Computational Methods (PHYS 2030)

Instructors: Prof. Christopher Bergevin (cberge@yorku.ca)

Schedule: Lecture: MWF 11:30-12:30 (CLH M)

Website: http://www.yorku.ca/cberge/2030W2018.html

York University
Winter 2018
Lecture 30

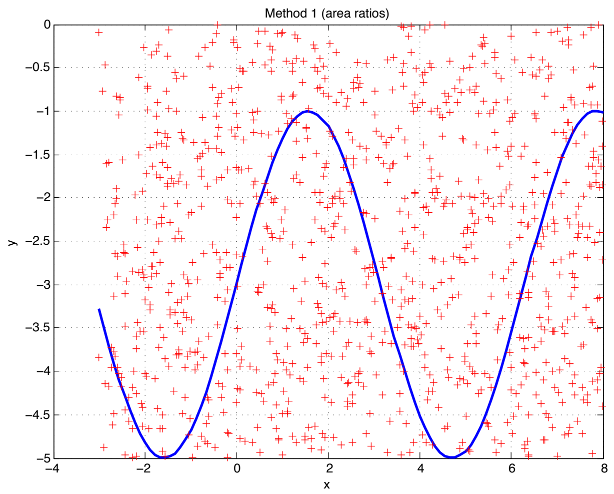
Ex. Integrals

EXintegrateMC1.m

- Can re-run a few times.....

→ Each run yields a slightly different value....

(except for `trapz.m`! why so stable?)



```
Integral calculated by trapz.m (i.e., Riemann sums)= -34.68724693
Integral calculated via Monte carlo Method 1 (area ratios) = -33.3834
Integral calculated via Monte carlo Method 2 (average value) = -34.1728

Integral calculated by trapz.m (i.e., Riemann sums)= -34.68724693
Integral calculated via Monte carlo Method 1 (area ratios) = -35.1434
Integral calculated via Monte carlo Method 2 (average value) = -34.7353

Integral calculated by trapz.m (i.e., Riemann sums)= -34.68724693
Integral calculated via Monte carlo Method 1 (area ratios) = -34.3734
Integral calculated via Monte carlo Method 2 (average value) = -34.4521

Integral calculated by trapz.m (i.e., Riemann sums)= -34.68724693
Integral calculated via Monte carlo Method 1 (area ratios) = -35.9683
Integral calculated via Monte carlo Method 2 (average value) = -35.8865
```

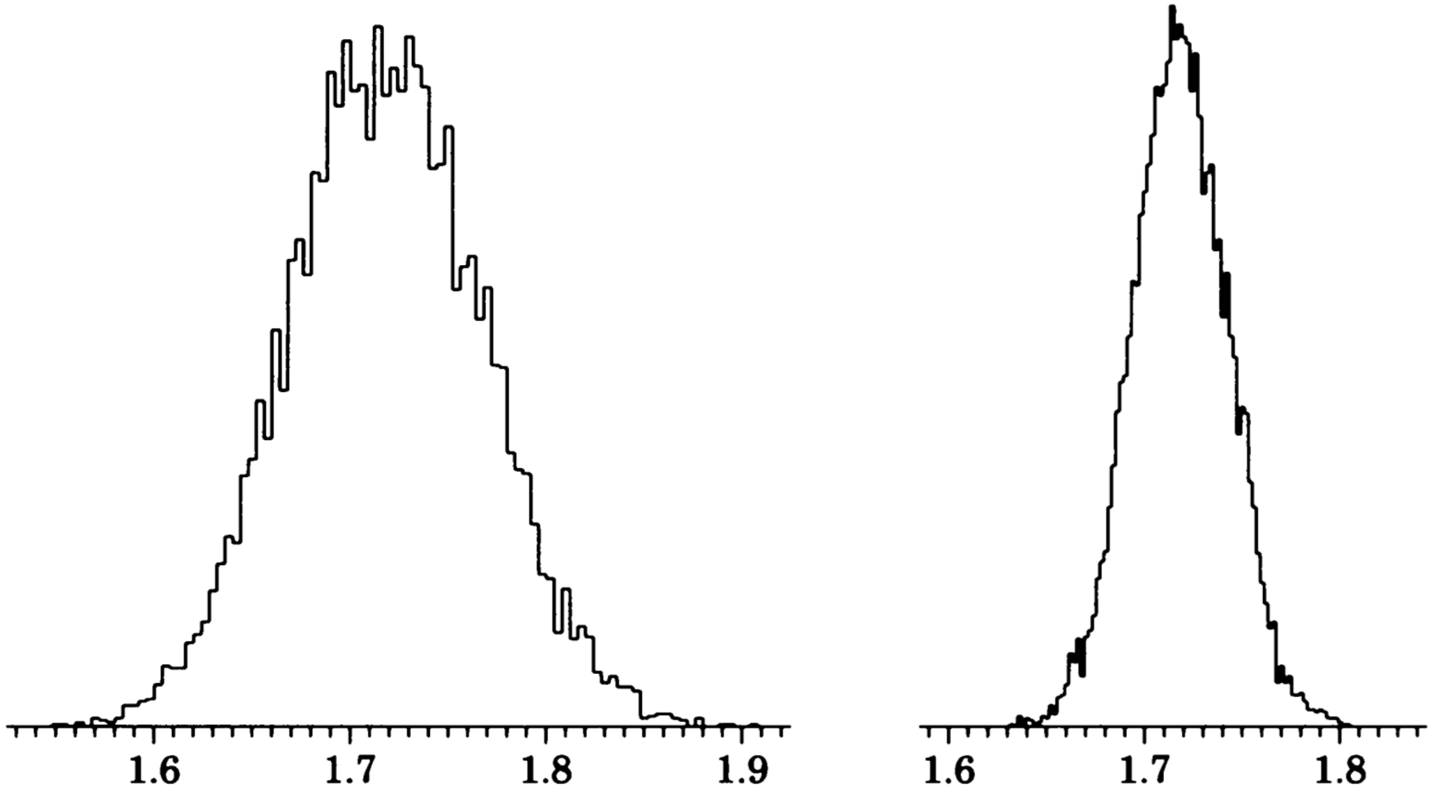


FIGURE 4.12 Distributions of 10,000 Monte Carlo estimates of the integral $\int_0^1 e^x\, dx$. On the left, each integral was evaluated with N = 100 points; on the right, with $N = 400$ points.

→ So the distributions looks like Gaussians..... (this is telling us something important!)

Devries (1994)

Ex. Making a Gaussian distribution

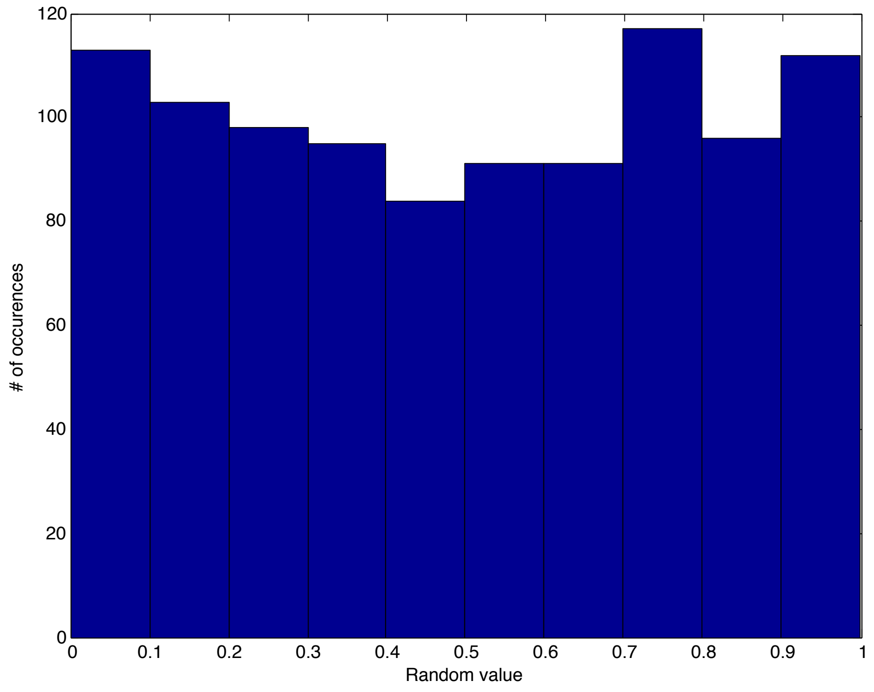
- Simple code to see if we can 'create' a normal distribution.....

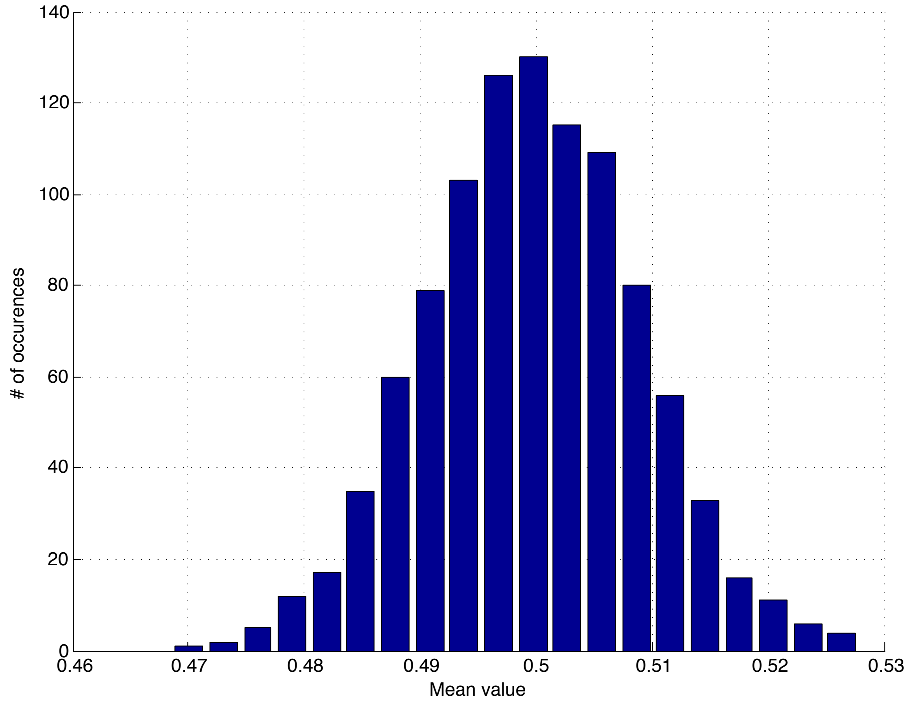
```
%  ### EXgaussian1.m ###
clear
% -----------
M= 1000;         % # of (uniformly distributed) random #s to average
N=1000;          % # of repeats (i.e., how many averages to compute) for histogram
binN= 20;       % # of bins for histogram
% -----------
figure(1); clf; hold on; grid on;
% +++
% loop thru to compute the N averages (each loop deals with the M random #s)
for nn=1:N
    xR= rand(M,1);  % determine array of M random #s
    mu(nn)= mean(xR);   % compute/store mean value
end
% +++
[jj,kk]=hist(mu,binN);   % detrmine histogram distribution
bar(kk,jj);             % plot the histogram (as a bar plot)
```

