Markov Chain Monte Carlo

Odds and Ends

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

Contents

In this lecture, we will explore some of the practical details of running and analyzing MCMC computations.

It is best to have a specific problem in mind to anchor our discussion.

(i) An Example Problem

(ii) Metropolis MCMC

(iii) Issues

- burn-in
- thinning
- traceplots
- convergence diagnostics

(iv) Error Analysis

- auto-correlation
- block averaging

Example Problem

The multivariate normal distribution has the form:

$$f_{\mathbf{x}}(x_1, \ldots, x_k) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right),$$

where $\mathbf{x}$ is a k-dimensional column vector and $|\boldsymbol{\Sigma}|$ is the determinant of the symmetric covariance matrix $\boldsymbol{\Sigma}$.

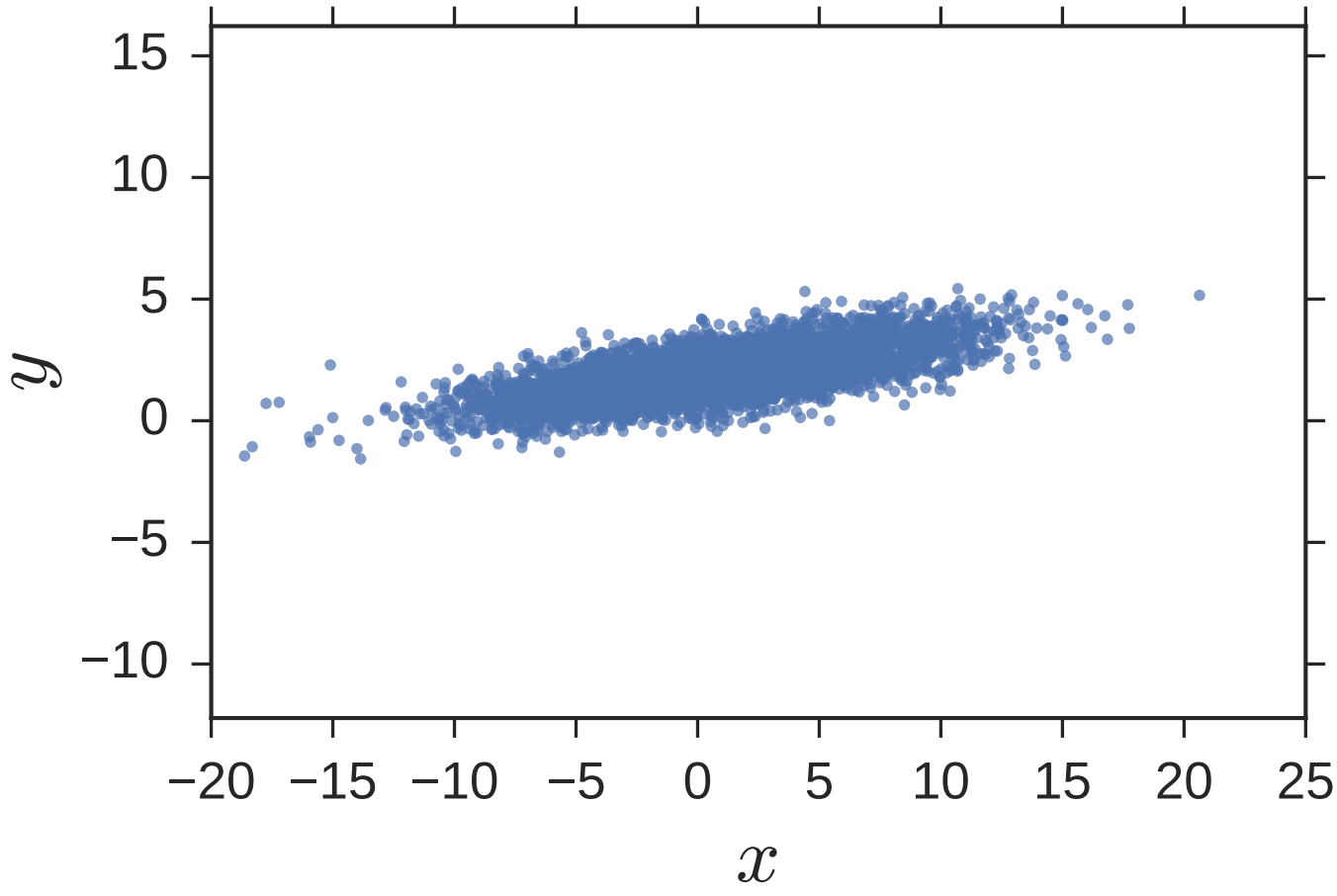
When $k = 2$, we have a bivariate normal distribution, that we saw earlier.

Let us consider a specific bivariate distribution,

$$\boldsymbol{\mu} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} 25 & 3.5 \\ 3.5 & 1 \end{bmatrix}$$

Specific Problem

This corresponds to the bivariate distribution with $\sigma_x = 5$, $\sigma_y = 1$, and $\rho = 0.7$.



Metropolis MCMC

While there are direct MC methods for sampling multivariate normal distributions, let us write a standard MCMC routine to learn how to analyze MCMC simulations.

We need to write python functions for:

(i) the distribution to sample,

(ii) the proposal function

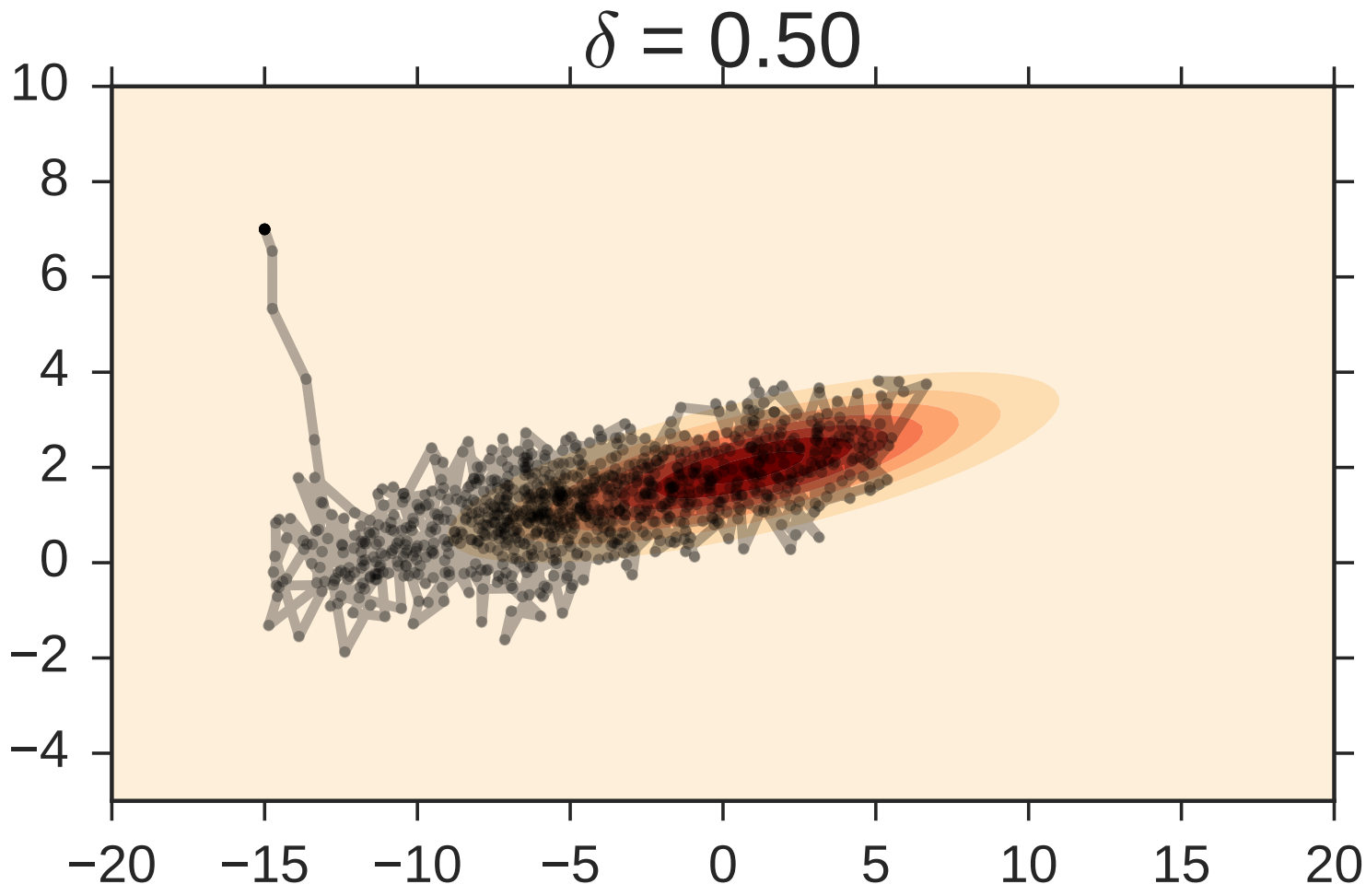
(iii) acceptance functions, and

(iv) the driver routine to glue everything together.

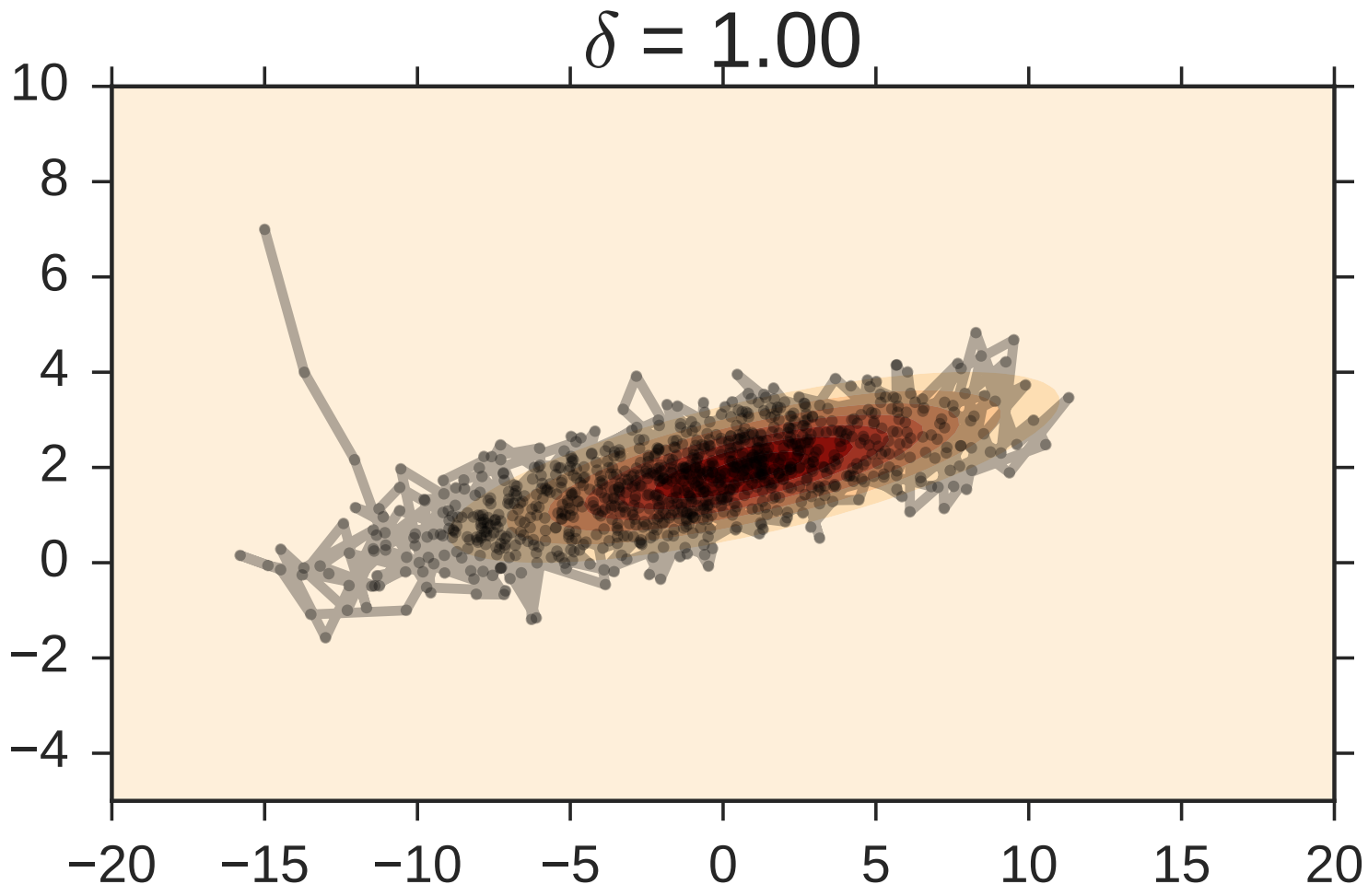
We will monitor the two components x and y, and the mean squared distance from the origin $a(x, y) = x^2 + y^2$.

