Non-Parametric representation of density functions

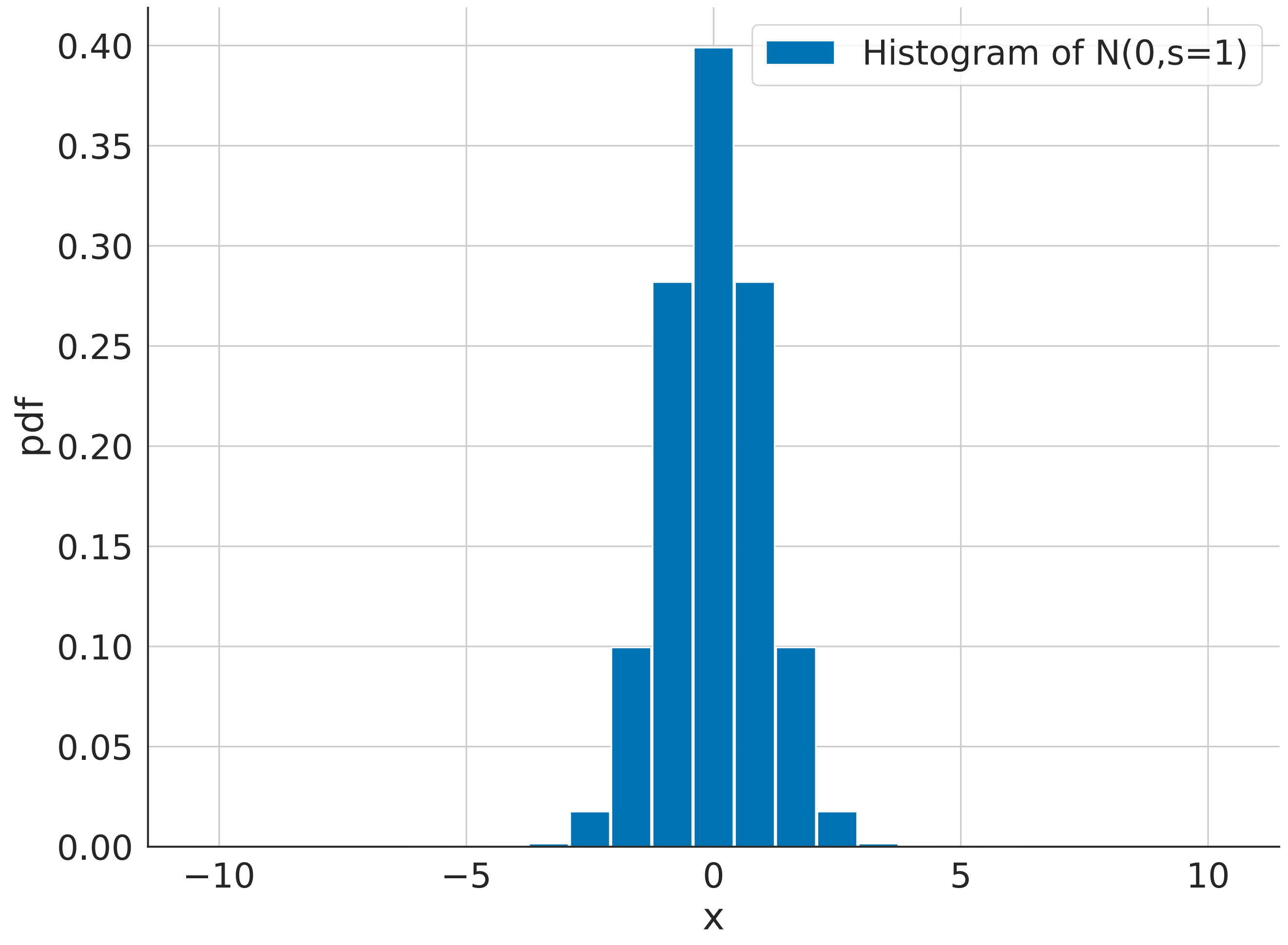
Density functions

As we have discussed (class 2), one way to represent density functions is by using parametric distributions.

> ## Parametric Functions
>
> Family of functions that are commonly used to model the probability distribution $p(x)$ of a rv x, given a finite set $\{x_1, \cdots, x_N\}$ of observations.
> The functions are fully defined by few `parameters`
> E.g., Gaussian (Normal) Distribution
>
> $$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$
>
> with parameters: μ and σ.

Histogram representation

Instead of using a parametric function, it is possible to discretize the continuous rv, and represent the equivalent probability mass function using a histogram.



Sample representation

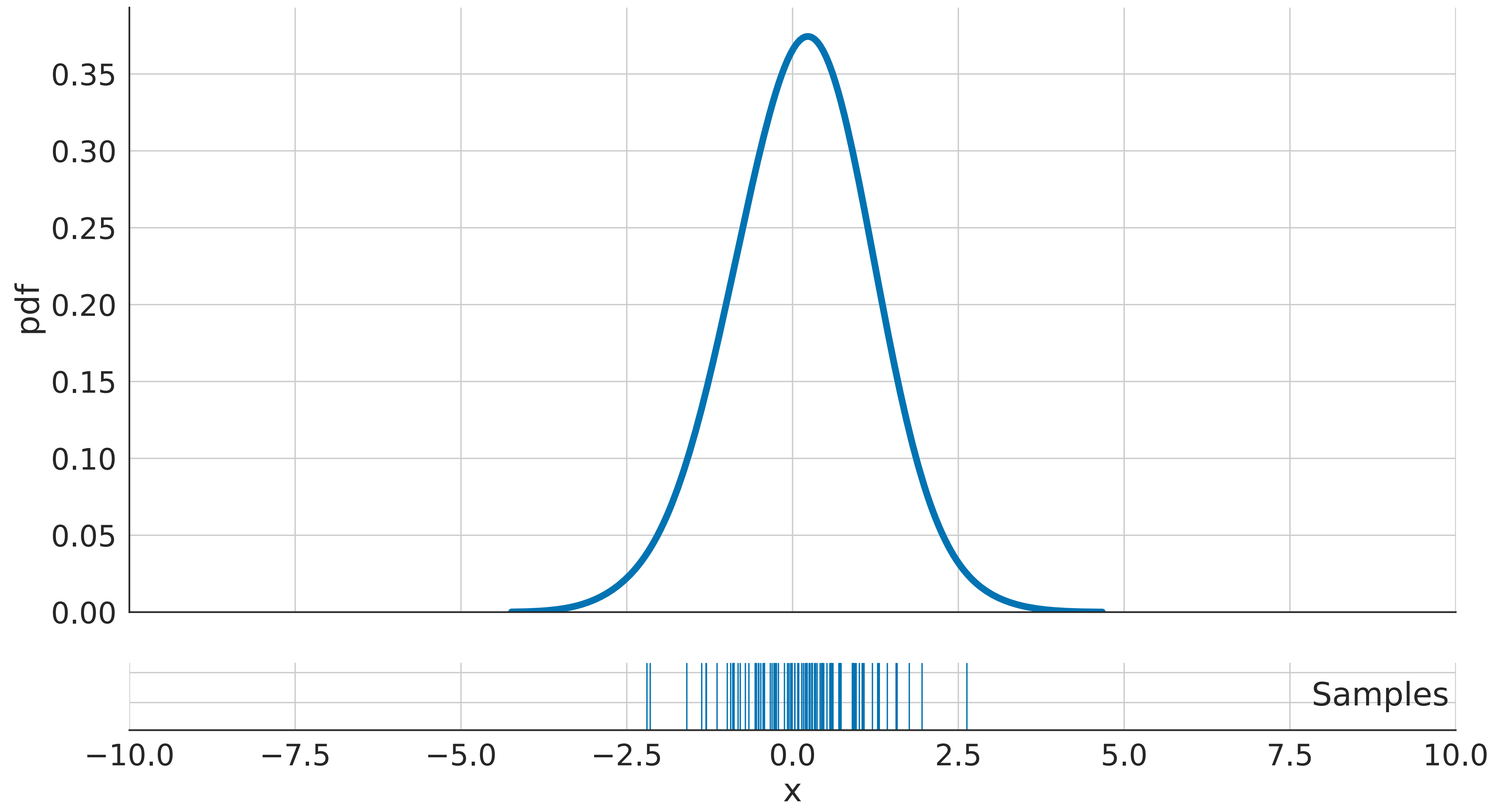
A different approach to represent density functions is by using a set of samples taken directly from the distribution

