Modern statistical methods

Permutation tests, bootstrapping, and cross-validation

Math tools for neural and cognitive science

Lyndon Duong

November 22, 2019

The most applicable ”tools” in math tools

- Linear algebra
  - Vector geometry
  - Singular value decomposition
- Linear systems and shift-invariant systems
  - Changes of basis
  - Fourier transforms
- Probability and statistics
  - Bayes theorem
  - **Computational methods** (bootstrap, permutation test, x-val.)

Computational methods have taken over

The randomization test is rarely used, and you may never encounter a paper that analyzes data using it. The randomization test (with more than a tiny sample) requires a computer, and computers weren't cheap until recently. And even with a computer, randomization tests can be slow. For these reasons, the t test is far more popular than the randomization test. If you find the logic of a randomization test to be easier to follow than the logic of the t test, you can consider the P value from a t test to be an approximation for the P value from the randomization test.

- Harvey Motulsky, Intuitive Biostatistics (1st Ed.) 1999

(Recent editions of the book no longer say this)

Computational methods have taken over

A random paper I was reading:
- 9 (main text) figures
- 1 t-test
- 15+ bootstrap statistical tests

Neuron

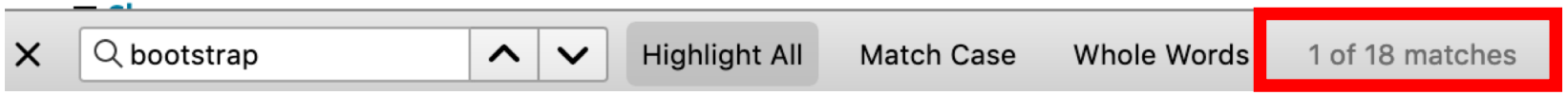
Volume 87, Issue 4, 19 August 2015, Pages 869-881

CellPress

Article

Abstract Context Representations in Primate Amygdala and Prefrontal Cortex

A. Saez [1], M. Rigotti [1, 2], S. Ostojic [3], S. Fusi [1, 4], C.D. Salzman [1, 4, 5, 6]



Confidence intervals

- Mean +/- std
- Mean +/- sem
- Mean +/- 1.96 sem

The t-test

Standard (one-sample):

Null hypothesis: my **one** collected dataset comes from a normal distribution with mean=0 and var=1

Unpaired (independent):

Null hypothesis: **two** collected datasets arise from the **same** underlying distribution

Paired (dependent):

Null hypothesis: The change **between two** collected datasets is not different than zero

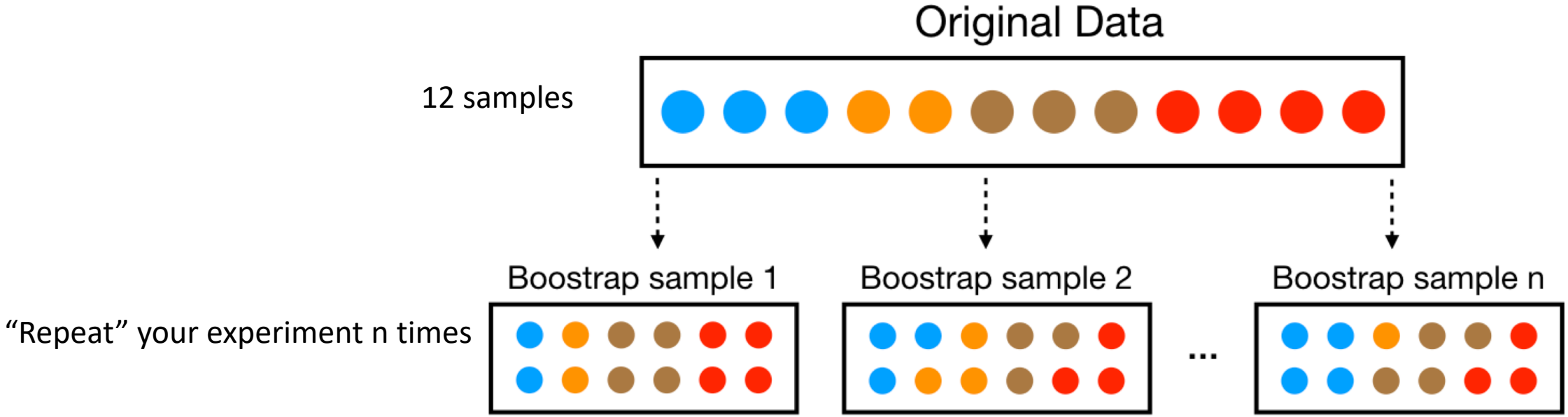
Assumptions in classical statistics

Parametric tests:

- Your data come from a **specified distribution**
- T-tests, ANOVA, pearson correlation (optimal when data is normal)
- Etc etc.

Bootstrapping:

"Running a more experiments" with the data you already collected



Bootstrap: Aspirin

	(fatal plus non-fatal) heart attacks	subjects
aspirin group:	104	11037
placebo group:	189	11034

Is this effect significant?

Bootstrap: Aspirin

	(fatal plus non-fatal) heart attacks	subjects
aspirin group:	104	11037
placebo group:	189	11034

$$Ratio = \frac{\frac{H_{Aspirin}}{Tot_{Aspirin}}}{\frac{H_{Placebo}}{Tot_{Placebo}}} = \frac{\frac{104}{11037}}{\frac{189}{11034}} = 0.5501$$

Aspirin group is subject to 55% of heart attacks as placebo group

But how sure are we of this estimate?

Bootstrap: Aspirin

	(fatal plus non-fatal) heart attacks	subjects
aspirin group:	104	11037
placebo group:	189	11034

How I think about this problem:

- You run *one* experiment with two groups: 11037 subjects, and 11034 subjects
- You don't have enough time/money to re-run this experiment
- *Assume* your data is a good representation of the true population
- Simulate N experiments, drawing samples from your collected data

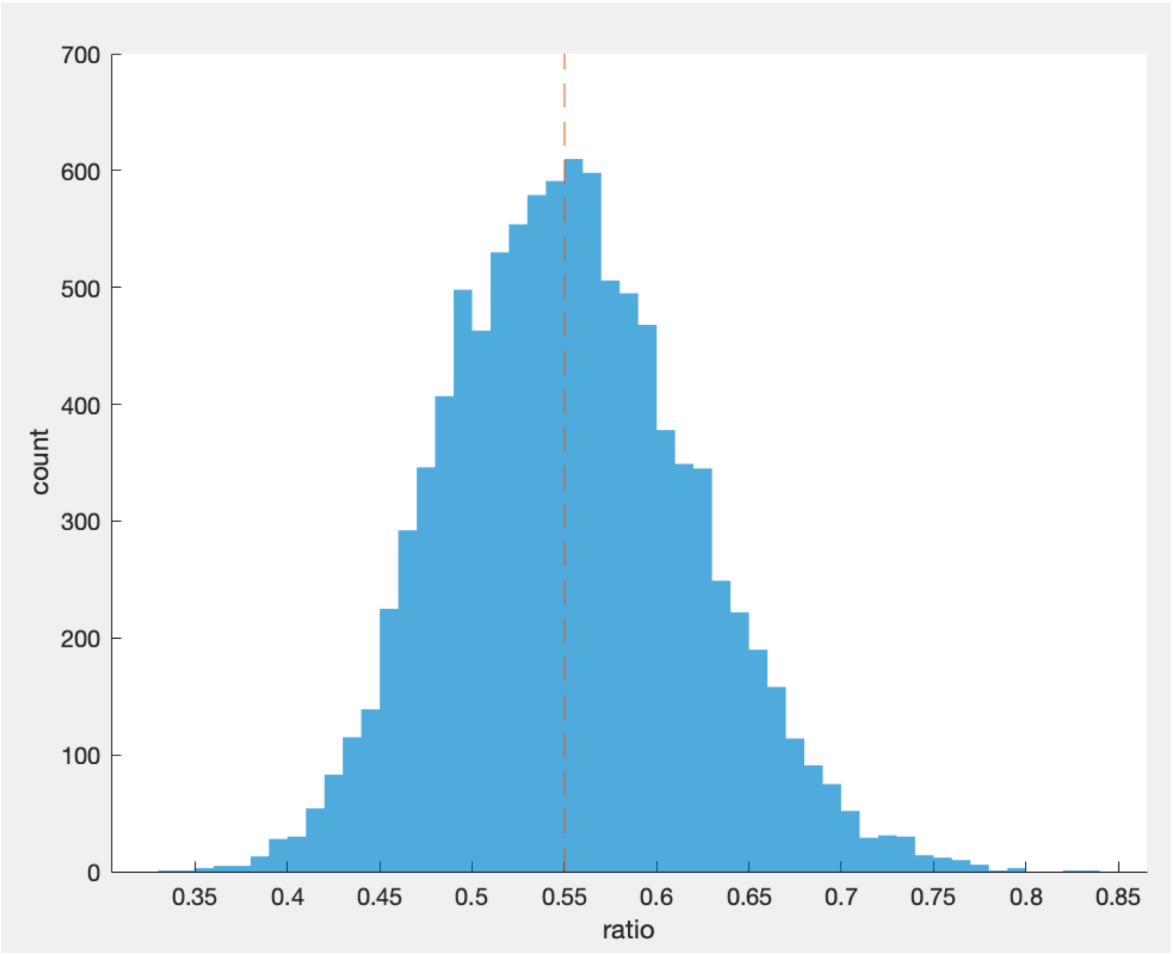
Bootstrap: Aspirin

```
aspirin_heart = 104;
aspirin_total = 11037;

placebo_heart = 189;
placebo_total = 11034;

% like flipping a coin…
asp_data = zeros(aspirin_total,1); % 11037 total subjects, all zero
asp_data(1:aspirin_heart) = 1; % set 104 subjects to 1 (1 = heart attack)

placebo_data = zeros(placebo_total, 1); % 11034 tot subjects, all zero
placebo_data(1:placebo_heart) = 1; % 189 had heart attacks
```

Bootstrap: Aspirin

How did I do this?

N=10,000 bootstrapped ratios

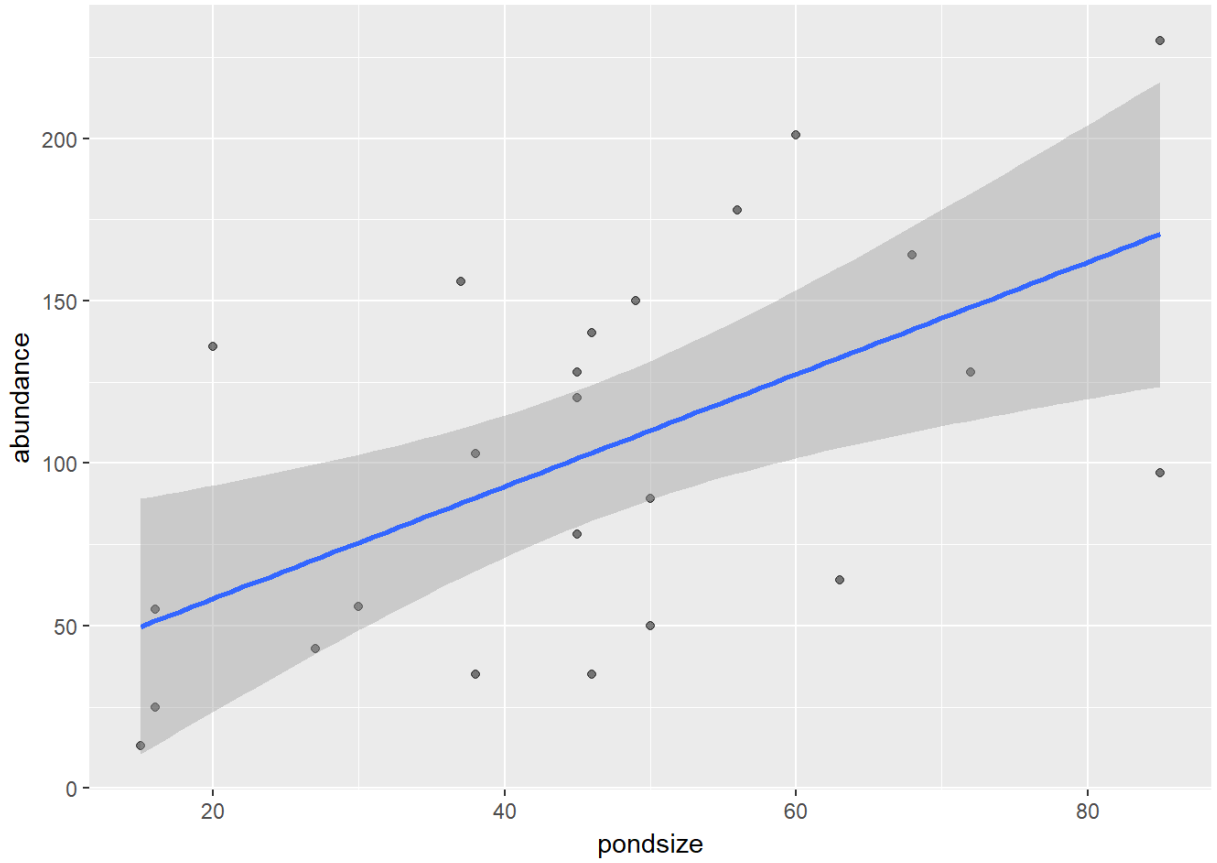


Bootstrap: Aspirin

```
aspirin_heart = 104;
aspirin_total = 11037;

placebo_heart = 189;
placebo_total = 11034;

% like flipping a coin…
asp_data = zeros(aspirin_total,1); % 11037 total subjects, all zero
asp_data(1:aspirin_heart) = 1; % set 104 subjects to 1 (1 = heart attack)

placebo_data = zeros(placebo_total, 1); % 11034 tot subjects, all zero
placebo_data(1:placebo_heart) = 1; % 189 had heart attacks
```

