

MCMC

October 12, 2026

1 Markov Chain Monte Carlo examples

Our goal is to simulate a normal distribution, where

$$P(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

We'll do this with a variety of Markov chain Monte Carlo techniques to see how it all works.

```
[1]: import numpy as np
import scipy.special as sp
import matplotlib.pyplot as plt
%matplotlib inline
rng = np.random.default_rng()
from numba import jit
```

```
[2]: def Pnorm(x, mu=0.0, sigma2=1.0):
    return np.exp(-0.5*(x-mu)**2/sigma2)/np.sqrt(2*np.pi*sigma2)
```

```
[3]: @jit(nopython=True)
def MC_norm(x0, xstates, delta, log_u, mu, sigma2):
    """Does MCMC, starting from x0 for N(mu, sigma2), storing in xstates
    where delta is the random (symmetric) displacements, and log_u is
    -np.log(u) for u from a uniform distribution"""
    Nstates = len(xstates)
    x = x0
    accept = 0
    for i in range(Nstates):
        d = delta[i]
        xp = x + d
        dE = 0.5*(d*d + 2*(x-mu)*d)
        if sigma2*log_u[i] > dE:
            x = xp
            accept += 1
        xstates[i] = x
    return accept/Nstates
```

```
[4]: mu, sigma2 = 1.0, 1.0
```

```

x0 = 0.0
Nstates = 2**16
Delta = 1.0

states = np.zeros(Nstates)
delta = rng.uniform(low=-Delta, high=Delta, size=Nstates)
log_u = -np.log(rng.uniform(size=Nstates))
accept = MC_norm(x0, states, delta, log_u, mu, sigma2)

```

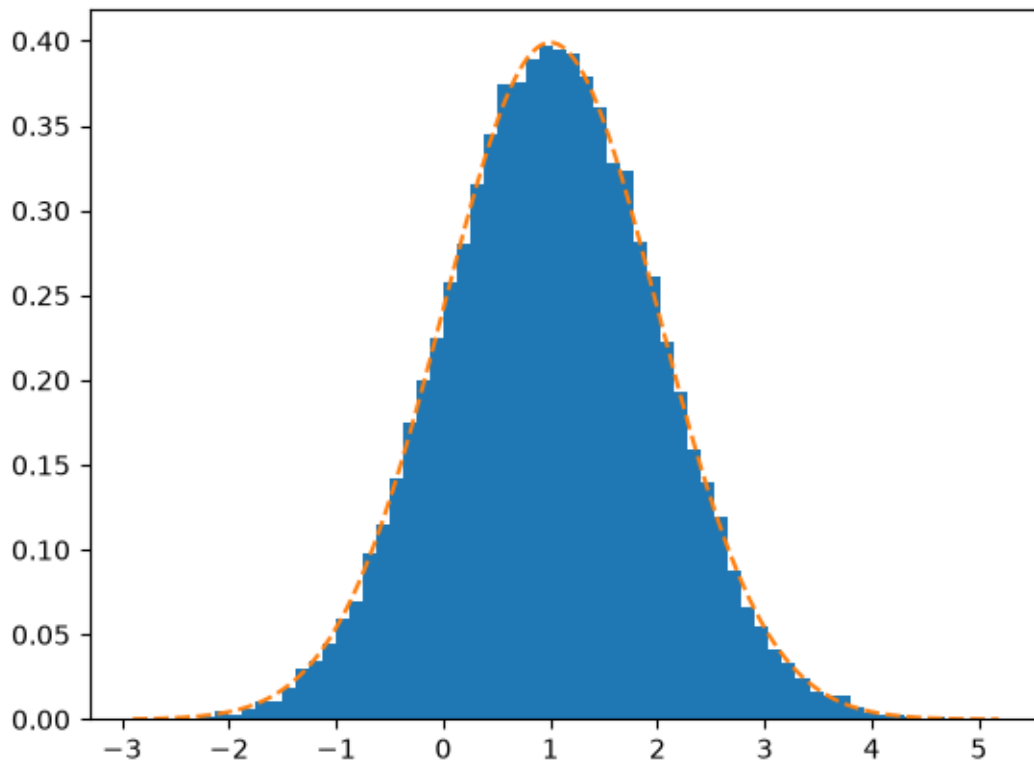
```
[5]: accept
```

```
[5]: 0.8032379150390625
```

```

[6]: plt.hist(states, bins=Nstates//1024, density=True);
x = np.sort(states)
plt.plot(x, Pnorm(x, mu, sigma2), '--');

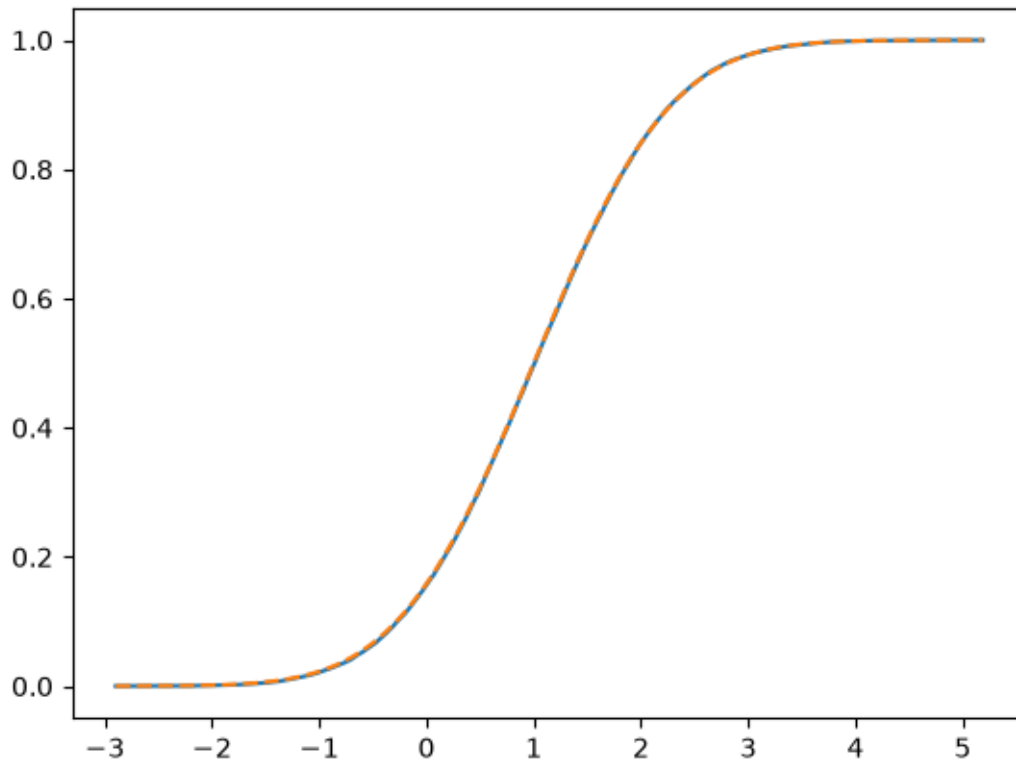
```



```

[7]: x = np.sort(states)
y = np.linspace(0, 1, len(x))
plt.plot(x, y);
plt.plot(x, 0.5*(sp.erf((x-mu)/np.sqrt(2*sigma2))+1), '--');