→ Simply determines a group of uniformly distributed numbers, then averages them. Subsequently, we keep track of the those mean values and plot as a '*histogram*'

Ex. Making a Gaussian distribution

```
M= 1000;        % # of (uniformly distributed) random #s to average
N=1000;         % # of repeats (i.e., how many averages to compute) for histogram
```



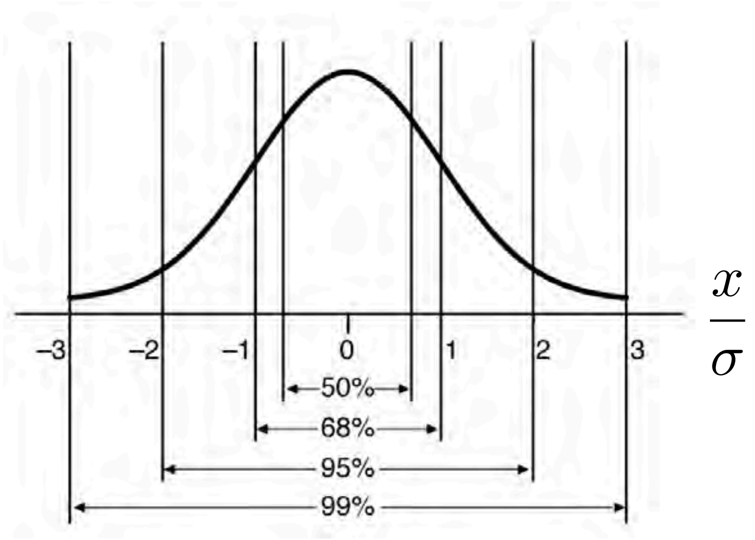


So within a given sample, the `M` points are uniformly distributed...

... but the average value (across `N` repetitions) is normally distributed!

→ This sort of observation demonstrates the notion of a normal distribution and is ultimately telling us something important about the nature of the underlying probability distribution!

Gaussian distributions



- Gaussian or 'normal' distributions (i.e., the 'bell-shaped curve') arise throughout many contexts in physics and engineering applications

- The parent distribution has a relatively simple analytic form:

$$f(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

Note: This expression can be generalized further and scaled

- This equation contains many of the quantities that we have come across already (e.g., the mean, standard deviation) and forms the basis for many common statistical measures (e.g., 95% confidence intervals)

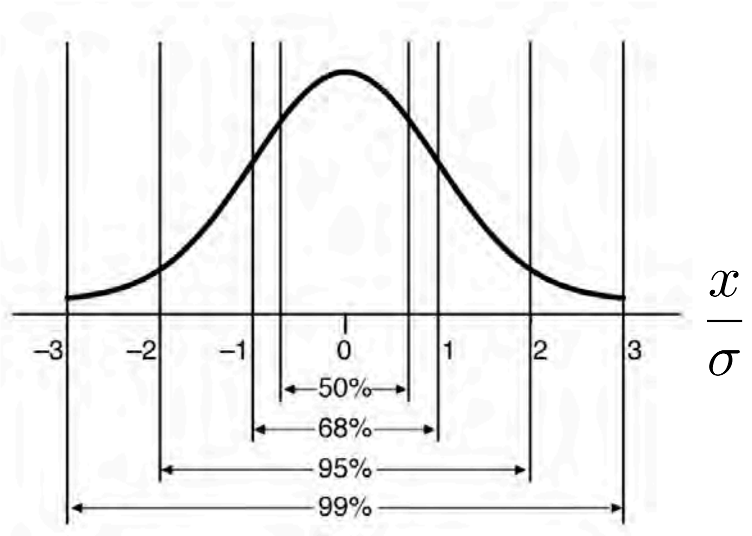
- Standard deviation (STD, or σ here) tells you what fraction of the area lays underneath the curve. For example:
  - 2/3 of the area is contained within +/- σ
  - 95% of the area is within +/- 2σ
  - Hence σ is the basis for: *confidence intervals, standard error, Student's t-tests, the coefficient of variation*, ...

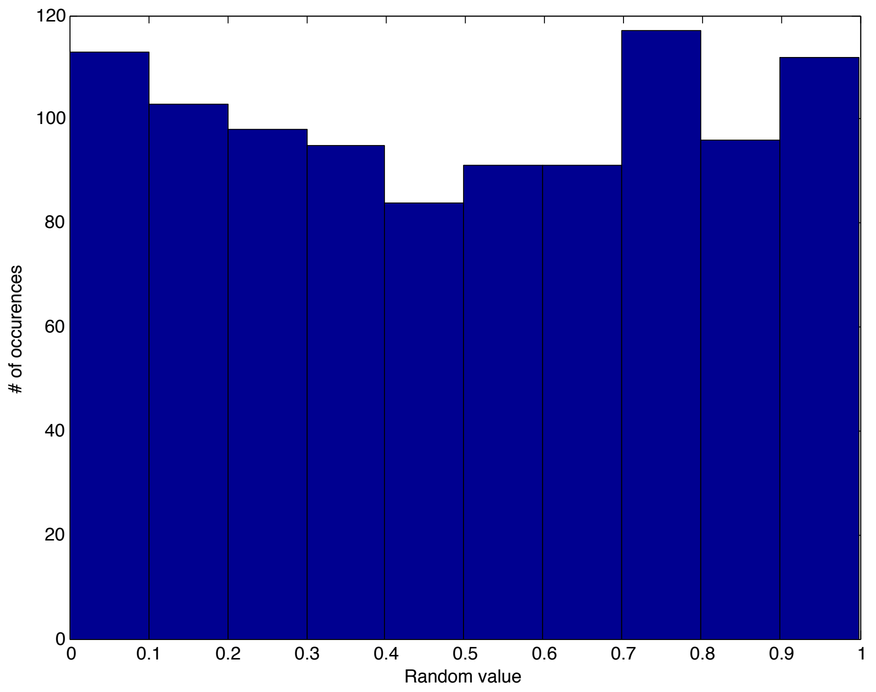
Freeman

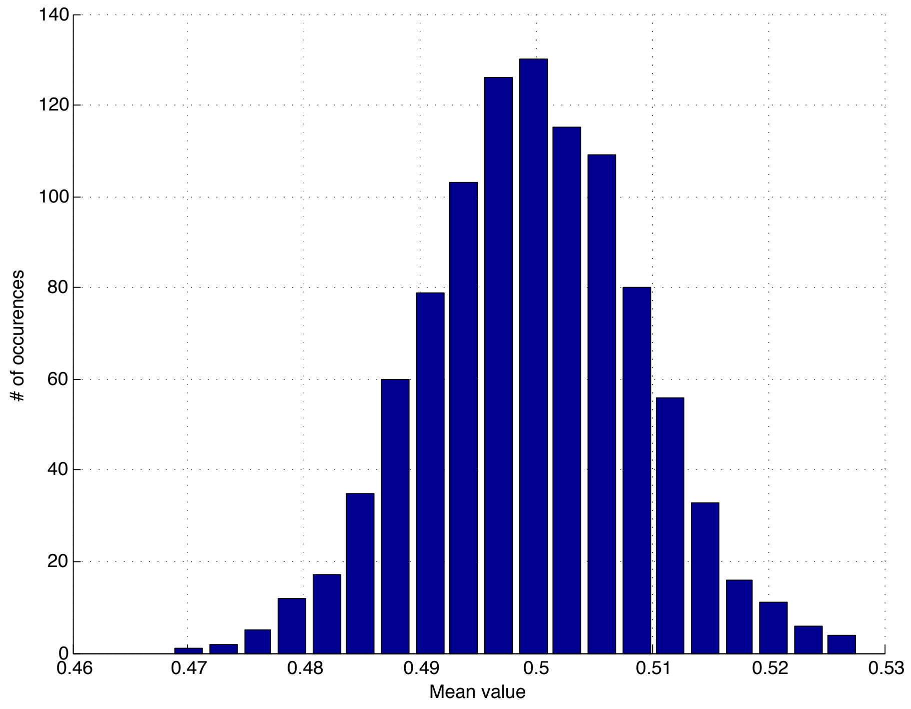
Gaussian distributions

Be careful!

→ A normal distribution doesn't necessarily tell you where an individual measurement lies, but the mean value across a set of measurements (i.e., what happens for compiling across REPEATED measurements)



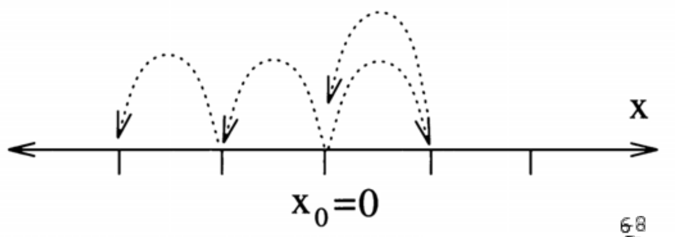


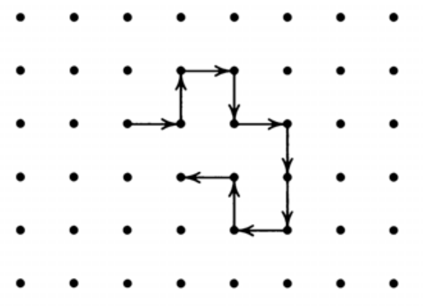


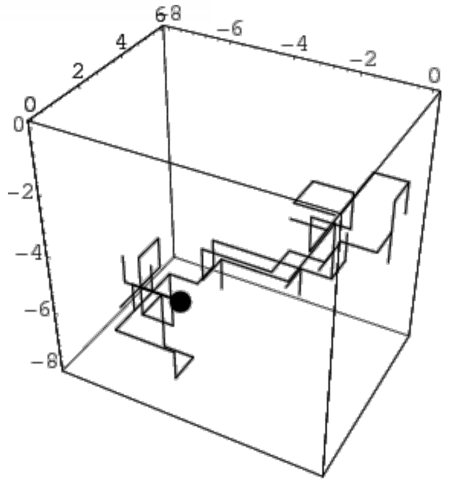
Ex. Random walks

- A common conceptual starting point in probability, statistical mechanics, and biophysics

- Also known as the ‘drunken sailor’ problem, we can do this in 1-D, 2-D, 3-D, or higher....