For example, for the density function $p(x) = \mathcal{N}(x|\mu = 0, \sigma = 1)$

We can represent its density function as a set of K samples $x^{(k)}$ drawn randomly sampled from the function's support $(-\infty, \infty)$ with probability $w = p(x)$.

I.e, sample in a way that there is a probability $p(0.1)$ of sampling $x = 0.1$, a probability $p(1)$ of sampling $x = 1$, etc.

Sample representation



Sample representation

Reconstructing the density function

If we want to reconstruct the density at a point x, we count how many samples are in the vicinity of x (and then normalize for $\int p(x) = 1$).

For example, to compute $p(1)$ we can count how many samples are between $x = 0.9$ and $x = 1.1$ (equivalent to finding the histogram of the samples)

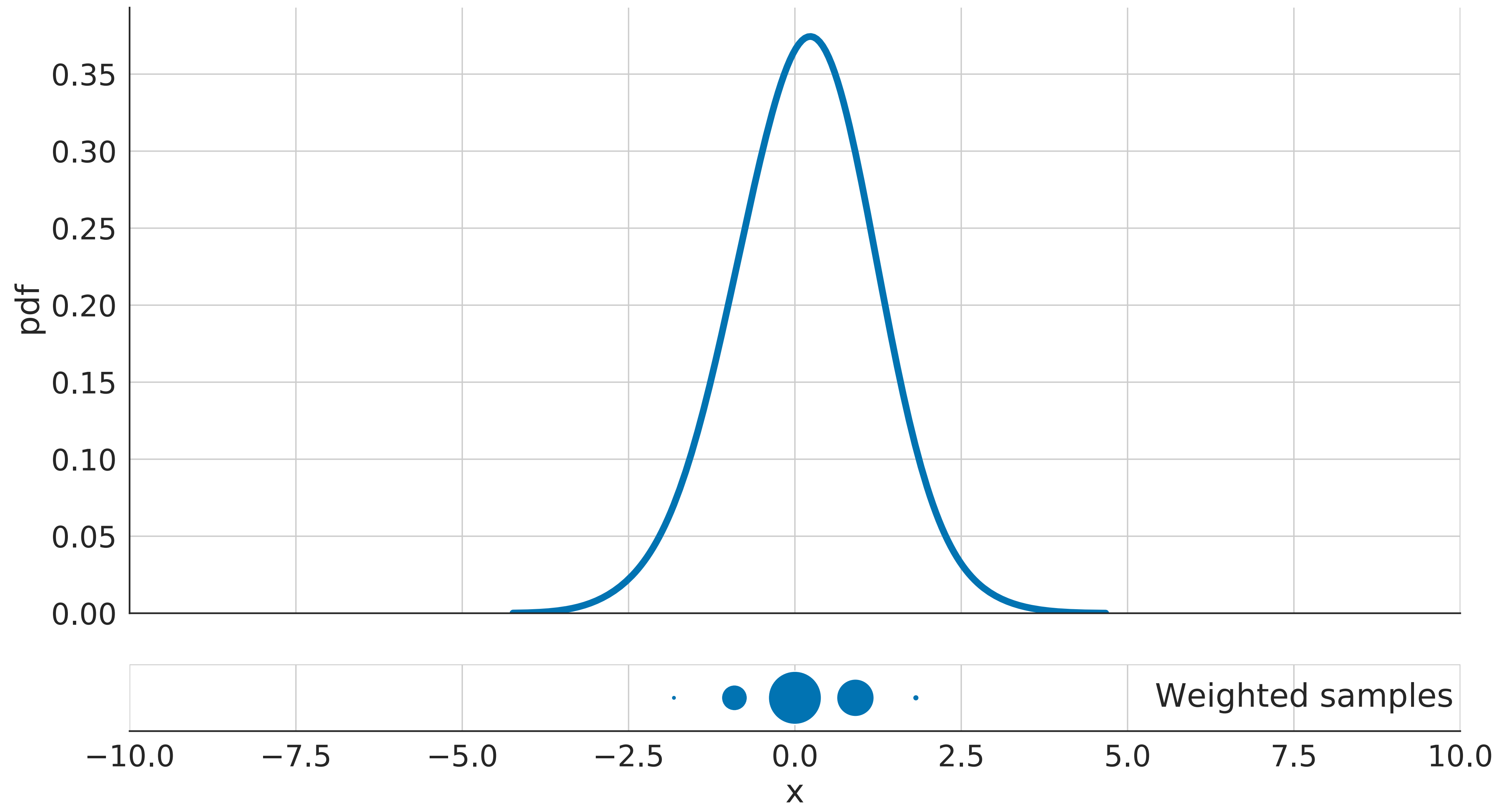
`Note` **Using $\pm$ 0.1 as the vicinity of the point is just an example, the ideal value for this vicinity (often called bandwidth) is computed depending on the function**

For more details on how to reconstruct densities from samples, read chapter 2.5 on the book Pattern Recognition and Machine Learning by Christopher Bishop or read online resources regarding kernel density estimation

Weighted Sample representation

We can also take K weighted samples $\langle x^{(k)}, w^{(k)} \rangle$, where each sample has an associated weight $w = p(x)$, so that when reconstructing the density, instead of counting the number of samples in the vicinity of a point, we add the weights of all the particles in the vicinity of the point.