The objective is to estimate the mean values $\langle x \rangle$ (μ_x=1), $\langle y \rangle$ (μ_y=2), and $\langle a \rangle$ ($\approx$ 31).

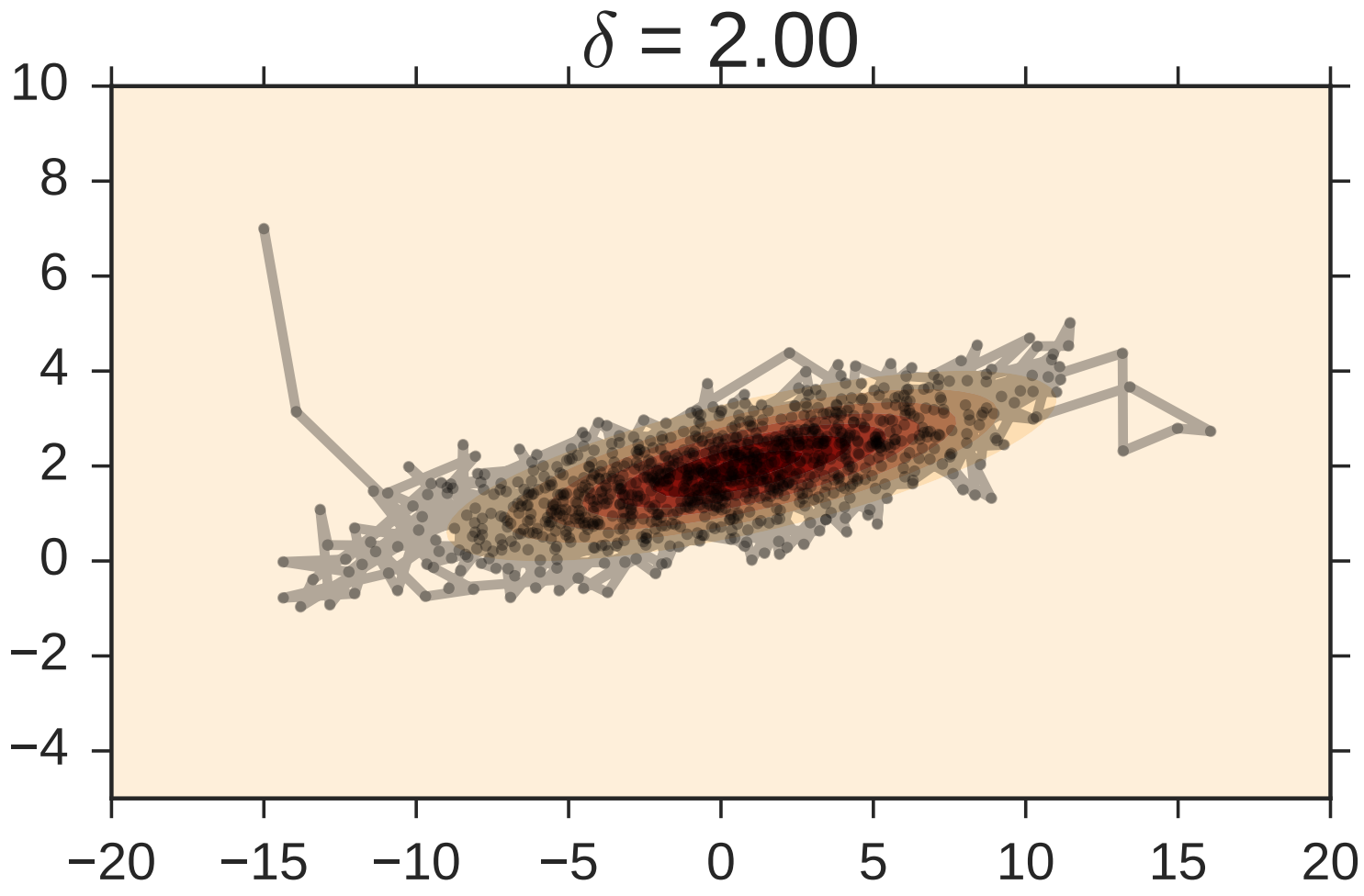
$\delta = 0.5$



$\delta = 1.0$



$\delta = 2.0$



Acceptance Ratio and Means

δ	accept	$\langle x \rangle$	$\langle y \rangle$	$\langle a \rangle$
0.3	0.08	-14.53	6.33	257.19
0.5	0.77	0.54	2.04	23.09
1.0	0.72	1.06	2.03	31.56
2.0	0.51	1.08	2.01	38.59

Recall that expected means are $\langle x \rangle = 1$, $\langle y \rangle = 2$, and $\langle a \rangle \approx 31$.

We have not furnished error-bars, because we don't know how to calculate them yet.

Issues

Let's consider four common issues:

- Burn-in
- Thinning
- Traceplots
- Convergence Diagnostics

Let us now look at some of the common practical issues, and understand the nature of typical tradeoffs involved.

Many of these choices are "fuzzy"; they fall in the category of things that you can do better with "experience".

Fuzzy does not imply unimportant.

Burn-in

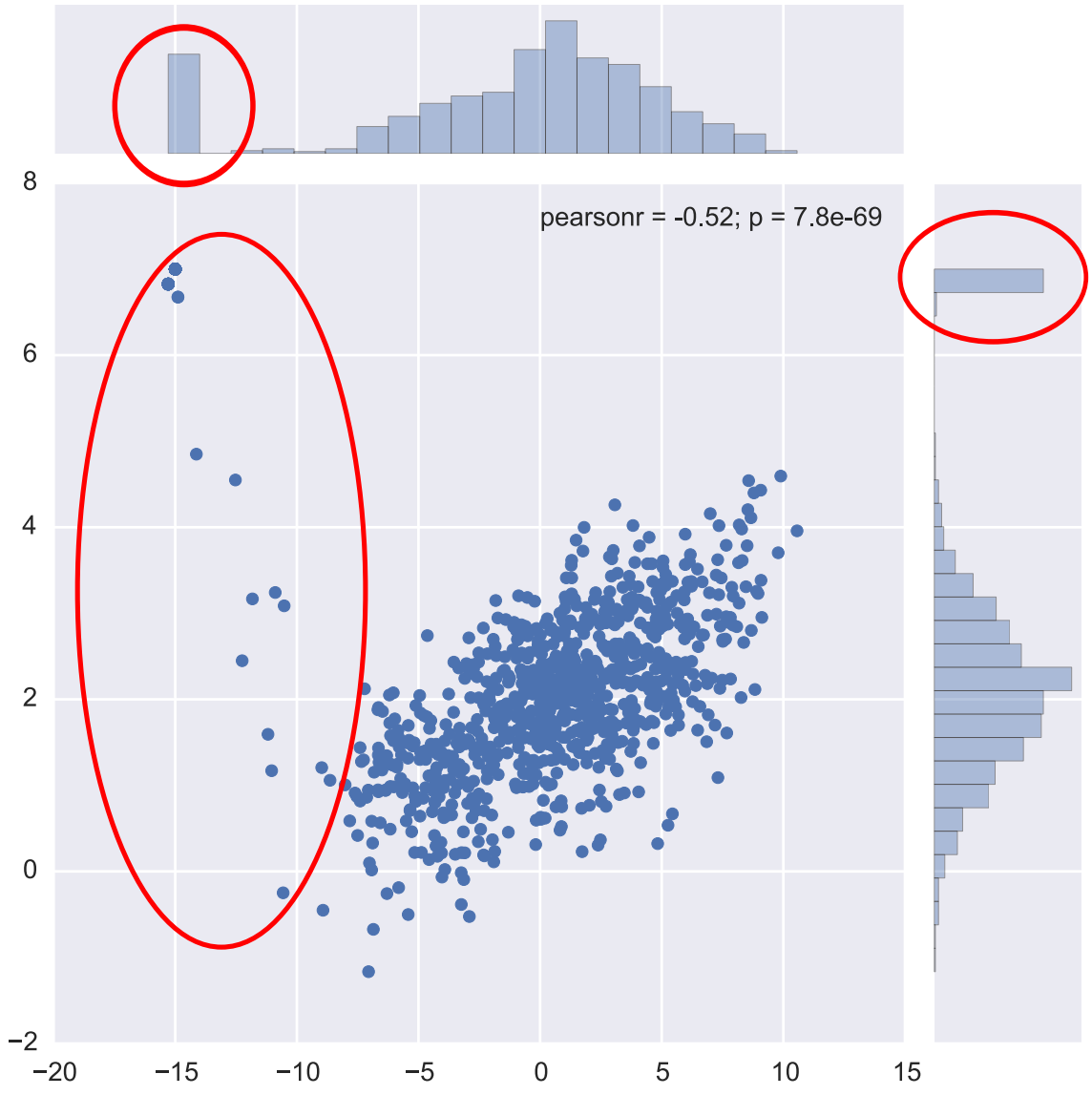
Discarding initial samples to forget the starting point.

In the example above, we started from an initial state that was not in the most important part of the distribution.

Consequently, it took a few steps for the chain to travel from the initial state to the heart of the distribution.

For example, with $\delta = 0.5$, this is reflected in the joint and marginal PDFs as a spurious peak near the starting point

Burn-in



Burn-in

Where does this dependence stem from?

