CS492: Probabilistic Programming

Introduction

Hongseok Yang
KAIST

This Review … discusses some of the state-of-the-art advances in the field, namely, probabilistic programming, Bayesian optimization, data compression and automatic model discovery.

Zoubin Gharahmani
2015 Nature Review

What is probabilistic programming?

(Bayesian) probabilistic modelling of data

1. Develop a new probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algo., fit the model to the data.

(Bayesian) probabilistic modelling of data in a prob. prog. language

1. Develop a new probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algo., fit the model to the data.

(Bayesian) probabilistic modelling of data in a prob. prog. language as a program

1. Develop a new probabilistic (generative) model.
2. Design an inference algorithm for the model.
3. Using the algo., fit the model to the data.

(Bayesian) probabilistic modelling of data

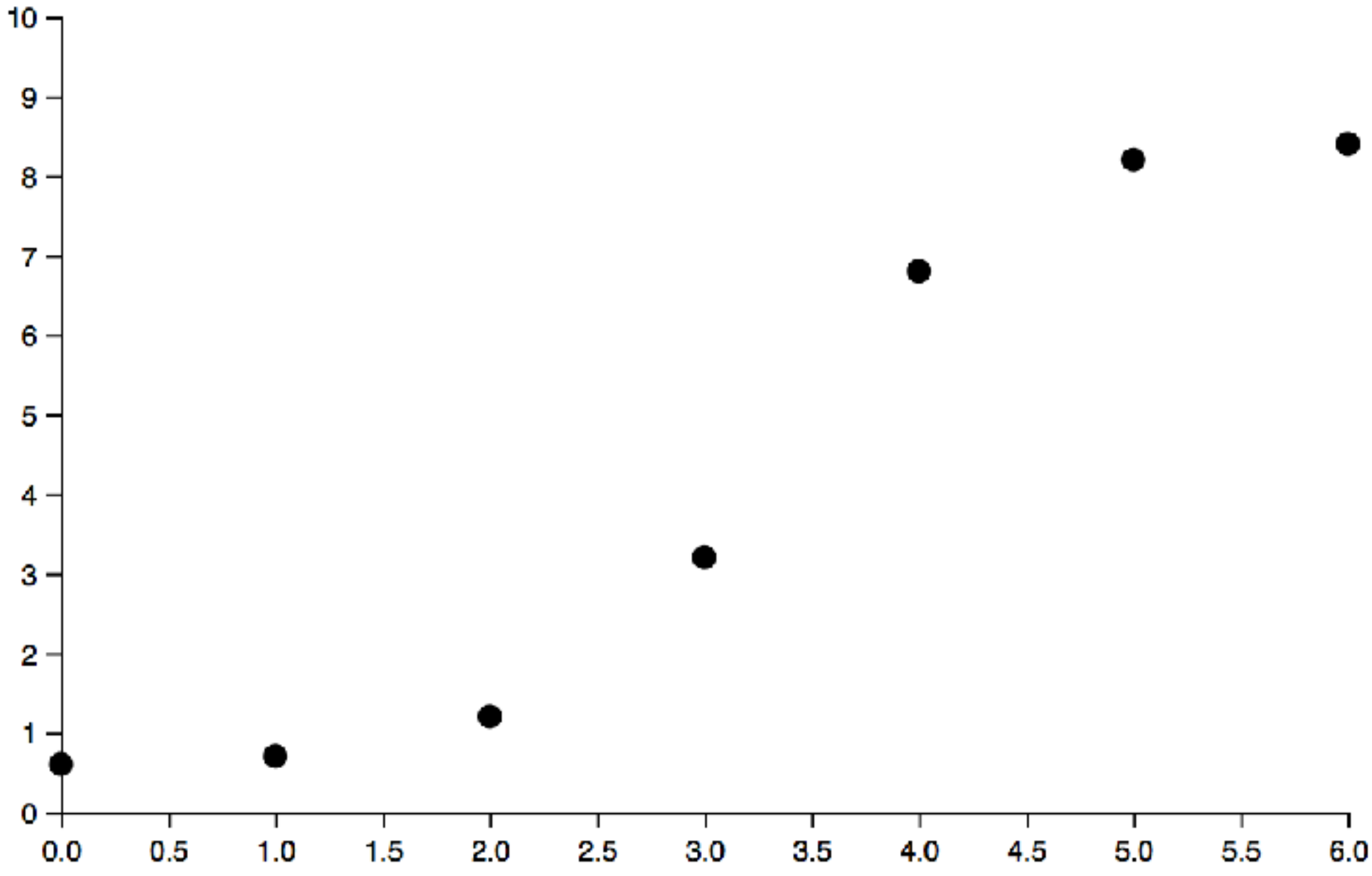
in a prob. prog. language

as a program

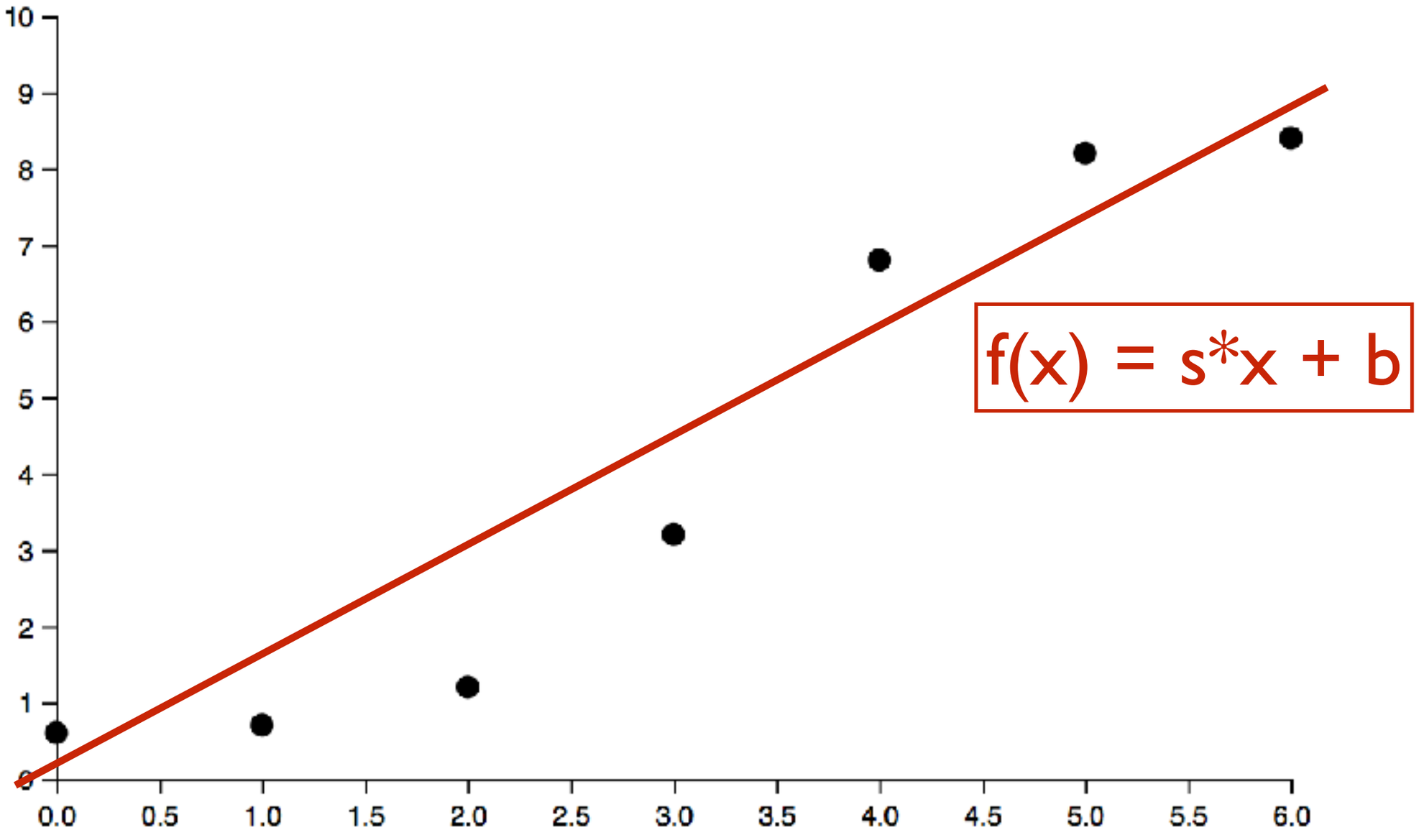
1. Develop a new probabilistic (generative) model.
2. ~~Design an inference algorithm for the model.~~
3. Using ~~the algo~~, fit the model to the data.