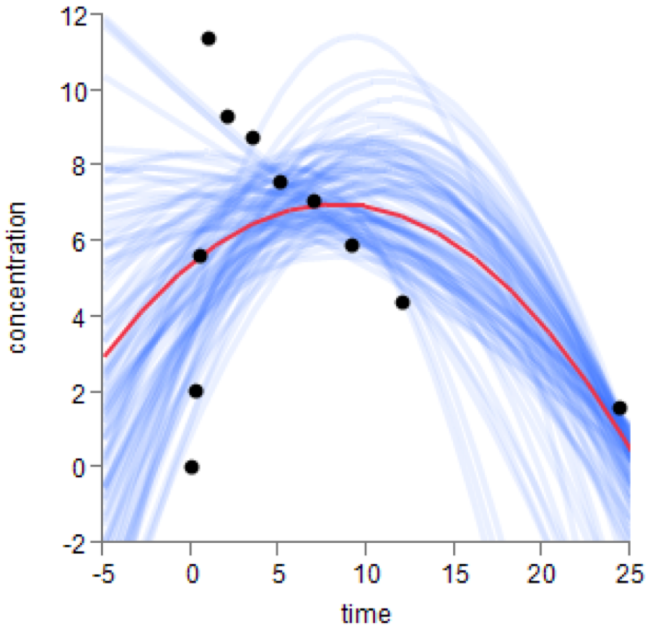
```
ratio_empirical = (aspirin_heart/aspirin_total)/(placebo_heart/placebo_total);
n_boot = 10000;
ratio_boot = zeros(n_boot, 1);

for boot = 1:n_boot
    boot_asp = randsample(asp_data, aspirin_total, 'true');
    boot_placebo = randsample(placebo_data, placebo_total, 'true');
    n_boot_asp = sum(boot_asp);
    n_boot_placebo = sum(boot_placebo);
    ratio_boot(boot) = (n_boot_asp/aspirin_total)/(n_boot_placebo/placebo_total);
end
```

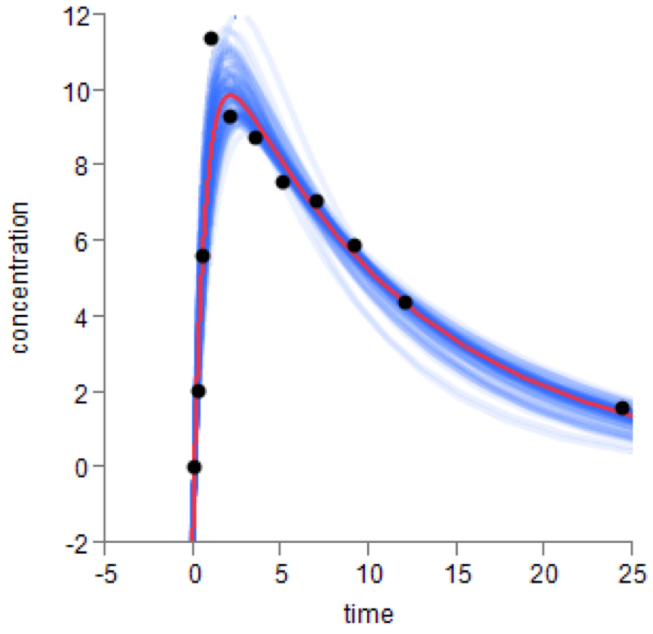
Bootstrap: model fitting

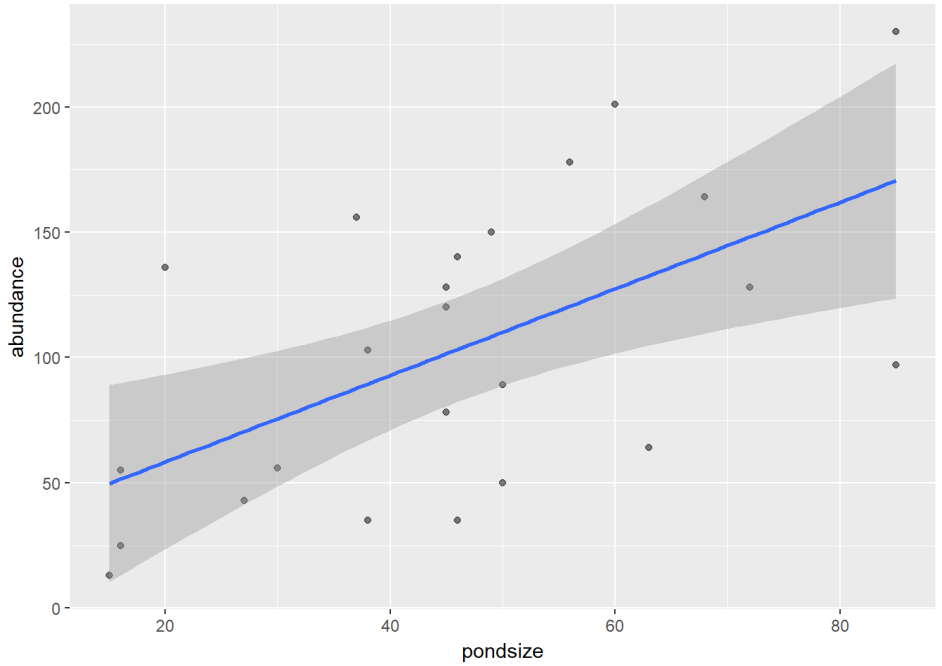


Bootstrap: model fitting

Fitting models:





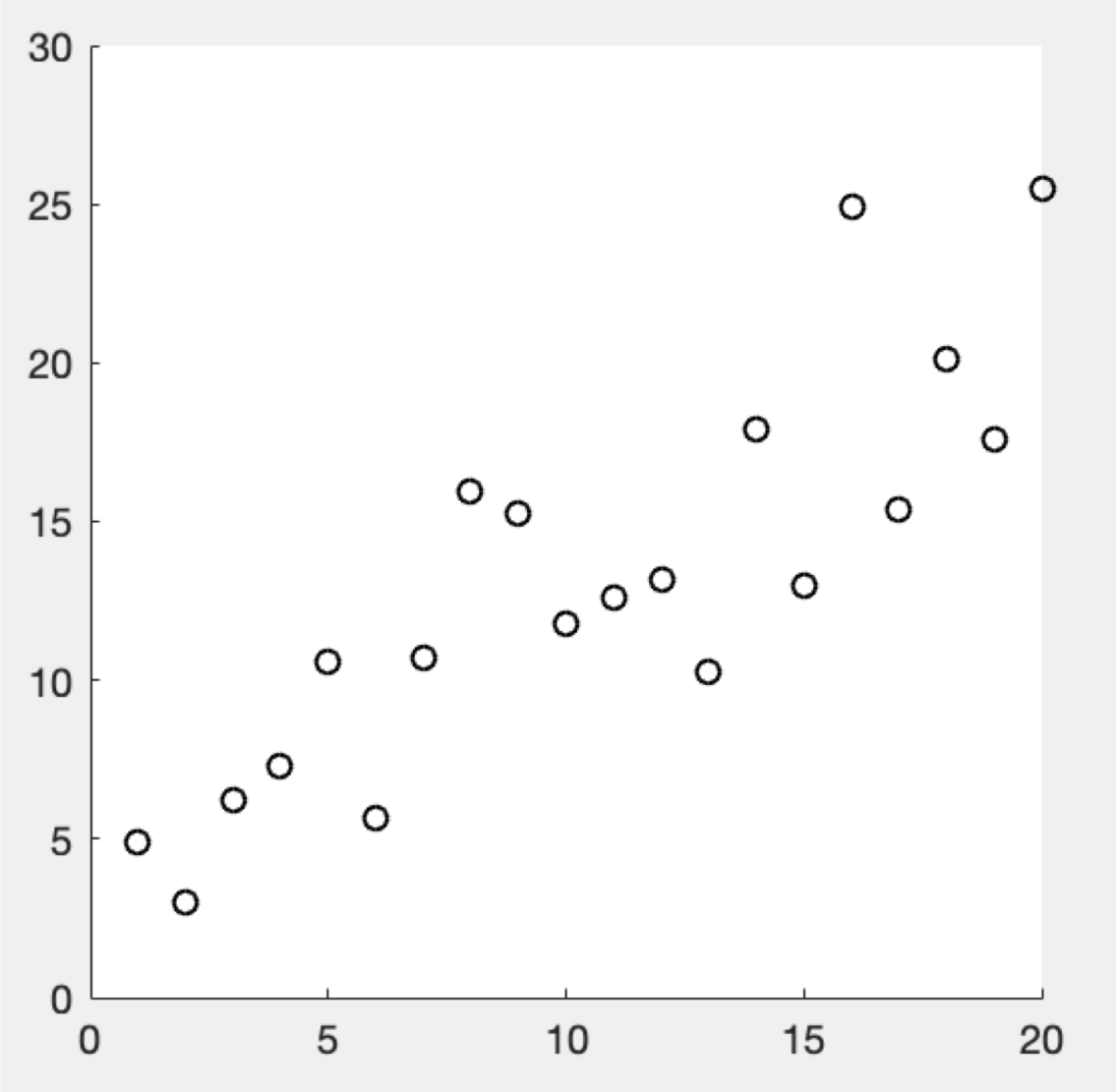


Bootstrap: model fitting

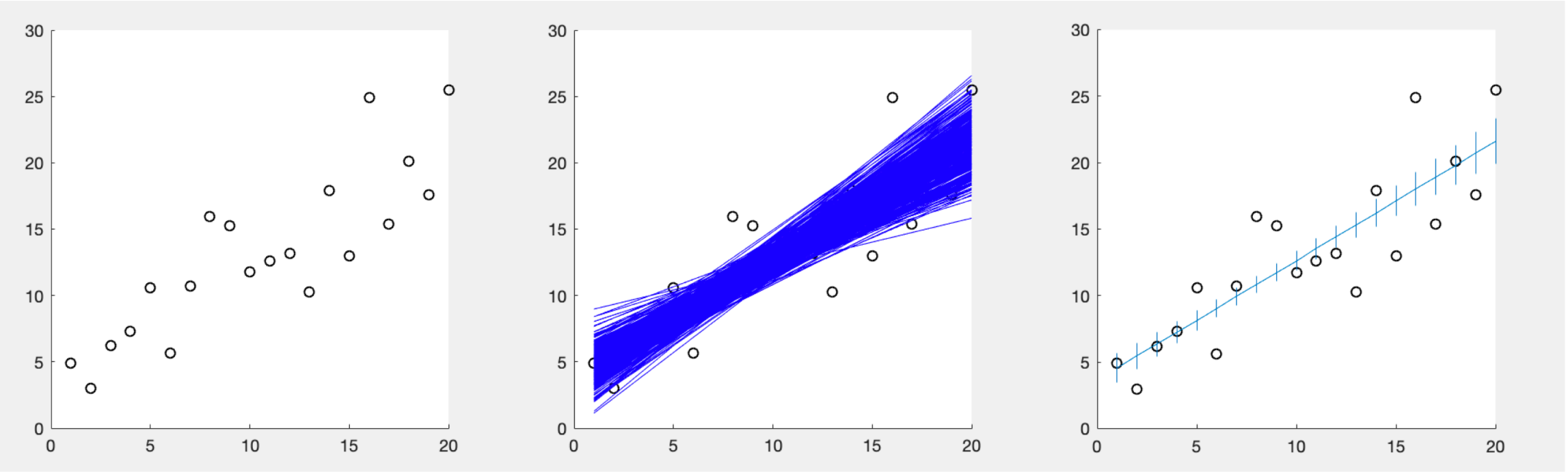
I created this linear, noisy data

```
n_pts = 20;
X = (1:n_pts)';
Y = (1*X) + 2 + 3*randn(n_pts,1);
```

How do I fit it and generate confidence intervals for my fits?



Bootstrap: model fitting:



Bootstrap: model fitting

```
n_boot = 500;
boot_fit = zeros(n_boot, n_pts); % create empty matrix for fits
boot_beta = zeros(n_boot,2); % create empty matrix for fit coeffs
model = [ones(n_pts,1) X];

for boot=1:n_boot
    boot_ind = randi(n_pts, n_pts, 1); % index to resample data
    boot_model = model(boot_ind,:); % subsample rows of model
    boot_Y = Y(boot_ind); % subsample % subsample corresponding Y- data

    beta = regress(boot_Y, boot_model); % fit this boostrapped data
    boot_beta(boot,:) = beta; % store the betas

    boot_fit(boot,:) = model*beta; % get the fit by applying beta to model
    plot(boot_fit(boot,:),'b')
end
```

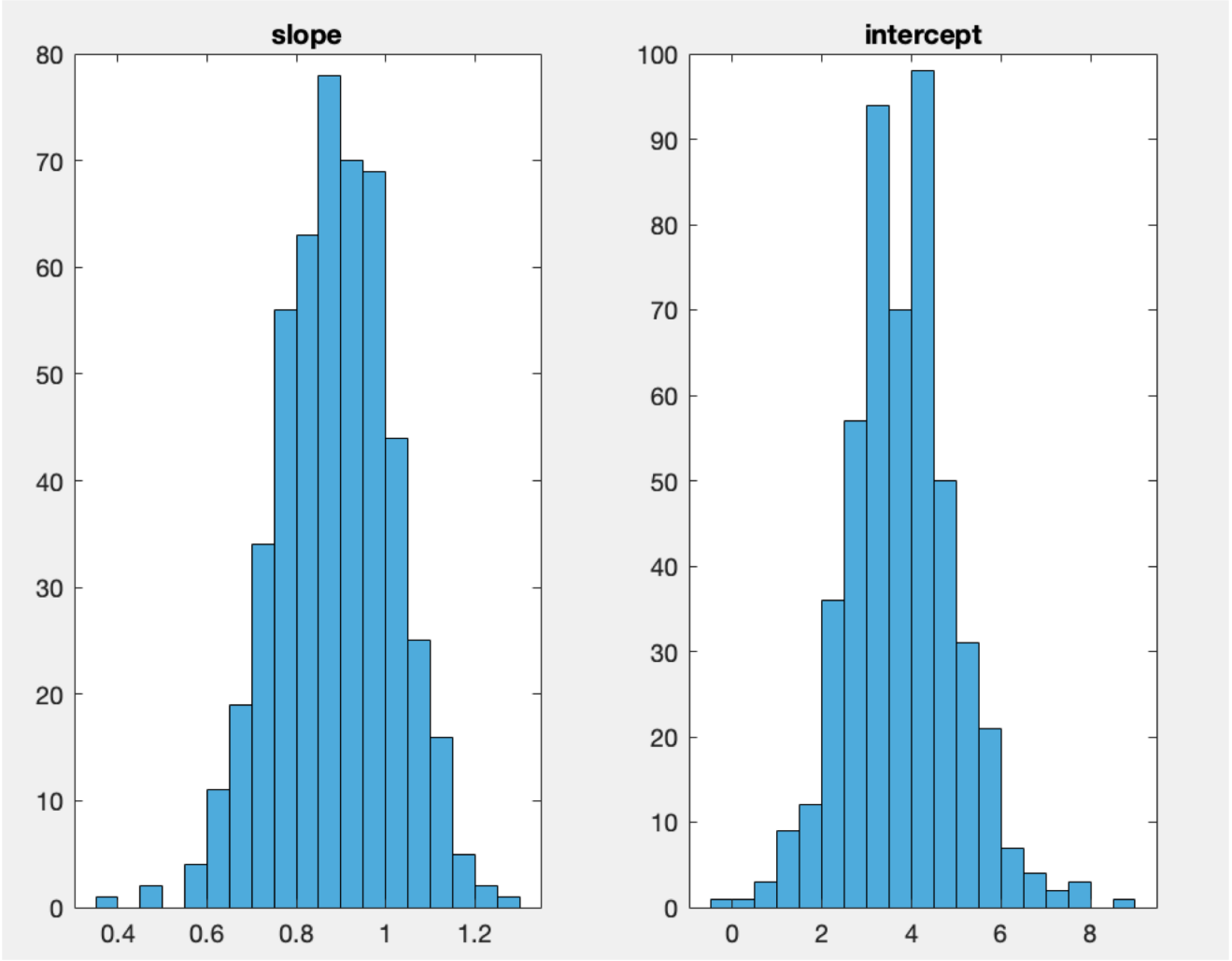
Bootstrap: model fitting

```
n_pts = 20;
X = (1:n_pts)';
Y = (1*X) + 2 + 3*randn(n_pts,1);
```

