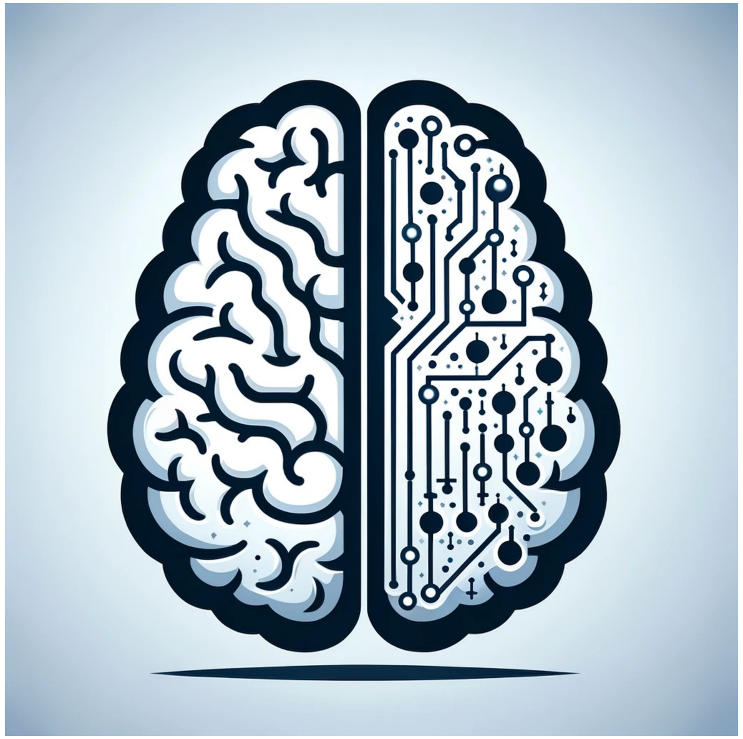
EN 601.473/601.673: Cognitive Artificial Intelligence (CogAI)



