{\mathbb{K}}^T| \tag{9}$$

$$\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} \triangleq \widehat{\mathcal{L}}_{\mathbb{S}+\mathbb{T}} - \widehat{\mathcal{L}}_{\mathbb{S}} - \widehat{\mathcal{L}}_{\mathbb{T}} \tag{10}$$

$$\widehat{\mathcal{I}}_{\mathcal{P}} \triangleq \log p_{\mathbf{x}}(\mathbf{x}) - \sum_{\mathbb{K}\in\mathcal{P}} \widehat{\mathcal{L}}_{\mathbb{K}} \tag{11}$$

In contrast to $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ and $\mathcal{I}_{\mathcal{P}}$, $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}}$ and $\widehat{\mathcal{I}}_{\mathcal{P}}$ are both negative $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}}, \widehat{\mathcal{I}}_{\mathcal{P}} \leq 0$. See Appendix B for more properties. The most important of these properties is the following Lemma:

Lemma 2. *$\widehat{\mathcal{I}}_{\mathcal{P}} = 0$ if and only if $\mathcal{I}_{\mathcal{P}} = 0$.*

We will see that flows that satisfy $\mathcal{I}_{\mathcal{P}} = 0$ are of interest, so this lemma provides an equivalent condition that can be computed by any normalizing flow architecture.

3. Principal Component Flows

We now present our main contributions. We will first define PCFs and discuss their theoretical properties then discuss learning algorithms to train PCFs and injective PCFs (iPCF).

3.1. PCF theory

Next, we formally define PCFs and provide a theorem stating their primary feature (see Theorem 2 for the proof).

Definition 6 (principal component flow)**.** *A principal component flow (PCF) is a normalizing flow that satisfies $\mathcal{I}_{\mathcal{P}} = 0$ at all of its samples.*

Theorem 1 (Contours of PCFs)**.** *The contours of a principal component flow are principal manifolds.*

A compelling byproduct of Theorem 1 is that PCFs can easily evaluate the probability density of their principal manifolds. As a result, PCFs can perform density estimation on manifolds at test time without making any assumptions about the dimensionality of the data manifold. This is in stark contrast to existing flow based algorithms for density estimation on manifolds where the manifold dimensionality is fixed when the flow is created (Brehmer & Cranmer, 2020). In order to exploit this ability, we need a method to identify which contours correspond to which principal manifolds. Recall that the principal components are ordered according to the eigenvalues of JJ^T. Consider a PCF where the partition size is 1. Then the diagonal elements of J^TJ will be equal to the top $\dim(\mathbf{z})$ eigenvalues of JJ^T because J will be the product of a semi-orthogonal matrix and a diagonal matrix (see Item 7), so J^TJ will be diagonal and its diagonal elements of J^TJ can be used to identify which contour corresponds to which principal manifold. In the general case, J^TJ is a block diagonal matrix so we can look at the square root of the determinant of each block matrix, $|J_{\mathbb{K}}^TJ_{\mathbb{K}}|^{\frac{1}{2}}$. $|J_{\mathbb{K}}^TJ_{\mathbb{K}}|^{\frac{1}{2}}$ is how much the density around $\mathbf{x}$ is "stretched" along the contour $f_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})$ to form the structure present in the data, so contours with small values of $|J_{\mathbb{K}}^TJ_{\mathbb{K}}|^{\frac{1}{2}}$ correspond to a direction that contributes little to the overall structure.

This check can be used filter out components of log likelihood that are due to small variations such as noise incurred in the data collection process. For example, consider a PCF trained on 2D data and at $\mathbf{x}$ we observe that $\log|J_1^TJ_1| \gg \log|J_2^TJ_2|$. Then the structure of the probability distribution at $\mathbf{x}$ should be primarily aligned with the contour $f_1(\mathbf{z}_1)$, so it might make sense to report the log likelihood of $\mathbf{x}$ on only this contour. We define this procedure below:

Definition 7 (Manifold corrected probability density)**.** *Let $\mathbf{x}$ be a sample from a PCF. The manifold corrected probability density of $\mathbf{x}$ is computed as*

$$\log p_{\mathcal{M}}(\mathbf{x}) = \sum_{\mathbb{K}\in\mathcal{S}} \mathcal{L}_{\mathbb{K}}, \quad \mathcal{S} = \{\mathbb{K} : |J_{\mathbb{K}}^TJ_{\mathbb{K}}|^{\frac{1}{2}} > \epsilon\} \tag{12}$$

In experiment Section 5.2 we demonstrate an example where PCFs correctly learns the density of data generated on a variable dimension manifold.

3.2. Learning algorithms

PCF objective

The PCF optimization problem is to minimize the negative log likelihood of data subject to the constraint that $\mathcal{I}_{\mathcal{P}} = 0$:

$$\text{argmin}_\theta - \sum_{\mathbf{x}\in\mathcal{D}} \log p_{\mathbf{x}}(\mathbf{x};\theta), \quad \text{s.t.} \quad \mathcal{I}_{\mathcal{P}}(\mathbf{x};\theta) = 0 \tag{13}$$

We solve this problem with a regularized maximum likelihood objective:

$$\begin{aligned} \mathrm{L}(\theta) &= \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{x}}(\mathbf{x};\theta) + \alpha\mathcal{I}_{\mathcal{P}}(\mathbf{x};\theta) \\ &= \sum_{\mathbf{z}=g(\mathbf{x}),\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(\mathbf{z}) - \frac{\alpha-1}{2}\log|J(\mathbf{z};\theta)^T J(\mathbf{z};\theta)| \\ &\quad + \frac{\alpha}{2}\sum_{\mathbb{K}\in\mathcal{P}} \log|J_{\mathbb{K}}(\mathbf{z};\theta)^T J_{\mathbb{K}}(\mathbf{z};\theta)| \end{aligned} \tag{14}$$

where α is a hyperparameter. Note that in the special case where the partition $\mathbb{K}$ is 1 dimensional and $\dim(\mathbf{x}) = \dim(\mathbf{z})$, this is the objective function used in (Gresele et al., 2021). Although $\mathrm{L}(\theta)$ is a valid objective, it requires that we compute $g(\mathbf{x})$ and a matrix vector product with the inverse Jacobian, making it impractical for flows where the inverse is difficult to evaluate. We remedy this issue by replacing the constraint $\mathcal{I}_{\mathcal{P}} = 0$ with $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$ as per Lemma 2. The result is a novel loss function for training flows to have orthogonal contours:

$$\begin{aligned} \mathrm{L}_{\text{PCF}}(\theta) &= \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{x}}(\mathbf{x};\theta) - \alpha\widehat{\mathcal{I}}_{\mathcal{P}}(\mathbf{x};\theta) \\ &= \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(g(\mathbf{x};\theta)) - \frac{\alpha+1}{2}\log|G(\mathbf{x};\theta)G(\mathbf{x};\theta)^T| \\ &\quad + \frac{\alpha}{2}\sum_{\mathbb{K}\in\mathcal{P}} \log|G_{\mathbb{K}}(\mathbf{x};\theta)G_{\mathbb{K}}(\mathbf{x};\theta)^T| \end{aligned} \tag{15}$$

$\mathrm{L}_{\text{PCF}}(\theta)$ is the objective of choice when $\dim(x) = \dim(z)$. It provides a lightweight change to maximum likelihood training that can be applied to any flow architecture.

iPCF objective

Next consider the case where $\dim(\mathbf{x}) > \dim(\mathbf{z})$. This appears in problems where we want to learn a low dimensional representation of data (Gemici et al., 2016; Brehmer & Cranmer, 2020; Caterini et al., 2021; Kumar et al., 2020). Although we can optimize $\mathrm{L}(\theta)$ to learn a PCF, naively optimizing Eq. (14) will require optimizing $\log|J^T J|$, which requires $\dim(\mathbf{z})$ Jacobian-vector products or an iterative algorithm (Caterini et al., 2021). We avoid this problem by setting $\alpha = 1$ in Eq. (14). This yields the iPCF objective:

$$\begin{aligned} \mathrm{L}_{\text{iPCF}}(\theta) &= \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{x}}(\mathbf{x};\theta) + \mathcal{I}_{\mathcal{P}}(\mathbf{x};\theta) \\ &= \sum_{\mathbf{z}=g(\mathbf{x}),\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(\mathbf{z}) + \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}} \log|J_{\mathbb{K}}^T J_{\mathbb{K}}| \end{aligned} \tag{16}$$

The iPCF objective is a novel lower bound on the log likelihood of a dataset that lies on a manifold. Clearly the bound is tight when $\mathcal{I}_{\mathcal{P}}(\mathbf{x};\theta) = 0$, so the learned model must trade off how close its contours are to principal manifolds with how well it represents data. The computational bottleneck of $\mathrm{L}_{\text{iPCF}}(\theta)$ is the $\log|J_{\mathbb{K}}^T J_{\mathbb{K}}|$ terms, which each require $|\mathbb{K}|$ Jacobian-vector products to compute. However, if $|\mathbb{K}| \ll \dim(\mathbf{z})$, then $\mathrm{L}_{\text{iPCF}}(\theta)$ is much more efficient to estimate than $\mathrm{L}(\theta)$ (see the next paragraph on unbiased estimates). Eq. (16) on its own cannot be used for training because there are no guarantees that training data will satisfy the condition $\mathbf{x} = f(\mathbf{z})$. Instead, we plug $\mathrm{L}_{\text{iPCF}}(\theta)$ into the algorithm described in section 4 of (Caterini et al., 2021). This algorithm projects training data onto the generative manifold and maximizes the likelihood of the projected data, while also minimizing the reconstruction error. See appendix Appendix D.3 for a full description.

Unbiased estimates of the objectives

In practice, we implement Eq. (15) and Eq. (16) by randomly selecting $\mathbb{K} \in \mathcal{P}$, constructing $|\mathbb{K}|$ one-hot vectors where each vector has a single 1 at an index in $\mathbb{K}$, and multiplying each vector with the transposed Jacobian (vjp) at $g(\mathbf{x})$ or Jacobian (jvp) at $f(\mathbf{z})$. If each $\mathbf{z}_{\mathbb{K}}$ is 1 dimensional, then the PCF objective only requires a single vjp or jvp. This means that the cost of training a PCF is only slightly more expensive than training a regular normalizing flow and the cost of training an iPCF is much more efficient than training an injective normalizing flow. We provide Python code in appendix Appendix A.

4. Related Work

Our work plugs a methodological gap in the normalizing flows (Papamakarios et al., 2019; Rezende & Mohamed, 2015) related to finding structure within flows. Although this is not crucial for applications such as density estimation or Neural-transport MCMC (Hoffman et al., 2019), the success of approaches in other deep generative models for finding low dimensional structure such as the β-VAE (Higgins et al., 2017; Alemi et al., 2018b) and Style GAN (Karras et al., 2019) are motivation to find structure in normalizing flows. A subarea of flows research focuses on learning densities on manifolds. Gemici et al. (2016) introduced a proof of concept for learning a density over a specified manifold and since then other methods have extended the idea to other kinds of manifolds such as toris, spheres and hyperbolic spaces (Rezende et al., 2020; Bose et al., 2020).