Real slope = 1
Real intercept = 2

Non-bootstrapped slope = 0.8979
Non-bootstrapped intercept = 3.65

<u>In a paper</u>
We fit a linear model to the data (slope: 0.89 ± 0.1 ; intercept: 3.65 ± 1.2, mean ± bootstrap stdev)



Std = 0.13

Std = 1.2

Permutation (randomization) testing

Bootstrapping: generally used to generate confidence intervals about an estimate

- "How certain am I about my observed estimate (e.g. mean, median, ratio, etc.)"

Permutation test: generate a ***null distribution of estimates*** with which to test your estimate against

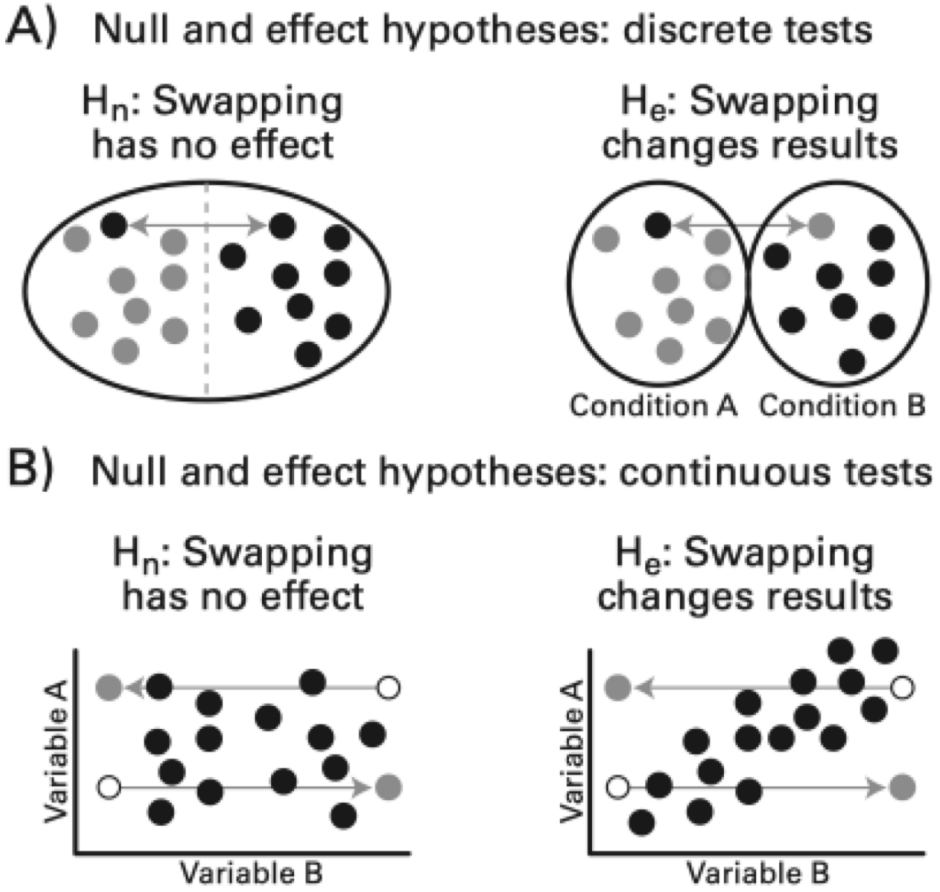
- "Is my observed estimate different than chance?"

Permutation testing

Steps:
1) Assess your data/generate some empirical observation
2) Shuffle your data
3) Re-assess/generate new observation using this shuffled data
4) Compare the shuffled observation to your empirical observation
5) Return to step 2 until you have shuffled a total of N times

More efficiently:
1) Assess your data/generate some empirical observation
2) Shuffle your data N times
   - Re-assess/generate new observation with each shuffle
   - This generates a ***distribution*** of N shuffled observations
3) Compare your empirical observation to the shuffled distribution

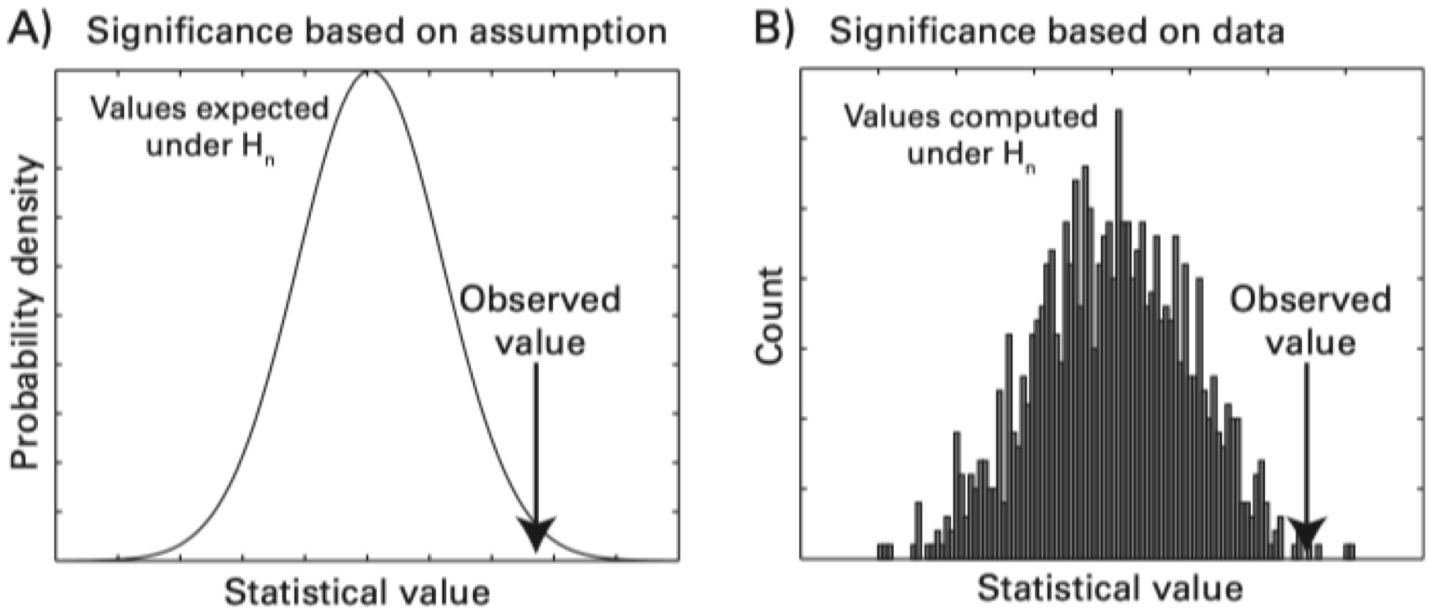
One shuffle example:



Analyzing neural time series data, Mike X. Cohen

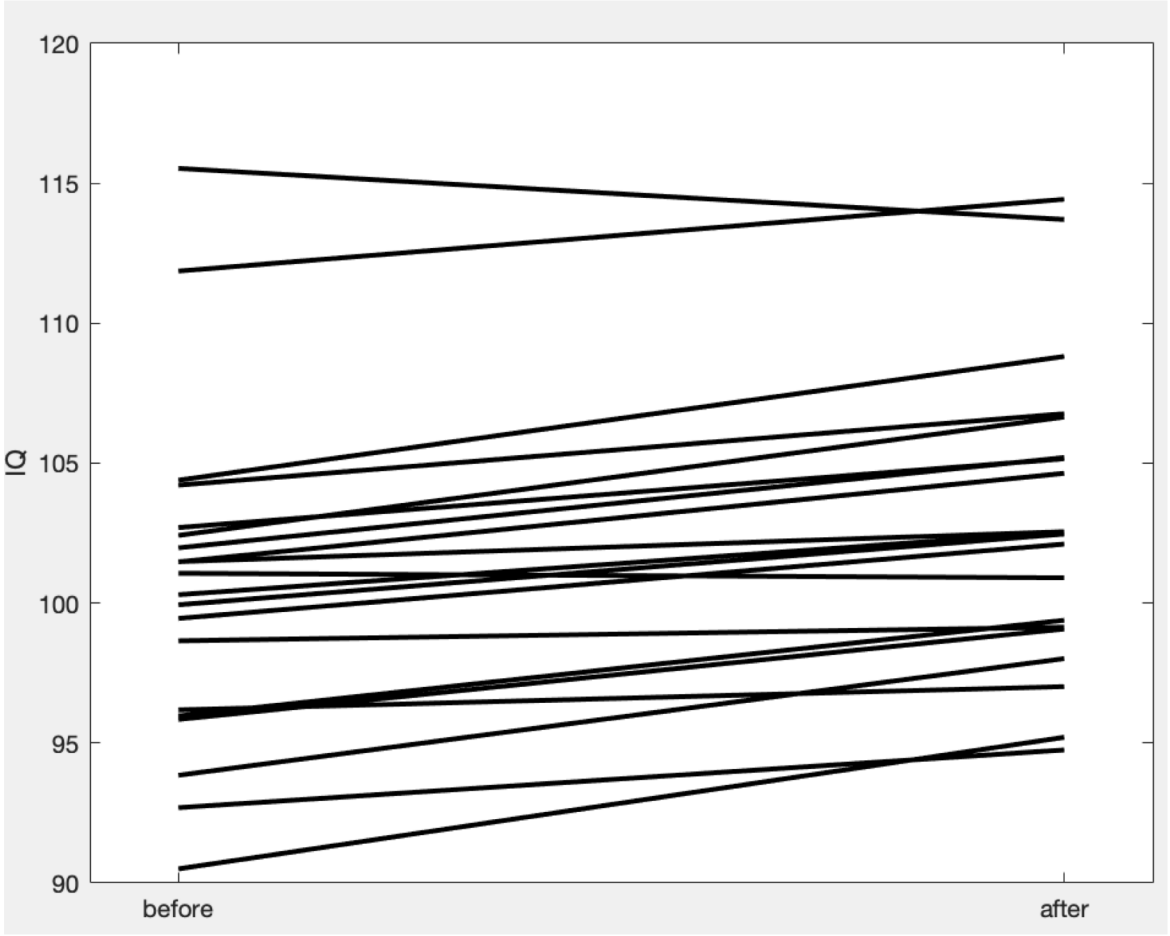
(Re)define p-values

- How many shuffled estimates lie below/above your real estimate?
- N = number of shuffles

