Advanced topics in Markov-chain Monte Carlo

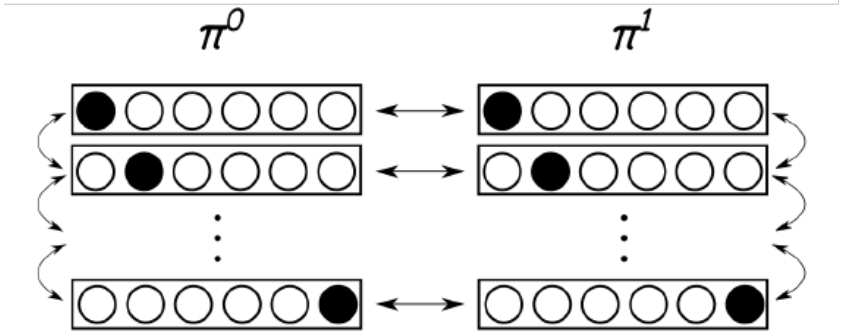
Lecture 8:
Meta algorithms, consensus sampling
Part 1/2: Simulated tempering (practical aspects)

Werner Krauth

ICFP -Master Course Ecole Normale Supérieure, Paris, France

16 March 2022

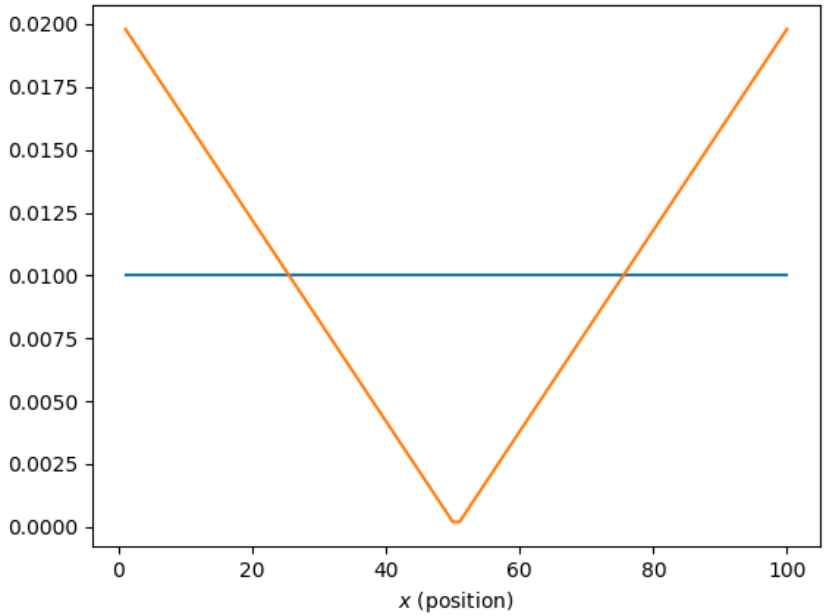
Simulated tempering 1/7



procedure temp-rev
input $\mathcal{P}^{\text{padded}}, \{x, \alpha\}, p_{\text{copy}}$
if $(\texttt{ran}\,(0,1) > p_{\text{copy}})$ **then**
$\left\{ \begin{array}{l} \sigma_x \leftarrow \texttt{choice}\{-1,1\} \\ \textbf{if } (\texttt{ran}\,(0,1) < \pi^{\alpha}_{x+\sigma_x}/\pi^{\alpha}_{x}) \textbf{ then} \\ \quad \{\ x \leftarrow x + \sigma_x \end{array} \right.$
else
$\left\{ \begin{array}{l} \sigma_\alpha \leftarrow \texttt{choice}\{-1,1\} \\ \textbf{if } (\texttt{ran}\,(0,1) < \pi^{\alpha+\sigma_\alpha}_{x}/\pi^{\alpha}_{x}) \textbf{ then} \\ \quad \{\ \alpha \leftarrow \alpha + \sigma_\alpha \end{array} \right.$
output $\{x, \alpha\}$