a generic inference algo.
of the language

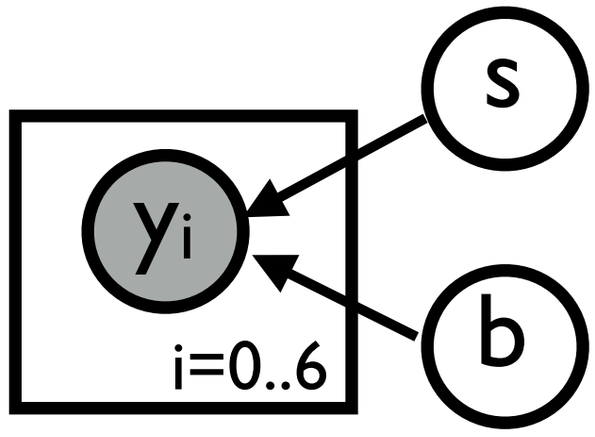
Line fitting



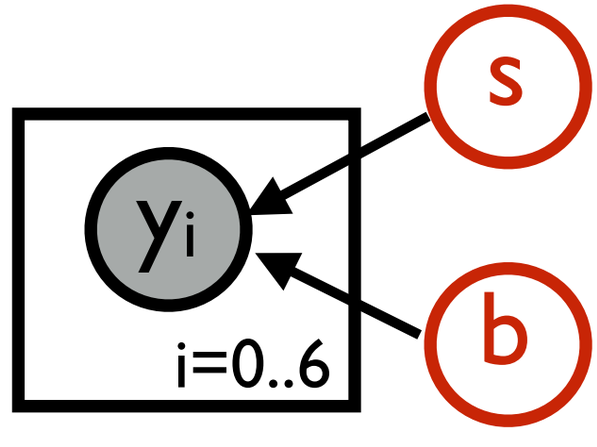
Line fitting



Bayesian generative model

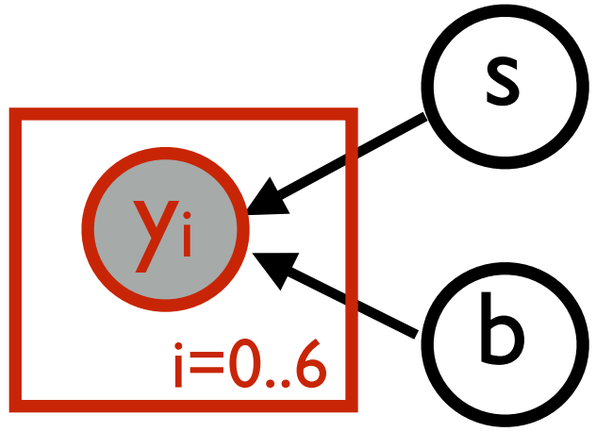


Bayesian generative model



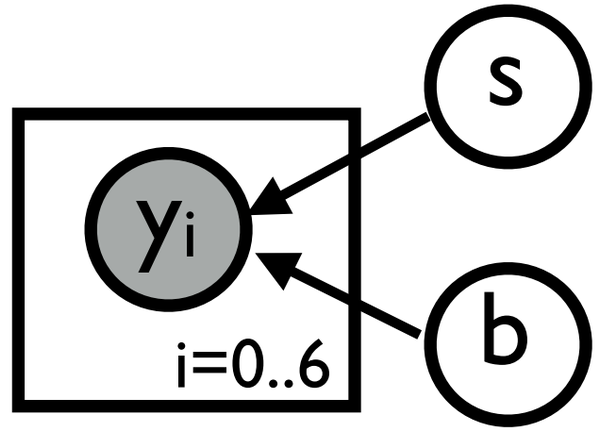

```
s     ~ normal(0, 2)
b     ~ normal(0, 6)
```

Bayesian generative model




```
s      ~ normal(0, 2)
b      ~ normal(0, 6)
f(x)   = s*x + b
y_i    ~ normal(f(i), 0.5)
           where i = 0 .. 6
```

Bayesian generative model




```
s      ~ normal(0, 2)
b      ~ normal(0, 6)
f(x)   = s*x + b
y_i    ~ normal(f(i), 0.5)
             where i = 0 .. 6
```

Q: posterior of (s,b) given y_0=0.6, …, y_6=8.4?

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Posterior of s and b given y_i's

$$p(s, b \mid y_0, .., y_6) = \frac{p(y_0, .., y_6 \mid s,b) \times p(s,b)}{p(y_0, .., y_6)}$$

Anglican program

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]
```

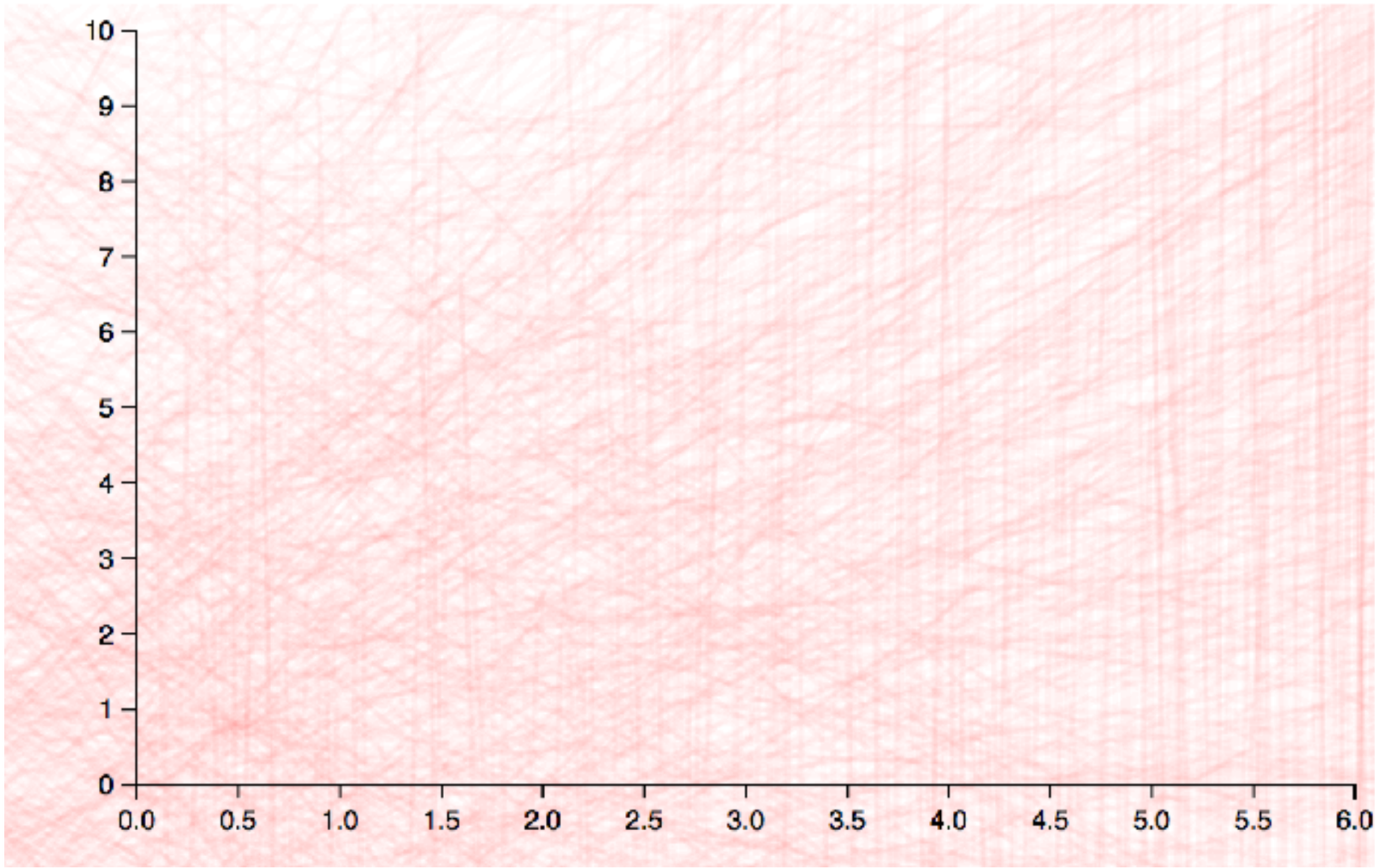
Anglican program

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)
```