```

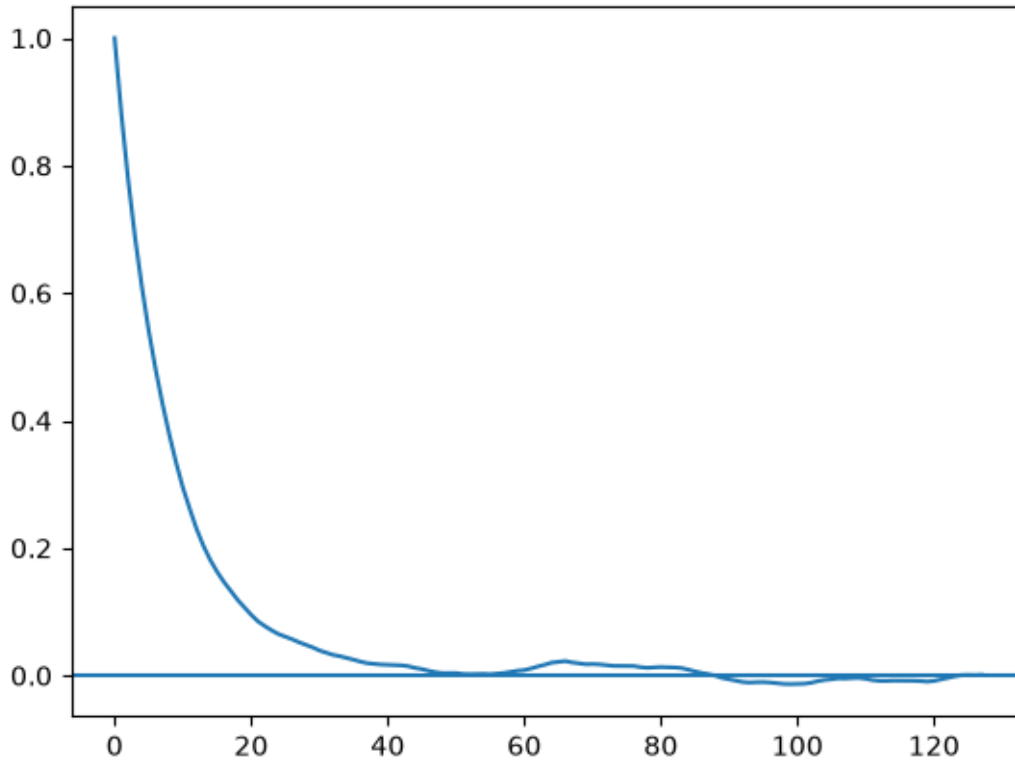


```
[8]: np.mean(states), np.var(states)
```

```
[8]: (np.float64(1.0090350825527041), np.float64(0.9893338480608971))
```