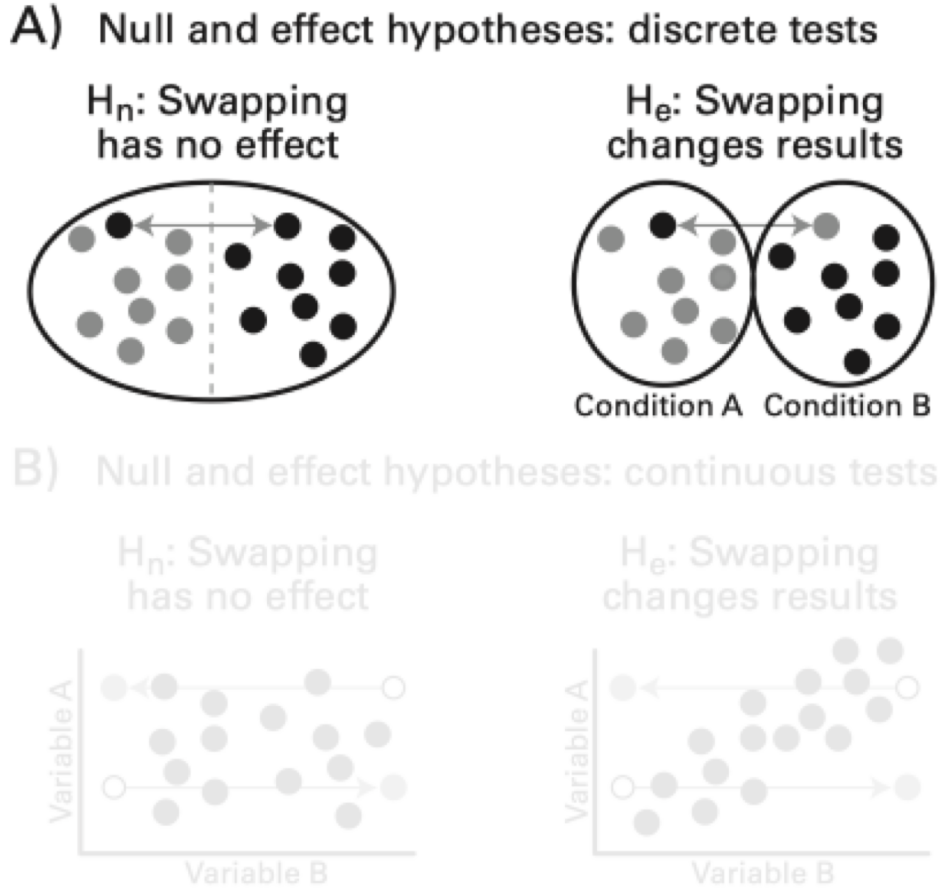


Analyzing neural time series data, Mike X. Cohen

Permutation test example: paired test



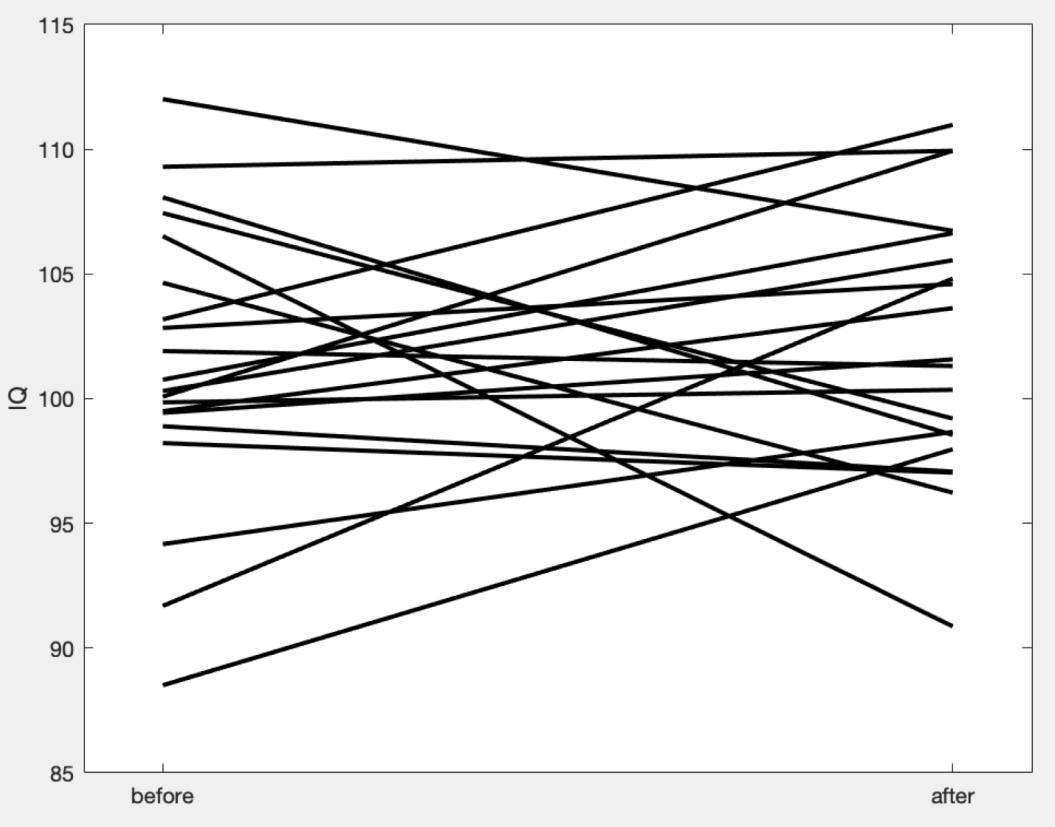
Empirical mean difference = 2.55 IQ pts



Analyzing neural time series data, Mike X. Cohen

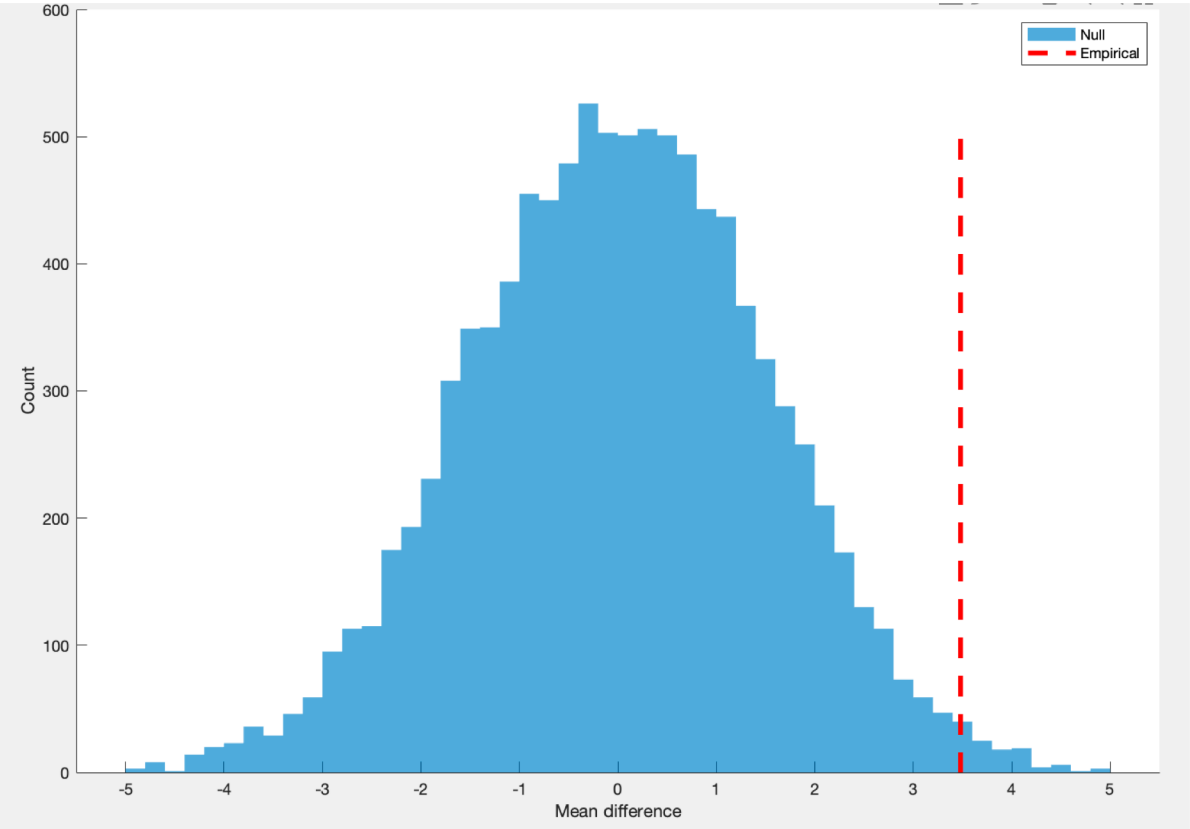
Suppose you created a drug that makes people smarter…. Can we prove that this effect is real *without a paired t test?*

Permutation test example: paired test



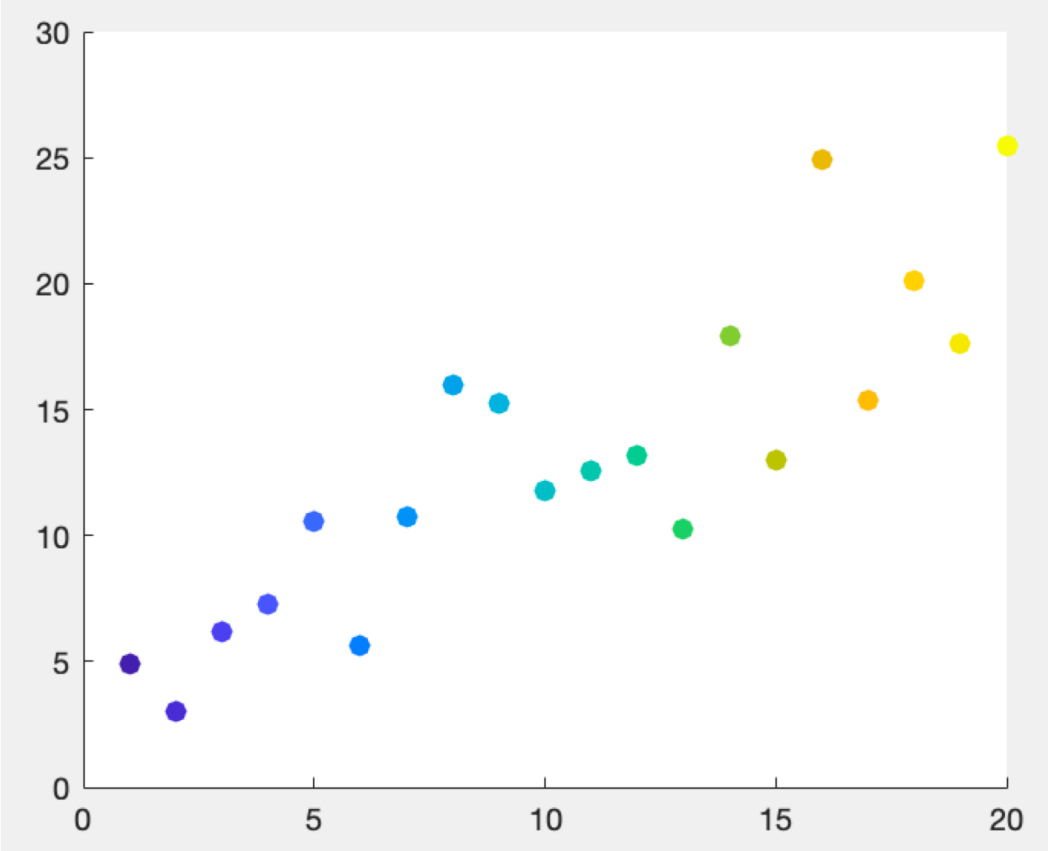
1 shuffle example

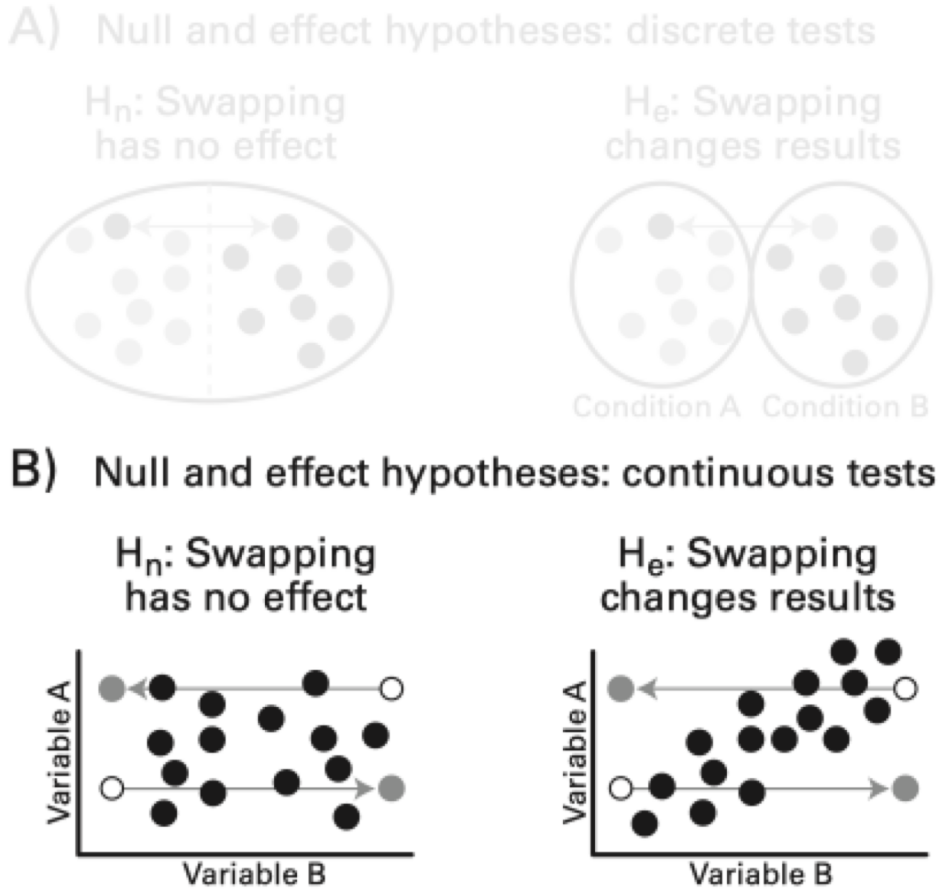
Mean diff = 0.7161 IQ pts



10K shuffles and their mean differences

Permutation test example: continuous example



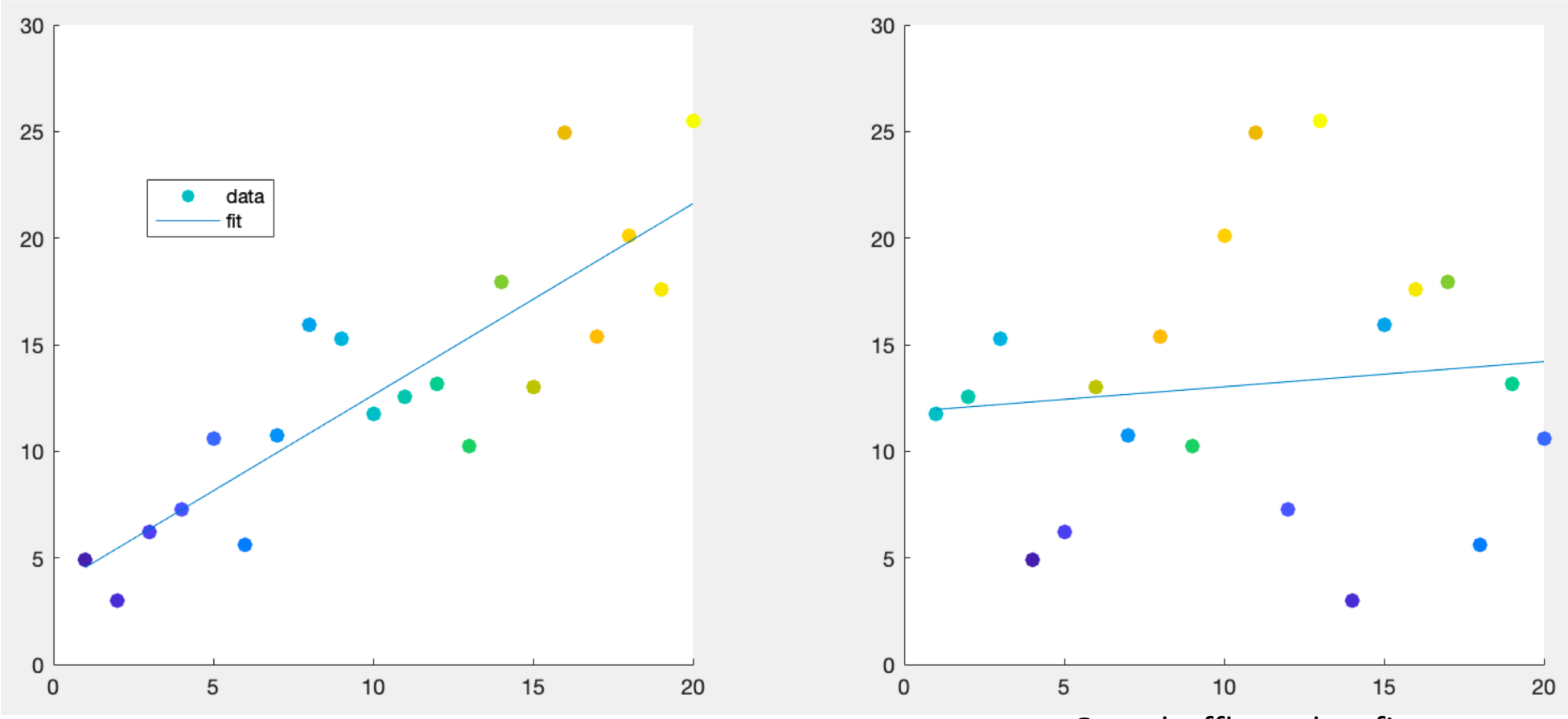


Analyzing neural time series data, Mike X. Cohen

How do I quantify statistical significance of my fitted coefficients with a permutation test?

