ABSTRACT

Probabilistic programming and statistical computing are vibrant areas in the development of the Julia programming language, but the underlying infrastructure dramatically predates recent developments. The goal of MeasureTheory.jl is to provide Julia with the right vocabulary and tools for these tasks.

In the package we introduce a well-chosen set of notions from the foundations of probability together with powerful combinators and transforms, giving a gentle introduction to the concepts in this article.

The task is foremost achieved by recognizing measure as the central object. This enables us to develop a proper concept of densities as objects relating measures with each others. As densities provide local perspective on measures, they are the key to efficient implementations.

The need to preserve this computationally so important locality leads to the new notion of locally-dominated measure, solving the so-called "base measure problem" and making work with densities and distributions in Julia easier and more flexible.

Keywords
Julia, Measure theory, Probability, Statistics

1. Why measures?

Distributions are an insufficient abstraction for probabilistic modeling.

Let’s first consider Bayesian modeling. In the posterior density of the parameter $\theta$ given the observation or data $x$,

$$p(\theta \mid x) = \frac{p(\theta) \ p(x \mid \theta)}{p(x)},$$

the denominator is called the model evidence. Computing it efficiently is often difficult or unfeasible.

Some sampling algorithms circumvent this problem and only require knowing the density up to a constant factor. We then can work with the “unnormalized posterior density” instead, but we want to distinguish it from a proper probability density, because that difference determines which algorithms are available.

In developments like this using distributions, it’s common to begin in terms of a distribution, but then carry around the unnormalized posterior as a function. Fortunately, the meaning of this function is usually clear from context. But structurally, this representation is now divorced from its meaning. It’s no longer a Distribution object, so the tools from the original library can no longer help. This is a perspective we will take often.

Thus, starting with elements of our class of interest, a simple and common operation has led us to something outside of this class. This is analogous to the way polynomials over the reals lead to the complex numbers. If “the shortest path between two truths in the real domain passes through the complex domain” (Hadamard), then the same argument can be made for measures as connection between distributions.

As a second example, people working in Bayesian modeling sometimes use improper priors. Whatever one’s position on the merits of this, we think it’s reasonable to make this approach possible. But an improper prior does not integrate to one, so it’s not a distribution. Also, other computational and statistical inverse problems can often be framed as Bayesian inversion problem with corresponding “prior” for example taking the role of a regularization parameter that might be improper, or even flat as in maximum likelihood estimation.

A final and very different concern is the structure of most distribution libraries, in which distributions are classified primarily according to whether they are “discrete” or “continuous”. Such systems often lack facilities to use these in combination; some go so far as to encode the distinction in the type system, making them fundamentally incompatible.

But “discrete vs continuous” is a false dichotomy. For example, in three dimensions we can (and often do) work with distributions over points, lines, planes, or the entire space, or over spaces like a simplex or the surface of a sphere. These can be combined in rich ways, for example as a spike and slab prior for sparse modeling, or as a parameterized subspace for low-rank modeling.

We can address all of these points by extending the system we work with. Instead of distributions, our primary class of interest is measures, with distributions as a special case.

Of course, this special case is particularly useful. The point is not to disregard distributions, but to change our focus to one allowing a richer calculus for reasoning. Adapting Hadamard’s argument, The shortest path between two distributional truths passes through non-distributional measures.
Contributions

Our work is novel in several ways. MeasureTheory.jl has

- Explicitly represented base measures with the same sophistication as the rest of the system, in particular more than just “discrete or continuous”;
- A local approach for determining absolute continuity which is usually a global characteristic;
- Multiple parameterizations for a given measure, without a proliferation of constructors; and
- Normalization and support constraints held separately from the data-dependent computation, allowing for greater efficiency.

Some parts of our approach can been seen in existing systems, though these are still far from universal:

- A rich set of combinators for building new measures from existing ones;
- Flexible type constraints, for example allowing measures with symbolic parameters;
- Light-weight measure construction, replacing a common as-

A rich set of combinators for building new measures from existing ones;
- Flexible type constraints, for example allowing measures with symbolic parameters;
- Light-weight measure construction, replacing a common axiom that the (probability) mass of sets obtained as union of disjoint component sets.

The space \( X \) could be the space \( \{1, 2, 3, 4, 5, 6\} \) suitable to model a six-sided die, or \( X \) might be the 3-dimensional Euclidean space \( \mathbb{R}^3 \) suitable to model “volume”, to give two examples.

One further such space that deserves mention is traditionally denoted \( \Omega \). This is an abstract space connected to the real-world notion of probability, which we denote as \( \text{Prob} \). Computationally, \( \Omega \) can be considered to be the set of possible initial states of a random number generator \( \text{rng} :: \text{AbstractRNG} \), which is the source of computational randomness.

\( \Omega \) has a special role of tying random quantities a program produces and their distributions together, through the notion of random variables, which are functions \( X, Y \) from \( \Omega \) taking values in spaces \( X, Y \) such as \( \{1, 2, 3, 4, 5, 6\} \) or \( \mathbb{R}^3 \).

The package Omega.jl uses random variables derived from \( \Omega \) as core principle.