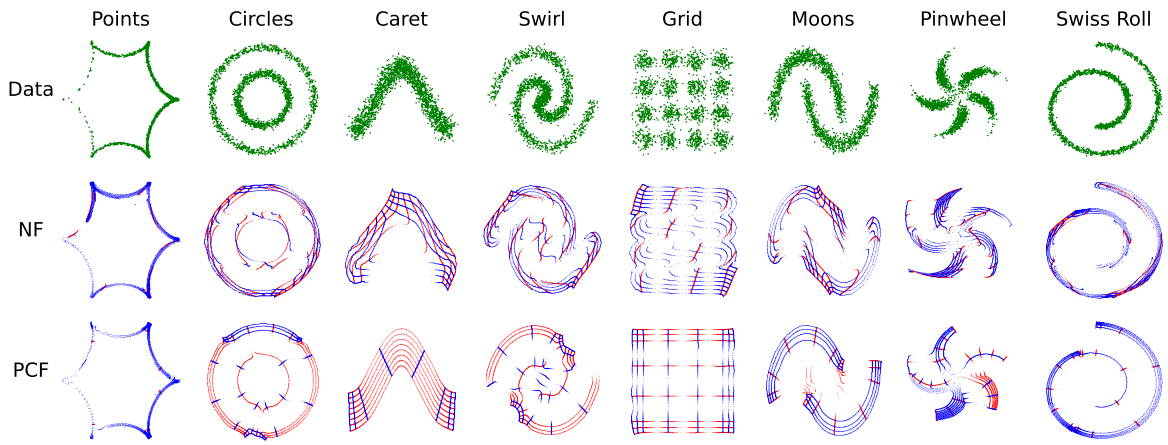


Figure 4. Contours for various synthetic datasets from a normalizing flow (NF) and principal component flow (PCF). Both flows learned to produce the correct samples (see Appendix D.1) but only the PCF learns the data's structure.

		Points	Circles	Caret	Swirl	Grid	Moons	Pinwheel	Swiss Roll
$\log p(x)(\uparrow)$	NF	-1.60	-3.10	-1.89	-0.19	-6.02	-0.64	-3.28	-4.67
	PCF	-1.62	-3.12	-1.89	-0.20	-6.02	-0.66	-3.29	-4.68
$\mathcal{I}_{\mathcal{P}}(\downarrow)$	NF	1.60	1.18	0.61	0.71	0.39	0.64	0.77	1.38
	PCF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 1. Numerical results for learning synthetic datasets. The PCF obtains a similar test set log likelihood to that of the normalizing flow (NF), but only the PCF has small pointwise mutual information ($\mathcal{I}_{\mathcal{P}}$). Small values of $\mathcal{I}_{\mathcal{P}}$ result in the orthogonal contours shown in Fig. 4.

A related class of flows are dedicated to both learning manifolds and densities over them (Kumar et al., 2020; Brehmer & Cranmer, 2020; Kalatzis et al., 2021; Caterini et al., 2021; Kothari et al., 2021). Our work is different because we focus on flows with density in the full data space and we do not focus on learning any single manifold. Additionally, there has been work in flows aimed at constructing architectures so that structure can emerge during training (Zhang et al., 2021; Cunningham et al., 2020; Cunningham & Fiterau, 2021), however these methods have no guarantees that they will recover the intended structure whereas PCFs do.

There are other works that impose orthogonality conditions on Jacobian matrices. Conformal embedding flows (Ross & Cresswell, 2021) constructs an injective flow that has an orthogonal times a scalar Jacobian matrix to learn densities over manifolds. Our Jacobian structure is more flexible because it only requires $J^T J$ to be block diagonal. We also note that our method can be used to learn conformal mappings if the regularizer is used on the Jacobian and its transpose. Dombrowski et al. (2021) presents a way to apply flows that have learned the structure of a dataset to generating counterfactuals by using optimization in the latent space, which is shown to adhere to the flow's generative manifold in the data space. Wei et al. (2021) and Gropp et al. (2020) propose regularizers to ensure that the Jacobian matrix of their models are orthogonal. As mentioned earlier, our Jacobian structure is much more flexible. The most similar work to ours is independent mechanism analysis (IMA) (Gresele et al., 2021). IMA is motivated by independent component analysis and causal inference while ours is motivated by uncovering the structure of data. We introduce novel insights on the geometry of flows and the densities on their contours, orthogonality conditions for both injective flows and flows that are not easily invertible, and a test time algorithm for computing densities on manifolds. PCA (Jolliffe, 2011) and its nonlinear extensions (Jolliffe, 2011; Gorban et al., 2008a) have the same goal as PCFs of finding the principal structure of data. Cramer et al. (2021) treat PCA as a linear PCF, but do not consider the nonlinear case. Work related to principal manifolds, such as locally linear embeddings (Ghojogh et al., 2021), differ from ours primarily in that we use parametric functions to learn the geometry of data.

5. Experiments

Our experiments showcase the capabilities of PCFs to learn the principal manifolds of data, perform density estimation on data that is generated on a variable dimensional dataset, and learn high dimensional data embedded on a low dimensional manifold. All of our experiments were written using

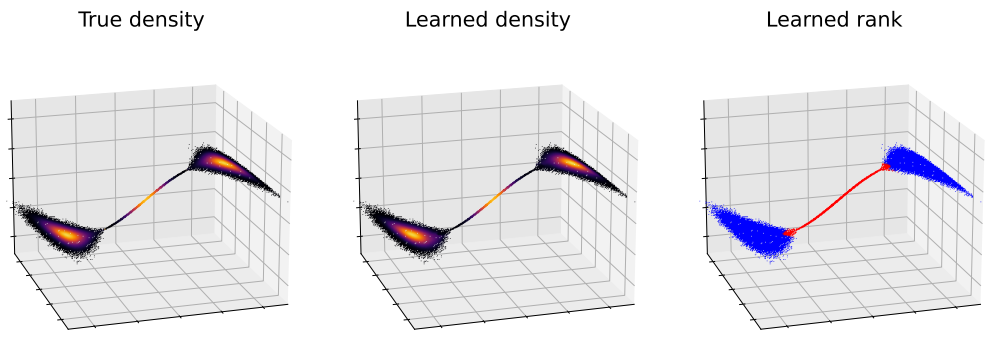


Figure 5. PCFs are the only class of flows that can learn densities on manifolds with variable dimensionality. The dataset is generated on a manifold that is 1D near the origin and 2D elsewhere. At test time, the density for each data point is computed using Definition 7.

the JAX (Bradbury et al., 2018) Python library. We provide extended results and details of our models in Appendix D.

5.1. 2D Synthetic Datasets

We trained standard normalizing flow and PCF on various synthetic 2D datasets. Both flows have an architecture with 10 coupling layers, each with a logistic mixture cdf with 8 components, logit and shift-scale transformer (Ho et al., 2019; Papamakarios et al., 2019) and 5 layer residual network with 64 hidden units conditioner. We applied a matrix vector product and act norm layer in between each coupling layer (Kingma & Dhariwal, 2018). Note that logistic mixture cdfs require an iterative algorithm to invert.

The log likelihood and pointwise mutual information ($\mathcal{I}_{\mathcal{P}}$) of the test sets are shown in Table 1. We see from the likelihoods that the PCF is able to learn the datasets as well as the standard flow while achieving a small value of $\mathcal{I}_{\mathcal{P}}$. The low $\mathcal{I}_{\mathcal{P}}$ is reflected by the contours in Fig. 4. In line with our theory, the contours of the PCF are orthogonal to each other and are oriented in the directions of maximum variance.

5.2. Learning manifold densities of varying rank

PCFs have the unique ability to learn densities on manifolds with unknown rank. All existing density estimation algorithms on manifolds using flows require specifying the dimensionality of the manifold beforehand, but PCFs do not because they will automatically learn the underlying structure of the dataset. The leftmost plot of Fig. 5 shows the target probability distribution whose samples lie on either a 1D or 2D manifold. The data is generated by first sampling two univariate random variables z_1 and z_2 from a Gaussian mixture model and standard Gaussian respectively and then transforming $z = (z_1, z_2)$ to the data space with the equation $x = (z_1, z_2 \max(0, 1 - |\frac{1}{z_1}|), \sin(z_1))$. Notice that x is one dimensional when $|z_1| < 1$ and two dimensional otherwise. During training we perturb the dataset with a small amount of Gaussian noise so that the training data has full rank. See Appendix D.2 for a full description of the data and model and extended results. We use the method described in Definition 7 to compute the rank and density of each data point in the test set. We see from the center plot of Fig. 5 that the PCF correctly recovers the densities of the test data samples. The final forward KL divergence from the learned density and true density is 0.0146.

5.3. iPCF

Here we show that the iPCF learning algorithm does in fact learn an injective flow with contours that are close to principal manifolds, and that the intuition about how contours relate to the principal manifolds does help explain the generative behavior of flows. We trained an iPCF and standard injective normalizing flow (iNF) on the MNIST dataset (Lecun et al., 1998). The iPCF and iNF both had the same architecture consisting of 20 layers of GLOW (Kingma & Dhariwal, 2018), a slice layer that removes all but 10 of the latent dimensions (so that the latent space is 10 dimensional), and then another 10 layers of neural spline flows (Durkan et al., 2019). See appendix Appendix D.3 for details on the model and the training. Note that the iPCF required roughly 10 times less resources to train because we computed a single jvp to estimate Eq. (16) while the iNF required 10 jvps to compute Eq. (2).

Fig. 6a shows a similarity plot between sorted contours of each model and the true principal components. The columns represent the principal components sorted by eigenvalue while the rows represent the tangents of the contours (columns of J) in increasing order of the diagonal of $J^T J$. The intensity of each cell is the average absolute value of the cosine similarity between J and a principal component. The plot of iPCF is highlighted along the diagonal, which indicates that the contours are mostly aligned with the principal components whereas the plot for the iNF is highlighted

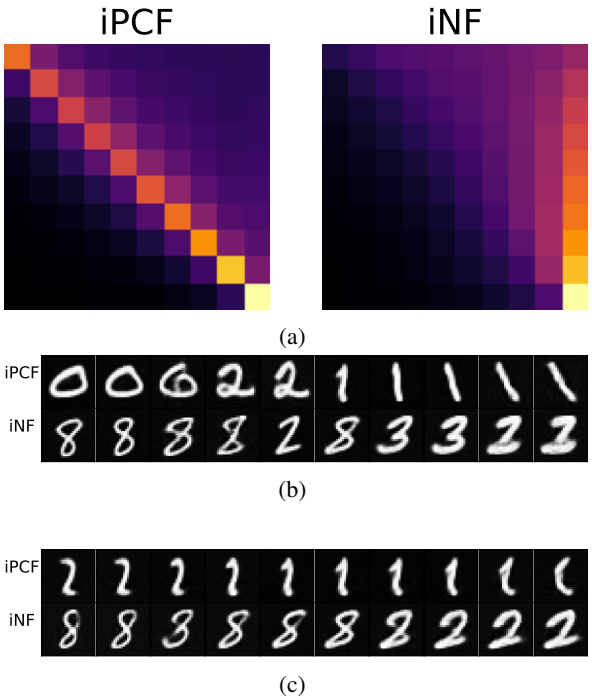


Figure 6. Similarity plot (Fig. 6a) between sorted contours of each model and the true principal components and traversal of the largest (Fig. 6b) and 5th largest (Fig. 6c) contours of the iPCF and iNF. See Section 5.3 for more details.

along the last column, which indicates that the contours are mostly aligned with only the largest principal component.

Fig. 6b and Fig. 6c show a traversal of the largest and 5th largest contours of the iPCF and iNF respectively. We moved along the contours by computing the Jacobian matrix of the flow at the current $\mathbf{z}$, ordering the contours according to the diagonal of $J^T J$, and then taking a step of 0.02 on the dimension of $\mathbf{z}$ corresponding to the contour we want to traverse. We took 500 of these steps and displayed every 50^{th} image in the figures. The images generated on the top contours for both models are varied as expected. The images on the 5th largest contours of the iPCF are only varied slightly, which matches the results from Fig. 6a that the 5th largest contours will be oriented similarly to the 5th largest principal manifold and should therefore result in only a minimal amount of change. The iNF, on the other hand, generates images on the 5th largest contour that are similar to those generated on the largest contour. This also matches the intuition from Fig. 6a that the contours of the iNF are mostly aligned with the largest principal manifold.