Permutation test example: continuous data

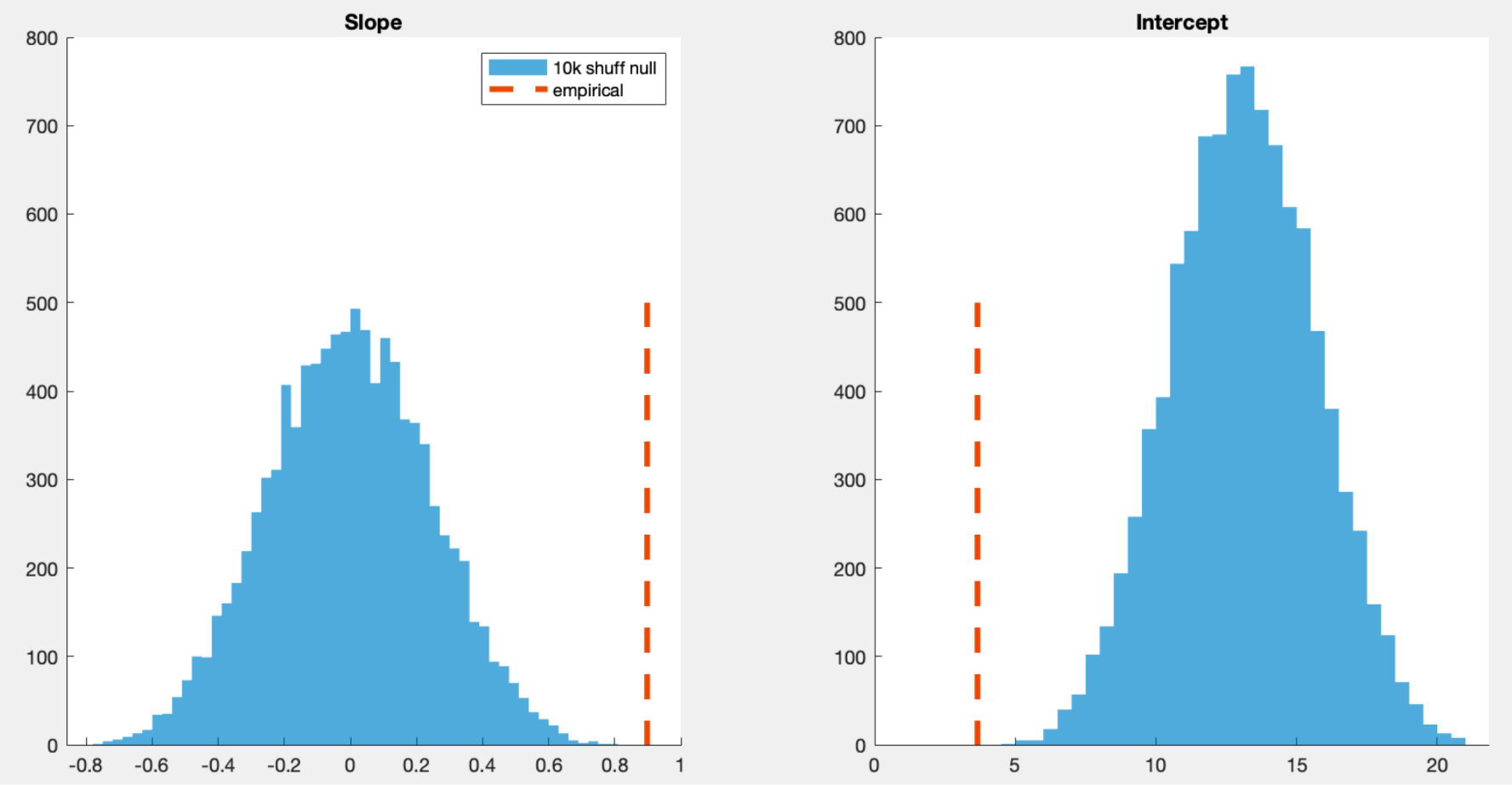
```
Y = (1*X) + 2 + 3*randn(n_pts,1); % Y = mx + b + noise
```



Real data and fit

One shuffle and re-fit
(shuffled x-values)

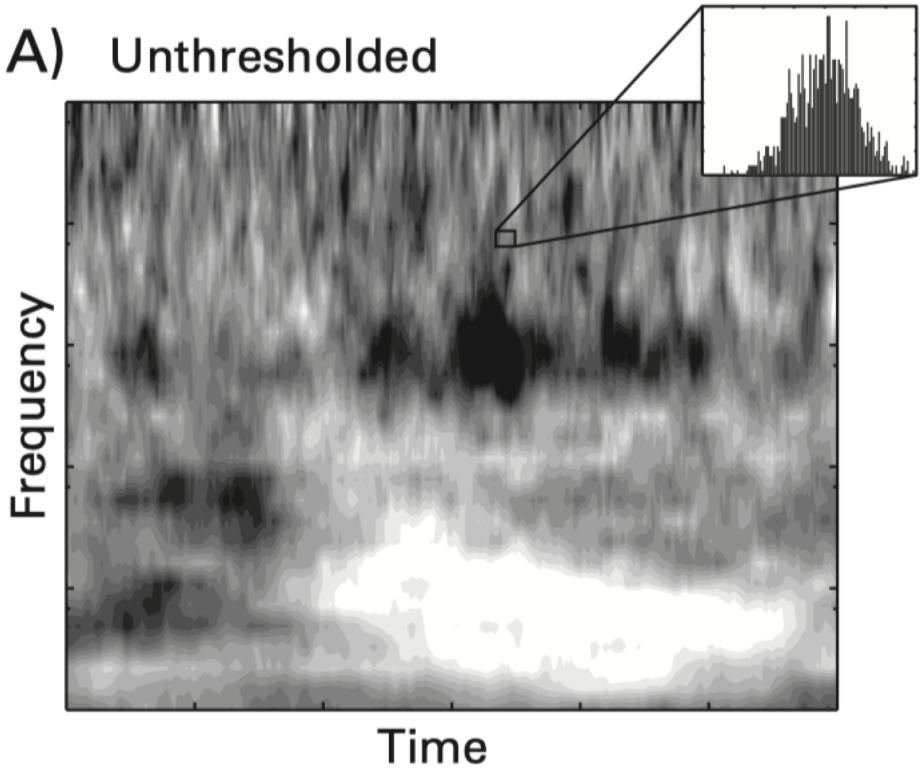
Permutation test example: continuous data



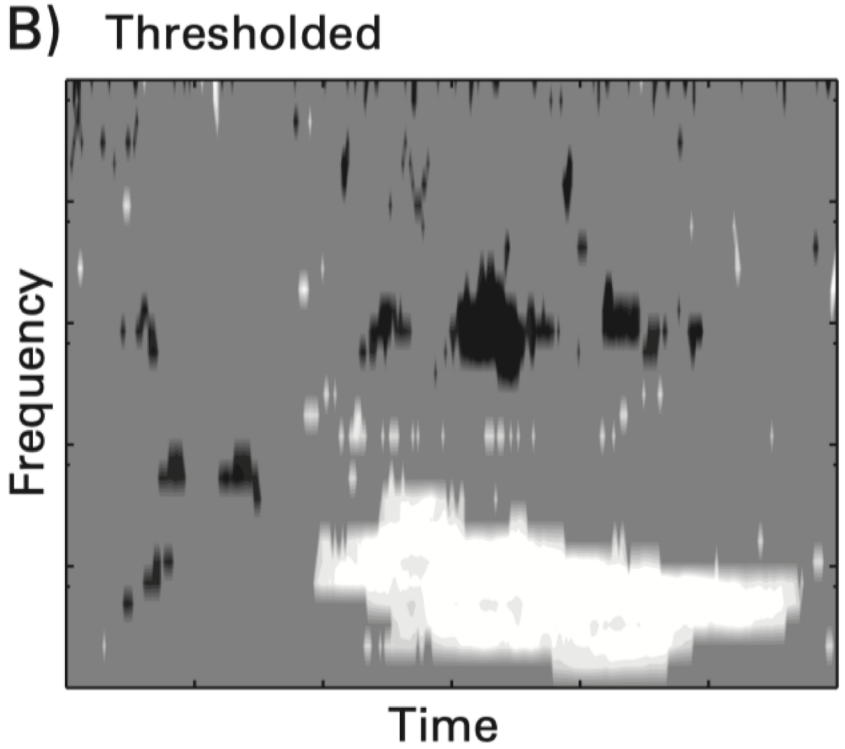
“We fit a linear model […] (slope=0.9, p=???; intercept=3.6, p=???)”

Permutation test example: spectrograms

Observed 'spectrogram' of EEG electrode
Power as a function of time and frequency



Using a permutation test *at each point* (pixel of image)
to see which values are higher/lower than chance



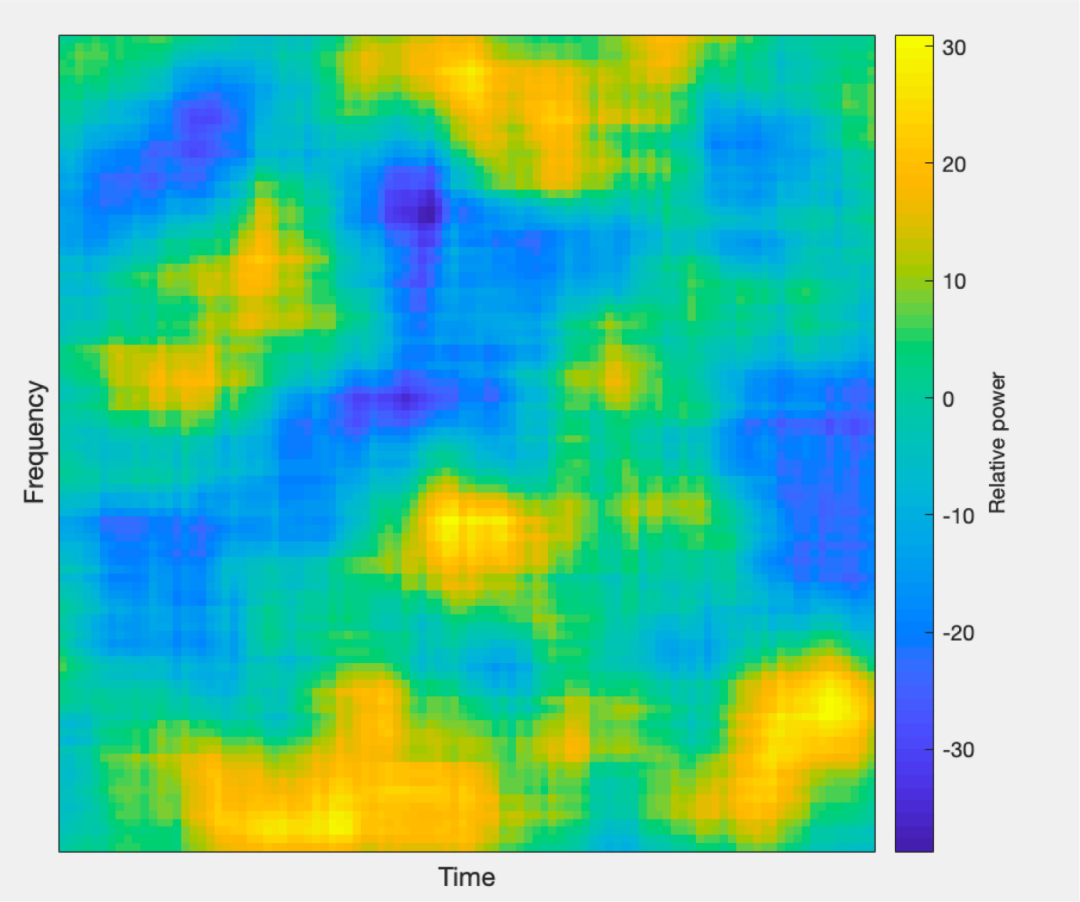
Analyzing neural time series data, Mike X. Cohen

Permutation test: spectrograms

What at which times and frequency is the power different than chance?

