Error analysis

Giovanni Bussi
Physics and Chemistry of Biological Systems
SISSA, Trieste, Italy

bussi@sissa.it
http://people.sissa.it/~bussi
@bussilab

Slides (will be) available

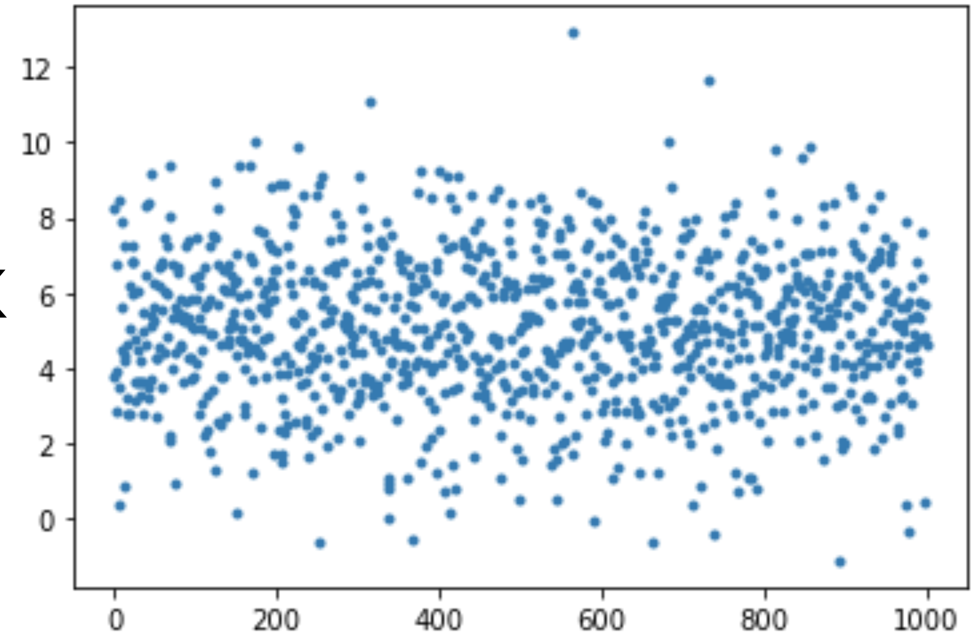
github.com/plumed/lugano2019

(I can share pictures of Max at a moderate price)

Two time series, same distribution

uncorrelated

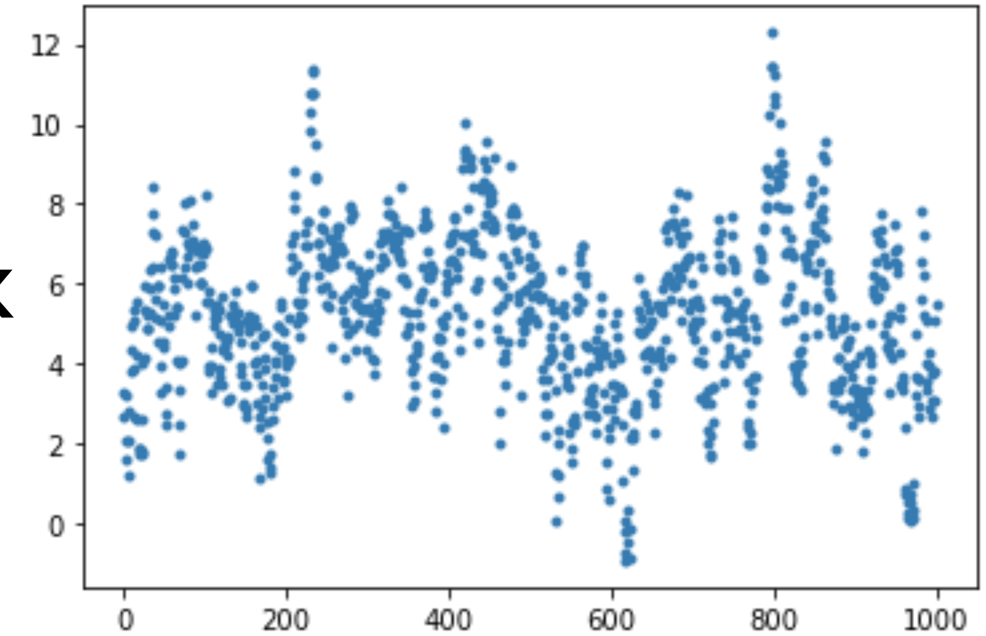
x



time

correlated

x



time

Both: average=5 stddev=2

uncorrelated: each sample is new
correlated: each sample is a mixture of old and new

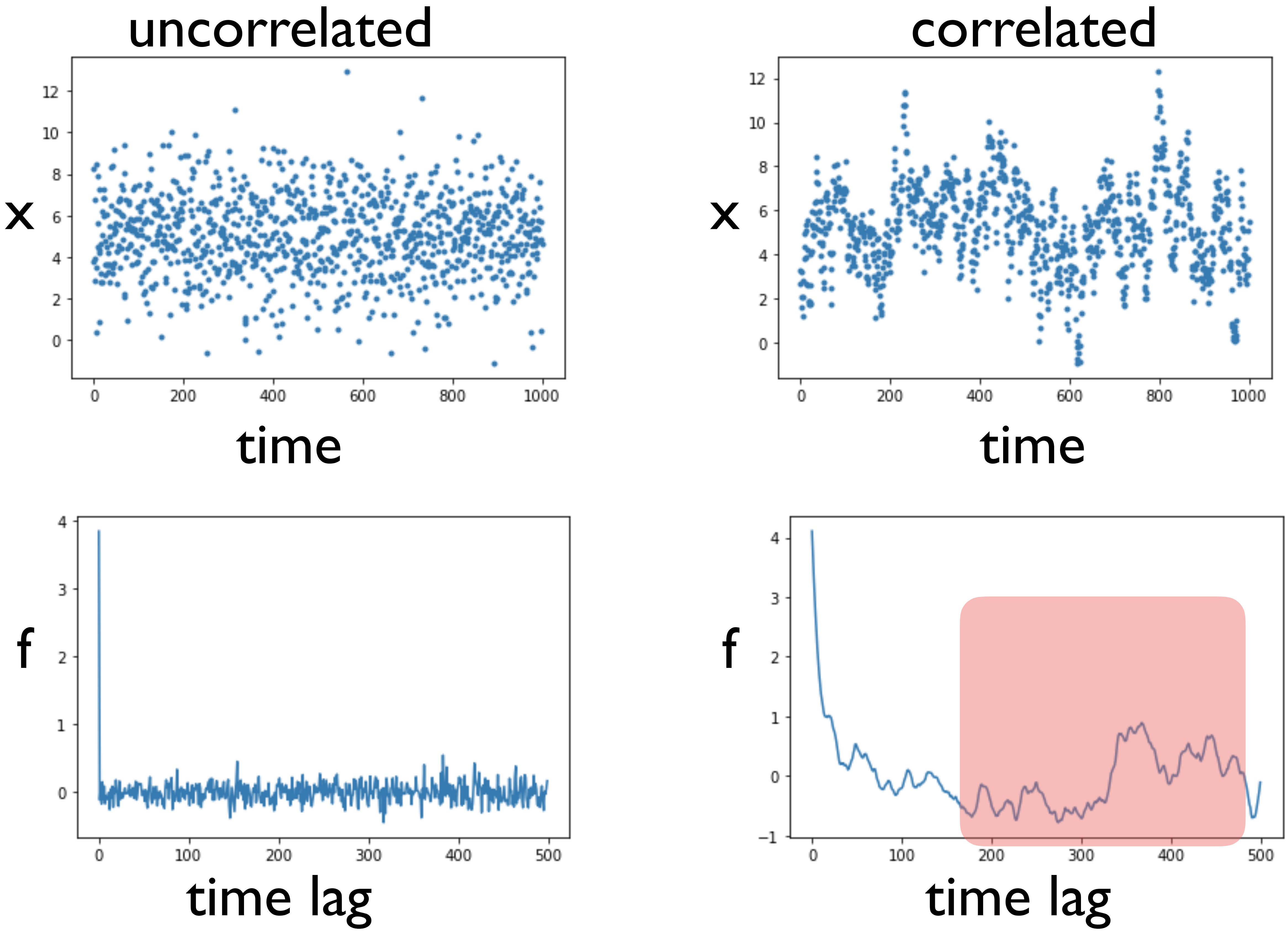
In practice

```
timeseries_uncorr=stddev*np.random.normal(size=nframes)+average
plt.plot(timeseries_uncorr,".")
plt.show()
```

```
timeseries_corr=[]
x=stddev*np.random.normal()+average
c1=np.exp(-0.1)
for i in range(nframes):
    xnew=stddev*np.random.normal()+average
    x=c1*(x-average)+np.sqrt(1-c1**2)*(xnew-average) + average
    timeseries_corr.append(x)
timeseries_corr=np.array(timeseries_corr)
plt.plot(timeseries_corr,".")
plt.show()
```

github.com/plumed/lugano2019/slides/Error Analysis.ipynb

Autocorrelation function



In practice

```
def autocorr(series):
    autoc=[]
    series=+np.array(series) # take a copy
    series-=np.average(series)
    autoc.append(np.average(series*series))
    for lag in range(1,int(len(series)/2)):
        autoc.append(np.average(series[:-lag]*series[lag:]))
    return np.array(autoc)
```

NB: there are better ways to compute autocorrelation functions!

Poor man's error analysis

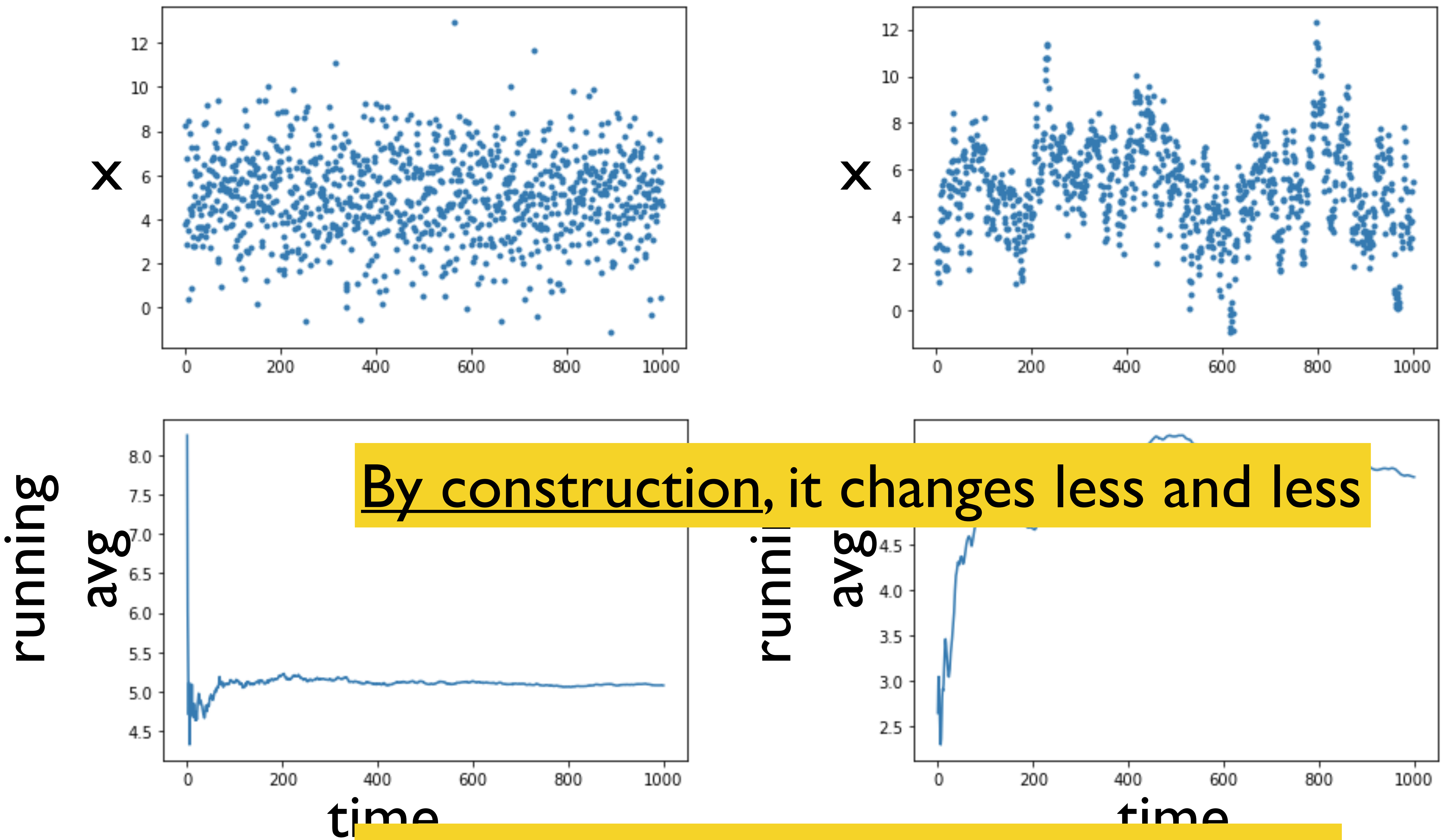
Repeat the simulation with different initial condition (as different as possible)

Repeat the simulation starting from <u>the end</u> of the first simulation (the most different state wrt the initial one)

Realize that this is the same as splitting a long simulation in 2 blocks.

Generalize to more than 2 blocks

Running average (don't do this!)



By construction, it changes less and less

Don't compare first half with whole!
Do compare first half with second half

Some equations

subtract average

$$a = x - \langle x \rangle$$

compute time-lagged correlations

$$f(t) = \langle a(0)a(t) \rangle$$

$$f(t) = \frac{1}{T}\int_0^T dt' a(t')a(t'+t)$$

normalize

$$f_n(t) = \frac{\langle a(0)a(t) \rangle}{\langle a^2 \rangle}$$

autocorrelation time

$$\tau_C = \int_0^\infty dt\, f_n(t)$$

Error of the average

average from a simulation of length T

$$a_T = \frac{1}{T}\int_0^T dt a(t)$$

typical deviation of a simulation of length T

$$\langle a_T^2\rangle = \frac{1}{T^2}\int_0^T dt \int_0^T dt' \langle a(t)a(t')\rangle = \frac{1}{T^2}\int_0^T dt \int_0^T dt' f(t-t')$$

Change into $x = \frac{t+t'}{2}$ and $y = t - t'$

$$= \frac{1}{T^2}\int_{-T}^T dy \int_{|y|/2}^{T-|y|/2} dx f(y)$$

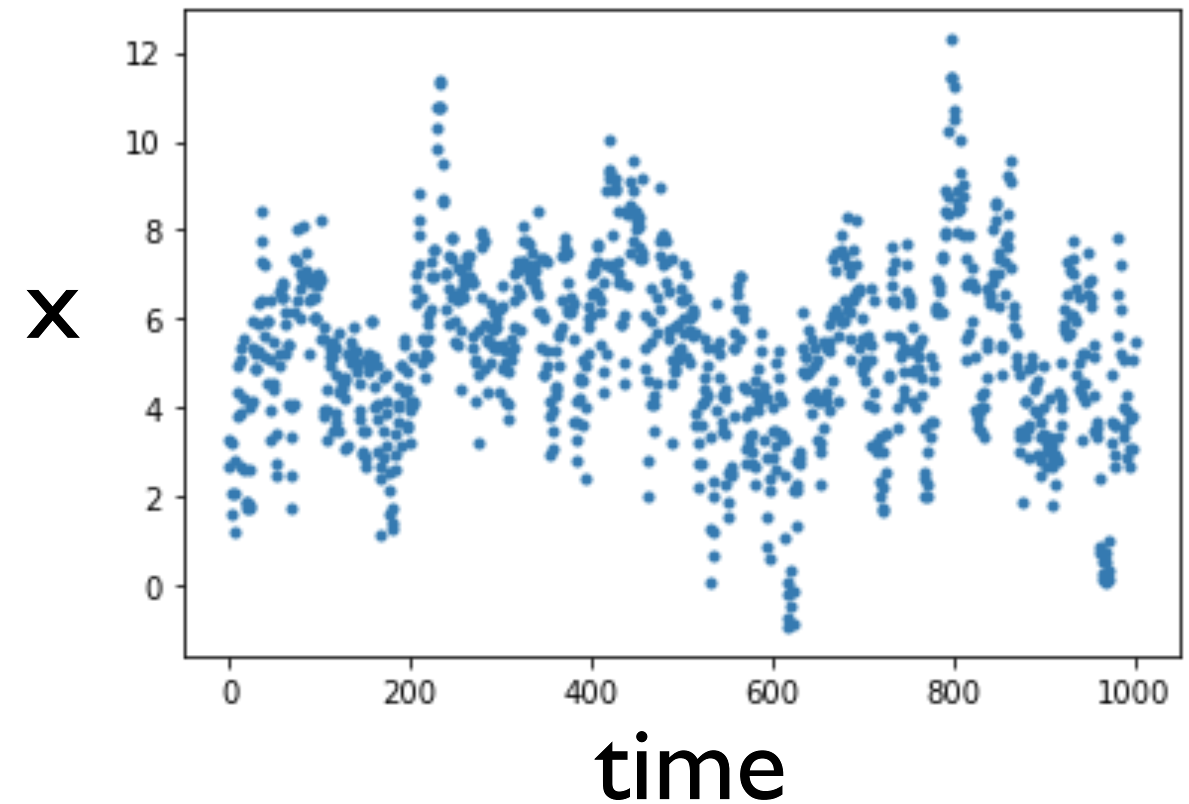
(determinant of the Jacobian is 1).

$$= \frac{\sigma^2}{T}\int_{-T}^T dy f_n(y)\left(1 - \frac{|y|}{T}\right)$$

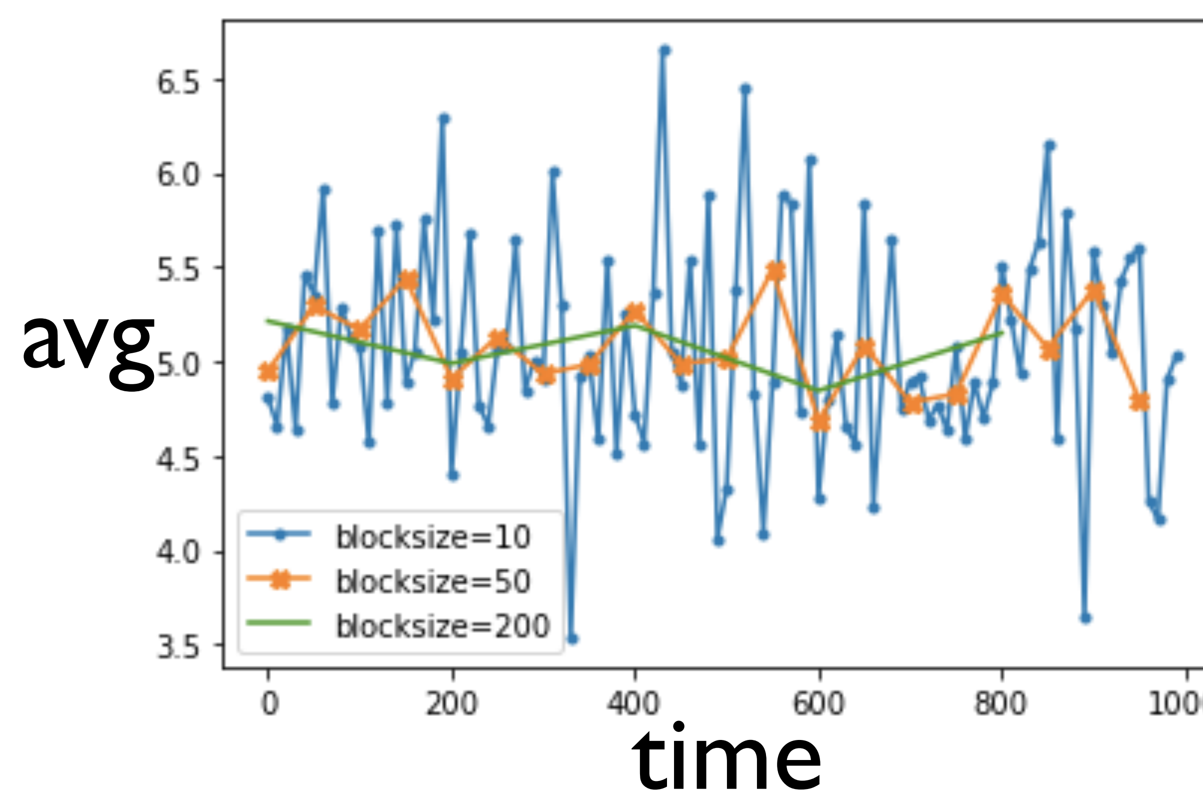
(for large T) $$= \frac{2\sigma^2\tau_C}{T}$$

Block averages

divide in N blocks
compute avg for each block



$$\sigma_N^2 = \frac{1}{N-1}\sum_i (a_i - \bar{a})^2$$



In practice

```
def plot_blocks(series):
    blocksize=10
    blocks_uncorr=[]
    x=[]
    for i in range(int(len(series)/blocksize)):
        x.append(i*blocksize)
        blocks_uncorr.append(np.average(series[i*blocksize:(i+1)*blocksize]))
    plt.plot(x,blocks_uncorr,".-",label="blocksize="+str(blocksize))
    blocksize=50
    blocks_uncorr=[]
    x=[]
    for i in range(int(len(series)/blocksize)):
        x.append(i*blocksize)
        blocks_uncorr.append(np.average(series[i*blocksize:(i+1)*blocksize]))
    plt.plot(x,blocks_uncorr,"X-",label="blocksize="+str(blocksize))
    blocksize=200
    blocks_uncorr=[]
    x=[]
    for i in range(int(len(series)/blocksize)):
        x.append(i*blocksize)
        blocks_uncorr.append(np.average(series[i*blocksize:(i+1)*blocksize]))
    plt.plot(x,blocks_uncorr,"-",label="blocksize="+str(blocksize))
    plt.legend()
    plt.show()
```

Error from the whole simulation

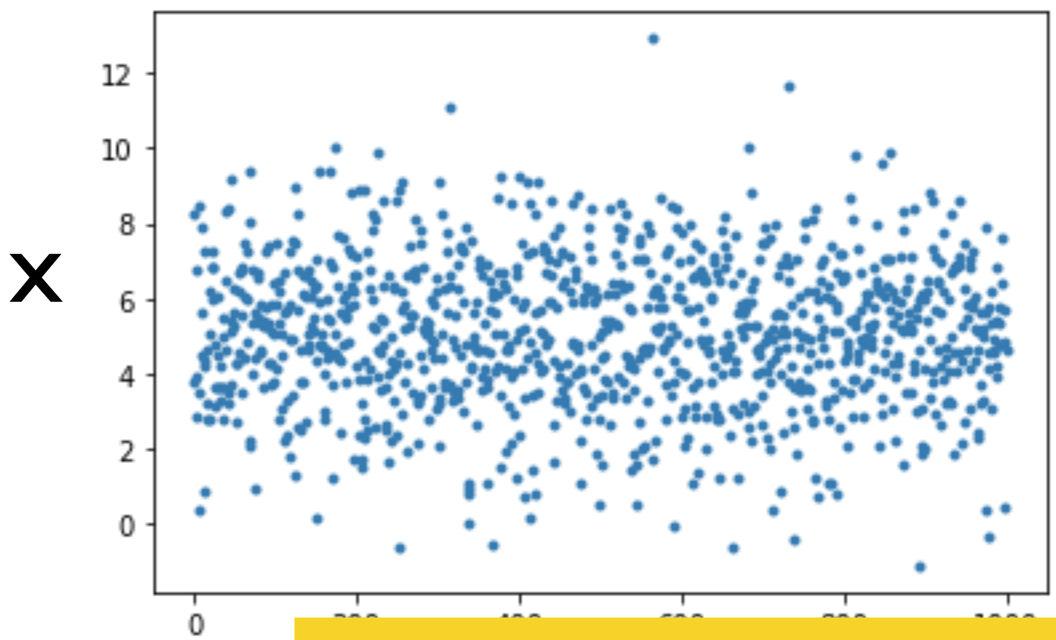
Error of each block is estimated from stddev between block averages

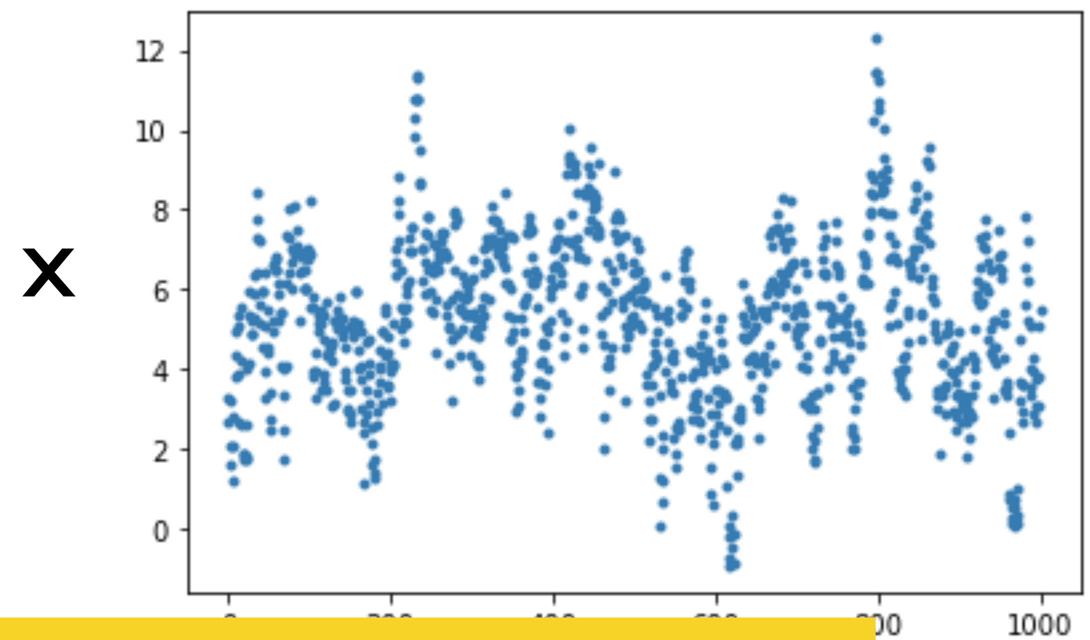
Error of the whole sim: error of the block / sqrt(N_{blocks}-1) *

```
def analyze(timeseries,blocksize):
    nblocks=int(len(timeseries)/blocksize)
    blockav=[]
    for i in range(nblocks):
        blockav.append(np.average(timeseries[i*blocksize:(i+1)*blocksize]))
    return (np.average(blockav),np.std(blockav)/np.sqrt(nblocks-1))
```

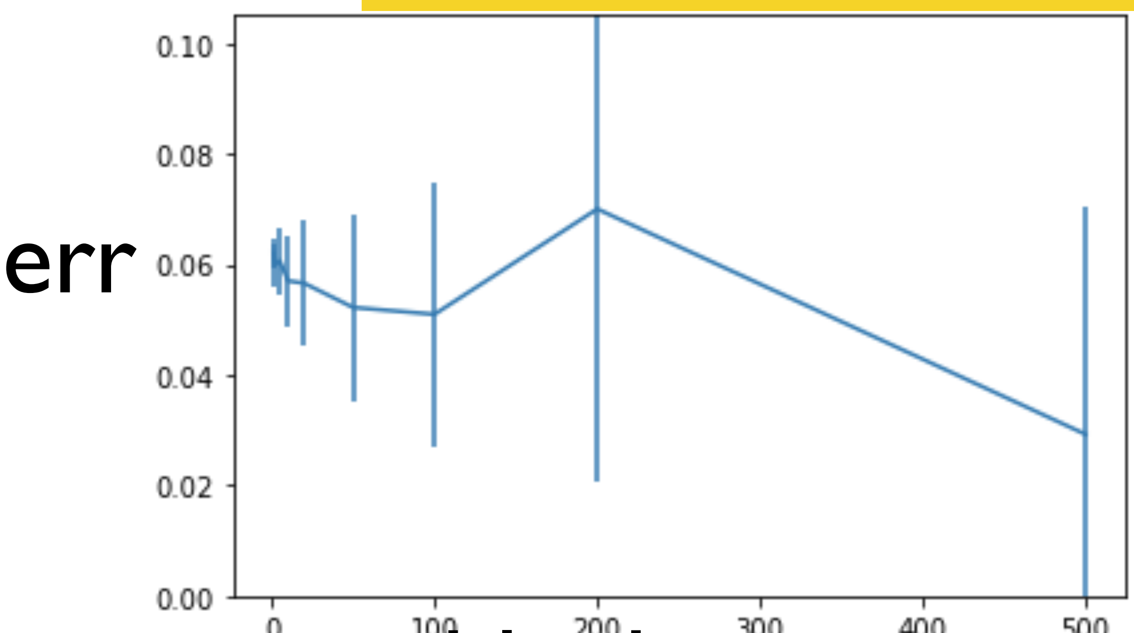
* N-1 gives an unbiased estimator for the variance

Error estimate vs block size



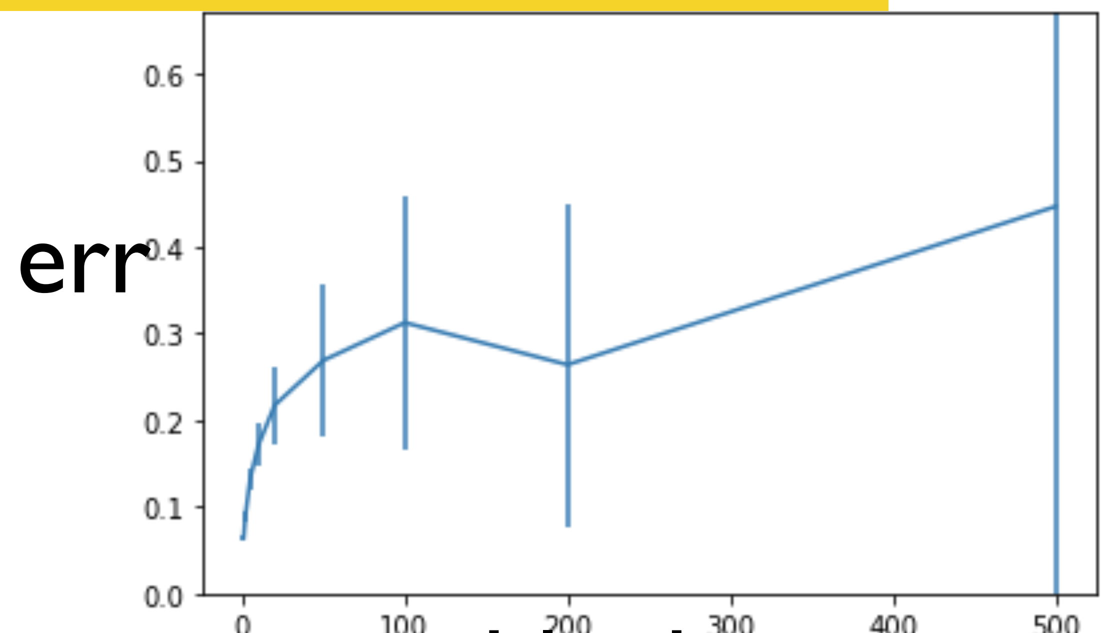


Note: Blocksize=1 is like not doing blocks



blocksize

av=5.07



← blocksize →

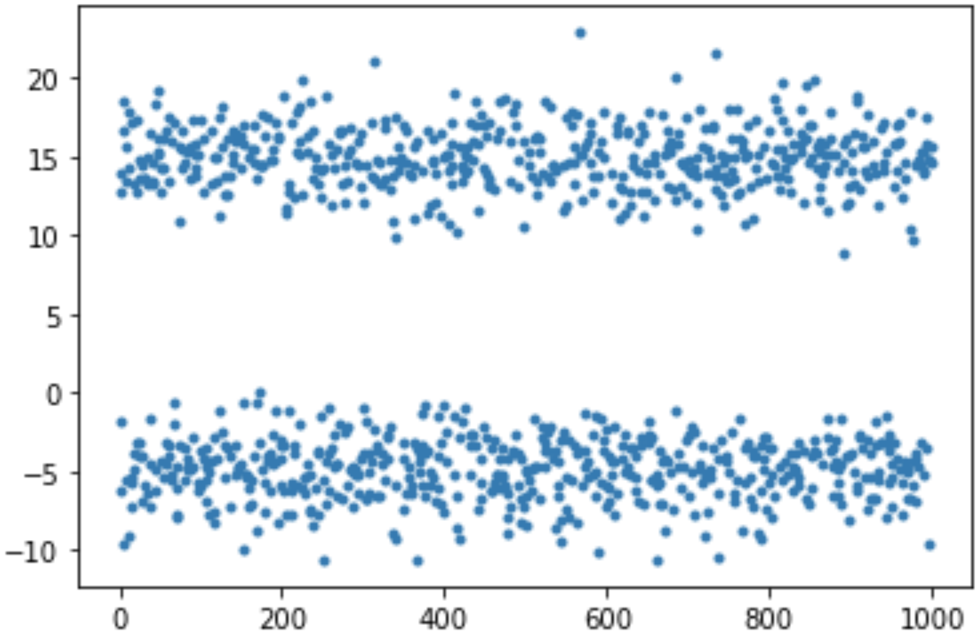
underestimate stat. error

av=5.24

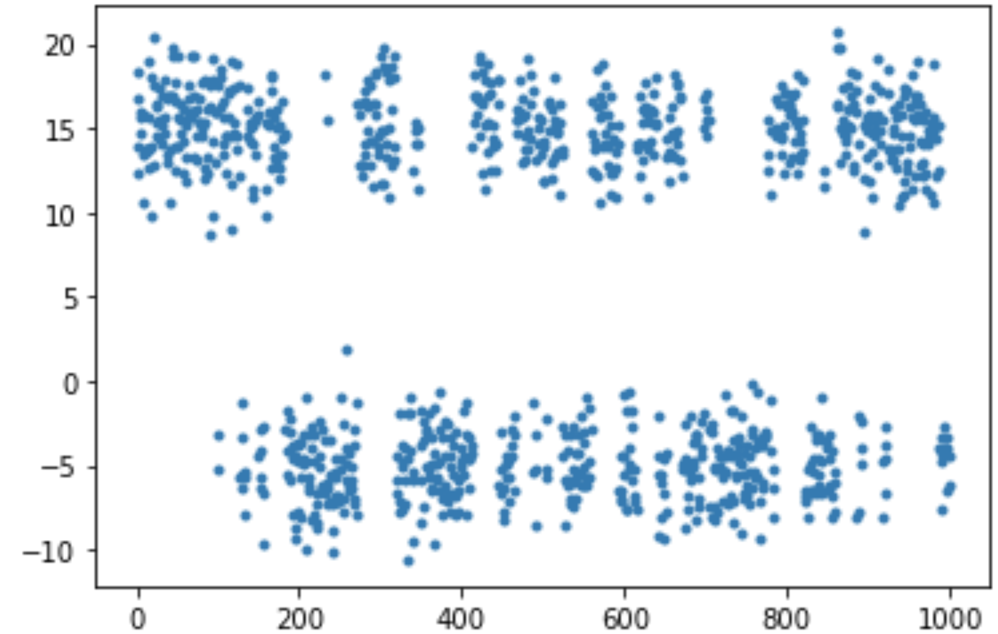
$av_{exact}=5$

Metastable states

uncorrelated



correlated



- Half of the points: Gaussian with av 15 and stddev 2
- Half of the points: Gaussian with av -5 and stddev 2

- Uncorrelated: each point is independent of the others
- Correlated: basin changes with prob 1/20 at each step

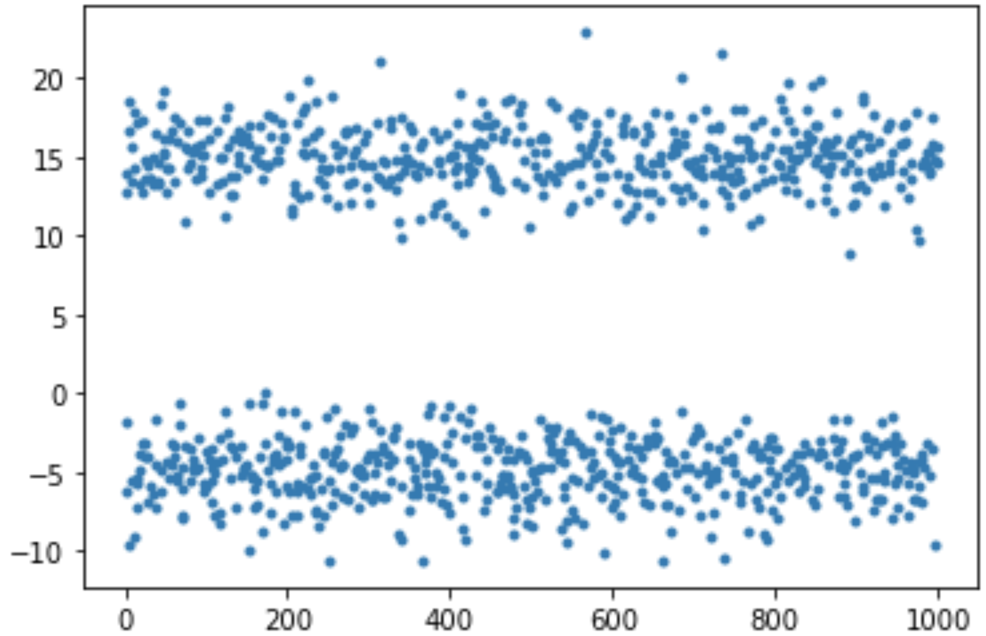
In practice

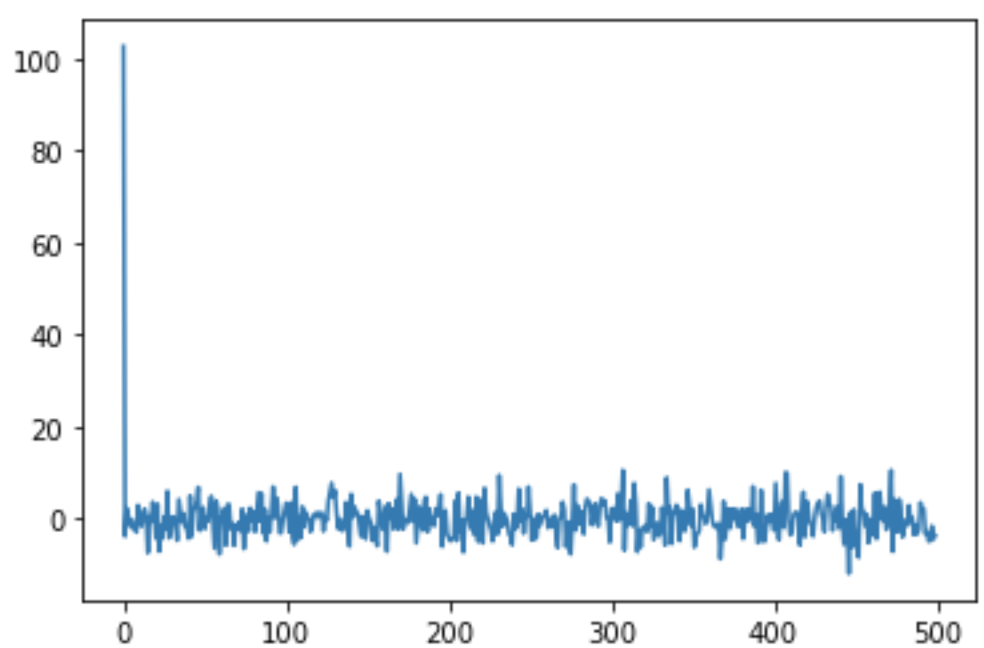
```
timeseries_easy=stddev*np.random.normal(size=nframes)+average + (np.random.randint(2,size=nframes)-0.5)*shift
```

```
tauc=10
shift=20
timeseries_difficult=stddev*np.random.normal(size=nframes)+average
x=-0.5
for i in range(nframes):
    if(np.random.uniform()<1.0/tauc) and np.random.uniform()>0.5: x=-x
    timeseries_difficult[i]+=x*shift
```

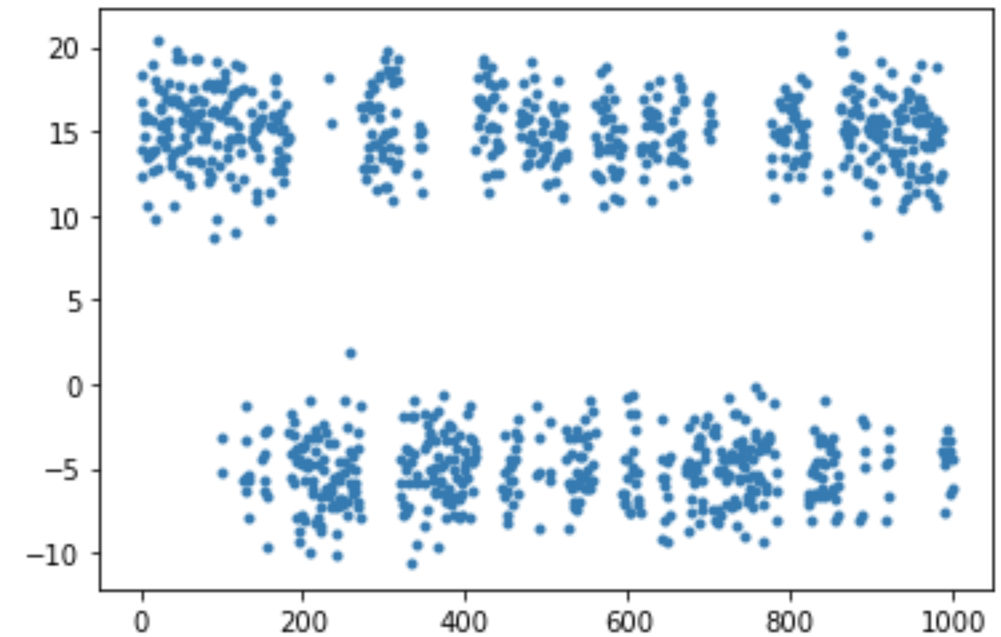
Autocorrelation function

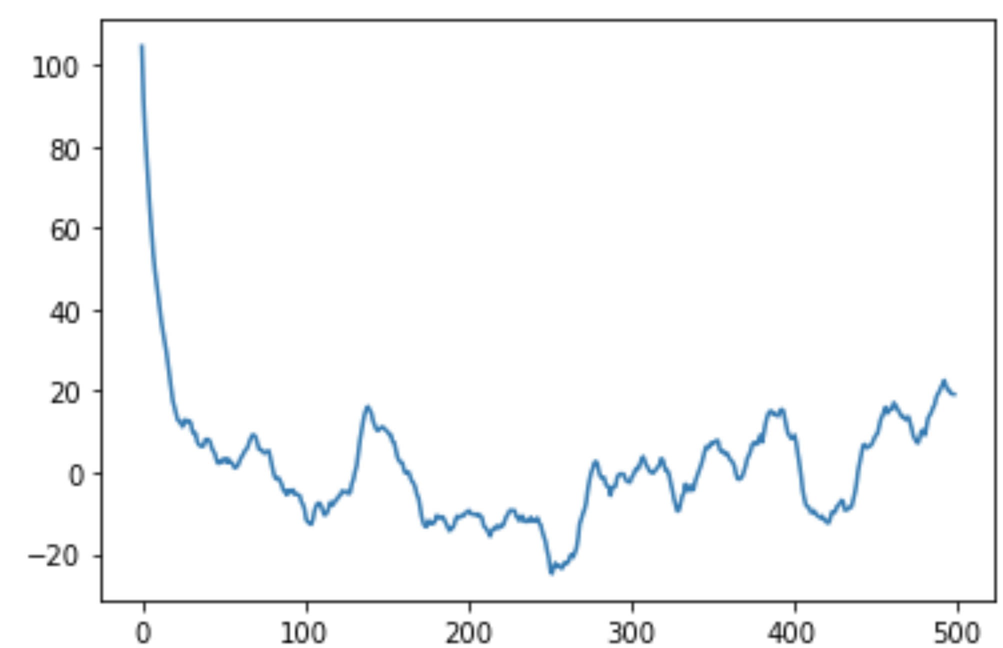
uncorrelated





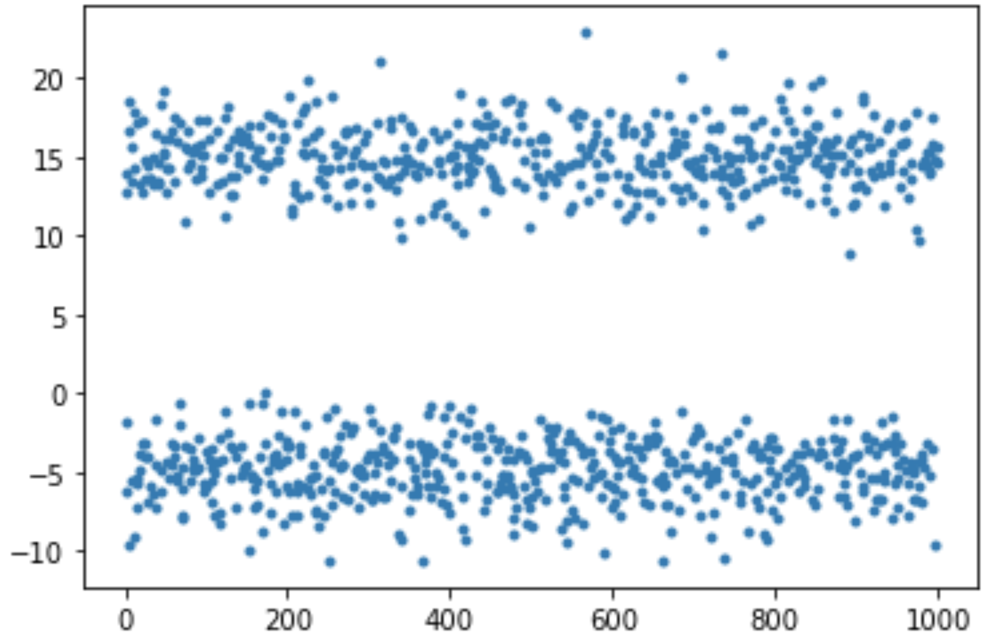
correlated



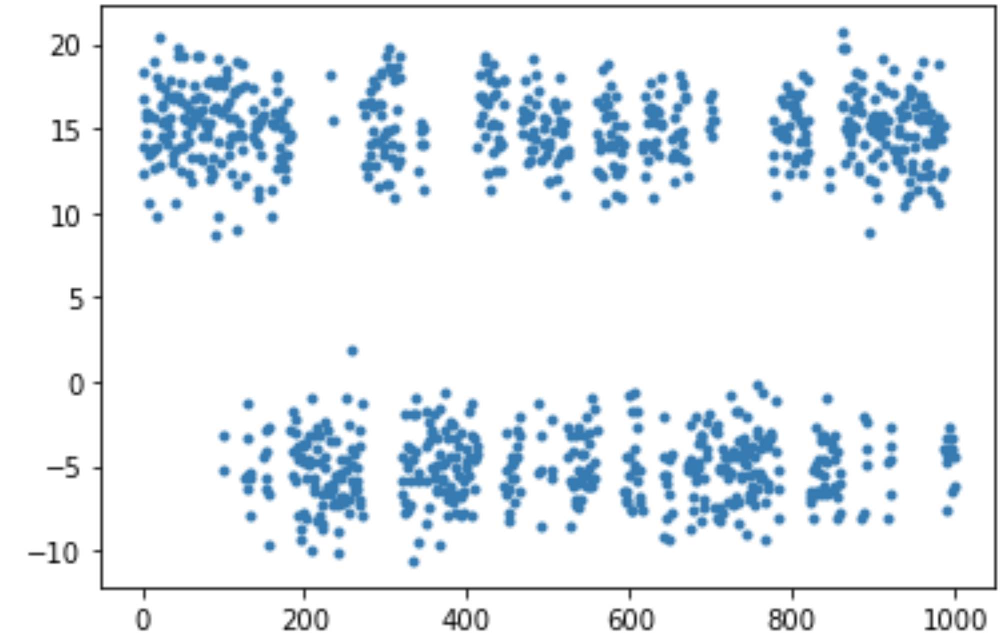


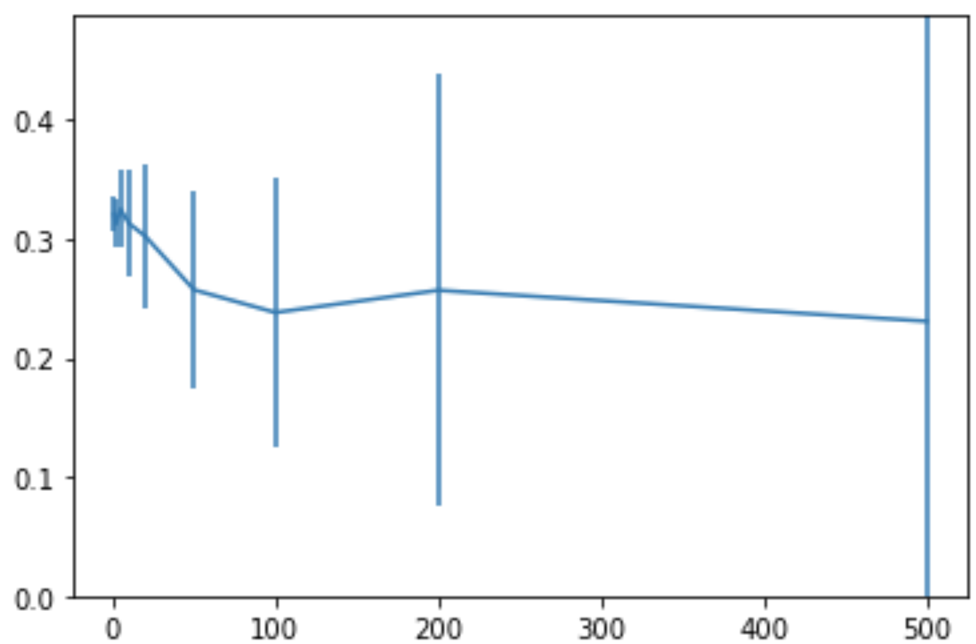
Block analysis

uncorrelated

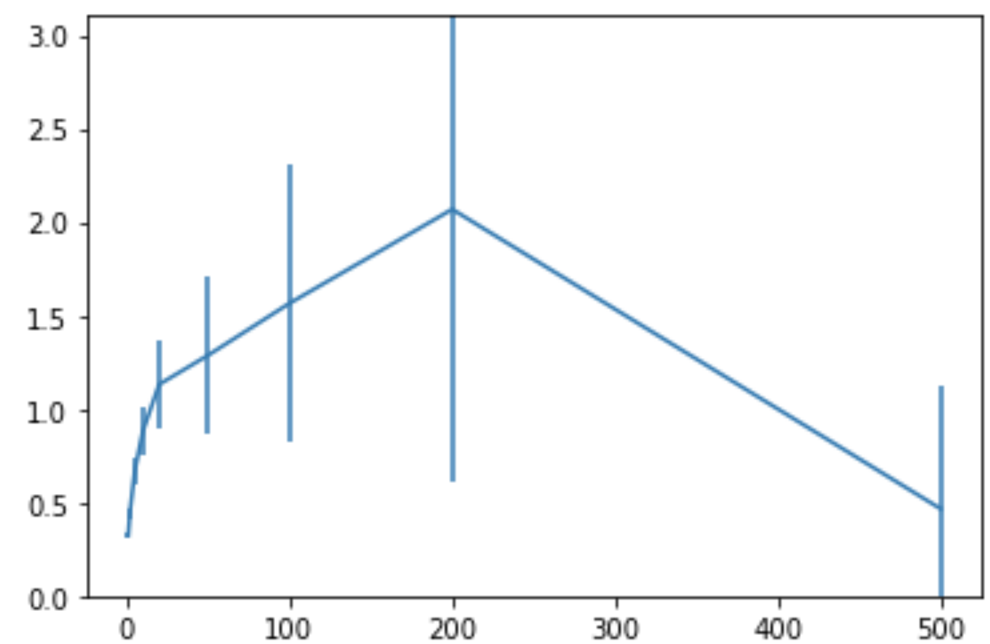


correlated





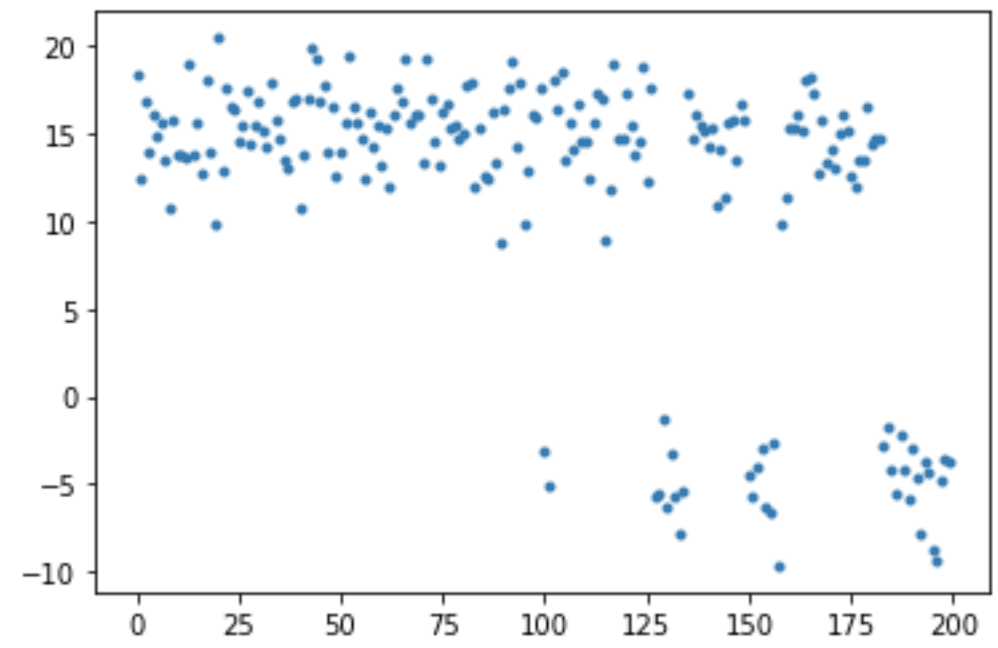
av=4.86



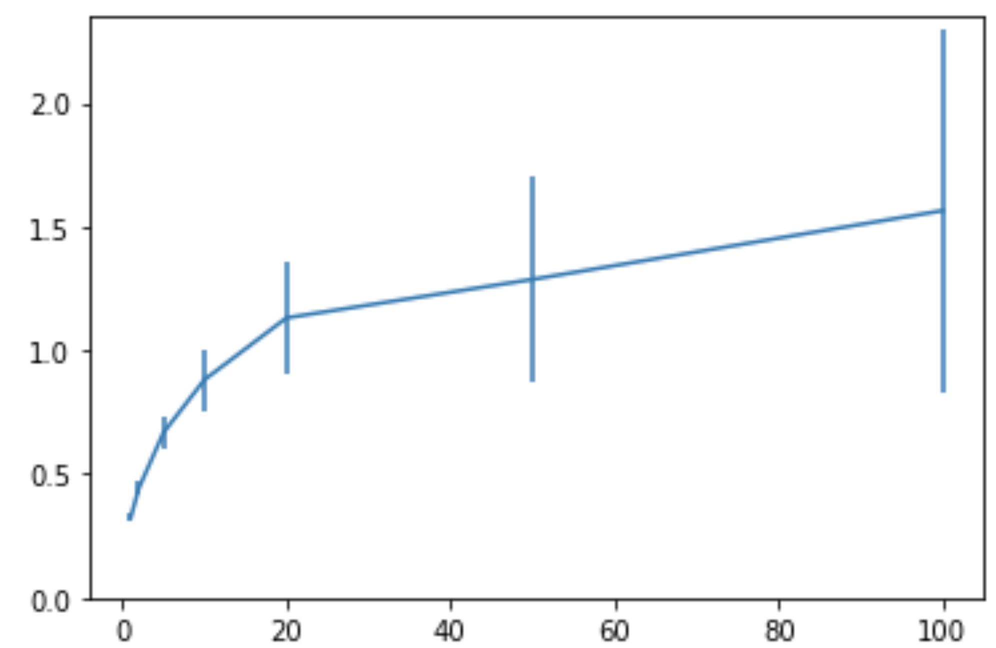
av=5.93

Even more difficult (first 200 frames)

Check your time series!



Block error does not converge



Summary of block analysis

Two conceptually separate steps:

1. Blocking
2. Error of blocks (as if they were independent)

Then, check robustness with block size.

As a quick alternative: pick a small number of blocks (e.g. 3 to 5) and cross your fingers

Error estimates on averages of correlated data

H. Flyvbjerg
The Niels Bohr Institute, Blegdamsvej 17, DK-2100 Copenhagen Ø, Denmark

H. G. Petersen
Institute of Mathematics and Computer Science, Odense University, DK-5230 Odense M, Denmark

(Received 8 February 1989; accepted 14 March 1989)

We describe how the true statistical error on an average of correlated data can be obtained with ease and efficiency by a renormalization group method. The method is illustrated with numerical and analytical examples, having finite as well as infinite range correlations.

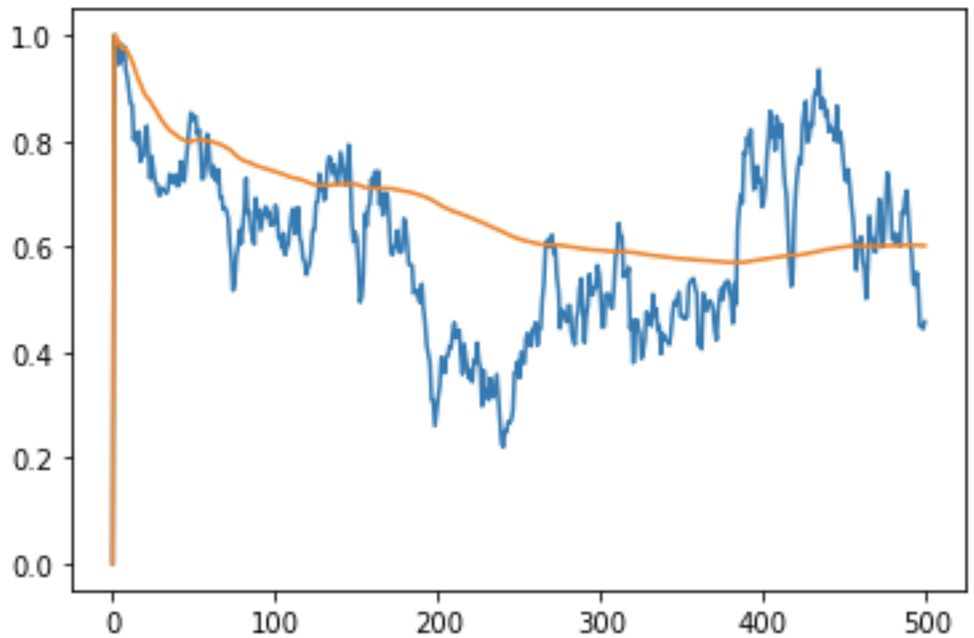
H. Flyvbjerg, H.G. Petersen, J. Chem. Phys. 91, 461 (1989).

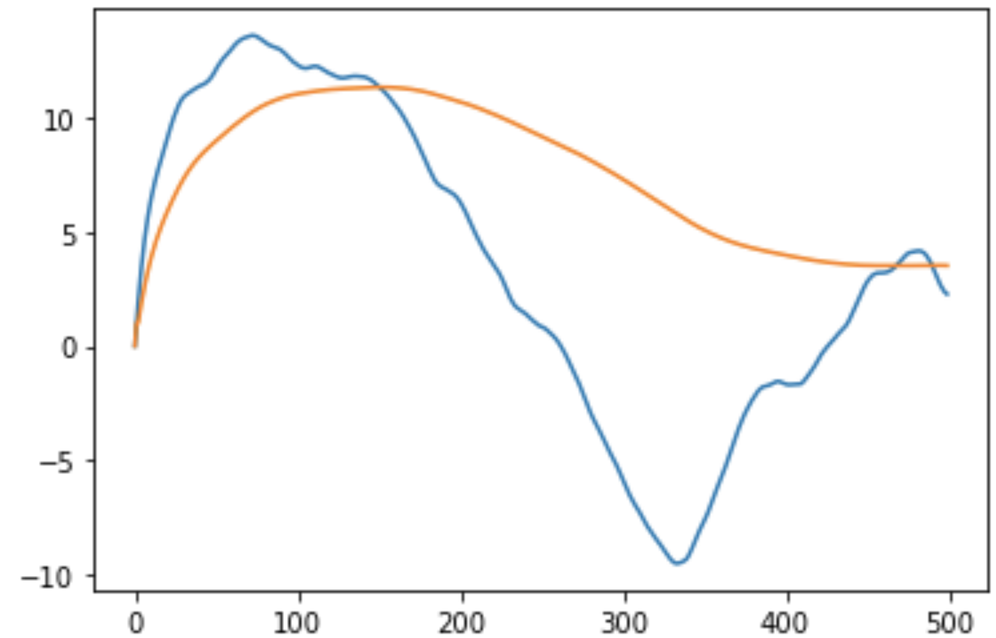
More on estimating autocorrelation time

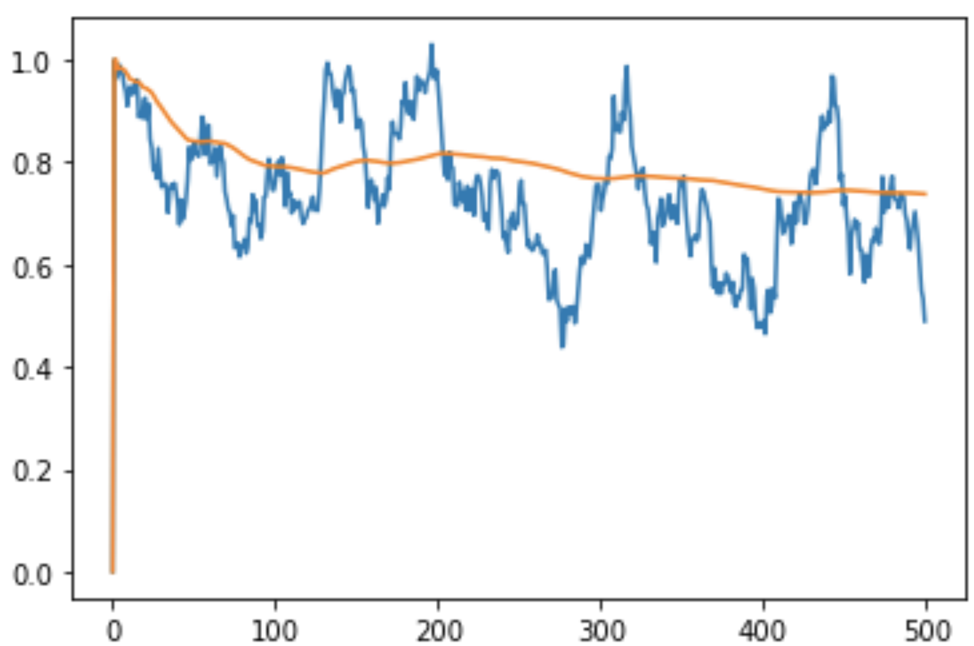
$$= \frac{\sigma^2}{T}\int_{-T}^{T} dy f_n(y)\left(1 - \frac{|y|}{T}\right) = \frac{2\sigma^2\tau_C}{T}$$

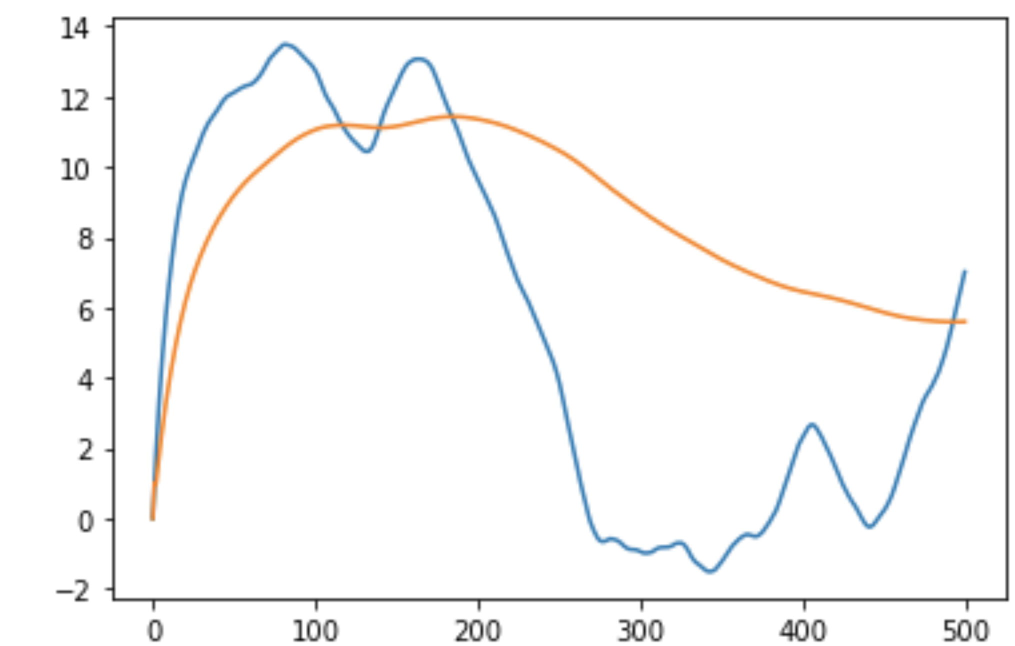
$$\tau_C = \int_0^T dt f_n(t)$$

$$\tau_C = \int_0^T dt f_n(t)\left(1 - \frac{t}{T}\right)$$









In practice

```
def autocorr(series):
    autoc=[]
    series=+np.array(series) # take a copy
    series-=np.average(series)
    autoc.append(np.average(series*series))
    for lag in range(1,int(len(series)/2)):
        autoc.append(np.average(series[:-lag]*series[lag:]))
    return np.array(autoc)
```

```
def compare_autoc_time(series):
    ac=autocorr(series)
    p=[]
    for i in range(len(ac)):
        p.append(np.sum(ac[:i])/ac[0])
    p=np.array(p)
    plt.plot(p)
    p=[]
    for i in range(len(ac)):
        p.append(np.sum(ac[:i] * np.linspace(1,0,i))/ac[0])
    p=np.array(p)
    plt.plot(p)
```

Alternative methods for error estimation

Jack-knife

Use all subsets of N_{blocks}-1 blocks
Take their stddev and multiply it times sqrt(N_{blocks}-1)

Bootstrap

Use random extractions of N_{blocks}
Take their stddev and multiply it times sqrt(N_{blocks}/(N_{blocks}-1))

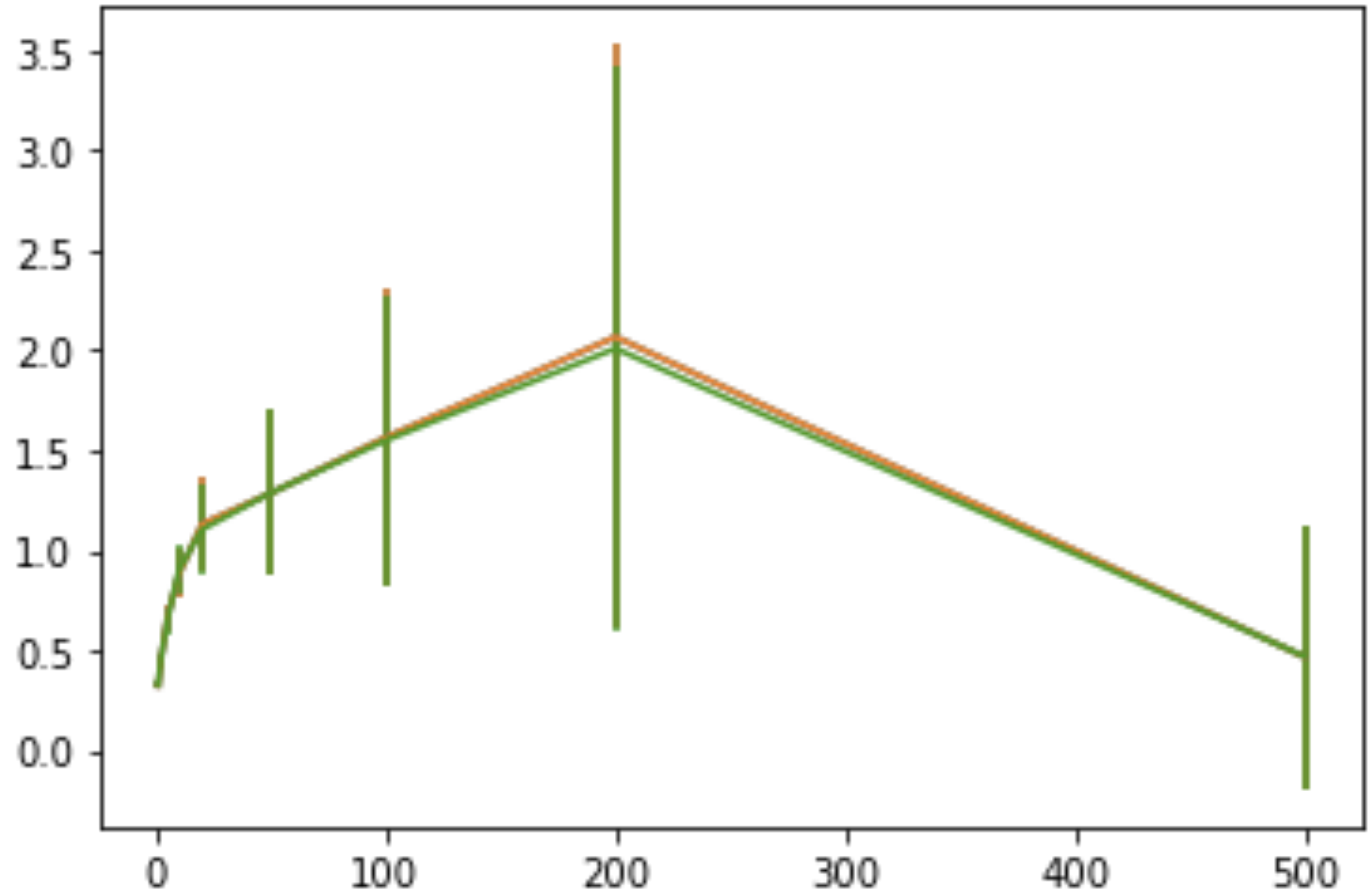
https://en.wikipedia.org/wiki/Jackknife_resampling
https://en.wikipedia.org/wiki/Bootstrapping_(statistics)

In practice:

```
def analyze(timeseries,blocksize):
    nblocks=int(len(timeseries)/blocksize)
    blockav=[]
    for i in range(nblocks):
        blockav.append(np.average(timeseries[i*blocksize:(i+1)*blocksize]))
    blockav=np.array(blockav)

    # error of the mean
    mean_error=(np.average(blockav),np.std(blockav)/np.sqrt(nblocks-1))

    # jack-knife
    jack=[]
    for i in range(nblocks):
        jack.append((np.sum(blockav)-blockav[i])/(nblocks-1))
    jack_error=(np.average(jack),np.std(jack)*np.sqrt(nblocks-1))

    # bootstrap
    boot=[]
    for i in range(1000):
        boot.append(np.average(blockav[np.int_(np.random.randint(0,nblocks,nblocks))]))
    boot_error=(np.average(boot),np.std(boot)*np.sqrt(nblocks/(nblocks-1)))

    return np.array((mean_error,jack_error,boot_error))
```

Comparison

err



block size

average=5.93

Results are identical!

Computing non-linear functions of avgs

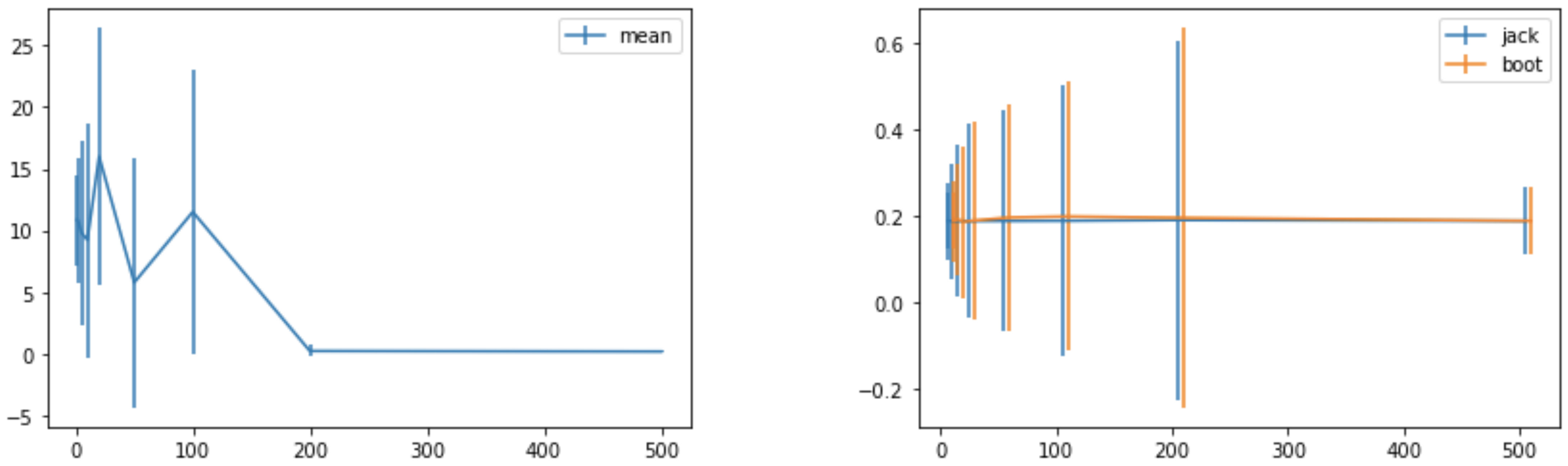
Let's say we want to estimate the free-energy difference between A and B (here we know should be 0).

Delta F = -kT*log(P_B/P_A)

We compute Delta F for each subsample, either:
single blocks (mean error)
sets of N_{blocks}-1 blocks (jack-knife)
random extraction of N_{blocks} blocks (bootstrap)

<u>Now average will also depend on block size</u>

Err-of-the-mean vs jack-knife vs bootstrap



Single blocks are horrible estimators!

Note: bootstrap is more expensive but gives access to other observables (e.g. histogram of predictions, etc)

In practice

```
def enediff(A):
    return np.log((A+1e-50)/(1-A+1e-50))

def analyze3(timeseries,blocksize):
    nblocks=int(len(timeseries)/blocksize)
    blockav=[]
    for i in range(nblocks):
        blockav.append(np.average(timeseries[i*blocksize:(i+1)*blocksize]>average))
    blockav=np.array(blockav)

    # error of the mean
    mean_error=(np.average(enediff(blockav)),np.std(enediff(blockav))/np.sqrt(nblocks-1))

    # jack-knife
    jack=[]
    for i in range(nblocks):
        jack.append((np.sum(blockav)-blockav[i])/(nblocks-1))
    jack=np.array(jack)
    jack_error=(np.average(enediff(jack)),np.std(enediff(jack))*np.sqrt(nblocks-1))

    # bootstrap
    boot=[]
    for i in range(1000):
        boot.append(np.average(blockav[np.int_(np.random.randint(0,nblocks,nblocks))]))
    boot=np.array(boot)
    boot_error=(np.average(enediff(boot)),np.std(enediff(boot))*np.sqrt(nblocks/(nblocks-1)))

    return np.array((mean_error,jack_error,boot_error))
```

What about weighted data?

Say you have an array of x_i and w_i?

There's not a unique answer, and the answer depends on the meaning you assign to the weights.

Useful readings:

http://www.analyticalgroup.com/download/weighted_variance.pdf

"Exponential smoothing weighted correlations", F. Pozzi, T. Matteo, T. Aste, Eur. Phys. J. B. 85, 175 (2012).

Final recommendations

When possible, look at your timeseries

Never assume data from MD or MC to be uncorrelated*

Perform block error with variable block size

As a bare minimum, compute error with few (say 3-5) blocks)

*If an error analysis tool assumes uncorrelated data, <u>do not use it</u>