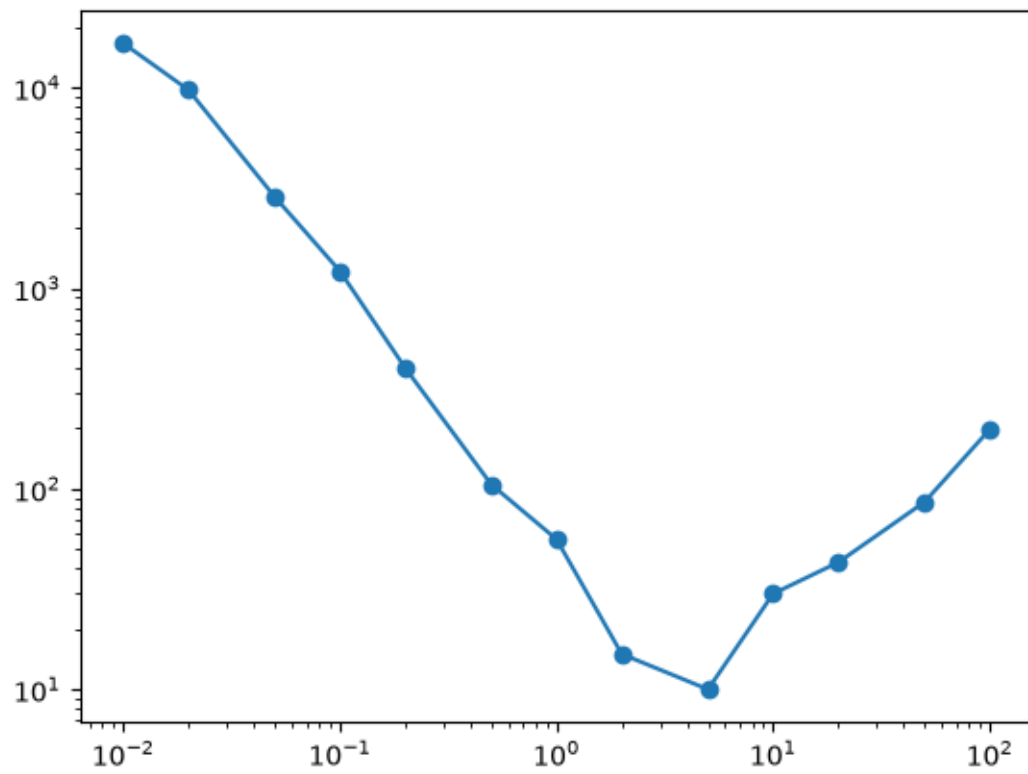
```
[9]: def acf_calc(v):
      """Returns the auto-correlation function for the sequence v"""
      v_ave = np.mean(v)
      v_len = len(v)
      return np.correlate(v-v_ave, v-v_ave, "full")[(v_len-1):] / \
         (np.var(v)*np.array([v_len - x for x in range(v_len)]))
```

```
[10]: plt.plot(acf_calc(states)[:128]);
      plt.axhline(0.0);
```

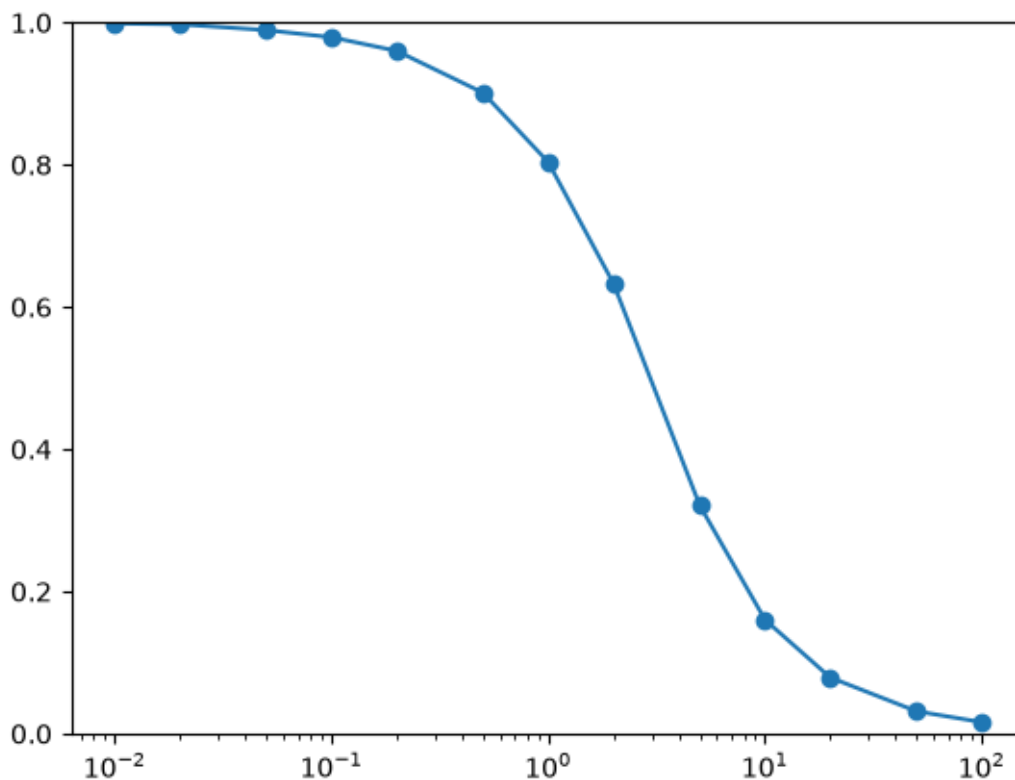


```
[11]: Delta_list = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0, 100.0]
autocorr_list = []
accept_list = []
for Delta in Delta_list:
    states = np.zeros(Nstates)
    delta = rng.uniform(low=-Delta, high=Delta, size=Nstates)
    log_u = -np.log(rng.uniform(size=Nstates))
    accept = MC_norm(x0, states, delta, log_u, mu, sigma2)
    autocorr = acf_calc(states)
    t_min, t_zero = 0, 0
    for t in range(len(autocorr)-1):
        if autocorr[t] < 0.0 and t_zero == 0:
            t_zero = t
        if autocorr[t] < autocorr[t+1] and t_min == 0:
            t_min = t
    autocorr_list.append(min(t_zero, t_min))
    accept_list.append(accept)
```

```
[12]: plt.plot(Delta_list, autocorr_list, 'o-');
plt.xscale('log');
plt.yscale('log');
```



```
[13]: plt.plot(Delta_list, accept_list, 'o-');  
plt.xscale('log');  
plt.ylim(0, 1);
```



```
[14]: # Use a normal distribution for delta instead

states = np.zeros(Nstates)
delta = rng.normal(size=Nstates)
log_u = -np.log(rng.uniform(size=Nstates))
accept = MC_norm(x0, states, delta, log_u, mu, sigma2)
autocorr = acf_calc(states)
t_min, t_zero = 0, 0
for t in range(len(autocorr)-1):
    if autocorr[t] < 0.0 and t_zero == 0:
        t_zero = t
    if autocorr[t] < autocorr[t+1] and t_min == 0:
        t_min = t
```

```
[15]: accept, t_zero, t_min
```

```
[15]: (0.7060699462890625, 21, 24)
```

```
[16]: min(autocorr_list)
```

```
[16]: 10
```

