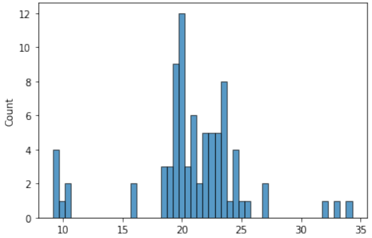
Nonparametric Involutive Markov Chain Monte Carlo

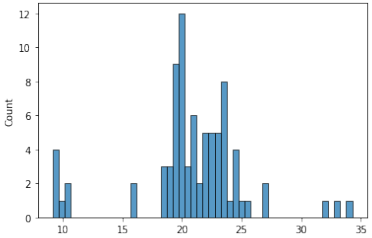
Carol Mak Fabian Zaiser Luke Ong
University of Oxford
ICML 2022

Probabilistic Programming



Probabilistic Programming

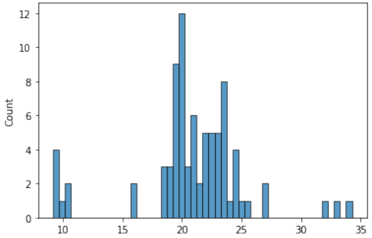
How many Gaussian mixtures there are?



Probabilistic Programming

How many Gaussian mixtures there are?

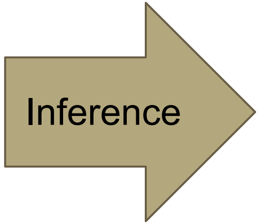
```
# parameter
K ~ Normal(3,1)

# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))

# data
for j in 1:len(data)
    data[j] ~ m
```

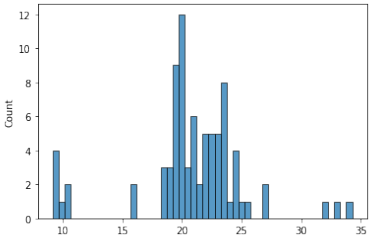


Probabilistic Programming

How many Gaussian mixtures there are?

```
# parameter
K ~ Normal(3,1)

# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))

# data
for j in 1:len(data)
    data[j] ~ m
```



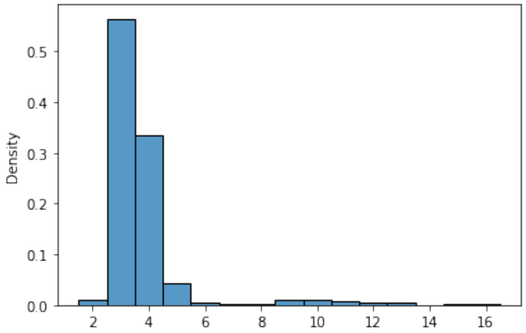


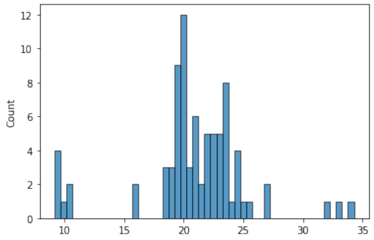
Probabilistic Programming

How many Gaussian mixtures there are?

```
# parameter
K ~ Normal(3,1)

# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))

# data
for j in 1:len(data)
    data[j] ~ m
```

Inference



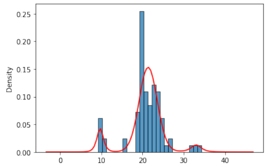


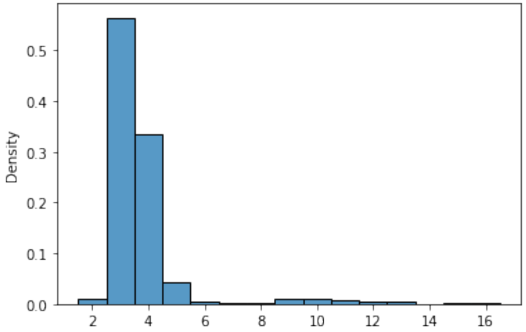
Probabilistic Programming

How many Gaussian mixtures there are?