Recall, the state after n applications of the transition probability matrix (TPM) $\mathbf{W}$ is:

$$\boldsymbol{\pi}_n = \mathbf{W}^n \boldsymbol{\pi}_0.$$

$\mathbf{x}_n$ independent $\mathbf{x}_0$ only for sufficiently large n.

Rate depends on λ_2 of TPM (not known *a priori*).

Generate traceplots or use convergence diagnotics (discussed shortly) to ascertain convergence.

Tradeoff: If you discard

- too much: risk throwing away good data;
- too little: risk contamination

Thinning

MCMC generates a large number of samples.

Large dimensional problems and/or long runs - run into memory problems, if we record all states visited.

Consecutive states are strongly correlated, and not very informative.

Analogy: monitor temperature of a room for a day.

Record temperature every millisecond?

Temperature doesn't change meaningfully over that scale.

How about a minute, or 10 minutes, an hour, 6 hours?

Meaningful trade-off between storing too much (millisecond) and too little (6 hours) data.

Happy balance (perhaps 10 minutes) depends on taste, computer capacity, and target application.

TracePlots

We can plot important variables in the problem, and try to visually inspect if they are "well-mixed"

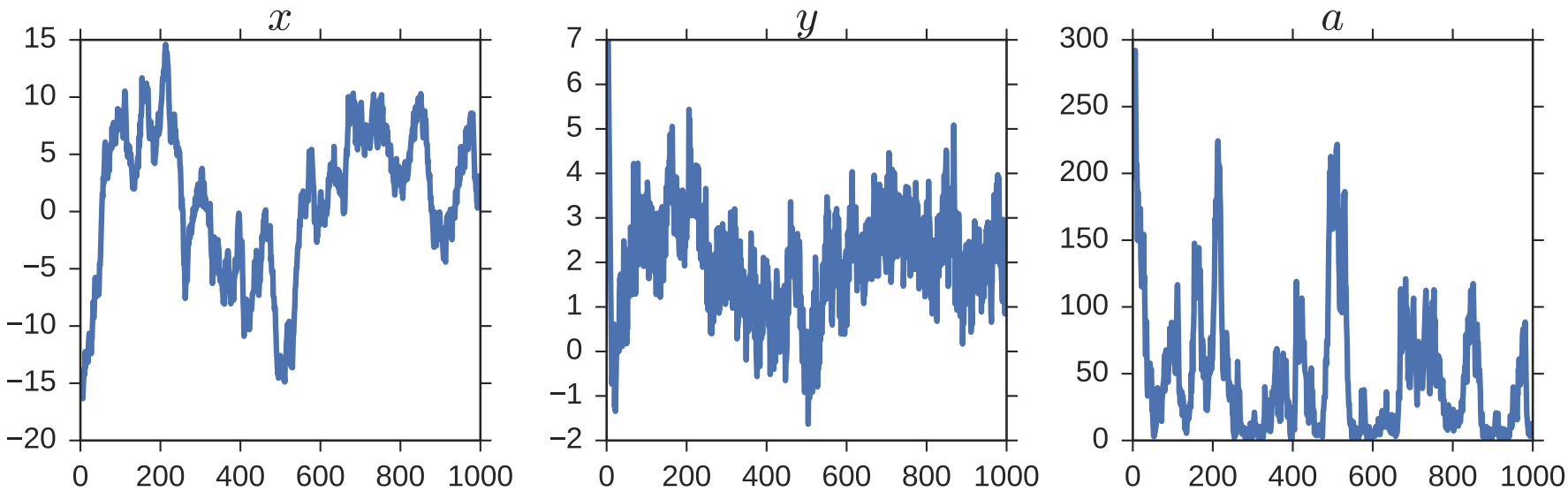
If the variables span the domain repeatedly over simulation run time, then that is evidence for good mixing.

Qualitative measure, but vitally important to assess!

As extreme cases above, let us consider $\delta = 0.3$ (poor mixing) and $\delta = 2.0$ (good mixing).

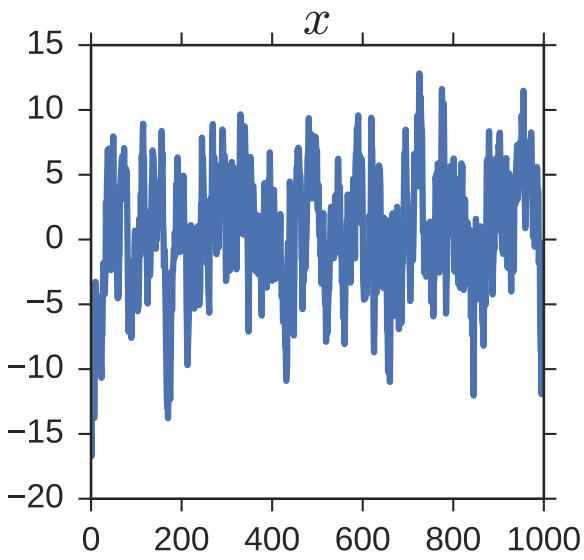
Poor Mixing

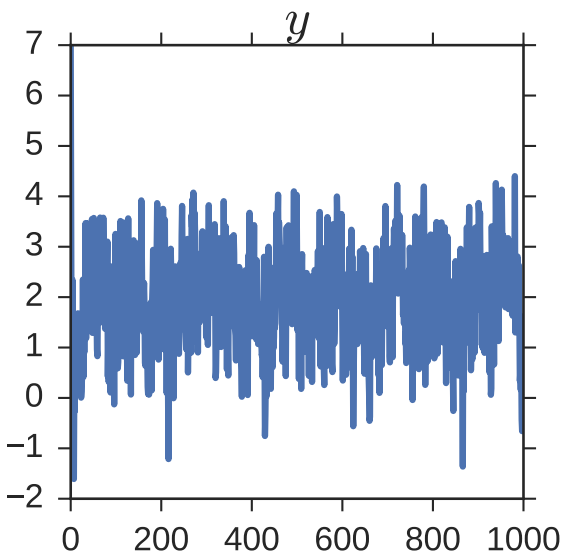
$$\delta = 0.3$$

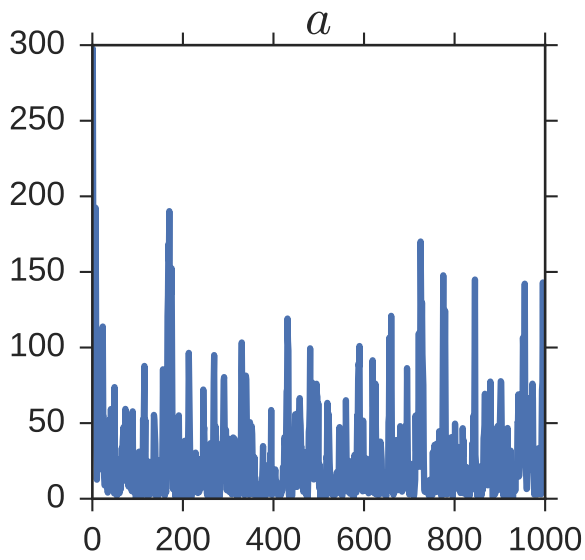


Good Mixing

$$\delta = 2.0$$







Convergence Diagnostics

Many diagnostics to test whether an MCMC run has converged.

None of them are perfect; they don't prove beyond a shadow of doubt that your MCMC simulation has converged.

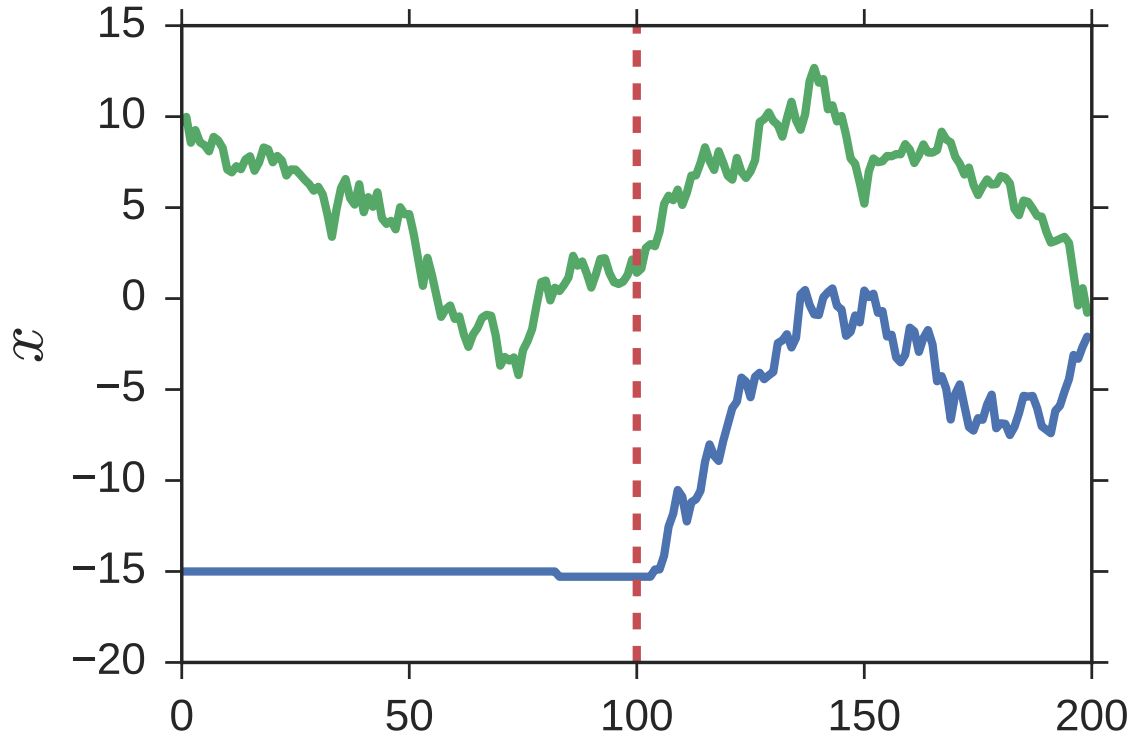
However, often they can diagnose poor mixing.

Perhaps the most famous of these convergence heuristics: Gelman-Rubin diagnostic.

Motivation: Run multiple MCMC chains/simulations from different starting points. We want to know "Is the run is long enough to forget the effect of the starting point?"