- Imagine a grid, upon which we take a step in a random direction. We can then trace out a path as time goes on

- If we consider an ‘ensemble’ of random walkers, each starting at the origin and independent of one another, computationally it’s easy for us to keep track of the *average net movement* (Mean Squared Distance, MSD)

$$< x^2 > = D\,t$$

$<x^2>$ - Mean-squared distance
D – ‘diffusion’ constant
t – time allowed before ‘checking’ $<x^2>$

Devries (1994)
Giordano (1997)
http://mathworld.wolfram.com/RandomWalk3-Dimensional.html

Ex. Random walks

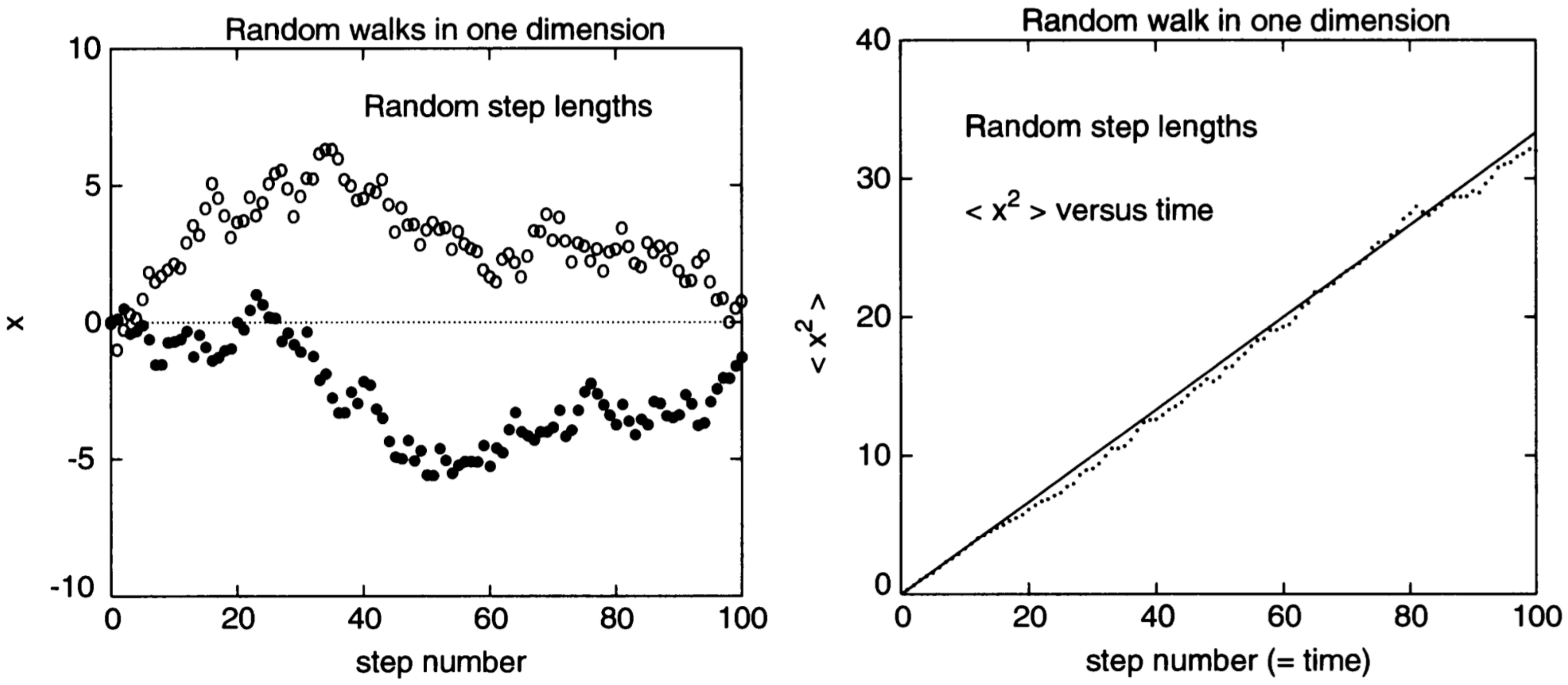


Figure 7.10: Left: x versus step number (that is, time) for two random walks in one dimension. Here the steps were of random lengths in the range -1 to 1. Right: $< x^2 >$ as a function of time for a collection of these one-dimensional random walks. The results for 500 walks were averaged.

→ So while each individual walker is random, the basic idea is that in an *ensemble average*, a repeatable/consistent trend emerges

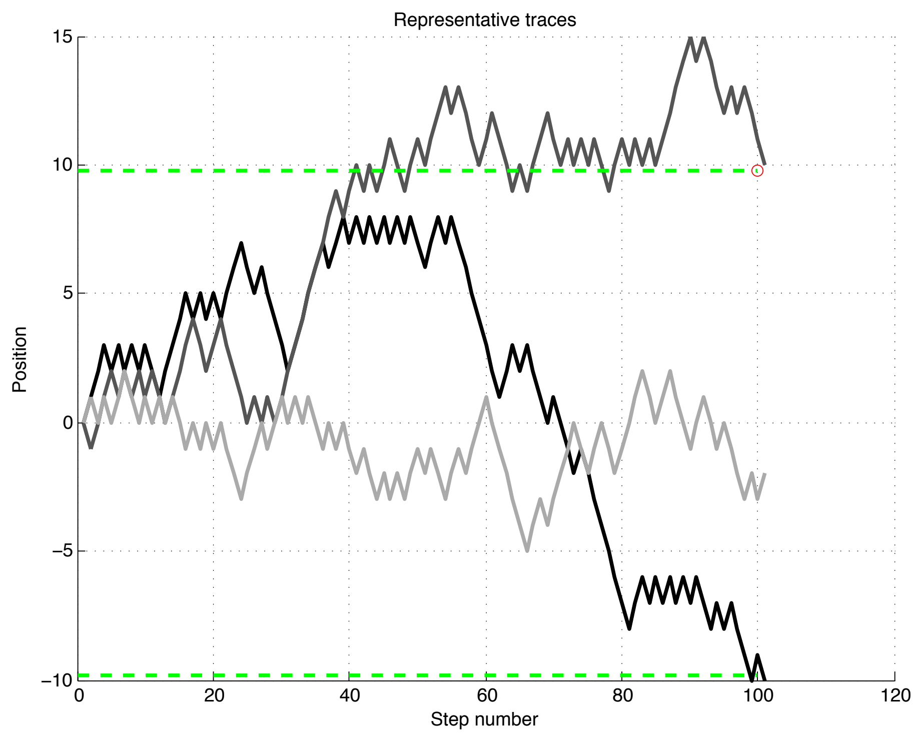
Giordano (1997)

EXrandomWalk1D.m

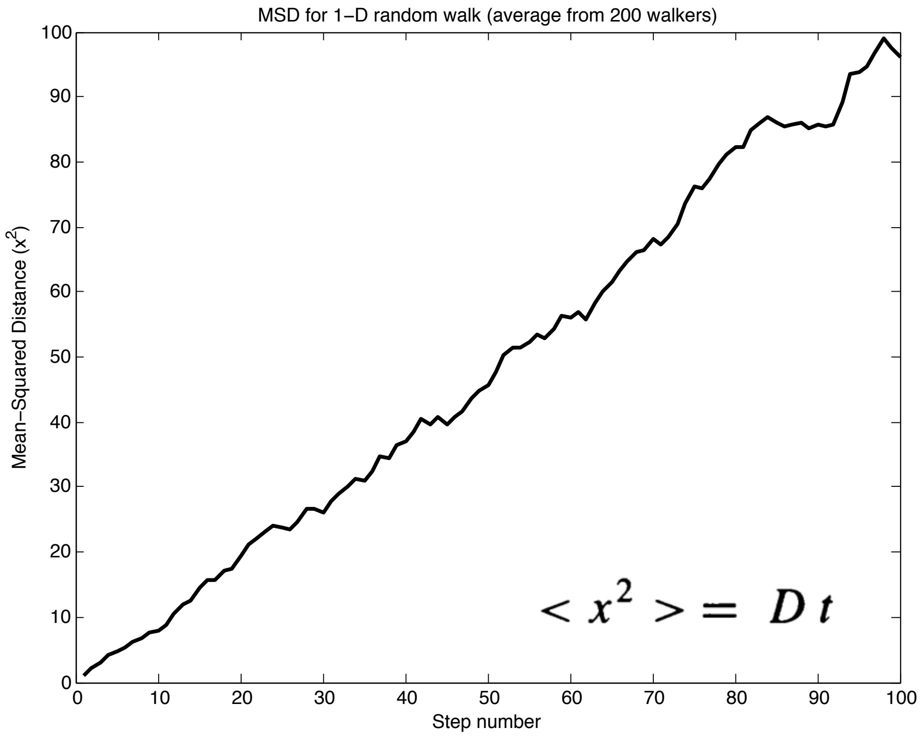
```
% ### EXrandomWalk1D.m ###    11.15.14
clear;
% -------------
N= 200;      % Total # of (independent) walkers (each starts at x=0)
M= 100;      % Total # of steps for each walker
K= 3;       % # of walkers to show individual traces for [3]
bias= 0.5;      % number between [0,1] to indicate bias for left vs right (0.5= equal prob.)
% -------------
% +++
step_number= zeros(1,M);        %
x2ave= zeros(1,M);              % allocate array to stored (suquentially averaged) MSD
step_number_array= [1:1:M];     %
% +++
%
% NOTE: the loop is set up in such a way to average x2ave across walkers
for r= 1:N
    x=0;    % initialize position for r'th walker
    position(r,1)= 0;
    % loop to go through M steps for r'th walker
    for nn=1:M;
        % conditional determines whether step is to the left or right
        if (rand<bias),  x=x+1;
        else    x=x-1;  end;
        x2ave(nn)=x2ave(nn)+x^2;    % store squared displacement (handles averging across r)
        position(r,nn+1)= x;          % store displacment for each walker and step
    end;
end;
x2ave= x2ave/N;     % Divide by number of walkers
% plot MSD
figure(1);
plot(step_number_array, x2ave, 'k'); hold on;
title('MSD for 1-D random walk');
xlabel('Step number'); ylabel('Mean-Squared Distance (x^2)');
% plot a subset of individual traces
figure(2); clf; hold on; grid on;
for nn=1:K
    shade= 1-(nn-1)/K;
    plot(position(nn,:),'Color',[1 1 1]-shade);
end
xlabel('Step number'); ylabel('Position'); title('Representative traces');
plot([0 M],[1 1]*sqrt(x2ave(end)),'g--','LineWidth',2) % include MSD bounds at step M
plot([0 M],[-1 -1]*sqrt(x2ave(end)),'g--','LineWidth',2)
plot(M,sqrt(mean(position(:,end).^2)),'ro');    % reality check (another way to compute final MSD)
disp(['Final mean (non-squared) distance = ',num2str(mean(position(:,end)))]);
```

- Ensemble of `N` (independent) walkers
- Each takes `M` total steps, each step either left or right
- Note that the `for` loop averages as it goes

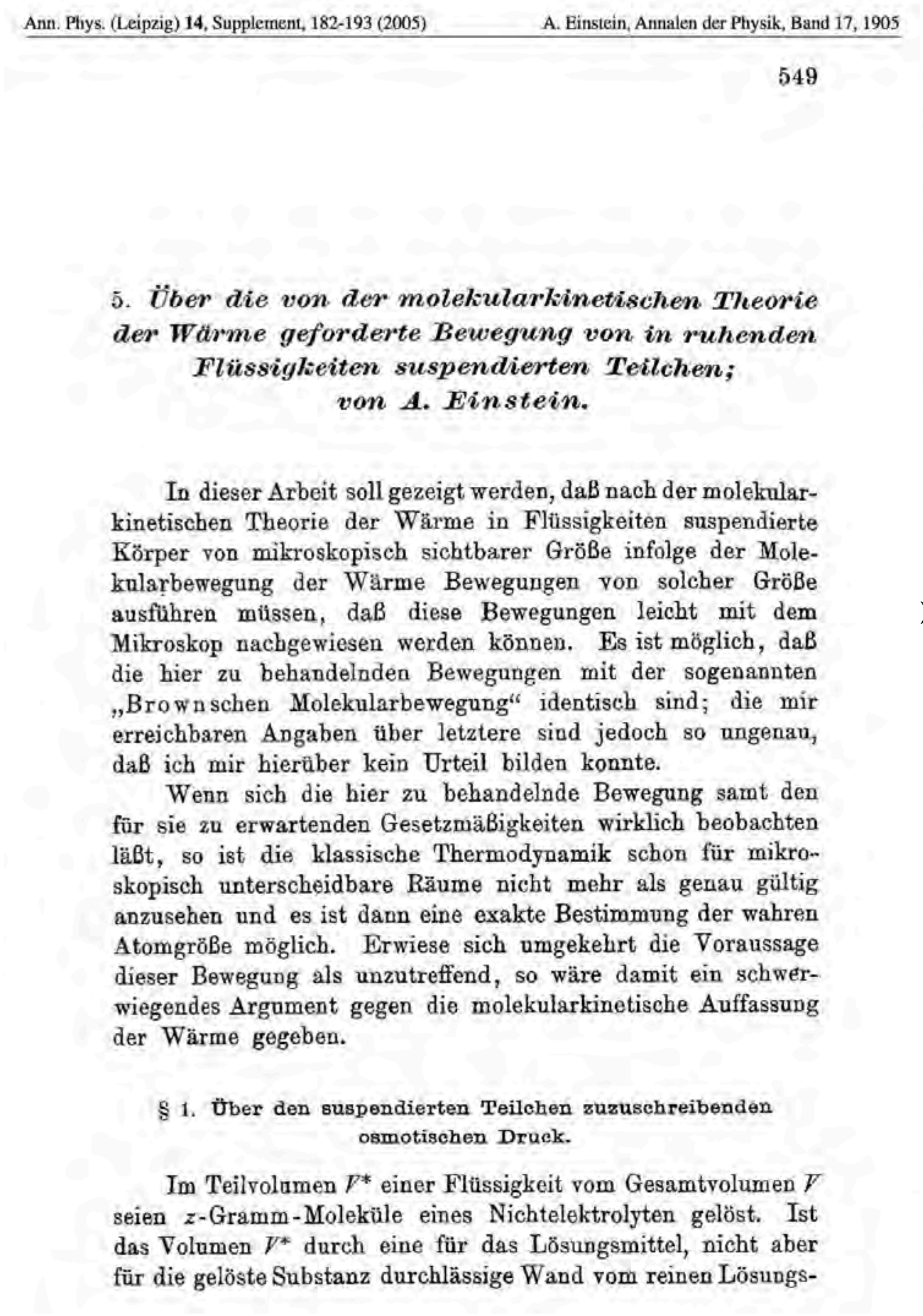
EXrandomWalk1D.m



Representative traces from three different (independent) walkers



Mean-squared distance traveled by a large ensemble of walkers



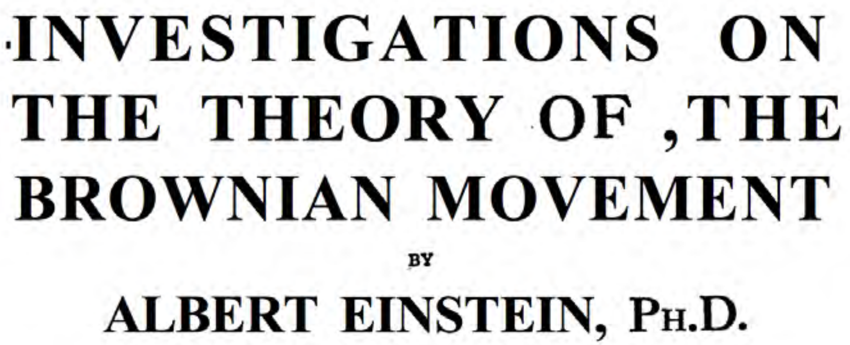
5. *Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen;*
von A. Einstein.

In dieser Arbeit soll gezeigt werden, daß nach der molekularkinetischen Theorie der Wärme in Flüssigkeiten suspendierte Körper von mikroskopisch sichtbarer Größe infolge der Molekularbewegung der Wärme Bewegungen von solcher Größe ausführen müssen, daß diese Bewegungen leicht mit dem Mikroskop nachgewiesen werden können. Es ist möglich, daß die hier zu behandelnden Bewegungen mit der sogenannten „Brownschen Molekularbewegung" identisch sind; die mir erreichbaren Angaben über letztere sind jedoch so ungenau, daß ich mir hierüber kein Urteil bilden konnte.

Wenn sich die hier zu behandelnde Bewegung samt den für sie zu erwartenden Gesetzmäßigkeiten wirklich beobachten läßt, so ist die klassische Thermodynamik schon für mikroskopisch unterscheidbare Räume nicht mehr als genau gültig anzusehen und es ist dann eine exakte Bestimmung der wahren Atomgröße möglich. Erwiese sich umgekehrt die Voraussage dieser Bewegung als unzutreffend, so wäre damit ein schwerwiegendes Argument gegen die molekularkinetische Auffassung der Wärme gegeben.

§ 1. Über den suspendierten Teilchen zuzuschreibenden osmotischen Druck.

Im Teilvolumen V^* einer Flüssigkeit vom Gesamtvolumen V seien z-Gramm-Moleküle eines Nichtelektrolyten gelöst. Ist das Volumen V^* durch eine für das Lösungsmittel, nicht aber für die gelöste Substanz durchlässige Wand vom reinen Lösungs-



INVESTIGATIONS ON THE THEORY OF THE BROWNIAN MOVEMENT

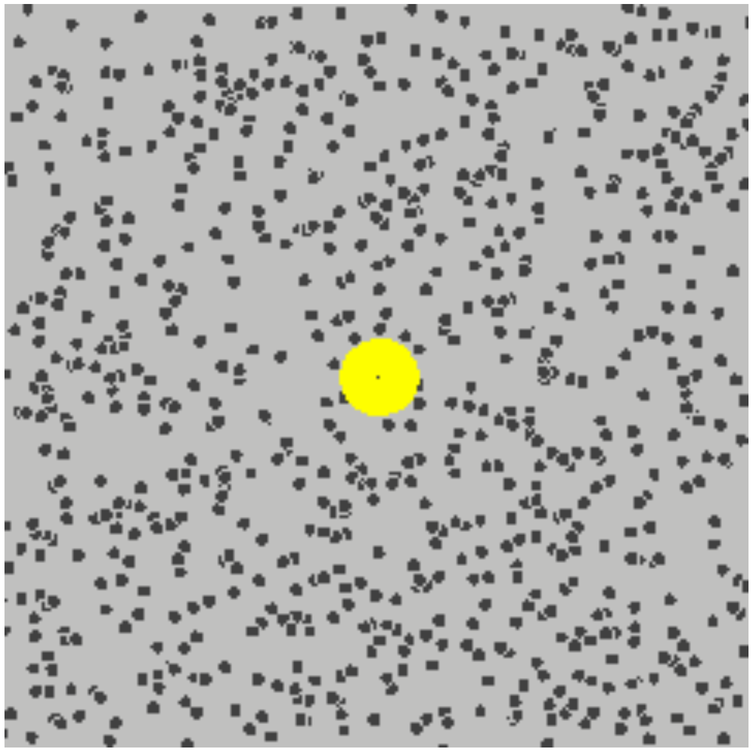
BY

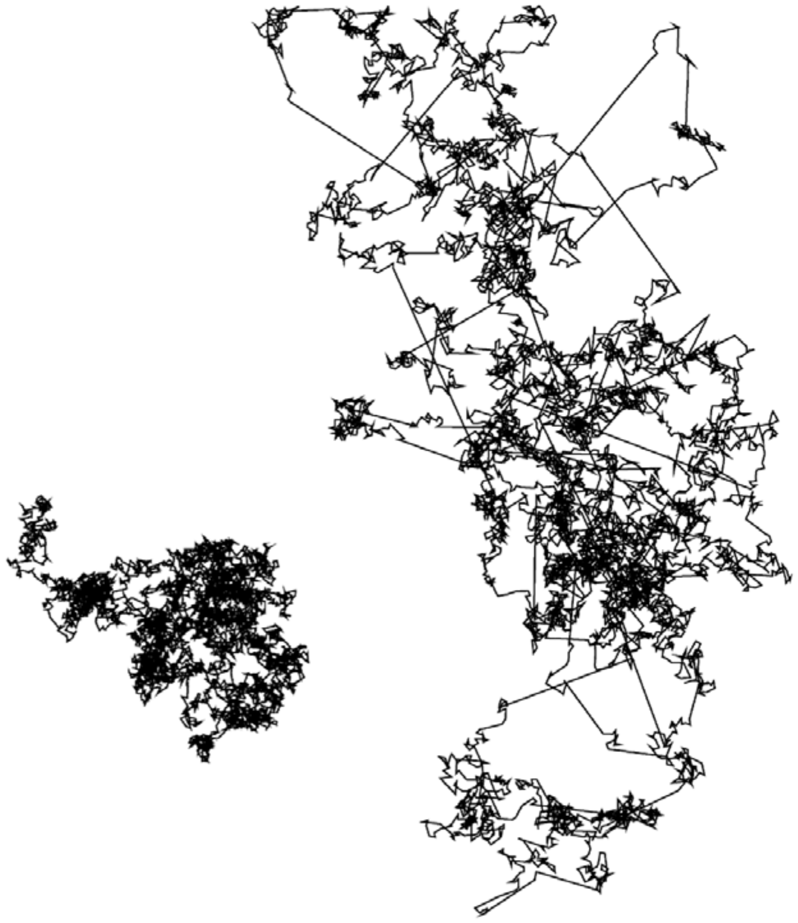
ALBERT EINSTEIN, PH.D.

