# Advanced topics in Markov-chain Monte Carlo

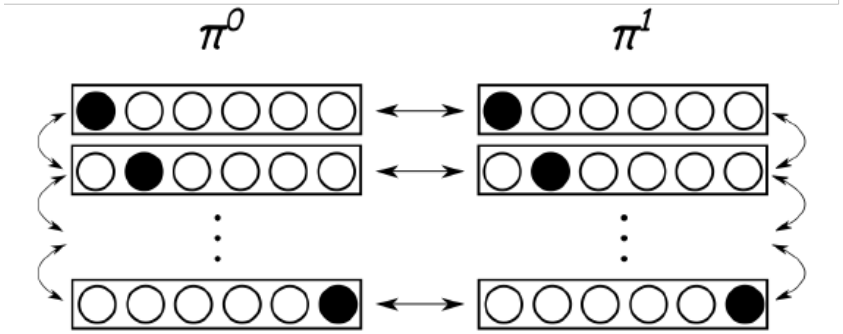
## Lecture 7:
## Meta algorithms (a practical approach)
## Part 2/2: Simulated tempering

Werner Krauth

ICFP -Master Course Ecole Normale Supérieure, Paris, France

9 March 2022

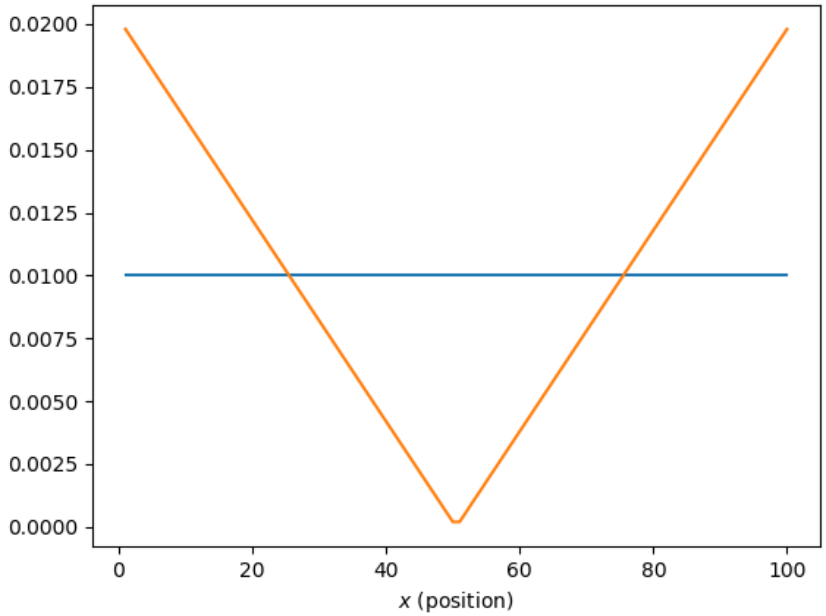
# Simulated tempering 1/7




**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** $\{x, \alpha\}$

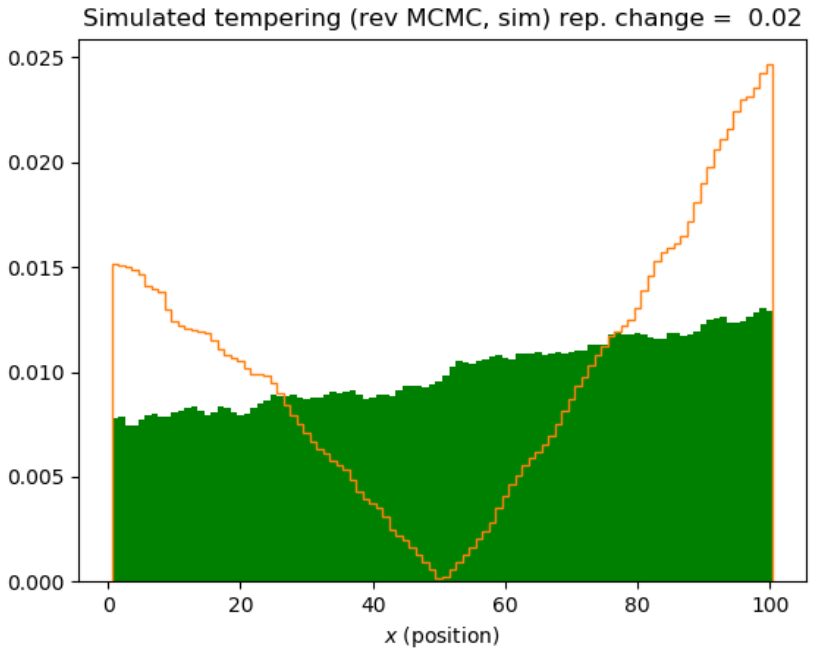
- Let the system evolve at several temperatures.
- Move between temperatures, move between positions.