Convergence

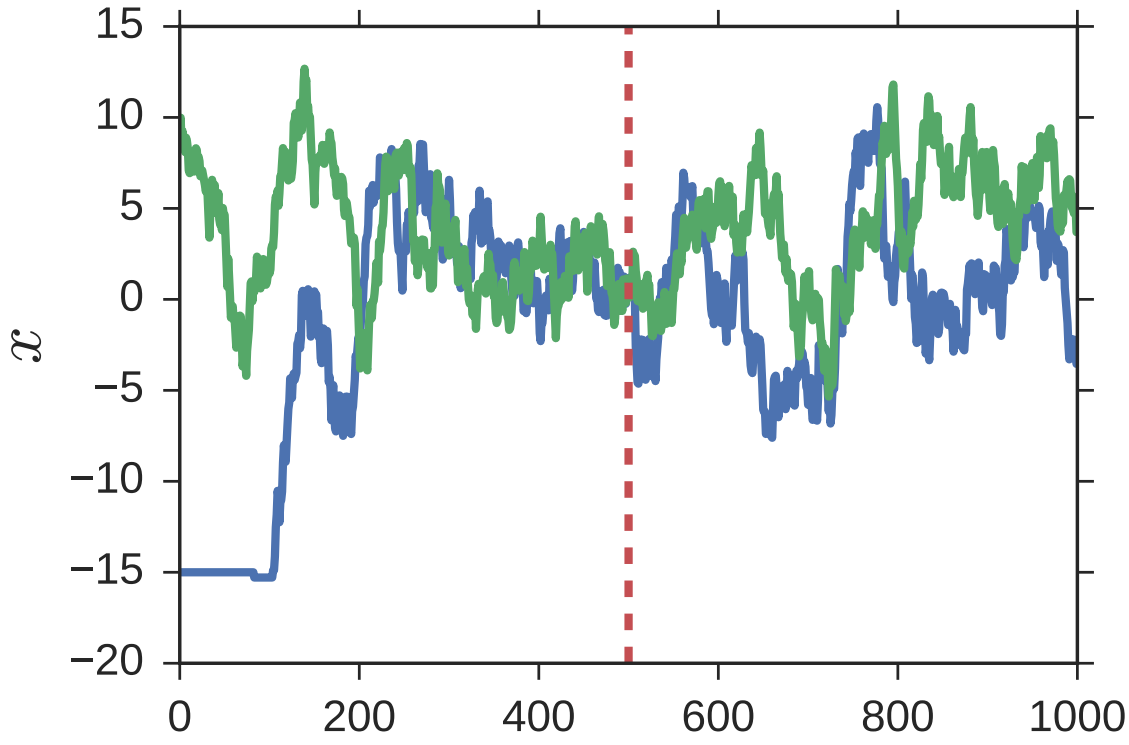
Consider a traceplot of x for $\delta = 0.5$ starting from two different points ([-15, 7] and [10, -2])



After the $2M = 200$ snapshots, the curves still seem to carry some memory of the initial condition.

Converged: Longer Run

The same simulations carried out to $2M = 1000$ snapshots. The two chains seem to have "merged".



Red dashed line marks simulation midpoint. For Gelman-Rubin diagnostic, we discard left half as burnin.

Gelman-Rubin Diagnostic

This diagnostic provides a quantitative metric to assess convergence.

Meta-observation: once a simulation runs long enough, the "variance" between the MCMC chains starting from different initial conditions becomes small.

Small compared to what? The variance within-chain. If,

- $W =$ within-chain variance
- $B =$ between-chain or inter-chain variance

If $B \geq MW$, then effect of starting point has not subsided.

When $W \gg B/M$, all the chains have escaped the influence of starting points and traversed to the target distribution.

Gelman-Rubin Diagnostic: Algorithm

Let us write down the recipe for calculating the PSRF $\hat{R}$. Then, we will look at an example.

For each scalar variable of interest A:

1. Run $n \geq 2$ chains of length $2M$ from overdispersed starting values.
2. Discard the first M samples in each chain.
3. Calculate the within-chain W and between-chain B variance.
4. Calculate the estimated variance of the parameter A
5. Calculate the potential scale reduction factor (PSRF).
6. Check if PSRF $=$ 1.0 - 1.2; if greater, then chains not converged

Example: Steps 1 and 2

Let's set $A = x$ (the first coordinate) and sample from the bivariate Gaussian.

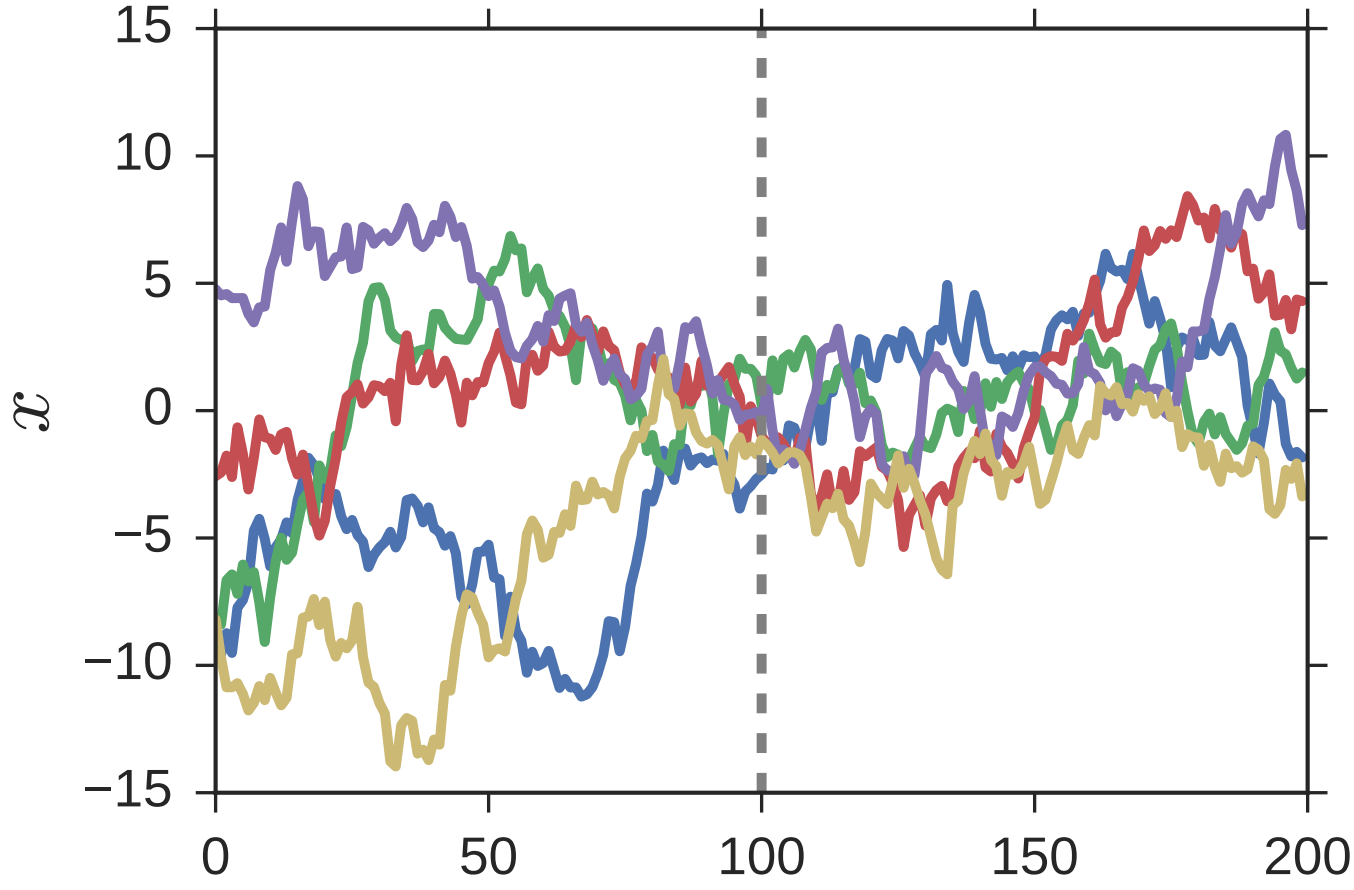
Let us compute the Gelman-Rubin for $\delta = 0.5$.

```
np.random.seed(1234)
n     = 5
thin  = 10
M     = 100
thin  = 10
delta = 0.5

driver(delta, 2*M*thin, thin)
```

Note: I modified the driver routine to pick random initial points.

$\delta = 0.5$

The $n = 5$ chains, complete trajectories:



Gelman-Rubin only considers the right half; here $A = x$.

Step 3: Within-Chain Variance W

Compute mean and variance of j^{th} chain:

$$\bar{A}_j = \frac{1}{M}\sum_{i=1}^{M} A_{ij}$$

$$s_j^2 = \frac{1}{M-1}\sum_{i=1}^{M}(A_{ij} - \bar{A}_j)^2$$

Hence, compute the within-chain variance W

$$W = \frac{1}{n}\sum_{j=1}^{n} s_j^2$$

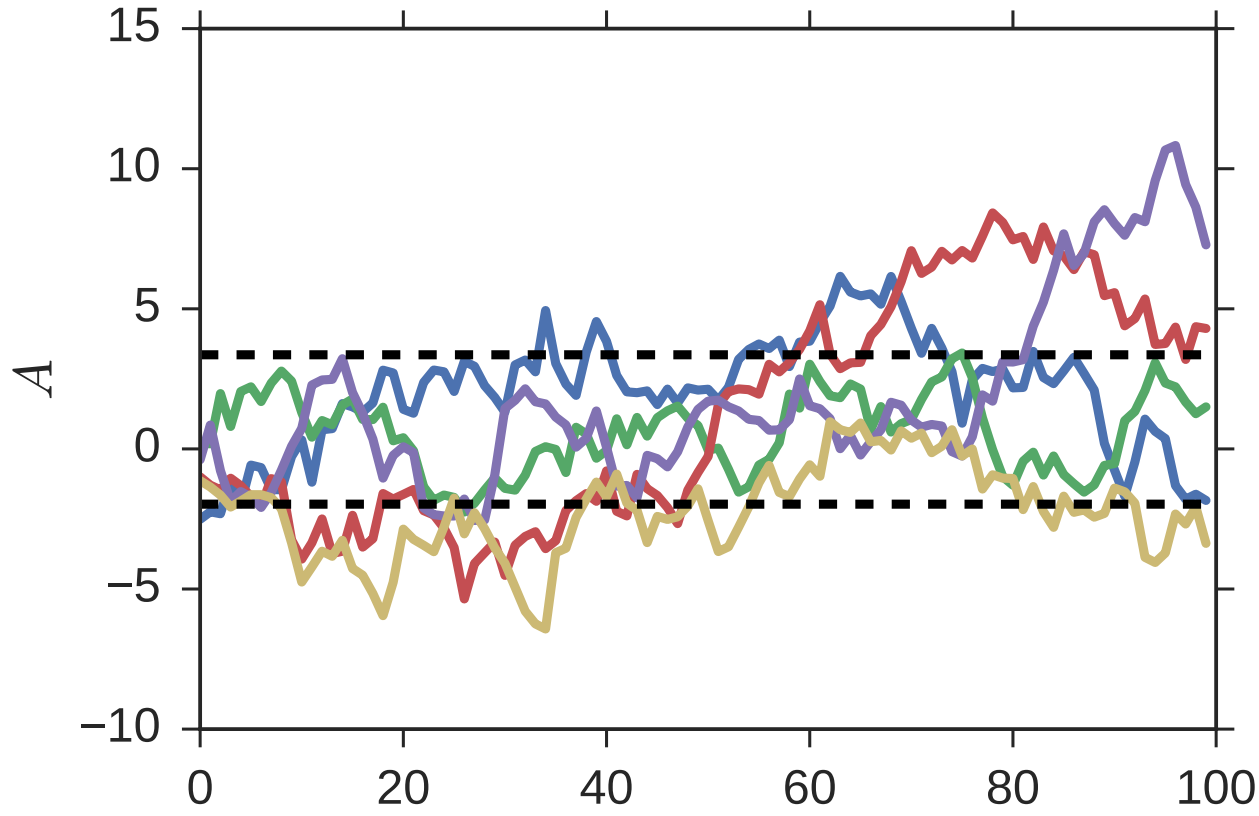
For short runs, W likely mis-estimates the true variance of the stationary distribution since our chains have probably not all reached the points of the stationary distribution.