- One of Einstein's *Annus Mirabilis* papers from 1905 (the other two are on special relativity and the photoelectric effect)

- Solidified the foundations of statistical mechanics and characterized the conditions for most living things on this planet (i.e., cell-sized entities)

Brownian motion


Random motion of large object (yellow circle) due to interaction with many little objects (black circles)


Can trace out path across time and space

$$< x^2 > = D\,t$$

wikipedia (Brownian motion)
Metzler & Klafter (2000)

Brownian motion & Diffusion

8 R. Metzler, J. Klafter / Physics Reports 339 (2000) 1–77

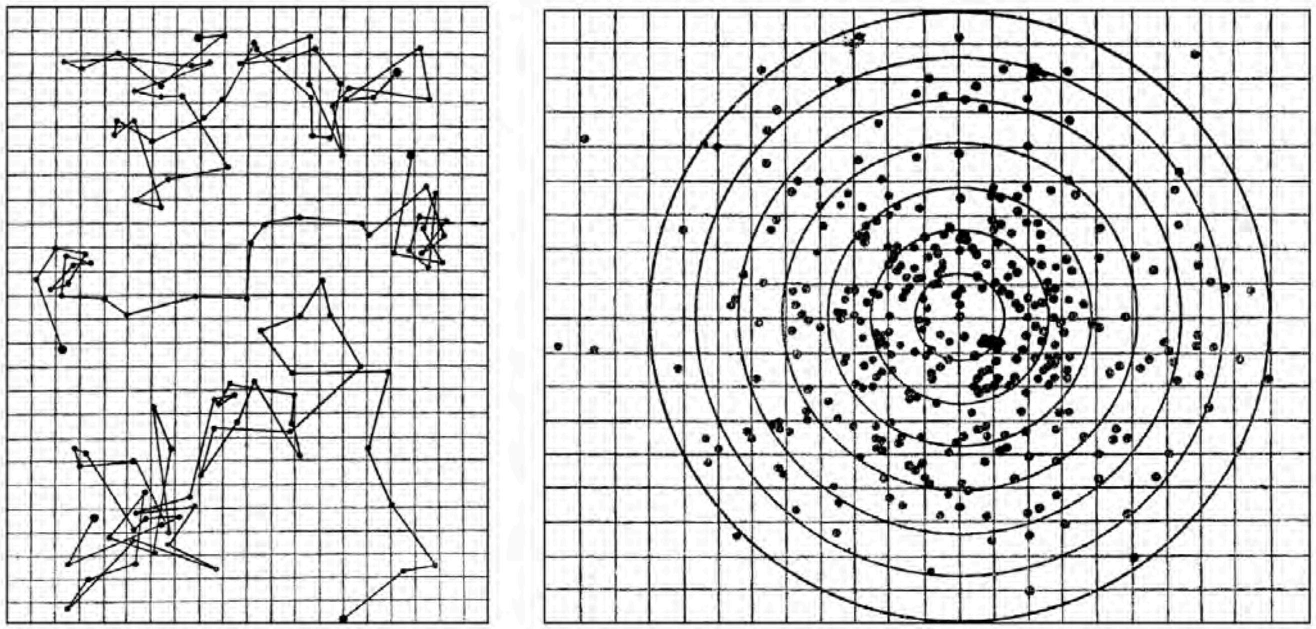
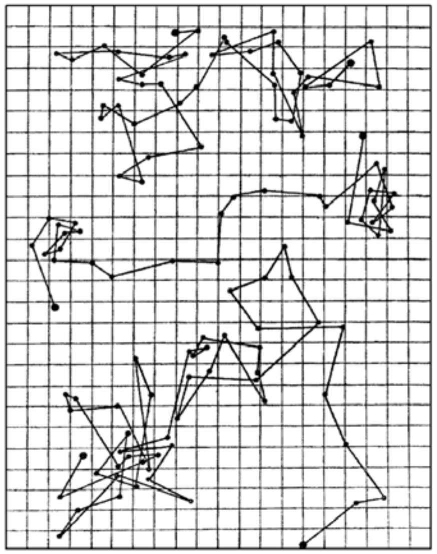
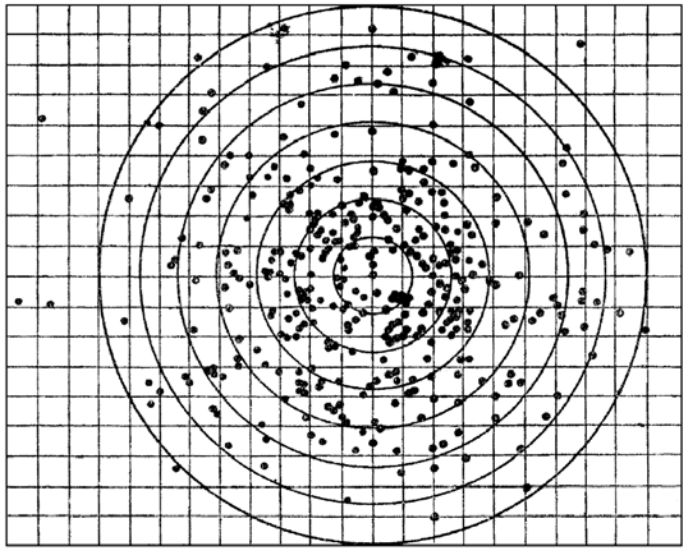


Fig. 2. Recorded random walk trajectories by Jean Baptiste Perrin [72]. Left part: three designs obtained by tracing a small grain of putty (*mastic*, used for varnish) at intervals of 30 s. One of the patterns contains 50 single points. Right part: the starting point of each motion event is shifted to the origin. The figure illustrates the pdf of the travelled distance r to be in the interval $(r, r + \mathrm{d}r)$, according to $(2\pi\xi^2)^{-1}\exp(-r^2/[2\xi^2])2\pi r\,\mathrm{d}r$, in two dimensions, with the length variance ξ^2. These figures constitute part of the measurement of Perrin, Dabrowski and Chaudesaigues leading to the determination of the Avogadro number. The result given by Perrin is 70.5×10^{22}. The remarkable œuvre of Perrin discusses all possibilities of obtaining the Avogadro number known at that time. Concerning the trajectories displayed in the left part of this figure, Perrin makes an interesting statement: "Si, en effet, on faisait des pointés de seconde en seconde, chacun de ces segments rectilignes se trouverait remplacé par un contour polygonal de 30 côtés relativement aussi compliqué que le dessin ici reproduit, et ainsi de suite". [If, veritably, one took the position from second to second, each of these rectilinear segments would be replaced by a polygonal contour of 30 edges, each itself being as complicated as the reproduced design, and so forth.] This already anticipates Lévy's cognisance of the self-similar nature, see footnote 9, as well as of the non-differentiability recognised by N. Wiener.

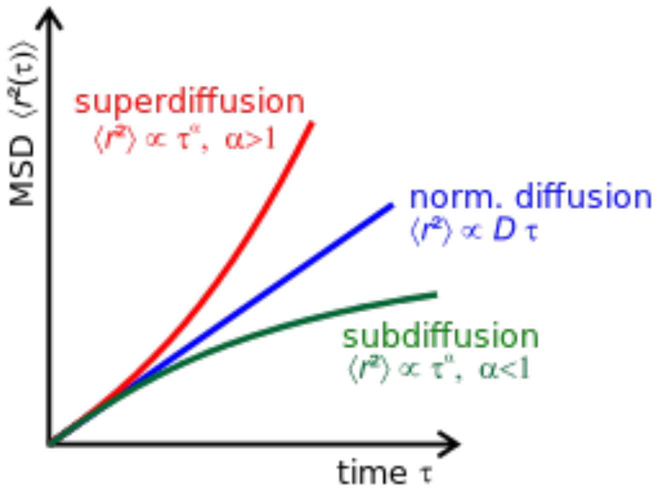
Jean Baptiste Perrin (1870-1942)

Brownian motion & Diffusion

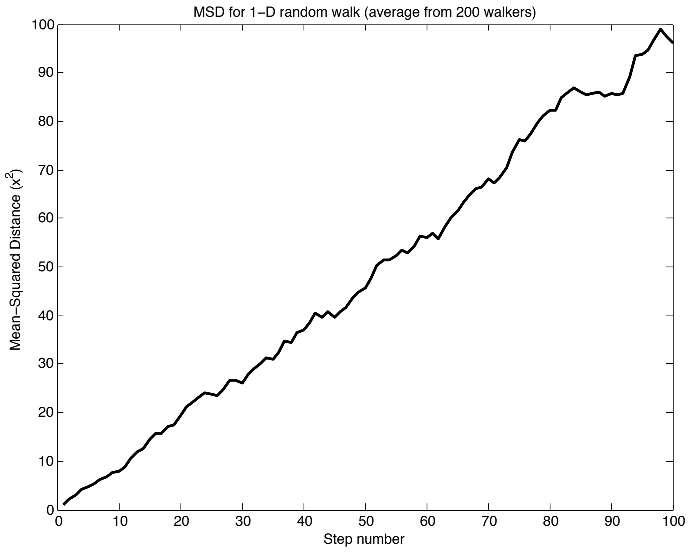




$$< x^2 > = D\,t$$



wikipedia (anomalous diffusion)

