CS6462

Probabilistic and Explainable AI

Lesson 11

FOPPL in Python

by Emil Vassev

February 10, 2025

$p(\theta \mid y)$

FOPPL – Revision

*First-Order Probabilistic Programming Language**:

- includes most common features of programming languages;
- conditional statements, e.g., ***if***
- primitive operations, e.g.,***+***,***-***,*******,***/***
- user-defined functions:
    - must be first order
    - cannot be recursive

```
(let [y-values [2.1 3.9 5.3 7.7 10.2]
      slope (sample (normal 0.0 10.0))
      intercept (sample (normal 0.0 10.0))]
  (foreach 5
    [x (range 1 6)
     y y-values]
    (let [fx (+ (* slope x) intercept)]
      (observe (normal fx 1.0) y))))
```

- FOPPL programs are models describing distributions over a finite number of random variables
- compile any program written in FOPPL to a data structure that represents a graphical model

** "An Introduction to Probabilistic Programming", book by J.W. van de Meent, B. Paige, H. Yang, F. Wood]*

FOPPL in Python

$p(\theta \mid y)$

Python Library developed by Tobias Kohn: PyFOPPL

- GitHub repository: https://github.com/Tobias-Kohn/PyFOPPL
- implementation of an Anglican/Clojure-based First Order Probabilistic Programming Language in Python
- takes FOPPL-code as input and creates a graph-based model for it
- system requirements: Python 3.4 +, torch, pygraphviz, networkx, matplotlib

FOPPL Translation to Graphical Model (recall): ***model*** object ***G=(V, A, P, Y)***

- ***V*** - a set of vertices that represent random variables
- ***A* = *V* × *V*** - a set of directed edges → conditional dependencies between variables
- ***P*** - a map mapping vertices to deterministic expressions of probability density or mass function for each random variable
- ***Y*** - a partial map mapping an observed random variable to deterministic expression ***E***

$$p(\theta \mid y)$$

FOPPL in Python (cont.)

How does it work?

1. provide an FOPPL model saved as a text file with the extension “***.clj***” in the parent directory of the PyFOPPL project
2. enable FOPPL-auto-imports through “***import foppl.imports***”
3. import the model as a normal Python module, e.g.:

```
import foppl.imports
import my_model
```

FOPPL Module integrated in PyFOPPL:

- ***model*** - the compiled model as a Python class → class-methods such as ***gen_prior_samples()***
- ***graph*** - the graph created from the original FOPPL program
- ***code*** - the Python-code created from the graph and then compiled into class

FOPPL in Python (cont.)

$p(\theta \mid y)$

Showing the FOPPL model as tuple:

PyFOPPL can show the tuple ***G=(V, A, P, Y)*** of the imported model:

```
import onegauss

print(help(onegauss.model))
```

Showing the PyFOPPL implementation of the FOPPL model:

PyFOPPL can show the generated python code for an FOPPL model:

```
print(onegauss.code)
```

Drawing the graph:

PyFOPPL can draw the graph model:

```
model = onegauss.model
model.graph.draw_graph()
```

required Python modules: networkx, matplotlib, graphviz

$$p(\theta \mid y)$$

PyFOPPL Modules

- *imports.py:* imports FOPPL programs into *Closure* data structures
- *foppl_objects.py:* data structures to store FOPPL programs
- *foppl_reader.py:* transforms loaded FOPPL programs into FOPPL Python objects
- *foppl_parser.py: transforms* FOPPL Python objects into an AST
- *foppl_ast.py*: abstract syntax tree (AST)
- *foppl_distributions.py:*
  - discrete distributions
  - continuous distributions
- *graphs.py:* creates visual graph model

```
discrete_distributions = {
    "Bernoulli",
    "Categorical",
    "Discrete",
    "Multinomial",
    "Poisson"
}

continuous_distributions = {
    "Beta",
    "Cauchy",
    "Dirichlet",
    "Exponential",
    "Gamma",
    "HalfCauchy",
    "LogNormal",
    "MultivariateNormal",
    "Normal",
    "Uniform"
}
```

$$p(\theta \mid y)$$

Example Python Program

FOPPL model/program *onegause.clj*:

(let [x (sample (normal 1.0 5.0))

y (+ x 1)]

(observe (normal y 2.0) 7.0)

y)