$\delta = 0.5$: Calculating W

Figure shows second half of trajectory data

The variances for individual chains,

$$\begin{bmatrix} s_1^2 \\ s_2^2 \\ s_3^2 \\ s_4^2 \\ s_5^2 \end{bmatrix} = \begin{bmatrix} 4.17 \\ 1.91 \\ 16.23 \\ 10.53 \\ 2.70 \end{bmatrix}$$



The mean s_j^2 is the within-chain variance $W = 7.11$ characterizes within-chain spread.

Distance between dashed lines $= 2\sqrt{W}$.

Step 3/4: Inter-Chain Variance B

Compute the grand mean:

$$A^* = \frac{1}{n}\sum_{j=1}^{n} \bar{A}_j$$

And hence, the average between-chain variance[1]

$$B = \frac{M}{n-1}\sum_{j=1}^{n}(\bar{A}_j - A^*)^2$$

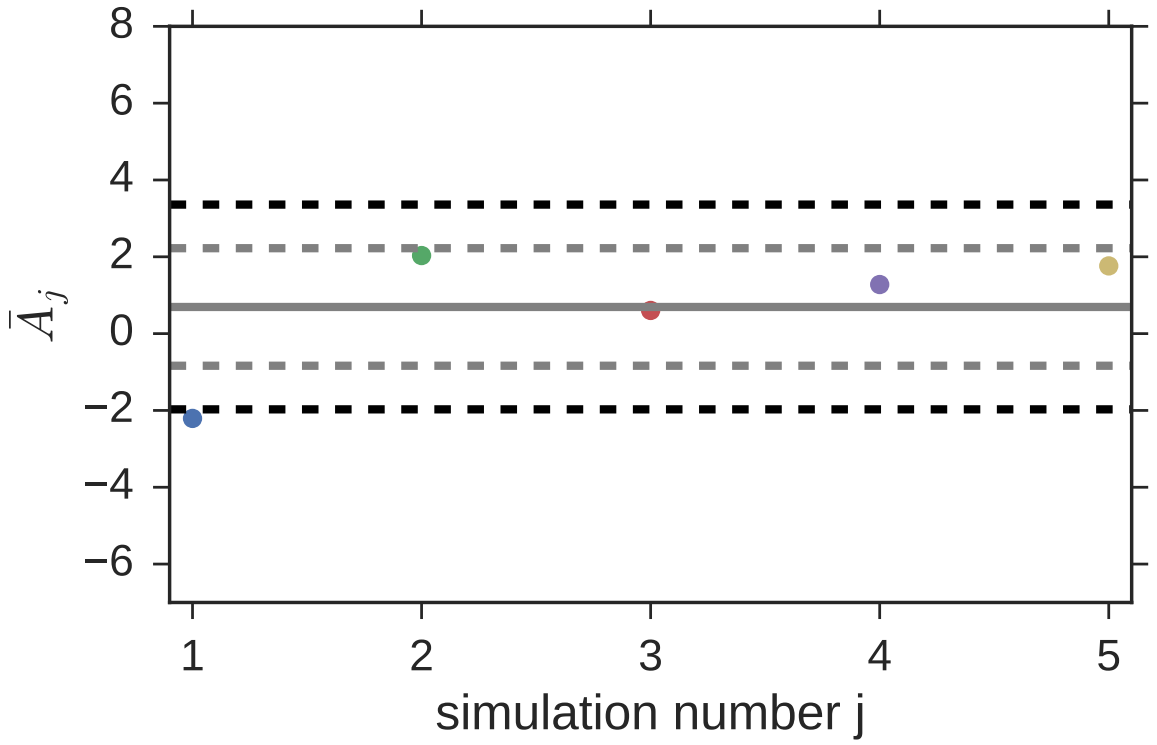
The estimated variance of A is given by:

$$\sigma_A^2 = \left(1 - \frac{1}{M}\right)W + \frac{1}{M}B.$$

[1] factor of M to account for variance of A (W) and $\bar{A}$ (B/M).

$\delta = 0.5$: Calculating B/M

Simulation averages (second-half)



Mean of $\bar{A}_j$:
$A^* = 0.69$
solid gray

Variance of $\bar{A}_j$:
$B/M = 2.34$
dashed gray $=$
$A^* \pm \sqrt{B/M}$

Dashed black lines are $A^* \pm \sqrt{W}$ from previous slide.

$\sigma_A^2 = 9.38$, $\hat{R} = 1.15$

Step 5: Shrink Factor

The potential scale reduction factor (PSRF) or the "shrink factor" is given by:

$$\hat{R} = \sqrt{\frac{\sigma_A^2}{W}} = \sqrt{\left(1 - \frac{1}{M}\right) + \frac{1}{M}\frac{B}{W}}.$$

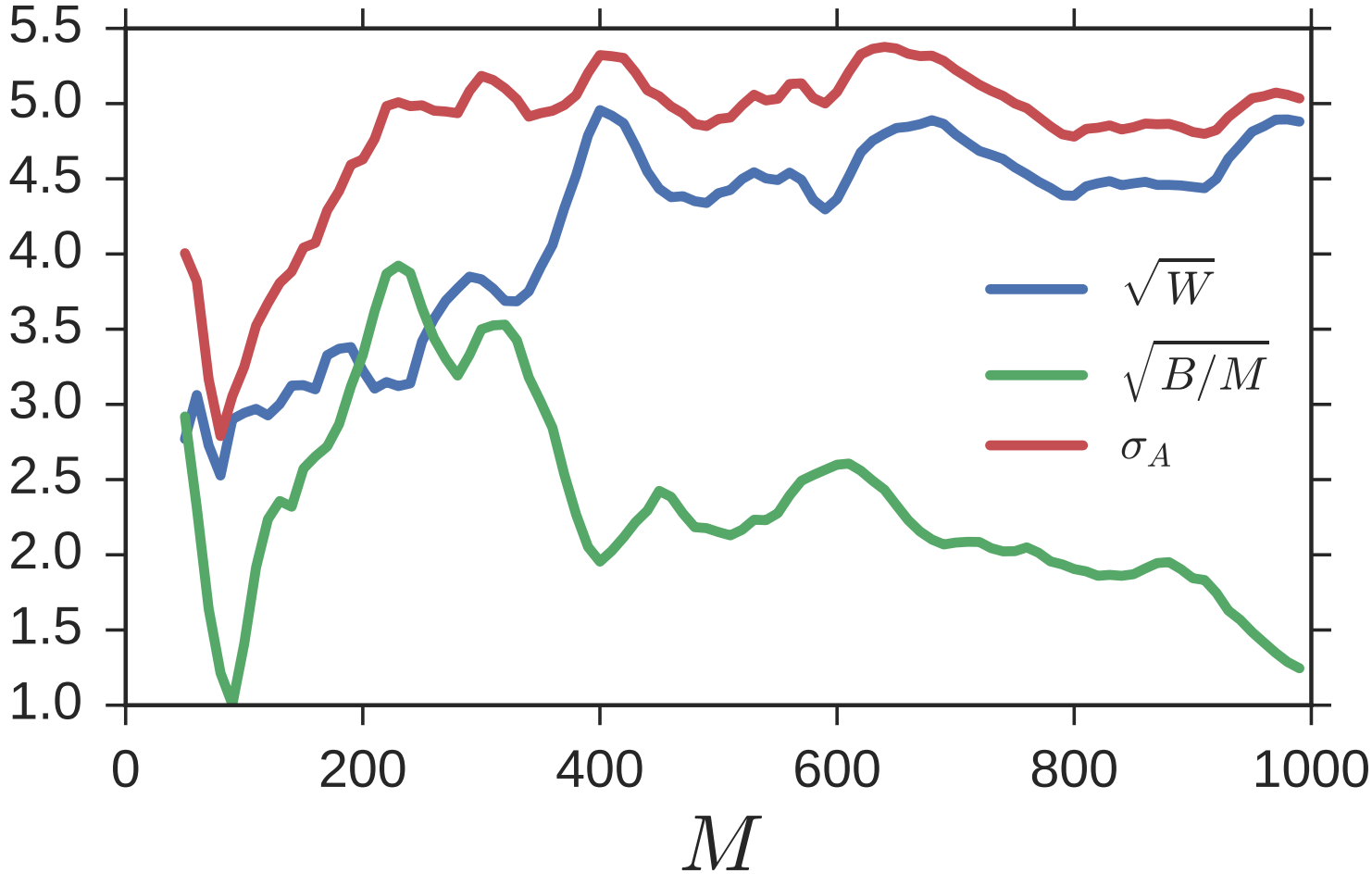
As $M \to \infty$, and $\hat{R} \to 1$.

In practice, when $\hat{R} > 1.1 - 1.2$, we surmise that convergence has not been attained.

In such cases, we need to run our chains longer to improve convergence to the stationary distribution.