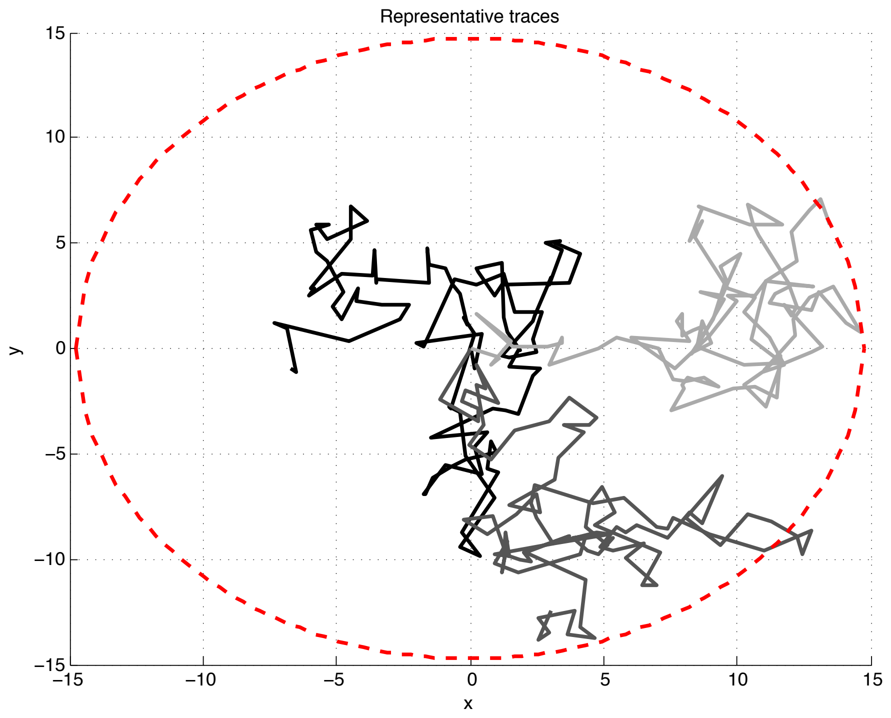


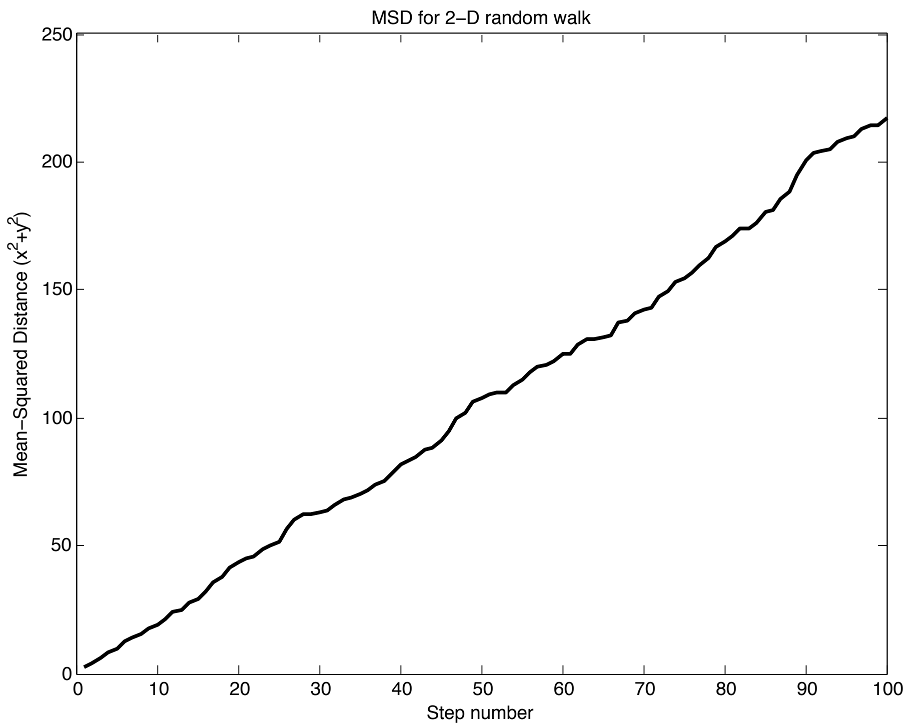
EXrandomWalk2D.m

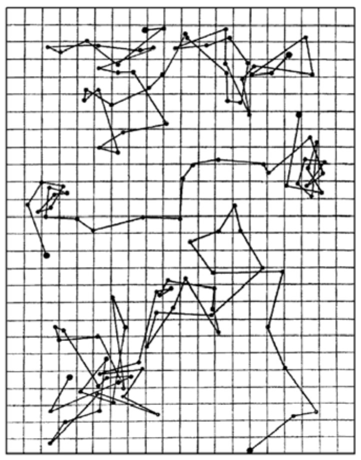
```
% ### EXrandomWalk2D.m ###
% o Method 1 - simply equal probability one step up/down (U/D) and left/right (L/R)
% o Method 2 - U/D and L/R steps are sampled from a uniform distribution over [-1,1]
% o Method 3 - U/D and L/R steps are sampled from a Gaussian distribution
clear;
% -------------
N= 200;      % Total # of (independent) walkers (each starts at x=0)
M= 100;      % Total # of steps for each walker
K= 3;       % # of walkers to show individual traces for [3]
method= 3;  % see comments above
% -------------
% +++
step_number= zeros(1,M);         %
posAvg= zeros(1,M);               % allocate array to stored (suquentially averaged) MSD
step_number_array= [1:1:M];      %
% +++
for r= 1:N
    x=0; y=0;   % initialize positions for r'th walker
    positionX(r,1)= 0;  positionY(r,1)= 0;
    % loop to go through M steps for r'th walker
    for nn=1:M;
        if method==1
            if (rand<0.5),  x=x+1;  % conditional for left or right
            else    x=x-1;  end;
            if (rand<0.5),  y=y+1;  % conditional for up or down
            else    y=y-1;  end;
        elseif method==2
            x= x+2*rand(1)-1;   y= y+2*rand(1)-1;
        elseif method==3
            x= x+randn(1);   y= y+randn(1);
        end
        posAvg(nn)= posAvg(nn)+ (x^2+y^2);    % store squared displacement (handles averging across r)
        positionX(r,nn+1)= x;          % store displacments for each walker and step
        positionY(r,nn+1)= y;
    end;
end;
posAvg= posAvg/N;     % Divide by number of walkers
% plot MSD
figure(1);
plot(step_number_array,posAvg, 'k'); hold on;
title('MSD for 2-D random walk');
xlabel('Step number'); ylabel('Mean-Squared Distance (x^2+y^2)');
% plot a subset of individual traces
figure(2); clf; hold on; grid on;
for nn=1:K
    shade= 1-(nn-1)/K;
    plot(positionX(nn,:),positionY(nn,:),'Color',[1 1 1]-shade);
end
xlabel('x'); ylabel('y'); title('Representative traces');
% also plot MSD (which is a circular arc in this case)
rBND= sqrt(posAvg(end));  % radius MSD
xBND= linspace(-rBND,rBND,100); yBND= sqrt(rBND^2-xBND.^2);
plot(xBND,yBND,'r--','LineWidth',2); plot(xBND,-yBND,'r--','LineWidth',2);
```

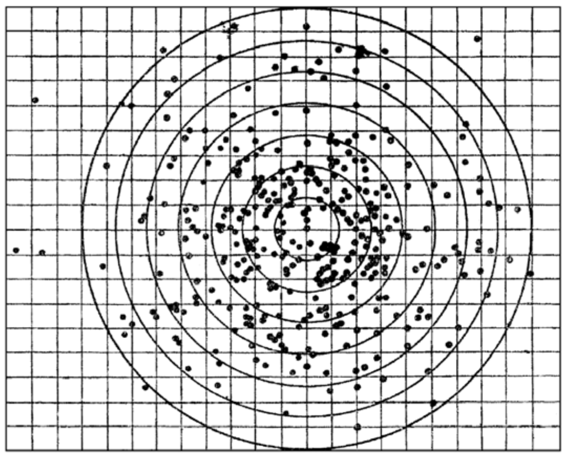
- Same basic idea as 1-D code, but allows for more flexibility

EXrandomWalk2D.m









Ex. Matrix permutations

```
groupSize= 5;    % # of elements per group
```

- Random permutations is a randomized reordering of a set of object

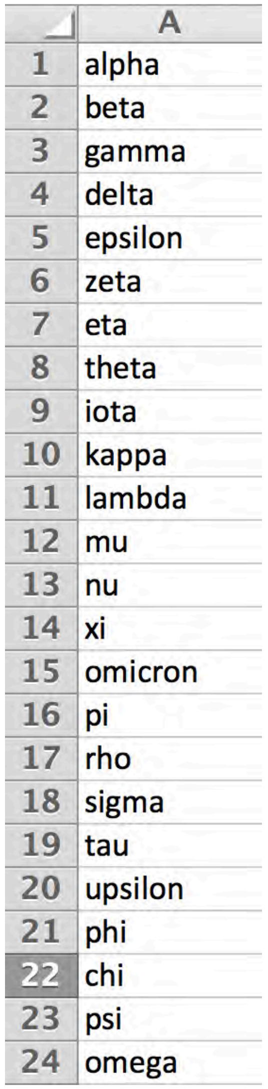
- Useful in a wide range of applications (e.g., shuffling a deck of cards, deciding upon a password or PIN, puzzles, mixing students up for project groups, ...)