From a computational perspective, a Julia function taking only an \( \text{rng} :: \text{AbstractRNG} \) argument, making a number of calls to \( \text{rand}(\text{rng}) \), and returning a value \( x \in X \) is a random variable \( X : \Omega \rightarrow X \). A simple example is

\[
X(\text{rng}=\text{Random}().\text{GLOBAL}_\text{RNG}) = \text{rand}(\text{rng}, 1:6)
\]

While mathematically \( X \) is a function from \( \Omega \), the argument \( \omega \) is often hidden like the function argument \( \text{rng} \) of its computational counterpart—not a coincidence.

Each random variable \( X \) is tied to its probability law \( D \), the distribution assigning probability to the events \( \{X \in A\} \) for each (measurable) set \( A \)

\[
\text{Prob}(X \in A) = D(A).
\]

In that sense, classical distribution packages restrict themselves very much to the task of providing a catalogue of useful probability laws, and a simple mechanism to provide a random draw (a random variable) with that law, as function \( \text{rand}(\text{rng}, \mathbb{D}) \). In the example,

\[
\text{Prob}(X \in A) = D(A) = \frac{|A|}{6}
\]

where \( |A| \) denotes the number of elements of \( A \subset \{1, 2, \cdots , 6\} \).

In MeasureTheory, measures have abstract super-type AbstractMeasure.

Kolmogorov’s axioms

We now introduce Kolmogorov’s axioms, which describe laws that characterize probability distributions, and, with one exception, also measures.

Measures and probability distributions are both required to obey the axiom that the (probability) mass of sets obtained as union of disjoint component sets equals the sum of the (probability) masses of the components. So if sets \( A \) and \( B \) are not disjoint, the mass of the union is computed from the mass of the components using inclusion-exclusion. For probabilities this is

\[
P(A \cup B) = P(A) + P(B) - P(A \cap B).
\]

Similarly for a measure \( \mu \),

\[
\mu(A \cup B) = \mu(A) + \mu(B) - \mu(A \cap B).
\]

In the case where \( A \) and \( B \) are disjoint, this reduces to

\[
P(A \cup B) = P(A) + P(B).
\]

\footnote{The stackoverflow discussion https://mathoverflow.net/q/11591 discusses books on measure theory; M.S. likes \cite{1} as starting point and uses \cite{10} and \cite{11} as reference. As we anticipate interest in connecting with the actual theory, we give some pointers to more technical details in the footnotes.}

\footnote{Additionally, one requires a random variable to be a measurable function that is, one for which function image of an event \( \subset X \) is also an event of \( \Omega \). This is a similar, but much weaker requirement to the definition of a continuous function (“the inverse image of an open set is also open”).}

\footnote{Precisely: union of a sequence of disjoint component sets.}
a property called additivity\footnote{The axioms also require a more general form of this to hold. If \(\{A_n | n \in \mathbb{N}\}\) are pairwise disjoint (that is, no two have any common elements), then it must be true that \(P(\bigcup_{n \in \mathbb{N}} A_n) = \sum_{n \in \mathbb{N}} P(A_n)\), and similarly for more general measures, replacing \(P\) with \(\mu\). This extension of additivity to the countably infinite case is called \(\sigma\)-additivity.}

The reader has encountered many measures before, whether they were given that name. For example, counting measure gives the number of elements of a set, and the Lebesgue measure gives length, area, volume, etc in Euclidean space (CountingMeasure and LebesgueMeasure in the package.)

So far our characterizations of measures and probability distributions are functionally identical. Indeed, the one distinguishing feature is the law of unit measure, which only for distributions requires that

\[P(X) = 1,\]

and also \(\text{Prob}(\Omega) = 1\). Thus \(\Omega\) can be considered the event “that anything at all will happen”.

By this axiom, probability mass is a proportion, a quantity between 0 and 1. Only this axiom, which for measures is replaced by weaker axiom \(\mu(\emptyset) = 0\), sets the two apart. To reinforce that a distribution is also a measure, we’ll sometimes refer to it as a probability measure.

This close relationship between measures and probability measures is illustrated by the Lebesgue measure \(\lambda\) on \(X = \mathbb{R}\). The measure of the full space is \(\lambda(\mathbb{R}) = \infty\). But restricting to the unit interval gives \(\lambda([0,1]) = 1\). This restricted measure is a probability distribution—the uniform distribution \(\mathcal{U}([0,1])\) describing the law of Julia function \(\text{rand}\) giving random numbers in the interval \([0,1]\).

We have not yet stated for which sets Kolmogorov’s axioms have to hold for something to be called a measure or a probability:

\(\sigma\)-algebras and how to like them

A (probability) measure on \(X\) need not assign a (probability) mass to every possible subset \(A \subset X\), but only to each \textit{event} for a probability measure, or \textit{measurable set} more generally. Statements we make about sets should be understood as restricted to this set-of-sets, which can be different from one measure to the next.

Now, Kolmogorov’s axioms require a given measure to be defined for \(X\), and they also require that this set of measurable sets is closed under the operations which the axioms allow (complements and unions of sets or sequences of sets). This imposes an algebraic structure to the measurable sets, called a \(\sigma\)-algebra, which is exactly that: A set of sets \(\subset X\) (or \(\subset \Omega\) in particular) closed under complementation and countable unions and intersections.