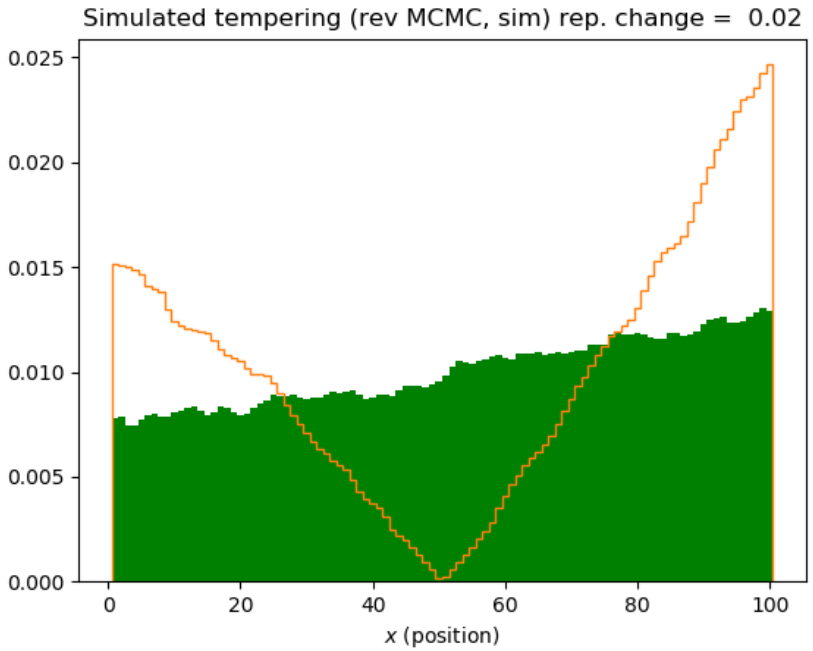
- Let the system evolve at several temperatures.
- Move between temperatures, move between positions.

Simulated tempering 2/7



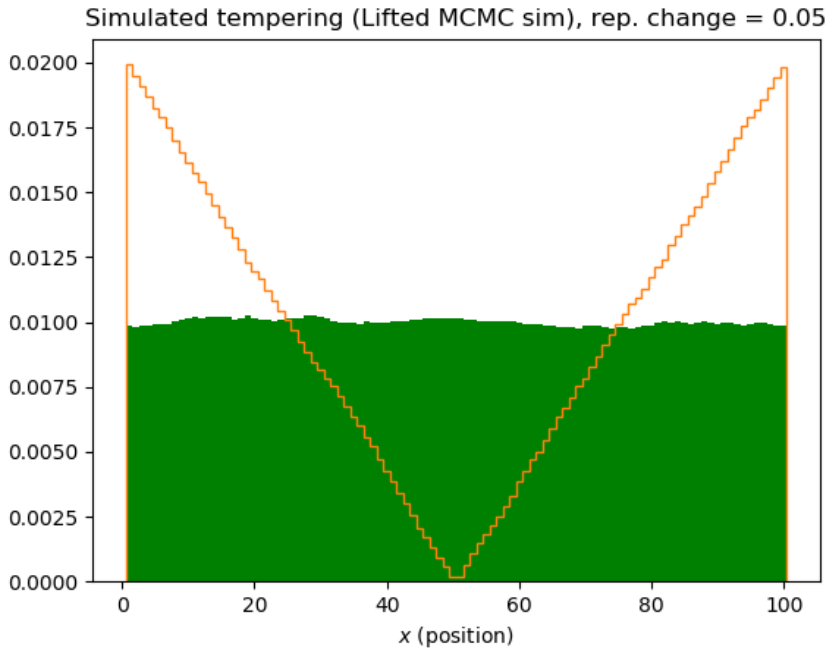
- $\alpha = 0$ and $\alpha = 1$ with transitions between the two.

Simulated tempering 3/7



- Replica index $\alpha = 0, 1$
- Ω is $2n$-dimensional

Simulated tempering 4/7



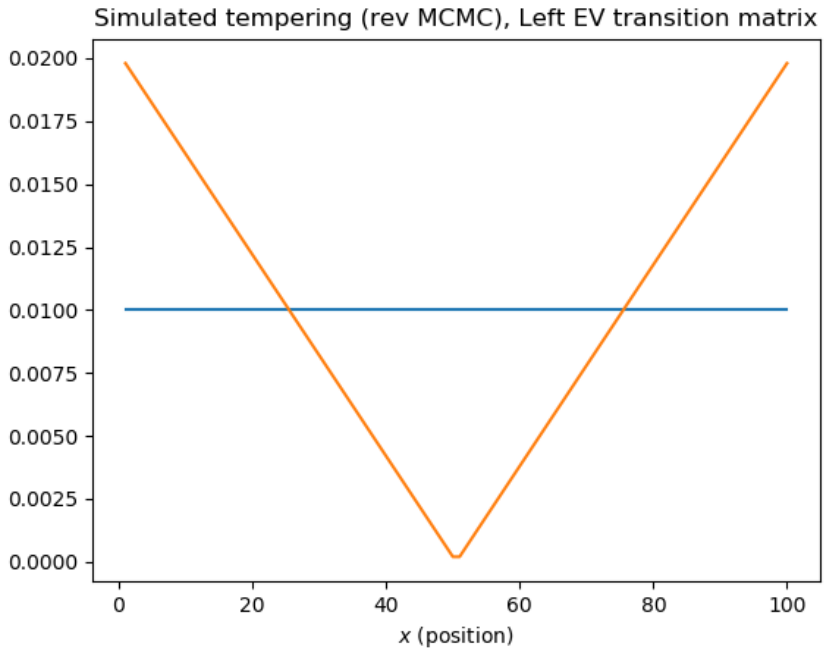
- Replica index $\alpha = 0, 1$
- Direction index dir $= 0, 1$
- Ω is $4n$-dimensional