- Fairly easy in Matlab via the built-in function `randperm.m`

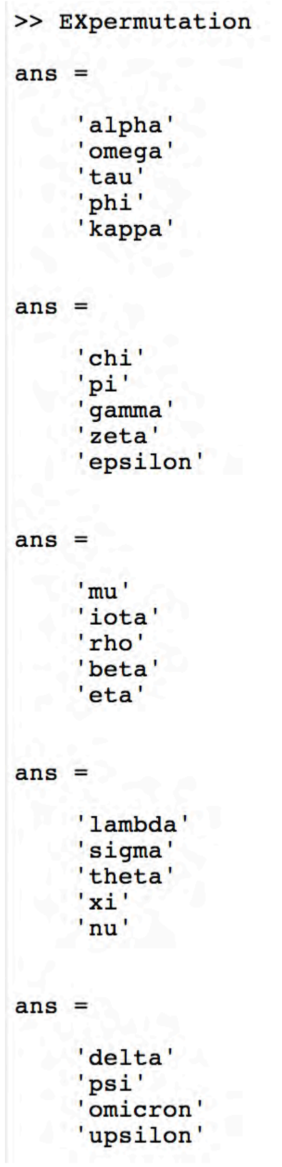
(see also http://en.wikipedia.org/wiki/Fisher-Yates_shuffle)

→ Works by creating a random set of array indices

Note: This is an example of '*without replacement*' (more on this in a bit)



	A
1	alpha
2	beta
3	gamma
4	delta
5	epsilon
6	zeta
7	eta
8	theta
9	iota
10	kappa
11	lambda
12	mu
13	nu
14	xi
15	omicron
16	pi
17	rho
18	sigma
19	tau
20	upsilon
21	phi
22	chi
23	psi
24	omega



```
>> EXpermutation

ans =

    'alpha'
    'omega'
    'tau'
    'phi'
    'kappa'


ans =

    'chi'
    'pi'
    'gamma'
    'zeta'
    'epsilon'


ans =

    'mu'
    'iota'
    'rho'
    'beta'
    'eta'


ans =

    'lambda'
    'sigma'
    'theta'
    'xi'
    'nu'


ans =

    'delta'
    'psi'
    'omicron'
    'upsilon'
```

```
% ### EXpermutation.m ###       10.27.14
% Reads in a specified Excel file and creates a random permutation into
% smaller groups (useful to create randomized small groups)
clear
% --------
fileL= 'GreekLetters.xlsx'; % list
groupSize= 5;   % # of elements per group
fileS= 'test.mat';     % file to save to (.mat)
% --------
[junk,A]= xlsread(fileL);   % read in file
p= randperm(numel(A));      % create a random permutation
% loop to group elements together
indx= 1;
for nn=indx:ceil(numel(A)/groupSize)
    % allow for odd-ish number of elements
    if (indx+groupSize>=numel(A))
        Group{nn}= A(p(indx:end));
    else
        Group{nn}= A(p(indx:indx+groupSize-1));
    end
    indx= indx+ groupSize;
    Group{nn}   % display to screen
end
% save to file?
if (1==0), save(fileS,'Group'); end
```

Resampling

- Basic idea: For a given set of 'data', take a random (sub)set and analyze. Rinse & repeat. Determine the associated statistics of such.

- Can be done '*with replacement*' or '*without replacement*'

Ex. Consider a container with 10 balls, each with a single letter (A-J) on the side. Now say we want to grab 6 balls at random from the container.

- For the '*with replacement*' condition, each time we grab a ball, we note what it was, then throw it back in. That is, each time we sample, we do such from 10 balls. Some examples of possible sets would be: [B G G H C I], [G F A D H C], [G F A A D C], [A A B B C C], [C C C H F A], [C H F C A C], ...
- For the '*without replacement*' condition, each time we grab a ball, we do not throw it back in. That is, each time we sample, there is one less ball than before. Thus, there would never be 'repeated' values (unless two balls had the same letter). Possible sets would be: [G A B I E F], [A B C D E F], [J A C F E B]

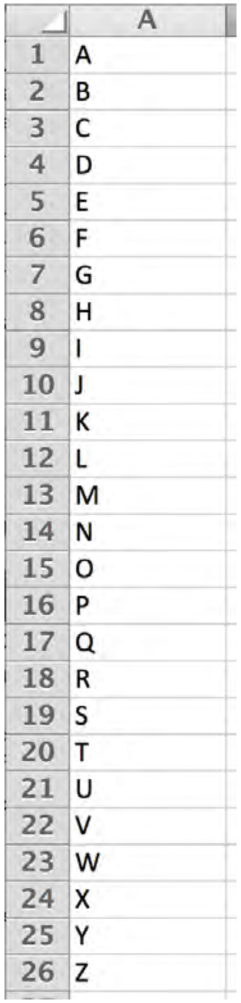
EXpermutation2.m

```
% ### EXpermutation2.m ###
% Demonstrates resampling into subsets both with and without replacement
clear
% --------
fileL= 'LatinLetters.xlsx'; % list
groupSize= 9;    % # of elements per group
fileS= 'test.mat';      % file to save to (.mat)
% --------
[junk,A]= xlsread(fileL);    % read in file
N= numel(A);
p= randperm(N);       % create a random permutation (w/o replacement)
NoReplace= A(p(1:groupSize))'
temp= floor(N*rand(1,N))+1;   % allow for possible repeated values
Replace= A(temp(1:groupSize))'
```

```
NoReplace =

    'R'    'Q'    'A'    'D'    'J'    'Z'    'F'    'W'    'V'


Replace =

    'E'    'K'    'P'    'C'    'V'    'S'    'T'    'S'    'T'
```

```
NoReplace =

    'M'    'U'    'D'    'O'    'V'    'K'    'E'    'G'    'A'

Replace =

    'H'    'P'    'T'    'A'    'F'    'C'    'X'    'S'    'W'
```

```
NoReplace =

    'M'    'S'    'O'    'L'    'V'    'B'    'R'    'T'    'G'


Replace =

    'I'    'O'    'P'    'V'    'T'    'H'    'O'    'D'    'W'
```



	A
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	K
12	L
13	M
14	N
15	O
16	P
17	Q
18	R
19	S
20	T
21	U
22	V
23	W
24	X
25	Y
26	Z

<u>Resampling</u>

- <u>Basic idea</u>: For a given set of 'data', take a random subset and analyze. Rinse & repeat. Determine the associated statistics of such.

- Can be done '*with replacement*' or '*without replacement*'

- Commonly used methods falls into several broad categories. For a (1-D) data set with N elements:

  - *Jackknifing* – Randomly sample all possible N-1 subsets, calculate your statistics for each, then aggregate all the results

  - *Bootstrapping* – Randomly sample with replacement a set with N elements, calculate your statistics, repeat (for some large number P), then aggregate all the results

  - *Subsampling* – Randomly sample a smaller subset with M elements (M<N), calculate your statistics, repeat (for some large number P), then aggregate all the results (a modification of jackknifing)

Ex. Jackknifing

Fit a straight line to the data set: (x,y) = [0,1.1], [1,1.9], [2,3.2], [3,4.1], [4,4.8]

1. Do a linear regression each on the following:

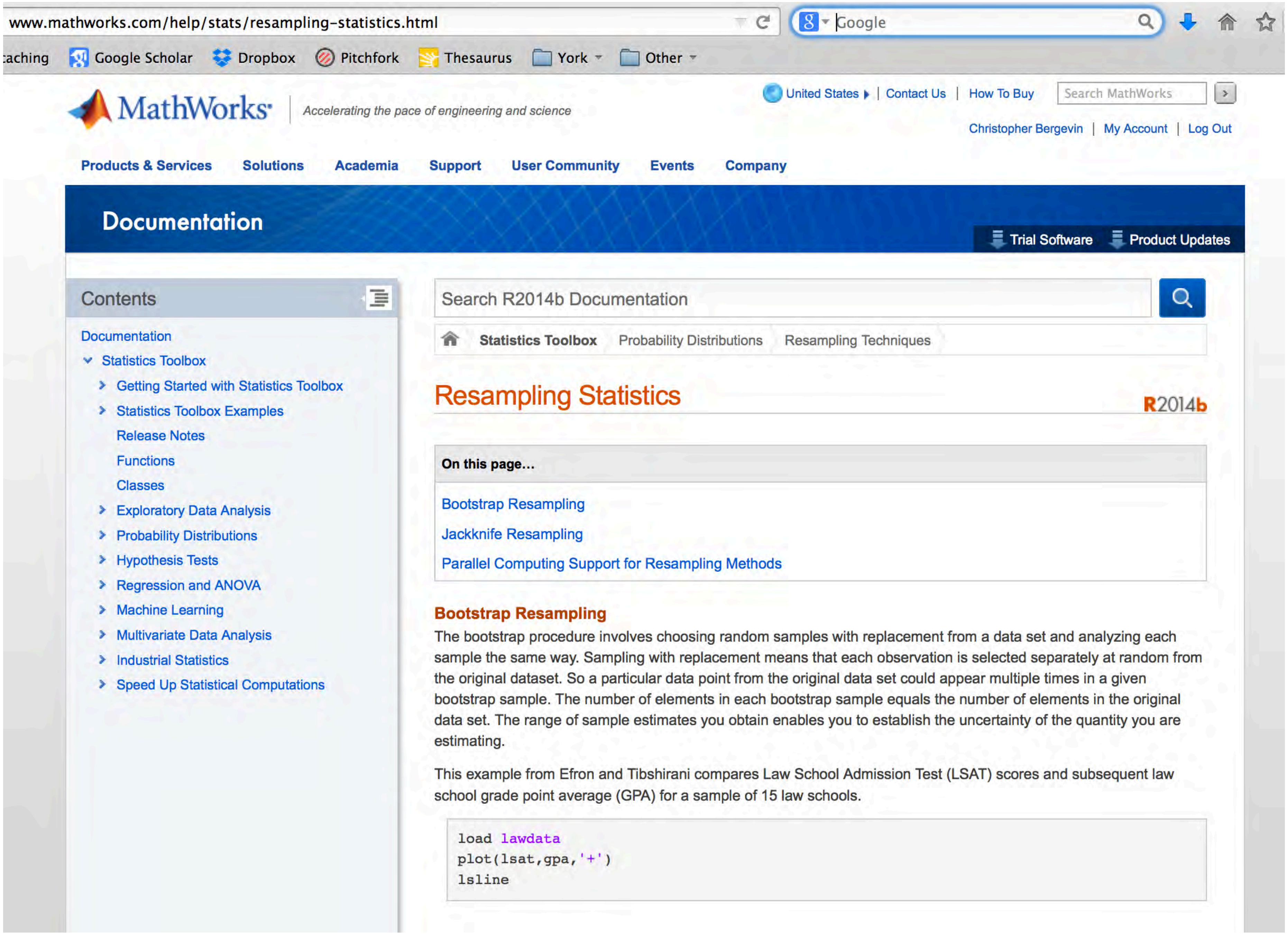
- [0,1.1], [1,1.9], [2,3.2], [3,4.1]
- [0,1.1], [1,1.9], [2,3.2], [4,4.8]
- [0,1.1], [1,1.9], [3,4.1], [4,4.8]
- [0,1.1], [2,3.2], [3,4.1], [4,4.8]
- [1,1.9], [2,3.2], [3,4.1], [4,4.8]

2. Now you'd have 5 slopes and 5 intercepts. From this, you can determine a mean value and an associated uncertainty

Question: How would such a set of parameters compare a fit to the entire data set? Or the uncertainty from the linear regression to the entire set?

$$\sigma_a^2 = \frac{\sigma^2}{\Delta'}\Sigma x_i^2 \qquad \text{and} \qquad \sigma_b^2 = N\frac{\sigma^2}{\Delta'}$$

(see previous lecture notes)



www.mathworks.com/help/stats/resampling-statistics.html

Resampling Statistics

R2014b

Bootstrap Resampling

The bootstrap procedure involves choosing random samples with replacement from a data set and analyzing each sample the same way. Sampling with replacement means that each observation is selected separately at random from the original dataset. So a particular data point from the original data set could appear multiple times in a given bootstrap sample. The number of elements in each bootstrap sample equals the number of elements in the original data set. The range of sample estimates you obtain enables you to establish the uncertainty of the quantity you are estimating.