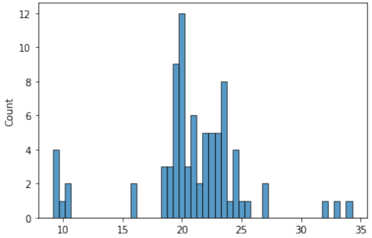
```
# parameter
K ~ Normal(3,1)

# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))

# data
for j in 1:len(data)
    data[j] ~ m
```

Inference





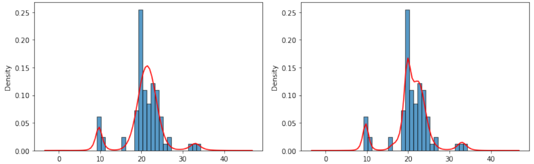
Probabilistic Programming

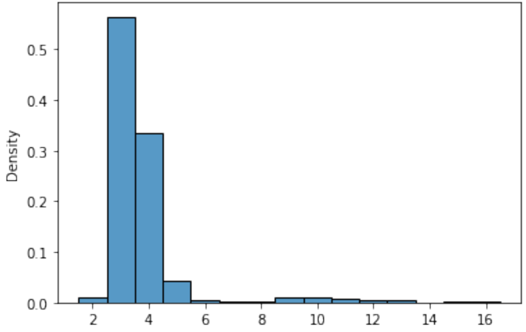


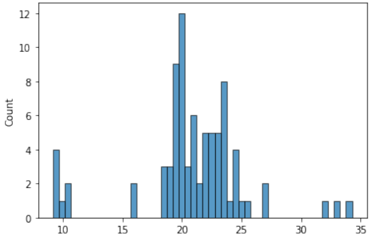
How many Gaussian mixtures there are?

```
# parameter
K ~ Normal(3,1)

# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))

# data
for j in 1:len(data)
    data[j] ~ m
```

Inference



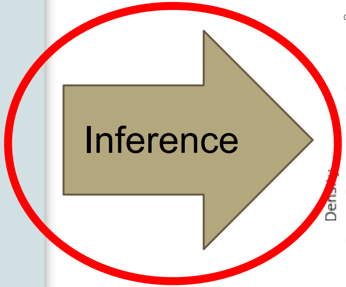


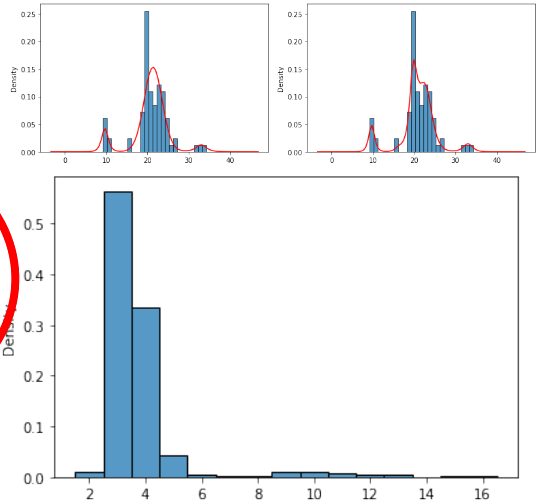


Probabilistic Programming

How many Gaussian mixtures there are?

```
# parameter
K ~ Normal(3,1)

# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))

# data
for j in 1:len(data)
    data[j] ~ m
```



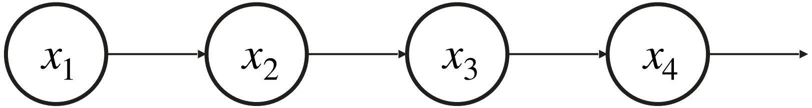


Inference for probabilistic programs in a Turing-complete language (Infinite GMM, Dirichlet process etc)

MCMC Inference for Probabilistic Programming

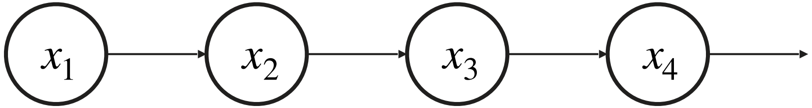
MCMC Inference for Probabilistic Programming

Simulate posterior by **updating states** in such a way that preserves the target distribution