This raises the question: Why not just use the power set, the “set of all sets”, which is, after thinking about it, such a \(\sigma\)-algebra?

A key observation is that having defined probability mass for a number of relatively simple sets, for example on all intervals, Kolmogorov’s axioms give a recipe to compute the probability mass of more complicated sets, and with those, even more in new applications of the axioms, etc. One says the \(\sigma\)-algebra is generated by these sets. Indeed, \texttt{Distributions.jl} does not give means to compute probability for arbitrary subsets of floating point numbers, but gives a way of computing probability mass just of intervals of the form \((-\infty, x]\) through the function \texttt{cdf}.

For spaces that are “small enough”, a meaningful probability can be assigned to each subset of the space. But even in moderately sized spaces this can become infeasible: Just representing a single arbitrary subset of the float range, the computational abstraction of the real line, would require staggering \(2^{306}\) Petabytes of memory. This is not practical. And for any space containing a continuum such as the real line it becomes not impractical but mathematically impossible to assign a meaningful probability to all subsets.

But even the mathematical measure theory needs to stay clear of the set of all subsets of uncountable spaces, because it turns out that not even volume can be properly defined for all subsets of the Euclidean space\footnote{Another thing to keep in mind is that, for a function \(f: X \to Y\) to be measurable requires that “the inverse image of a measurable set must be measurable”. So allowing more sets to \(B \subset Y\) to be measurable requires either allowing more measurable \(A \subset X\), or restricting the set of measurable functions.}.

Accordingly, as of this writing, \texttt{MeasureTheory} does not have first-class \(\sigma\)-algebras, but rather considers them to be implicit to a given measure. Having said that, \(\sigma\)-algebras do have a role to play in applied measure theory, but we give only a pointer here. The practical importance of \(\sigma\)-algebras here lies therein that they also encode available information. Observe that each event \(A\) corresponds to a question, with a probabilistic answer. For example, an event \(A = \{X \in [c, d]\}\) corresponds to the question whether the random variable \(X\) is in the interval \([c, d]\) and there is a probability \(\text{Prob}(A)\) that this will be the case. Assume we know whether some of such questions \(A, B\) are answered (each with yes or no).

Then we also know that the answer to the complement \(A^c\) (no or yes), and we always know that \(\Omega\) has probability 1 and the corresponding question is answered by ‘yes’. Likewise, if both \(A\) and \(B\) can be answered given what we know, we also know the answer to \(A \cup B\).

As the reader might now suspect, such systems of known events form a \(\sigma\)-algebra. In particular, for each random variable \(X\) there is a \(\sigma\)-algebra associated with \(X\) that encodes what else we know knowing \(X\) and how we should assign (conditional) probabilities to other events knowing \(X\).

This is a very relevant question for probabilistic programming tasks and causal inference. But we’ve already mentioned that \texttt{MeasureTheory} does not explicitly represent \(\sigma\)-algebras. Instead, questions like these can be addressed using kernels \texttt{KernelTheory}(Section\footnote{The axioms also require a more general form of this to hold. If \(\{A_n | n \in \mathbb{N}\}\) are pairwise disjoint (that is, no two have any common elements), then it must be true that \(P(\bigcup_{n \in \mathbb{N}} A_n) = \sum_{n \in \mathbb{N}} P(A_n)\), and similarly for more general measures, replacing \(P\) with \(\mu\). This extension of additivity to the countably infinite case is called \(\sigma\)-additivity.}), structures which can hold the conditional probabilities given we find a way to assign them in a computationally efficient way.

Densities

A measure is often described in terms of its \textit{density}. Before getting to a more technical discussion, we hope a physical analogy can help build some intuition. Imagine a wooden board with some knots in it, where we might be interested in the mass of some two-dimensional cut-out shape. This mass depends not only on the
shape but also on its location and orientation, in particular the inclusion of knots. In this way we can start with a measure (here Lebesgue measure) and use a density (the physical density of the wood) to construct a new measure (the mass of any given cut-out shape).

Of course, we do this all the time with distributions, building a continuous distribution in terms of a probability density function (pdf) over Lebesgue measure, or a discrete distribution in terms of a probability mass function (pmf), which is just a density over counting measure.

Somewhat more formally, a probability density in Euclidean space is a local ratio of probability assigned to an infinitesimally small volume/area $dx$ around each point $x$, relative to that volume itself

$$f(x) = \frac{\text{Probability}(dx)}{\text{Volume}(dx)}.$$

“Volume” is not a distribution, but is in fact the Lebesgue measure described above. Discrete distributions can be expressed similarly using counting measure. There are certainly more events (sets) than outcomes (elements), so a probability density gives a local and parsimonious description of a distribution. More generally, for measures $\mu$ and $\nu$ (and an absolute continuity condition, to be discussed), the density is

$$f(x) = \frac{\mu(dx)}{\nu(dx)} = \frac{d\mu}{d\nu} (x).$$

As we see, density only makes sense relative to some base measure. Limiting ourselves to distributions makes this awkward to even discuss, but allowing measures as first-class objects means we can make this characterization more explicit, and thus more flexible. What sets MeasureTheory apart from most libraries is that we don’t sweep this under the rug, but rather address it head-on.

