Statistical models in Neuroscience

Math Tools: Week 5

(Now over half-way through the lectures!)

Overview

1. Introduction to statistical models in neuroscience

2. Explore different models as a way to learn the techniques
    1. Estimating mean spike rate using a noise model
    2. Estimating spike rate as a function of a stimulus using least-squares
    3. Connection between least-squares and maximum likelihood
    4. Estimating spike rate as a nonlinear function of a stimulus assuming Gaussian and Poisson noise

3. Along the way: learn how to fit models using maximum likelihood, how to quantify model fit, how to compare 2 competing models…

4. A note: we are going to analytically solve a lot of problems, but real-world sometimes require numerical approaches. We'll touch on numerical approaches at the end, once we get a foundation built with simpler problems.

5. Another note: this lecture will be very systems-neuroscience specific. However, the general framework that we will be learning (fitting by maximizing likelihood) is used in many other fields

What is a statistical model?

From Rob Kass, 'Statistical Issues in the Analysis of Neuronal Data':

A parametric statistical model is a probability model with a fixed set of parameters that may be estimated from the data

Probability model: 'mathematical representation of a random event'

Example: Normalized distribution of spike counts over a trial

Types of parameters we will encounter: Parameters that describe the relationship between firing rate and a stimulus

As we will see, these models are very useful in providing insight to what's going on with experimental data

1. Help us form estimates of biologically relevant parameters
2. Provide a formalism for evaluating theoretical predictions (this goes for all models)
   - You can actually compute how likely your model is!

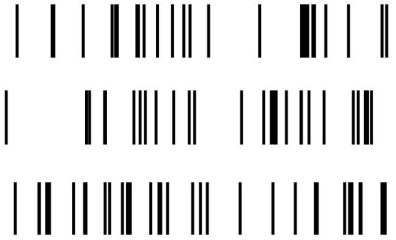
What is a statistical model and why is it important?

These models give us a handle on the mysterious ‘neural code’

If we know the stimulus, knowing the code means we can predict the spike train **ENCODING**

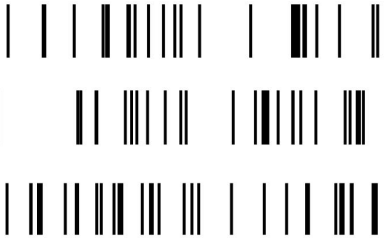
Our model!

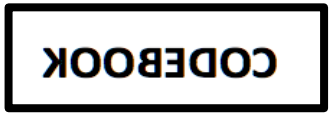
Neural response



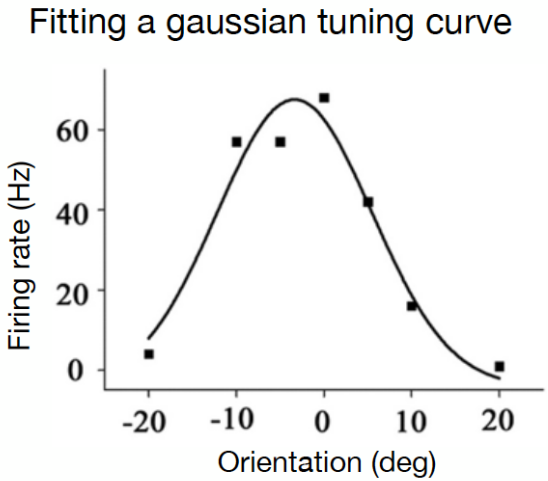
If we know the spike train, knowing the code means we can predict the stimulus **DECODING**