MCMC Inference for Probabilistic Programming

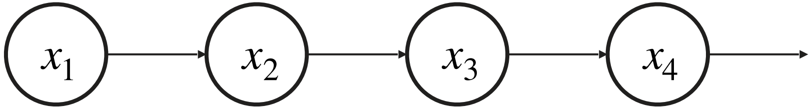
Simulate posterior by **updating states** in such a way that preserves the target distribution



State: A **list of samples drawn** in the course of a particular run of the program

MCMC Inference for Probabilistic Programming

Simulate posterior by **updating states** in such a way that preserves the target distribution

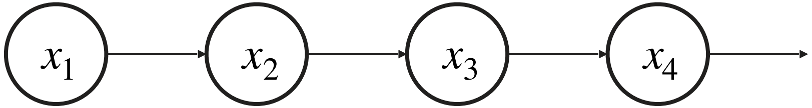


State: A **list of samples drawn** in the course of a particular run of the program

```
# parameter
K ~ Normal(3,1)
# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))
# data
for j in 1:len(data)
    data[j] ~ m
```

MCMC Inference for Probabilistic Programming

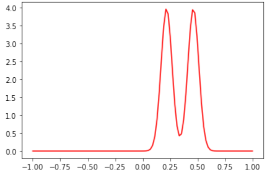
Simulate posterior by **updating states** in such a way that preserves the target distribution



State: A **list of samples drawn** in the course of a particular run of the program

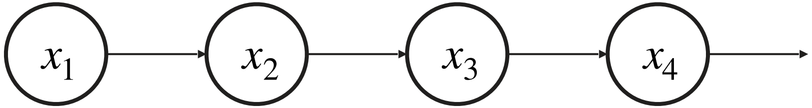
```
# parameter
K ~ Normal(3,1)
# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))
# data
for j in 1:len(data)
    data[j] ~ m
```

K	2.847
μs	0.218 0.462



MCMC Inference for Probabilistic Programming

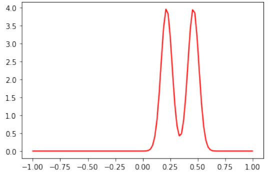
Simulate posterior by **updating states** in such a way that preserves the target distribution



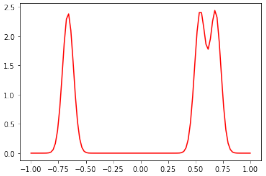
State: A **list of samples drawn** in the course of a particular run of the program

```
# parameter
K ~ Normal(3,1)
# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))
# data
for j in 1:len(data)
    data[j] ~ m
```

K	2.847
μs	0.218 0.462

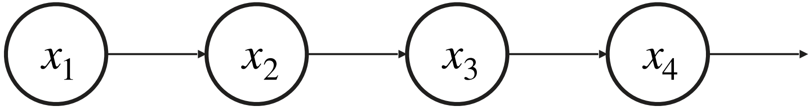


K	3.514
μs	0.542 -0.662 0.683



MCMC Inference for Probabilistic Programming

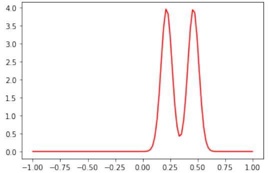
Simulate posterior by **updating states** in such a way that preserves the target distribution



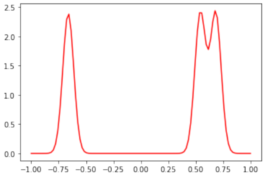
State: A **list of samples drawn** in the course of a particular run of the program

```
# parameter
K ~ Normal(3,1)
# model
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(map(lambda μ:Normal(μ,1),μs))
# data
for j in 1:len(data)
    data[j] ~ m
```

K	2.847
μs	0.218 0.462



K	3.514
μs	0.542 -0.662 0.683



Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
K ~ Normal(3,1)
for i in 1:floor(K)
      μs[i] ~ Normal(0,1)
m = MixtureModel(
      map(lambda μ:Normal(μ,1),μs))
for j in 1:len(data)
      data[j] ~ m
```

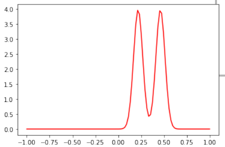
Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
K ~ Normal(3,1)
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(
    map(lambda μ:Normal(μ,1),μs))
for j in 1:len(data)
    data[j] ~ m
```

x0

K	2.847
μs	0.218 0.462

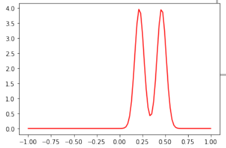


Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
K ~ Normal(3,1)
for i in 1:floor(K)
      μs[i] ~ Normal(0,1)
m = MixtureModel(
      map(lambda μ:Normal(μ,1),μs))
for j in 1:len(data)
      data[j] ~ m
```

	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662



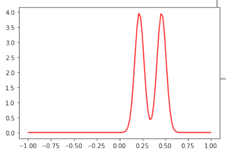
Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
K ~ Normal(3,1)
for i in 1:floor(K)
    μs[i] ~ Normal(0,1)
m = MixtureModel(
    map(lambda μ:Normal(μ,1),μs))
for j in 1:len(data)
    data[j] ~ m
```

	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662

	x	v
K	3.514	2.847
μs	0.542 -0.662	0.218 0.462



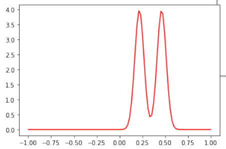
Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
K ~ Normal(3,1)
for i in 1:floor(K)
      μs[i] ~ Normal(0,1)
m = MixtureModel(
      map(lambda μ:Normal(μ,1),μs))
for j in 1:len(data)
      data[j] ~ m
```

	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662

	x	v
K	3.514	2.847
μs	0.542 -0.662	0.218 0.462



Not a state

A state is a **list of samples drawn** in the course of a particular run of the program

Metropolis-Hastings (MH)

```
Inputs: target density w with proposal
distribution Normal(0,1)(v)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
K ~ Normal(3,1)
for i in 1:floor(K)
      μs[i] ~ Normal(0,1)
m = MixtureModel(
      map(lambda μ:Normal(μ,1),μs))
for j in 1:len(data)
      data[j] ~ m
```

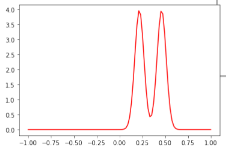
	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662

	x	v
K	3.514	2.847
μs	0.542 -0.662	0.218 0.462

Not a state

A state is a **list of samples drawn** in the course of a particular run of the program

```
w(x) = 0
Return x0
```



Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
```

```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

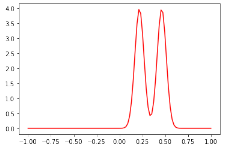
```
Return x[1:k] with prob min{1, w(x[1:k])/w(x0[1:k0])*
                               pdfnormal(x,v)/pdfnormal(x0,v0)}
Else Return x0[1:k0]
```

Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
```

```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

```
Return x[1:k] with prob min{1, w(x[1:k])/w(x0[1:k0])*
                               pdfnormal(x,v)/pdfnormal(x0,v0)}
Else Return x0[1:k0]
```



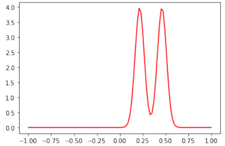
	x0
K	2.847
μs	0.218 0.462

Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
```

```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

```
Return x[1:k] with prob min{1, w(x[1:k])/w(x0[1:k0])*
                              pdfnormal(x,v)/pdfnormal(x0,v0)}
Else Return x0[1:k0]
```



	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662

	x	v
K	3.514	2.847
μs	0.542 -0.662	0.218 0.462

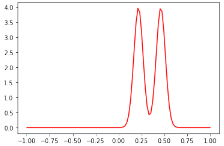
Not a state

Nonparametric Metropolis-Hastings (NP-MH)

```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
```

```
Return x[1:k] with prob min{1, w(x[1:k])/w(x0[1:k0])*
                              pdfnormal(x,v)/pdfnormal(x0,v0)}
Else Return x0[1:k0]
```



	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662
	0.105	0.683

	x	v
K	3.514	2.847
μs	0.542 -0.662	0.218 0.462

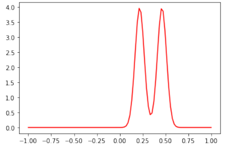
Not a state

Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
```

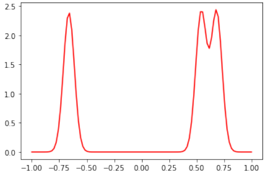
```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

```
Return x[1:k] with prob min{1, w(x[1:k])/w(x0[1:k0])*
                               pdfnormal(x,v)/pdfnormal(x0,v0)}
Else Return x0[1:k0]
```



	x0	v0
K	2.847	3.514
μs	0.218 0.462	0.542 -0.662
	0.105	0.683

	x	v
K	3.514	2.847
μs	0.542 -0.662	0.218 0.462
	0.683	0.105

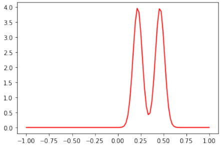


Nonparametric Metropolis-Hastings (NP-MH)

```
function NPMHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
```

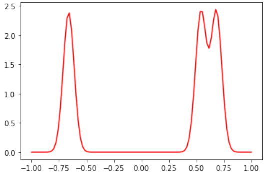
```
While w(x[1:k]) == 0 for all k
    append!(x0, Normal(0,1))
    append!(v0, Normal(0,1))
    (x,v) = (v0,x0)
```

```
Return x[1:k] with prob min{1, w(x[1:k])/w(x0[1:k0])*
                               pdfnormal(x,v)/pdfnormal(x0,v0)}
Else Return x0[1:k0]
```



	x0	v0
K	2.847	3.514
µs	0.218 0.462	0.542 -0.662
	0.105	0.683

	x	v
K	3.514	2.847
µs	0.542 -0.662	0.218 0.462
	0.683	0.105



`Return x with some prob > 0`

Does such extension work for other MCMC inferences?

Does such extension work for other MCMC inferences?

Involutive MCMC (Neklyudov et al. ICML 2020) provides a unified view of many known MCMC algorithms

Does such extension work for other MCMC inferences?

Involutive MCMC (Neklyudov et al. ICML 2020) provides a unified view of many known MCMC algorithms

```
Inputs: target density w
        auxiliary kernel k
        involution f
function iMCMCstep(x0)
    v0 ~ k(x0,-)
    (x,v) = f(x0,v0)
    Return x with probability
        min{1,(w(x)*k(x,v))/
              (w(x0)*k(x0,v0))*
              absdetjac(f(x,v))}
    Else Return x0
```

Does such extension work for other MCMC inferences?

Involutive MCMC (Neklyudov et al. ICML 2020) provides a unified view of many known MCMC algorithms

```
Inputs: target density w
        auxiliary kernel k
        involution f
function iMCMCstep(x0)
    v0 ~ k(x0,-)
    (x,v) = f(x0,v0)
    Return x with probability
        min{1,(w(x)*k(x,v))/
              (w(x0)*k(x0,v0))*
              absdetjac(f(x,v))}
    Else Return x0
```

Name & Citation	Appendix
Metropolis-Hastings (Hastings, 1970)	B.1
Mixture Proposal (Habib & Barber, 2018)	B.2
Multiple-Try Metropolis (Liu et al., 2000)	B.3
Sample-Adaptive MCMC (Zhu, 2019)	B.4
Reversible-Jump MCMC (Green, 1995)	B.5
Hybrid Monte Carlo (Duane et al., 1987)	B.6
RMHMC (Girolami & Calderhead, 2011)	B.7
NeuTra (Hoffman et al., 2019)	B.8
A-NICE-MC (Song et al., 2017)	B.9
L2HMC (Levy et al., 2017)	B.10
Persistent HMC (Horowitz, 1991)	B.11
Gibbs (Geman & Geman, 1984)	B.12
Look Ahead (Sohl-Dickstein et al., 2014)	B.13
NRJ (Gagnon & Doucet, 2019)	B.14
Lifted MH (Turitsyn et al., 2011)	B.15

Table 1: List of algorithms that we describe by the Involutive MCMC framework. See their descriptions and formulations in terms of iMCMC in corresponding appendices.

Does such extension work for other MCMC inferences?

Involutive MCMC (Neklyudov et al. ICML 2020) provides a unified view of many known MCMC algorithms

```
Inputs: target density w
        auxiliary kernel k
        involution f
function iMCMCstep(x0)
    v0 ~ k(x0,-)
    (x,v) = f(x0,v0)
    Return x with probability
        min{1,(w(x)*k(x,v))/
              (w(x0)*k(x0,v0))*
              absdetjac(f(x,v))}
    Else Return x0
```

Name & Citation	Appendix
Metropolis-Hastings (Hastings, 1970)	B.1
Mixture Proposal (Habib & Barber, 2018)	B.2

```
Inputs: target density w
        k(x,v) := Normal(0,1)(v)
        f(x,v) := (v,x)
function MHstep(x0)
    v0 ~ Normal(0,1)
    (x,v) = (v0,x0)
    Return x with probability
      min{1,w(x)/w(x0)*
      pdfnormal(x,v)/pdfnormal(x0,v0)}
    Else Return x0
```

Table 1: List of algorithms that we describe by the Involutive MCMC framework. See their descriptions and formulations in terms of iMCMC in corresponding appendices.

Nonparametric Involutive MCMC (NP-iMCMC)

Nonparametric Involutive MCMC (NP-iMCMC)

```
target density w
auxiliary kernel K[n] for each n
involution f[n] for each n
function NPiMCMCstep(x0)
    k0 = len(x0)
    v0 ~ K[k0](x0,-)
    (x,v) = f[k0](x0,v0)
    while w(x[1:k]) == 0 for all k
        append!(x0, Normal(0,1));
        append!(v0, Normal(0,1));
        (x,v) = f[len(x0)](x0,v0)
    Return x[1:k] with prob min{1,P}
    Else Return x0[1:k0]
```

Nonparametric Involutive MCMC (NP-iMCMC)

```
target density w
auxiliary kernel K[n] for each n
involution f[n] for each n
function NPiMCMCstep(x0)
    k0 = len(x0)
    v0 ~ K[k0](x0,-)
    (x,v) = f[k0](x0,v0)
    while w(x[1:k]) == 0 for all k
        append!(x0, Normal(0,1));
        append!(v0, Normal(0,1));
        (x,v) = f[len(x0)](x0,v0)
    Return x[1:k] with prob min{1,P}
    Else Return x0[1:k0]
```

$$P := \frac{w(x^{1\dots k})\cdot \mathrm{pdf}K^{(k)}(x^{1\dots k},v^{1\dots k})\cdot\varphi(x,v)}{w(x^{1\dots k_0})\cdot \mathrm{pdf}K^{(k_0)}(x^{1\dots k_0},v^{1\dots k_0})\cdot\varphi(x_0,v_0)}\left|\det \nabla\Phi^{(n)}(x_0,v_0)\right|$$

Nonparametric Involutive MCMC (NP-iMCMC)

```
target density w
auxiliary kernel K[n] for each n
involution f[n] for each n
function NPiMCMCstep(x0)
    k0 = len(x0)
    v0 ~ K[k0](x0,-)
    (x,v) = f[k0](x0,v0)
    while w(x[1:k]) == 0 for all k
        append!(x0, Normal(0,1));
        append!(v0, Normal(0,1));
        (x,v) = f[len(x0)](x0,v0)
    Return x[1:k] with prob min{1,P}
    Else Return x0[1:k0]
```

- **Target density** w is (A1) integrable and (A2) almost surely terminating
- **Auxiliary kernel** $K^{(n)}$ for each dim. n
- **Involution** $\Phi^{(n)}$ for each dim. n satisfying the (A3) projection commutation property:

For all (x, v) where |x| = m, if $w(x^{1..n}) > 0$ for some n, then for all k = n, …, m, $\text{take}_k(\Phi^{(m)}(x, v)) = \Phi^{(k)}(\text{take}_k(x, v))$

$$P := \frac{w(x^{1\dots k})\cdot \text{pdf}K^{(k)}(x^{1\dots k},v^{1\dots k})\cdot\varphi(x,v)}{w(x^{1\dots k_0})\cdot \text{pdf}K^{(k_0)}(x^{1\dots k_0},v^{1\dots k_0})\cdot\varphi(x_0,v_0)}\left|\det \nabla\Phi^{(n)}(x_0,v_0)\right|$$

Generalisations of NP-iMCMC

Generalisations of NP-iMCMC

Hybrid NP-iMCMC

for discrete and continuous samplers

```
Sum = 0;
while Bern(0.5): count += Normal(0,1)
```

Generalisations of NP-iMCMC

Hybrid NP-iMCMC

for discrete and continuous samplers

```
Sum = 0;
while Bern(0.5): count += Normal(0,1)
```

Multiple Step NP-iMCMC

for complex involutions

```
function leapfrog(q0,p0)
     q = q0; p = p0
     for i in 1:L
          p = p - ep/2 * grad(U)(q)
          q = q + ep * grad(K)(p)
          p = p - ep/2 * grad(U)(q)
     return (q,-p)
```

Generalisations of NP-iMCMC

Hybrid NP-iMCMC

for discrete and continuous samplers

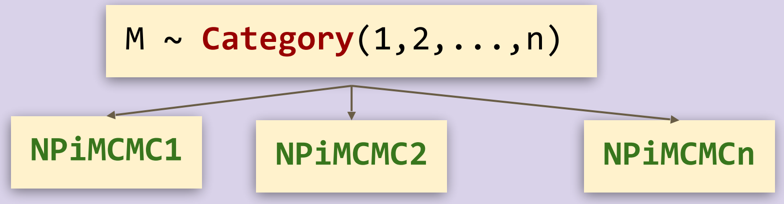
```
Sum = 0;
while Bern(0.5): count += Normal(0,1)
```

Multiple Step NP-iMCMC

for complex involutions

```
function leapfrog(q0,p0)
     q = q0; p = p0
     for i in 1:L
          p = p - ep/2 * grad(U)(q)
          q = q + ep * grad(K)(p)
          p = p - ep/2 * grad(U)(q)
     return (q,-p)
```

State-dependent Mixtures of NP-iMCMC



Generalisations of NP-iMCMC

Hybrid NP-iMCMC

for discrete and continuous samplers

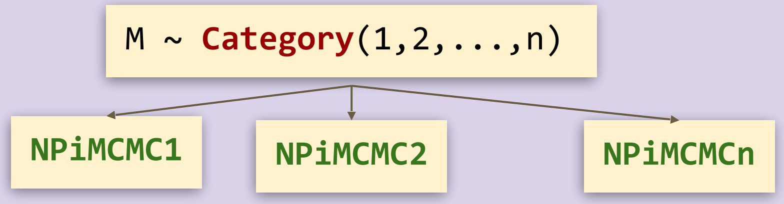
```
Sum = 0;
while Bern(0.5): count += Normal(0,1)
```

Multiple Step NP-iMCMC

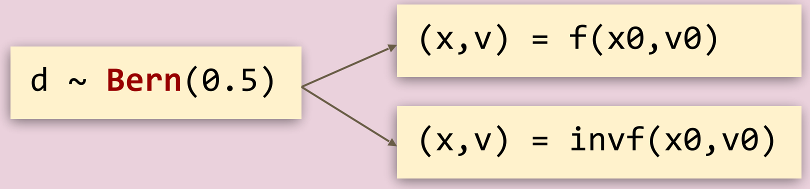
for complex involutions

```
function leapfrog(q0,p0)
     q = q0; p = p0
     for i in 1:L
          p = p - ep/2 * grad(U)(q)
          q = q + ep * grad(K)(p)
          p = p - ep/2 * grad(U)(q)
     return (q,-p)
```

State-dependent Mixtures of NP-iMCMC



Direction NP-iMCMC



Generalisations of NP-iMCMC

Hybrid NP-iMCMC

for discrete and continuous samplers

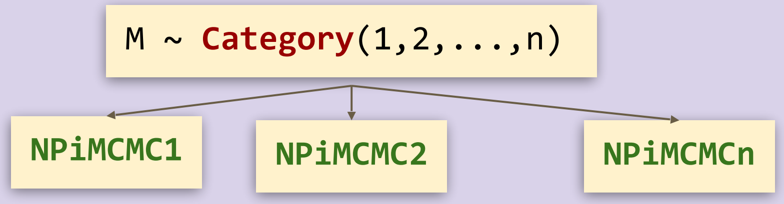
```
Sum = 0;
while Bern(0.5): count += Normal(0,1)
```

Multiple Step NP-iMCMC

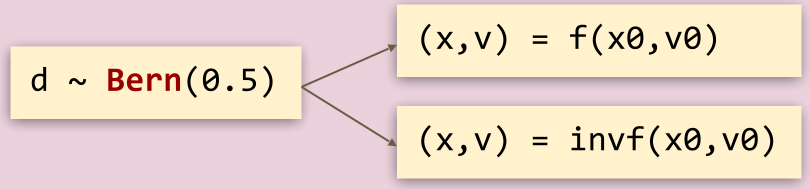
for complex involutions

```
function leapfrog(q0,p0)
     q = q0; p = p0
     for i in 1:L
          p = p - ep/2 * grad(U)(q)
          q = q + ep * grad(K)(p)
          p = p - ep/2 * grad(U)(q)
     return (q,-p)
```

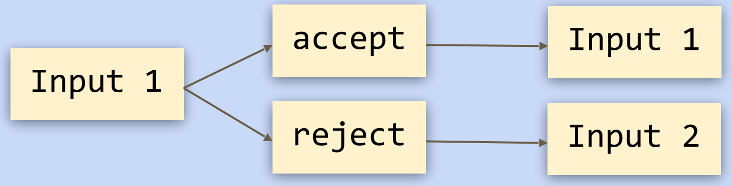
State-dependent Mixtures of NP-iMCMC



Direction NP-iMCMC



Persistent NP-iMCMC



NP-HMC on Infinite GMM

NP-HMC on Infinite GMM

HMC ⟶ NP-HMC (Mak et al 2021)

NP-HMC on Infinite GMM

HMC ⟶ NP-HMC (Mak et al 2021)

Generalised HMC (Horowitz 1991)

Look Ahead HMC
(Sohl-Dickstein et al., 2014;
Campos & Sanz-Serna, 2015)

NP-HMC on Infinite GMM

HMC → NP-HMC (Mak et al 2021)

Generalised HMC (Horowitz 1991) → NP-HMC with persistence

Look Ahead HMC (Sohl-Dickstein et al., 2014; Campos & Sanz-Serna, 2015) → NP-Look Ahead HMC

NP-HMC on Infinite GMM

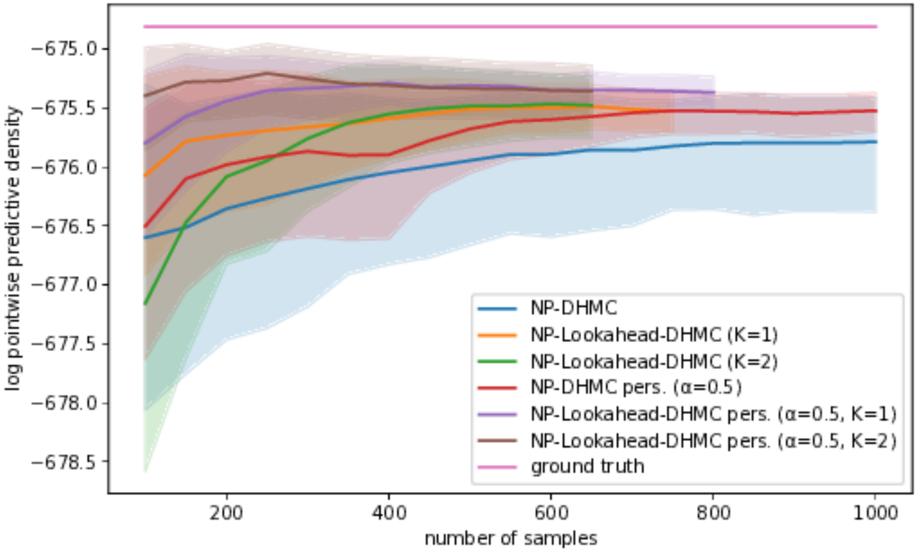
HMC → NP-HMC (Mak et al 2021)

Generalised HMC (Horowitz 1991) → NP-HMC with persistence

Look Ahead HMC (Sohl-Dickstein et al., 2014; Campos & Sanz-Serna, 2015) → NP-Look Ahead HMC

Infinite GMM

- 200 training data points generated from a mixture of 9 components
- Compute LPPD on 50 data points



Summary

- **NP-iMCMC** transform many existing MCMC inference to **work on probabilistic programs**
- NP-iMCMC is **theoretically correct**
- Existing strengths of iMCMC algorithms **carry over to their nonparametric extensions**