Simulated tempering 5/7

```
#
# Examples of matrix manipulation in numpy
#
import numpy as np
a = np.array([(1.5, 2), (4, 5)], dtype = float)
b = np.array([(1, 2), (2, 4)], dtype = float)
c = np.matmul(a, b)
c = a @ b
d = np.zeros((3, 4))
e = np.eye(3)
print('a square matrix')
print(a)
print('b square matrix')
print(b)
print('c = a * b')
print(c)
print('d zero matrix')
print(d)
print('e diagonal matrix, using eye')
print(e)
```

- Matrix multiplication in numpy

Simulated tempering 6/7



- Dominant $\lambda = 1$-eigenvalue combines two sectors.
- Same result for lifted model.

Simulated tempering 7/7

```
def OutsideFlow(S):
    Flow = 0.0
    Weight = 0.0
    for i in S:
        Weight += PiStat[Table[i]]
        for j in range(2 * n):
            if j not in S:
                Flow += PiStat[Table[i]] * P[i, j]
    print(Weight, 'Weight')
    return Weight, Flow / Weight
```

- Flow computation.
- Conductance $1/(2n)$ (conjecture).
- NB: Conductance of Hildebrand model $1/n^2$.

Computing conductances 1/3

```
def OutsideFlow(S):
    Flow = 0.0
    Weight = 0.0
    for i in S:
        Weight += PiStat[Table[i]]
        for j in range(2 * n):
            if j not in S:
                Flow += PiStat[Table[i]] * P[i, j]
    print(Weight, 'Weight')
    return Weight, Flow / Weight
```

- Conductance:

$$\Phi = \min_{S \subset \Omega, \pi(S) \leq \frac{1}{2}} \frac{1}{\pi(S)} \sum_{i \in S, j \notin S} \mathcal{F}_{ij}$$

- Powerset:
  $S \subset \Omega$ is an element of the powerset of Ω, the set of all its subsets.

Computing conductances 2/3

- Combinations:

```
from itertools import combinations
s = [1,2,3,4]
for x in combinations(s,2): print(x)
```

- Powerset:

```
from itertools import chain, combinations
def powerset(iterable): # without empty set
    s = list(iterable)
    print(s, 's')
    return chain.from_iterable(combinations(s, r) for r in range(1, len(s) + 1))

x = powerset([i for i in range(1, 21)])
for i in x:
    print(list(i))
```

Computing conductances 3/3

- Conductance - Enumeration

```
x = powerset([i for i in range(2 * n)])
MinFlow = 100.00
for i in x:
    S = set(list(i))
    Weight, Flow =  OutsideFlow(S)
    if Weight <= 0.5:
        if Flow < MinFlow:
            MinFlow = Flow
            MinS = S
print(MinS, MinFlow, '1 / 2n + 1 / n^2 = ', 1.0 / (2.0 * n) + 1.0 / (n **
```

- Results (for sufficient inter-replica rates)

$$\Phi = \frac{1}{2n} + \frac{1}{n^2} \tag{1}$$

- ... can be proven analytically

Computing TVDs with numpy 1/2

- Computing the TVD for the transition matrix:

```
P = PTrans @ PReplica
PiT = np.zeros((2*n), dtype = float)
PiT[0] = 1.0

iter = 0
while True:
        iter += 1
        PiT = PiT @ P
        TVD = (np.absolute(PiT - Stat)).sum() / 2.0
#        print(iter, TVD)
        if TVD < 0.25:
            print(iter / n ** 2, TVD)
            break
```

- Results (reversible simulated tempering)

$$t_{\text{mix}} \sim \mathcal{O}\left(n^2\right) \tag{2}$$

Computing TVDs with numpy 2/2

- Computing the TVD for the transition matrix:

```
P = PTrans @ PReplica @ PLift
PiT = np.zeros((4*n), dtype = float)
PiT[0] = 1.0

iter = 0
while True:
        iter += 1
        PiT = PiT @ P
        TVD = (np.absolute(PiT - Stat)).sum() / 2.0
        if TVD < 0.25:
            print(iter / n,  TVD)
            break
```

- Results (lifted simulated tempering)

$$t_{\text{mix}} \sim \mathcal{O}\left(n \log\left(n\right)\right) \tag{3}$$