Sample representation



Weighted Sample representation

Reconstructing the density function

Example,

For weighted particles

$$\{\langle 0.8, 1\rangle, \langle 0.9, 2\rangle, \langle 1.0, 3\rangle, \langle 1.1, 2\rangle\}$$

considering the vicinity of ± 0.1 for x in $[0.7, 0.8, 0.9, 1.0, 1.1, 1.2]$

$$p(x) \propto [1, 3, 6, 7, 5, 2]$$

After normalization we would have

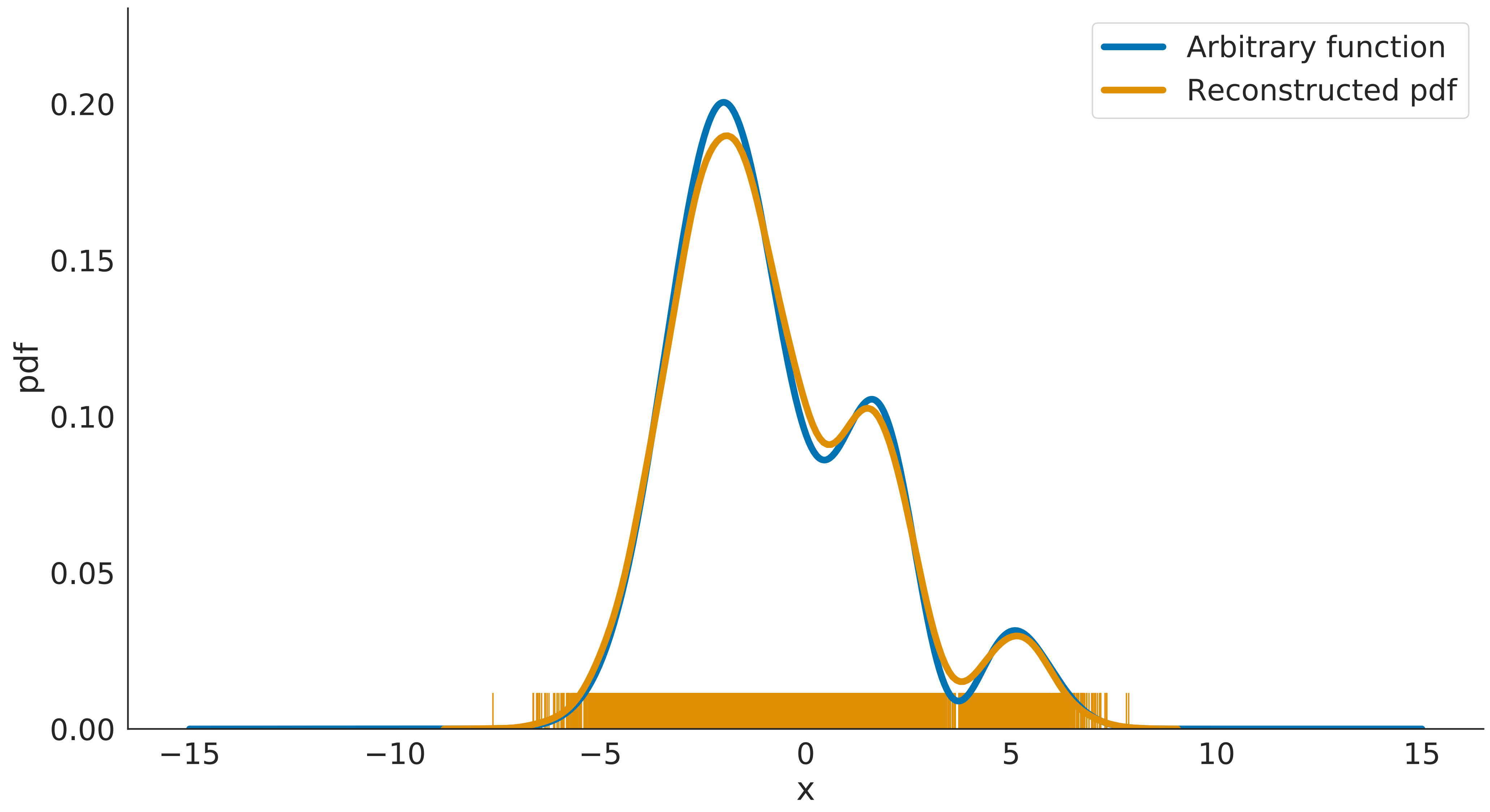
$$p(x) = [0.042, 0.125, 0.25, 0.292, 0.208, 0.083]$$

Representing arbitrary density functions

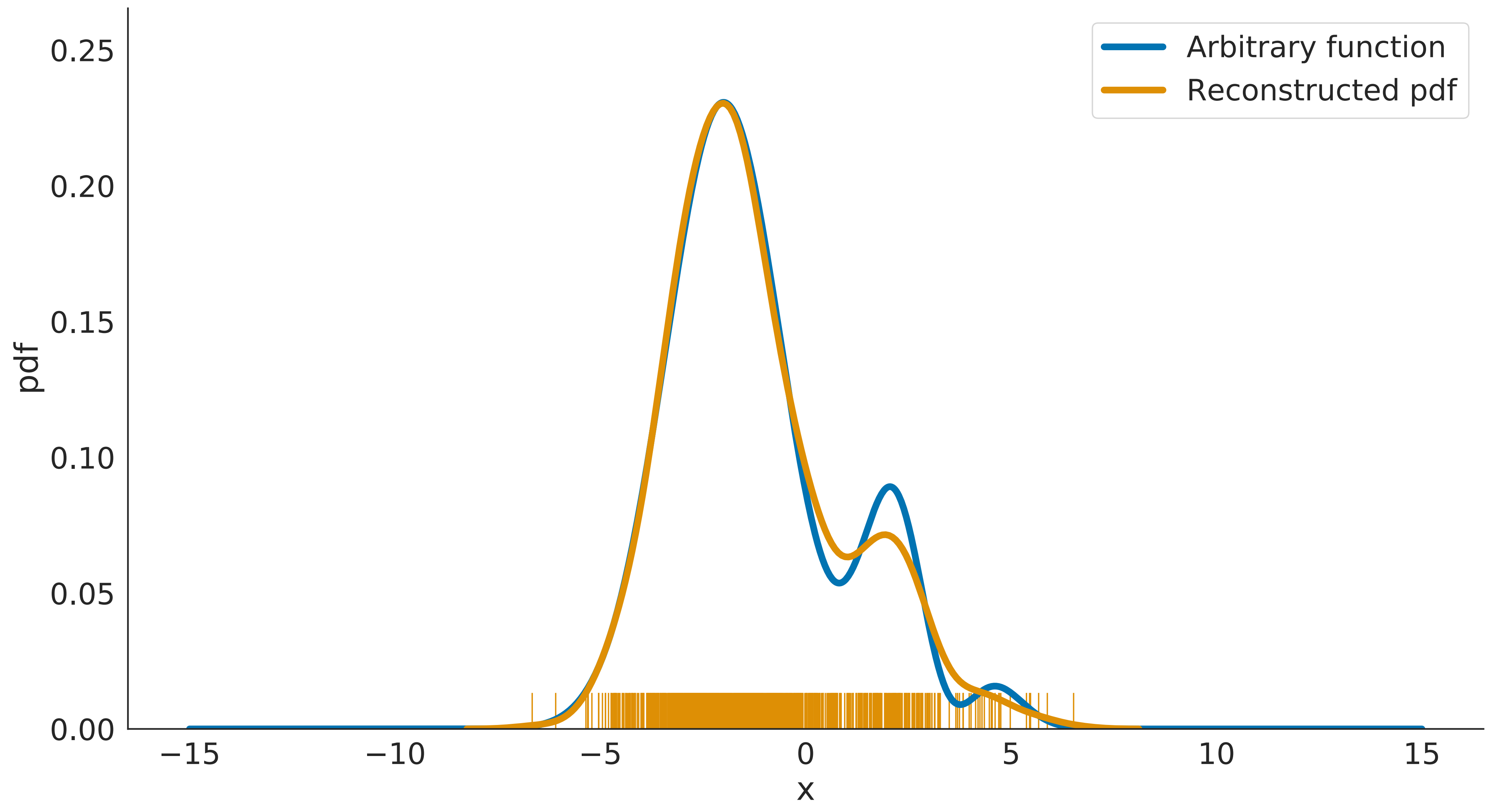
Samples can be used to represent any density function.