In place of Lebesgue or counting measure, any measure we can express can play the role of $d\nu$ above. This becomes crucial in high-dimensional spaces, where working with other reference measures such as a product of normal distributions becomes a numerical requirement. (Mathematically, there is no infinite dimensional Lebesgue measure, so numerically, even high-dimensional Lebesgue measures can be problematic.)

Density computations are typically done in log space. For example, if we’re interested in $\frac{d\mu}{d\nu}$, we’ll typically instead compute $\log\text{density}_\text{rel}(\mu, \nu, x)$. Here and below, we’ll often write the math in terms of densities and the code in terms of log-densities, despite unfortunate inconsistency between the two.

There are several benefits to working in log-space. Results are much less likely to overflow or underflow, and the many products and exponents become sums and products, respectively, which are much more efficient to compute and to differentiate.

Working locally and in log-space also gives us a convenient antisymmetry,

$$\log\text{density}_\text{rel}(\mu, \nu, x) = -\log\text{density}_\text{rel}(\nu, \mu, x).$$

In particular, $\log\text{density}_\text{rel}(\mu, \nu, x)$ takes special values in some common cases:

<table>
<thead>
<tr>
<th>$\nu \ll_s \mu$</th>
<th>(else)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu \ll s \nu$</td>
<td>(finite)</td>
</tr>
<tr>
<td>(else)</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

4. Base measures and log-densities

It’s very useful for users to be able to call $\log\text{density}_\text{rel}$ for arbitrary measures. But defining things directly in these terms would require a definition for every pair of possible measures, which is of course intractable.

We’ll now discuss the mechanics of defining a measure, followed by a description of the end-user perspective.

---

Density and log-density are always relative; the \text{rel} here is to indicate that we want the density relative to a second measure which is explicitly specified.
Defining a measure

To define a new measure requires a base measure, the log-density with respect to that base measure, and a support. A base measure can be defined locally or globally. In the latter case, the local base measure defaults to

\[
\text{basemeasure}(\mu::\text{AbstractMeasure}, x) = \text{basemeasure}(\mu)
\]

This allows users to define a base measure specific to a given neighborhood if they prefer, or to define a global base measure when that suffices.

For example, for a standard Normal measure, we have

\[
\text{basemeasure}(\mu::\text{Normal}()) = \frac{1}{\sqrt{2\pi}} \times \text{Lebesgue}(\mathbb{R})
\]

\[
\text{logdensity_def}(\mu::\text{Normal}(), x) = - x^2 / 2
\]

\[
\text{insupport}(\mu::\text{Normal}(), x) = \text{true}
\]

Note the difference from a typical distribution-oriented implementation; the log-density consists only of \(-x^2/2\), with the normalization term pushed into the base measure.

The normalization term is present for any normal distribution, even those with a different mean or variance. So if we have a product of normals, for example in a regression problem, the normalization can be shared across them. For the simple example of \(n\) iid standard Normals, this effectively rewrites the log-pdf

\[
\sum_{j=1}^{n} \left( -\frac{1}{2} \log 2\pi - \frac{x_j^2}{2} \right) \quad \text{as} \quad \frac{n}{2} \log 2\pi + \sum_{j=1}^{n} \left( -\frac{x_j^2}{2} \right)
\]

Further simplification is clearly possible; doing this in a composable way without sacrificing efficiency is the subject of ongoing work.

Just as \text{logdensity_def}(\mu, x)\) defines the log-density relative to the default base measure, it’s often useful to be able to define the log-density between two measures that are given explicitly. For this, we have the three-argument \text{logdensity_def}(\mu, \nu, x)\).

Though this three-argument form is available as a primitive, it would be intractable to use it to define log-densities for every pair of measures. Instead, \text{basemeasure} forms a forest (a collection of trees, with \(\mu \rightarrow \nu\) if \(\nu = \text{basemeasure}(\mu)\)). The three-argument \text{logdensity_def} connects trees to form a graph. For any two measures, MeasureTheory can traverse this graph at compile time to determine how to compute the log-density.

The end-user perspective

From our above discussion of moving terms between the log-density and the base measure, the reader may suspect that this can be done arbitrarily, leaving no “natural” log-density for a given measure. This is indeed the case; the log-density of a measure is only determined relative to a particular base measure.

Our choice of \text{basemeasure} doesn’t reflect any mathematical invariant, only computational convenience. Accordingly, \text{basemeasure} and \text{logdensity_def} should be considered implementation details, and not part of the user-facing interface.

The way MeasureTheory.jl is set up, repeated application of \text{basemeasure} will eventually reach a fix point. That is, some measure will eventually be its own base measure. For such a measure, \text{logdensity_def} is required to return zero. We refer to the fix point reached in this way from a given measure as that measure’s root measure.