Neural response

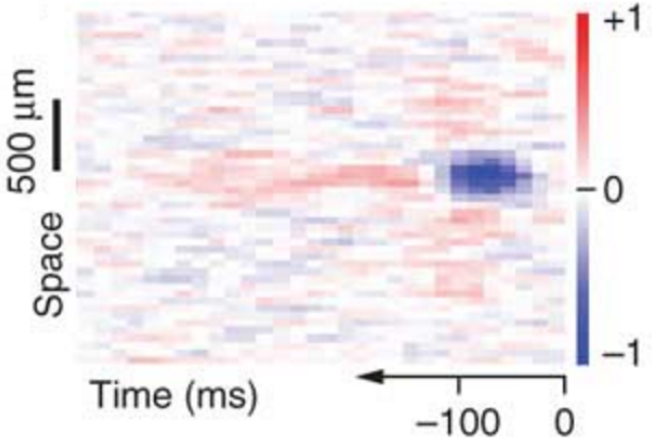




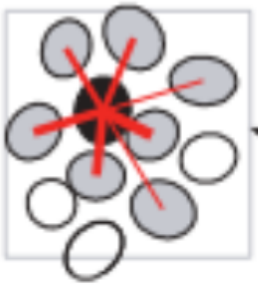
Some motivating examples



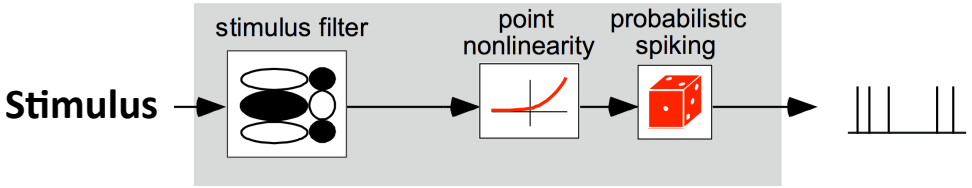
FR as a function of bar orientation
FR as a function of reach angle (monkey)



Find firing rate as a function of space and time



[Retinal ganglion] cell activity might be predicted by knowing the activity of it's neighbors



Spikes depend on the stimulus, the stimulus filter, a nonlinearity, and a stochastic process

Before we dive in… a brief flashback

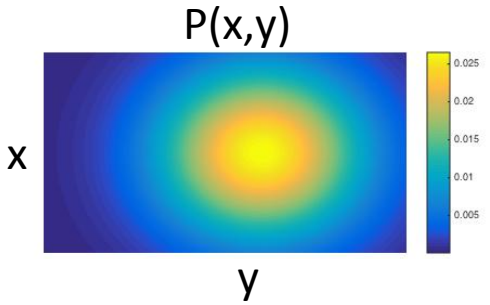
1. Probability

   P(x) → a probability distribution over values of x

   P(x,y) → joint probability distribution over x and y

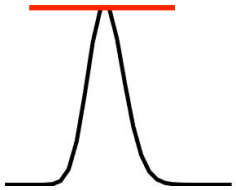
   P(x|y) → conditional probability over x values, given a specific y value

   - can be seen as a function of x (conditional probability) or as a function of y (likelihood function)



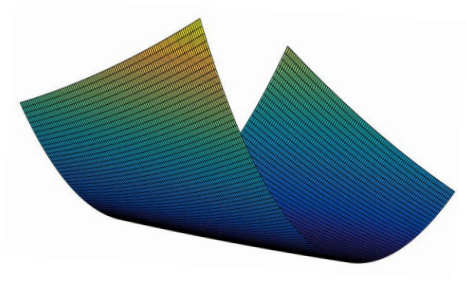
2. Finding the maximum and minimum of a function

   Take the derivative, and set it equal to zero



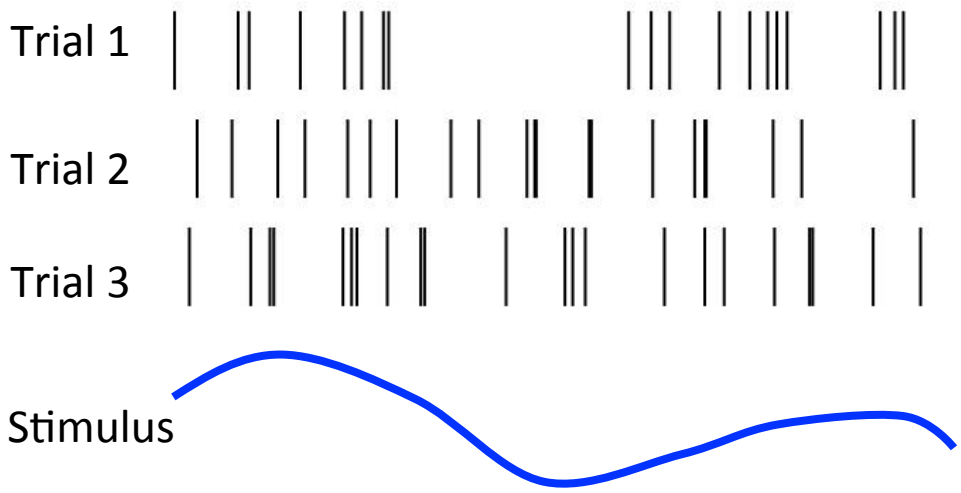
   If a function has multiple variables, we take the gradient: derivative with respect to each variable

   In this case, the gradient will be a 2-d vector (derivative in x- and y-direction)



Let's derive and fit our own statistical models!

A toy example:



Simple question: what is most likely mean firing rate during each trial?
→ Need a noise model

More complex question: what is the firing rate as a function of a stimulus?