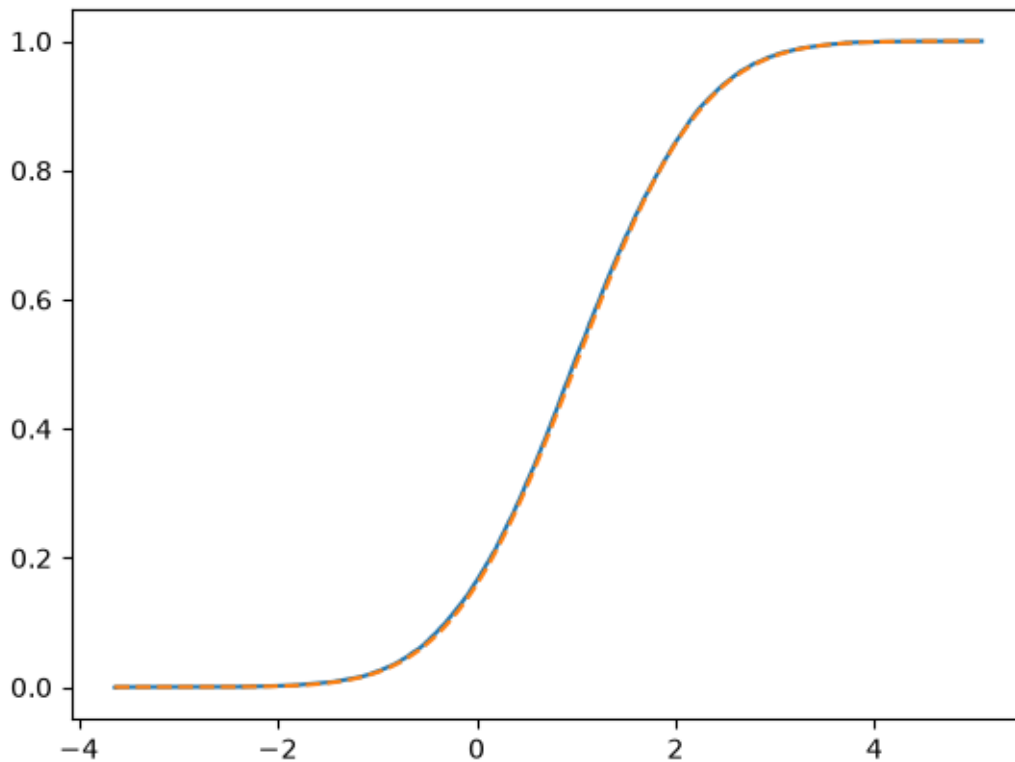
```
[17]: # Use a Cauchy distribution for delta instead

states = np.zeros(Nstates)
delta = rng.standard_cauchy(size=Nstates)
log_u = -np.log(rng.uniform(size=Nstates))
accept = MC_norm(x0, states, delta, log_u, mu, sigma2)
autocorr = acf_calc(states)
t_min, t_zero = 0, 0
for t in range(len(autocorr)-1):
    if autocorr[t] < 0.0 and t_zero == 0:
        t_zero = t
    if autocorr[t] < autocorr[t+1] and t_min == 0:
        t_min = t
```

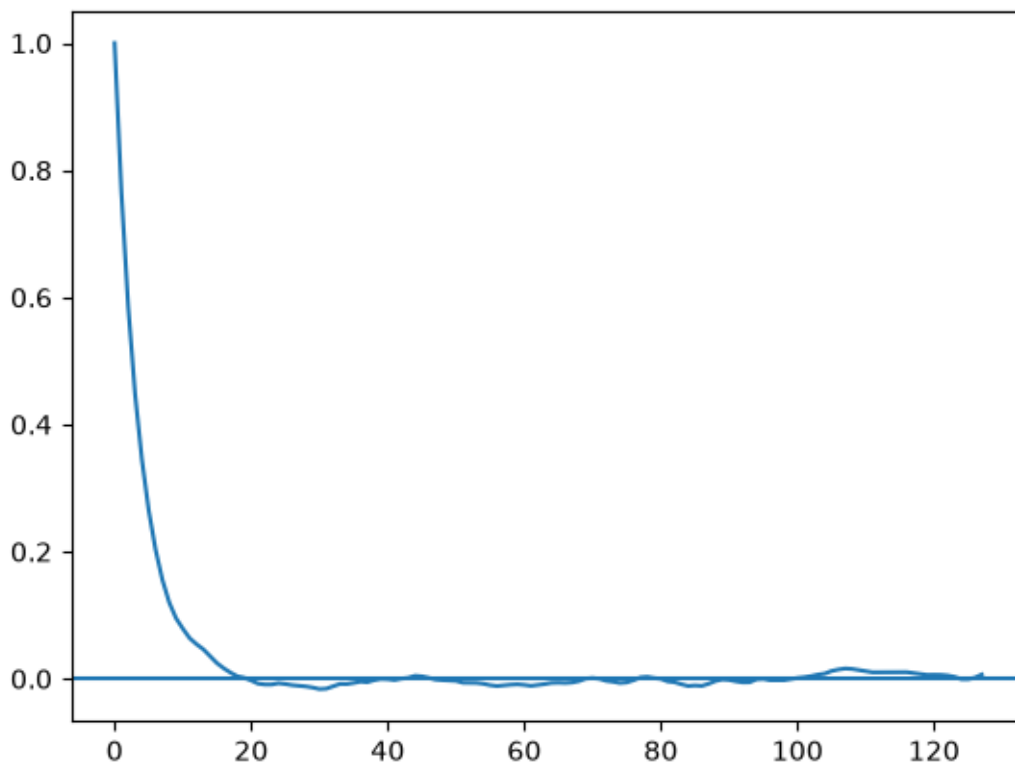
```
[18]: accept, t_zero, t_min
```

```
[18]: (0.53662109375, 20, 23)
```

```
[19]: x = np.sort(states)
y = np.linspace(0, 1, len(x))
plt.plot(x, y);
plt.plot(x, 0.5*(sp.erf((x-mu)/np.sqrt(2*sigma2))+1), '--');
```