Instead of \text{logdensity_def}, users should typically call \text{logdensity_rel}(\mu, \nu, x)\) which computes \frac{\text{logdensity}(x)}{\text{logdensity}(\mu)}), or \text{logdensityof}(\mu, x)\), which gives the log-density relative to the root measure. \text{logdensityof} was originally defined in [20], and provides a convenient common ground for those not accustomed to thinking of densities as being relative. We expect this to be especially useful for new users.

5. Parameterized measures

A common challenge in building a library of distributions is the choice of parameterizations. For example, in Stan [7], a negative binomial distribution is parameterized by \(\alpha\) and \(\beta\), where

\[
\text{NegBinomial}(y \mid \alpha, \beta) = \binom{y + \alpha - 1}{\alpha - 1} \left( \frac{\beta}{\beta + 1} \right)^\alpha \left( \frac{1}{\beta + 1} \right)^y.
\]

In the Julia package Distributions.jl [12], this is instead given by

\[
\text{NegBinomial}(y \mid r, p) = \binom{y + r - 1}{r - 1} p^r (1 - p)^y.
\]

These are equivalent (let \(r = \alpha\) and \(p = \frac{\beta}{\beta + 1}\), and the inconsistency alone gives some evidence that it might be reasonable to prefer one or the other depending on the circumstances. Yet most libraries only allow one or the other (or yet another alternative) as the negative binomial distribution. Other parameterizations require entirely different names.

In MeasureTheory, our approach avoids this problem. A parameterized measure is defined by a struct of the appropriate type with a named tuple \text{par} field. For example,

```
struct NegativeBinomial{N,T} <: ParameterizedMeasure{N}
    par :: NamedTuple{N,T}
end
```

We can then write

```
NegativeBinomial(r=10, p=0.75)
```

or

```
NegativeBinomial(\alpha=10, \beta=3)
```

Calls to \text{rand}, \text{logdensity_def}, etc then delegate to methods according to the appropriate names. We use \text{KeywordCalls.jl} [18], so all names are resolved statically at compile time.

6. Kernels

A kernel is a (measurable)\(^7\) function \(\kappa\) that returns a measure. Equivalently, it represents a family of measures parameterized by

\[^7\text{That is, the function} x \mapsto \kappa(x)(A) \text{must be measurable for every fixed} A.\]
A Markov kernel, in particular, corresponds to the distribution of a measure transparent. A prominent application is a [conditional distribution]. In this case $\kappa$ is further restricted to be a Markov kernel $\kappa: X \rightarrow P(Y)$, where $P(Y)$ represents “probability measures on $Y$”. If $X$ and $Y$ are random variables defined on $X$ and $Y$, the kernel $\kappa$ defines a probability measure assigning every measurable $B \subset Y$ probability $P(Y \in B \mid X = x) := (\kappa(x))(B)$. In MeasureTheory, a kernel is represented as either a Julia function or an AbstractTransitionKernel object. As a common special case, a ParameterizedTransitionKernel pairs a measure constructor with a mapping into its parameter space, and makes the functional relationship between argument of $\kappa$ and the returned measure transparent.

A Markov kernel, in particular, corresponds to the distribution of a parameterized random variable, a function with random outcomes (or to “mechanisms” in causal inference, for example in [16]). For example, the kernel $\kappa: x \mapsto \text{Normal}(x, \sqrt{T})$ corresponds to the random function $f(\text{rng}, x) = x + \sqrt{x} * \text{randn}(\text{rng})$ and can be expressed in MeasureTheory as

```julia
κ = kernel(Normal) do x
    (μ=x, σ=√x)
end
```

or

```julia
κ = kernel(Normal; μ=identity, σ=sqrt)
```

The latter formulation decomposes the kernel into separate “parameter maps”, making relationships between $x$ and the parameters of $\kappa(x)$ more explicit.

This could of course also be expressed as a parameterized measure. The difference is that kernels are lighter weight to build and have all the dynamism of functions, while the static nature of parameterized measures can make it easier to express some optimizations.

### 7. Pointwise products and likelihoods

The pointwise product in probability often describes the fusion of information, in this case the fusion of information from prior and from the observations via the likelihood, but also shows up in related situations, such as [sensor fusion] in signal processing.

To begin, let’s consider the case of probability distributions dominated by Lebesgue measure. For a parameter $\theta$ and data $x$, suppose we have a prior density $p(\theta)$ and likelihood $p(x | \theta)$. Then Bayes’s Law gives the posterior density $p(\theta \mid x) = \frac{p(\theta) p(x \mid \theta)}{p(x)}$.

In practice, we rarely know the normalization factor $p(x)$, so we often work in terms of the unnormalized posterior density, and write

$$p(\theta \mid x) \propto p(\theta) p(x \mid \theta).$$

Because the product does not include normalization, in general it does not integrate to one, and so it’s not a probability density function. But there are no such problems as a density for a measure. We now consider this more general formulation.

Given a parameter space $\Theta$ and observation space $X$, a Likelihood $\ell$ consists of

1. a kernel $\ell_\theta : \Theta \rightarrow \mathcal{M}(X)$,
2. an observation $z \in X$, and
3. a base measure $\ell_\beta \in \mathcal{M}(X)$.