However, the accuracy of the reconstruction depends on the number of samples used.

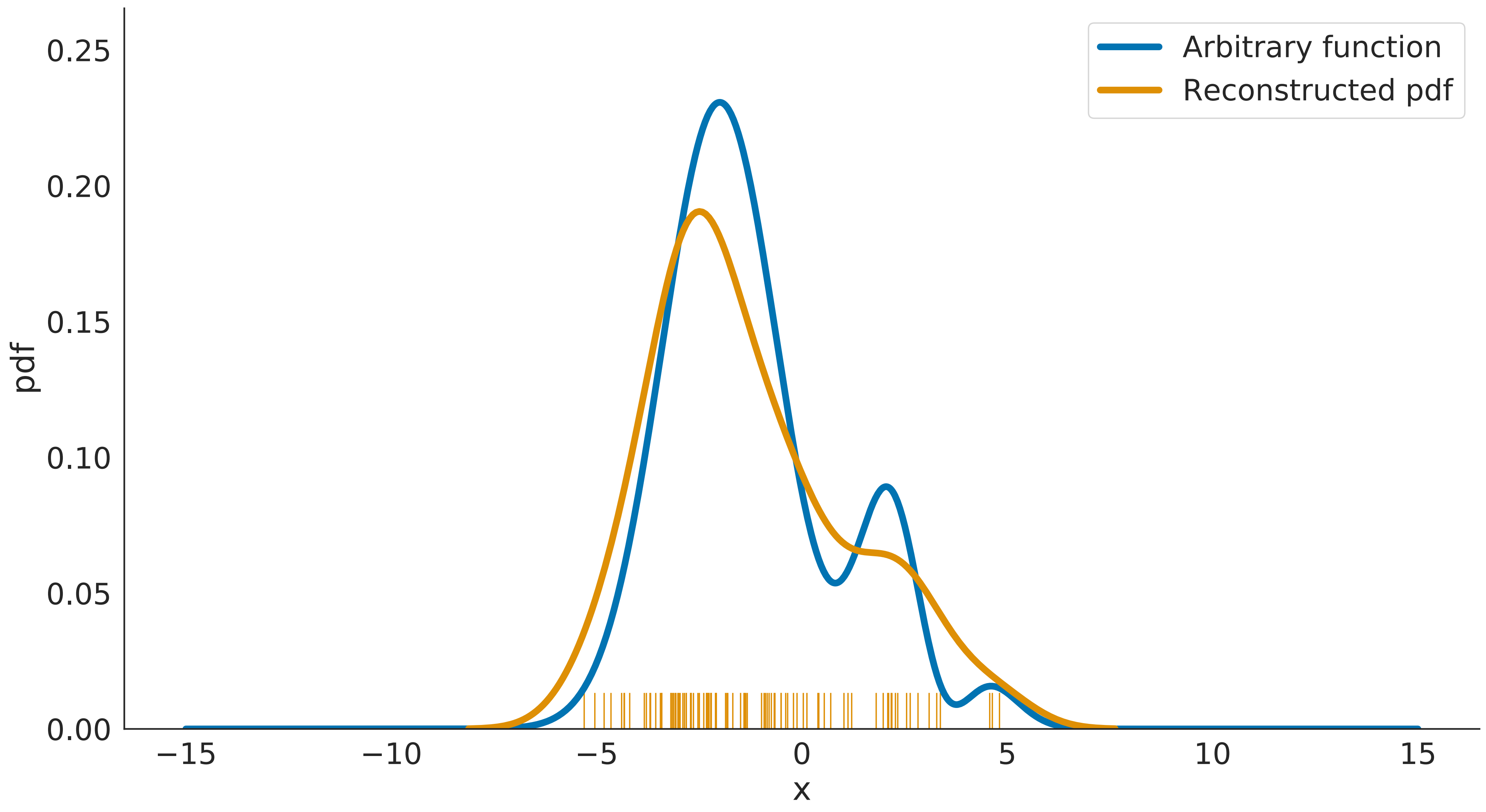
Representing an arbitrary function using 10K samples



Representing an arbitrary function using 1K samples



Representing an arbitrary function using 100 samples



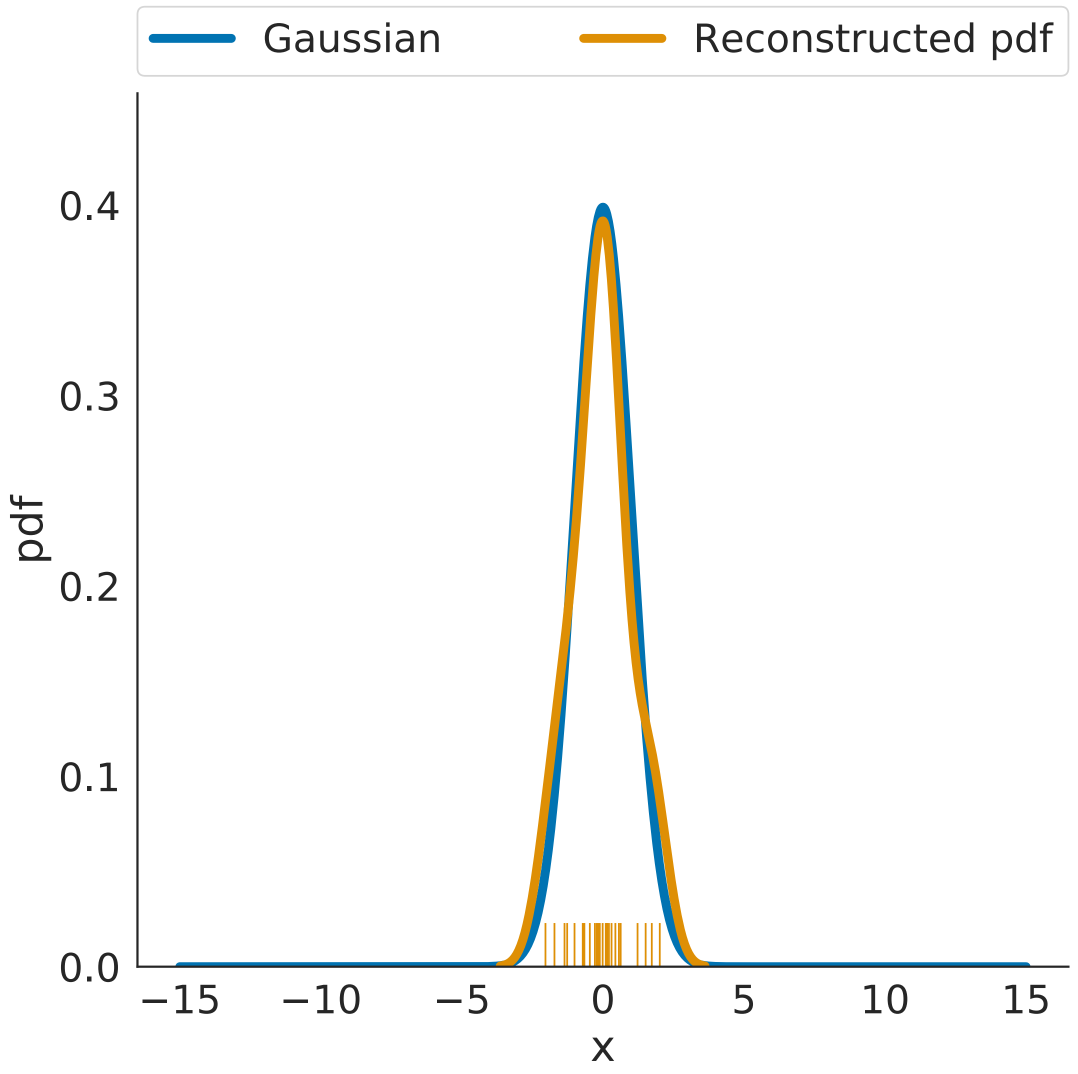
How many samples do we need?

It depends ...

Shape of the function

Simpler functions require less samples

For the arbitrary density around 10K samples were required. For a Gaussian distribution, around 25 samples can make a good reconstruction

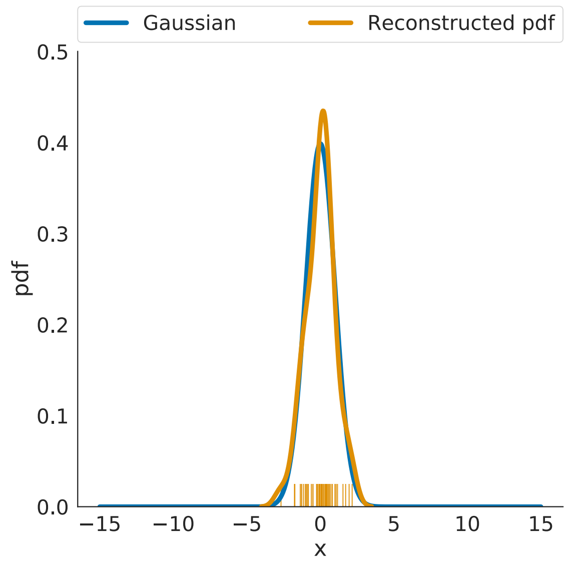


How many samples do we need?

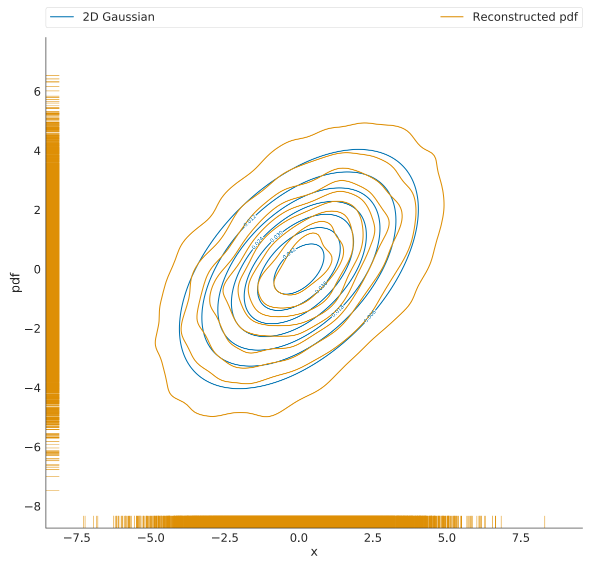
It depends ...

Dimensionality of x

The higher the number of dimensions the exponentially larger number of samples required. A 2D vector x requires much more samples than a 1D vector x.



50 samples



10K samples

Why not use a large number of particles?

Computational resources

Histogram Filter and Grid Localization

Histogram Filters

Implementation of the recursive Bayes Filter where posterior distributions are represented by discretizing the state space into finitely many regions and representing the probability of each region by a single probability value.

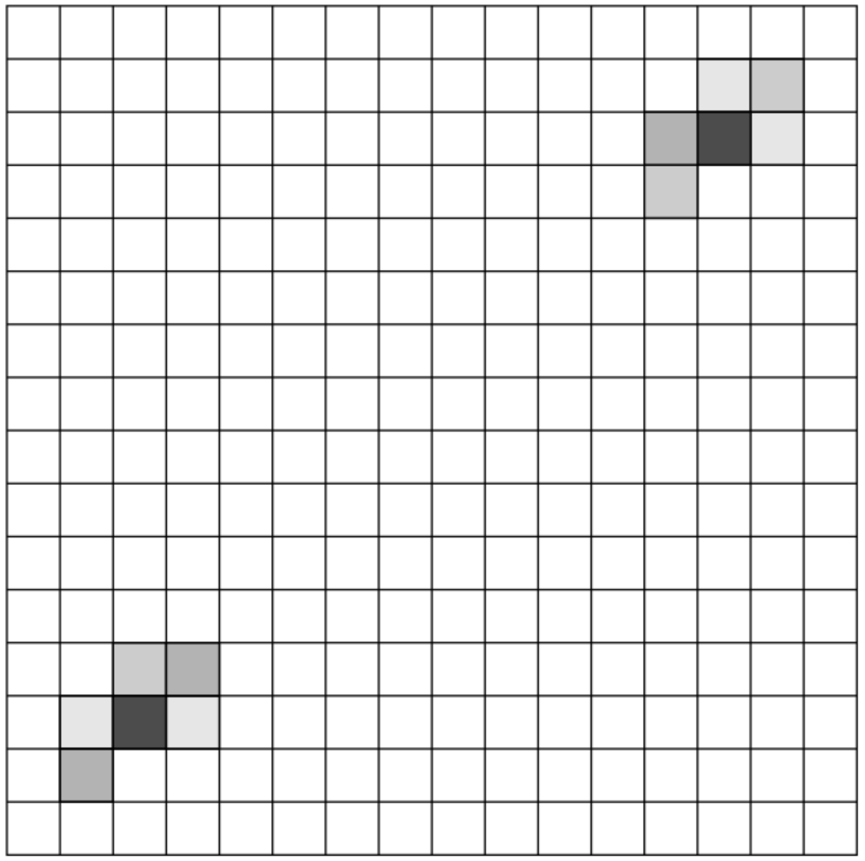
The posterior distribution over each region is updated using Bayes Rule over the discretized support.

State space decomposition

Static Techniques

The decomposition (discretization) is fixed and chose in advance, irrespective of the shape of the posterior to approximate.

