PplOps (ppl.Ppl.Dist.PplOps)

Table of Contents

Table of Contents

Module `Dist.PplOps`

Operators for distributions

```
type 'a dist

val (+~) : int dist -> int dist -> int dist
val (-~) : int dist -> int dist -> int dist
val (*~) : int dist -> int dist -> int dist
val (/~) : int dist -> int dist -> int dist
val (+.~) : float dist -> float dist -> float dist
val (-.~) : float dist -> float dist -> float dist
val (*.~) : float dist -> float dist -> float dist
val (/.~) : float dist -> float dist -> float dist
val (&~) : bool dist -> bool dist -> bool dist
val (|~) : bool dist -> bool dist -> bool dist
val not : bool dist -> bool dist
val (^~) : string dist -> string dist -> string dist
```

Module `Ppl.Dist`

Module used for defining probabilistic models

Contains a type `dist` which is used to represent probabilistic models.


```
exception Undefined
```

```
module Prob : Ppl__.Sigs.Prob
```

```
type prob = Prob.t
```
A type for which values need to sum to 1

```
type likelihood = Prob.t
```
A type for which values don't need to sum to 1

```
type 'a samples = ('a * prob) list
```
A set of weighted samples, summing to one

```
type _ dist = private
```

`| Return : 'a -> 'a dist`	distribution with a single value
`| Bind : 'a dist * ('a -> 'b dist) -> 'b dist`	monadic bind
`| Primitive : 'a Primitive.t -> 'a dist`	primitive exact distribution
`| Conditional : ('a -> likelihood) * 'a dist -> 'a dist`	variant that defines likelihood model

Type for representing distributions

Condition Operators

```
val condition' : ('a -> float) -> 'a dist -> 'a dist
val condition : bool -> 'a dist -> 'a dist
val score : float -> 'a dist -> 'a dist
val observe : 'a -> 'a Primitive.t -> 'b dist -> 'b dist
val from_primitive : 'a Primitive.t -> 'a dist
```

Monad Functions

```
include Ppl__.Monad.Monad with type 'a t := 'a dist
```

```
type 'a t
```

```
val return : 'a -> 'a t
val bind : 'a t -> ('a -> 'b t) -> 'b t
val (>>=) : 'a t -> ('a -> 'b t) -> 'b t
val let* : 'a t -> ('a -> 'b t) -> 'b t
val fmap : ('a -> 'b) -> 'a t -> 'b t
val liftM : ('a -> 'b) -> 'a t -> 'b t
val liftM2 : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t
val mapM : ('a -> 'b t) -> 'a list -> 'b list t
val sequence : 'a t list -> 'a list t
```

Primitives

These functions create dist values which correspond to primitive distributions so that they can be used in models. Ok

```
type 'a primitive

val binomial : int -> float -> int primitive
```

Create a binomial distribution, the output is the number of successes from n independent trials with probability of success p

```
val normal : float -> float -> float primitive
val categorical : ('a * float) list -> 'a primitive
val discrete_uniform : 'a list -> 'a primitive
val beta : float -> float -> float primitive
val gamma : float -> float -> float primitive
val continuous_uniform : float -> float -> float primitive

val bernoulli : float -> bool dist
val choice : float -> 'a dist -> 'a dist -> 'a dist
```

Sampling

```
val sample : 'a dist -> 'a
val sample_n : int -> 'a dist -> 'a array
val sample_with_score : 'a dist -> 'a * likelihood
val dist_of_n_samples : int -> 'a dist -> 'a list dist
```

Prior Distribution

```
val prior' : 'a dist -> 'a dist
val prior : 'a dist -> ('a * likelihood) dist
val prior_with_score : 'a dist -> ('a * likelihood) dist
val support : 'a dist -> 'a list

module PplOps : Ppl__.Sigs.Ops with type 'a dist := 'a dist
```

Operators for distributions

Up â ppl » Ppl » Inference

Module `Ppl.Inference`

Implementation of inference algorithms

Inference algorithms to be called on probabilistic models defined using `Dist`


```
exception Undefined

type 'a samples = ('a * Dist.prob) list
```

Helpers

```
val unduplicate : 'a samples -> 'a samples
val resample : 'a samples -> 'a samples Dist.dist
val normalise : 'a samples -> 'a samples
val flatten : ('a samples * Dist.prob) list -> 'a samples
```

Exact Inference

```
val enumerate : 'a Dist.dist -> float -> 'a samples
val exact_inference : 'a Dist.dist -> 'a Dist.dist
```

Importance Sampling

```
val importance : int -> 'a Dist.dist -> 'a samples Dist.dist
val importance' : int -> 'a Dist.dist -> 'a Dist.dist
```

Rejetion Sampling

```
type rejection_type =

| Hard
| Soft

val pp_rejection_type : Stdlib.Format.formatter -> rejection type -> unit
val show_rejection_type : rejection type -> string
val create' : int -> 'a option Dist.dist -> 'a list -> 'a list
val create : int -> 'a option Dist.dist -> 'a list
```

```
val reject_transform_hard : ? threshold:float -> 'a Dist.dist -> ('a *
Dist.prob) Dist.dist
val reject'' : 'a Dist.dist -> 'a option Dist.dist
val reject_transform_soft : 'a Dist.dist -> ('a * Dist.prob) Dist.dist
val rejection_transform : ? n:int -> rejection type -> 'a Dist.dist -> 'a
Dist.dist
val rejection_soft : 'a Dist.dist -> ('a * Dist.prob) option Dist.dist
val rejection_hard : ? threshold:Core.Float.t -> 'a Dist.dist -> ('a *
Dist.prob) option Dist.dist
val rejection : ? n:int -> rejection type -> 'a Dist.dist -> 'a Dist.dist
```

Sequential Monte Carlo

```
val smc : int -> 'a Dist.dist -> 'a samples Dist.dist
val smc' : int -> 'a Dist.dist -> 'a Dist.dist
val smcStandard : int -> 'a Dist.dist -> 'a samples Dist.dist
val smcStandard' : int -> 'a Dist.dist -> 'a Dist.dist
val smcMultiple : int -> int -> 'a Dist.dist -> 'a samples Dist.dist
val smcMultiple' : int -> int -> 'a Dist.dist -> 'a Dist.dist
```

Metropolis Hastings

```
val mh' : int -> 'a Dist.dist -> 'a Dist.dist
val mh'' : int -> 'a Dist.dist -> 'a Dist.dist
val mh_sampler : int -> 'a Dist.dist -> 'a list Dist.dist
val mh_transform : burn:int -> 'a Dist.dist -> 'a Dist.dist
```

Particle Independent Metropolis Hastings

```
val pimh : int -> 'a Dist.dist -> 'a samples list Dist.dist
val pimh' : int -> int -> 'a Dist.dist -> 'a Dist.dist
```

Particle Cascade

```
val resamplePC : ('a * float) list -> int -> ('a * Dist.prob) list
Dist.dist
val cascade : int -> 'a Dist.dist -> 'a samples Dist.dist
val cascade' : int -> 'a Dist.dist -> 'a Dist.dist
```

Common

```
type infer_strat =

| MH of int
| SMC of int
| PC of int
| PIMH of int
| Importance of int
```

```
| Rejection of int * rejection type
| RejectionTrans of int * rejection type
| Prior
| Enum
| Forward

val pp_infer_strat : Stdlib.Format.formatter -> infer strat -> unit
val show_infer_strat : infer strat -> string
val print_infer_strat : infer strat -> string
val print_infer_strat_short : infer strat -> string
val infer : 'a Dist.dist -> infer strat -> 'a Dist.dist
val infer_sampler : 'a Dist.dist -> infer strat -> unit -> 'a
```

Up â ppl » Ppl » Primitive » PRIM_DIST

Module type `Primitive.PRIM_DIST`

The signature for new primitives distributions

```
type t

val sample : unit -> t
val pdf : t -> float
val cdf : t -> float
val ppf : t -> float
val support : t support
```

Up â ppl » Ppl » Primitive

Module `Ppl.Primitive`

Module defining a type for primitive distributions


```
type 'a support =
```

`| DiscreteFinite of 'a list`	A list of valid values
`| DiscreteInfinite`	discrete dist with infinite support e.g. poisson
`| ContinuousFinite of ('a * 'a) list`	set of endpoints
`| ContinuousInfinite`	continuous dist with an infinite support e.g.
`| Merged of 'a support * 'a support`	combination of any of the above

The type of supports - the values with a distribution can take

```
module type PRIM_DIST = sig ... end
```

The signature for new primitives distributions

```
type 'a t
```

Type of primitive dists wrapping a module

New Distributions

```
val create_primitive : sample:(unit -> 'a) -> pdf:('a -> float) ->
cdf:('a -> float) -> support:'a support -> ppf:('a -> float) -> 'a t
```

Predefined Distributions

```
val binomial : int -> float -> int t
val categorical : ('a * float) list -> 'a t
val normal : float -> float -> float t
val discrete_uniform : 'a list -> 'a t
val beta : float -> float -> float t
val gamma : float -> float -> float t
val poisson : float -> int t
val continuous_uniform : float -> float -> float t
```

Basic Operations

```
val pdf : 'a t -> 'a -> float
val logpdf : 'a t -> 'a -> float
val cdf : 'a t -> 'a -> float
val ppf : 'a t -> 'a -> float
val sample : 'a t -> 'a
val support : 'a t -> 'a support
```

Other

```
val merge_supports : ('a support * 'a support) -> 'a support
```

Up â ppl » Ppl » Helpers

Module `Ppl.Helpers`

Utilities for working with distributions

A set of utilities for generating statistics and printing distributions

Samples

```
val sample_mean : ? n:int -> float Dist.dist -> float
val sample_variance : ? n:int -> float Dist.dist -> float
val take_k_samples : int -> 'a Dist.dist -> 'a array
val unduplicate : ('a * Dist.prob) list -> ('a * Dist.prob) list
```

Removes duplicates and sums the probabilities associated so that each value appears once

```
val flatten : (('a * Dist.prob) list * Dist.prob) list -> ('a *
Dist.prob) list
val normalise : ('a * Dist.prob) list -> ('a * Dist.prob) list
val weighted_dist : ? n:int -> 'a Dist.dist -> ('a,Â int) Core.Map.Poly.t
```

Printing

```
val print_exact_exn : (module Base.Stringable.S with type t = 'a) -> 'a
Dist.dist -> unit
val print_exact_bool : bool Dist.dist -> unit
val print_exact_int : int Dist.dist -> unit
val print_exact_float : float Dist.dist -> unit
```

Others

```
val time : (unit -> 'a) -> 'a * float
val memo : ('a -> 'b) -> 'a -> 'b
val memo_no_poly : (module Base__.Hashtbl_intf.Key.S with type t = 'a) ->
('a -> 'b) -> 'a -> 'b
```


Up â ppl » Ppl » Evaluation

Module `Ppl.Evaluation`

A module for evaluating the correctness of models and inference procedures

Contains functionality to perform hypothesis tests and KL-divergences for both continuous and discrete distributions


```
type 'a samples = 'a Empirical.Discrete.t
type 'a dist = 'a Dist.dist
```

KL-Divergence

```
val kl_discrete : ? n:int -> 'a Primitive.t -> 'a dist -> float
```
Find the KL divergence for two discrete distributions

```
val kl_continuous : ? n:int -> float Primitive.t -> float dist -> float
```
Find the KL divergence for two continuous distributions

```
val kl_cum_discrete : int array -> bool Primitive.t -> bool dist -> (int
* float) array
val kl_cum_continuous : int array -> float Primitive.t -> float dist ->
(int * float) array
```

Hypothesis Tests

```
val kolmogorov_smirnov : ? n:int -> ? alpha:float -> float dist -> float
Primitive.t -> Owl_stats.hypothesis
```
Perform kolmogorov smirnov test, returns a `hypothesis` which is true if the null hypothesis is rejected

```
val chi_sq : ? n:int -> ? alpha:float -> 'a dist -> 'a Primitive.t ->
Owl_stats.hypothesis
```
Perform chi-squared test, returns a `hypothesis` which is true if the null hypothesis is rejected

Up â ppl » Ppl » Plot

Module `Ppl.Plot`

Plotting utilies

`Plot` provides helper functions that wrap `Owl_plplot` to graph PPL distributions


```
type options = [

| `X_label of string
| `Y_label of string
| `Title of string
]
```

Histograms

```
val hist_dist_continuous : ? h:handle -> ? n:int -> ? fname:string ->
? options:options list -> float dist -> handle
val hist_dist_discrete : ? h:handle -> ? n:int -> ? fname:string ->
? options:options list -> float dist -> handle
```

Other Plots

```
val qq_plot : ? h:handle -> ? n:int -> ? fname:string -> ? options:options
list -> float dist -> float Primitive.t -> handle
val pp_plot : ? h:handle -> ? n:int -> ? fname:string -> ? options:options
list -> float dist -> float Primitive.t -> handle
val ecdf_continuous : ? h:handle -> ? n:int -> ? fname:string ->
? options:options list -> float Dist.dist -> handle
val ecdf_discrete : ? h:handle -> ? n:int -> ? fname:string ->
? options:options list -> float dist -> handle
val add_exact_pdf : ? scale:float -> dist:float Primitive.t -> handle ->
handle
val show : handle -> unit
```

Up â ppl » Ppl » Empirical

Module `Ppl.Empirical`

```
module type S = sig ... end
module Discrete : S
module ContinuousArr : sig ... end
```

Up â ppl » Ppl » Empirical » Discrete

Module `Empirical.Discrete`

```
type 'a t

val from_dist : ? n:int -> 'a Dist.dist -> 'a t
```
Create a empirical distribution from a distribution object, using n samples to approximate it

```
val empty : 'a t
```
Create an empty distribution

```
val add_sample : 'a t -> 'a -> 'a t
```
Add another sample to the distribution

```
val get_num : 'a t -> 'a -> int
```
Get the numer of samples with the value

```
val get_prob : 'a t -> 'a -> float
```
Get the probability of a particular value

```
val to_pdf : 'a t -> 'a -> float
```
Create a pdf function

```
val print_map : (module Core.Pretty_printer.S with type t = 'a) -> 'a t
-> unit
```
print the entire distribution

```
val to_arr : 'a t -> ('a * int) array
val to_norm_arr : 'a t -> ('a * float) array
val support : 'a t -> 'a list
```
Get the set of values for the distribution

Up â ppl » Ppl » Empirical » ContinuousArr

Module Empirical.ContinuousArr

```
type 'a t = {

samples : float array;
n : int;
max_length : int;
}

val empty : 'a t
val from_dist : ? n:int -> float Dist.dist -> 'a t
val add_sample : 'a t -> float -> 'b t
val to_cdf_arr : 'a t -> float array * float array
val to_pdf_arr : 'a t -> float array * float Core.Array.t
val to_cdf : 'a t -> Core.Float.t -> Core.Float.t
val to_pdf : 'a t -> Core.Float.t -> float
val values : 'a t -> float Core.Array.t
val print : 'a t -> Base.unit
```

Up â ppl » Ppl » Empirical » S

Module type `Empirical.S`

```
type 'a t

val from_dist : ? n:int -> 'a Dist.dist -> 'a t
```
Create a empirical distribution from a distribution object, using n samples to approximate it

```
val empty : 'a t
```
Create an empty distribution

```
val add_sample : 'a t -> 'a -> 'a t
```
Add another sample to the distribution

```
val get_num : 'a t -> 'a -> int
```
Get the numer of samples with the value

```
val get_prob : 'a t -> 'a -> float
```
Get the probability of a particular value

```
val to_pdf : 'a t -> 'a -> float
```
Create a pdf function

```
val print_map : (module Core.Pretty_printer.S with type t = 'a) -> 'a t
-> unit
```
print the entire distribution

```
val to_arr : 'a t -> ('a * int) array
val to_norm_arr : 'a t -> ('a * float) array
val support : 'a t -> 'a list
```
Get the set of values for the distribution