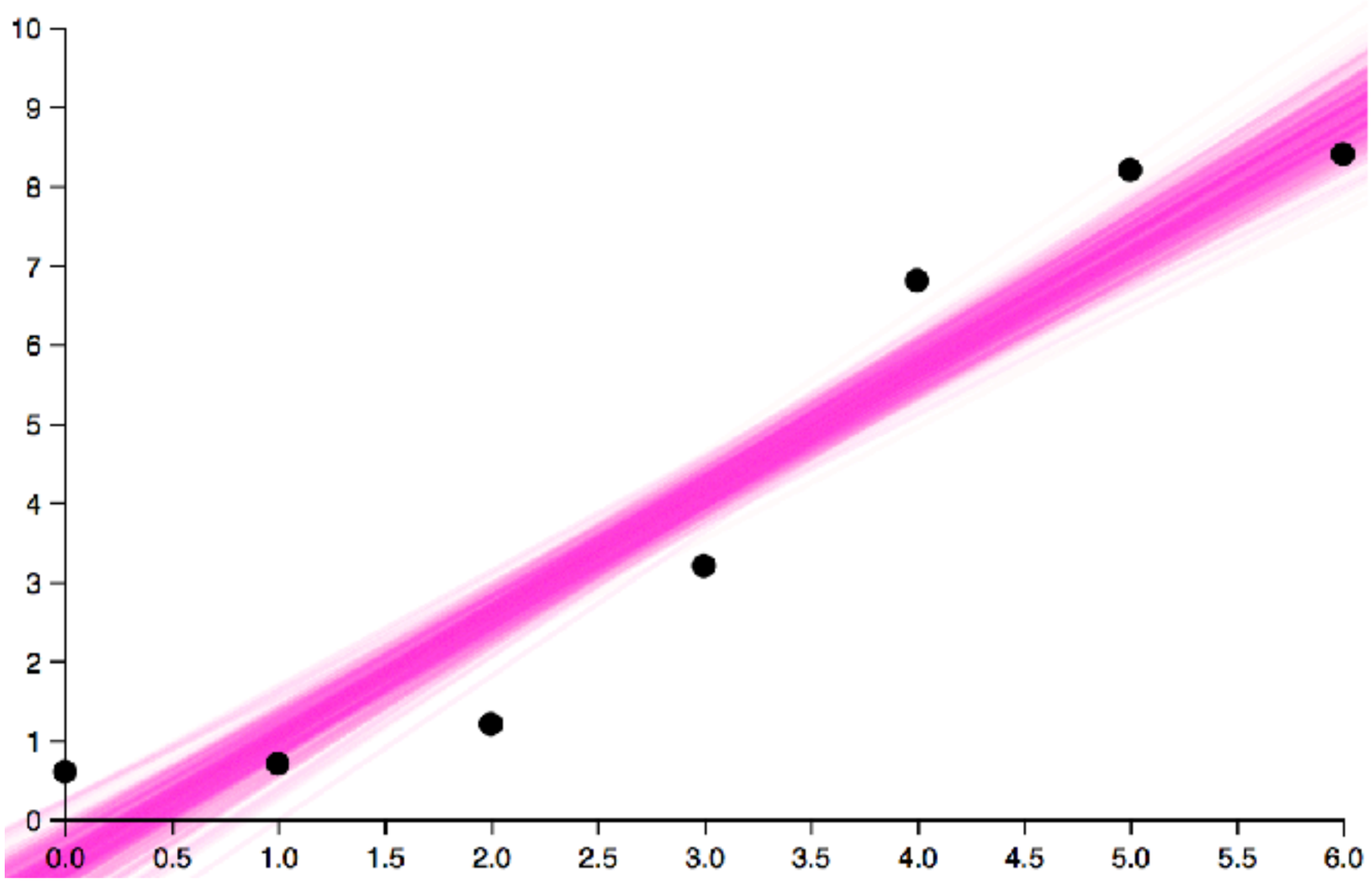
Anglican program

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)

[s b])
```

Samples from prior



Samples from posterior

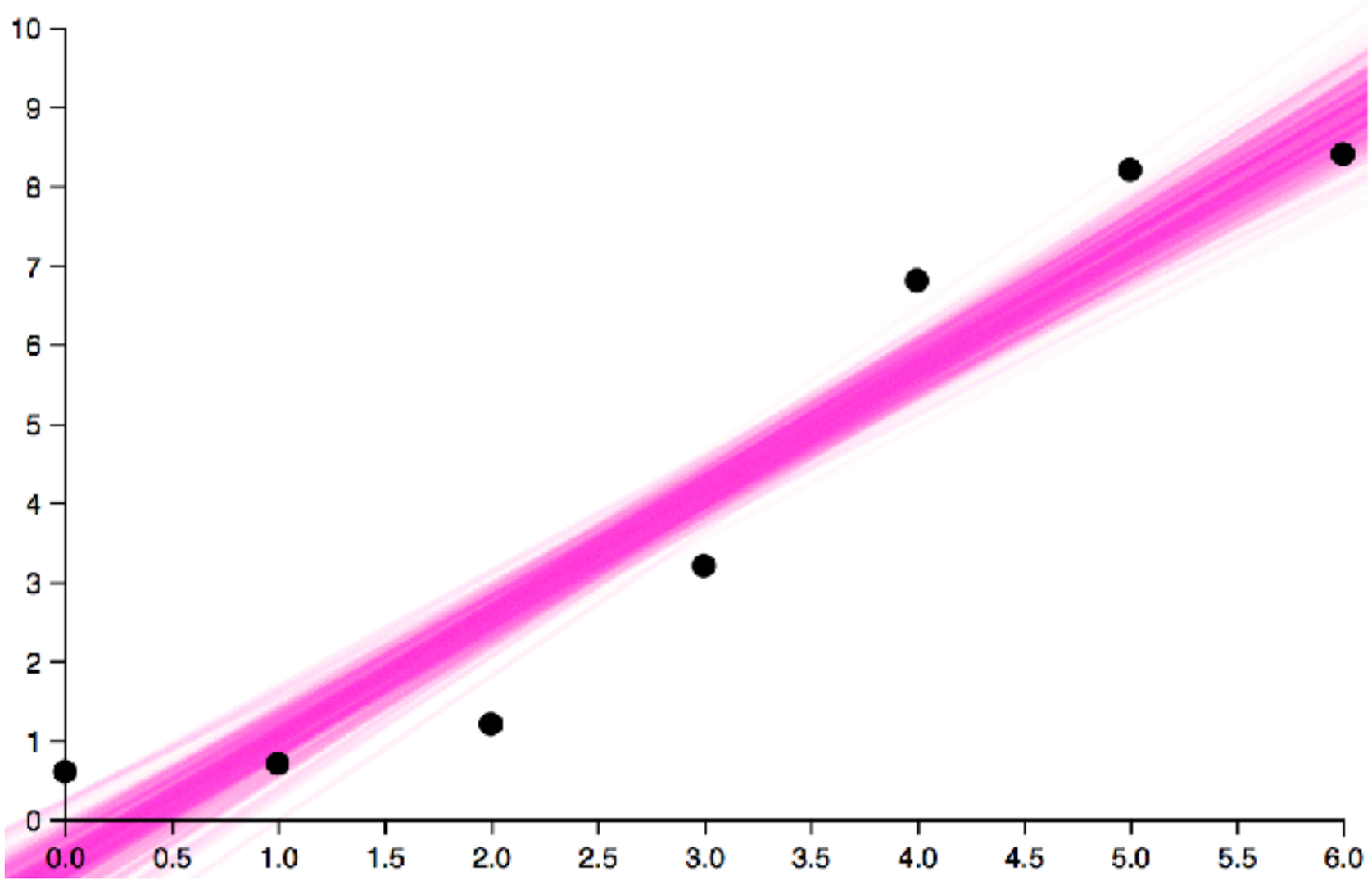


Why should one care about prob. programming?

"Because probabilistic programming is a good way to build an AI." (My ML colleague)

Prob. programming languages enable one to build and explore highly complex models.

Underfit?



```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]


    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

  [s b])
```

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]


    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

[s b])
```

Functions as first-class citizen.

```
(let [s (sample (normal 0 2))
      b (sample (normal 0 6))
      f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

[s b])
f)
```

Functions as first-class citizen.

```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (F)]
  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)
```

~~[s b]~~
f)

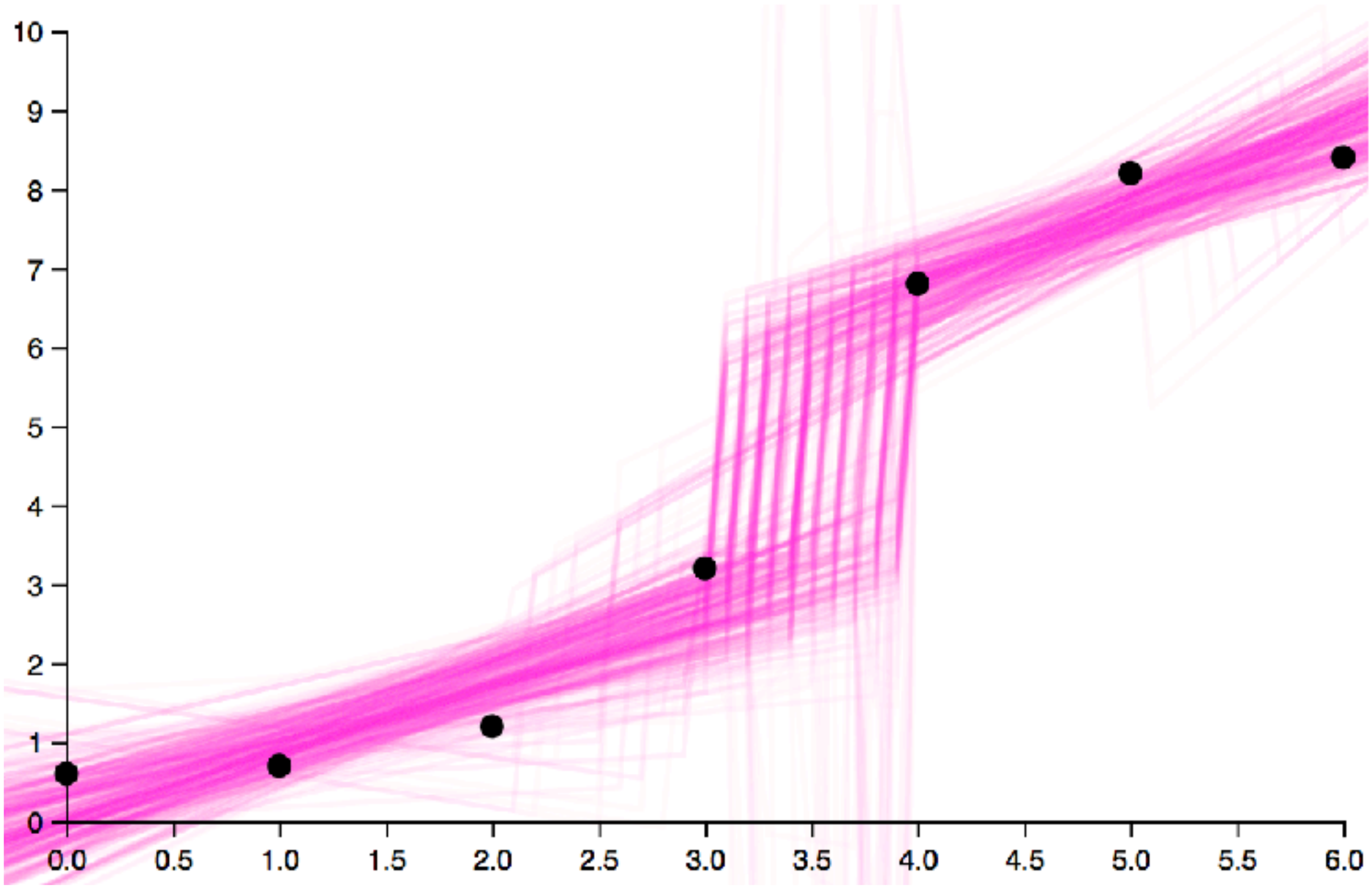
Functions as first-class citizen.

```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6)]
  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)
```

~~[s b])~~
f)

Functions as first-class citizen.