Python code processing *onegause.clj*:

```
"""
A simple example of FOPPL in Python.

After importing FOPPL, we display the entire documentation for the
generated model.
"""
import foppl.imports

import onegauss

print(help(onegauss.model))

print(onegauss.code)

model = onegauss.model
model.graph.draw_graph()
```

```
class model(builtins.object)
 |  Vertices V:
 |    x20001, y20002
 |  Arcs A:
 |    (x20001, y20002)
 |  Conditional densities C:
 |    x20001 -> dist.Normal(mu=1.0, sigma=2.23606797749979)
 |    y20002 -> dist.Normal(mu=(x20001 + 1), sigma=1.4142135623730951)
 |  Observed values O:
 |    y20002 -> 7.0
 |
```

```
import math
import numpy as np

class model(object):
        @classmethod
        def get_vertices(self):
                vertices = {'x20001', 'y20002'}
                return list(vertices)

        @classmethod
        def get_arcs(self):
                arcs = {('x20001', 'y20002')}
                return list(arcs)

        @classmethod
        def get_discrete_distributions(self):
                disc_dists = {}
                return disc_dists

        @classmethod
        def get_continuous_distributions(self):
                cont_dists = {
                  'x20001': 'Normal'
                }
                return cont_dists
```

$$p(\theta \mid y)$$

Example Python Program (cont.)

FOPPL model/program *onegause.clj*:

(let [x (sample (normal 1.0 5.0))

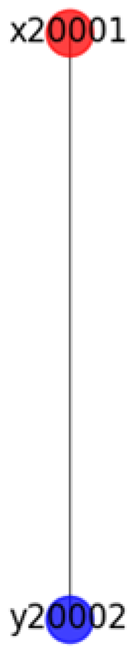
y (+ x 1)]

(observe (normal y 2.0) 7.0)

y)

Python code processing *onegause.clj*:

```
"""
A simple example of FOPPL in Python.

After importing FOPPL, we display the entire documentation for the
generated model.
"""
import foppl.imports

import onegauss

print(help(onegauss.model))

print(onegauss.code)

model = onegauss.model
model.graph.draw_graph()
```