A likelihood can be treated as a function $\Theta \rightarrow \mathbb{R}_{\geq 0}$, by defining

$$\ell(\theta) = \frac{d\ell_\theta(\theta)}{d\ell_\beta}(\ell_z)$$

Of particular interest is the context where the base measure $\ell_\beta$ is

$$\ell_\beta = \hat{\theta} \equiv \arg \max_{\theta} \frac{d\ell_\theta(\theta)}{d\rho}(\ell_z)$$

for any non-singular measure $\rho$; this corresponds to the relative likelihood. In MeasureTheory, we allow the base measure to be specified, but take as a default the root measure of $\ell_z(\theta)$.

Note that a likelihood $\ell$ takes a point in $\Theta$ to an evaluated density in a different space (the observation space $X$). In particular, a likelihood is not a measure.

Rather, a given likelihood acts on measures, taking a “prior” measure $\mu$ to “unnormalized posterior” measure $\mu \odot \ell$. Relative to another measure $\alpha$, this has density

$$\frac{d(\mu \odot \ell)}{d\alpha}(\theta) \equiv \frac{d\mu}{d\alpha}(\theta) \ell(\theta)$$

Treating likelihoods and pointwise products explicitly in this way gives an easy route to many performance optimizations. For example, the likelihood of a linear model can be computed very efficiently in terms of basic linear algebra, and in cases of conjugacy the pointwise product simplifies to a measure that can be expressed in closed form.

### 8. Product and power measures

Given measures $\mu$ on $X$ and $\nu$ on $Y$, the product measure $\mu \otimes \nu$ is a measure on $X \times Y$. Just as Lebesgue measure on $\mathbb{R}$ is generated by defining its value on intervals, the product measure is generated by defining

$$(\mu \otimes \nu)(A \times B) = \mu(A) \nu(B),$$

for any measurable $A \subset X$ and $B \subset Y$. MeasureTheory code uses exactly this notation, $\mu \otimes \nu$. This can be extended recursively to products of any finite number of measures.
If $\mu$ is defined in terms of a base measure $\alpha$ and likewise $\nu$ over $\beta$, the product measure has base measure $\alpha \otimes \beta$ and density

$$\frac{d(\mu \otimes \nu)}{d(\alpha \otimes \beta)} = \frac{d\mu}{d\alpha} \frac{d\nu}{d\beta}.$$ 

A special case of this is when $\mu = \nu$. In MeasureTheory we refer to this as a power measure, written $\mu^2$ or $\mu^n$ for higher dimensions. More generally, the second argument can be a tuple, so for example $\mu^2(2,3)$ extends $\mu$ as a product measure over $2 \times 3$ matrices.

A second special case is when we have a collection of values together with a kernel. For this we use the For combinator. So in a regression model, we might express the response as coming from

For(1:n) do  
  Normal($\beta * x[j], \sigma$)  
end

This is exactly the product measure

$$\otimes \text{Normal}(\beta x_j, \sigma).$$

More generally, suppose we have a measure $\mu$ on a space $X$ and a kernel $\kappa : X \to M(Y)$. Then we can define a measure on $X \times Y$ as follows.

For a given $x \in X$ with $\mu \ll \kappa$, let $\nu_x = \kappa(x)$ have base measure $\beta_x$. Then similarly to the independent product, we have

$$\frac{d(\mu \otimes \nu_x)}{d(\alpha \otimes \beta_x)} = \frac{d\mu}{d\alpha} \frac{d\nu_x}{d\beta_x}.$$ 

9. Superposition and mixtures

A superposition is the measure-theoretic analog of a mixture model. For measures $\mu$ and $\nu$ with $f = \frac{d\mu}{d\alpha}$ and $g = \frac{d\nu}{d\beta}$, the superposition, written $\mu + \nu$, is a measure with base measure $\alpha + \beta$ and density

$$\frac{d(\mu + \nu)}{d(\alpha + \beta)} = \frac{f \, d\alpha + g \, d\beta}{d\alpha + d\beta} = \frac{f \, d\alpha}{d\alpha + d\beta} + \frac{g \, d\beta}{d\alpha + d\beta} = \frac{f}{1 + \frac{d\alpha}{d\beta}} + \frac{g}{d\alpha + 1}.$$ 

Using the inverse in this final line allows $\frac{d\alpha}{d\beta}$ to be computed once and then re-used. Also note that in the special case where $\alpha = \beta$, this reduces to $\frac{f + g}{f + g}$. An important special case of superposition occurs when $\mu$ and $\nu$ are finite measures on some space $\Omega$, and $\mu(\Omega) + \nu(\Omega) = 1$. This is equivalent to a convex combination of probability densities, called a mixture. Conveniently, a product of measures distributes over superposition. That is, if $\alpha$ and $\beta$ are measures on a common space $X$ and $\gamma$ and $\delta$ are measures on $Y$, then

$$\alpha \otimes (\gamma + \delta) = \alpha \otimes \gamma + \alpha \otimes \delta,$$

and

$$(\alpha + \beta) \otimes \gamma = \alpha \otimes \gamma + \beta \otimes \gamma.$$ 

A very common special case of superposition is a spike and slab prior, a mixture of a Dirac measure (a point mass) with a continuous measure. This is useful for sparse Bayesian modeling, as implemented in the sparse ZigZag sampler, described in [3] and implemented in [15] using MeasureTheory.