Samples from posterior



```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6)]
  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)
```

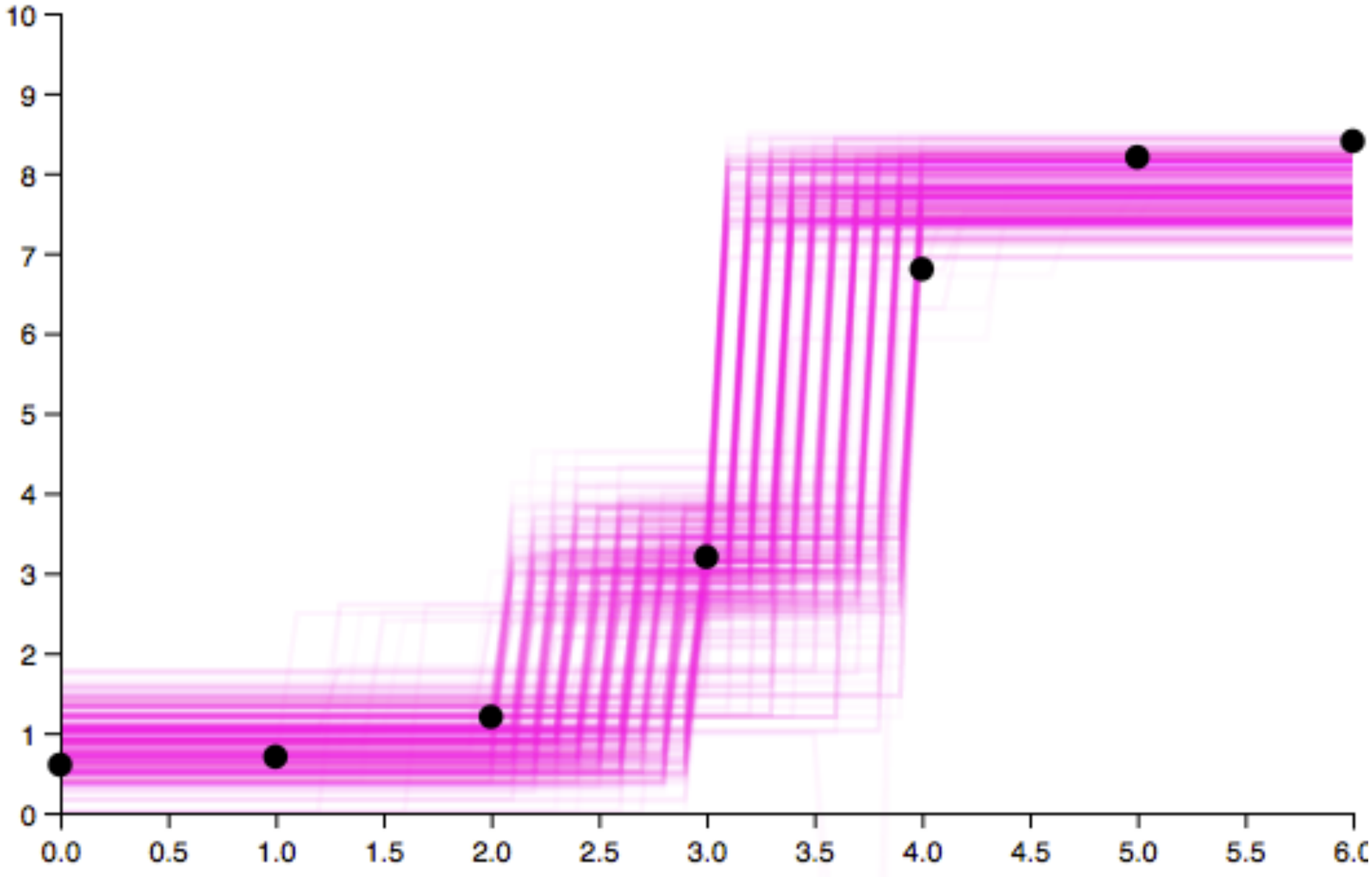
~~[s b]~~
f)

[Q] Change the model so that it finds constant functions with change points.

```
(let [F (fn []
          (let [s (sample (normal 0 2))
                b (sample (normal 0 6))]
            (fn [x] (+ (* s x) b))))
      f (add-change-points F 0 6)]
  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)
  [s b])
  f)
```

[Q] Change the model so that it finds constant functions with change points.

Samples from posterior



Success story 1: Captcha breaking

SMKBDF

Le, Baydin, Wood [2016]

Success story 1: Captcha breaking

SMKBDF

1. Sample a string.

Le, Baydin, Wood [2016]

Success story 1: Captcha breaking

SMKBDF

1. Sample a string.
2. Generate an image.

Le, Baydin, Wood [2016]