Lecture 9: importance sampling in Gen, MCMC

Tianmin Shu

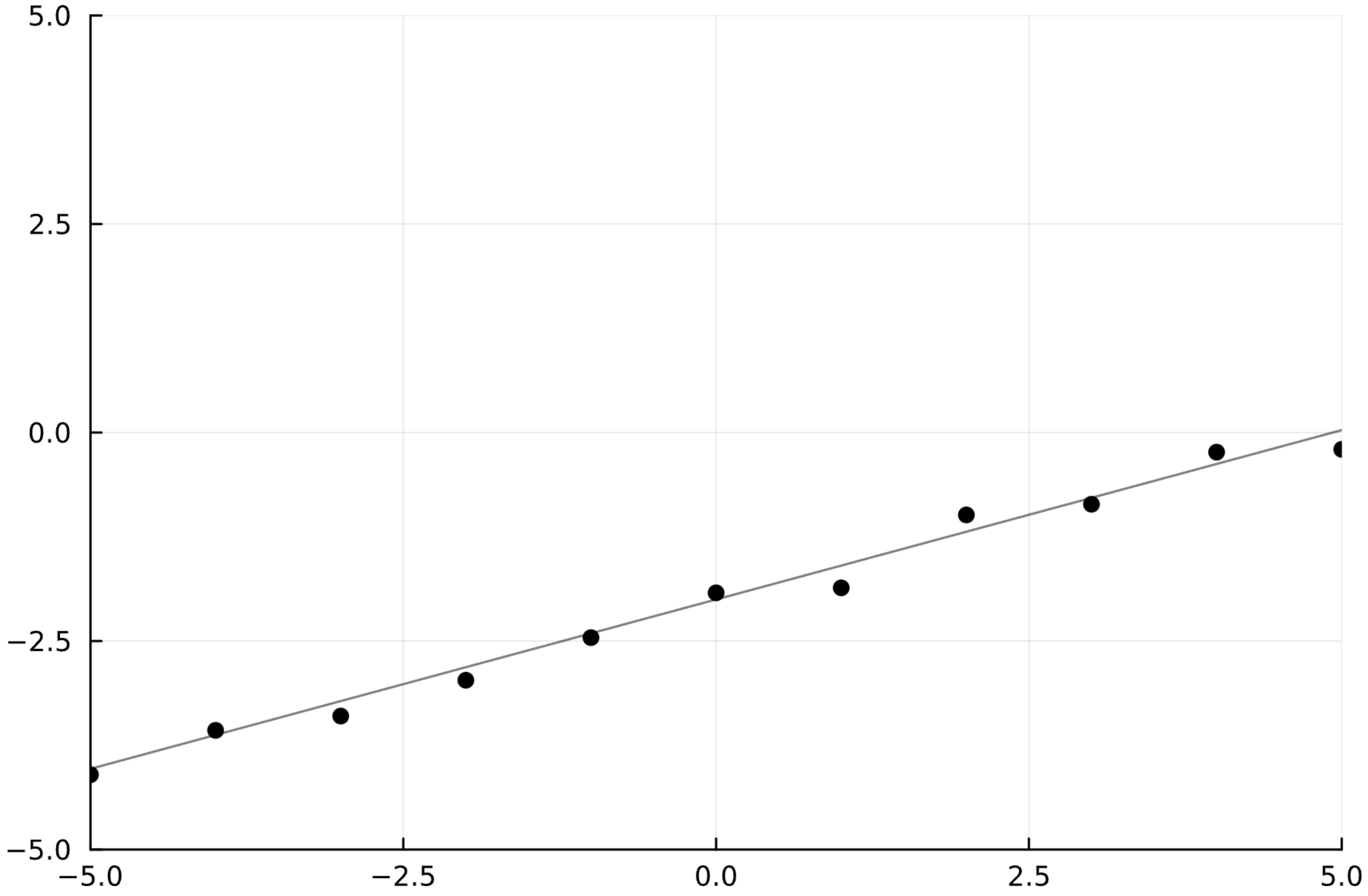
Recap

- Intro to PPL, WebPPL, Gen

Readings

- Jupyter notebooks
- Book chapters on intuitive physics and intuitive psychology

A simple example: Bayesian curve-fitting



Writing the generative model

```
slope = {:slope} ~ normal(0, 1)
```

```
# Desugars to "slope = {:slope} ~ normal(0, 1)"
slope ~ normal(0, 1)
```

```
for i=1:10
    y = {(:y, i)} ~ normal(0, 1) # OK: the address is different each time.
    println(y)
end
```

```
for i=1:10
    y ~ normal(0, 1) # The name :y will be used more than once!!
    println(y)
end
```

A simple example: Bayesian curve-fitting

See the jupyter notebook

Calling other generative functions

- **(NOT RECOMMENDED)** using regular Julia function call syntax: `f(x)`
- using the `~` snytax with an address for the call: `{addr} ~ f(x)`
- using the `~` syntax with a wildcard address: `{*} ~ f(x)`

```julia
@gen function foo()
    {:y} ~ normal(0, 1)
end
```

```julia
@gen function bar()
    {:x} ~ bernoulli(0.5)
    # Call `foo` with a wildcard address.
    # Its choices (:y) will appear directly
    # within the trace of `bar`.
    {*} ~ foo()
end
```

Conflict if foo() gets called by multiple functions

```
│
├── :y : -1.208648154503792
│
└── :x : false
```

```julia
@gen function bar_using_namespace()
    {:x} ~ bernoulli(0.5)
    # Call `foo` with the address `:z`.
    # The internal choice `:y` of `foo`
    # will appear in our trace at the
    # hierarchical address `:z => :y`.
    {:z} ~ foo()
end;
```

With a namespace z

```
│
├── :x : false
│
└── :z
    │
    └── :y : -0.2305260495498657
```