This example from Efron and Tibshirani compares Law School Admission Test (LSAT) scores and subsequent law school grade point average (GPA) for a sample of 15 law schools.

```
load lawdata
plot(lsat,gpa,'+')
lsline
```

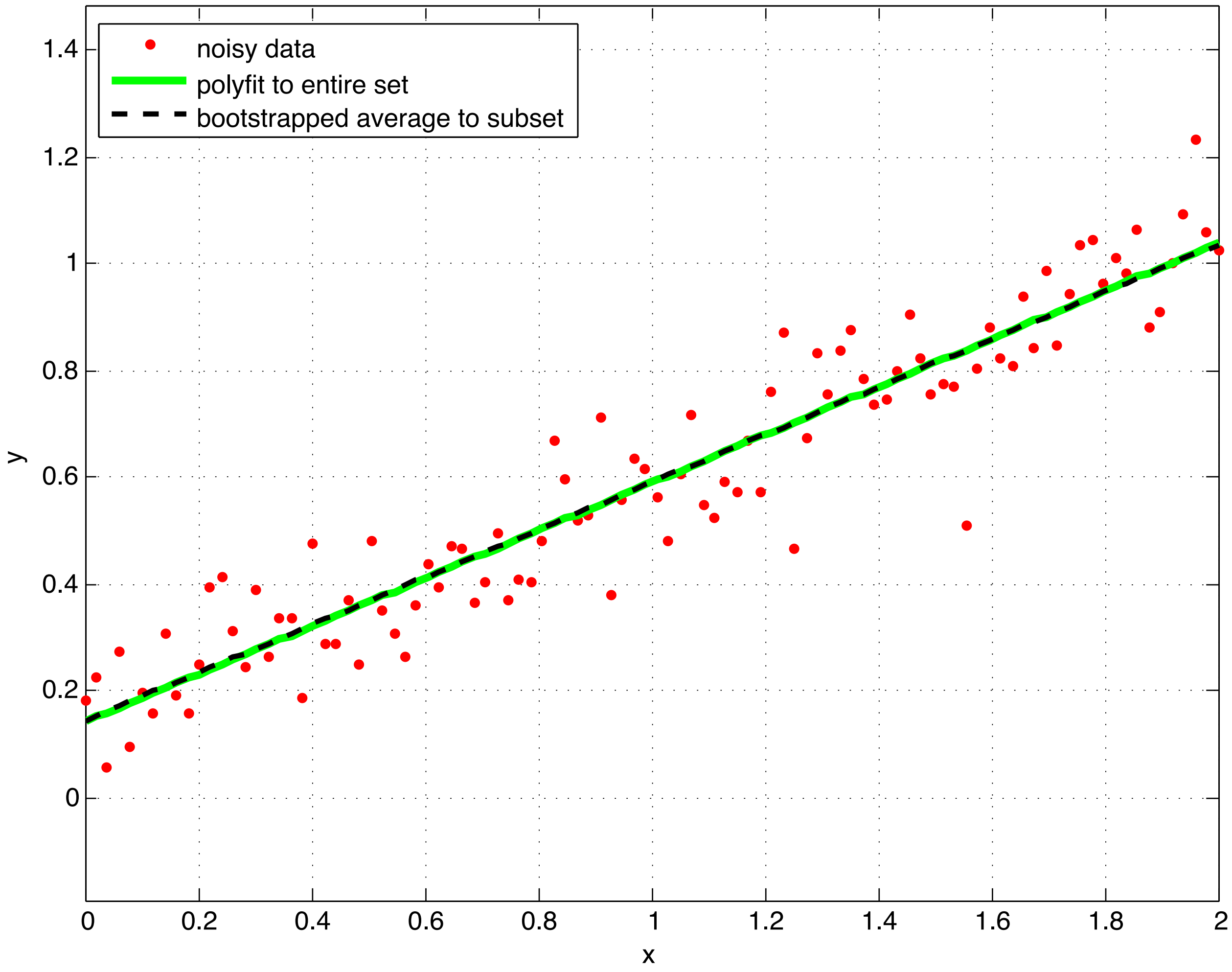
EXresampleLinearFit.m

Note: This subsampling method **does not use replacement** (i.e., a given data point can appear once at most in a resampled set)

```
% ### EXresampleLinearFit.m ###
% [see also EXregression1.m]
clear
% -------------------------------------
xR= [0 2];     % specify range of x-values
xNum= 100;  % number of 'data' points
%  choose parameters of linear function (y=aD+bD*x+noise)
aD= 0.15;   % intercept
bD= 0.44;   % slope
noiseF= 0.1;   % noise factor {0.1}
M= 100;  % # of resamples
range= 1/3; % range to vary span over [0,1]
% -------------------------------------
data.x= linspace(xR(1),xR(2),xNum);
data.y= aD+ bD*data.x + noiseF*randn(numel(data.x),1)'; % determine noisy (straight) line
disp(['==================================']);
disp(['Base function: y= a+b*x= ',num2str(aD),' + ',num2str(bD),'*x (+ Gaussian noise)']);
N= numel(data.x);    % total number of points
% ------------
% Linear regression via built-in Matlab function to ENTIRE data set
[p0,S]= polyfit(data.x,data.y,1);
% also calculate r^2 (coefficient of determination) via
% http://www.mathworks.com/help/matlab/data_analysis/linear-regression.html?refresh=true
yfit = polyval(p0,data.x);    % this line is equivalent to > yfit =  p(1) * x + p(2);
yresid = data.y - yfit;
SSresid = sum(yresid.^2);
SStotal = (length(data.y)-1) * var(data.y);
RS0= 1 - SSresid/SStotal;
disp(['polyfit to entire data set: y= ',num2str(p0(2)),' + ',num2str(p0(1)),'*x (r^2 = ',num2str(RS0),')']);
% ------------
% Resampling: regression on randomly chosen subsets of the data
for nn=1:M
    temp= randperm(N);              % create randomresampling index
    tINDX= temp(1:ceil(range*N));   % trim to subset of specified size
    [p,k]= polyfit(data.x(tINDX),data.y(tINDX),1);  % fit for subset
    fP(nn,:)= p;  % store values away
end
msg = ['Bootstrapped fit (+/- STD): y =',num2str(mean(fP(:,2))),' (',num2str(std(fP(:,2))),...
    ') + x*',num2str(mean(fP(:,1))),' (',num2str(std(fP(:,1))),')']; disp(msg);
% ------------
% visualize
figure(1); clf;
plot(data.x,data.y,'r.');   % original 'data' points
hold on; grid on; xlabel('x'); ylabel('y'); axis([xR(1) xR(2) min(data.y)-0.25 max(data.y)+0.25]);
plot(data.x,p0(2)+p0(1)*data.x,'g-','LineWidth',3); % fit to entire data set
plot(data.x,mean(fP(:,2))+mean(fP(:,1))*data.x,'k--','LineWidth',2); % fit from bootstrapped calculation
legend('noisy data','polyfit to entire set','mean resampled trend','Location','NorthWest')
```

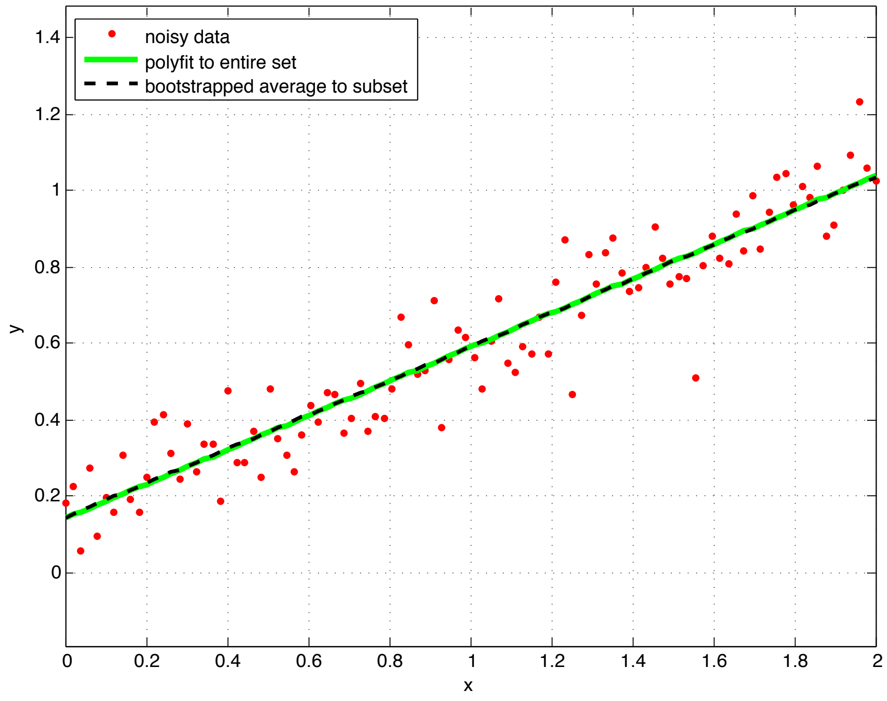
EXresampleLinearFit.m

```
M= 100;  % # of resamples
range= 1/3; % range to vary span over [0,1]
```



```
% Base function: y= a+b*x= 0.15 + 0.44*x (+ Gaussian noise)
% polyfit to entire data set: y= 0.14192 + 0.44855*x (r^2 = 0.89384)
% Bootstrapped fit (+/- STD): y =0.14512 (0.026083) + x*0.44623 (0.023877)
```

- Wait. What was the point of this?



```
% Base function: y= a+b*x= 0.15 + 0.44*x (+ Gaussian noise)
% polyfit to entire data set: y= 0.14192 + 0.44855*x (r^2 = 0.89384)
% Bootstrapped fit (+/- STD): y =0.14512 (0.026083) + x*0.44623 (0.023877)
```

Good rule of thumb (from Dahlquist & Bjorck):

"*If a part of a problem can be treated with analytical or traditional numerical methods, then one should use such methods*"

Answer: The value of this demonstration was to illustrate, using an intuitive example, how resampling might be useful in the context of resampling a set of data

→ Note that we used resampling to not only determine a best fit, but also obtain some measure of *uncertainty* associated with such!