Success story 1: Captcha breaking

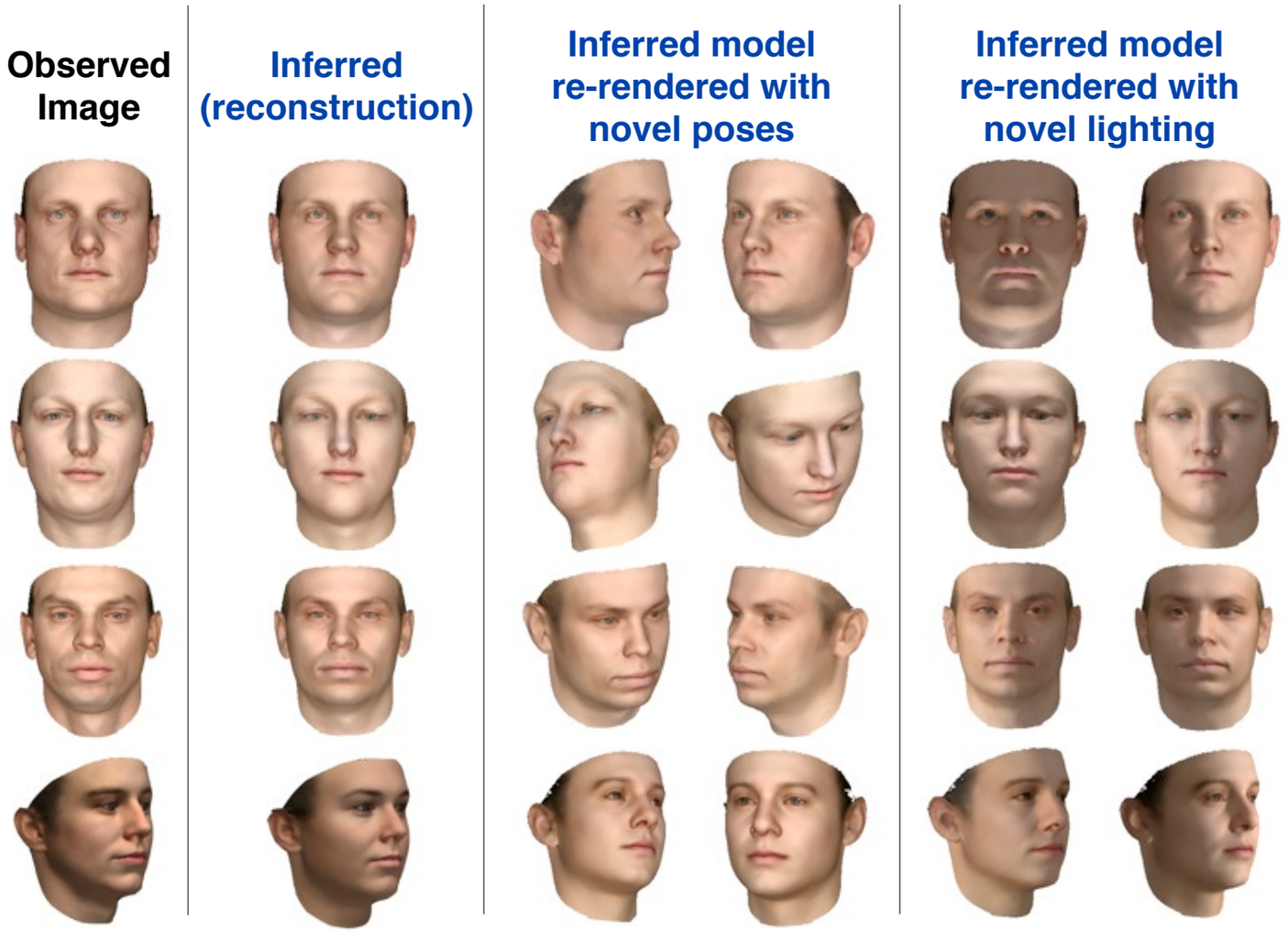
posterior inference

SMKBDF

1. Sample a string.
2. Generate an image.

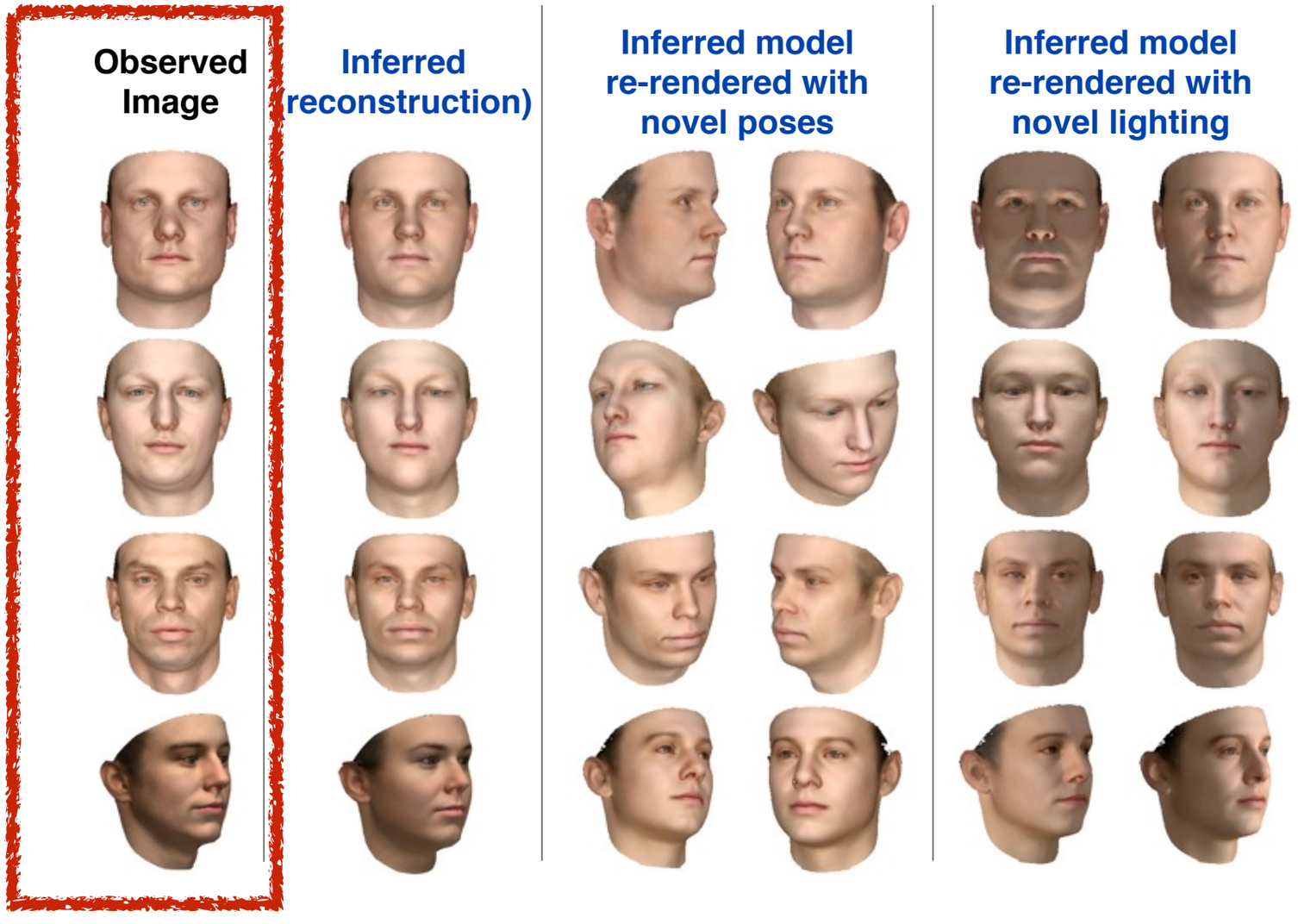
Le, Baydin, Wood [2016]

Success story 2: Inverse graphics



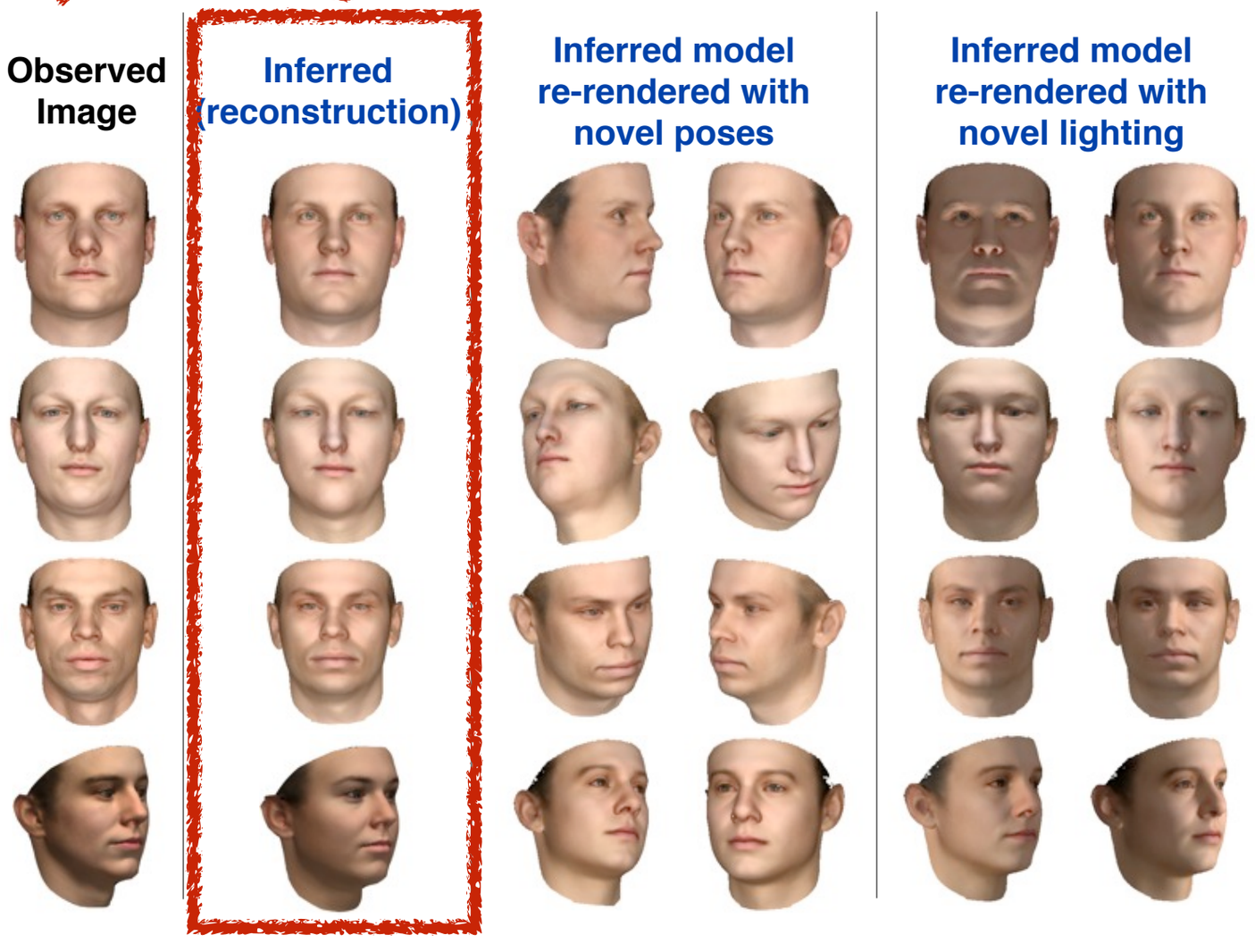
Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Success story 2: Inverse graphics



Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

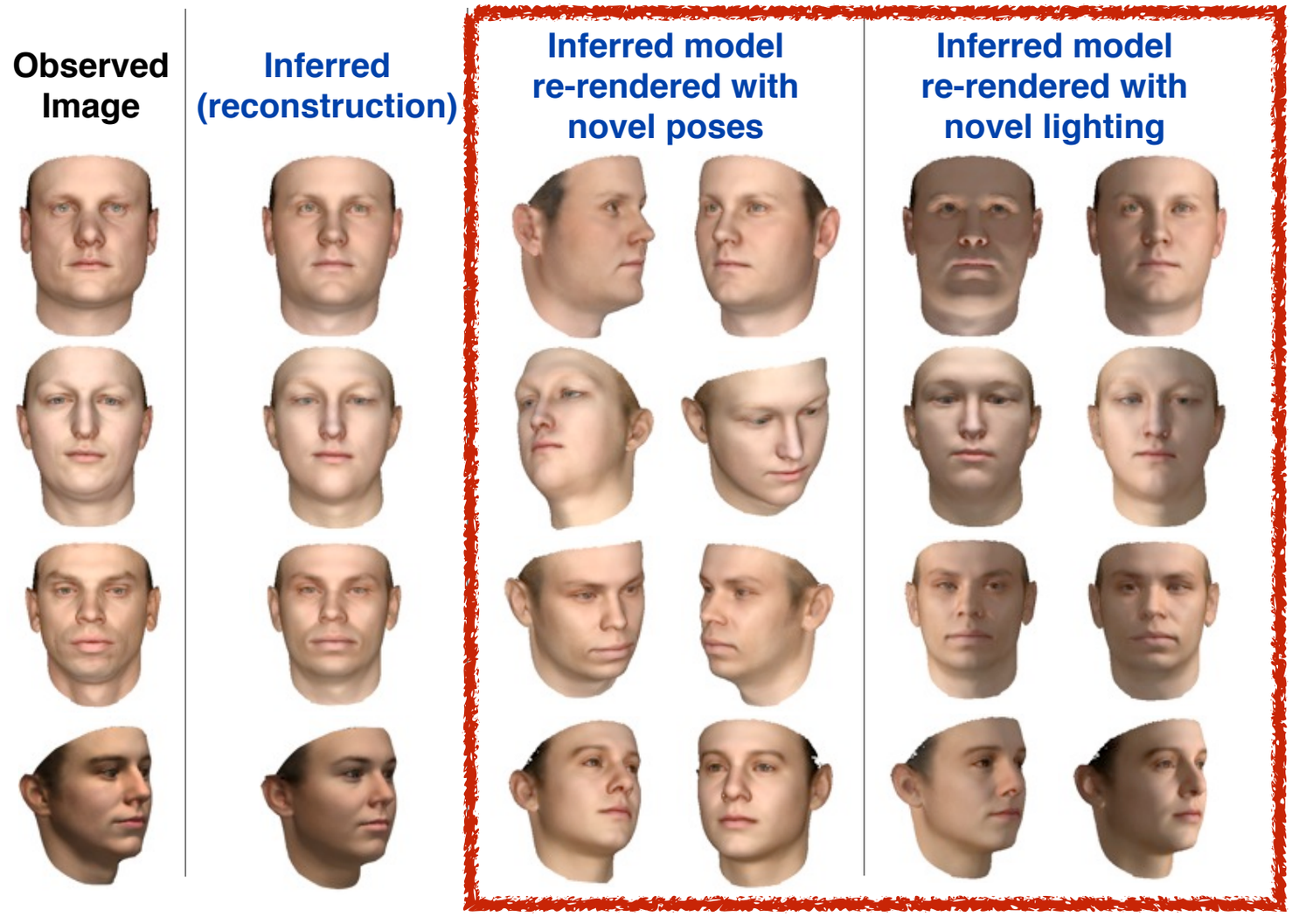
Success story 2: Inverse graphics



3D model

Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Success story 2: Inverse graphics