Hierarchical Address Spaces

```
│
├── :x : false
│
└── :z
    │
    └── :y : -0.2305260495498657
```

- How to get access to :y in this hierarchal address space
- option 1: pair
  - trace[Pair(:z, :y)]
- option 2: path
  - trace[:z => :y]

Hierarchical address spaces

```
@gen function bar_using_namespace()
    {:x} ~ bernoulli(0.5)
    # Call `foo` with the address `:z`.
    # The internal choice `:y` of `foo`
    # will appear in our trace at the
    # hierarchical address `:z => :y`.
    {:z} ~ foo()
end;
```

```
@gen function baz()
    {:a} ~ bar_using_namespace()
end
```

```
└── :a
    │
    ├── :x : false
    │
    └── :z
        │
        └── :y : 1.5440483281515782
```

trace[Pair(:a, Pair(:z, :y))]

trace[:a => :z => :y]

Sampling-based inference algorithms

- Monte Carlo methos:

- Rejection sampling
- Importance sampling
- Markov chain Monte Carlo (MCMC)
- Sequential Monte Carlo (SMC)

How to use Gen to implement sampling-based inference algorithms?

Markov chain Monte Carlo

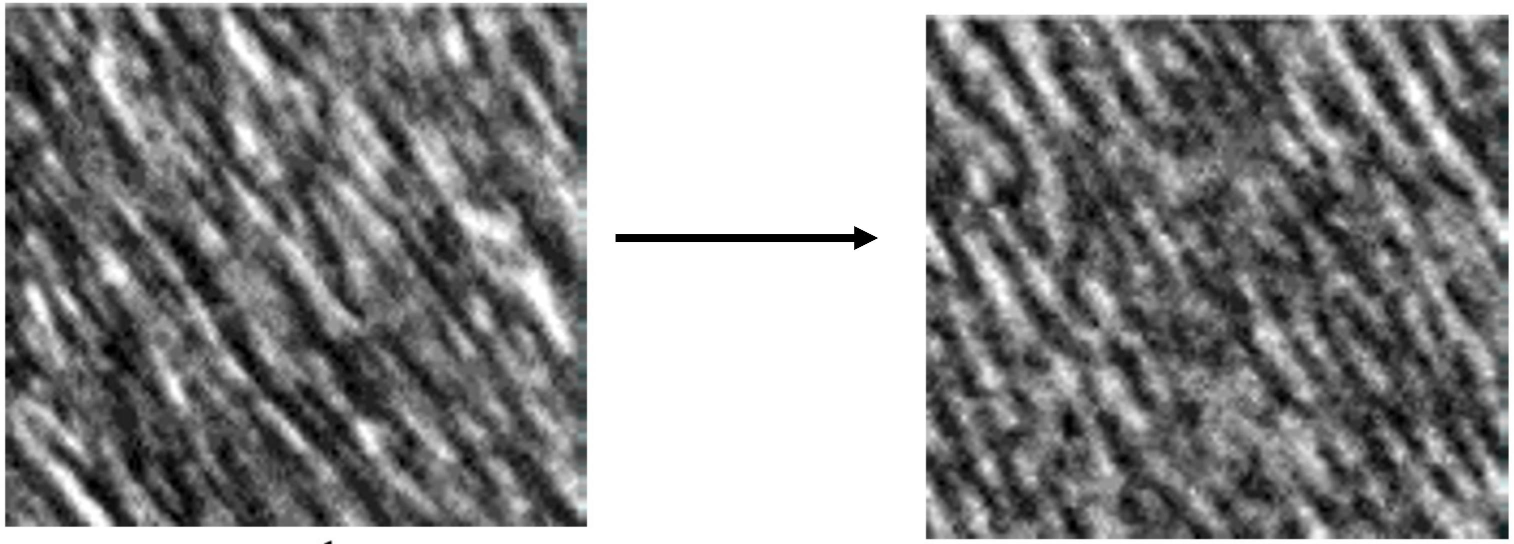
Top 10 algorithms, Computing in Science & Engineering, Feb 2000:

1. Metropolis Algorithm for Monte Carlo
2. The Decompositional Approach to Matrix Computations
3. The Simplex Method for Linear Programming
4. Quicksort Algorithm for Sorting
5. The FORTRAN Optimizing Compiler
6. Krylov Subspace Iteration Methods
7. Fast Fourier Transform
8. The QR Algorithm for Computing Eigenvalues
9. Integer Relation Detection
10. The Fast Multipole Method

One of the earliest use cases of MCMC in AI (Vision)

- Zhu et al. (1996)
- The first generative model of texture

MCMC samples a synthetic texture image that is visually the same as the observed texture image



I^{obs} $\qquad$ $I^{syn} \sim \Omega(h)$ k=7

Markov chain Monte Carlo

- Markov chains
- Gibbs sampling
- Metropolis-Hastings

Markov chain Monte Carlo

- Markov chains
- Gibbs sampling
- Metropolis-Hastings

Markov chain Monte Carlo

- Basic idea: construct a Markov chain that will converge to the target distribution (in our case, the posterior probability distribution), and draw samples from that chain.

- More ***efficient*** than rejection sampling or importance sampling, because it "constructs" high probability samples with a stochastic analog to a search process – it doesn't just have to hit on them luckily.

- Can work in state spaces of ***arbitrary structure and size***

- Can often be made more efficient by exploiting model structure

Markov chain Monte Carlo

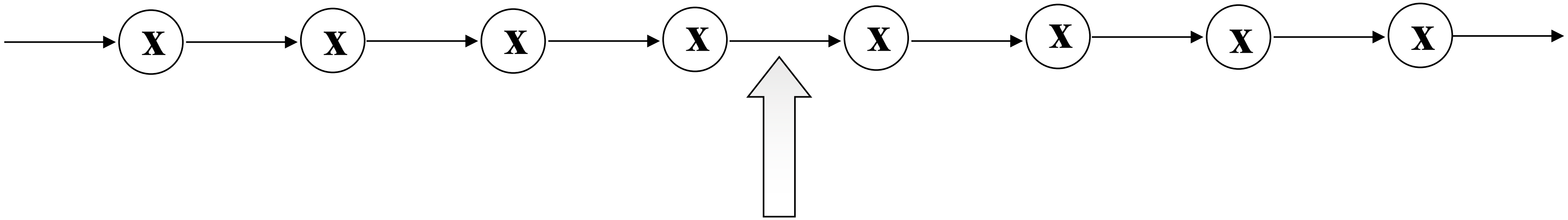
- After initial “burn in” period, samples are independent of starting conditions (though we will talk about how to get a better starting condition via data-driven approaches)

- Waiting sufficiently long between subsequent samples (“skip”) will generate a sequence of independent posterior samples, h ~ p(h|d)

- Nowadays, mostly automated by probabilistic programming languages
  - Gen allows customization over standard, built-in MCMC algorithms