E.g. Regularly spaced grid

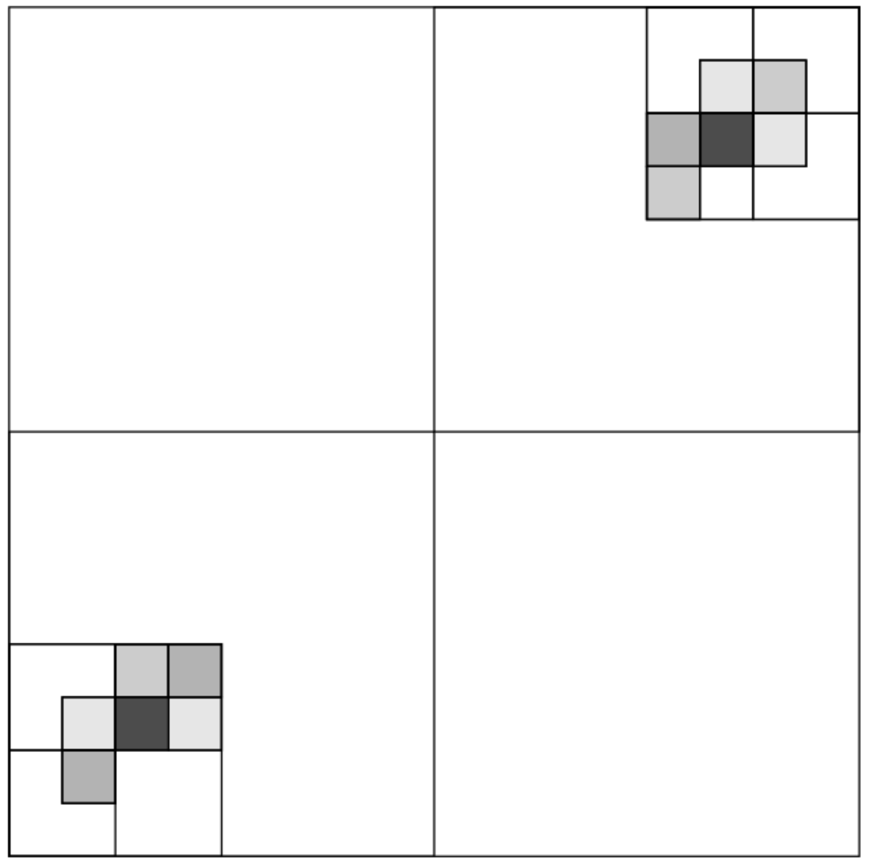


State space decomposition

Dynamic techniques

Adapt the decomposition to the specific shape of the posterior distribution.
The less likely a region, the coarser the decomposition

E.g. Density Trees



Grid Localization

Use of Histogram Filter for the particular case of **Robot Localization**

Why use Grid Localization?

Solve the limitations of the Kalman Filters

1. Can work with non-gaussian distributions
2. Can work with any non-linear model
3. Can address the Global Localization Problem

Grid Localization

As any Bayes Filter, it has two steps

1. Prediction
2. Correction

Prediction

For the centroid of each region, $x_k \in \{x_0, \cdots, x_{K-1}\}$,compute the posterior probability using the motion model.

$$\bar{p}(x_k) = \sum_i \text{motion\_model}(x_k, u_t, x_i)$$

Correction

For the centroid of each region, compute the posterior distribution after incorporating sensor measurements as

$$p(x_k) = \eta \text{measurement\_model}(z, x_k) \bar{p}(x_k)$$

where η is a normalization constant so $\sum_k p(x_k) = 1$

Grid Resolution

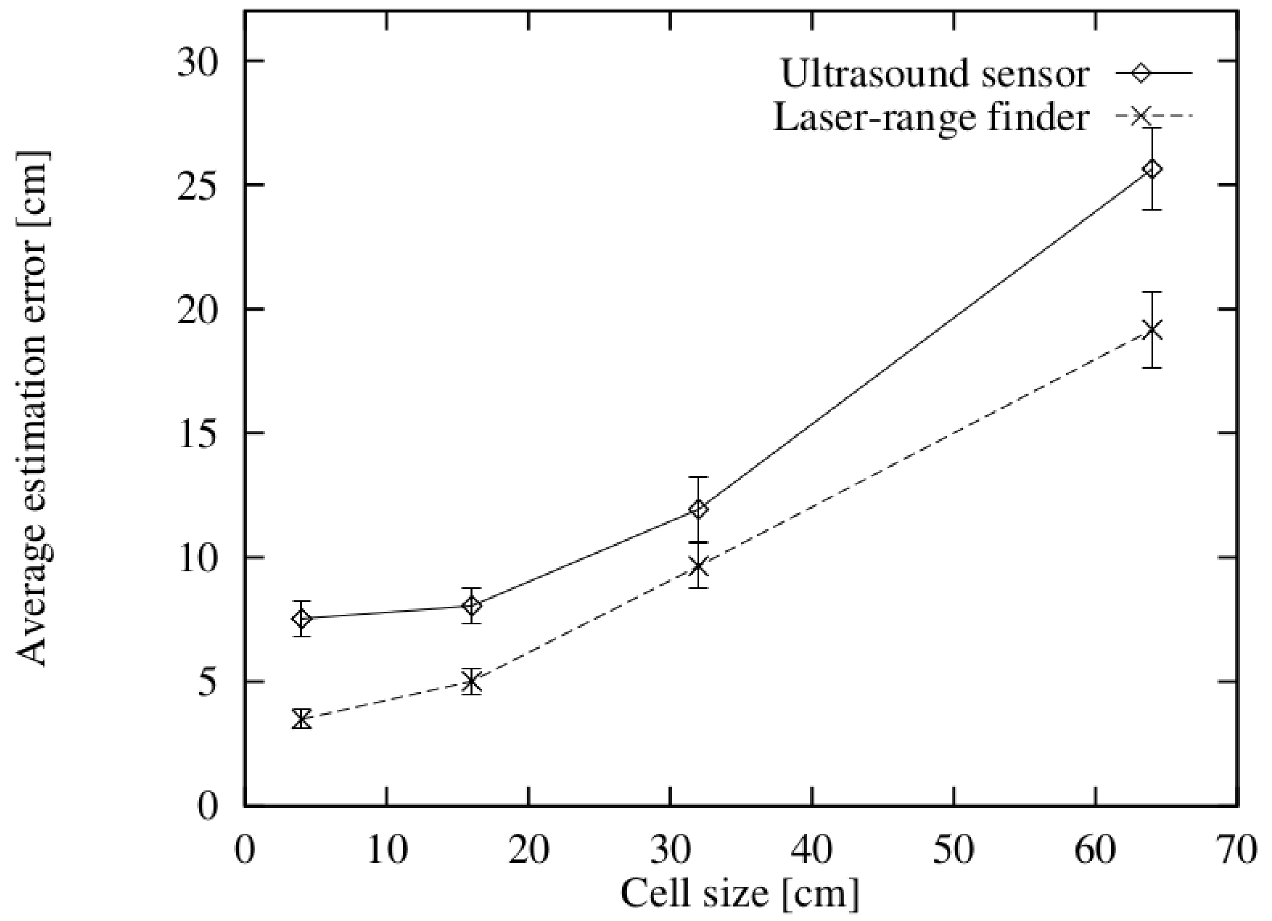
A key parameter for the grid localization algorithm is the resolution of the grid used for discretization.

- Corse, variable-resolution grids
  The space is decomposed into regions with unique characteristics or landmarks.
  E.g., into corridors/rooms, areas near whichever landmark is used, such as windows/tables/doors.
- Fine, fixed-resolution grids
  Equally spaced grids. Cell sizes vary depending on the application and desired accuracy.
  For indoors cell sizes vary between 5-15 cm, while for outdoors they are often between 20-50 cm.