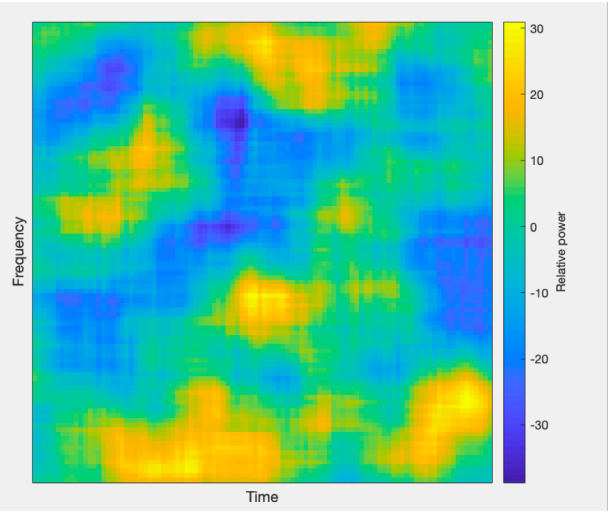
Plot the data

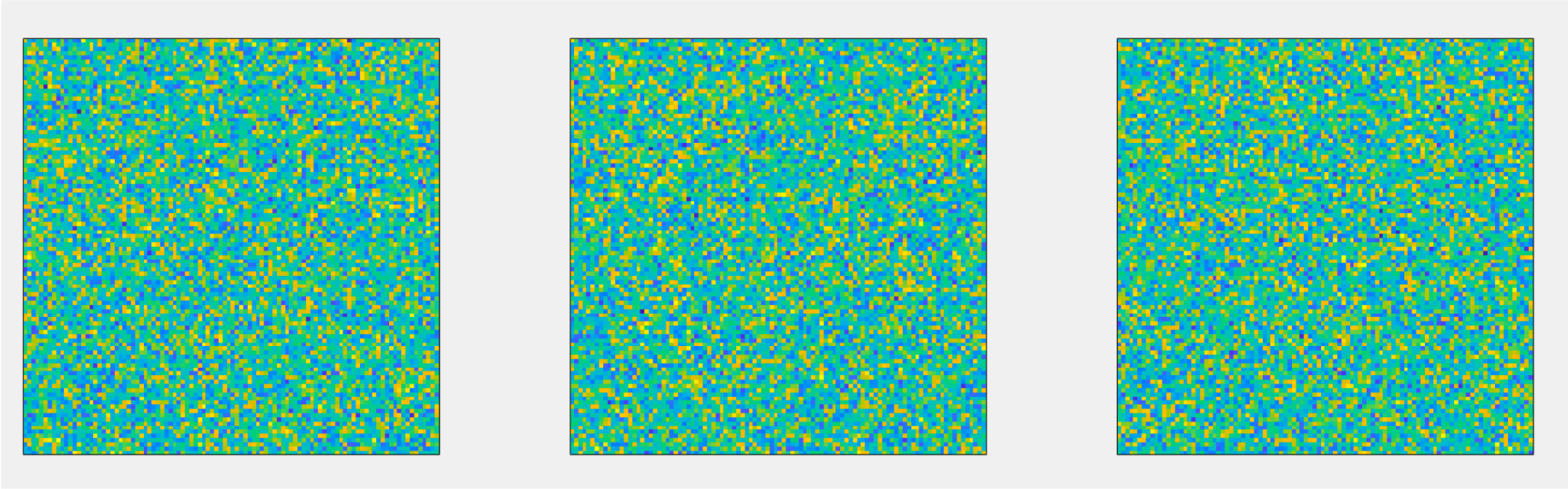
```
clear all
load eeg.mat

imagesc(eeg)
set(gca,'xtick',[],'ytick',[])
xlabel('Time')
ylabel('Frequency')
h=colorbar;
axis square
ylabel(h,'Relative power')
```



Permutation test: spectrograms

- Create a null distribution *at every point* of the spectrogram (‘image pixel’)





Original

Shuffle #1

Shuffle #2

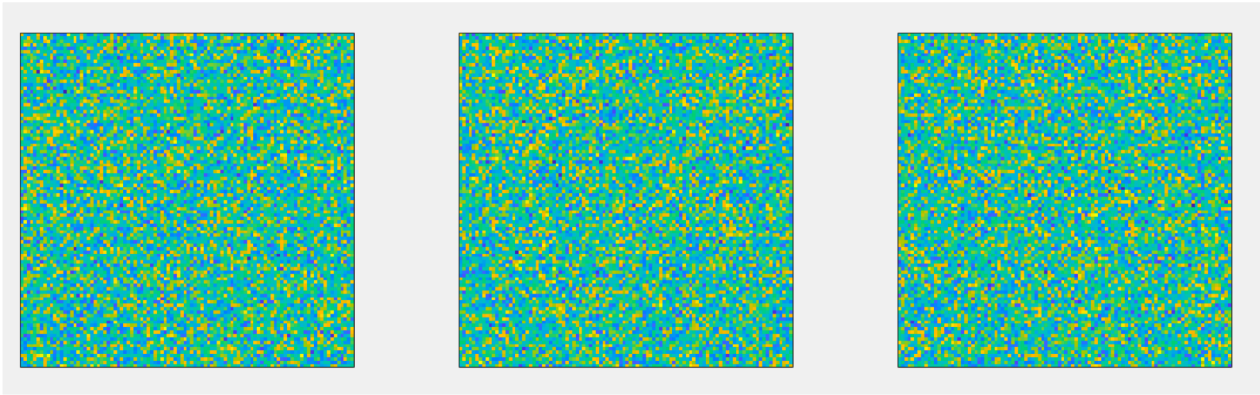
…

Shuffle N

Permutation test: spectrograms

```
n_perm = 5000;  % number of permutations
perm_eeg = zeros(size(eeg, 1), size(eeg,2), n_perm);  % Create 3D null matrix
n_eeg = numel(eeg); % number of elements in eeg data 'image'

for perm = 1:n_perm
    perm_ind = randperm(n_eeg, n_eeg);  % permute indices of matrix
    perm_eeg(:,:,perm) = reshape(eeg(perm_ind), size(eeg));  % create null image
end
```

[This code takes approx. 4sec to run on my machine]



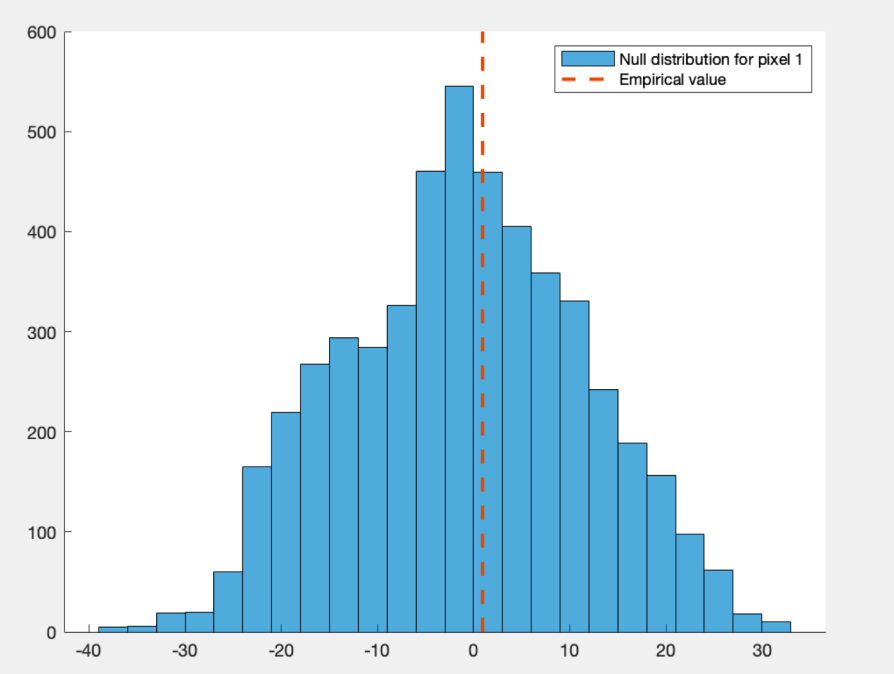
Shuffle #1 Shuffle #2 … Shuffle N

Permutation test: spectrograms

We have a 3D matrix (tensor) of null images

Each individual pixel has a null distribution!

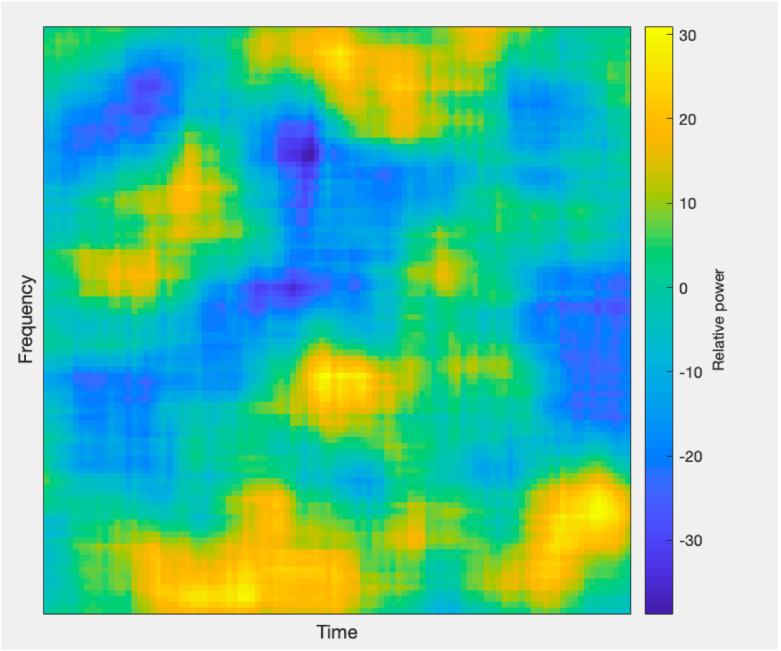
```
figure
hold on
histogram(perm_eeg(1,1,:))  % grab every null value at first pixel
plot([1, 1]*eeg(1), [0,1] * 600,'--','linewidth',2)  % plot the empirical value
legend('Null distribution for pixel 1','Empirical value')
```



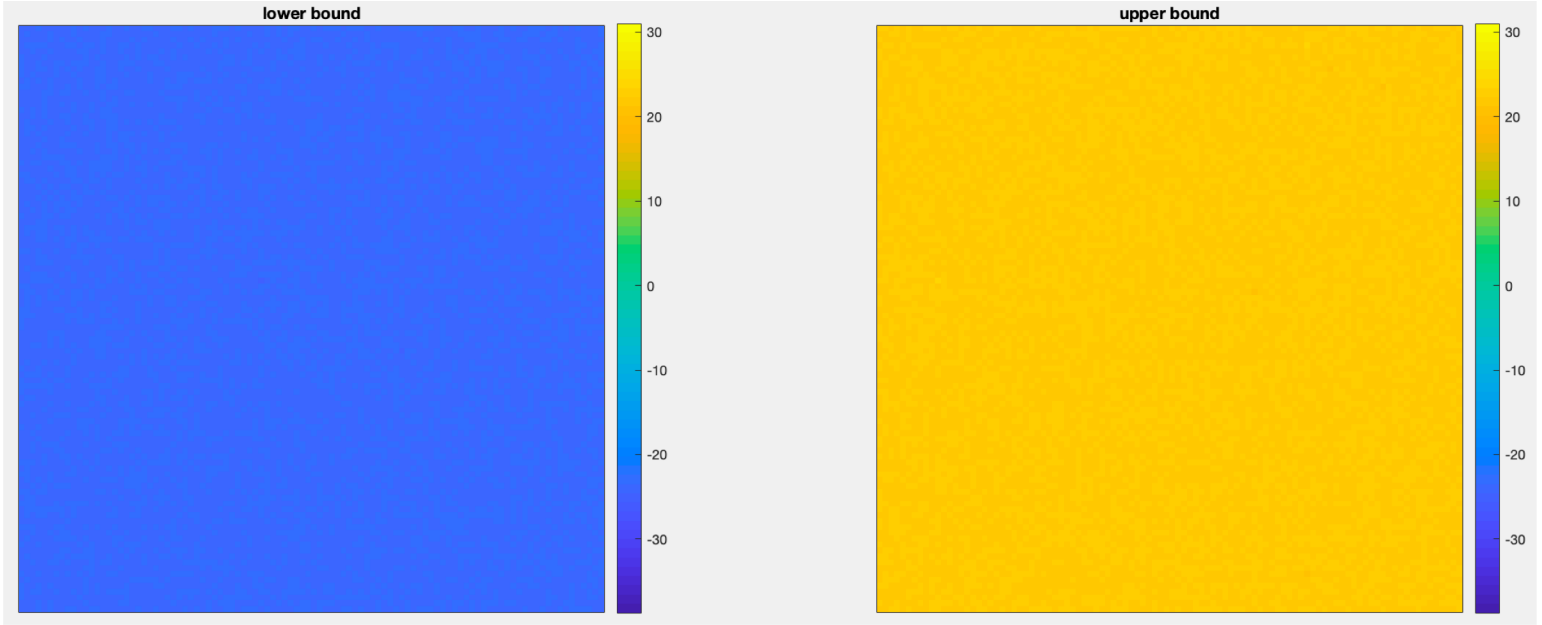
Is the empirical value different than the null?