Markov chain

- A random process that generate a sequence of states



Transition kernel

$k(\boldsymbol{x} \rightarrow \boldsymbol{x}')$

A special case: a finite discrete state space $x \in \{1, \cdots M\}$

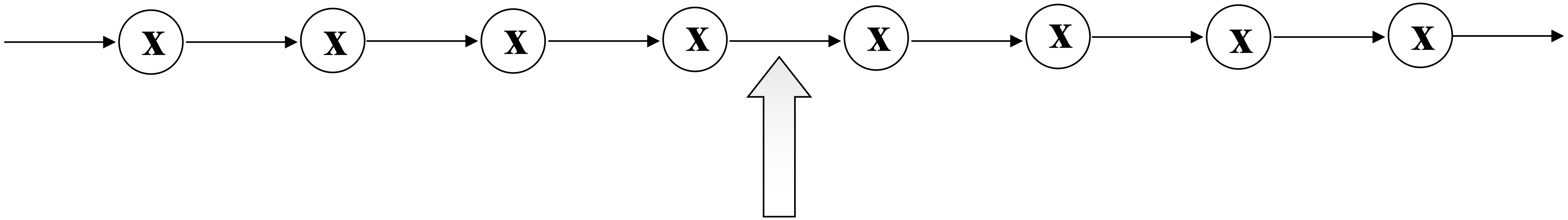
Transition matrix

$$\mathrm{K} = \begin{bmatrix} p_{11} & \cdots & p_{1M} \\ \vdots & \ddots & \vdots \\ p_{M1} & \cdots & P_{MM} \end{bmatrix}$$

p_{ij}: probability of a transition to state j starting from state i

Markov chain

- A random process that generate a sequence of states



Transition kernel

$k(\boldsymbol{x} \to \boldsymbol{x}')$

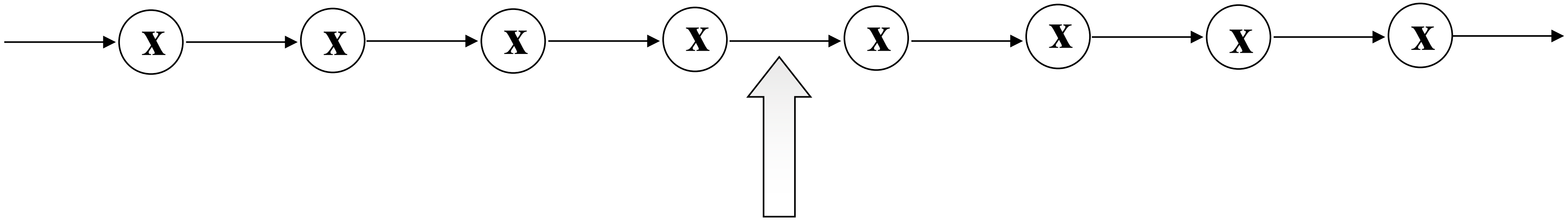
Variables $\boldsymbol{x}'$ ***independent*** of all previous variables given immediate predecessor $\boldsymbol{x}$.

The probability that the system is in $\boldsymbol{x}$ at time t: $\pi_t(\boldsymbol{x})$.

$$\pi_{t+1}(\boldsymbol{x}') = \sum_{\boldsymbol{x}} \pi_t(\boldsymbol{x}) k(\boldsymbol{x} \to \boldsymbol{x}')$$

Markov chain

- A random process that generate a sequence of states



Transition kernel

$k(\boldsymbol{x} \to \boldsymbol{x}')$

The chain has reached its stationary distribution if $\pi_t = \pi_{t+1}$

The defining equation of stationary distribution

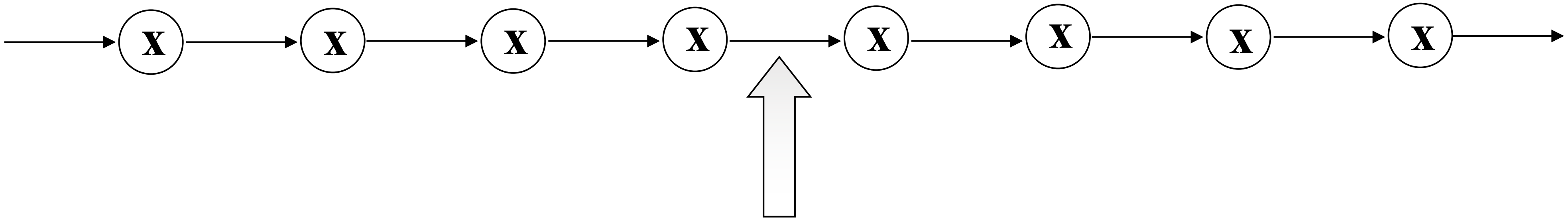
$$\pi(\boldsymbol{x}') = \sum_{\boldsymbol{x}} \pi(\boldsymbol{x}) k(\boldsymbol{x} \to \boldsymbol{x}')$$