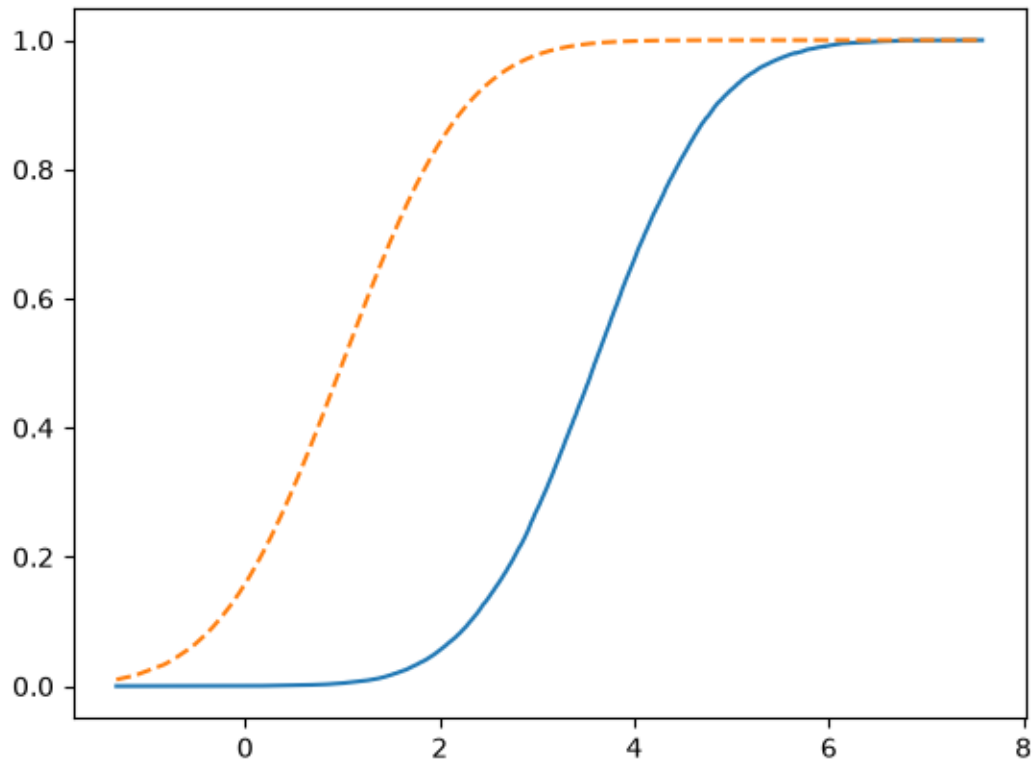
```
[20]: plt.plot(acf_calc(states)[:128]);
plt.axhline(0.0);
```