Permutation test: spectrograms

```
% Find 2.5 and 97.5 percentile for each pixel

thresh_low = prctile(perm_eeg, 2.5, 3);  % find 2.5 %ile
thresh_high = prctile(perm_eeg, 97.5, 3); % find 97.5 %ile
```

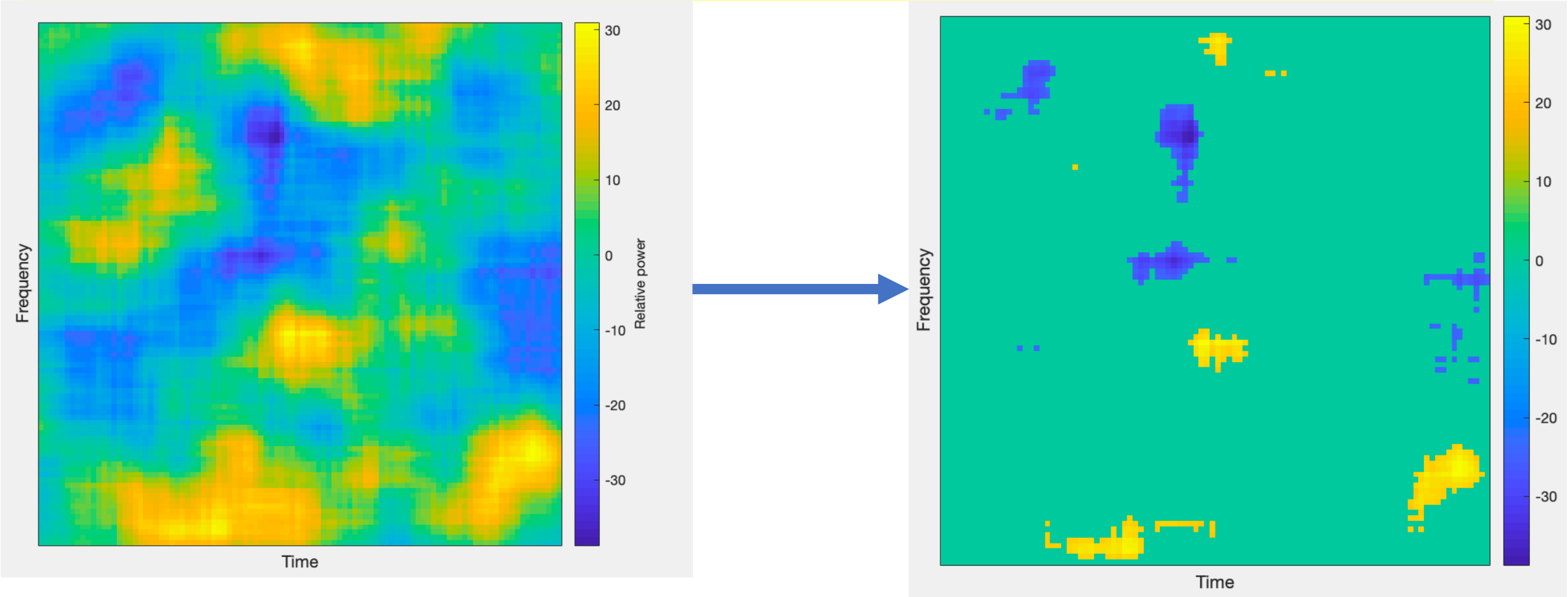


Original



Permutation test: spectrograms

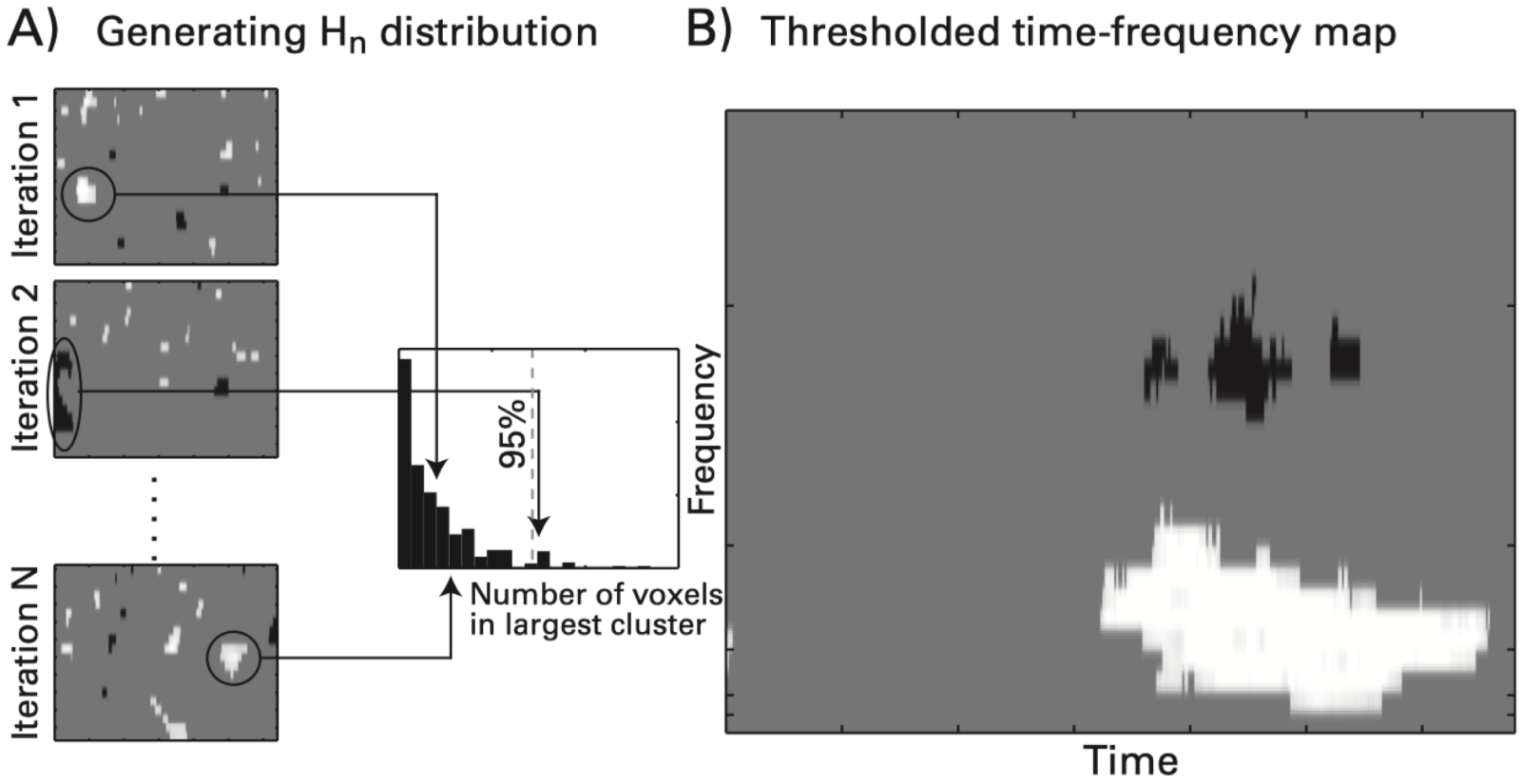
```
% Find 2.5 and 97.5 percentile for each pixel

thresh_low = prctile(perm_eeg, 2.5, 3);  % find 2.5 %ile along 3rd dimension
thresh_high = prctile(perm_eeg, 97.5, 3);
mask = eeg<thresh_low | eeg>thresh_high;
thresholded = eeg .* mask;
```



Permutation test: powerful enough to test *anything* you want

Statistically significant clustering

Test statistic:
Number of contiguous pixels
Shuffle and count # pixels in largest cluster



Analyzing neural time series data, Mike X. Cohen

Cross-validation

“I wish we had more data…”

- You, probably

How do you assess whether your fit or model is “good”?

R-squared

Mean-squared error

Why do we create models in the first place?

- 99.999% of the time, we only have part of a complete dataset
- We've encountered this idea before: sample vs population statistics
- E.g. Expected value & variance → What I expect the mean and variance to be in the **limit of infinite data**

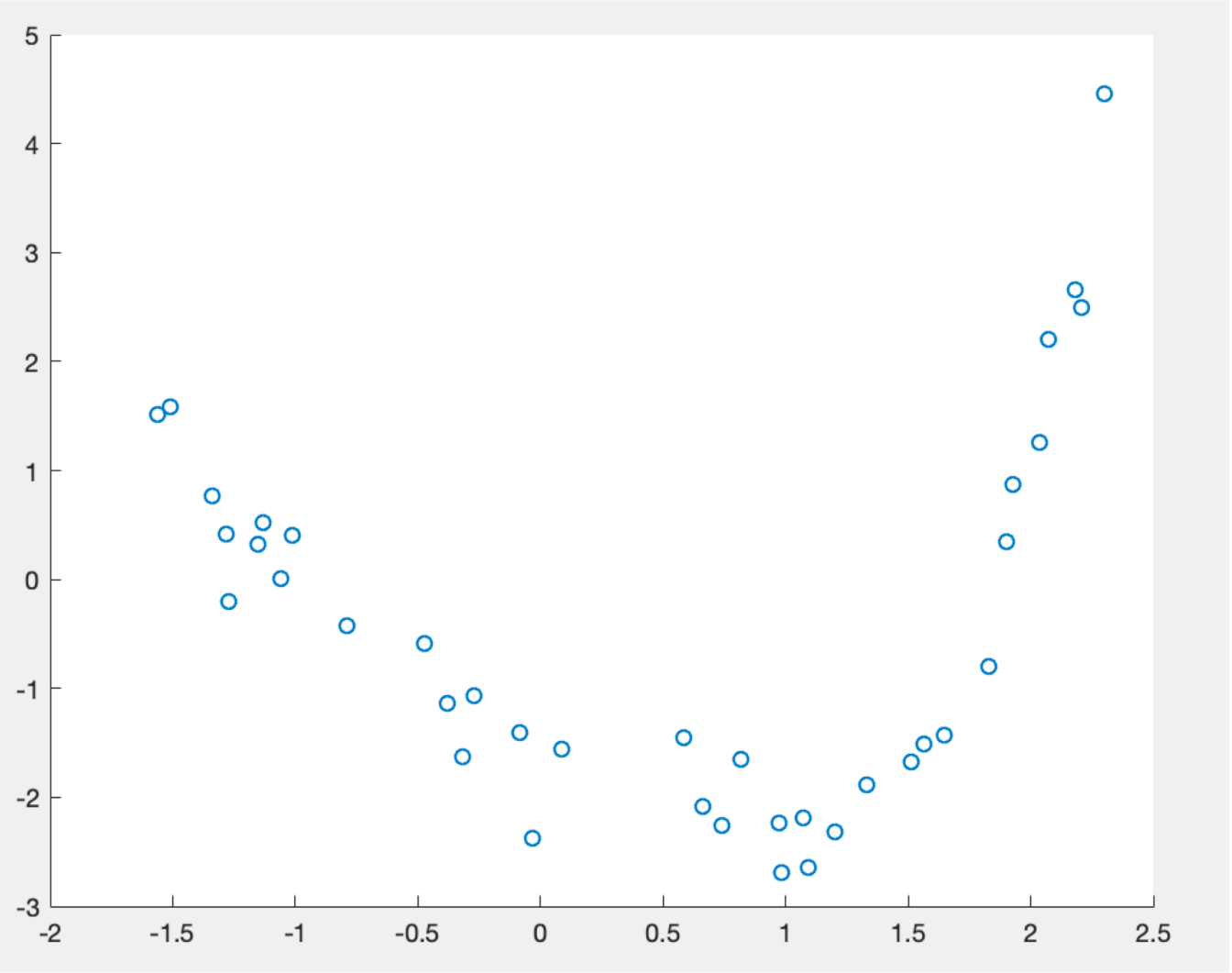
Why do we create models in the first place?

We create models to wish to describe some general underlying process

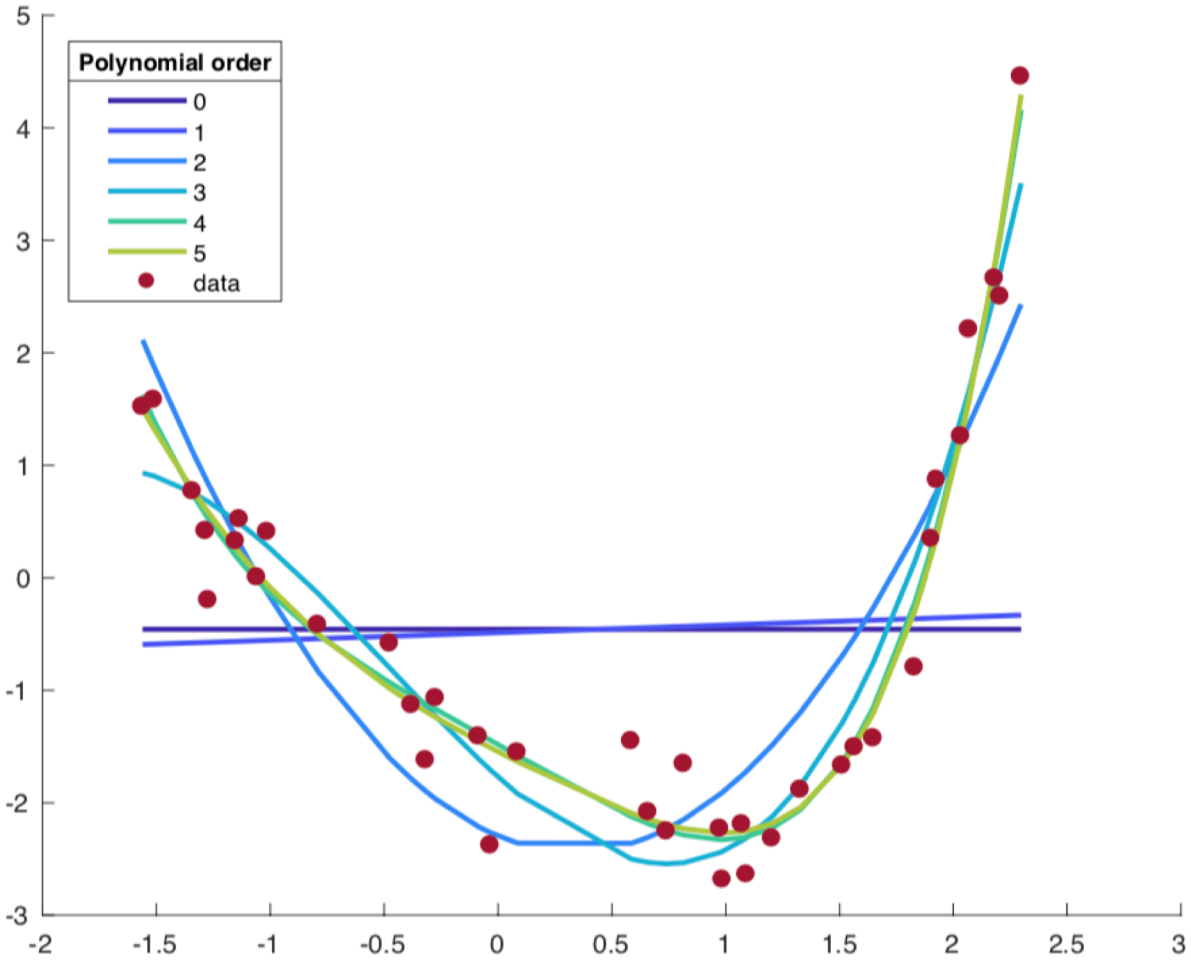
The models we fit should be able to predict **out-of-sample** data