If k is **ergodic**

- **Connectivity:** every state is reachable from every other state π is **unique**
- **Aperiodicity:** there are no strictly periodic cycles

Markov chain

- A random process that generate a sequence of states



Transition kernel

$$k(\boldsymbol{x} \to \boldsymbol{x}')$$

The chain has reached its stationary distribution if $\pi_t = \pi_{t+1}$

The defining equation of stationary distribution

$$\pi(\boldsymbol{x}') = \sum_{\boldsymbol{x}} \pi(\boldsymbol{x}) k(\boldsymbol{x} \to \boldsymbol{x}')$$

Detailed balance:

$$\pi(\boldsymbol{x}) k(\boldsymbol{x} \to \boldsymbol{x}') = \pi(\boldsymbol{x}') k(\boldsymbol{x}' \to \boldsymbol{x})$$

A simple example: weather

- Consider a simple weather model with two states: {rainy, sunny}.

- If it’s rainy today, there is a **50%** chance it will be rainy the next day, and a **50%** chance it will be sunny the next day.

- If it’s sunny today, there is a **20%** chance it will be rainy the next day, and an **80%** chance it will be sunny the next day.

A simple example: weather

- Given that it is rainy today, how many days do I expect to wait to see a sunny day?
- Given that it is sunny today, how many days do I expect to wait to see a rainy day?
- Over the long haul, what fraction of days are sunny?

Markov chain

- 2 states, rainy = 0, sunny = 1

$$K = \begin{pmatrix} .5 & .5 \\ .2 & .8 \end{pmatrix}$$

- Ergodic
- Initial state distribution $\pi_0 : (p_0, p_1)$
- State distribution at time 1

$$\pi_1 = \pi_0 K = (p_0, p_1) \begin{pmatrix} .5 & .5 \\ .2 & .8 \end{pmatrix}$$

- State distribution at time 2

$$\pi_2 = \pi_0 K^2 = (p_0, p_1) \begin{pmatrix} .35 & .65 \\ .26 & .74 \end{pmatrix}$$

- State distribution at time n

$$\pi_n = \pi_0 K^n$$

Markov chain

- 2 states, rainy = 0, sunny = 1

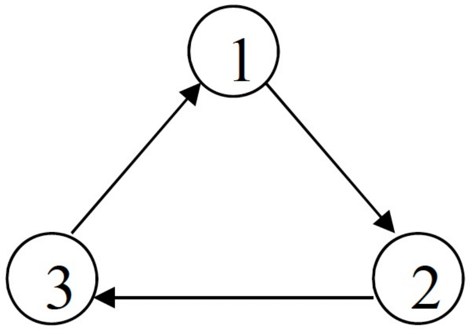
$$K = \begin{pmatrix} .5 & .5 \\ .2 & .8 \end{pmatrix}$$

- Ergodic
- Stationary state distribution $\pi = \lim_{n \to \infty} \pi_0 K^n$
  - For any starting condition π_0, it always converge to a unique stationary distribution

$$\pi = \left(\frac{2}{7}, \frac{5}{7}\right)$$

$$\pi = \pi K = \left(\frac{2}{7}, \frac{5}{7}\right) = \left(\frac{2}{7}, \frac{5}{7}\right) \begin{pmatrix} .5 & .5 \\ .2 & .8 \end{pmatrix}$$

Periodic Markov chain does not converge



$$K = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Markov chain may not always converge to a unique stationary distribution

$$\pi = \left(\tfrac{1}{3} \;\; \tfrac{1}{3} \;\; \tfrac{1}{3}\right) \quad K = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad \left(\frac{1}{3}\,\frac{1}{3}\,\frac{1}{3}\right)\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \left(\frac{1}{3}\,\frac{1}{3}\,\frac{1}{3}\right)$$

$$\begin{tikzcd} (1\;\;0\;\;0) \arrow[r] & (0\;\;1\;\;0) \arrow[r] & (0\;\;0\;\;1) \arrow[ll, bend left] \end{tikzcd} \qquad (1\;0\;0)\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = (0\;1\;0)$$