6. Conclusion

We introduced principal component flows, a type of normalizing flow whose latent variables generate its principal manifolds. We investigated the generative behavior of flows using principal manifolds and contours to understand how a flow assign probability density to its samples. This analysis helped us define PCFs and develop an efficient general purpose learning algorithm. Furthermore, we found an objective function to train injective PCFs that avoided the need to compute a difficult Jacobian determinant during training. We showed how to interpret the contours of PCFs and proposed a simple test to match a contour with a principal manifold. This test was then shown to help perform density estimation on the true data manifold at test time. Our experiments demonstrated the PCFs are effective tools for learning the principal manifolds of low dimensional data, or high dimensional data that is embedded on a low dimensional manifold, and that PCFs are capable of performing density estimation on data that is generated on a variable dimensional manifold.

Acknowledgments

This material is based upon work supported by U.S. Army Research Laboratory Cooperative Research Agreement W911NF-17-2-0196, U.S. National Science Foundation(NSF) grants #1740079, and the United States Air Force and DARPA under Contract No. FA8750-20-C-0002. The views, opinions and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. Fixing a Broken ELBO. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 159–168. PMLR, July 2018a. URL https://proceedings.mlr.press/v80/alemi18a.html.

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. Fixing a broken ELBO. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 159–168. PMLR, 10–15 Jul 2018b. URL https://proceedings.mlr.press/v80/alemi18a.html.

Ardizzone, L., Mackowiak, R., Rother, C., and Köthe, U. Training Normalizing Flows with the Information Bottleneck for Competitive Generative Classification. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33,

pp. 7828–7840. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/593906af0d138e69f49d251d3e7cbed0-Paper.pdf.

Bose, A. J., Smofsky, A., Liao, R., Panangaden, P., and Hamilton, W. L. Latent variable modelling with hyperbolic normalizing flows. *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brehmer, J. and Cranmer, K. Flows for simultaneous manifold learning and density estimation. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 442–453. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/051928341be67dcba03f0e04104d9047-Paper.pdf.

Caterini, A. L., Loaiza-Ganem, G., Pleiss, G., and Cunningham, J. P. Rectangular Flows for Manifold Learning. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. URL https://openreview.net/forum?id=s-Fg3dXQzyS.

Chen, R. T. Q., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. Residual Flows for Invertible Generative Modeling. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5d0d5594d24f0f955548f0fc0ff83d10-Paper.pdf.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial nets. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2180–2188, 2016.

Cornish, R., Caterini, A. L., Deligiannidis, G., and Doucet, A. Relaxing bijectivity constraints with continuously indexed normalising flows, 2019.

Cramer, E., Mitsos, A., Tempone, R., and Dahmen, M. Principal component density estimation for scenario generation using normalizing flows. *arXiv preprint arXiv:2104.10410*, 2021.

Cunningham, E. and Fiterau, M. A change of variables method for rectangular matrix-vector products. In Banerjee, A. and Fukumizu, K. (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 2755–2763. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/cunningham21a.html.

Cunningham, E., Zabounidis, R., Agrawal, A., Fiterau, I., and Sheldon, D. Normalizing Flows Across Dimensions. *arXiv:2006.13070 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2006.13070. arXiv: 2006.13070.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

Dombrowski, A.-K., Gerken, J. E., and Kessel, P. Diffeomorphic explanations with normalizing flows. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. URL https://openreview.net/forum?id=ZBR9EpEl6G4.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural Spline Flows. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf.

Fefferman, C., Mitter, S., and Narayanan, H. Testing the Manifold Hypothesis. *arXiv:1310.0425 [math, stat]*, December 2013. URL http://arxiv.org/abs/1310.0425. arXiv: 1310.0425.

Gemici, M. C., Rezende, D., and Mohamed, S. Normalizing Flows on Riemannian Manifolds. *arXiv:1611.02304 [cs, math, stat]*, November 2016. URL http://arxiv.org/abs/1611.02304. arXiv: 1611.02304.

Ghojogh, B., Ghodsi, A., Karray, F., and Crowley, M. Generative locally linear embedding. In *arXiv preprint arXiv:2104.01525*, 2021.

Gorban, A., Kégl, B., Wunsch, D., and Zinovyev, A. *Principal Manifolds for Data Visualisation and Dimension Reduction, LNCSE 58*. January 2008a. ISBN 978-3-540-73750-6.

Gorban, A. N., Kégl, B., Wunsch, D. C., Zinovyev, A. Y., et al. *Principal manifolds for data visualization and dimension reduction*, volume 58. Springer, 2008b.

Gresele, L., Kügelgen, J. V., Stimper, V., Schölkopf, B., and Besserve, M. Independent mechanism analysis, a new concept? In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Rnn8zoAkrwr.

Gropp, A., Atzmon, M., and Lipman, Y. Isometric Autoencoders. *arXiv:2006.09289 [cs, stat]*, October 2020. URL http://arxiv.org/abs/2006.09289. arXiv: 2006.09289.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. Beta-vae: Learning Basic Visual Concepts with a Constrained Variational Framework. 2017.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2722–2730. PMLR, June 2019. URL http://proceedings.mlr.press/v97/ho19a.html.

Hoffman, M. D., Sountsov, P., Dillon, J., Langmore, I., Tran, D., and Vasudevan, S. NeuTra-lizing Bad Geometry in Hamiltonian Monte Carlo Using Neural Transport. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1903.03704.

Huang, C.-W., Chen, R. T. Q., Tsirigotis, C., and Courville, A. Convex Potential Flows: Universal Probability Distributions with Optimal Transport and Convex Optimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=te7PVH1sPxJ.

Hyvärinen, A. and Pajunen, P. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Netw.*, 12(3):429–439, apr 1999. ISSN 0893-6080. doi: 10.1016/S0893-6080(98)00140-3. URL https://doi.org/10.1016/S0893-6080(98)00140-3.

Jolliffe, I. *Principal Component Analysis*, pp. 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_455. URL https://doi.org/10.1007/978-3-642-04898-2_455.

Kalatzis, D., Ye, J. Z., Wohlert, J., and Hauberg, S. Multi-chart flows. *arXiv:2106.03500 [cs, stat]*, June 2021. URL http://arxiv.org/abs/2106.03500. arXiv: 2106.03500.

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

Kingma, D. P. and Dhariwal, P. Glow: Generative Flow with Invertible 1x1 Convolutions. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf.

Kothari, K., Khorashadizadeh, A., de Hoop, M., and Dokmanić, I. Trumpets: Injective Flows for Inference and Inverse Problems. *arXiv:2102.10461 [cs, eess]*, February 2021. URL http://arxiv.org/abs/2102.10461. arXiv: 2102.10461.

Kumar, A., Poole, B., and Murphy, K. Regularized autoencoders via relaxed injective probability flow. In Chiappa, S. and Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 4292–4301. PMLR, 26–28 Aug 2020. URL http://proceedings.mlr.press/v108/kumar20a.html.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Oehlert, G. W. A note on the delta method. *The American Statistician*, 46(1):27–29, 1992. doi: 10.1080/00031305.1992.10475842. URL https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475842.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv:1912.02762 [cs, stat]*, December 2019. URL http://arxiv.org/abs/1912.02762. arXiv: 1912.02762.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In Bach, F.

and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/rezende15.html.

Rezende, D. J., Papamakarios, G., Racaniere, S., Albergo, M., Kanwar, G., Shanahan, P., and Cranmer, K. Normalizing Flows on Tori and Spheres. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8083–8092. PMLR, July 2020. URL http://proceedings.mlr.press/v119/rezende20a.html.

Ross, B. L. and Cresswell, J. C. Conformal embedding flows: Tractable density estimation on learned manifolds. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. URL https://openreview.net/forum?id=8QV-tt2Q8X.

van den Berg, R., Hasenclever, L., Tomczak, J., and Welling, M. Sylvester normalizing flows for variational inference. In *proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Wei, Y., Shi, Y., Liu, X., Ji, Z., Gao, Y., Wu, Z., and Zuo, W. Orthogonal jacobian regularization for unsupervised disentanglement in image generation. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2021.

Zhang, M., Sun, Y., McDonagh, S., and Zhang, C. On the latent space of flow-based models, 2021. URL https://openreview.net/forum?id=mWnfMrd9JLr.

Zhuang, J., Tang, T., Ding, Y., Tatikonda, S. C., Dvornek, N., Papademetris, X., and Duncan, J. AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. *Advances in Neural Information Processing Systems*, 33, 2020.

A. Python implementation

Below are Python implementations of the PCF objective function. The code uses the JAX (Bradbury et al., 2018) Python library.

```
import jax
import jax.numpy as jnp
import jax.scipy.stats.multivariate_normal as gaussian
import einops
from jax.random import randint

def PCF_objective_brute_force(flow, x, P, alpha=5.0):
  """ Brute force implementation of the PCF objective.
      Implemented for unbatched 1d inputs for simplicity

  Inputs:
    flow  - Function that accepts an unbatched 1d input
            and returns a 1d output and the log determinant
    x     - Unbatched 1d input
    P     - List of numpy arrays that form a partition
            over range(x.size)
    alpha - Regularization hyperparameter

  Outputs:
     objective - PCFs objective
  """
  # Evaluate log p(x) with a Gaussian prior
  z, log_det = flow(x)
  log_pz = gaussian.logpdf(z, 0.0, 1.0).sum()
  log_px = log_pz + log_det

  # Create the Jacobian matrix for every item in the batch
  G = jax.jacobian(lambda x: flow(x)[0])(x)

  # Compute Ihat_P
  Ihat_P = -log_det
  for k in P:
    Gk = G[k,:]
    Ihat_P += 0.5*jnp.linalg.slogdet(Gk@Gk.T)[1]

  objective = -log_px + alpha*Ihat_P
  return objective.mean()

def PCF_objective_unbiased(flow, x, rng_key, alpha=5.0):
  """ Unbiased estimate of the PCF objective when the partition size is 1

  Inputs:
    flow    - Function that accepts an unbatched 1d input
              and returns a 1d output and the log determinant
    x       - Unbatched 1d input
    rng_key - JAX random key
    alpha   - Regularization hyperparameter

  Outputs:
```

```
    objective - PCFs objective
  """
  # Evaluate log p(x) with a Gaussian prior and construct the vjp function
  z, vjp, log_det = jax.vjp(flow, x, has_aux=True)
  log_pz = gaussian.logpdf(z, 0.0, 1.0).sum()
  log_px = log_pz + log_det

  # Sample an index in the partition
  z_dim = z.shape[-1]
  k = random.randint(rng_key, minval=0, maxval=z_dim, shape=(1,))
  k_onehot = (jnp.arange(z_dim) == k).astype(z.dtype)

  # Evaluate the k'th row of G and compute an unbiased estimate of Ihat_P
  Gk, = vjp(k_onehot)
  GkGkT = (Gk**2).sum()
  Ihat_P = -log_det + z_dim*0.5*jnp.log(GkGkT)

  objective = -log_px + alpha*Ihat_P
  return objective.mean()

def iPCF_objective_unbiased(flow, x, rng_key, gamma=10.0):
  """ Unbiased estimate of the iPCF objective when the partition size is 1

  Inputs:
    flow    - Function that accepts an unbatched 1d input
              and returns a 1d output and the log determinant
    x       - Unbatched 1d input
    rng_key - JAX random key
    gamma   - Regularization hyperparameter

  Outputs:
     objective - iPCFs objective
  """
  # Pass x through to the latent space and compute the prior
  z, _ = flow(x)
  log_pz = gaussian.logpdf(z, 0.0, 1.0).sum()

  # Sample an index in the partition
  z_dim = z.shape[-1]
  k = random.randint(rng_key, minval=0, maxval=z_dim, shape=(1,))
  k_onehot = (jnp.arange(z_dim) == k).astype(z.dtype)

  # Compute the reconstruction and k'th row of J
  x_reconstr, Jk = jax.jvp(lambda x: flow(x, inverse=True)[0], (z,), (k_onehot,))
  JkTJk = (Jk**2).sum()
  reconstruction_error = jnp.sum((x - x_reconstr)**2)

  # Compute the objective function
  objective = -log_pz + 0.5*jnp.log(JkTJk) + gamma*reconstruction_error
  return objective.mean()

def construct_partition_mask(index, z_shape):
  """ In general we can find the i'th row of a matrix A
      by computing A.T@mask where mask is zeros everywhere
```

```
      except at the i'th index where it is 1.

      This function finds all of the masks needed to find
      the rows in G that are in the index'th partition.

  Inputs:
    index   - Batched array of integers
    z_shape - Shape of the latent variable

  Outputs:
     masks - Array of 0s and 1s that will be used to
             find the rows of G within the index'th partition.
  """
  batch_size, H, W, C = z_shape
  n_partitions = C

  # The only non zero element of i'th row of
  # partition_mask is at the index[i]'th position
  # This is used to select a partition.
  # shape is (batch_size, C)
  partition_mask = jnp.arange(n_partitions) == index[:,None]

  # Create masks that will let us find the k'th rows of G using masked vjps.
  partition_size = H*W
  G_selection_mask = jnp.eye(partition_size)
  G_selection_mask = G_selection_mask.reshape((partition_size, H, W))

  # Put the masks together
  masks = jnp.einsum("bc,phw->pbhwc", partition_mask, G_selection_mask)
  return masks

def unbiased_objective_image(flow, x, rng_key, alpha=5.0, vectorized=True):
  """ PCFs objective function for images.  Number of partitions is given
      by number of channels of output.

  Inputs:
    flow      - Function that accepts an batched 3d input
                and returns a batched 3d output and the log determinant
    x         - Batched 3d input with channel on last axis
    rng_key   - JAX random key
    alpha     - Regularization hyperparameter
    vectorize - Should all of the vjps be evaluated in parallel?

  Outputs:
     objective - PCFs objective for images
  """
  # Assume that we partition over the last axis of z
  # and that x is a batched image with channel on the last axis
  batch_size, H, W, C = x.shape

  # Evaluate log p(x) and retrieve the function that
  # lets us evaluate vector-Jacobian products
  z, _vjp, log_det = jax.vjp(flow, x, has_aux=True)
  vjp = lambda v: _vjp(v)[0] # JAX convention to return a tuple
```

```
log_pz = gaussian.logpdf(z, 0.0, 1.0).sum(axis=range(1, z.ndim))
log_px = log_pz + log_det

# Randomly sample the index of the partition we will evaluate
n_partitions = z.shape[-1]
index = randint(rng_key, minval=0, maxval=n_partitions, shape=(batch_size,))

# Construct the masks that we'll use to find the index'th partition of G.
# masks.shape == (partition_size, batch_size, H, W, C)
masks = construct_partition_mask(index, z.shape)

# Evaluate the vjp each of the n_partition masks
if vectorized:
  # This is memory intensive but fast
  Gk = jax.vmap(vjp)(masks)
else:
  # This is slow but memory efficient
  Gk = jax.lax.map(vjp, masks)

# Each element of GG^T is the dot product between rows of G
# Construct GkGk^T and then take its log determinant
Gk = einops.rearrange(Gk, "p b H W C -> b p (H W C)")
GkGkT = jnp.einsum("bij,bkj->bik", Gk, Gk)
Ihat_P = 0.5*jnp.linalg.slogdet(GkGkT)[1]*n_partitions - log_det

objective = -log_px + alpha*Ihat_P
return objective.mean()
```

B. Contour Cookbook

Below we list properties of contour densities and pointwise mutual information and their inverse variants. Recall from our assumptions stated in the main text that a normalizing flow generates samples under the model $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z}) = \prod_{\mathbb{K} \in \mathcal{P}} p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}}), \quad \mathbf{x} = f(\mathbf{z})$ where $\dim(\mathbf{x}) \geq \dim(\mathbf{z})$. $f(\mathbf{z})$ has the inverse $\mathbf{z} = f^{-1}(\mathbf{x}) = g(\mathbf{x})$ and Jacobian matrix $J = \frac{df(\mathbf{z})}{d\mathbf{z}}$ while the Jacobian matrix of the inverse function is $G = \frac{dg(\mathbf{x})}{d\mathbf{x}}$. $J_{\mathbb{K}}$ is the matrix whose columns are the columns of J with indices in $\mathbb{K}$ and $G_{\mathbb{K}}$ is the matrix whose rows are the rows of G with indices in $\mathbb{K}$.

Below we assume that $\mathbb{S}$ and $\mathbb{T}$ are disjoint subsets of the integers in $[1, \ldots, \dim(\mathbf{z})]$ and that the indices of $\mathbb{S}$ and $\mathbb{T}$ are ordered so that results with block matrices can be presented clearly. This is a valid assumption because the latent dimension can always be renumbered.

B.1. Definitions

1. $\mathcal{L}_{\mathbb{K}} \triangleq \log p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}}) - \frac{1}{2} \log |J_{\mathbb{K}}^T J_{\mathbb{K}}|$
2. $\widehat{\mathcal{L}}_{\mathbb{K}} \triangleq \log p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}}) + \frac{1}{2} \log |G_{\mathbb{K}} G_{\mathbb{K}}^T|$
3. $\mathcal{L} \triangleq \log p_{\mathbf{x}}(\mathbf{x}) = \log p_{\mathbf{z}}(\mathbf{z}) - \frac{1}{2} |J^T J|$
4. $\widehat{\mathcal{L}} \triangleq \log p_{\mathbf{z}}(\mathbf{z}) + \frac{1}{2} |G G^T|$
5. $\mathcal{I}_{\mathbb{S},\mathbb{T}} \triangleq \mathcal{L}_{\mathbb{S}+\mathbb{T}} - \mathcal{L}_{\mathbb{S}} - \mathcal{L}_{\mathbb{T}}$
6. $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} \triangleq \widehat{\mathcal{L}}_{\mathbb{S}+\mathbb{T}} - \widehat{\mathcal{L}}_{\mathbb{S}} - \widehat{\mathcal{L}}_{\mathbb{T}}$
7. $\mathcal{I}_{\mathcal{P}} \triangleq \mathcal{L} - \sum_{\mathbb{K} \in \mathcal{P}} \mathcal{L}_{\mathbb{K}}$
8. $\widehat{\mathcal{I}}_{\mathcal{P}} \triangleq \widehat{\mathcal{L}} - \sum_{\mathbb{K} \in \mathcal{P}} \widehat{\mathcal{L}}_{\mathbb{K}}$

B.2. Claims

1. $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2} \log |J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| + \frac{1}{2} \log |J_{\mathbb{S}}^T J_{\mathbb{S}}| + \frac{1}{2} \log |J_{\mathbb{T}}^T J_{\mathbb{T}}|$
2. $\mathcal{I}_{\mathcal{P}} = -\frac{1}{2} \log |J^T J| + \frac{1}{2} \sum_{\mathbb{K} \in \mathcal{P}} \log |J_{\mathbb{K}}^T J_{\mathbb{K}}|$
3. $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2} \log |I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}|$ where $A^{\parallel} = A(A^T A)^{-1} A^T$ denotes the projection matrix of A.
4. $\mathcal{I}_{\mathbb{S},\mathbb{T}} \geq 0$
5. $\mathcal{I}_{\mathcal{P}} \geq 0$
6. $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$ if and only if $J_{\mathbb{S}+\mathbb{T}} = U_{\mathbb{S}+\mathbb{T}}^{\parallel} \Sigma_{\mathbb{S}+\mathbb{T}} \begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{T}}^T \end{bmatrix}$ where $U_{\mathbb{S}+\mathbb{T}}^{\parallel}$ is semi-orthogonal, $V_{\mathbb{S}}$ and $V_{\mathbb{T}}$ are orthogonal and $\Sigma_{\mathbb{S}+\mathbb{T}}$ is diagonal.
7. $\mathcal{I}_{\mathcal{P}} = 0$ if and only if $J = U^{\parallel} \Sigma \begin{bmatrix} V_{\mathcal{P}_1}^T & 0 & 0 & 0 \\ 0 & V_{\mathcal{P}_2}^T & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \ldots & V_{\mathcal{P}_{|\mathcal{P}|}}^T \end{bmatrix}$ where $U^{\parallel}$ is a semi orthogonal matrix, Σ is a diagonal matrix and each $V_{\mathcal{P}_k}^T$ is an orthogonal matrix with same number of rows and columns as the $k'th$ element of $\mathcal{P}$.
8. $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$ if and only if $f_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})$ and $f_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})$ intersect orthogonally.
9. $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \frac{1}{2} \log |G_{\mathbb{S}+\mathbb{T}} G_{\mathbb{S}+\mathbb{T}}^T| - \frac{1}{2} \log |G_{\mathbb{S}} G_{\mathbb{S}}^T| - \frac{1}{2} \log |G_{\mathbb{T}} G_{\mathbb{T}}^T|$
10. $\widehat{\mathcal{I}}_{\mathcal{P}} = \frac{1}{2} \log |G G^T| - \frac{1}{2} \sum_{\mathbb{K} \in \mathcal{P}} \log |G_{\mathbb{K}} G_{\mathbb{K}}^T|$

11. $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \frac{1}{2}\log|I - G_{\mathbb{S}}^{\|}G_{\mathbb{T}}^{\|}|$

12. $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} \leq 0$

13. $\widehat{\mathcal{I}}_{\mathcal{P}} \leq 0$

14. $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = 0$ if and only if $G_{\mathbb{S}+\mathbb{T}} = \begin{bmatrix} V_{\mathbb{S}} & 0 \\ 0 & V_{\mathbb{T}} \end{bmatrix} \Sigma_{\mathbb{S}+\mathbb{T}} {U_{\mathbb{S}+\mathbb{T}}^{\|}}^T$ where ${U_{\mathbb{S}+\mathbb{T}}^{\|}}^T$ is semi-orthogonal, $V_{\mathbb{S}}$ and $V_{\mathbb{T}}$ are orthogonal and $\Sigma_{\mathbb{S}+\mathbb{T}}$ is diagonal.

15. $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$ if and only if $J = \begin{bmatrix} V_{\mathcal{P}_1} & 0 & 0 & 0 \\ 0 & V_{\mathcal{P}_2} & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V_{\mathcal{P}_{|\mathcal{P}|}} \end{bmatrix} \Sigma {U^{\|}}^T$ where ${U^{\|}}^T$ is a semi orthogonal matrix, Σ is a diagonal matrix and each $V_{\mathcal{P}_k}$ is an orthogonal matrix with same number of rows and columns as the $k'th$ element of $\mathcal{P}$.

16. If $\dim(\mathbf{x}) = \dim(\mathbf{z})$, then $\mathcal{I}_{\mathcal{P}} = 0$ if and only if $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$

B.3. Proofs

Proof of claim 1 $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2}\log|J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| + \frac{1}{2}\log|J_{\mathbb{S}}^T J_{\mathbb{S}}| + \frac{1}{2}\log|J_{\mathbb{T}}^T J_{\mathbb{T}}|$

Proof.

$$\mathcal{I}_{\mathbb{S},\mathbb{T}} = \mathcal{L}_{\mathbb{S}+\mathbb{T}} - \mathcal{L}_{\mathbb{S}} - \mathcal{L}_{\mathbb{T}} \tag{17}$$

$$= \underbrace{\log \frac{p_{\mathbb{S}+\mathbb{T}}(\mathbf{z}_{\mathbb{S}+\mathbb{T}})}{p_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})p_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})}}_{=0 \text{ by assumption of how prior factors}} - \frac{1}{2}\log|J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| + \frac{1}{2}\log|J_{\mathbb{S}}^T J_{\mathbb{S}}| + \frac{1}{2}\log|J_{\mathbb{T}}^T J_{\mathbb{T}}| \tag{18}$$

$$= -\frac{1}{2}\log|J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| + \frac{1}{2}\log|J_{\mathbb{S}}^T J_{\mathbb{S}}| + \frac{1}{2}\log|J_{\mathbb{T}}^T J_{\mathbb{T}}| \tag{19}$$

□

Proof of claim 2 $\mathcal{I}_{\mathcal{P}} = -\frac{1}{2}\log|J^T J| + \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}}\log|J_{\mathbb{K}}^T J_{\mathbb{K}}|$

Proof.

$$\mathcal{I}_{\mathcal{P}} = \mathcal{L} - \sum_{\mathbb{K}\in\mathcal{P}} \mathcal{L}_{\mathbb{K}} \tag{20}$$

$$= \underbrace{\log \frac{p_{\mathbf{z}}(\mathbf{z})}{\prod_{\mathbb{K}\in\mathcal{P}} p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})}}_{=0 \text{ by assumption of how prior factors}} - \frac{1}{2}\log|J^T J| + \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}}\log|J_{\mathbb{K}}^T J_{\mathbb{K}}| \tag{21}$$

$$= -\frac{1}{2}\log|J^T J| + \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}}\log|J_{\mathbb{K}}^T J_{\mathbb{K}}| \tag{22}$$

□

Proof of claim 3 $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2}\log|I - J_{\mathbb{S}}^{\|}J_{\mathbb{T}}^{\|}|$ *where* $A^{\|} = A(A^TA)^{-1}A^T$ *denotes the projection matrix of* A.

Proof.

$$|J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| = \left|\begin{bmatrix} J_{\mathbb{S}}^T \\ J_{\mathbb{T}}^T \end{bmatrix} \begin{bmatrix} J_{\mathbb{S}} & J_{\mathbb{T}} \end{bmatrix}\right| \tag{23}$$

$$= \left|\begin{bmatrix} J_{\mathbb{S}}^T J_{\mathbb{S}} & J_{\mathbb{S}}^T J_{\mathbb{T}} \\ J_{\mathbb{T}}^T J_{\mathbb{S}} & J_{\mathbb{T}}^T J_{\mathbb{T}} \end{bmatrix}\right| \tag{24}$$

$$= |J_{\mathbb{S}}^T J_{\mathbb{S}}||J_{\mathbb{T}}^T J_{\mathbb{T}} - J_{\mathbb{T}}^T \underbrace{J_{\mathbb{S}}(J_{\mathbb{S}}^T J_{\mathbb{S}})^{-1} J_{\mathbb{S}}^T}_{J_{\mathbb{S}}^{\parallel}} J_{\mathbb{T}}| \tag{25}$$

$$= |J_{\mathbb{S}}^T J_{\mathbb{S}}||J_{\mathbb{T}}^T J_{\mathbb{T}}||I - J_{\mathbb{S}}^{\parallel} \underbrace{J_{\mathbb{T}}(J_{\mathbb{T}}^T J_{\mathbb{T}})^{-1} J_{\mathbb{T}}^T}_{J_{\mathbb{T}}^{\parallel}}| \tag{26}$$

$$= |J_{\mathbb{S}}^T J_{\mathbb{S}}||J_{\mathbb{T}}^T J_{\mathbb{T}}||I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}| \tag{27}$$

Therefore $\frac{1}{2}\log|J_{\mathbb{S}}^T J_{\mathbb{S}}| + \frac{1}{2}\log|J_{\mathbb{T}}^T J_{\mathbb{T}}| - \frac{1}{2}\log|J_{\mathbb{S}+\mathbb{T}}^T J_{\mathbb{S}+\mathbb{T}}| = -\frac{1}{2}\log|I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}|$. An application of claim 1 completes the proof that $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2}\log|I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}|$. □

Proof of claim 4 $\mathcal{I}_{\mathbb{S},\mathbb{T}} \geq 0$

Proof. First we will show that $I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}$ is a positive semi-definite matrix. Let x be some vector.

$$x^T(I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel})x = x^T x - x^T J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel} x \tag{28}$$

$$= |x|_2^2(1 - \underbrace{\frac{x}{|x|_2}}_{\hat{x}} J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel} \frac{x}{|x|_2}) \tag{29}$$

$J_{\mathbb{S}}^{\parallel}$ and $J_{\mathbb{T}}^{\parallel}$ are orthogonal projection matrices, so their operator norm is less than or equal to 1. By definition of the operator norm, we have that $||J_{\mathbb{S}}^{\parallel}\hat{x}||_{\text{op}} \leq 1$ and $||J_{\mathbb{T}}^{\parallel}\hat{x}||_{\text{op}} \leq 1$. It follows that $\hat{x} J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}\hat{x} \leq ||J_{\mathbb{S}}^{\parallel}\hat{x}||_{\text{op}}||J_{\mathbb{T}}^{\parallel}\hat{x}||_{\text{op}} \leq 1$. So

$$|x|_2^2(1 - \hat{x} J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}\hat{x}) \geq |x|_2 \geq 0 \tag{30}$$

It is known that if A is positive semi-definite, then $\log|A| \leq \text{Tr}(A - I)$. We can now apply this bound to $I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}$:

$$-\frac{1}{2}\log|I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}| \geq -\frac{1}{2}\text{Tr}\Big(I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel} - I\Big) \tag{31}$$

$$= \frac{1}{2}\text{Tr}\Big(J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}\Big) \tag{32}$$

$$\geq 0 \text{ because the trace of a positive semi-definite matrix is non negative.} \tag{33}$$

This proves that $\mathcal{I}_{\mathbb{S},\mathbb{T}} \geq 0$. □

Proof of claim 5 $\mathcal{I}_{\mathcal{P}} \geq 0$

Proof. As per Eq. (7), $\mathcal{I}_{\mathcal{P}}$ can be written as the sum of various $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ terms, each of which are non-negative by claim 4. Therefore $\mathcal{I}_{\mathcal{P}} \geq 0$. □

Proof of claim 6 $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$ *if and only if* $J_{\mathbb{S}+\mathbb{T}} = U_{\mathbb{S}+\mathbb{T}}^{\parallel}\Sigma_{\mathbb{S}+\mathbb{T}}\begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{T}}^T \end{bmatrix}$ *where* $U_{\mathbb{S}+\mathbb{T}}^{\parallel}$ *is semi-orthogonal,* $V_{\mathbb{S}}$ *and* $V_{\mathbb{T}}$ *are orthogonal and* $\Sigma_{\mathbb{S}+\mathbb{T}}$ *is diagonal.*

Proof. Let A be a tall matrix with full rank. Its singular value decomposition can be written as:

$$A = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T \tag{34}$$

$$= \begin{bmatrix} U^{\parallel} & U^{\perp} \end{bmatrix} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T \tag{35}$$

$$= U^{\parallel} \Sigma V^T \tag{36}$$

$U^{\parallel}$ is an orthonormal basis for the image of A and $U^{\perp}$ is an orthonormal basis for the orthogonal complement of the image. We can write the SVD of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ as well:

$$J_{\mathbb{S}} = U_{\mathbb{S}}^{\parallel} \Sigma_{\mathbb{S}} V_{\mathbb{S}}^T \tag{37}$$

$$J_{\mathbb{T}} = U_{\mathbb{T}}^{\parallel} \Sigma_{\mathbb{T}} V_{\mathbb{T}}^T \tag{38}$$

Assume that $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$. We must have that $J_{\mathbb{S}}^T J_{\mathbb{T}} = 0$ because if $\mathcal{I}_{\mathbb{S},\mathbb{T}} = -\frac{1}{2} \log |I - J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel}| = 0$, then it must be the case that $J_{\mathbb{S}}^{\parallel} J_{\mathbb{T}}^{\parallel} = 0$, so the images of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ must be orthogonal. This means that $U_{\mathbb{S}}^{\parallel}$ and $U_{\mathbb{T}}^{\parallel}$ are mutually orthogonal because these matrices form orthonormal bases for the images of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$, so their columns form an orthonormal basis for $J_{\mathbb{S}+\mathbb{T}}$. Next we can write out $J_{\mathbb{S}+\mathbb{T}}$:

$$J_{\mathbb{S}+\mathbb{T}} = \begin{bmatrix} J_{\mathbb{S}} & J_{\mathbb{T}} \end{bmatrix} \tag{39}$$

$$= \begin{bmatrix} U_{\mathbb{S}}^{\parallel} \Sigma_{\mathbb{S}} V_{\mathbb{S}}^T & U_{\mathbb{T}}^{\parallel} \Sigma_{\mathbb{T}} V_{\mathbb{T}}^T \end{bmatrix} \tag{40}$$

$$= \underbrace{\begin{bmatrix} U_{\mathbb{S}}^{\parallel} & U_{\mathbb{T}}^{\parallel} \end{bmatrix}}_{U_{\mathbb{S}+\mathbb{T}}^{\parallel}} \underbrace{\begin{bmatrix} \Sigma_{\mathbb{S}} & 0 \\ 0 & \Sigma_{\mathbb{T}} \end{bmatrix}}_{\Sigma_{\mathbb{S}+\mathbb{T}}} \begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{T}}^T \end{bmatrix} \tag{41}$$

$$= U_{\mathbb{S}+\mathbb{T}}^{\parallel} \Sigma_{\mathbb{S}+\mathbb{T}} \begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{T}}^T \end{bmatrix} \tag{42}$$

$U_{\mathbb{S}+\mathbb{T}}^{\parallel}$ is a semi-orthogonal matrix because all of its columns form an orthonormal basis.

Next assume that $J_{\mathbb{S}+\mathbb{T}} = U \Sigma_{\mathbb{S}+\mathbb{T}} \begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{S}}^T \end{bmatrix}$ where U is semi-orthogonal, $\Sigma_{\mathbb{S}+\mathbb{T}}$ is diagonal and $V_{\mathbb{S}}$ and $V_{\mathbb{T}}$ are orthogonal.

$$J_{\mathbb{S}+\mathbb{T}} = U \Sigma_{\mathbb{S}+\mathbb{T}} \begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{S}}^T \end{bmatrix} \tag{43}$$

$$= \begin{bmatrix} U^{\parallel} & U^{\perp} \end{bmatrix} \begin{bmatrix} \Sigma_{\mathbb{S}} & 0 \\ 0 & \Sigma_{\mathbb{T}} \end{bmatrix} \begin{bmatrix} V_{\mathbb{S}}^T & 0 \\ 0 & V_{\mathbb{S}}^T \end{bmatrix} \tag{44}$$

$$= \begin{bmatrix} U^{\parallel} \Sigma_{\mathbb{S}} V_{\mathbb{S}}^T & U^{\perp} \Sigma_{\mathbb{T}} V_{\mathbb{T}}^T \end{bmatrix} \tag{45}$$

$$= \begin{bmatrix} J_{\mathbb{S}} & J_{\mathbb{T}} \end{bmatrix} \tag{46}$$

$U^{\parallel} \Sigma_{\mathbb{S}} V_{\mathbb{S}}^T$ and $U^{\perp} \Sigma_{\mathbb{T}} V_{\mathbb{T}}^T$ are the SVD of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ respectively, so $J_{\mathbb{S}}^{\parallel} = U^{\parallel} {U^{\parallel}}^T$ and $J_{\mathbb{T}}^{\parallel} = U^{\perp} {U^{\perp}}^T$. Plugging this into claim 3 yields the result $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$ because ${U^{\parallel}}^T U^{\perp} = 0$. □

Proof of claim 7 *$\mathcal{I}_{\mathcal{P}} = 0$ if and only if $J = U^{\parallel} \Sigma \begin{bmatrix} V_{\mathcal{P}_1}^T & 0 & 0 & 0 \\ 0 & V_{\mathcal{P}_2}^T & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V_{\mathcal{P}_{|\mathcal{P}|}}^T \end{bmatrix}$ where $U^{\parallel}$ is a semi orthogonal matrix, Σ is a diagonal matrix and each $V_{\mathcal{P}_k}^T$ is an orthogonal matrix with same number of rows and columns as the $k'th$ element of $\mathcal{P}$.*

Proof. Let $\mathcal{P}'$ be a partition over $[1, \dots, \dim(\mathbf{z})]$ with $k < |\mathcal{P}|$ elements where the first $k-1$ elements of $\mathcal{P}$ and $\mathcal{P}'$ are identical and the k'th element of $\mathcal{P}'$ is the union of the final $|\mathcal{P}| - k$ elements of $\mathcal{P}$. We will use $\mathcal{P}_k$ to denote the k'th

element of $\mathcal{P}$, $\mathcal{P}_{:k}$ to denote the union of the first k elements of $\mathcal{P}$ and $\mathcal{P}_{k:}$ to denote the union of the k'th to last elements of $\mathcal{P}$. We will use a proof by induction to prove one direction of the claim where we assume that $\mathcal{I}_{\mathcal{P}} = 0$.

The base case is when $\mathcal{P}'$ contains only $\mathcal{P}_1$ and $\mathcal{P}_{2:}$. From Section 2.5 we know that we can construct the partition using a tree that has a parent node equal to $\mathcal{P}'$ with children that are $\mathcal{P}_1$ and $\mathcal{P}_{2:}$. This means $\mathcal{I}_{\mathcal{P}}$ will be the sum of $\mathcal{I}_{1,2:}$ and other $\mathcal{I}$ terms and because we assumed that $\mathcal{I}_{\mathcal{P}} = 0$, it must be that $\mathcal{I}_{1,2:} = 0$, so we can apply claim 6 to satisfy the inductive hypothesis.

Next assume $\mathcal{P}'$ contains the first $k-1$ elements of $\mathcal{P}$ and an element containing the union of the remainder of $\mathcal{P}$. Assuming that the inductive hypothesis is true, we can write the Jacobian matrix as

$$J = U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_{:k-1}} & 0 \\ 0 & V^T_{\mathcal{P}_{k:}} \end{bmatrix} \quad \text{where} \tag{47}$$

$$V^T_{\mathcal{P}_{:k-1}} = \begin{bmatrix} V^T_{\mathcal{P}_1} & 0 & 0 & 0 \\ 0 & V^T_{\mathcal{P}_2} & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V^T_{\mathcal{P}_{k-1}} \end{bmatrix} \tag{48}$$

We can rewrite J to isolate the columns in the $\mathcal{P}_k$ partition:

$$J = U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_{:k-1}} & 0 \\ 0 & V^T_{\mathcal{P}_{k:}} \end{bmatrix} \tag{49}$$

$$= \begin{bmatrix} U^{\parallel}_{\mathcal{P}_{:k-1}} & U^{\parallel}_{\mathcal{P}_{k:}} \end{bmatrix} \begin{bmatrix} \Sigma_{\mathcal{P}_{:k-1}} & 0 \\ 0 & \Sigma_{\mathcal{P}_{k:}} \end{bmatrix} \begin{bmatrix} V^T_{\mathcal{P}_{:k-1}} & 0 \\ 0 & V^T_{\mathcal{P}_{k:}} \end{bmatrix} \tag{50}$$

$$= \begin{bmatrix} U^{\parallel}_{\mathcal{P}_{:k-1}}\Sigma_{\mathcal{P}_{:k-1}}V^T_{\mathcal{P}_{:k-1}} & U^{\parallel}_{\mathcal{P}_{k:}}\Sigma_{\mathcal{P}_{k:}}V^T_{\mathcal{P}_{k:}} \end{bmatrix} \tag{51}$$

Next, let $J_{\mathcal{P}_{k:}} = U^{\parallel}_{\mathcal{P}_{k:}}\Sigma_{\mathcal{P}_{k:}}V^T_{\mathcal{P}_{k:}}$. $J_{\mathcal{P}_{k:}}$ contains the columns of J with indices in the final $|\mathcal{P} - k|$ elements of $\mathcal{P}$. Choose a partition from these final elements, $\mathcal{P}_k$. Let $J_{\mathcal{P}_k}$ contain the columns of $J_{\mathcal{P}_{k:}}$ that are in $\mathcal{P}_k$ and let $J_{\mathcal{P}_{k+1:}}$ contain the remaining columns. Because $\mathcal{P}_k \in \mathcal{P}$, it must be true that $\mathcal{I}_{\mathcal{P}_k, \mathcal{P}_{k+1:}} = 0$. Therefore we can apply claim 6 to decompose $J_{\mathcal{P}_k}$

$$J_{\mathcal{P}_{k:}} = U^{\parallel}_{\mathcal{P}_{k:}}\Sigma_{\mathcal{P}_{k:}}V^T_{\mathcal{P}_{k:}} \tag{52}$$

$$= U^{\parallel}_{\mathcal{P}_{k:}}\Sigma_{\mathcal{P}_{k:}} \begin{bmatrix} V^T_{\mathcal{P}_k} & 0 \\ 0 & V^T_{\mathcal{P}_{k+1:}} \end{bmatrix} \tag{53}$$

Plugging this back into Eq.49 and yields

$$J = U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_{:k-1}} & 0 \\ 0 & V^T_{\mathcal{P}_{k:}} \end{bmatrix} \tag{54}$$

$$= U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_{:k-1}} & 0 & 0 \\ 0 & V^T_{\mathcal{P}_k} & 0 \\ 0 & 0 & V^T_{\mathcal{P}_{k+1:}} \end{bmatrix} \tag{55}$$

$$= U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_{:k}} & 0 \\ 0 & V^T_{\mathcal{P}_{k+1:}} \end{bmatrix} \tag{56}$$

So by induction, $J = U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_1} & 0 & 0 & 0 \\ 0 & V^T_{\mathcal{P}_2} & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V^T_{\mathcal{P}_{|\mathcal{P}|}} \end{bmatrix}$.

For the other direction, assume that $J = U^{\parallel}\Sigma \begin{bmatrix} V^T_{\mathcal{P}_1} & 0 & 0 & 0 \\ 0 & V^T_{\mathcal{P}_2} & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V^T_{\mathcal{P}_{|\mathcal{P}|}} \end{bmatrix}$. Clearly $J^T J$ will be a block diagonal matrix, so

$\log|J^TJ| = \sum_{\mathbb{K}\in\mathcal{P}} \log|J_{\mathbb{K}}^T J_{\mathbb{K}}|$. It trivially follows from claim 2 that $\mathcal{I}_{\mathcal{P}} = 0$. □

Proof of claim 8 *$\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$ if and only if $f_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})$ and $f_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})$ intersect orthogonally.*

Proof. We saw in the proof of claim 6 that the image of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ are orthogonal when $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$. At the point that $f_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})$ and $f_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})$ intersect, they are aligned with the images of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ respectively, so they will intersect orthogonally. Similarly,if $f_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})$ and $f_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})$ intersect orthogonally, their definition tells us that the image of $J_{\mathbb{S}}$ and $J_{\mathbb{T}}$ are orthogonal, so we must have $\mathcal{I}_{\mathbb{S},\mathbb{T}} = 0$. □

Proof of claim 9 $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \frac{1}{2}\log|G_{\mathbb{S}+\mathbb{T}}G_{\mathbb{S}+\mathbb{T}}^T| - \frac{1}{2}\log|G_{\mathbb{S}}G_{\mathbb{S}}^T| - \frac{1}{2}\log|G_{\mathbb{T}}G_{\mathbb{T}}^T|$

Proof.

$$\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \widehat{\mathcal{L}}_{\mathbb{S}+\mathbb{T}} - \widehat{\mathcal{L}}_{\mathbb{S}} - \widehat{\mathcal{L}}_{\mathbb{T}} \tag{57}$$

$$= \underbrace{\log\frac{p_{\mathbb{S}+\mathbb{T}}(\mathbf{z}_{\mathbb{S}+\mathbb{T}})}{p_{\mathbb{S}}(\mathbf{z}_{\mathbb{S}})p_{\mathbb{T}}(\mathbf{z}_{\mathbb{T}})}}_{=0 \text{ by assumption of how prior factors}} + \frac{1}{2}\log|G_{\mathbb{S}+\mathbb{T}}G_{\mathbb{S}+\mathbb{T}}^T| - \frac{1}{2}\log|G_{\mathbb{S}}G_{\mathbb{S}}^T| - \frac{1}{2}\log|G_{\mathbb{T}}G_{\mathbb{T}}^T| \tag{58}$$

$$= \frac{1}{2}\log|G_{\mathbb{S}+\mathbb{T}}G_{\mathbb{S}+\mathbb{T}}^T| - \frac{1}{2}\log|G_{\mathbb{S}}G_{\mathbb{S}}^T| - \frac{1}{2}\log|G_{\mathbb{T}}G_{\mathbb{T}}^T| \tag{59}$$

□

Proof of claim 10 $\widehat{\mathcal{I}}_{\mathcal{P}} = -\frac{1}{2}\log|GG^T| + \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}}\log|G_{\mathbb{K}}G_{\mathbb{K}}^T|$

Proof.

$$\widehat{\mathcal{I}}_{\mathcal{P}} = \widehat{\mathcal{L}} - \sum_{\mathbb{K}\in\mathcal{P}} \widehat{\mathcal{L}}_{\mathbb{K}} \tag{60}$$

$$= \underbrace{\log\frac{p_{\mathbf{z}}(\mathbf{z})}{\prod_{\mathbb{K}\in\mathcal{P}} p_{\mathbb{K}}(\mathbf{z}_{\mathbb{K}})}}_{=0 \text{ by assumption of how prior factors}} + \frac{1}{2}\log|GG^T| - \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}}\log|G_{\mathbb{K}}G_{\mathbb{K}}^T| \tag{61}$$

$$= \frac{1}{2}\log|GG^T| - \frac{1}{2}\sum_{\mathbb{K}\in\mathcal{P}}\log|G_{\mathbb{K}}G_{\mathbb{K}}^T| \tag{62}$$

□

Proof of claim 11 $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \frac{1}{2}\log|I - G_{\mathbb{S}}^{\parallel}G_{\mathbb{T}}^{\parallel}|$.

Proof.

$$|G_{\mathbb{S}+\mathbb{T}}G_{\mathbb{S}+\mathbb{T}}^T| = \left|\begin{bmatrix}G_{\mathbb{S}}\\G_{\mathbb{T}}\end{bmatrix}\begin{bmatrix}G_{\mathbb{S}}^T & G_{\mathbb{T}}^T\end{bmatrix}\right| \tag{63}$$

$$= \left|\begin{bmatrix}G_{\mathbb{S}}G_{\mathbb{S}}^T & G_{\mathbb{S}}G_{\mathbb{T}}^T\\G_{\mathbb{T}}G_{\mathbb{S}}^T & G_{\mathbb{T}}G_{\mathbb{T}}^T\end{bmatrix}\right| \tag{64}$$

$$= |G_{\mathbb{S}}G_{\mathbb{S}}^T||G_{\mathbb{T}}G_{\mathbb{T}}^T - G_{\mathbb{T}}\underbrace{G_{\mathbb{S}}^T(G_{\mathbb{S}}G_{\mathbb{S}}^T)^{-1}G_{\mathbb{S}}}_{G_{\mathbb{S}}^{\parallel}}G_{\mathbb{T}}^T| \tag{65}$$

$$= |G_{\mathbb{S}}G_{\mathbb{S}}^T||G_{\mathbb{T}}G_{\mathbb{T}}^T||I - G_{\mathbb{S}}^{\parallel}\underbrace{G_{\mathbb{T}}^T(G_{\mathbb{T}}G_{\mathbb{T}}^T)^{-1}G_{\mathbb{T}}}_{G_{\mathbb{T}}^{\parallel}}| \tag{66}$$

$$= |G_{\mathbb{S}}G_{\mathbb{S}}^T||G_{\mathbb{T}}G_{\mathbb{T}}^T||I - G_{\mathbb{S}}^{\parallel}G_{\mathbb{T}}^{\parallel}| \tag{67}$$

Therefore $\frac{1}{2}\log|G_{\mathbb{S}+\mathbb{T}}G_{\mathbb{S}+\mathbb{T}}^T| - \frac{1}{2}\log|G_{\mathbb{S}}G_{\mathbb{S}}^T| - \frac{1}{2}\log|G_{\mathbb{T}}G_{\mathbb{T}}^T| = \frac{1}{2}\log|I - G_{\mathbb{S}}^{\parallel}G_{\mathbb{T}}^{\parallel}|$. An application of claim 9 completes the proof that $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \frac{1}{2}\log|I - G_{\mathbb{S}}^{\parallel}G_{\mathbb{T}}^{\parallel}|$. □

Proof of claim 12 $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} \leq 0$

Proof. Notice that we can prove that $-\frac{1}{2}\log|I - G_{\mathbb{S}}^{\parallel}G_{\mathbb{T}}^{\parallel}| \geq 0$ using an identical proof as the one used to prove 4. Therefore it must be that $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = \frac{1}{2}\log|I - G_{\mathbb{S}}^{\parallel}G_{\mathbb{T}}^{\parallel}| \leq 0$. □

Proof of claim 13 $\widehat{\mathcal{I}}_{\mathcal{P}} \leq 0$

Proof. The same steps used in Eq. (7) to write $\mathcal{I}_{\mathcal{P}}$ as the sum of various $\mathcal{I}_{\mathbb{S},\mathbb{T}}$ terms can be used to write $\widehat{\mathcal{I}}_{\mathcal{P}}$ as the sum of various $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}}$ terms. Since each $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} \leq 0$, it must be that $\widehat{\mathcal{I}}_{\mathcal{P}} \leq 0$. □

Proof of claim 14 $\widehat{\mathcal{I}}_{\mathbb{S},\mathbb{T}} = 0$ *if and only if* $G_{\mathbb{S}+\mathbb{T}} = \begin{bmatrix} V_{\mathbb{S}} & 0 \\ 0 & V_{\mathbb{T}} \end{bmatrix} \Sigma_{\mathbb{S}+\mathbb{T}} {U_{\mathbb{S}+\mathbb{T}}^{\parallel}}^{T}$ *where* ${U_{\mathbb{S}+\mathbb{T}}^{\parallel}}^{T}$ *is semi-orthogonal,* $V_{\mathbb{S}}$ *and* $V_{\mathbb{T}}$ *are orthogonal and* $\Sigma_{\mathbb{S}+\mathbb{T}}$ *is diagonal.*

Proof. The proof is identical to that of 6 except that the matrices are transposed. □

Proof of claim 15 $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$ *if and only if* $J = \begin{bmatrix} V_{\mathcal{P}_1} & 0 & 0 & 0 \\ 0 & V_{\mathcal{P}_2} & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V_{\mathcal{P}_{|\mathcal{P}|}} \end{bmatrix} \Sigma {U^{\parallel}}^{T}$ *where* ${U^{\parallel}}^{T}$ *is a semi orthogonal matrix,* Σ *is a diagonal matrix and each* $V_{\mathcal{P}_k}$ *is an orthogonal matrix with same number of rows and columns as the* $k'th$ *element of* $\mathcal{P}$.

Proof. The proof is identical to that of 7 except that the matrices are transposed. □

Proof of claim 16 *If* $\dim(\mathbf{x}) = \dim(\mathbf{z})$, *then* $\mathcal{I}_{\mathcal{P}} = 0$ *if and only if* $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$

Proof. First assume that $\mathcal{I}_{\mathcal{P}} = 0$. Then by 6, $J = U\Sigma V^T$ where U is orthogonal, Σ is diagonal and V^T is a block diagonal matrix with orthogonal blocks. Because $\dim(\mathbf{x}) = \dim(\mathbf{z})$, $G = J^{-1} = V\Sigma U^T$. Then by 14, $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$. The reverse clause is proven in the same manner. Assuming $\widehat{\mathcal{I}}_{\mathcal{P}} = 0$, we can use 14 to decompose G, take its inverse and use 6 to prove $\mathcal{I}_{\mathcal{P}} = 0$. □

C. PCF Proofs

Lemma 3. *Each contour of a PCF at* $\mathbf{x}$ *is spanned by a unique set of principal components.*

Proof. The principal components of a flow at $\mathbf{x}$ are the eigenvectors of JJ^T. From claim 7 we know that $J = U^{\parallel}\Sigma \begin{bmatrix} V_{\mathcal{P}_1}^T & 0 & 0 & 0 \\ 0 & V_{\mathcal{P}_2}^T & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & V_{\mathcal{P}_{|\mathcal{P}|}}^T \end{bmatrix}$ where $\mathcal{P}_k$ is the $k'th$ element of the partition of the latent space. The $U^{\parallel}$ and Σ matrices form an eigendecomposition of JJ^T because $JJ^T = U^{\parallel}\Sigma^2{U^{\parallel}}^T$, so the columns of $U^{\parallel}$ are the principal components of the PCF. Next, we can rewrite J so that the dependence of each contour on the principal components is explicit:

$$J = \begin{bmatrix} U_{\mathcal{P}_1}^{\parallel}\Sigma_{\mathcal{P}_1}V_{\mathcal{P}_1}^T & U_{\mathcal{P}_2}^{\parallel}\Sigma_{\mathcal{P}_2}V_{\mathcal{P}_2}^T & \dots & U_{\mathcal{P}_{|\mathcal{P}|}}^{\parallel}\Sigma_{\mathcal{P}_{|\mathcal{P}|}}V_{\mathcal{P}_{|\mathcal{P}|}}^T \end{bmatrix} \tag{68}$$

$U_{\mathcal{P}_k}^{\parallel}\Sigma_{\mathcal{P}_k}V_{\mathcal{P}_k}^T$ is the k'th contour of the PCF. We can clearly see that its image is spanned by the principal components with indices in $\mathcal{P}_k$. Because $\mathcal{P}_i \bigcap \mathcal{P}_j = \emptyset, \forall i, j$ we conclude that each contour of a PCF at $\mathbf{x}$ is spanned by a unique set of principal components. □

Theorem 2. *The contours of a principal component flow are principal manifolds.*

Proof. The principal manifold of a flow is found by integrating along the direction of a principal component.

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{w}_{\mathbb{K}}(\mathbf{x}(t)) \tag{69}$$

Lemma 3 tells us that the contours of a PCF are locally spanned by the principal components and that the eigenvectors and eigenvalues of JJ^T are equal to $U^{\|}$ and Σ^2 respectively. So we can simplify by letting $\mathbf{x}(t) = f(\mathbf{z}(t))$.

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{w}_{\mathbb{K}}(\mathbf{x}(t)) \tag{70}$$

$$J\frac{d\mathbf{z}(t)}{dt} = U^{\|}_{\mathcal{P}_k}\Sigma_{\mathcal{P}_k} \tag{71}$$

$$\frac{d\mathbf{z}(t)}{dt} = J^{+}U^{\|}_{\mathcal{P}_k}\Sigma_{\mathcal{P}_k} \tag{72}$$

$$= \begin{bmatrix} 0 & \dots & V_{\mathcal{P}_k}\Sigma_{\mathcal{P}_k}V^T_{\mathcal{P}_k} & \dots & 0 \end{bmatrix}^T \tag{73}$$

This derivation tells us that the principal manifold, when traced out in the latent space, is equal to a manifold that only varies along dimensions in $\mathbb{K}$. This is exactly how contours are generated, therefore the principal manifolds of a PCF are its contours. □

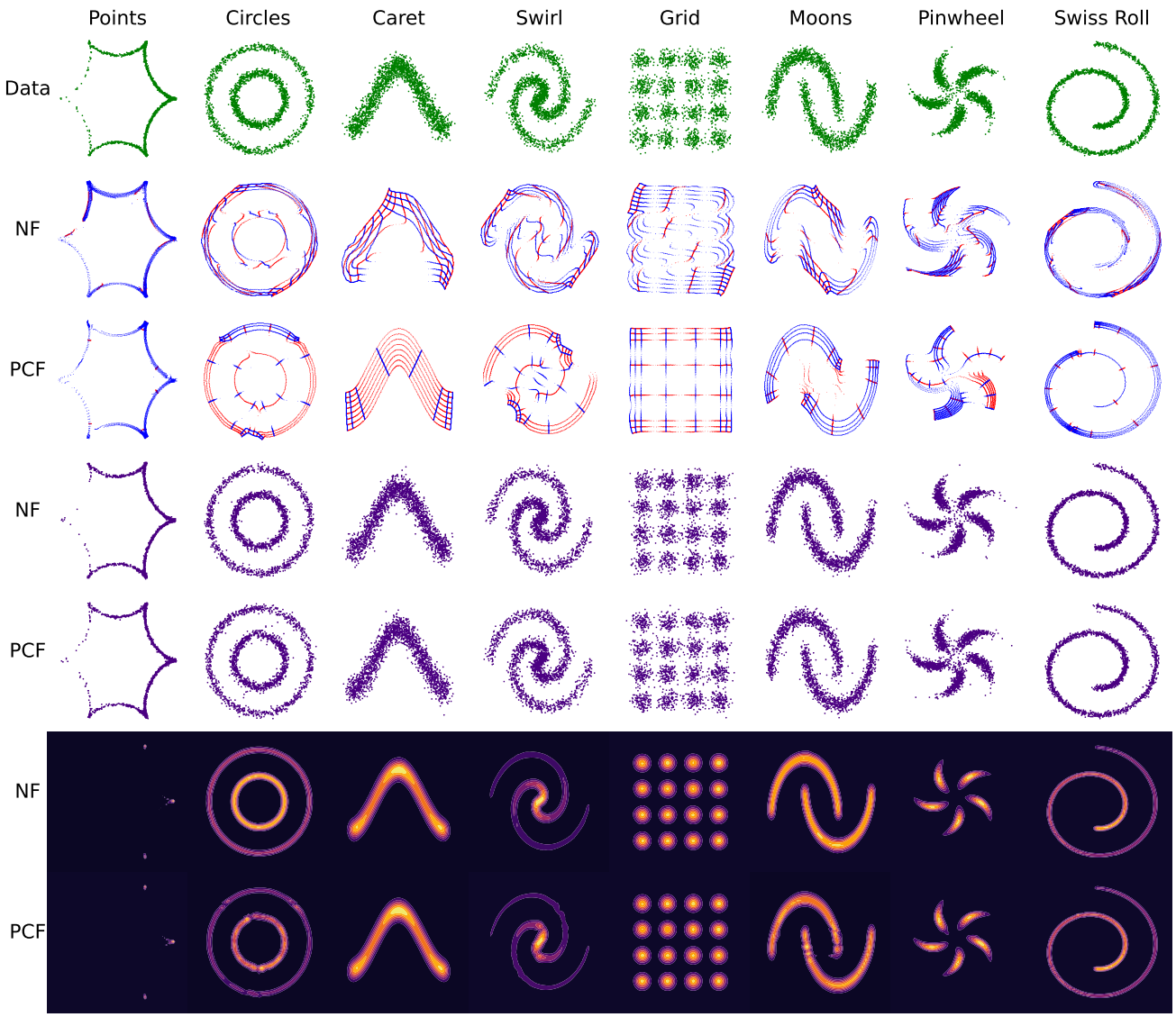


Figure 7. Extended results for synthetic datasets. Top row has true samples, the next two rows are contours, the next two are samples and the last two are probability densities computed by the models.

D. Additional details on experiments

D.1. 2D experiments

See Fig.7 for extended results. Each dataset was generated with 1,000,000 data points and split into 700,000 for training and 300,000 for testing. As mentioned in the main text, the architecture used on all of the datasets consisted of 10 coupling layers with logistic mixture cdf layers that used 8 mixture components and an affine coupling layer that shared the same conditioner network. Each conditioner consisted of 5 residual layers with a hidden layer size of 64. The models were trained using the AdaBelief (Zhuang et al., 2020) optimization algorithm with a learning rate of 1×10^{-3} and a batch size of 256, and $\alpha = 10.0$. Each model was trained for approximately 4 hours on either a NVIDIA 1080ti or 2080ti.