What happens if we use an asymmetric jump distribution for $T(s \rightarrow s')$ but don't account for it the acceptance?

```
[21]: states = np.zeros(Nstates)
Delta = 1.0
delta = rng.triangular(-Delta, Delta, Delta, size=Nstates)
log_u = -np.log(rng.uniform(size=Nstates))
accept = MC_norm(x0, states, delta, log_u, mu, sigma2)
```

```
[22]: x = np.sort(states)
y = np.linspace(0, 1, len(x))
plt.plot(x, y);
plt.plot(x, 0.5*(sp.erf((x-mu)/np.sqrt(2*sigma2))+1), '--');
```



```
[23]: @jit(nopython=True)
def MC_norm_asymm(x0, xstates, delta, deltaE, log_u, mu, sigma2):
    """Does MCMC, starting from x0 for N(mu, sigma2), storing in xstates
    where delta is the random (asymmetric) displacements, with "energy
    correction" deltaE, and log_u is -np.log(u) for u from a uniform
    distribution"""
    Nstates = len(xstates)
    x = x0
    accept = 0
    for i in range(Nstates):
        d = delta[i]
        xp = x + d
        dE = 0.5*(d*d + 2*(x-mu)*d) + deltaE[i]
        if sigma2*log_u[i] > dE:
            x = xp
            accept += 1
        xstates[i] = x
    return accept/Nstates
```

```
[24]: states = np.zeros(Nstates)
Delta = 1.0
delta = rng.triangular(-Delta, Delta, Delta, size=Nstates)
```

```

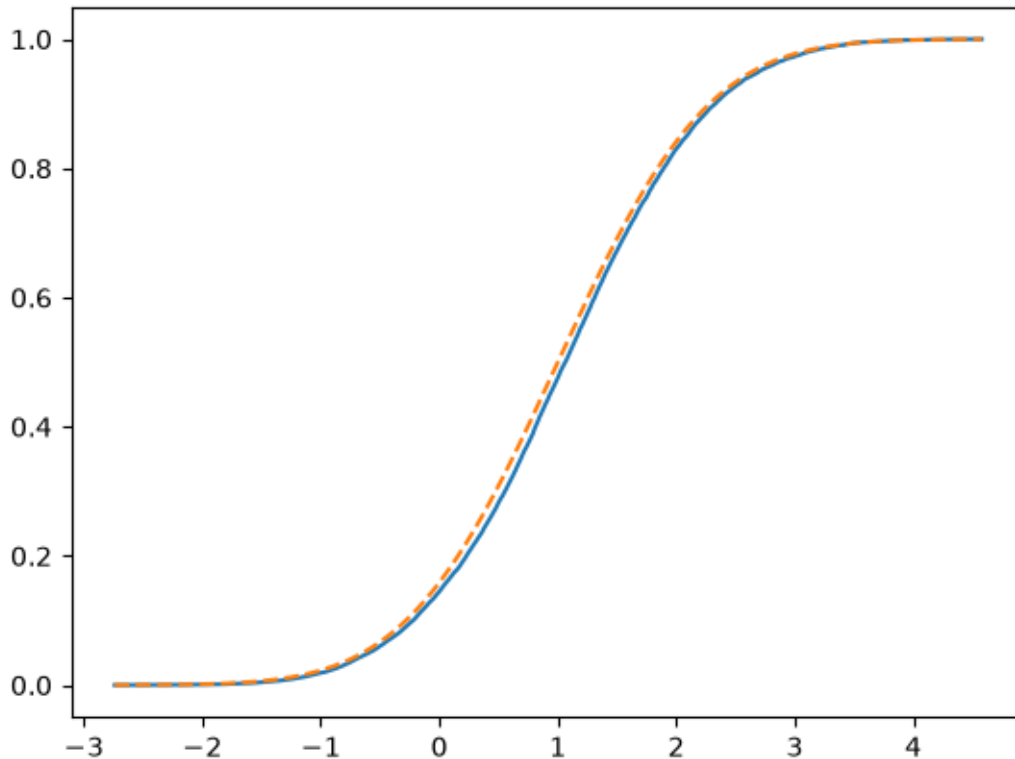
deltaE = -sigma2*np.log((Delta-delta)/(Delta+delta))
log_u = -np.log(rng.uniform(size=Nstates))
accept = MC_norm_asymm(x0, states, delta, deltaE, log_u, mu, sigma2)

```

```

[25]: x = np.sort(states)
      y = np.linspace(0, 1, len(x))
      plt.plot(x, y);
      plt.plot(x, 0.5*(sp.erf((x-mu)/np.sqrt(2*sigma2))+1), '--');

```



What happens if we loop until we get a move we accept?

```

[26]: states = np.zeros(Nstates)
      x = x0
      Delta = 1.0
      for i in range(Nstates):
          accept = False
          while not accept:
              d = rng.uniform(-Delta, Delta)
              xp = x + d
              dE = 0.5*(d*d + 2*(x-mu)*d)
              if sigma2*log_u[i] > dE:
                  x = xp

```

```

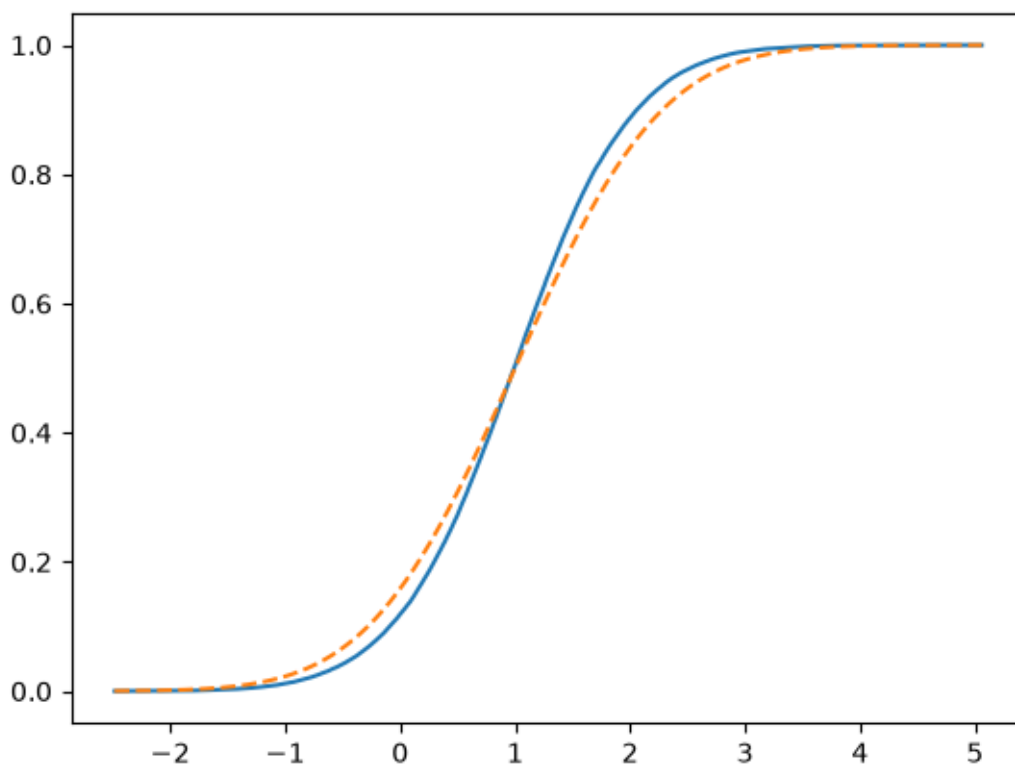
        accept = True
        states[i] = x