Grid Resolution and Accuracy

In general, smaller grids allow for higher accuracy (at the cost of increased computational cost)

The exact trade-off depends heavily on the accuracy of the sensor



Particle Filters and Monte Carlo Localization

Particle Filters

Implementation of the Bayes Filter, where distributions are represented with samples instead of parametric functions.
Each possible state is called a *Particle*, hence the name.

Monte Carlo Localization

Is the particular application of Particle Filters for **Robot Localization**
For robot localization in 2D, each particle would be $x^{(i)} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$

Why use MCL?

Solve the limitations of the Kalman Filters

1. Can work with non-gaussian distributions
2. Can work with any non-linear model
3. Can address the Global Localization Problem

More efficient (computation and memory-wise) than Grid Localization

MCL

As any Bayes Filter, it has two steps

1. Prediction
2. Correction

Prediction

Apply the motion model (with sampled noise) to each particle.

for all $x^{(i)}$ apply

$$x_t^{(i)} = g(x_{t-1}^{(i)}, u_t) + \mathcal{N}(0, \sigma)$$

Example

Considering particles $x = \{1, 1.2, 0.8, 1.8\}$, and $g(x, u) = x + 1$ (i.e, robot moved 1 meter)

1. Apply $g(x, u)$ to each particle.
   $x = \{1, 1.2, 0.8, 1.8\} \rightarrow \{2, 2.2, 1.8, 2.8\}$
2. Sample some white noise with standard deviation σ,
   For example: $\{0.4, -0.4, -0.6, 0.4\}$
3. Add the noise to the samples
   $x = \{2, 2.2, 1.8, 2.6\} \rightarrow \{2.4, 1.8, 1.2, 3.2\}$

Important: Do not use the same sampled noise for each particle, nor the same set for each update. Always get new noise samples.

Correction

For the correction step,
Compute $w^{(i)} = p(z|x^{(i)})$ to obtain a set of weighted samples $\langle x^{(i)}, w^{(i)} \rangle$

Example

Consider the robot observes a landmark at distance $z = 3$, and the corresponding landmark is located in the feature map at $x = 5$.

Compute w considering the following measurement model (Gaussian with variance=1)

$$p(z|x) = \varphi(z - (5 - x)) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\big(z - (5 - x)\big)^2 \right)$$

For the correction step, apply $p(z = 3|x^{(i)}) = \varphi(x^{(i)} - 2)$ to each particle

For x computed in the prediction step: $x = \{2.4, 1.8, 1.2, 3.2\}$
That is,

$$w^{(i)} = \varphi(x^{(i)} - 2)$$

Resulting in

$$w = \{\varphi(0.4), \varphi(-0.2), \varphi(-0.8), \varphi(1.2)\} = \{0.37, 0.39, 0.29, 0.19\}$$

Recursive implementation

We can keep applying the prediction and correction step for each new measurement by updating the previous particles' weights as the product of their previous weights times the new weights from $p(z|x)$)

$$
\begin{aligned}
\text{For } t = 1: \quad & x_1^{(i)} = g(x_0^{(i)}, u_1) + \mathcal{N}(0, \sigma) \quad , \quad w_1^{(i)} = p(z_1 | x_1^{(i)}) \\
\text{For } t = 2: \quad & x_2^{(i)} = g(x_1^{(i)}, u_2) + \mathcal{N}(0, \sigma) \quad , \quad w_2^{(i)} = w_1^{(i)} p(z_2 | x_2^{(i)}) \\
\vdots \\
\text{For } t = n: \quad & x_n^{(i)} = g(x_{n-1}^{(i)}, u_n) + \mathcal{N}(0, \sigma) \quad , \quad w_n^{(i)} = w_{n-1}^{(i)} p(z_n | x_n^{(i)})
\end{aligned}
$$

Resampling

If we have infinite computational resources, we can use an extremely large set of particles to adequately represent distributions.

However, we do not. To better allocate our resources we want to move particles from areas of low probabilities to areas of higher probabilities.

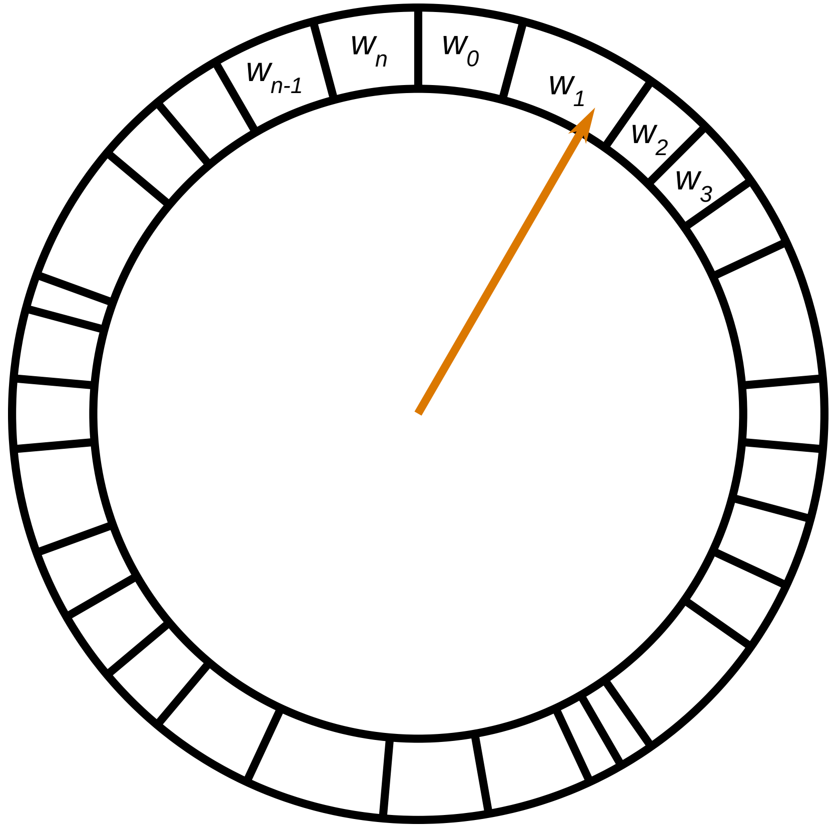
Resampling

To do this, we can transform our weighted samples $\langle x^{(i)}, w^{(i)} \rangle$ into a new set of samples $\langle x^{(i)} \rangle$ that represent the same distribution. This process is called *resampling*

Resampling Wheel

A simple implementation of this procedure is to imagine that the samples are located in a roulette wheel, with the arc of each sample being proportional to its weight.