Detailed balance: $\pi(\boldsymbol{x})k(\boldsymbol{x} \rightarrow \boldsymbol{x}') = \pi(\boldsymbol{x}')k(\boldsymbol{x}' \rightarrow \boldsymbol{x})$

Markov chain Monte Carlo

- Markov chains
- Gibbs sampling
- Metropolis-Hastings

Gibbs sampling

- Directly sample from a joint probability $P(x_1, x_2, \cdots, x_n)$ is hard
- Much easier to sample from the conditional probability:

$$x_i \sim P(x_i | x_{-i}), \forall i = 1, \cdots, n$$

$$x_{-i}: excluding\ x_i$$

- Gibbs sampling:
- Start from a random value for each variable
- One **step** in Gibbs sampling: sampling one variable

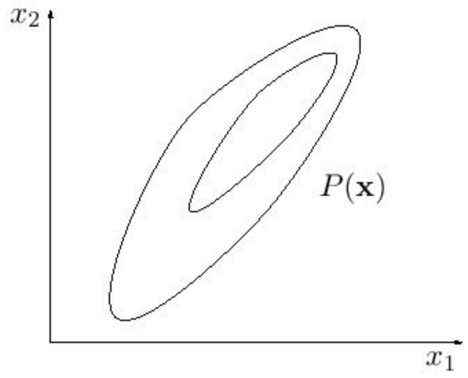
$$(x_1, \cdots, x_i, \cdots, x_n) \rightarrow (x_1, \cdots, x_i{}', \cdots, x_n)$$

- A **sweep** in Gibbs sampling: sampling every variable once

Example: Sampling from a 2D Gaussian distribution

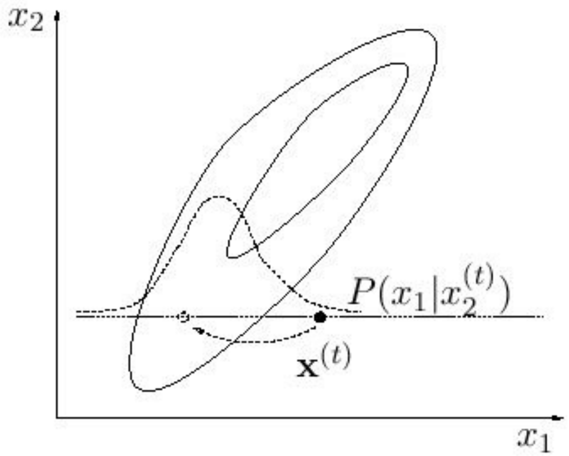
- $(x_1, x_2) \sim N(0, \Sigma), \Sigma = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$
- Conditional distributions:

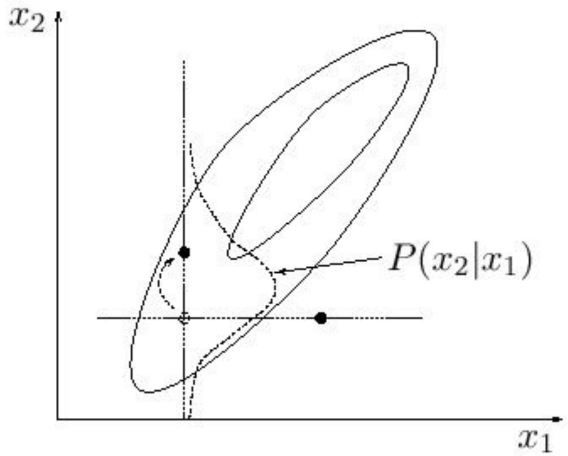
$$\mathrm{x}_1 | x_2 \sim N(\rho x_2, 1 - \rho^2)$$
$$\mathrm{x}_2 | x_1 \sim N(\rho x_1, 1 - \rho^2)$$