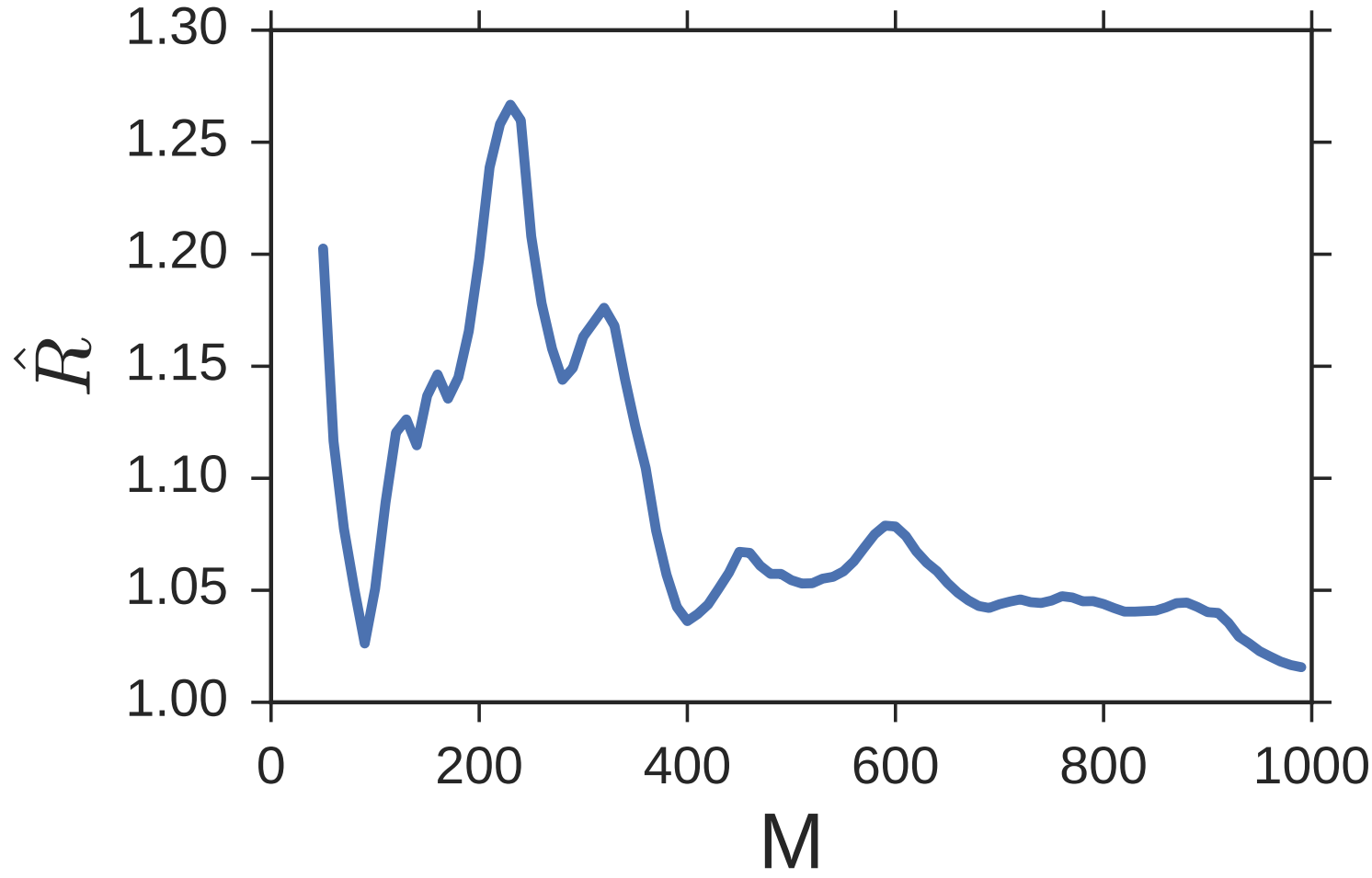
Simulation Time

As simulation time increases $W > B/M$.



Here simulation time $= 2M$

Simulation Time



Gelman-Rubin Diagnostic

Based on the criterion of the shrink factor, the diagnosis is that $M \approx 400$ or higher results in stable converged MCMC simulations.

Criticisms

- burden of finding over-dispersed starting configurations is on the user
- somewhat inefficient, since a large number of early iterations $(M \times n)$ are discarded, compared with a single long run
- the method internally relies on some uncontrolled normal approximations
- since the criterion is based on scalars (A), it does not do a good job of picking up on correlations; in such cases the true convergence is slower than the estimate

Error Analysis Redux

Direct sampling: draw independent samples of property we want to measure.

Here the quantities we are after are the means of the two co-ordinates $\langle x \rangle$, $\langle y \rangle$, and $\langle a \rangle$.

In direct sampling, with M independent samples, we have for property A,

$$\sigma^2(\langle A \rangle) = \frac{\sigma^2(A)}{M}.$$

However, in an MCMC calculation the samples are not independent.

What are the consequences of this fact for error estimation?

Auto-correlation

What would happen if we applied the error formula meant for independent variables to correlated variables?

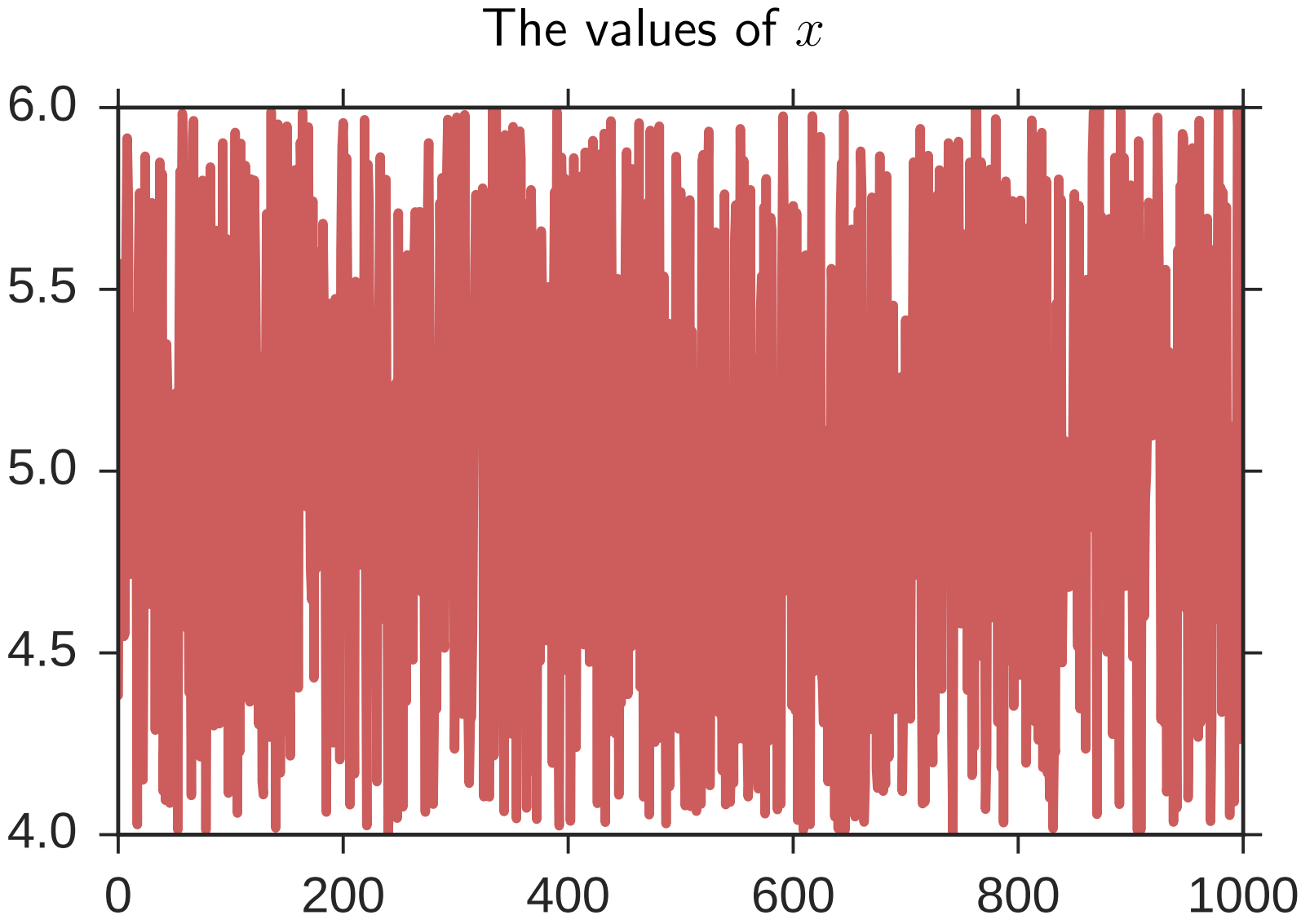
Consider a simple example.

I generated time series with $M = 1000$ independent samples of a random number with mean value of 5 and some noise.

```
M = 1000
x = np.random.uniform(4, 6, M)
```

The mean and standard deviation of x were 5.02, and 0.59, respectively.

Independent Variables



Autocorrelation

The auto-correlation function ϕ of a "time series" series A_i with $i = 0, 1, ..., M$, is defined as:

$$\phi_i = \frac{\langle A_i A_0 \rangle - \langle A \rangle^2}{\langle A^2 \rangle - \langle A \rangle^2}.$$

If $\langle A \rangle = 0$, then it simplifies to:

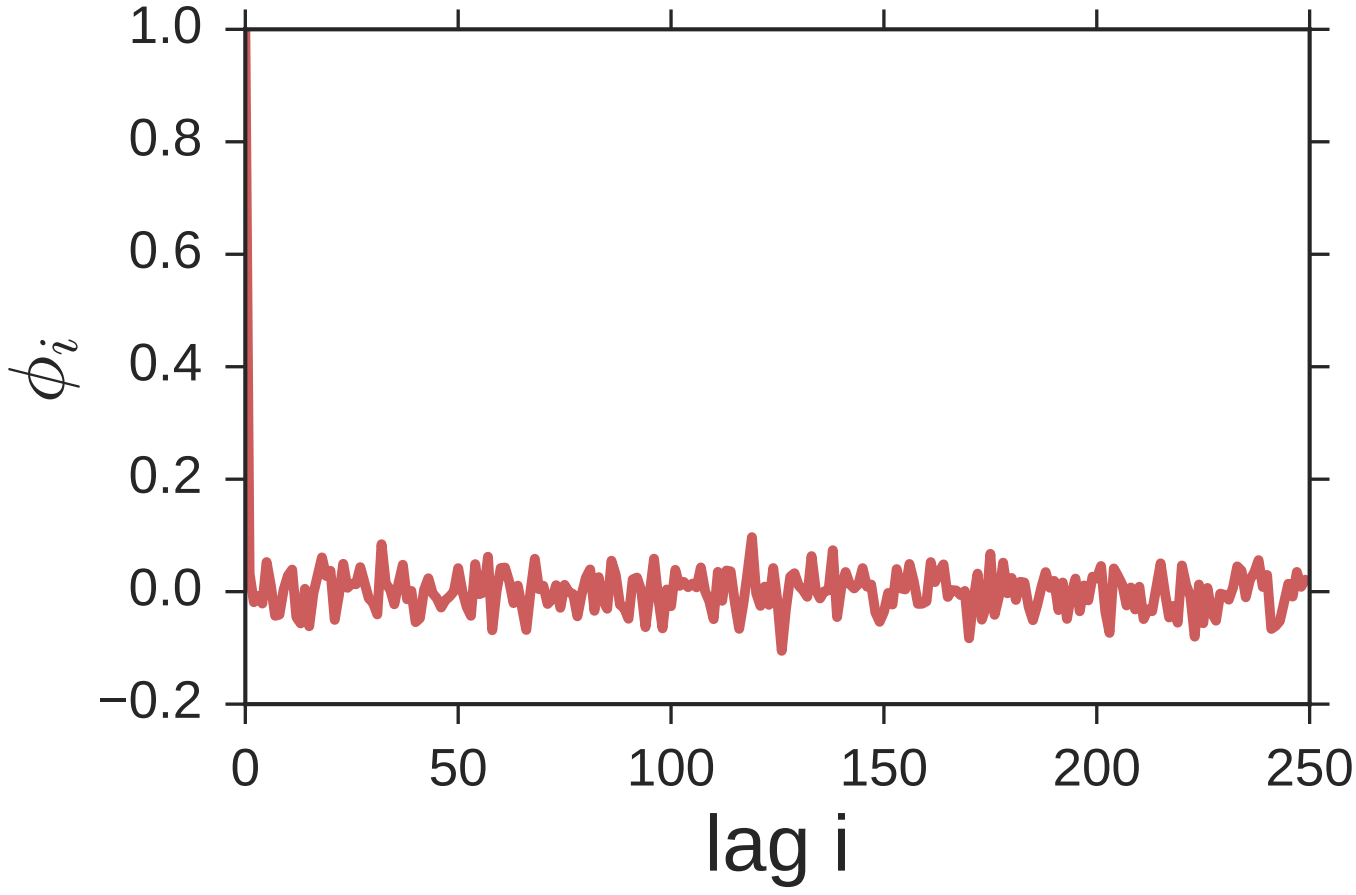
$$\phi_i = \frac{\langle A_i A_0 \rangle}{\langle A^2 \rangle}.$$

In computing the autocorrelation with lag i, we consider all sets of observations that are separated by a lag i. Thus,

$$\langle A_1 A_0 \rangle = \frac{\sum_{i=0}^{M-1} A_{i+1} A_i}{M} = \frac{A_1 A_0 + A_2 A_1 + ... + A_M A_{M-1}}{M}$$

ACF

```
ac = autocorr(x, M/4)
```



ACF: Independent Variables

Here, $\sigma_x \approx 0.59$ and $\langle x \rangle = 5.02 \pm 0.02$, where the standard deviation of the mean is $0.59/\sqrt{1000}$.

