An Algebraic Theory of Conditioning

Dario Stein, Sam Staton

University of Oxford

LAFI'21

How to express dependence on data?

Scoring vs **Exact Conditioning**

Scoring

```
position = normal(0, 100)

observe(normal(position, 5), 42)
```

i.e.

```
score(pdf_normal(position, 5)(42))
```

[Stan, WebPPL]

Exact Conditioning

```
# generative model
position    = normal(0, 100)
measurement = normal(position, 5)

# conditioning
measurement =:= 42
```

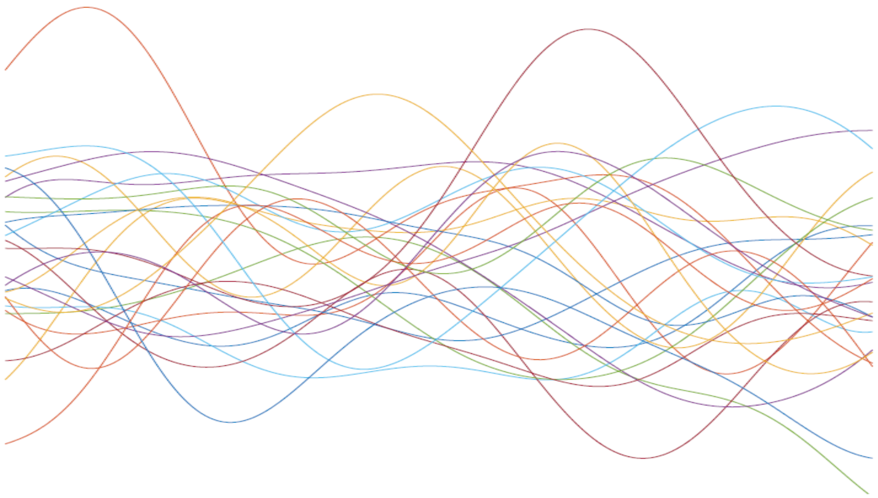
[Hakaru, Infer.NET]

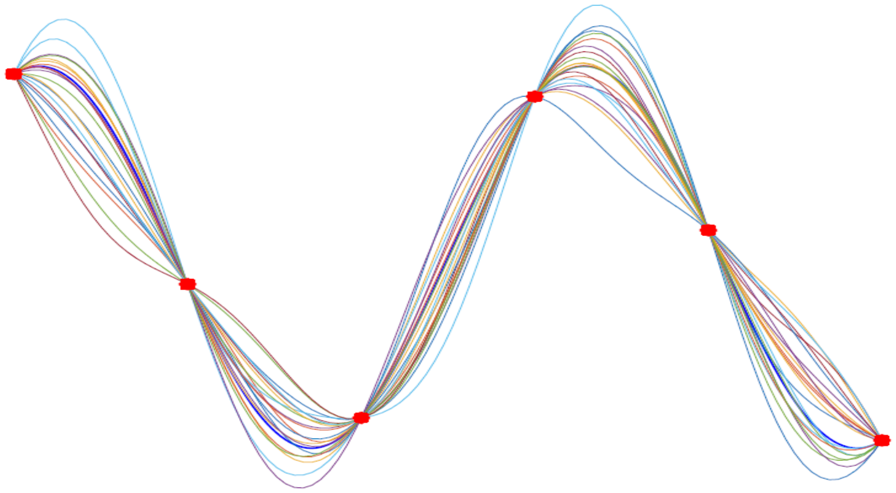
Advantages of Exact Conditioning

- Modularity

scoring statements would need to be interleaved with `gp_sample`

```
ys = gp_sample(n=100, kernel=rbf)
for (i,obs) in observations:
  ys[i] =:= obs
```





Advantages of Exact Conditioning

- Intuitiveness & Correctness

```
x = normal(0,1)
y = normal(0,1)
x =:= y
```

provably equal

```
x = normal(0,sqrt(1/2))
y = x

# x = y holds exactly
```

Closest approximation using scoring …

```
x = normal(0,1)
y = normal(0,1)

observe(normal(0,0.01), x-y)

# x = y does not hold!
# why 0.01?
```