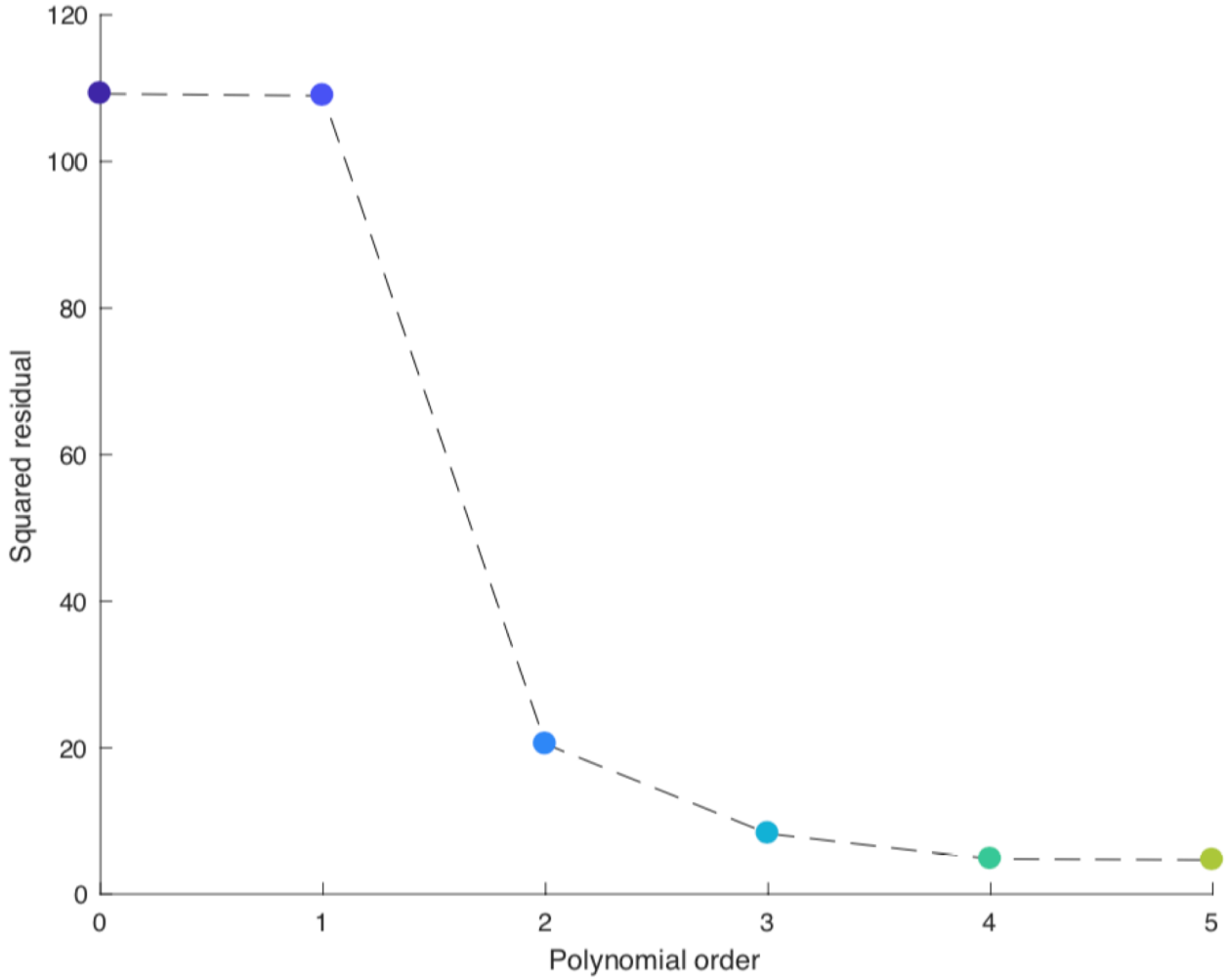
Our old friend from HW2

We tested 6 different models and assessed which was ‘best’.



Revisiting HW2





Training and testing sets

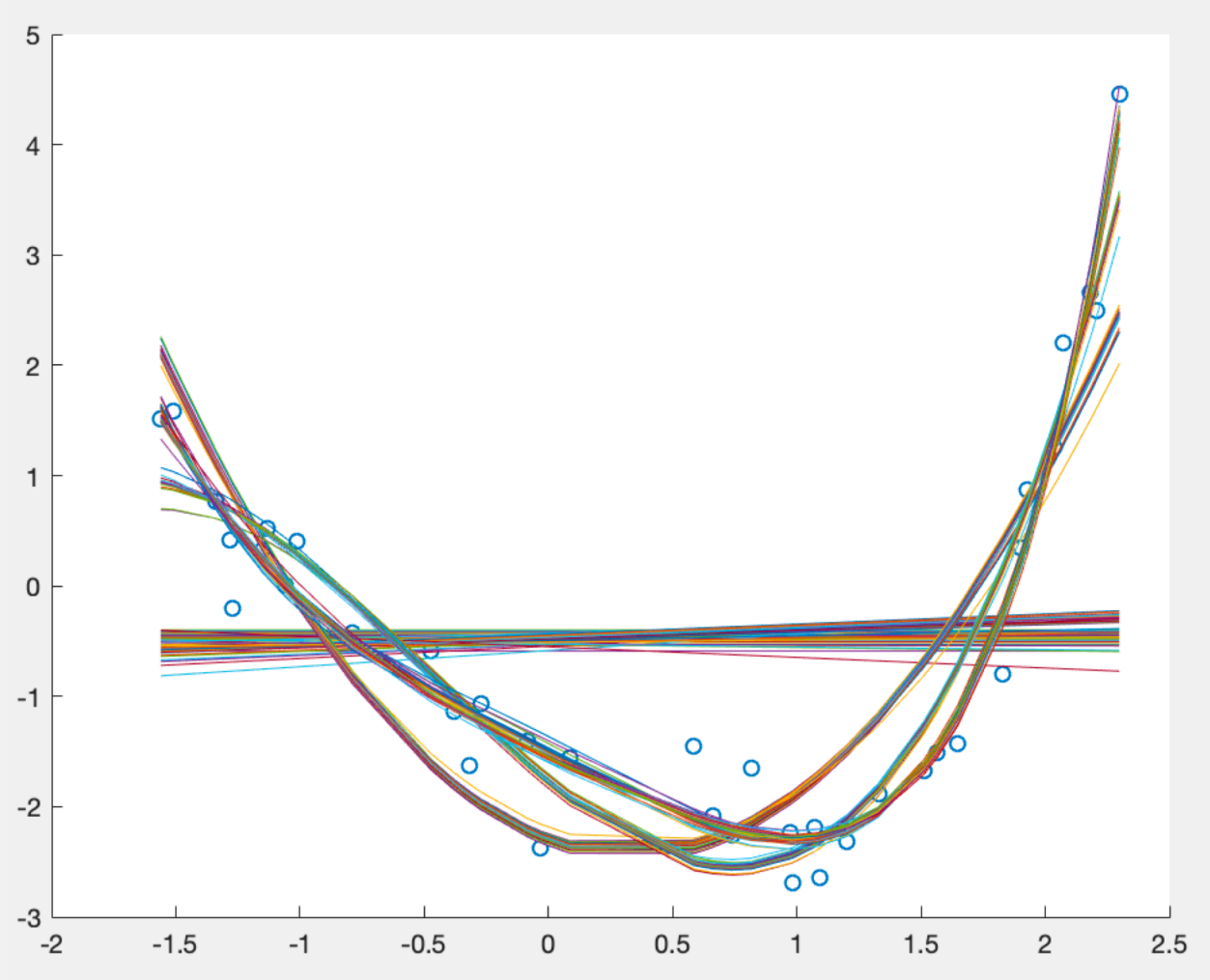
K-fold cross-validation
K=5

If you have N data points, what happens if K=N?

Different kinds of cross-validation

- K-fold where K is an integer that divides N
  - K=1?
  - K=N?

Leave-one-out cross validation (K=N)-fold



Take N-1 data samples
Fit a model
Do this N times

Each model will have a test-error

N=20 here

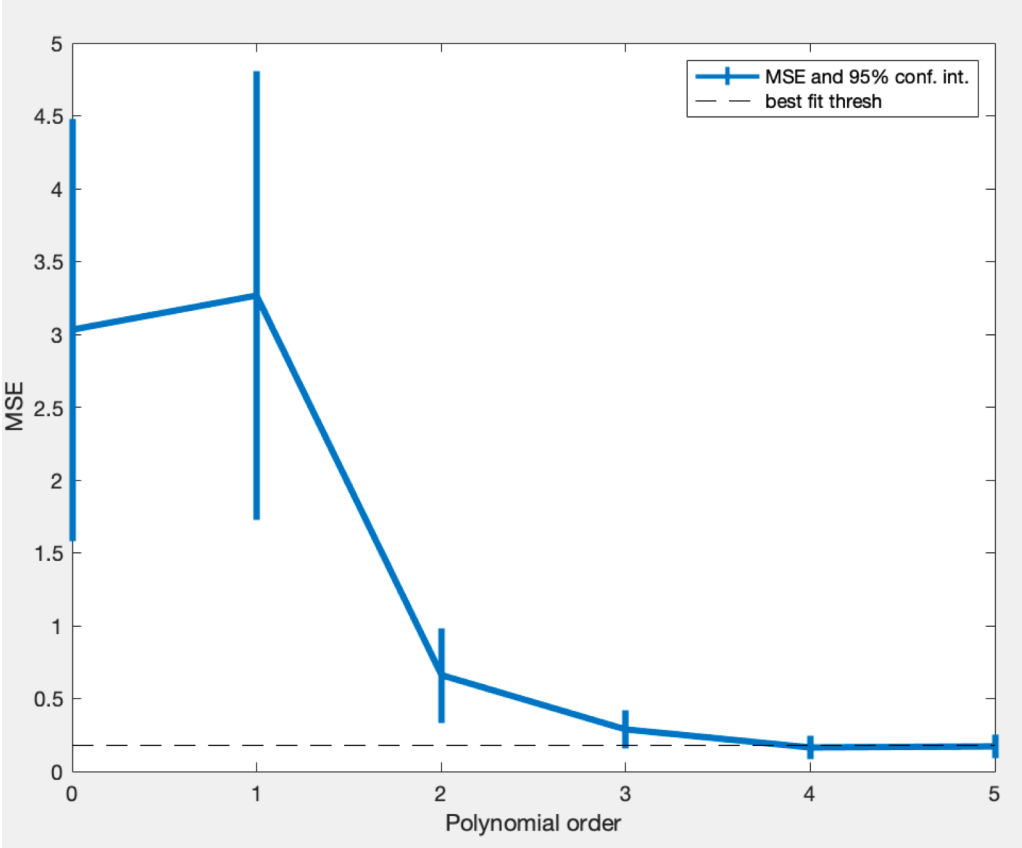
So we have 6 models * 20 cross-validations folds

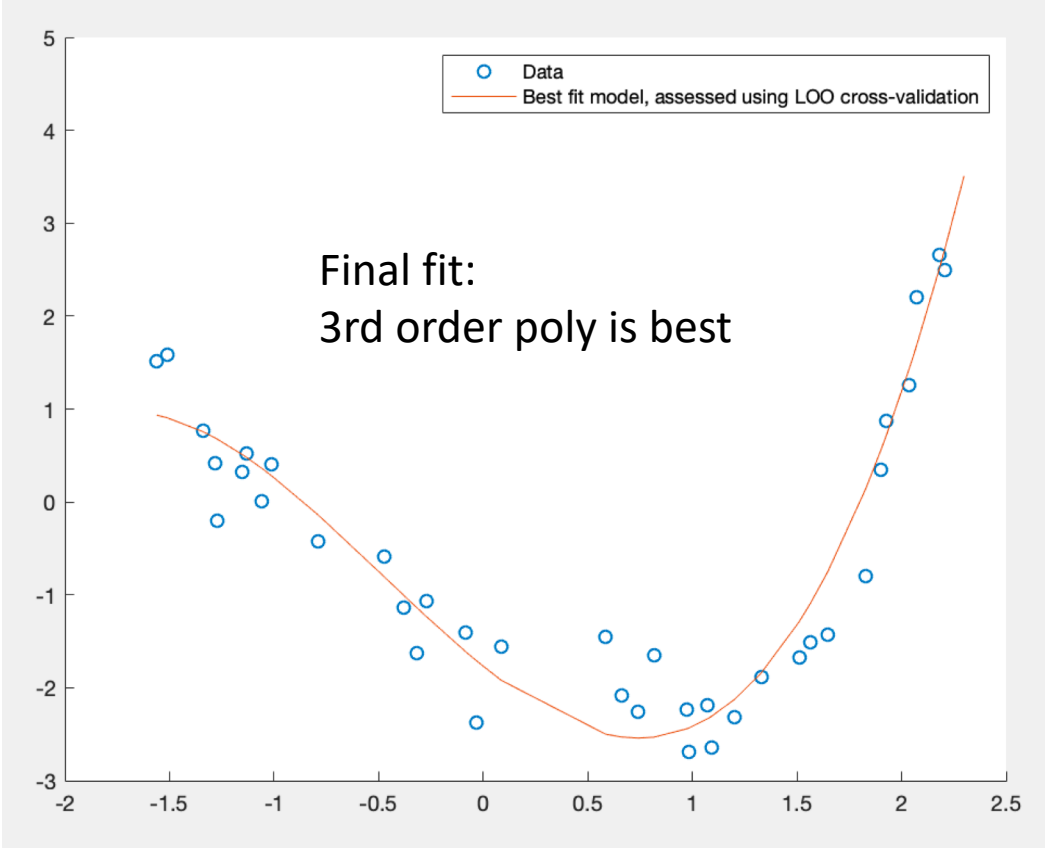
Each of the 6 models will have a mean+stdev error

Whichever has the “best” error is the model we should choose

How do we define “best”?

Leave-one-out cross validation (K=N)-fold





LOO cross-validated MSE and 95% confidence intervals
The minimum error is polynomial order 4
The simplest model that lies within poly4's error bars (black dashed line) is poly3