```

```

[27]: x = np.sort(states)
      y = np.linspace(0, 1, len(x))
      plt.plot(x, y);
      plt.plot(x, 0.5*(sp.erf((x-mu)/np.sqrt(2*sigma2))+1), '--');

```



Let's look at doing Force Bias MC, where we sample our displacement from an exponential in the range $[-\Delta, \Delta]$ with a PDF given by

$$T(\delta) = \frac{\alpha}{2 \sinh(\alpha \Delta)} \exp(\alpha \delta)$$

where $\alpha = -(x - \mu)/\sigma^2$ is the derivative of $\log$ probability at x .

```

[28]: @jit(nopython=True)
      def MC_norm_forcebias(x0, xstates, Delta, u1, u2, mu, sigma2):
          """Does MCMC, starting from x0 for N(mu, sigma2), storing in xstates,
              where Delta is the range over which we sample displacements, and
              where u1 and u2 are uniformly distributed random numbers to be used
              to generate displacements and generate acceptance"""
          Nstates = len(xstates)
          x = x0

```

```

accept = 0
for i in range(Nstates):
    alpha = -(x-mu)/sigma2
    expap = np.exp(alpha*Delta)
    expam = 1.0/expap
    d = np.log(u1[i]*(expap-expam) + expam)/alpha
    xp = x + d
    dE = 0.5*(d*d + 2*(x-mu)*d)/sigma2
    probb_accept = np.exp(dE)*(xp-mu)/(x-mu) * \
        np.sinh((x-mu)*Delta/sigma2)/np.sinh((xp-mu)*Delta/sigma2)
    # print(f'{x} {d} {alpha} {dE} {probb_accept}')
    if u2[i] < probb_accept:
        x = xp
        accept += 1
    xstates[i] = x
return accept/Nstates

```

```

[29]: states = np.zeros(Nstates)
Delta = 2.5
accept = MC_norm_forcebias(x0, states, Delta,
                           rng.uniform(size=Nstates), rng.uniform(size=Nstates),
                           mu, sigma2)