- Initial value (x_1^0, x_2^0)

Sweep 1 $\quad \mathrm{x}_1^1 \sim N(\rho x_2^0, 1 - \rho^2) \quad \mathrm{x}_2^1 \sim N(\rho x_1^1, 1 - \rho^2)$



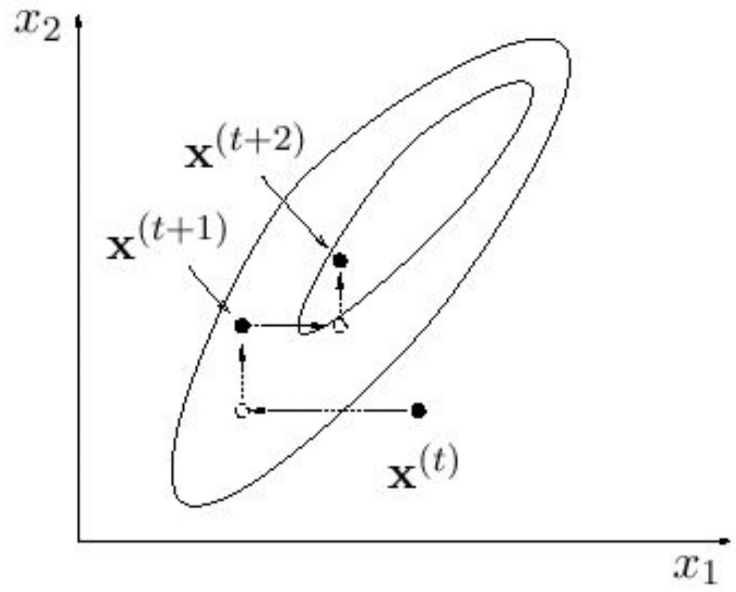


Example: Sampling from a 2D Gaussian distribution

- $(x_1, x_2) \sim N(0, \Sigma), \Sigma = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$
- Conditional distributions:

$$\mathrm{x}_1 | x_2 \sim N(\rho x_2, 1 - \rho^2)$$
$$\mathrm{x}_2 | x_1 \sim N(\rho x_1, 1 - \rho^2)$$

- Initial value (x_1^0, x_2^0)



Sweep 1	$\mathrm{x}_1^1 \sim N(\rho x_2^0, 1 - \rho^2)$	$\mathrm{x}_2^1 \sim N(\rho x_1^1, 1 - \rho^2)$
	$\vdots$	$\vdots$
Sweep t	$\mathrm{x}_1^t \sim N(\rho x_2^{t-1}, 1 - \rho^2)$	$\mathrm{x}_2^t \sim N(\rho x_1^t, 1 - \rho^2)$

Gibbs sampling for Bayesian inference

$$p(h|d = D) = \frac{p(D|h)p(h)}{Z}$$

$$p\left(h_i \middle| h_{-i} = H^t_{-i}, d = D\right) = \frac{p\left(D \middle| h_i, h_{-i} = H^t_{-i}\right)p(h_i, h_{-i} = H^t_{-i})}{Z}$$

$$= \frac{p\left(D \middle| h_i, h_{-i} = H^t_{-i}\right)p(h_i, h_{-i} = H^t_{-i})}{\sum_{h_i} p\left(D \middle| h_i, h_{-i} = H^t_{-i}\right)p(h_i, h_{-i} = H^t_{-i})}$$

Only need to enumerate all possible values of one variable

A major problem with Gibbs sampling

- For a joint probability whose probability mass is focused on a 1D line segment, sampling two 1D variables iteratively is inefficient, i.e., the chain is “jagging”

This is because the two variables are ***tightly coupled***. It is best if we move along the direction of the line.