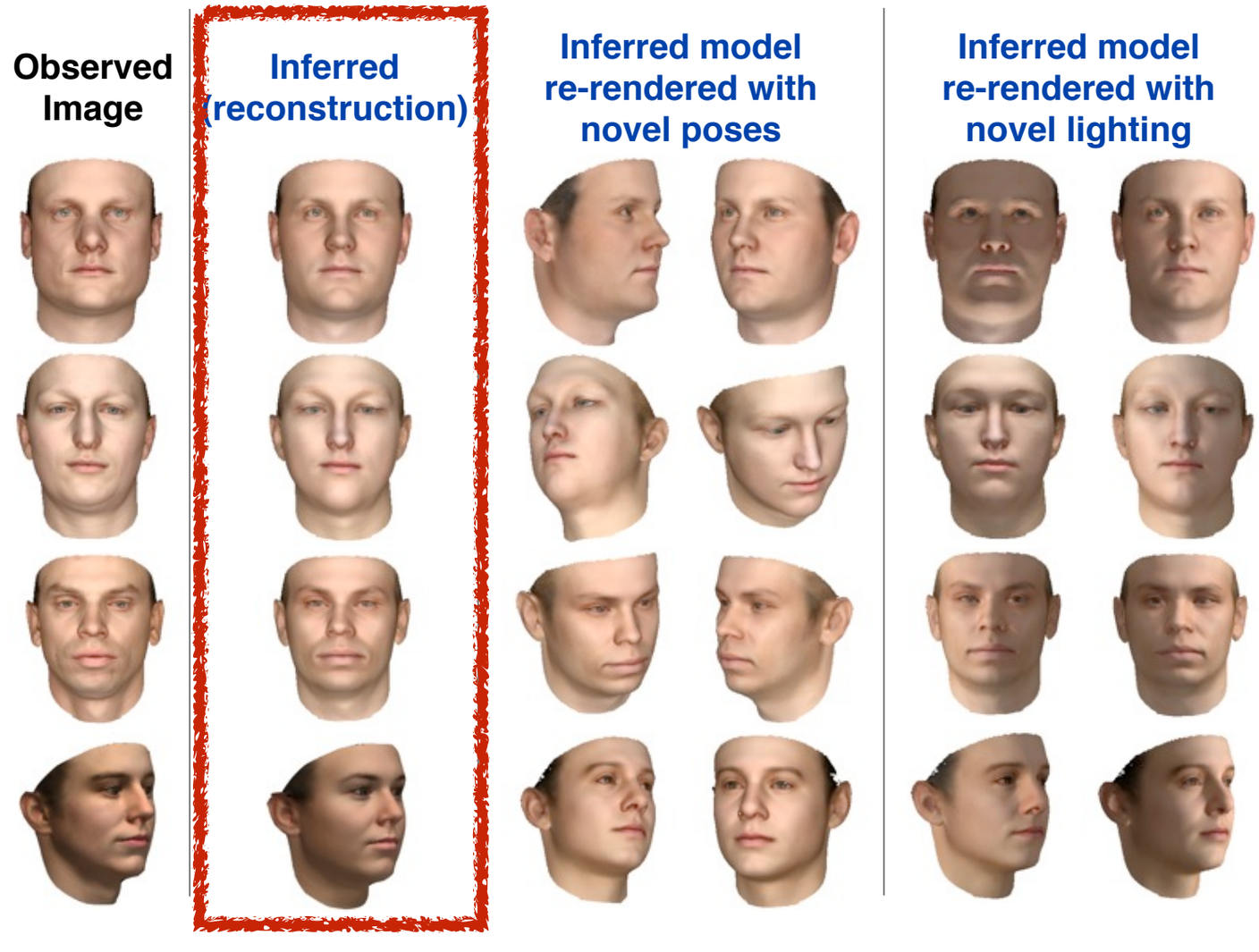


Transformed 3D models

Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

ss story 2: e graphics

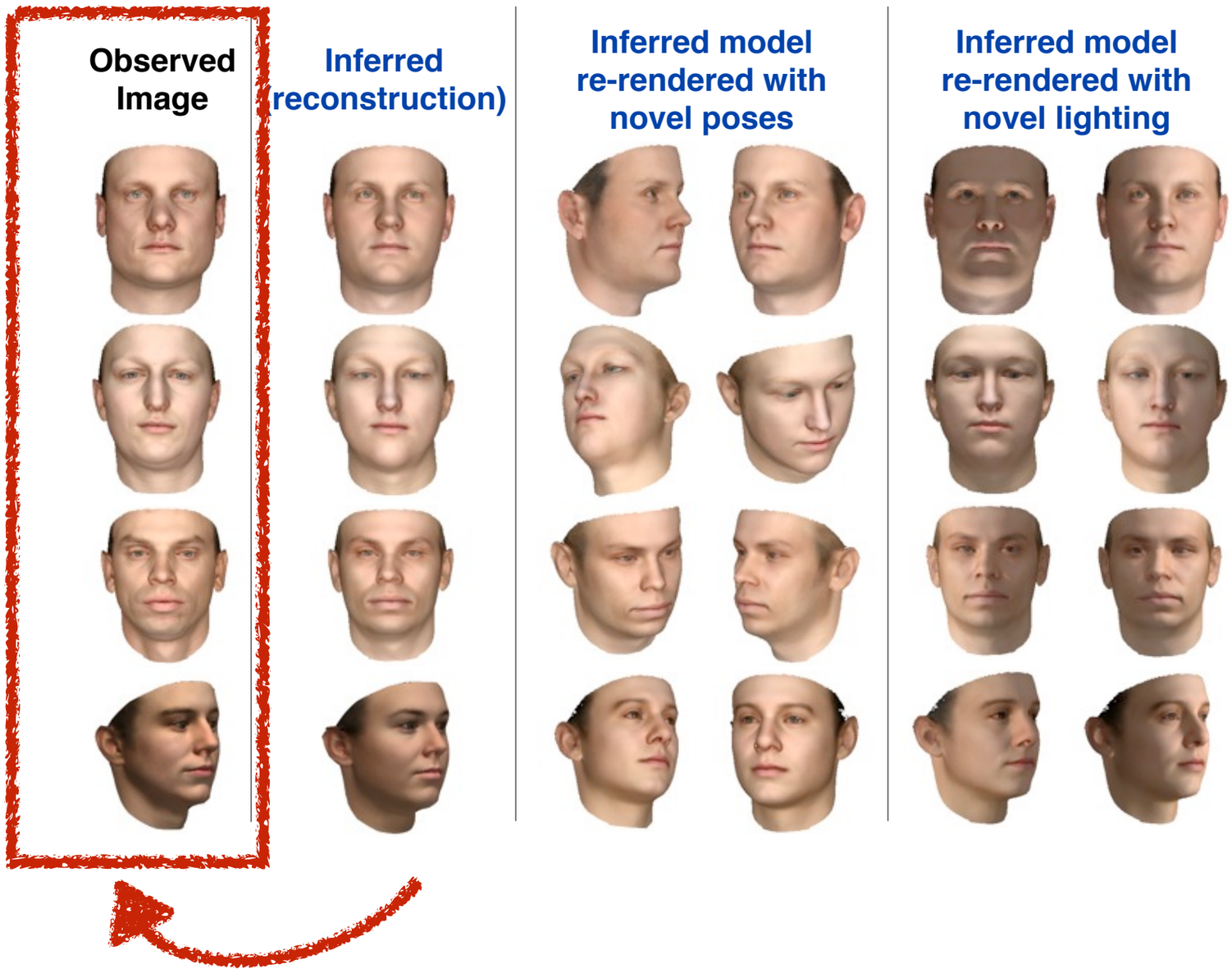
1. Sample a 3D model.



Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Success story 2: inverse graphics

1. Sample a 3D model.
2. Generate an image.

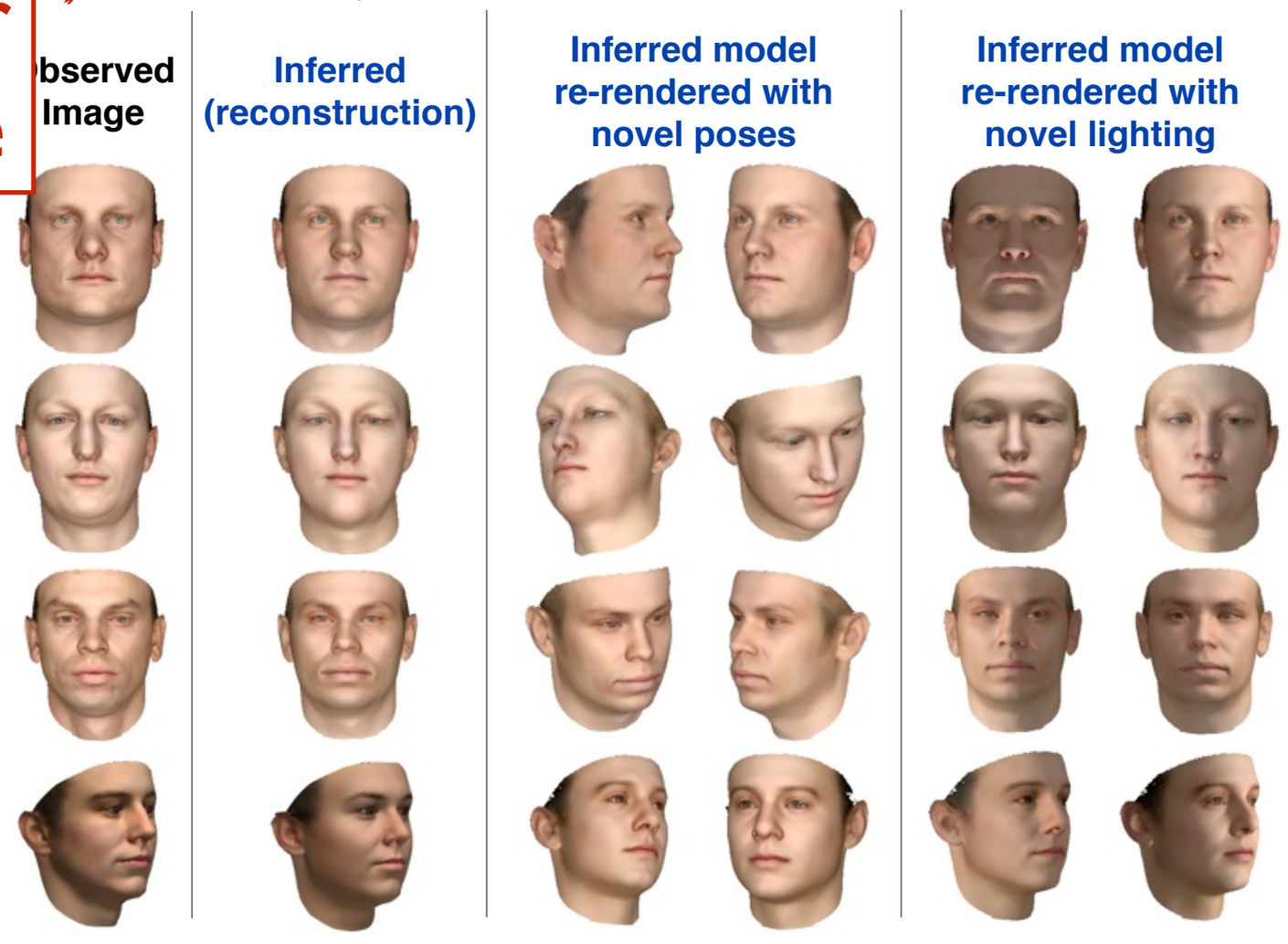


Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

...ss story 2: ...e graphics

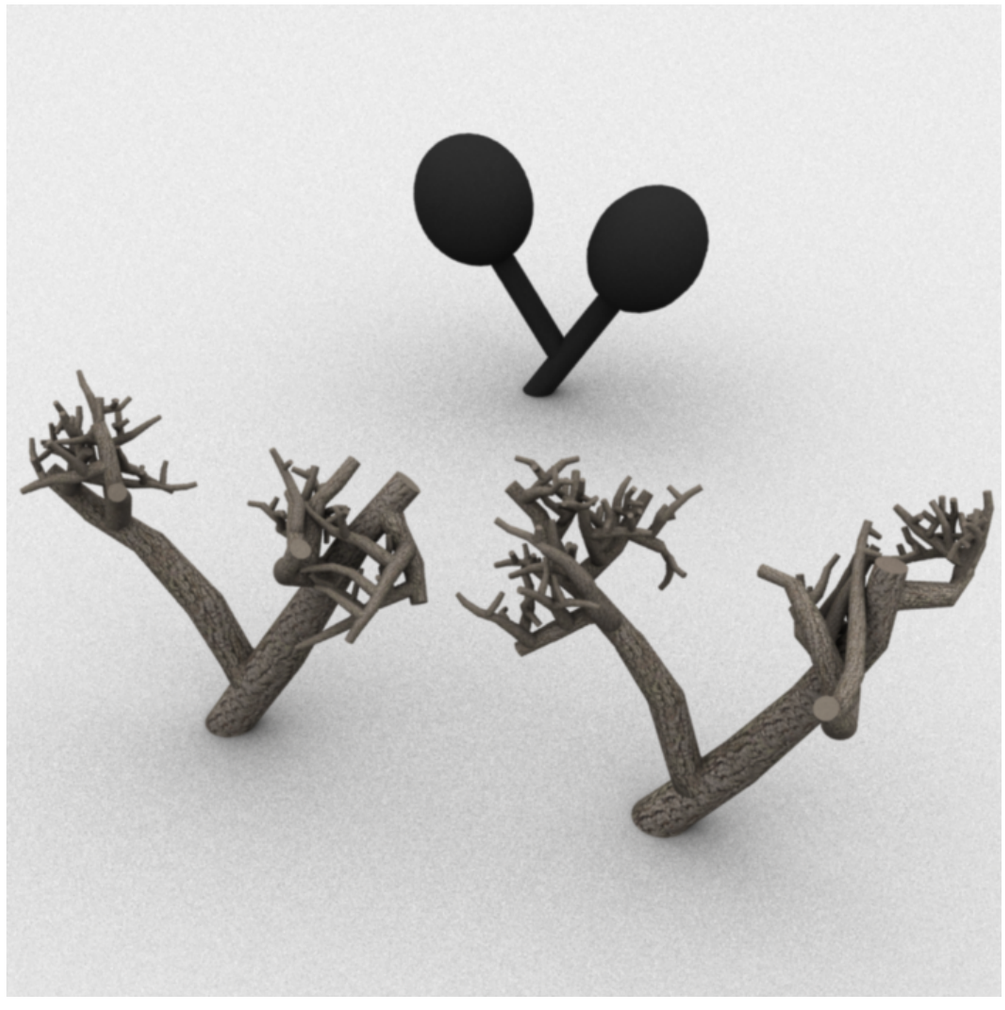
1. Sample a 3D model.
2. Generate an image.

posterior inference



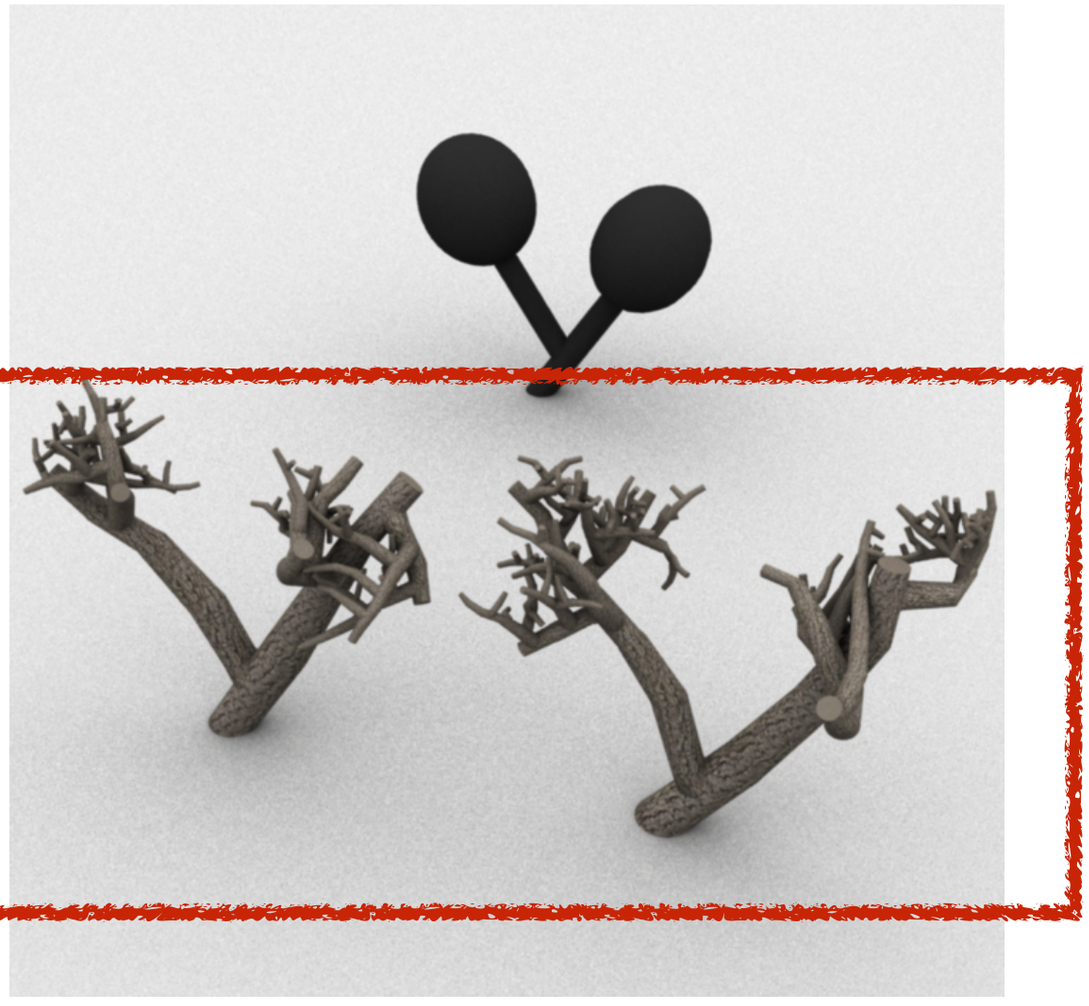
Kulkarni, Kohl, Tenenvaum, Mansinghka [CVPR'15]