D.2. Variable dimension manifold

We used a flow with 20 coupling based neural spline (Durkan et al., 2019) layers, each with 8 knot points, followed directly by an affine coupling layer that is parametrized by the same conditioner network as done in (Ho et al., 2019). The conditioner networks all consisted of a 5 layer residual network with a hidden dimension of 32. We used a unit Gaussian prior. The

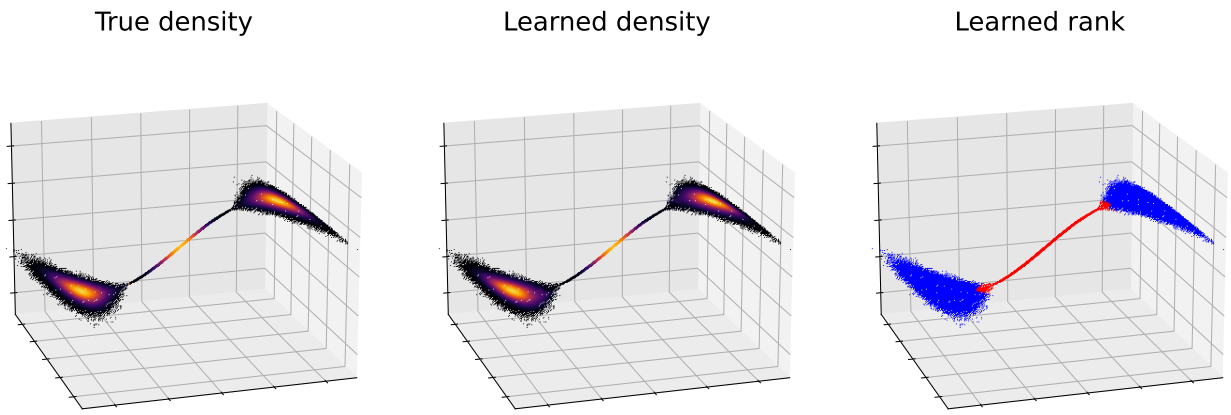


Figure 8. A larger version of Fig. 5

model was trained for around 4 hours on a NVIDIA 3090 gpu with a learning rate of 1×10^{-4} with the AdaBelief (Zhuang et al., 2020) optimization algorithm and a batch size of 2048 and $\alpha = 5.0$. We trained on 2,100,000 data points and evaluated on 900,000. As stated in the main text, the data was augmented with Gaussian noise with a standard deviation of 0.01 to ensure that the model did not collapse during training. The true generative model for the data is:

$$z_1 \sim \frac{1}{3}(N(-2, 0.3) + N(0, 0.3) + N(2, 0.3)) \tag{74}$$

$$z_2 \sim N(0, 1) \tag{75}$$

$$x = f(z_1, z_2) = \begin{bmatrix} z_1 \\ z_2 \text{max}(0, 1 - |\frac{1}{z_1}|) \\ \sin(z_1) \end{bmatrix} \tag{76}$$

The true density was computed in a piecewise manner. If $x_1 = 0$, then

$$p(x) = p(z_1)|\frac{df(z)}{dz_1}^T \frac{df(z)}{dz_1}|^{\frac{-1}{2}} \tag{77}$$

Otherwise,

$$p(x) = p(z_1)p(z_2)|\frac{df(z)}{dz}^T \frac{df(z)}{dz}|^{\frac{-1}{2}} \tag{78}$$

The Jacobian determinants were computed with automatic differentiation.

In Fig. 8 we show a larger version of Fig. 5, in Fig. 9 we showcase samples pulled from the PCF and the contours learned by the model. In Fig. 10 we see that the PCF does extremely well in predicting the log likelihood of the test set. The final KL divergence between the true data distribution and learned was 0.0146. To choose the rank at test time, we compared the three contour likelihoods provided by the model and filtered out the likelihoods that were negligible compared to the others.

D.3. iPCF

Preprocessing For training, we preprocessed incoming batches of data using uniform dequantization and a scaling layer + logit transformation as described in (Dinh et al., 2017). Then each $(28 \times 28 \times 1)$ image was flattened into a 784 dimensional vector.

Model architectures The iPCF and iNF architectures were composed of two parts. The first is a flow in the full 784 dimensional ambient space that consisted of 20 layers of GLOW (Kingma & Dhariwal, 2018) with each conditioner network consisting of 3 residual networks with a hidden dimension of 32 and dropout rate of 0.2. After the GLOW layers, the output was sliced so that the resulting dimensionality was 10 and this low dimensional vector was passed to a unit Gaussian prior.

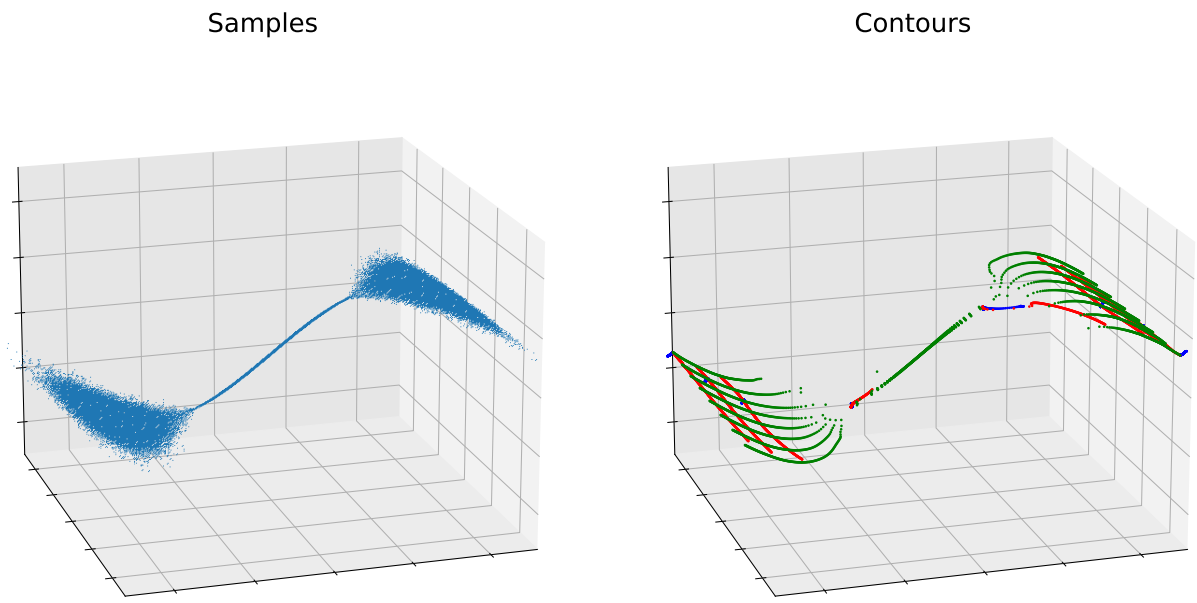


Figure 9. Samples and contours from the model trained for Section 5.2

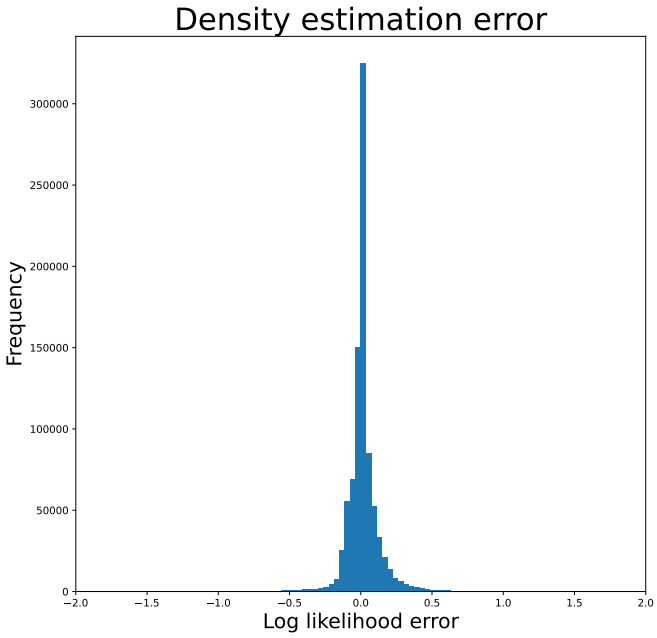


Figure 10. Histogram of the difference between the true log likelihood and the predicted for the experiment in Section 5.2

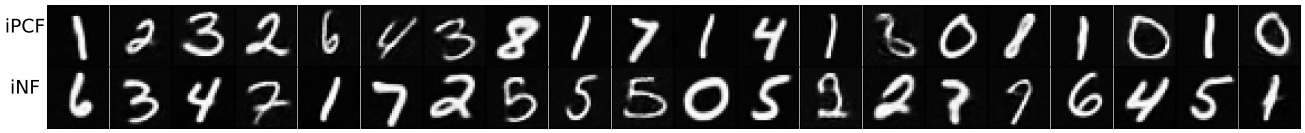


Figure 11. Random samples from the iPCF and injective normalizing flow trained in Section 5.3

The second part of the architecture, used during fine-tuning, consisted of 10 coupling based neural spline layers with 8 knots and affine coupling layers. Each conditioner contained 4 residual network layers with a hidden dimension of 4. The input to this flow is the 10 dimensional output of the first component and the output is fed into a unit Gaussian prior.

Training The overall model was trained in two stages. The first stage optimized the objective in section 4 of (Caterini et al., 2021) using only the GLOW layers. The objectives we optimized were:

$$\text{Objective1}_{\text{iPCF}} = \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(g(\mathbf{x})) + \frac{\dim(\mathbf{z})}{2}\log\left(\sum_i J_{ki}(g(\mathbf{x}))^2\right) + \gamma||f(g(\mathbf{x})) - x||^2, \quad k \sim \text{Uniform}(1,\dots,10) \tag{79}$$

$$\text{Objective1}_{\text{iNF}} = \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(g(\mathbf{x})) + \frac{1}{2}\log|J(g(\mathbf{x}))^T J(g(\mathbf{x}))| + \gamma||f(g(\mathbf{x})) - x||^2 \tag{80}$$

For both models we set $\gamma = 10$, used a batch size of 64, learning rate of 1×10^{-4} and the AdaBelief (Zhuang et al., 2020) optimization algorithm. We found that it was crucial to use a small learning rate, otherwise training would fail. These models were trained for approximately 36 hours on either a NVIDIA 3090ti or RTX8000 gpu.

After this stage of training, we combined the GLOW layers with the neural spline layers into one normalizing flow. We then froze the parameters for the GLOW layers and trained the parameters of the spline layers using a learning rate of 1×10^{-3} for another 24 hours on the same objective as before, but without the reconstruction error term:

$$\text{Objective2}_{\text{iPCF}} = \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(g(\mathbf{x})) + \frac{\dim(\mathbf{z})}{2}\log\left(\sum_i J_{ki}(g(\mathbf{x}))^2\right), \quad k \sim \text{Uniform}(1,\dots,10) \tag{81}$$

$$\text{Objective2}_{\text{iNF}} = \sum_{\mathbf{x}\in\mathcal{D}} -\log p_{\mathbf{z}}(g(\mathbf{x})) + \frac{1}{2}\log|J(g(\mathbf{x}))^T J(g(\mathbf{x}))| \tag{82}$$

E. Practical considerations

PCFs have many nice theoretical properties, but can be difficult to train and interpret in practice. We find that the constraint $\mathcal{I}_{\mathcal{P}} = 0$ can only be satisfied with normalizing flows that are very expressive. We conjecture that the reason is because the constraint $\mathcal{I}_{\mathcal{P}} = 0$ requires that each $O(2^{|\mathcal{P}|})$ possible contour that can be constructed are orthogonal to the other contours. As a result, we find it necessary to keep $|\mathcal{P}|$ small by using iPCFs to model high dimensional data, increasing the size of each partition or using a feature extractor flow that can transform data into a simpler form for the PCF. Furthermore, the latent space of PCFs is not trivial to interpret. While it is true that the latent variables of PCFs correspond to different principal manifolds, the index of the dimension corresponding to different principal manifolds can change (see Fig. 4 for clear examples of this). This means that a smooth path through over a principal manifold may require a discontinuous path through the latent space.