```

```

[30]: accept

```

```

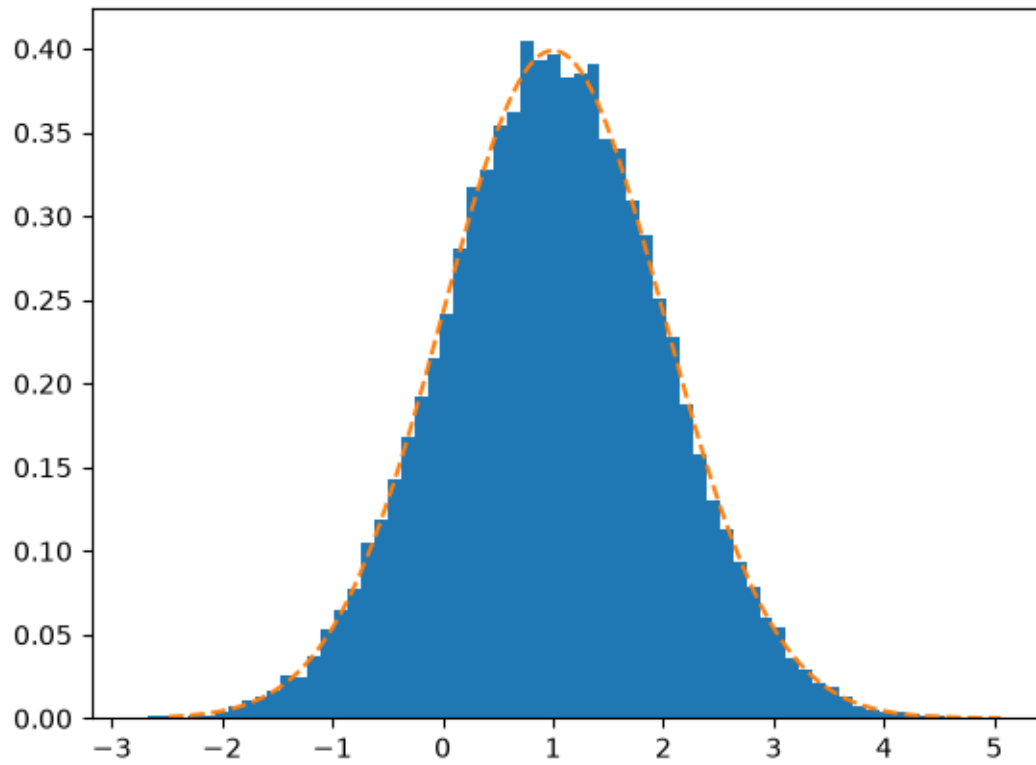
[30]: 0.8331146240234375

```

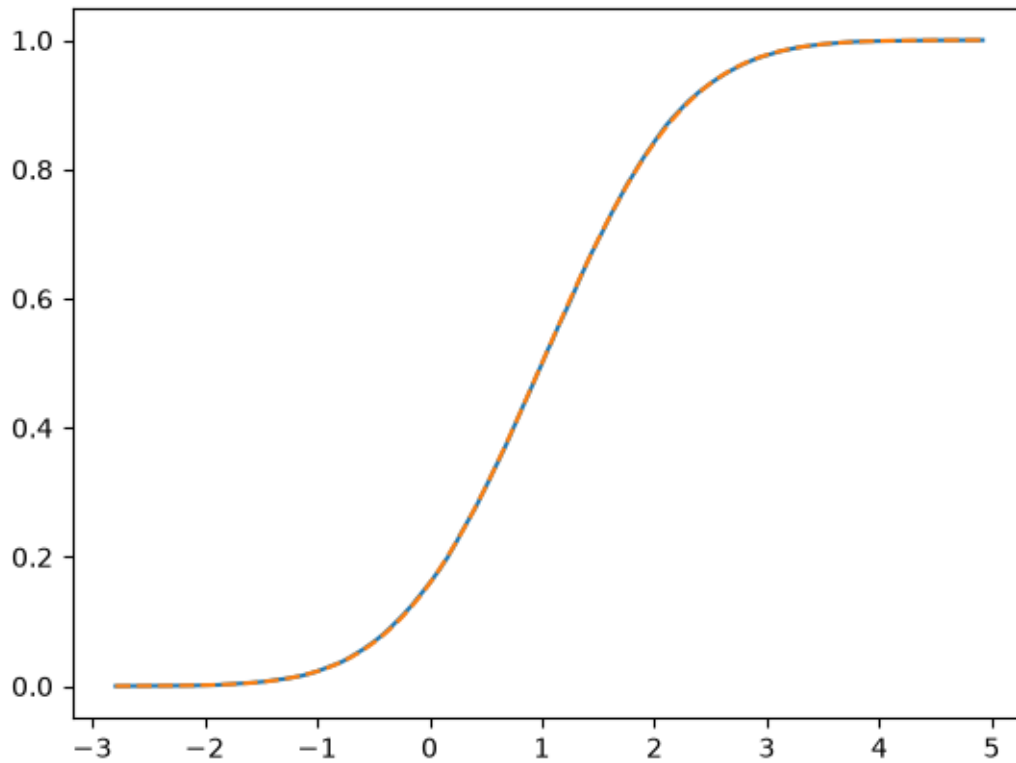
```

[31]: plt.hist(states, density=True, bins=64);
plt.plot(x, Pnorm(x, mu, sigma2), '--');

```



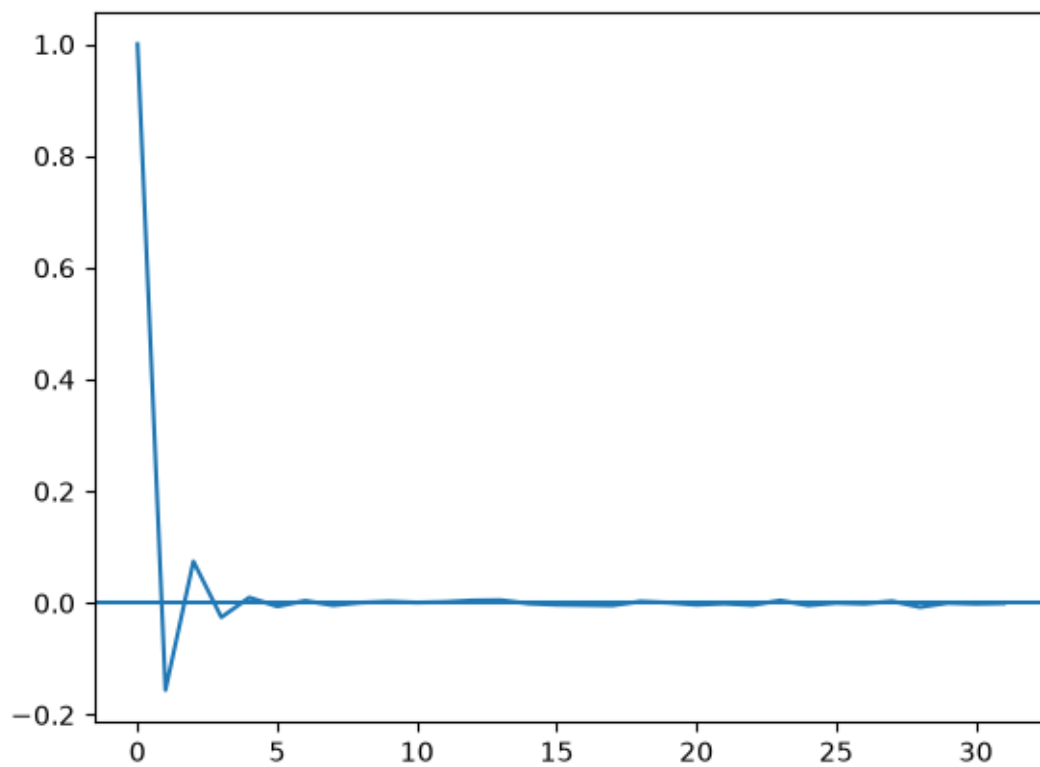
```
[32]: x = np.sort(states)
      y = np.linspace(0, 1, len(x))
      plt.plot(x, y);
      plt.plot(x, 0.5*(sp.erf((x-mu)/np.sqrt(2*sigma2))+1), '--');
```



```
[33]: autocorr = acf_calc(states)
      t_min, t_zero = 0, 0
      for t in range(len(autocorr)-1):
          if autocorr[t] < 0.0 and t_zero == 0:
              t_zero = t
          if autocorr[t] < autocorr[t+1] and t_min == 0:
              t_min = t
      print(t_min, t_zero)
```

1 1

```
[34]: plt.plot(autocorr[:32]);
      plt.axhline(0.0);
```



[]: