Paradoxes of Probabilistic Programming

and how to condition on events of measure zero with infinitesimal probabilities (to appear at POPL'21)

Jules Jacobs

TU Delft $\mapsto$ Radboud University Nijmegen
julesjacobs@gmail.com

November 25, 2020

Probabilistic programming

- Domain specific language for statistical and machine learning models
- Normal programming language extended with rand, observe, and run

Probabilistic programming

Example:

- Men's height is distributed according to $\mathit{Normal}(1.8, 0.5)$ meters
- Women's height is distributed according to $\mathit{Normal}(1.7, 0.5)$ meters
- A scientist randomly samples a man and a woman and compares their height
- The scientist tells us that the heights are equal

Question: What's the expected value of the height in this situation?

Probabilistic programming

Example:

- Men's height is distributed according to $\mathit{Normal}(1.8, 0.5)$ meters
- Women's height is distributed according to $\mathit{Normal}(1.7, 0.5)$ meters
- A scientist randomly samples a man and a woman and compares their height
- The scientist tells us that the heights are equal

Question: What's the expected value of the height in this situation?

```
function meters(){
  h = rand(Normal(1.7, 0.5))
  observe(Normal(1.8, 0.5), h)
  return h
}
samples = run(meters, 1000)
estimate = average(samples)
```

Answer: ≈ 1.75

Probabilistic programming

Example:

- Men's height is distributed according to *Normal*(1.8, 0.5) meters
- Women's height is distributed according to *Normal*(1.7, 0.5) meters
- A scientist randomly samples a man and a woman and compares their height
- The scientist tells us that the heights are equal

Question: What's the expected value of the height in this situation?

```
function meters(){
  h = rand(Normal(1.7, 0.5))
  observe(Normal(1.8, 0.5), h)
  return h
}
samples = run(meters, 1000)
estimate = average(samples)
```

Answer: ≈ 1.75

```
function centimeters(){
  h = rand(Normal(170, 50))
  observe(Normal(180, 50), h)
  return h
}
samples = run(meters, 1000)
estimate = average(samples)
```

Answer: ≈ 175

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling in Anglican

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling in Anglican
- The issue is fundamental and not limited to Anglican

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling in Anglican
- The issue is fundamental and not limited to Anglican
- Even happens in formal operational semantics (e.g. Commutative or Quasi-Borel)

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling in Anglican
- The issue is fundamental and not limited to Anglican
- Even happens in formal operational semantics (e.g. Commutative or Quasi-Borel)
- Unclear what the answer *should* be, or whether this program should be disallowed

Paradox 2

Objection: you shouldn't do observe a variable number of times based on coin flip

Suppose the scientist is drunk, and measures the weight half of the time...

Paradox 2

Objection: you shouldn't do observe a variable number of times based on coin flip

Suppose the scientist is drunk, and measures the weight half of the time...

```
h = rand(Normal(1.7, 0.5))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 1.75

Paradox 2

Objection: you shouldn't do observe a variable number of times based on coin flip

Suppose the scientist is drunk, and measures the weight half of the time...

```
h = rand(Normal(1.7, 0.5))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 1.75

```
h = rand(Normal(170, 50))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 170

- The same number of observes regardless of the outcome of the coin flip
- The output still depends on whether we use meters or centimeters

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

- Whether we use linear scale or log scale shouldn't matter, just like meters or centimeters shouldn't matter

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

- Whether we use linear scale or log scale shouldn't matter, just like meters or centimeters shouldn't matter
- No conditionals at all, but the output still depends on the scale we use

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

- Whether we use linear scale or log scale shouldn't matter, just like meters or centimeters shouldn't matter
- No conditionals at all, but the output still depends on the scale we use
- What do probabilistic programs really mean?

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

- Whether we use linear scale or log scale shouldn't matter, just like meters or centimeters shouldn't matter
- No conditionals at all, but the output still depends on the scale we use
- What do probabilistic programs really mean?
- What does probililistic conditioning really mean?

Paradox 3

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

- Whether we use linear scale or log scale shouldn't matter, just like meters or centimeters shouldn't matter
- No conditionals at all, but the output still depends on the scale we use
- What do probabilistic programs really mean?
- What does probililistic conditioning really mean?
- Related to the Borel-Komolgorov paradox

Overview

Problem:

- Probabilistic programs are not invariant under parameter transformations
- It's not clear what `observe` really means

Overview

Problem:

- Probabilistic programs are not invariant under parameter transformations
- It's not clear what `observe` really means

Key ideas:

1. Figure out what `observe` should do, by analogy with the discrete case
2. Change the language: `observe` conditions on *intervals* instead of points
3. Take interval width to be infinitesimally small to condition on measure zero events

Overview

Problem:

- Probabilistic programs are not invariant under parameter transformations
- It's not clear what `observe` really means

Key ideas:

1. Figure out what `observe` should do, by analogy with the discrete case
2. Change the language: `observe` conditions on *intervals* instead of points
3. Take interval width to be infinitesimally small to condition on measure zero events

Result:

- New language is invariant under arbitrary parameter transformations
- Programs have clear probabilistic meaning via *rejection sampling*
- Implemented as a DSL in Julia

Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$."

What's the probability distribution of x now?

Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$."

What's the probability distribution of x now? Use **rejection sampling**:

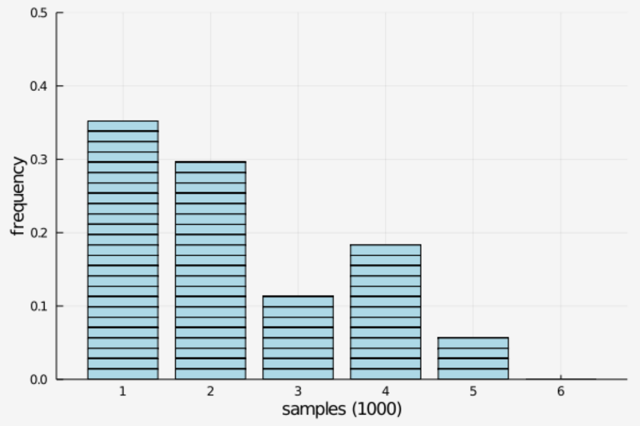
```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  if(z == x + y){
    samples.append(x)
  }
}
```

Probabilistic programming 101: manual rejection sampling

Someone: “I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$.”

What's the probability distribution of x now? Use **rejection sampling**:

```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUnif
  y = rand(DiscreteUnif
  z = rand(DiscreteUnif
  if(z == x + y){
    samples.append(x)
  }
}
```

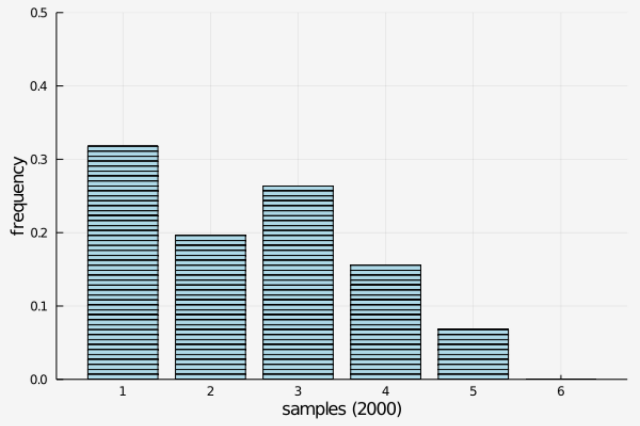


Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$."

What's the probability distribution of x now? Use **rejection sampling**:

```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUnif
  y = rand(DiscreteUnif
  z = rand(DiscreteUnif
  if(z == x + y){
    samples.append(x)
  }
}
```

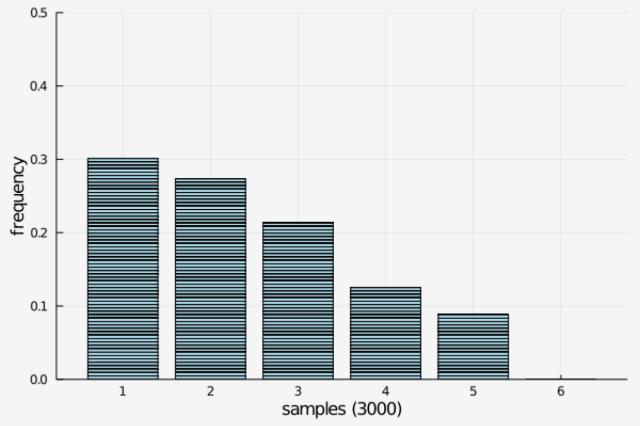


Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$."

What's the probability distribution of x now? Use **rejection sampling**:

```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUnif
  y = rand(DiscreteUnif
  z = rand(DiscreteUnif
  if(z == x + y){
    samples.append(x)
  }
}
```

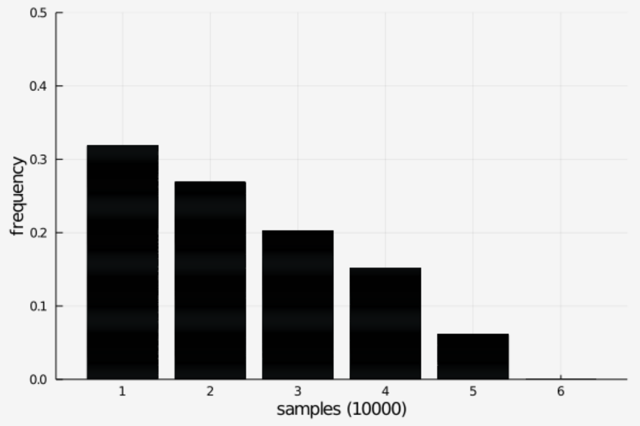


Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$."

What's the probability distribution of x now? Use **rejection sampling**:

```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUnif
  y = rand(DiscreteUnif
  z = rand(DiscreteUnif
  if(z == x + y){
    samples.append(x)
  }
}
```

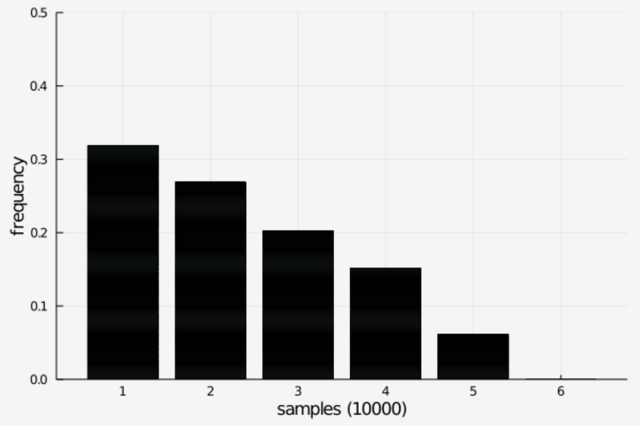


Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1,2,3,4,5,6\}$ and observed that $x + y = z$."

What's the probability distribution of x now? Use **rejection sampling**:

```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUnif
  y = rand(DiscreteUnif
  z = rand(DiscreteUnif
  if(z == x + y){
    samples.append(x)
  }
}
```



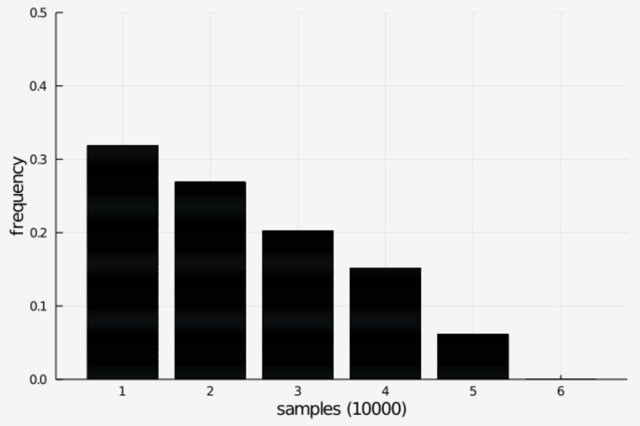
Key idea of PP: answer probabilistic inference questions by repeated simulation + filtering

Probabilistic programming 101: manual rejection sampling

Someone: "I rolled three dice $x, y, z \in \{1, 2, 3, 4, 5, 6\}$ and observed that $x + y = z$."

What's the probability distribution of x now? Use **rejection sampling**:

```
samples = []
for(i in 1..1000){
  x = rand(DiscreteUnif
  y = rand(DiscreteUnif
  z = rand(DiscreteUnif
  if(z == x + y){
    samples.append(x)
  }
}
```



Key idea of PP: answer probabilistic inference questions by repeated simulation + filtering

$\Longrightarrow$ **Probabilistic Programming Language = DSL for probabilistic simulations**

Probabilistic programming 101: DSL for rejection sampling

Probabilistic programming language:

- Normal programming language + `rand(D)`
- `observe(b)` – filtering/conditioning
- `run(func, k)` – run simulation
  `func()` k times, return array of samples

Probabilistic programming 101: DSL for rejection sampling

Probabilistic programming language:

- Normal programming language + `rand(D)`
- `observe(b)` – filtering/conditioning
- `run(func, k)` – run simulation `func()` k times, return array of samples

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

Probabilistic programming 101: DSL for rejection sampling

Probabilistic programming language:

- Normal programming language + `rand(D)`
- `observe(b)` – filtering/conditioning
- `run(func, k)` – run simulation `func()` k times, return array of samples

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

DSL implementation:

```
weight = 1
function observe(b){
  if(!b) weight = 0
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    result = func()
    if(weight == 1){
      samples.append(result)
    }
  }
  return samples
}
```

Probabilistic programming 101: DSL for rejection sampling

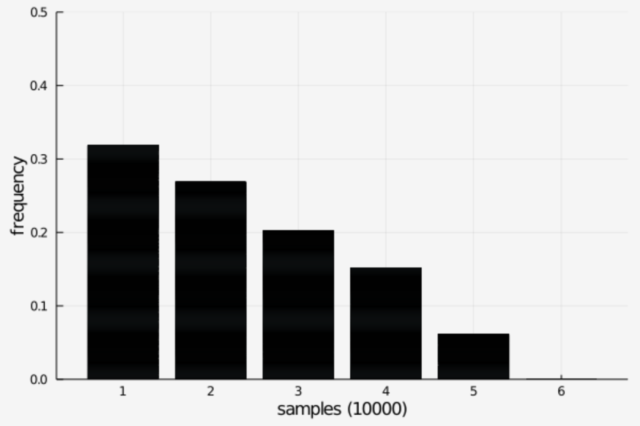
Probabilistic programming language:

- Normal programming language + rand(D)
- observe(b) – filtering/conditio
- run(func, k) – run simulation func() *k* times, return array of

```
function threeDice(){
  x = rand(DiscreteUnifor
  y = rand(DiscreteUnifor
  z = rand(DiscreteUnifor
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

DSL implementation:

```
weight = 1
)
}
```

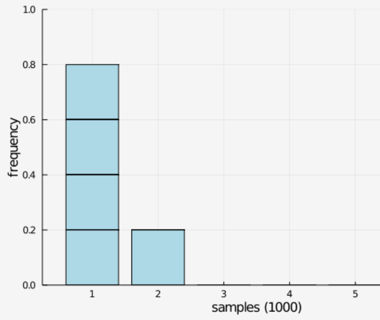


Probabilistic programming 101

```
function multiDice(){
 x = rand(DiscreteUniform(1,6))
 for(i in 1:x){
  y = rand(DiscreteUniform(1,6))
  observe(rand(DiscreteUniform(1,y)) == 3)
 }
 observe(rand(DiscreteUniform(1,6))+x == 5)
 return x
}
samples = run(multiDice, 1000)
```

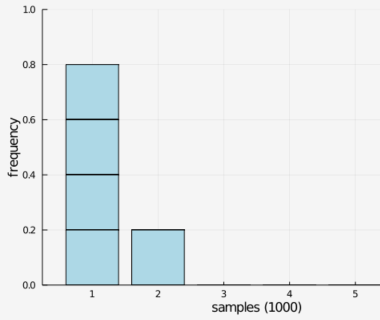
Probabilistic programming 101

```
function multiDice(){
 x = rand(DiscreteUniform(1,6))
 for(i in 1:x){
  y = rand(DiscreteUniform(1,6))
  observe(rand(DiscreteUniform(1,y)) == 3)
 }
 observe(rand(DiscreteUniform(1,6))+x == 5)
 return x
}
samples = run(multiDice, 1000)
```



Probabilistic programming 101

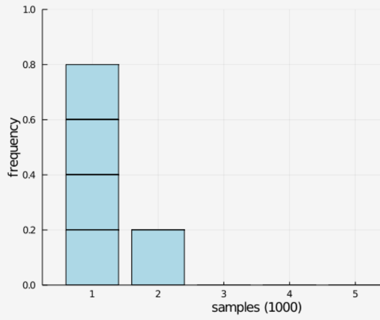
```
function multiDice(){
 x = rand(DiscreteUniform(1,6))
 for(i in 1:x){
  y = rand(DiscreteUniform(1,6))
  observe(rand(DiscreteUniform(1,y)) == 3)
 }
 observe(rand(DiscreteUniform(1,6))+x == 5)
 return x
}
samples = run(multiDice, 1000)
```



Problem: most samples get rejected $\implies$ convergence is slow

Probabilistic programming 101

```
function multiDice(){
 x = rand(DiscreteUniform(1,6))
 for(i in 1:x){
  y = rand(DiscreteUniform(1,6))
  observe(rand(DiscreteUniform(1,y)) == 3)
 }
 observe(rand(DiscreteUniform(1,6))+x == 5)
 return x
}
samples = run(multiDice, 1000)
```



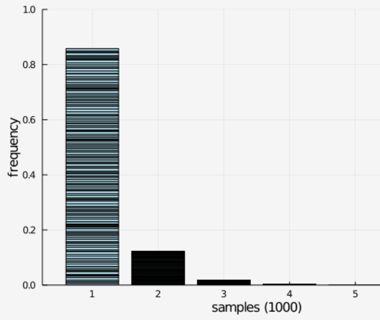
Problem: most samples get rejected $\Longrightarrow$ convergence is slow

Standard solution: importance sampling

- change `observe(rand(D) == x)` $\mapsto$ `observe(D,x)`
- `function observe(D,x){ weight *= probability(D,x) }`
- weights are now numbers between 0..1 instead of only 0,1
- `run` returns an array of weighted samples

Probabilistic programming 101: importance sampling

```
function multiDice(){
 x = rand(DiscreteUniform(1,6))
 for(i in 1:x){
  y = rand(DiscreteUniform(1,6))
  observe(DiscreteUniform(1,y), 3)
 }
 observe(DiscreteUniform(1,6), 5-x)
 return x
}
samples = run(multiDice, 1000)
```



Problem: most samples get rejected $\implies$ convergence is slow

Solution:

- change `observe(rand(D) == x)` $\mapsto$ `observe(D,x)`
- `function observe(D,x){ weight *= probability(D,x) }`
- weights are now numbers between 0..1 instead of only 0,1
- `run` returns an array of weighted samples

Probabilistic programming 101: continuous distributions

Continuous distributions are problematic because `probability(D,x) = 0`.

When doing observe(D, x) for continuous distributions D,

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Probabilistic programming 101: continuous distributions

Continuous distributions are problematic because `probability(D,x) = 0`.

When doing observe(D, x) for continuous distributions D,

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Standard solution: use the probability density function `pdf(D,x)` instead.

```
function observe(D,x){ weight *= probability(D,x) }
```

$$\mapsto$$

```
function observe(D,x){ weight *= pdf(D,x) }
```

Probabilistic programming 101: continuous distributions

Continuous distributions are problematic because `probability(D,x) = 0`.

When doing observe(D, x) for continuous distributions D,

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Standard solution: use the probability density function `pdf(D,x)` instead.

```
function observe(D,x){ weight *= probability(D,x) }
```

$$\mapsto$$

```
function observe(D,x){ weight *= pdf(D,x) }
```

Intuition: pdf$(D, x) \propto$ the probability that rand(D) is close to x. **Formally:**

$$\mathrm{cdf}(D, x) = \mathbb{P}[\mathrm{rand}(D) < x] \qquad \mathrm{pdf}(D, x) = \frac{d}{dx}\mathrm{cdf}(D, x)$$

End of probabilistic programming 101. □

Probabilistic programming 101: continuous distributions

Continuous distributions are problematic because `probability(D,x) = 0`.

When doing observe(D, x) for continuous distributions D,

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Standard solution: use the probability density function `pdf(D,x)` instead.

```
function observe(D,x){ weight *= probability(D,x) }
```

$$\mapsto$$

```
function observe(D,x){ weight *= pdf(D,x) }
```

Intuition: pdf$(D, x) \propto$ the probability that rand(D) is close to x. **Formally:**

$$\text{cdf}(D, x) = \mathbb{P}[\text{rand}(D) < x] \qquad \text{pdf}(D, x) = \frac{d}{dx}\text{cdf}(D, x)$$

End of probabilistic programming 101. $\square$

Using the pdf instead of the probability is the source of the strange behaviour!

What went wrong: conditionals

Recall the drunk scientist:

```
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
```

```
function observe(D,x){
  weight *= pdf(D,x)
}
```

What went wrong: conditionals

Recall the drunk scientist:

```
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
```

```
function observe(D,x){
  weight *= pdf(D,x)
}
```

- An observe(D, x) call multiplies the weight by pdf(D, x)
- The pdf is not unitless! $\text{pdf}(\text{Normal}(\mu, \sigma), x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$
- The weight has units m^{-1} in some trials and kg^{-1} in other trials
- Results in unit errors when computing the weighted average:

$$\mathbb{E}[bmi] \approx \frac{\sum_{k=1}^{N}(weight_k) \cdot (bmi_k)}{\sum_{k=1}^{N}(weight_k)}$$

- **The sum adds $m^{-1} + kg^{-1}$!**

What went wrong: nonlinear parameter transformations

Recall the log scale scientist:

`observe(Normal(1.8, 0.5), h)` vs `observe(LogNormal(1.8, 0.5), H)`

Conditioning on events of measure zero is ambiguous!

What went wrong: nonlinear parameter transformations

Recall the log scale scientist:

`observe(Normal(1.8, 0.5), h)` vs `observe(LogNormal(1.8, 0.5), H)`

Conditioning on events of measure zero is ambiguous!

$$A_\epsilon = \{(x, y) \in \mathbb{R}^2 : |x - y| \leq \epsilon\}$$
$$B_\epsilon = \{(x, y) \in \mathbb{R}^2 : |\exp(x) - \exp(y)| \leq \epsilon\}$$

What went wrong: nonlinear parameter transformations

Recall the log scale scientist:

`observe(Normal(1.8, 0.5), h)` vs `observe(LogNormal(1.8, 0.5), H)`

Conditioning on events of measure zero is ambiguous!

$$A_\epsilon = \{(x, y) \in \mathbb{R}^2 : |x - y| \leq \epsilon\}$$
$$B_\epsilon = \{(x, y) \in \mathbb{R}^2 : |\exp(x) - \exp(y)| \leq \epsilon\}$$

"Although the sequences A_ϵ and B_ϵ tend to the same limit "$x = y$", the conditional densities $\mathbb{P}(x|A_\epsilon)$ and $\mathbb{P}(x|B_\epsilon)$ tend to different limits. As we see from this, merely to specify "$x = y$" without any qualifications is ambiguous. Whenever we have a probability density on one space and we wish to generate from it one on a subspace of measure zero, the only safe procedure is to pass to an explicitly defined limit by a process like A_ϵ and B_ϵ. In general, the final result will and must depend on which limiting operation was specified. This is extremely counter-intuitive at first hearing; yet it becomes obvious when the reason for it is understood."
– E.T. Jaynes (paraphrased)

Solution: don't condition on measure zero events

Problem: conditioning on events of measure zero is ambiguous.
Solution: condition on intervals.

```
observe(D, Interval(x,w))
```

Meaning: rand(D) is in an interval of width w around x.

Solution: don't condition on measure zero events

Problem: conditioning on events of measure zero is ambiguous.
Solution: condition on intervals.

```
observe(D, Interval(x,w))
```

Meaning: rand(D) is in an interval of width w around x.

Rejection sampling:

```
function observe(D,I){
  if(abs(rand(D) - I.midpoint) > I.width/2){ weight = 0 }
}
```

Solution: don't condition on measure zero events

Problem: conditioning on events of measure zero is ambiguous.
Solution: condition on intervals.

```
observe(D, Interval(x,w))
```

Meaning: rand(D) is in an interval of width w around x.

Rejection sampling:

```
function observe(D,I){
  if(abs(rand(D) - I.midpoint) > I.width/2){ weight = 0 }
}
```

Importance sampling:

```
function observe(D,I){ weight *= probability(D,I) }
```

For intervals, `probability(D,I)` is nonzero.

Example of conditioning on intervals

Example:

```
function centimeters(){
  h = rand(Normal(170, 50))
  if(flip(0.5)){
    observe(Normal(180, 10), Interval(h, 10))
  }
}
function meters(){
  h = rand(Normal(1.7, 0.5))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, 0.1))
  }
}
```

Example of conditioning on intervals

Example:

```
function centimeters(){
  h = rand(Normal(170, 50))
  if(flip(0.5)){
    observe(Normal(180, 10), Interval(h, 10))
  }
}
function meters(){
  h = rand(Normal(1.7, 0.5))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, 0.1))
  }
}
```

Same output & no unit errors, even though observe is conditionally executed!

Example of conditioning on intervals

Example:

```
function centimeters(){
  h = rand(Normal(170, 50))
  if(flip(0.5)){
    observe(Normal(180, 10), Interval(h, 10))
  }
}
function meters(){
  h = rand(Normal(1.7, 0.5))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, 0.1))
  }
}
```

Same output & no unit errors, even though observe is conditionally executed!
Rejection sampling and importance sampling converge to the same answer!

Take the limit

We still want to condition on measure zero events

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit $\mathit{width} \to 0$

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit *width* $\to 0$

```
function drunk(width){
  h = rand(Normal(1.7, 0.5))
  w = rand(Normal(60, 10))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, A*width))
  }else{
    observe(Normal(70, 10), Interval(w, B*width))
  }
}
```

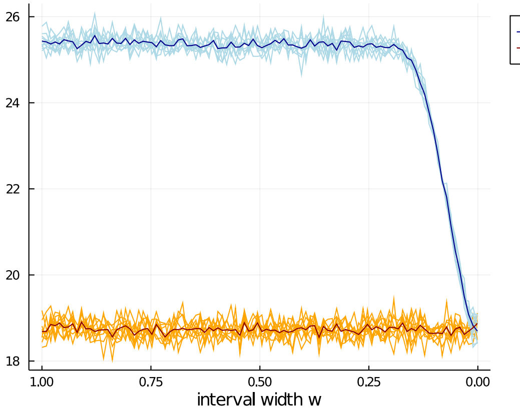
Since *width* is unitless, we must introduce constants A and B with units m and kg.
The relative size matters even as *width* $\to 0$!

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit $width \to 0$

```
function drunk(width){
  h = rand(Normal(1.7, 0.5))
  w = rand(Normal(60, 10))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Int
  }else{
    observe(Normal(70, 10), Inter
  }
}
```



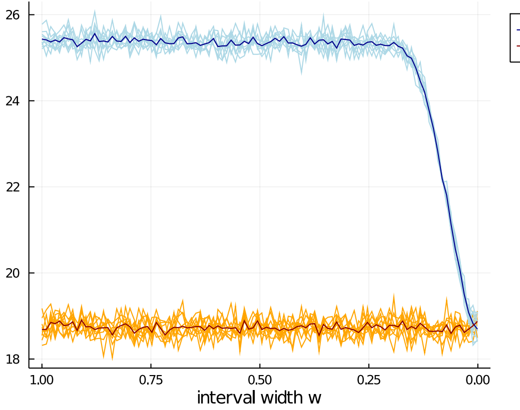
Since ***width*** **is unitless, we must introduce constants** A **and** B **with units** m **and** kg**.**
The relative size matters even as $width \to 0$**!**

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit *width* $\to 0$

```
function drunk(width){
  h = rand(Normal(1.7, 0.5))
  w = rand(Normal(60, 10))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Int
  }else{
    observe(Normal(70, 10), Inter
  }
}
```



Since ***width*** **is unitless, we must introduce constants** A **and** B **with units** m **and** kg**.**
The relative size matters even as ***width*** $\to 0$**!**

Can we compute the limit $w \to 0$ **directly?**

Infinitesimal numbers

Definition

An infinitesimal number is a pair $(r, n) \in \mathbb{R} \times \mathbb{Z}$, which we write as $r\epsilon^n$.

Infinitesimal numbers

Definition

An infinitesimal number is a pair $(r, n) \in \mathbb{R} \times \mathbb{Z}$, which we write as $r\epsilon^n$.

Arithmetic with infinitesimals:

$$r\epsilon^n \pm s\epsilon^k = \begin{cases} (r \pm s)\epsilon^n & \text{if } n = k \\ r\epsilon^n & \text{if } n < k \\ \pm s\epsilon^k & \text{if } n > k \end{cases}$$

$$(r\epsilon^n) \cdot (s\epsilon^k) = (r \cdot s)\epsilon^{n+k}$$

$$(r\epsilon^n)/(s\epsilon^k) = \begin{cases} (r/s)\epsilon^{n-k} & \text{if } s \neq 0 \\ \text{undefined} & \text{if } s = 0 \end{cases}$$

Infinitesimal numbers

The probability that rand(D) lies in the interval $[x - r\epsilon^n, x + r\epsilon^n]$:

$$\text{probability}(D, \text{Interval}(x, r\epsilon^n)) = \begin{cases} \text{cdf}(D, x + \frac{1}{2}r) - \text{cdf}(D, x - \frac{1}{2}r) & \text{if } n = 0 \\ \text{pdf}(D, x) \cdot r\epsilon^n & \text{if } n > 0 \end{cases}$$

Infinitesimal numbers

The probability that rand(D) lies in the interval $[x - r\epsilon^n, x + r\epsilon^n]$:

$$\text{probability}(D, \text{Interval}(x, r\epsilon^n)) = \begin{cases} \text{cdf}(D, x + \frac{1}{2}r) - \text{cdf}(D, x - \frac{1}{2}r) & \text{if } n = 0 \\ \text{pdf}(D, x) \cdot r\epsilon^n & \text{if } n > 0 \end{cases}$$

Infinitesimals unify cdf and pdf!

Correctness properties

Consistency with existing probabilistic programming languages:

`observe(D,Interval(x,eps))` gives the same result as `observe(D,x)` outside conditionals

Correctness properties

Consistency with existing probabilistic programming languages:
`observe(D,Interval(x,eps))` gives the same result as `observe(D,x)` outside conditionals

Consistency with non-zero width intervals:
`observe(D,Interval(x,eps))` gives the same result as `observe(D,Interval(x,width))` and then taking the limit width $\to 0$

Theorem

If $f(x)$ is given by a "probability expression" and $f(\epsilon) = r\epsilon^n$, then $\lim_{x\to 0} \frac{f(x)}{x^n} = r$.

Correctness properties

Consistency with existing probabilistic programming languages:
`observe(D,Interval(x,eps))` gives the same result as `observe(D,x)` outside conditionals

Consistency with non-zero width intervals:
`observe(D,Interval(x,eps))` gives the same result as `observe(D,Interval(x,width))` and then taking the limit width $\to 0$

Theorem

If $f(x)$ is given by a "probability expression" and $f(\epsilon) = r\epsilon^n$, then $\lim_{x\to 0} \frac{f(x)}{x^n} = r$.

Definition

We say that $f(x)$ is a "probability expression" in the variable x if $f(x)$ is defined using the operations $+, -, \cdot, /$, constants, and probability$(D, \text{Interval}(s, rx))$ where $r, s \in \mathbb{R}$ are constants, and D is a probability distribution with differentiable cdf.

Importance sampling computes a probability expression.

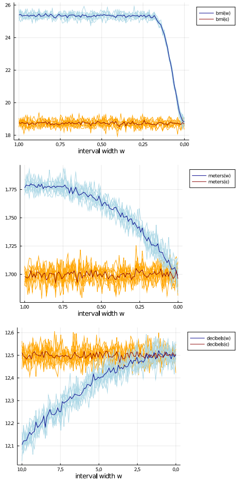
Infinitesimal numbers

Example programs:

```
function bmi(width){
  h = rand(Normal(1.70, 0.2))
  w = rand(Normal(70, 30))
  if(flip(0.5)){
        observe(Normal(2.0,0.1), Interval(h,10*width))
  }else{
        observe(Normal(90,5), Interval(w,width))
  }
  return w / h^2
}

function meters(width){
  h = rand(Normal(1.7,0.5))
  if(flip(0.5)){
        observe(Normal(2.0,0.1), Interval(h,width))
  }
  return h
}

function decibels(width){
  x = rand(Normal(10,5))
  observe(Normal(15,5),Interval(x,width))
  return x
}
```



Theorem works: we can condition on events of measure zero without paradoxes

Parameter transformations

The factor in front of ϵ allows us to do parameter transformations correctly:

$$\text{A function } f \text{ maps Interval}(x, \epsilon) \text{ to Interval}(f(x), f'(x)\epsilon).$$

Parameter transformations

The factor in front of ϵ allows us to do parameter transformations correctly:

$$\text{A function } f \text{ maps Interval}(x, \epsilon) \text{ to Interval}(f(x), f'(x)\epsilon).$$

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),
        Interval(h,eps))
return h
```

Answer: 1.75

Parameter transformations

The factor in front of ϵ allows us to do parameter transformations correctly:

$$\text{A function } f \text{ maps Interval}(x, \epsilon) \text{ to Interval}(f(x), f'(x)\epsilon).$$

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),
        Interval(h,eps))
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),
        Interval(H,H*eps))
return log(H)
```

Answer: 1.75

Same output $\Longrightarrow$ parameter transformation correctly applied

Parameter transformations

Language support for parameter transformations $f : \mathbb{R} \to \mathbb{R}$.

- Define $f(D)$ for distributions by defining rand, pdf, cdf of $f(D)$
- Define $f(I)$ for finite width intervals and infinitesimal width intervals

Requires that f is monotone and differentiable.

Original program:

```
observe(Normal(1.8,0.5),
        Interval(h,eps))
```

Answer: 1.75

Parameter transformations

Language support for parameter transformations $f : \mathbb{R} \to \mathbb{R}$.

- Define $f(D)$ for distributions by defining rand, pdf, cdf of $f(D)$
- Define $f(I)$ for finite width intervals and infinitesimal width intervals

Requires that f is monotone and differentiable.

Original program:

```
observe(Normal(1.8,0.5),
        Interval(h,eps))
```

Answer: 1.75

Logarithmic ruler program:

```
observe(exp(Normal(1.8,0.5)),
        exp(Interval(h,eps)))
```

Answer: 1.75

Parameter transformations

Language support for parameter transformations $f : \mathbb{R} \to \mathbb{R}$.

- Define $f(D)$ for distributions by defining rand, pdf, cdf of $f(D)$
- Define $f(I)$ for finite width intervals and infinitesimal width intervals

Requires that f is monotone and differentiable.

Original program:

```
observe(Normal(1.8,0.5),
        Interval(h,eps))
```

Answer: 1.75

Logarithmic ruler program:

```
observe(exp(Normal(1.8,0.5)),
        exp(Interval(h,eps)))
```

Answer: 1.75

In general: `observe(f(D), f(I))` $\equiv$ `observe(D, I)`

$\implies$ **programs are invariant under parameter transformations**

Recap

- Paradoxical behaviour: seemingly equivalent probabilistic programs give different outputs
- Root of the problem: conditioning on measure-zero events is ambiguous
- Solution: condition on intervals
- Restores rejection sampling as ground truth semantics
- Model measure-zero events as a limit, computed using infinitesimal arithmetic
- Semantics of `observe(D, Interval(x, eps))` agrees with the old `observe(D, x)` in most cases
- Programs are now invariant under parameter transformations
- Implementation in Julia

Comments or questions?

julesjacobs@gmail.com

Acknowledgements I thank Sriram Sankaranarayanan and the anonymous POPL reviewers for their outstanding feedback. I'm grateful to Ike Mulder, Arjen Rouvoet, Paolo Giarrusso, Dongho Lee, Ahmad Salim Al-Sibahi, Sam Staton, Christian Weilbach, and Robbert Krebbers for help, inspiration, and discussions.