10. Density decomposition

Especially for probability measures, it’s common for the log-density with respect to Lebesgue or counting measure to have several types of terms. Given an observation $x$ and any relevant parameters, there are typically

- Terms that are data-dependent, each involving some nontrivial function of $x$ (and possibly also of the parameters),
- Terms that are parameter-dependent, and
- Terms that are constant.

In addition, it’s common to have an “argument check” to be sure $x$ is in the support of the distribution.

Depending on the application, we can often ignore some of these. For example, we may know $x$ is in the support by construction, or from a previous check. In these cases, any use of resources to check arguments is wasteful.

There are many cases where it’s important for performance to have constant and parameter-dependent terms separated from data-dependent ones. Suppose $\mu$ is a measure with log-density of the form

$$\log \frac{d\mu}{dx}(x) = \ell(x; \theta) = f(x, \theta) + g(\theta) + C.$$ 

If we instead observe an iid product of $N$ observations, the log-density is

$$\sum_{j=1}^N \ell(x_j; \theta) = \sum_{j=1}^N [f(x_j, \theta) + g(\theta) + C] = N[g(\theta) + C] + \sum_{j=1}^N f(x_j, \theta).$$

This final form can be much more efficient, because it reduces $N$ repeated computations of $g(\theta) + C$ to one.

In some cases it’s important to avoid computing $g(\theta) + C$ at all. For correlation matrices, it’s common to use the LKJ prior [11].

Rather than work with correlation matrices directly, it’s convenient to work in terms of the Cholesky decomposition. For this purpose, MeasureTheory includes an LKJCholesky measure. This is typically used as a prior with fixed parameters $k$ and $\eta$, which give the dimensionality and “concentration” of the measure. The relative cost of normalization (which is irrelevant for MCMC) can be computed as
function relative_normcost(k, η)
    µ = LKJCholesky(k, η)
    L = rand(µ).
    f_cost = @elapsed logdensity($(µ, L))
    g_plus_C_cost = @elapsed Dists.lkj_log0($(k, η))
    return g_plus_C_cost / (g_plus_C_cost + f_cost)
end

So for example, in 10 dimensions with η = 2.0 this gives

    julia> relative_normcost(10, 2.0)
    0.76428528899935

That is, 76% of the time is spent in normalization. If our application

• Parameter-dependent terms, with the constant and parameter-dependent terms
• Data-dependent terms,

The two-argument log density then computes only the data-

Constant terms, −1/2 log 2π.
Parameter-dependent terms, − log σ.
Data-dependent terms, − 1/2 \left(\frac{z-\mu}{\sigma}\right)^2.

The two-argument log density then computes only the data-
dependent terms, with the constant and parameter-dependent terms pushed to the base measure. This makes it easy to defer computation of these terms until they are required.

11. Affine transforms

A particularly expressive way to build new measures in terms of

eXisting ones is through a pushforward. Given a measure µ and a

function f defined on its support, the pushforward of µ through f

is a measure \( f_\ast \mu \) defined by

\[ f_\ast \mu(S) = \mu(f^{-1}(S)). \]

In the following sections, we first discuss in the context of probability measures before the more general case.

Forward parameterization

Starting with a \( k \)-dimensional multivariate random variable \( z \), an affine transform is a “linear transform with a shift”. We can use this to define a new random variable \( x \), as

\[ X = \sigma Z + x_0, \]

with \( \sigma \) and \( x_0 \) of the appropriate dimensions.\[ \text{If } E[Z] = 0, \text{ then } x \text{ has mean } E[X] = \sigma E[Z] + x_0 = x_0 \]

and variance matrix

\[ \text{Var}[X] = \text{Var}(X - x_0)(X - x_0)^\top = \sigma^2 \text{Var}[Z] \]

Note that if \( E[Z] = I_k \), we get \( \text{Var}[X] = \sigma^2 \Sigma \), so we can arrive at a given positive semidefinite \( \Sigma \) by taking \( \sigma \) to be its lower

8 We would typically use \( \mu \) in place of \( x_0 \), if not for the unfortunate potential confusion with \( \mu \) as a name for a measure.

\[ \text{Cholesky factor} \]

Also, in the special case of a one-dimensional Gaussian, this gives the familiar \( \sqrt{\Sigma} = \sigma \).

We call this the forward parameterization because it’s especially convenient for sampling, sometimes referred to as “running the model forward”. Unfortunately, the cost of this convenience is a relatively awkward expression for the density. In this case, we start with \( x \) and need to solve \( z \), finally adjusting according to the determinant of the transformation:

\[ p_X(x) = \frac{1}{|\sigma|} p_Z(z) = \frac{1}{|\sigma|} p_Z(\sigma^{-1}(x - x_0)), \]

where \(|\sigma|\) is the determinant of the square matrix, and the second equality comes from solving for \( z \), which gives \( z = \sigma^{-1}(x - x_0) \).

More generally (when the transform is not expressed as a matrix), this role is played by the determinant of the Jacobian.