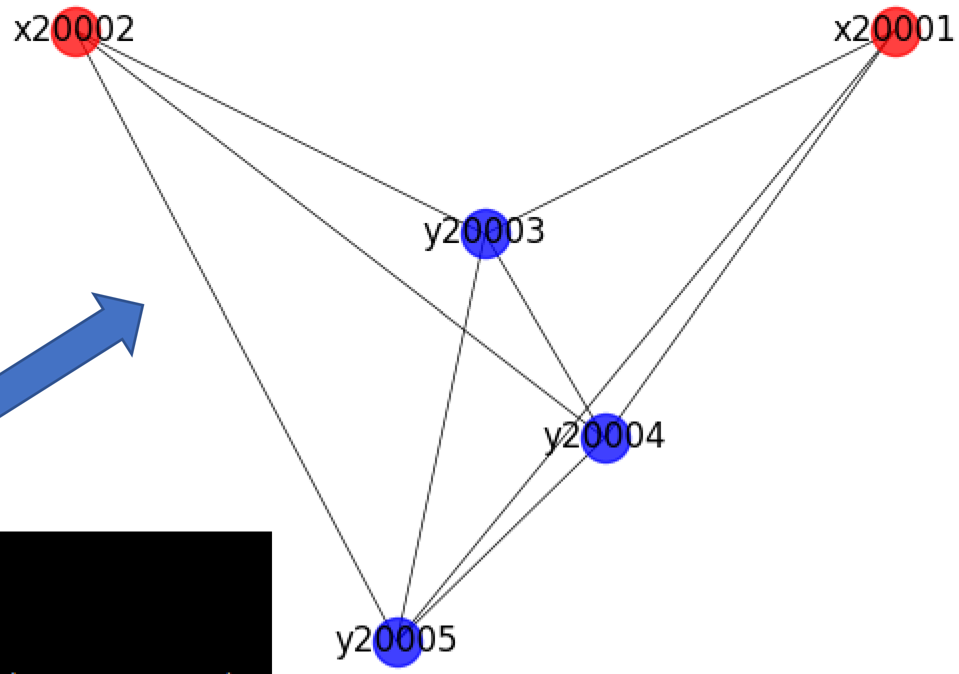


$$p(\theta \mid y)$$

Example – Linear Regression

```
(defn observe-data [_ data slope bias]
  (let [xn (first data)
        yn (second data)
        zn (+ (* slope xn) bias)]
    (observe (normal zn 1.0) yn)
    (rest (rest data))))

(let [slope (sample (normal 0.0 10.0))
      bias  (sample (normal 0.0 10.0))
      data  (vector 1.0 2.1 2.0 3.9 3.0 5.3)]
  (loop 3 data observe-data slope bias)
   (vector slope bias))
```

```
lass model(builtins.object)
 |  Vertices V:
 |    x20001, x20002, y20003, y20004, y20005
 |  Arcs A:
 |    (x20002, y20005), (x20002, y20004), (x20001, y20003), (y20004, y20005), (y20003, y20005), (y20003, y20004),
0001, y20005), (x20001, y20004), (x20002, y20003)
 |  Conditional densities C:
 |    x20001 -> dist.Normal(mu=0.0, sigma=3.1622776601683795)
 |    x20002 -> dist.Normal(mu=0.0, sigma=3.1622776601683795)
 |    y20003 -> dist.Normal(mu=((x20001 * [1.0, 2.1, 2.0, 3.9, 3.0, 5.3][0]) + x20002), sigma=1.0)
 |    y20004 -> dist.Normal(mu=((x20001 * [1.0, 2.1, 2.0, 3.9, 3.0, 5.3][1:][1:][0]) + x20002), sigma=1.0)
 |    y20005 -> dist.Normal(mu=((x20001 * [1.0, 2.1, 2.0, 3.9, 3.0, 5.3][1:][1:][1:][1:][0]) + x20002), sigma=1.0)
 |  Observed values O:
 |    y20003 -> [1.0, 2.1, 2.0, 3.9, 3.0, 5.3][1]
 |    y20004 -> [1.0, 2.1, 2.0, 3.9, 3.0, 5.3][1:][1:][1]
 |    y20005 -> [1.0, 2.1, 2.0, 3.9, 3.0, 5.3][1:][1:][1:][1:][1]
 |
```