You spin the roulette and wherever the arrow falls, you take that value as a new sample



Resampling wheel

Implementation

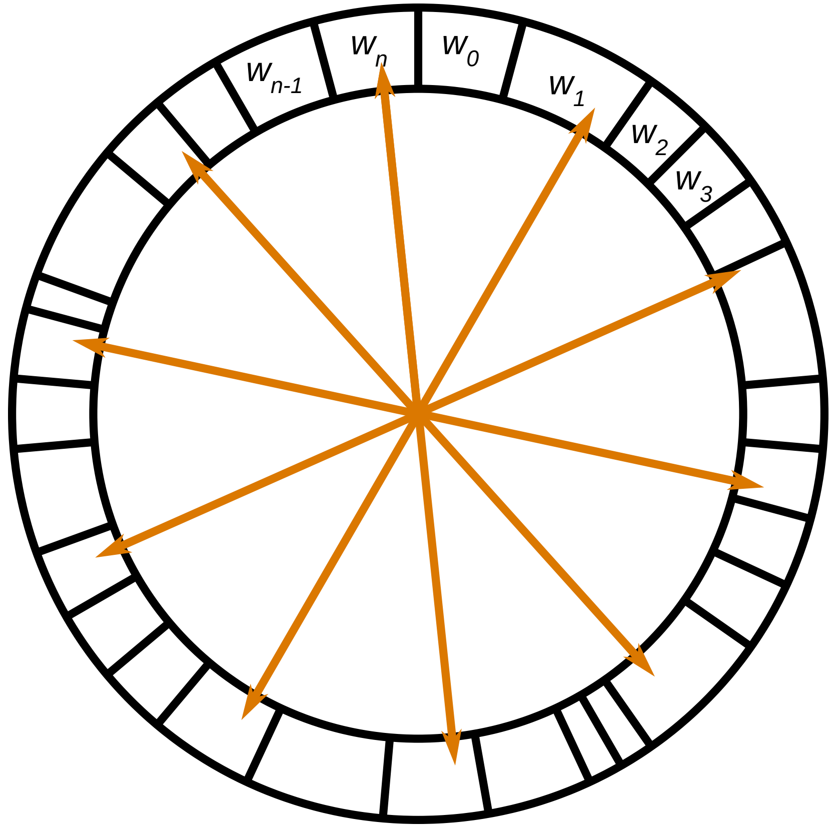
To sample M particles

1. Normalize weights, so they add up to 1
   $w_i = w_i / \sum w_i$
2. For M times
   i. draw a random number between 0 and 1
      $u \sim \text{uniform}(0, 1)$
   ii. Find the particle i that satisfies $\sum_{0:i} w_i < u < \sum_{0:i+1} w_i$
   iii. Add particle x_i

Low variance sampler

A more efficient and optimal implementation of the resampling wheel

To acquire M samples, imagine a roulette with M evenly distributed arrows.
Now, spin the roulette only once, and get the M samples where the arrows fall.



Low variance sampler

Implementation

1. Normalize weights, so they add up to 1
   $w_i = w_i / \sum w_i$
2. Draw a random number between 0 and $1/M$
   $u \sim \text{uniform}(0, 1/M)$
3. For m in $0 : M - 1$
    i. Find the particle i that satisfies $\sum_{0:i} w_i < u + m/M < \sum_{0:i+1} w_i$

Resampling wheel vs Low variance sampler

Low variance sampler should always be chosen

- Computationally Faster, $\mathcal{O}(M)$ vs $\mathcal{O}(MlogM)$
- Avoids particle depletion/starvation

MCL

Considering resampling, our algorithm becomes

1. Prediction
   $\bar{x}^{(i)} = g(x^{(i)}, u) + \mathcal{N}(0, \sigma)$
2. Correction
   $w^{(i)} = p(z|\bar{x}^{(i)})$
3. Resampling
   $x^{(i)} \sim \langle \bar{x}^{(i)}, w^{(i)} \rangle$

Prediction

Simply apply the motion model (with sampled noise) to each particle.

For the odometry model

$$\begin{aligned}
\hat{\delta}_{rot1} &\sim \mathcal{N}(\delta_{rot1}, \alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2) \\
\hat{\delta}_{trans} &\sim \mathcal{N}(\delta_{trans}, \alpha_3 \delta_{trans}^2 + \alpha_4 (\delta_{rot1}^2 + \delta_{rot2}^2)) \\
\hat{\delta}_{rot2} &\sim \mathcal{N}(\delta_{rot2}, \alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2)
\end{aligned}$$

$$\begin{bmatrix} x^{(i)} \\ y^{(i)} \\ \theta^{(i)} \end{bmatrix} = \begin{bmatrix} x^{(i)} + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1}) \\ y^{(i)} + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1}) \\ \theta^{(i)} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \end{bmatrix}$$

Correction

For the correction step, compute

$$w^{(i)} = p(z|x^{(i)})$$

Resampling

Obtain a new set of particles with probability proportional to $w^{(i)}$

$$x^{(i)} = low\_variance\_sampler(\langle x, w \rangle^{(i)})$$

Video from http://robots.stanford.edu/videos.html

Cons of MCL

Computational requirements are much higher than for EKF

MCL variants

Adaptive Monte Carlo Localization (AMCL)

An implementation that dynamically decides the number of particles to use

The core idea is to use a large number of particles when the posterior distribution is spread around the environment (i.e., the location is uncertain), and reduce it once the localization converges.

AMCL: KLD-Sampling

One implementation uses the Kullback-Leibler Divergence to determine how many particles suffice to approximate the true posterior.

The implementation stores a grid of K cells (all initially empty)
It takes a new sample and sorts it into the grid, if it ends in an empty bin, the bin is *filled*, and the number of occupied bins k is increased by 1.
The number of required particles is a function of k.

KLD is a measure of the difference between two arbitrary probability distributions (they do not have to be Gaussian or uni-modal. More details at: Probabilistic Robotics, Chapter 8.3.7

Dual Monte Carlo Localization (dMCL)

In the standard MCL algorithm, we perform

1. Prediction using the Motion Model
2. Correction using the Perceptual Model
3. Resampling

For the Dual MCL, we perform

1. Prediction using the Perceptual Model
2. Correction using the Motion Model
3. Resampling

dMCL

Specifically

1. $x_t^{(i)} \sim p(z_t | x_t)$
2. $w^{(i)} = \sum_j p(x_t^{(i)} | x_{t-1}^{(j)}, u_t)$ Correction using the Motion Model
3. Resampling

This involves sampling from the perceptual model, which depending on the model may not be trivial.
This approach is immune robust to the Kidnapped Robot Problem, but localization accuracy is typically lower

Mixed Monte Carlo Localization

A portion of the particles are updated using MCL, while the remainder are updated using dMCL

Variations focused on Robustness

Random Particle MCL

At the resampling step, a portion of the particles are drawn at random

The portion of the samples is determined using the ratio of *fast* to *slow* average of particle weights

Sensor Resetting Localization

Resets the set of particles when the algorithm detects a localization failure

S. Lenser and M. Veloso, "Sensor resetting localization for poorly modeled mobile robots", 2000 IEEE International Conference on Robotics and Automation (ICRA), pp. 1225–1232.

Expansion Resetting

Artificially inflates the particles (by increasing the noise in the odometry model) when the algorithm detects a localization failure

R. Ueda, T. Arai, K. Sakamoto, T. Kikuchi, and S. Kamiya, "Expansion resetting for recovery from fatal error in Monte Carlo localization comparison with sensor resetting methods", 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2481–2486

Resetting/Bootstrapping

Resets the set of particles when the algorithm detects a localization failure. First running dMCL until convergence, then using MCL

Y. Seow, R. Miyagusuku, A. Yamashita, and H. Asama, "Detecting and Solving the Kidnapped Robot Problem Using Laser Range Finder and Wifi Signal, 2017 IEEE International Conference on Real-time Computing and Robotics (RCAR), 2017, pp. 303-308