This requires solving a linear system. Even with \( \sigma \) being lower-triangular, this involves division operations and the allocation of a temporary vector for storage of \( z \).

In many cases, we prefer the density (or log-density, really) to be fast to evaluate. This leads us to a kind of dual approach to the above.

Inverse parameterization

An alternative parameterization of a multivariate Gaussian is in terms of its precision matrix \( \Lambda = \text{Var}[X]^{-1} \). Similarly to above, we’ll write \( \lambda \) for the lower Cholesky factor of \( \Lambda \), so \( \lambda \lambda^\top = \Lambda \). The parameterization is then specified by

\[ X = \lambda^{-1}Z + \mu. \]

In this parameterization, \( \lambda \) is an inverse scale or rate parameter.

Solving for \( z \) is of course now very simple, and the density becomes

\[ p_X(x) = |\lambda| p_Z(z) = |\lambda| p_Z(\lambda^{-1}(x - \mu)). \]

In exchange, forward sampling becomes awkward,

\[ X = \lambda^{-1}Z + \mu. \]

Generalization

Given an injective map \( f : z \mapsto x \) and measures \( \mu \ll_\alpha \sigma \), the pushforward has density

\[ \frac{d f_\ast \mu}{d f_\ast \sigma}(x) = \frac{d \mu}{d \sigma}(f^{-1}(x)) \frac{d f^{-1}(x)}{d \alpha}(z). \]

Note that there’s no Jacobian to be found! This is because the measure and base measure are either both transformed, or both not.

In fact, the Jacobian \( \frac{d f_\ast \mu}{d f_\ast \sigma} \) only comes into play when we compute “across the transform”, and even then it’s not in every case.

Changing our notation slightly, the general case is

\[ \frac{f_\ast \mu(dx)}{\alpha(dz)} = \frac{1}{\int_\alpha \sigma(dx)} \frac{\mu(dx)}{\alpha(dz)}. \]

This decomposes the problem into two subproblems. First we must compute \( \int_\alpha \sigma(dx) \). This plays the role of the determinant of the Jacobian, but is specific to the base measure \( \alpha \). In particular, \( \alpha \) might
be a discrete measure, in which case this factor is one. Finally, we compute \( \frac{d\sigma}{dz} \), which is just the (pre-transformation) density, more familiar from previous discussion as \( \frac{d\mu}{dz} \).

For the Lebesgue case, if the Jacobian \( \left| \frac{dx}{dz} \right| \) is not square but, say, \( n \times k \) for \( n > k \), the resulting measure will be embedded into a \( k \)-dimensional affine subspace of \( \mathbb{R}^n \). This can be convenient for low-rank modeling, which can be important for high-dimensional data.

\[ \sigma \]

Despite it being a very new package, MeasureTheory.jl is already active work to build up on it and to extend it.

PointProcesses.jl defines point processes in particular requiring the concept of random measure. MultivariateMeasures.jl implement measures on a manifold using Hausdorff measure as the base measure. Mitosis.jl gives high-performance implementations of logdensity for multivariate measures, using LoopVectorization.jl.

Soss.jl is a probabilistic programming language that has recently adopted MeasureTheory.jl as a foundation. In particular, every Soss Model is an instance of AbstractMeasure, and has another Soss model as its base measure.

As mentioned in Section ZigZagBoomerang.jl allows sampling with a spike and slab prior for sparse Bayesian inference and makes use the freedom to choose appropriate reference measures. These models can be expressed using Soss.

Mitosis.jl uses MeasureTheory.jl to represent Bayesian networks via Markov kernels and defines transformations on those.

12. Extensions

While the vast majority of research in computational statistics works explicitly in terms of probability distributions, a few authors have addressed measures more generally:

- Borgström et al describe the Fun system in F# in terms of measure transformer semantics, but discusses only finite measures.
- Narayanan et al describe Hakaru, a system for Bayesian modeling using measures. Here, a measure is a functional

\[ \mu[f] = \int f \, d\mu, \]

represented as a program. Hakaru’s combinators are then expressed as compilers taking programs as the inputs.

- Radul and Alexeev describe the base measure problem of losing track of a base measure when applying a transformation, and suggest standardizing around Hausdorff measure as a solution. This problem doesn’t arise for us, because the base measure is always taken into account.

In Julia, the Distributions.jl package is very popular for computations on distributions. The drawbacks of Distributions.jl are essentially those described in the first few sections of this paper. Current advantages over MeasureTheory are the extensive range of distributions it implements and its popularity and familiarity to many Julia users. MeasureTheory.jl currently has Distributions.jl as a dependency, and uses it as a fall-back for many computations. For convenience, we also re-export the Distributions module under the handle Dists.

14. Conclusion

We have introduced the concepts and implementation of MeasureTheory.jl. This package is very new, so we expect there will be some changes as it matures. For this reason we have limited our discussion to aspects of the implementation we believe are relatively stable.

We hope this work can become a common foundation for probabilistic modeling in Julia. In particular, we believe this approach is especially well-suited for use in probabilistic programming, for which Julia has such a robust and active community.

We welcome discussion and community involvement with this package, as well as additional extensions to those we have described.

15. References