This gives a "2σ" range for $\langle x \rangle = 4.98 - 5.06$

We also see the autocorrelation function quickly decays from 1 to zero.

This is a hallmark of an uncorrelated process.

If successive samples were correlated, then the autocorrelation function takes its own sweet time to fall to zero (as we shall see in an example shortly).

The amount of time it takes to fall to zero or "decorrelate" is a measure of how correlated the data is.

Correlated Variables

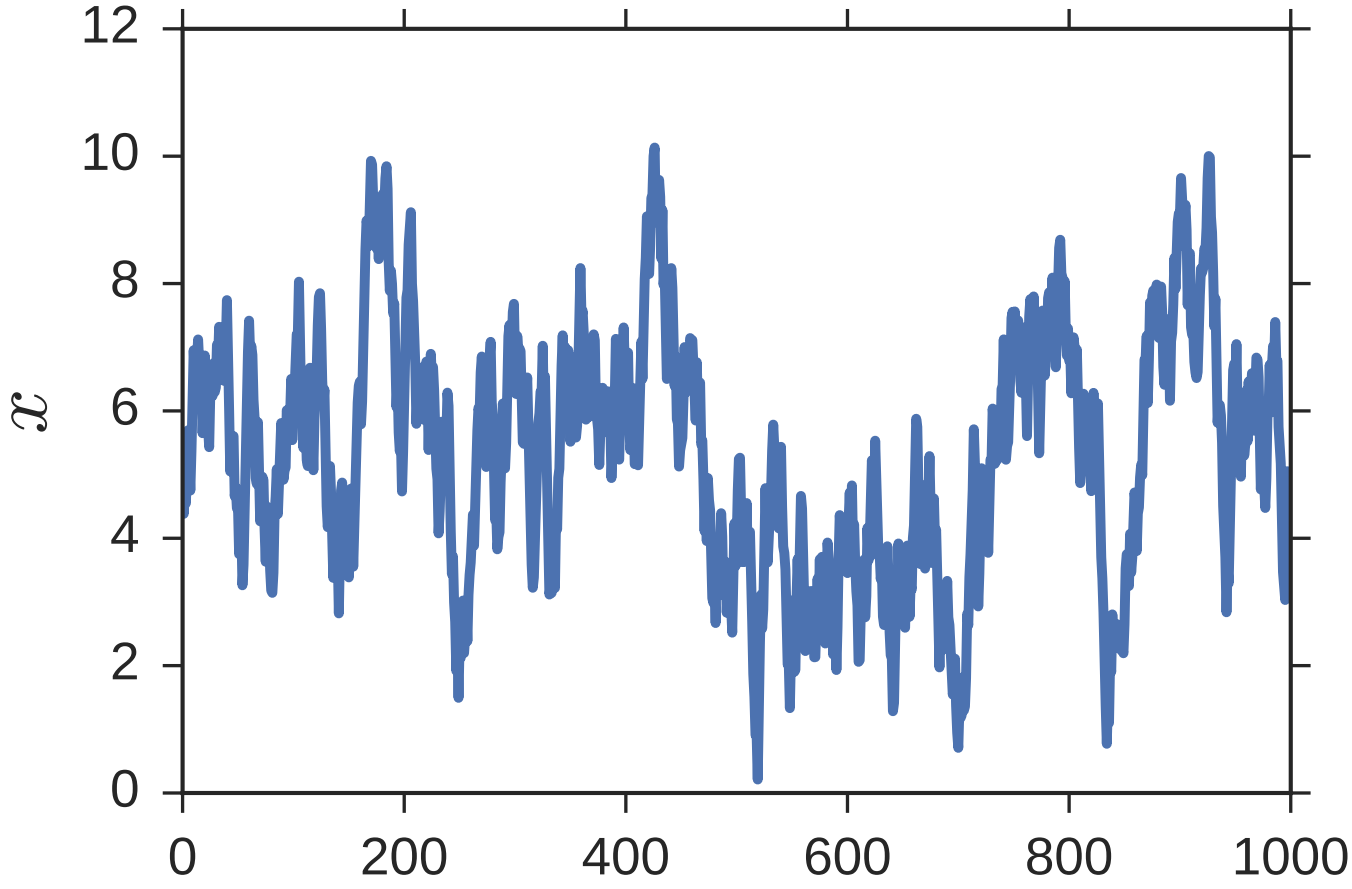
Now let's make up correlated process.

```
x[0] = 0.0
for i in range(1,M):
    x[i] = x[i-1] * 0.95 + np.random.uniform(-1, 1)

x = x + 5.0 * np.ones(x.shape)
```

This is an autoregressive model, but we don't have to know anything about that here.

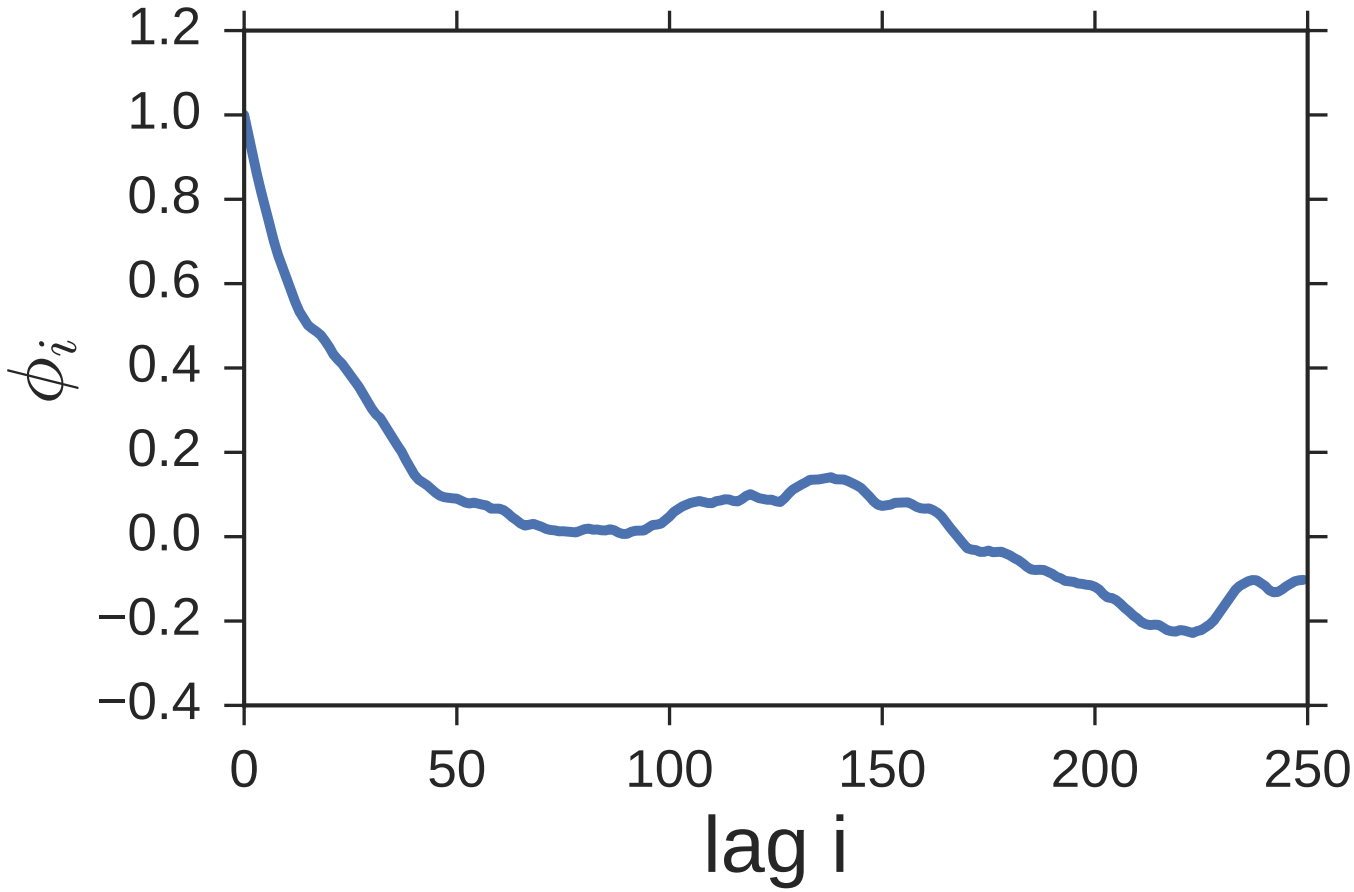
The time-series and its autocorrelation function look as follows:

Correlated Variables



The series fluctuates around "5" in a qualitatively different manner from the uncorrelated series.

ACF



The ACF decays to 0 around $i = 50$.

Partial Summary

Here $\sigma_x = 1.92$, and naively using,

$$\sigma_{\langle x \rangle} = \sigma_x / \sqrt{N} \approx 0.06,$$

we get $\langle x \rangle = 5.36 \pm 0.12$.

This 2σ range for the mean between 5.24 and 5.48 is not the "correct" bound for the mean value of "5".

Thus, the take-home message is:

assuming samples to be uncorrelated, when they are actually correlated, leads you to assign errorbars that are smaller than they should be.

Block Averaging

The correct expression for error for correlated samples:

$$\sigma^2(\langle A \rangle) = \frac{\sigma^2(A)}{M}\left[1 + 2\sum_i (1 - i/M)\phi_i\right],$$

Note the additional term in square brackets, which goes to 1, when $\phi \approx 0$ (uncorrelated variables).

Detailed error analysis of MCMC samples would require us to estimate the autocorrelation function

The estimated autocorrelation is itself noisy, which further needs to be accounted for.

