

Ppl (ppl.Ppl)

Table of Contents

Module Ppl.....	1
<u>Submodules.....</u>	1
<u>Condition Operators.....</u>	2
<u>Monad Ffunctions.....</u>	2
<u>Primitives.....</u>	2
<u>Sampling.....</u>	3
<u>Prior Distribution.....</u>	3
<u>Helpers.....</u>	3
<u>Exact Inference.....</u>	3
<u>Importance Sampling.....</u>	3
<u>Rejection Sampling.....</u>	4
<u>Sequential Monte Carlo.....</u>	4
<u>Metropolis Hastings.....</u>	4
<u>Particle Independent Metropolis Hastings.....</u>	4
<u>Particle Cascade.....</u>	4
<u>Common.....</u>	5
<u>Samples.....</u>	5
<u>Printing.....</u>	5
<u>Others.....</u>	6
Module Dist.PplOps.....	7
Module Ppl.Dist.....	8
<u>Condition Operators.....</u>	8
<u>Monad Ffunctions.....</u>	8
<u>Primitives.....</u>	9
<u>Sampling.....</u>	9
<u>Prior Distribution.....</u>	9
Module Ppl.Inference.....	10
<u>Helpers.....</u>	10
<u>Exact Inference.....</u>	10
<u>Importance Sampling.....</u>	10
<u>Rejection Sampling.....</u>	10
<u>Sequential Monte Carlo.....</u>	11
<u>Metropolis Hastings.....</u>	11
<u>Particle Independent Metropolis Hastings.....</u>	11
<u>Particle Cascade.....</u>	11
<u>Common.....</u>	11
Module type Primitive.PRIM_DIST.....	13
Module Ppl.Primitive.....	14
<u>New Distributions.....</u>	14
<u>Predefined Distributions.....</u>	14
<u>Basic Operations.....</u>	14
<u>Other.....</u>	15

Table of Contents

<u>Module Ppl.Helpers</u>	16
<u>Samples</u>	16
<u>Printing</u>	16
<u>Others</u>	16
<u>Module Ppl.Evaluation</u>	17
<u>KL-Divergence</u>	17
<u>Hypothesis Tests</u>	17
<u>Module Ppl.Plot</u>	18
<u>Histograms</u>	18
<u>Other Plots</u>	18
<u>Module Ppl.Empirical</u>	19
<u>Module Empirical.Discrete</u>	20
<u>Module Empirical.Continuous</u>	21
<u>Module type Empirical.S</u>	22

Module Pp1

DSL for Probabilistic Programming

Universal PPL in OCaml

- Submodules

Submodules

```
module Plot : sig ... end
  Plotting utilities
```

```
module Primitive : sig ... end
  Module defining a type for primitive distributions
```

```
module Evaluation : sig ... end
  A module for evaluating the correctness of models and inference procedures
```

```
module Empirical : sig ... end
```

```
module Inference : sig ... end
  Implementation of inference algorithms
```

```
module Dist : sig ... end
  Module used for defining probabilistic models
```

```
module Helpers : sig ... end
  Utilities for working with distributions
```

```
module Samples : Empirical.S
  include Dist
```

Module used for defining probabilistic models

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

```
exception Undefined
```

```
type prob = float
  A type for which values need to sum to 1
```

```
type likelihood = float
  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 =
```

Ppl (ppl.Ppl)

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

Type for representing distributions

Condition Operators

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

Monad Ffunctions

```
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
```

Ppl (ppl.Ppl)

```
val bernoulli : likelihood -> bool dist  
val choice : likelihood -> '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 = Dist.PplOps  
    Operators for distributions
```

```
include Inference
```

Implementation of inference algorithms

Inference algorithms to be called on probabilistic models defined using Dist

```
exception Undefined
```

```
type 'a samples = ('a * float) 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 * float) 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
```

Rejection 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 * float) Dist.dist
val reject'' : 'a Dist.dist -> 'a option Dist.dist
val reject_transform_soft : 'a Dist.dist -> ('a * float) Dist.dist
val rejection_transform : ? n:int -> rejection_type -> 'a Dist.dist -> 'a Dist.dist
val rejection_soft : 'a Dist.dist -> ('a * float) option Dist.dist
val rejection_hard : ? threshold:Core.Float.t -> 'a Dist.dist -> ('a * float) 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 : burn:int -> 'a Dist.dist -> unit -> 'a
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 samples -> int -> 'a samples 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

include 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 * float) list -> ('a * float) list
    Removes duplicates and sums the probabilities associated so that each value appears once

val flatten : (('a * float) list * float) list -> ('a * float) list
val normalise : ('a * float) list -> ('a * float) list
val weighted_dist : ? n:int -> 'a Dist.dist -> ('a,  $\hat{\mathbb{A}}$  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
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 » Dist » PplOps

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
```

Up â _ppl » Ppl » Dist

Module Ppl.Dist

Module used for defining probabilistic models

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

- Condition Operators
- Monad Ffunctions
- Sampling
- Prior Distribution

exception Undefined

type prob = float
A type for which values need to sum to 1

type likelihood = float
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 =

| 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 -> float) * 'a dist -> 'a dist

variant that defines likelihood model

Type for representing distributions

Condition Operators

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

Monad Ffunctions

```
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
```

Ppl (ppl.Ppl)

```
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 : likelihood -> bool dist
```

```
val choice : likelihood -> '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 Pp1 . Inference

Implementation of inference algorithms

Inference algorithms to be called on probabilistic models defined using Dist

- Helpers
- Exact Inference
- Importance Sampling
- Rejection Sampling
- Sequential Monte Carlo
- Metropolis Hastings
- Particle Independent Metropolis Hastings
- Particle Cascade
- Common

exception Undefined

type 'a samples = ('a * float) 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 * float) 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
```

Rejection 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
```

Ppl (ppl.Ppl)

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

Ppl (ppl.Ppl)

- | 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 support : t support
```

Up â ppl » Ppl » Primitive

Module Ppl.Primitive

Module defining a type for primitive distributions

- New Distributions
- Predefined Distributions
- Basic Operations
- Other

```
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 <u>support</u> * 'a <u>support</u>	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 -> '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 sample : 'a t -> 'a  
val support : 'a t -> 'a support
```

Ppl (ppl.Ppl)

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](#)
- [Printing](#)
- [Others](#)

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 * float) list -> ('a * float) list
```

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

```
val flatten : (('a * float) list * float) list -> ('a * float) list
val normalise : ('a * float) list -> ('a * float) list
val weighted_dist : ? n:int -> 'a Dist.dist -> ('a, 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
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

- KL-Divergence
- Hypothesis Tests

```
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 -> 'a Primitive.t -> 'a 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 utilities

Plot provides helper functions that wrap Owl_plplot to graph PPL distributions

- [Histograms](#)
- [Other Plots](#)

Histograms

```
val hist_dist_continuous : ? h:Owl_plplot.Plot.handle -> ? n:int ->  
? fname:string -> float Dist.dist -> Owl_plplot.Plot.handle  
val hist_dist_discrete : ? h:Owl_plplot.Plot.handle -> ? n:int ->  
? fname:string -> float Dist.dist -> Owl_plplot.Plot.handle
```

Other Plots

```
val qq_plot : ? h:Owl_plplot.Plot.handle -> ? n:int -> ? fname:string ->  
float Dist.dist -> float Primitive.t -> Owl_plplot.Plot.handle  
val prob_plot : ? h:Owl_plplot.Plot.handle -> ? n:int -> ? fname:string ->  
float Dist.dist -> float Primitive.t -> Owl_plplot.Plot.handle
```

Up â _ppl » Ppl » Empirical

Module Ppl.Empirical

```
module type S = sig ... end
module Discrete : S
module Continuous : 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 number 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](#) » Continuous

Module Empirical.Continuous

```
type bin = float * float
type element = float
type endpoints = {
  start : float;
  finish : float;
}
type 'a t = Owl_stats.histogram

val empty : Owl_stats.histogram
val from_dist : ? n:int -> float Dist.dist -> Owl_stats.histogram
```

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
```