Success story 3: Procedural modelling



Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

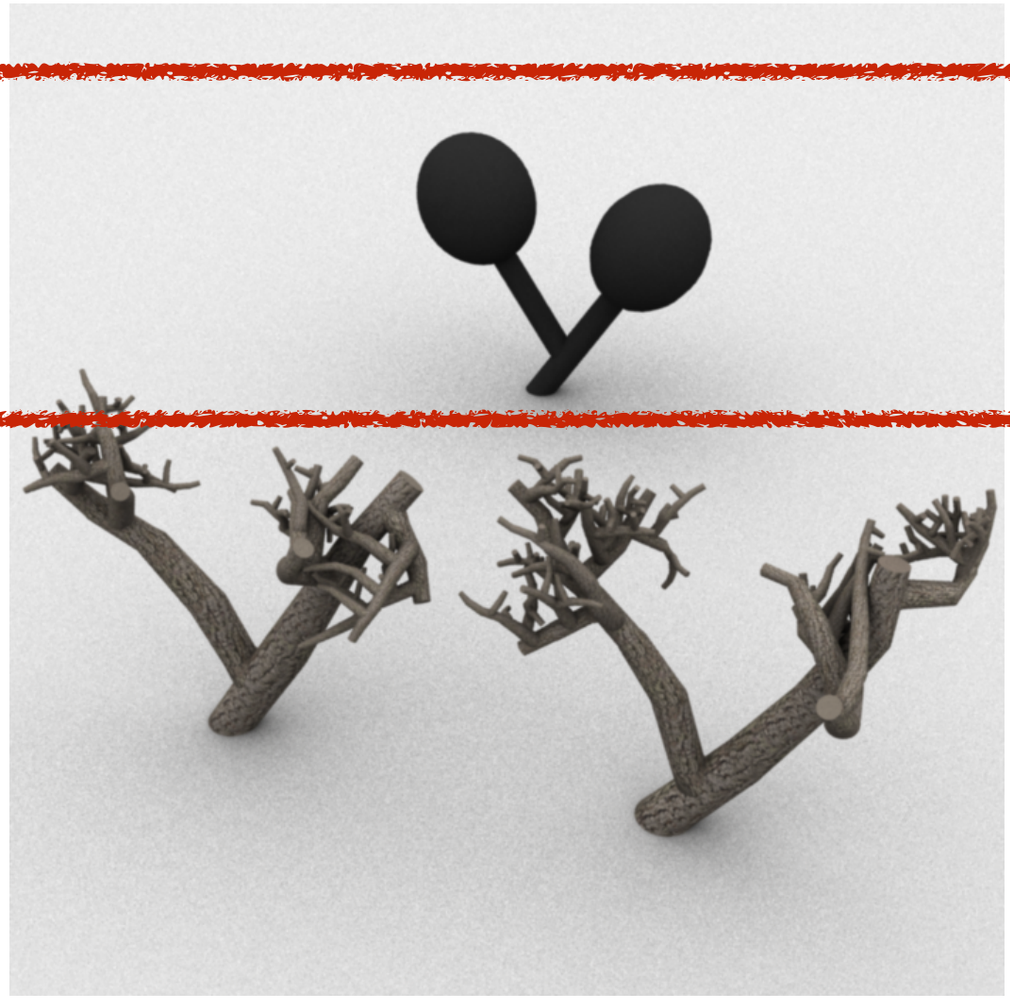
Success story 3: Procedural modelling



1. Sample a 3D object.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

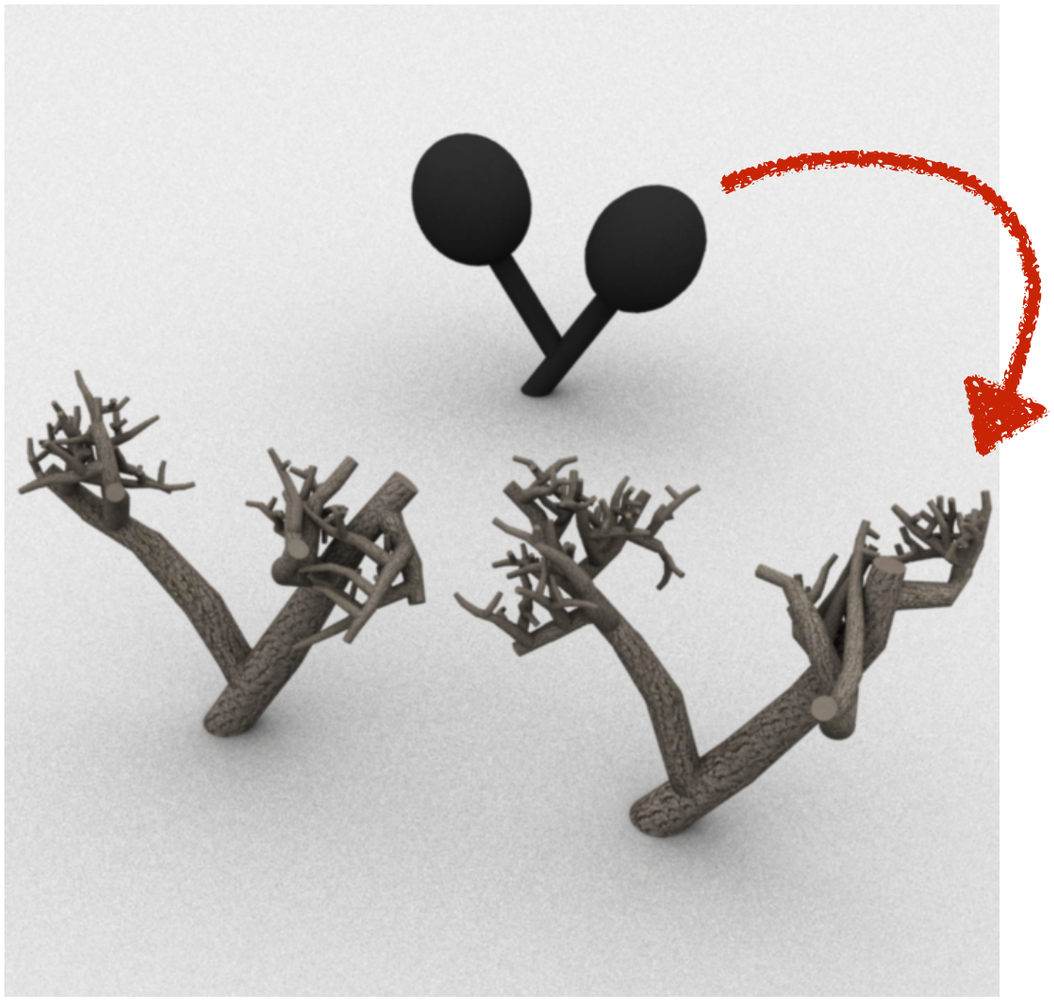
Success story 3: Procedural modelling



1. Sample a 3D object.
2. Score the object.

Ritchie, Mildenhall, Goodman, Hanrahan [SIGGRAPH'15]

Success story 3: Procedural modelling



1. Sample a 3D object.
2. Score the object.

Posterior inference generates objects with high scores.

Ritchie et al. [SIGGRAPH 15]

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Prog. Languages,

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Prog. Languages, Machine Learning,

Techniques used

- Changepoints: CPS transformation and new foundation of probability theory.
- Captcha and inverse graphics: neural nets and inference amortisation.
- Procedural modelling: sequential Monte Carlo algorithms and stochastic future.

Prog. Languages, Machine Learning, Probability Theory

Overview of the course

Objective

1. Learn how to write and reason about models in a prob. prog. language (PPL).
2. Learn results from ML/PL/prob. theory that are used for building effective PPLs.
3. Contribute to probabilistic programming.

Objective

1. Learn how to write and reason about models in a prob. prog. language (PPL).
2. Learn results from ML/PL/prob. theory that are used for building effective PPLs.
3. Contribute to probabilistic programming.

   Group project by a group of 2-3 students.

Webpage

https://github.com/hongseok-yang/probprog17

All the important announcements will be made in this webpage.

Evaluation

- 3-5 homework exercises (20%).
- Group project (40%).
- Final exam (40%).

Evaluation

- 3-5 homework exercises (20%).
- Group project (40%).
- Final exam (40%).

2-3 students form a group.

- Track A: New cool application of a PPL.
- Track B: Research on PPLs. Some topics suggested in the course webpage.

Tasks: i) project. ii) 3 meetings with me. iii) lecture on project and its outcome. iv) report.

Really important announcement

1. Form a group and inform me and Byungsoo by the midnight of Sep 11 (Monday).
2. Each group will meet me twice privately.
   (a) 7-9pm on Sep 21-22 : Topic selection.
   (b) 7-9pm on Oct 26-27 : Progress check.
   (c) 7-9pm on Nov 9-10 : Progress check.
3. No lectures on Sep 4 and 6.

Somewhat important announcement

- Lecturer: Prof Hongseok Yang

  email: hongseok.yang@kaist.ac.kr

- TA: Byungsoo Kim

  email: bskim90@kaist.ac.kr

- Install Anglican. Try the changepoint example. Get hints from the webpage.

Webpage

https://github.com/hongseok-yang/probprog17

All the important announcements will be made in this web page.