# Simulated tempering 2/7



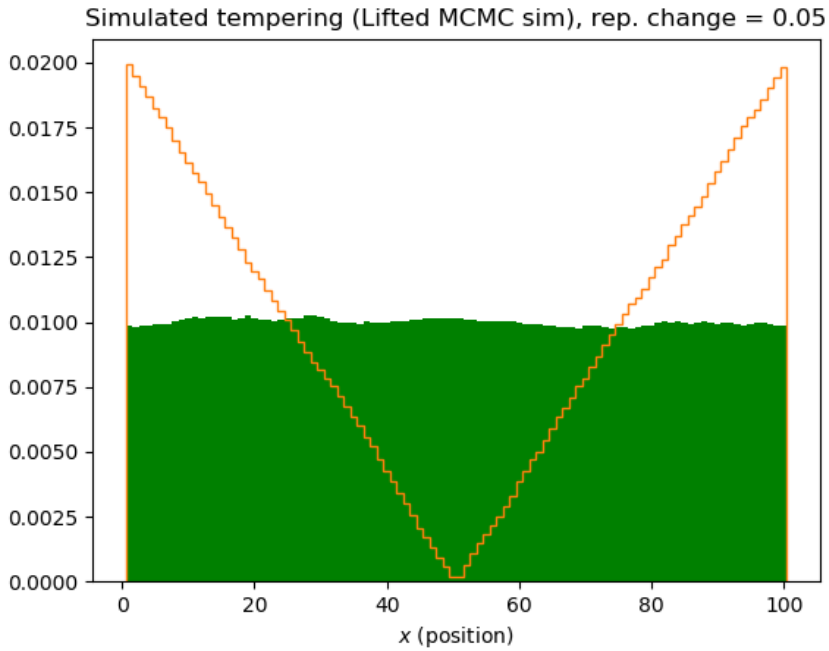

- $\alpha = 0$ and $\alpha = 1$ with transitions between the two.

# Simulated tempering 3/7




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

# Simulated tempering 4/7



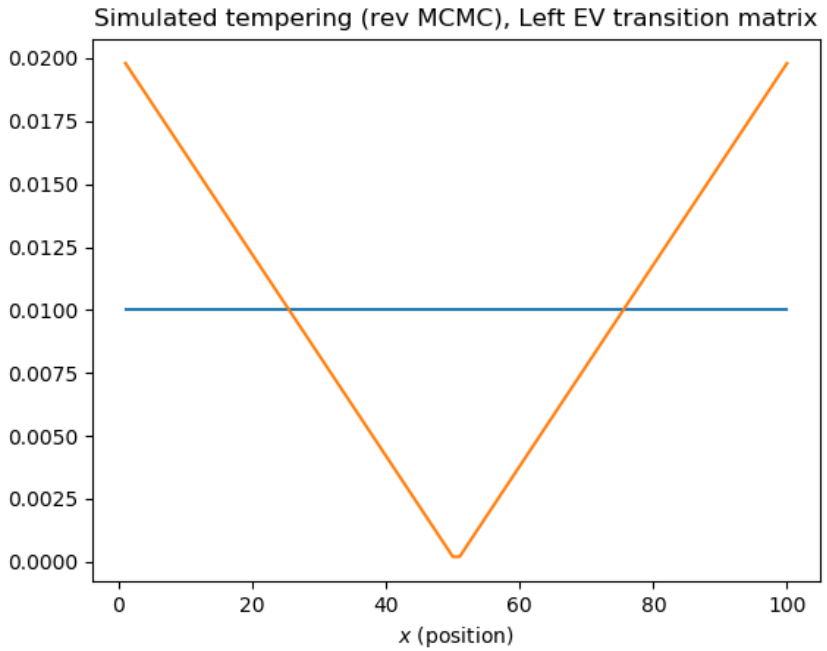

- Replica index $\alpha = 0, 1$
- Direction index dir $= 0, 1$
- $\Omega$ is $4n$-dimensional

# 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 $1/n^2$.