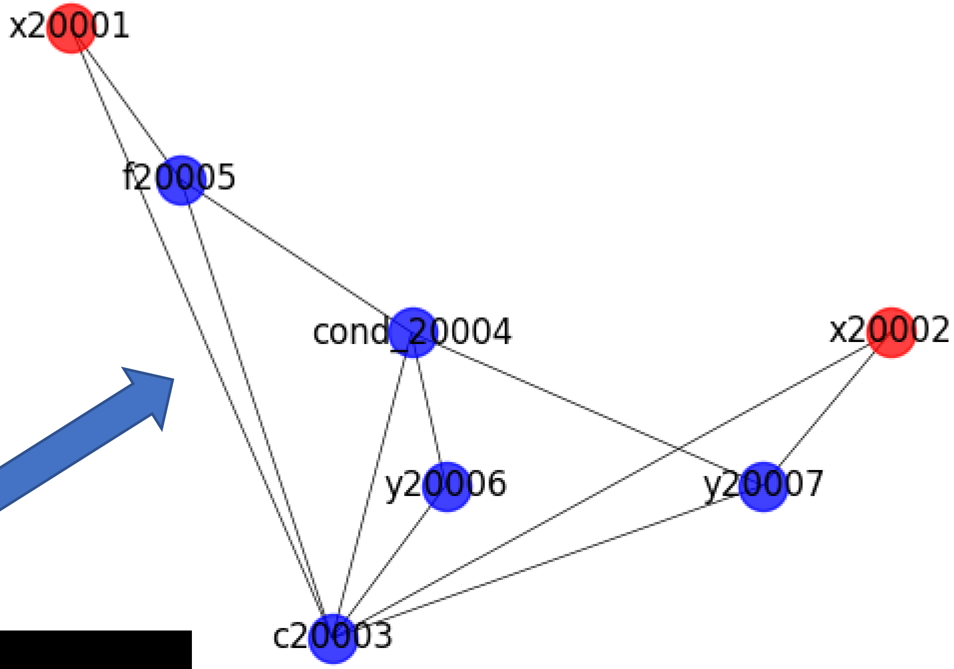


$$p(\theta \mid y)$$

Example – Normal Distribution with If

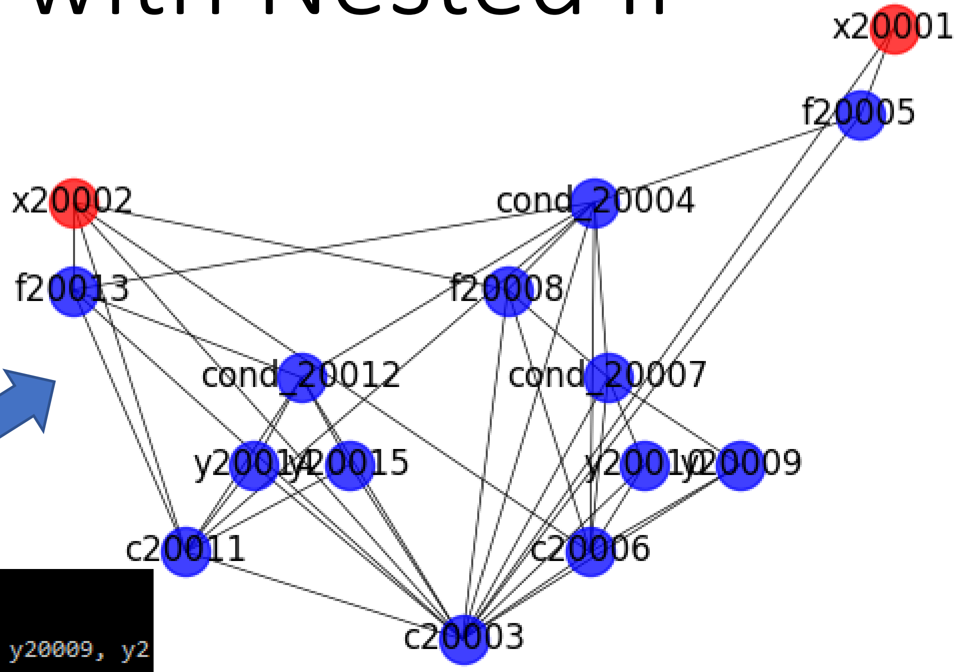
```
(let [x1 (sample (normal 0 2))
      x2 (sample (normal 0 4))]
  (if (> x1 0)
    (observe (normal x2 1) 1)
    (observe (normal -1 1) 1))
  x1)
```

```
class model(builtins.object)
 |   Vertices V:
 |     c20003, cond_20004, f20005, x20001, x20002, y20006, y20007
 |   Arcs A:
 |     (cond_20004, y20006), (cond_20004, c20003), (x20002, y20007), (x20001, f20005), (y20006, c20003), (cond_20004,
y20007), (f20005, c20003), (x20001, c20003), (f20005, cond_20004), (y20007, c20003), (x20002, c20003)
 |   Conditional densities C:
 |     x20001 -> dist.Normal(mu=0, sigma=1.4142135623730951)
 |     f20005 -> -x20001
 |     cond_20004 -> (f20005 >= 0).data[0]
 |     y20006 -> dist.Normal(mu=-1, sigma=1.0)
 |     x20002 -> dist.Normal(mu=0, sigma=2.0)
 |     y20007 -> dist.Normal(mu=x20002, sigma=1.0)
 |     c20003 -> y20006 if cond_20004 else y20007
 |   Observed values O:
 |     y20006 -> 1
 |     y20007 -> 1
```



Example – Normal Distribution with Nested If

```
(let [x (sample (normal 0 1))
      z (sample (categorical [3 4 5]))]
  (if (> x 0)
    (if (< z 1)
      (observe (normal 0.5 1) 1)
      (observe (normal 2 1) 1))
    (if (> z -1)
      (observe (normal -0.5 1) 1)
      (observe (normal -2 1) 1)))
  x)
```

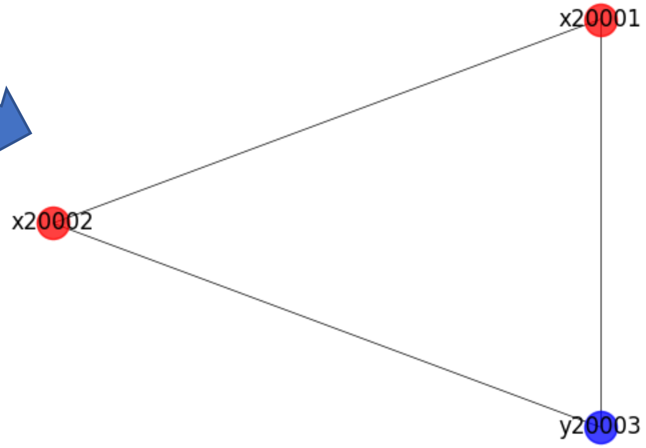



```
lass model(builtins.object)
 |  Vertices V:
 |    c20003, c20006, c20011, cond_20004, cond_20007, cond_20012, f20005, f20008, f20013, x20001, x20002, y20009, y2
0010, y20014, y20015
 |  Arcs A:
 |    (f20013, c20011), (cond_20007, y20010), (c20006, c20003), (x20002, f20008), (y20010, c20003), (cond_20007, c20
003), (y20009, c20006), (cond_20004, c20006), (cond_20012, y20015), (x20002, c20011), (f20005, c20003), (x20002, f20
013), (c20011, c20003), (y20009, c20003), (cond_20004, c20003), (f20008, cond_20007), (f20005, cond_20004), (y20014,
 c20003), (cond_20012, c20003), (y20015, c20003), (cond_20004, cond_20012), (cond_20012, y20014), (f20013, c20003),
(f20008, c20006), (x20002, c20006), (cond_20004, f20008), (f20013, cond_20012), (cond_20007, y20009), (cond_20004, c
20011), (x20002, c20003), (f20008, c20003), (cond_20004, f20013), (y20014, c20011), (x20001, f20005), (cond_20012, c
20011), (x20001, c20003), (y20015, c20011), (cond_20004, cond_20007), (y20010, c20006), (cond_20007, c20006)
 |  Conditional densities C:
 |    x20001 -> dist.Normal(mu=0, sigma=1.0)
 |    f20005 -> -x20001
 |    cond_20004 -> (f20005 >= 0).data[0]
 |    x20002 -> dist.Categorical(ps=[3, 4, 5])
 |    f20008 -> (-1 - x20002)
 |    cond_20007 -> (f20008 >= 0).data[0]
 |    y20009 -> dist.Normal(mu=-2, sigma=1.0)
 |    y20010 -> dist.Normal(mu=-0.5, sigma=1.0)
 |    c20006 -> y20009 if cond_20007 else y20010
```

Examples – Hidden Markov Model

```
(defn data [n]
  (let [points (vector 0.9 0.8 0.7 0.0 -0.025
                       5.0 2.0 0.1 0.0 0.13
                       0.45 6.0 0.2 0.3 -1.0 -1.0)]
    (get points n)))

;; Define the init, transition, and observation distributions
(defn get-init-params []
  (vector (/ 1. 3.) (/ 1. 3.) (/ 1. 3.)))

(defn get-trans-params [k]
  (nth (vector (vector 0.1  0.5  0.4 )
               (vector 0.2  0.2  0.6 )
               (vector 0.7 0.15 0.15 )) k))

(defn get-obs-dist [k]
  (nth (vector (normal -1. 1.)
               (normal  1. 1.)
               (normal  0. 1.)) k))

;; Function to step through HMM and sample latent state
(defn hmm-step [n states]
  (let [next-state (sample (categorical (get-trans-params (last states))))]
    (observe (get-obs-dist next-state) (data n))
    (conj states next-state)))

;; Loop through the data
(let [init-state (sample (categorical (get-init-params)))]
  (loop 1 (vector init-state) hmm-step))
```

```
class model(builtins.object)
 |  Vertices V:
 |    x20001, x20002, y20003
 |  Arcs A:
 |    (x20001, x20002), (x20002, y20003), (x20001, y20003)
 |  Conditional densities C:
 |    x20001 -> dist.Categorical(ps=[0.3333333333333333, 0.3333333333333333, 0.3333333333333333])
 |    x20002 -> dist.Categorical(ps=[[0.1, 0.5, 0.4], [0.2, 0.2, 0.6], [0.7, 0.15, 0.15]][int(x20001)])
 |    y20003 -> [dist.Normal(mu=-1.0, sigma=1.0), dist.Normal(mu=1.0, sigma=1.0), dist.Normal(mu=0.0, sigma=1.0)][int(x20002)]
 |  Observed values O:
 |    y20003 -> [0.9, 0.8, 0.7, 0.0, -0.025, 5.0, 2.0, 0.1, 0.0, 0.13, 0.45, 6.0, 0.2, 0.3, -1.0, -1.0][0]
 |
```



$p(\theta \mid y)$

Summary

FOPPL in Python:

- Python Library developed by Tobias Kohn: PyFOPPL
- implementation of an Anglican/Clojure-based FOPPL in Python
- takes FOPPL-code as input and creates a graph-based model for it

How does it work?

1. provide an FOPPL model saved as a text file with the extension "***.clj***" in the parent directory of the PyFOPPL project
2. enable FOPPL-auto-imports through "***import foppl.imports***"
3. import the model as a normal Python module, e.g.:

Next Lesson:

Bayesian Generalized Linear Models - Likelihood and Maximum Likelihood Principles

Thank You!

Questions?