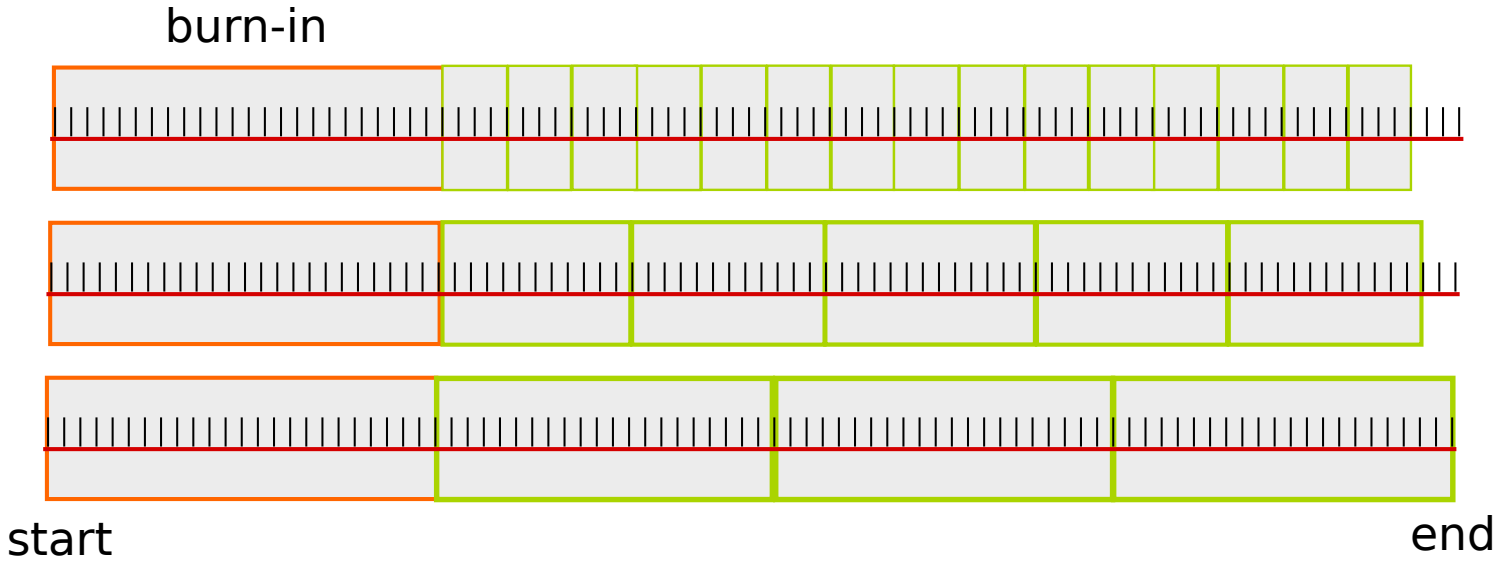
I recommend a simpler practical method called **block averaging**.

Block Averaging

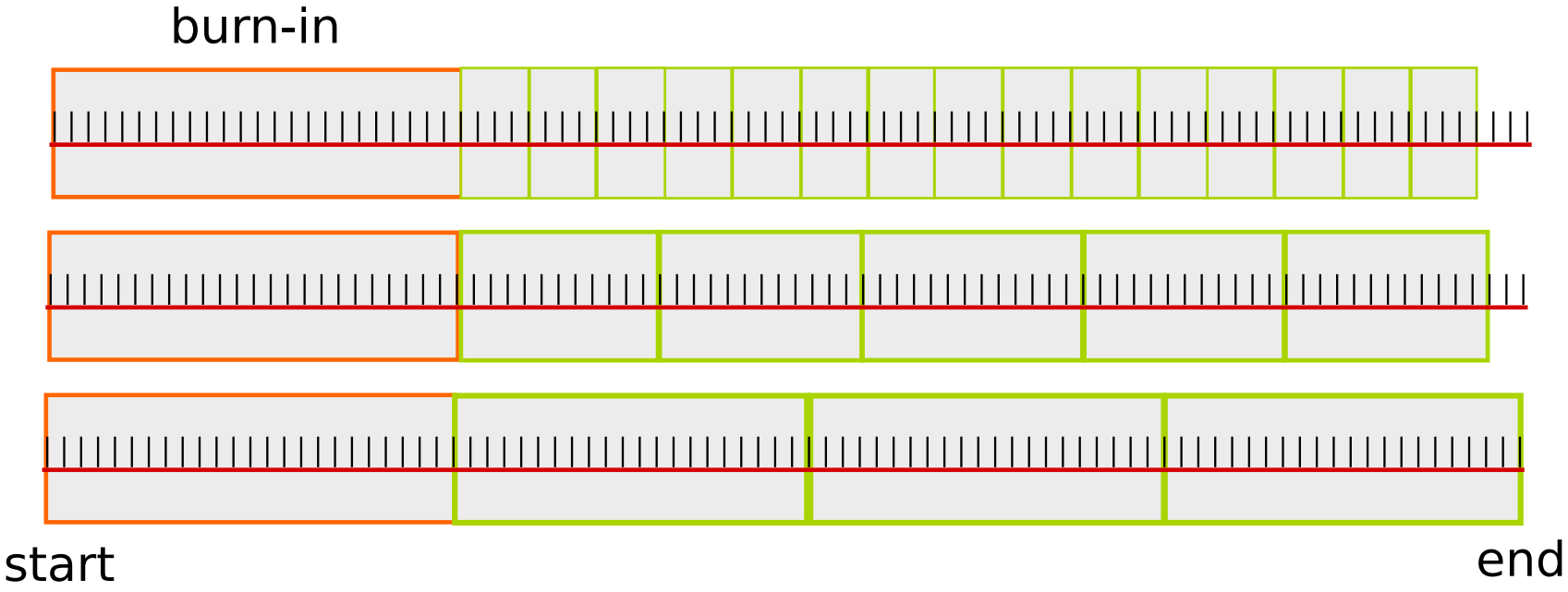
In block averaging, we discard the burn-in period, and retain M correlated samples.

We chop this portion of the dataseries into N_b blocks of different sizes b so that $bN_b \approx M$.

Unused datapoints near the end; don't matter much.



Block Averaging



Top row of the schematic $b = 4$, and $N_b = 15$

Last row $b = 21$, and $N_b = 3$.

Note that $bN_b \approx M = 63$.

Algorithm

1. Define blocks of length $b = 1, 2, 3, \ldots, M$. The upper limit defined by the length of the simulation run.
2. For each block length b, let
   - $\bar{A}_j$, $j = 1, ..., N_b$ be the block average of the j^{th} block.
   - $\langle A \rangle_b$ be the mean of the $\bar{A}_j$s.
3. Compute the variance estimator:

$$\sigma^2_{\langle A \rangle_b} = \frac{1}{N_b - 1} \sum_{j=1}^{N_b} \bar{A}_j^2 - \langle A \rangle_b^2.$$

4. Plot $\sigma^2_{\langle A \rangle_b}$ versus b. The plot initially rises, and then reaches a plateau.

The plateau value is your best estimate of the true variance.

Block Averaging: Why it works?

If the A_i are correlated, $\bar{A}_j$ block averages will be decreasingly correlated as block size increases

That is the $\bar{A}_j$s for $b = 2$ are going to be more correlated than the $\bar{A}_j$ for $b = 100$, where $j = 1, 2, ...N_b$.

Once, the block size exceeds a (unknown) correlation time present in the data, block averages will be independent

Need to find the minimum block length for which this happens

- too short: no improvement over correlated data
- too long: small number of blocks available (unreliable variance)

Correlated Example

Recall the autoregressive model

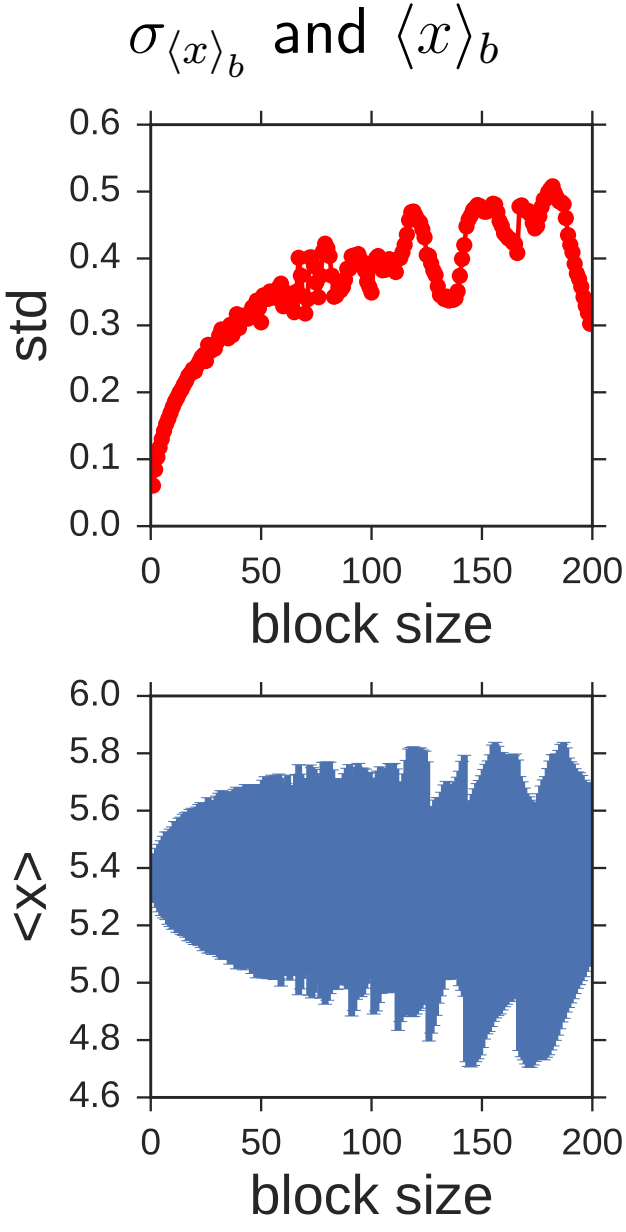
```
x[0] = 0.0
for i in range(1,M):
    x[i] = x[i-1] * 0.95 + np.random.uniform(-1, 1)

x = x + 5.0 * np.ones(x.shape)

v, s, m = blockAverage(x, True, 100)
```

The program yields:

$$\langle x \rangle = 5.38 \pm 0.36,$$

which correctly reflects the lower certainty of correlated variables.

Correlated Example



Appendix

1. Multidimensional and Bivariate Normal

The multivariate normal distribution has the form:

$$f_{\mathbf{x}}(x_1, \ldots, x_k) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right),$$

where $\mathbf{x}$ is a k-dimensional column vector and $|\boldsymbol{\Sigma}|$ is the determinant of the symmetric covariance matrix $\boldsymbol{\Sigma}$.

When $k = 2$, we have a bivariate normal distribution, that we saw earlier.

General 2D normal or Gaussian distribution:

$$f(x, y) = A \exp\left(-BC\right)$$

where,

$$
\begin{aligned}
A &= \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \\
B &= \frac{1}{2(1-\rho^2)} \\
C &= \frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}
\end{aligned}
$$

ρ is the correlation coefficient

The relationship between the "matrix" and "explicit" forms for 2D normal distribution is described via:

$$
\boldsymbol{\mu} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}.
$$