A language for exact conditioning

- Toy language: Only Gaussians & affine maps + conditioning
  - Kalman filters, Ridge regression, Gaussian processes
- Reference implementation
  - Calling `normal(μ,σ)` allocates a latent RV
  - Maintain a joint prior over all RVs
  - When conditioning, update the prior
    - Symbolic inference: Gaussians are self-conjugate

```
import numpy as np
import scipy.linalg as linalg

class Gauss:

    def __init__(self, dom, cod, A=None,b=No
        self.dom = dom
        self.cod = cod

        self.A = A if A is not None else np
        self.b = b if b is not None else np
        self.Sigma = Sigma if Sigma is not N


    # Condition on the first n variables
    # cond : (a -> n + k) -> (a + n -> k)
    def cond(self, n):
        m = self.cod-n
        SigmaX = self.Sigma[0:n,0:n]
        SigmaY = self.Sigma[n:,n:]
        SigmaYX = self.Sigma[n:,0:n]
        M = self.A[0:n,:]
        N = self.A[n:,:]
        s = self.b[0:n]
        t = self.b[n:]

        SXi = np.linalg.pinv(SigmaX)
        D = SigmaYX.dot(SXi)
        A = np.block([D, N - D.dot(M)])
        b = t - D.dot(s)
        S = SigmaY - D.dot(SigmaYX.T)
        return Map(A,b,S)
```

Verifying properties

Commutativity

```
A1 =:= A2
B1 =:= B2
```

$\approx^?$

```
B1 =:= B2
A1 =:= A2
```

✓

Substitutivity

```
x =:= y ; C[x]
```

$\approx^?$

```
x =:= y ; C[y]
```

✓

Equivalent conditions

```
2x =:= -4y + 2
```

$\approx^?$

```
x + 2y =:= 1
```

✓

MIND BOREL'S PARADOX! [Shan]

```
x/y =:= 1
```

$\not\approx$

```
x - y =:= 0
```

Hard questions

- How to generalize to a *non-toy language*?
- Which nice behavior transfers?
- What should the general properties of (=:=) be?

WANTED:

General[1] compositional[2] semantics for exact conditioning

I. General

- Exact conditioning on continuous variables is hard
  - Borel's paradox & [Jules Jacobs,POPL'21]
- Conditioning is about …
  - Densities ✘
  - Limits ✘
  - Measure Theory ✘
  - **Universal property → Markov categories** [Fritz, Cho&Jacobs] ✓

II. Compositional

- Markov category conditionals are still a transformation of whole (closed) programs
- *Cond-construction*: Explain equivalence of *open programs*

```
x |- let y = normal(0,1) in x =:= y; return (x,y)
```

Markov categories + Cond-construction
= Compositional Exact Conditioning

Summary

- Language for Gaussian conditioning with good properties
- Those good properties generalize!
  - Conditioning via universal properties
- Markov categories + Cond = Compositional Exact Conditioning
  - Denotational semantics for symbolic disintegration [Shan]
- Study well-behaved Markov categories!

Bonus

- We can fully axiomatize the Gaussian language!
- The only things you need to know is
  - The language is commutative & discardable
  - IID Gaussians are invariant under rotations
  - Nice laws for conditioning

$$a, b \mid \varphi : 1 \vdash (a = b); \varphi[a] \equiv (a = b); \varphi[b] \tag{10}$$

$$a, b \mid \varphi : 1 \vdash (a = b); \nu x.\varphi[x] \equiv \nu x.(a = b); \varphi[x] \tag{11}$$

$$- \mid \varphi : 1 \vdash \nu x.(x = \underline{c}); \varphi[x] \equiv \varphi[\underline{c}] \tag{12}$$

BBonus: There is no Borel's paradox in Gaussian probability