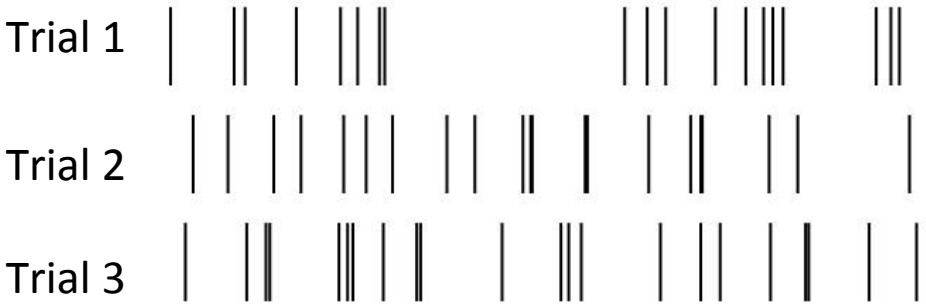
→ Need a noise + stimulus model

Game plan:

1. Examine the simple question first. We'll assume a probability model for spiking, and find the mean of this distribution. To 'find the mean', we will use maximum-likelihood.

2. Extend this framework to examine situations in which the firing rate varies with a stimulus, and the neuron stochastically fires spikes

Using maximum likelihood to find mean firing rate



Simple question: what is the most likely mean firing rate?

What you usually do: Compute average number of spikes per trial, divide by seconds/trial

$$\bar{y} = \frac{1}{n}\sum_i y_i \qquad FR = \frac{\bar{y}}{T}$$

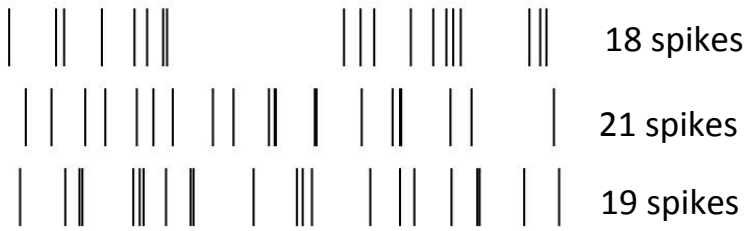
y_i = # of spikes for i^{th} trial
n = number of trials

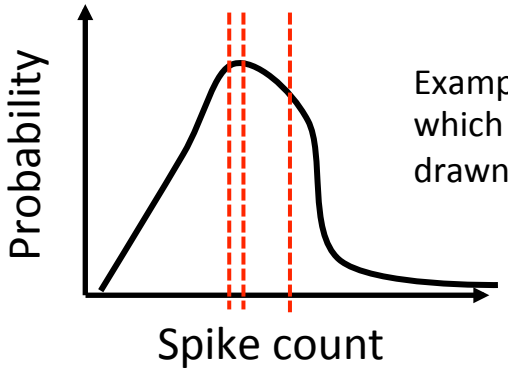
T = # of seconds per trial
(but let's just assume that all trials are the same length)

As an intro to statistical models, we will formulate a noise model for spikes and show that this answer matches our intuition

Use maximum likelihood to answer this question: Given the noisy spiking data, what is the *most likely* underlying firing rate? To answer this, we will compute the *maximum likelihood*

“Noise models”, aka probability distributions, in neuroscience





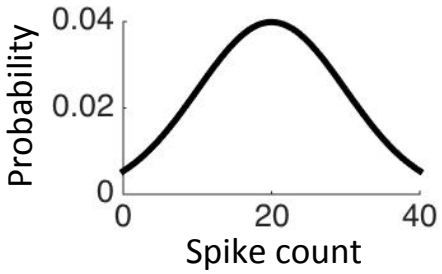
Example distribution from which spike counts are drawn

If we had an infinite number of trials, we could fill out this distribution, and find the mean value

But since we will never be in this situation, we usually guess the general distribution, and use model-fitting to find the parameters of the distribution

One example distribution:

Normal distribution!

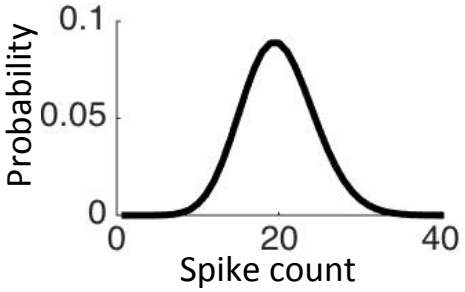


Parameterized by two numbers: μ and σ

$$P(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(\mu-y)^2}{2\sigma^2}}$$

Another example distribution:

Poisson distribution!

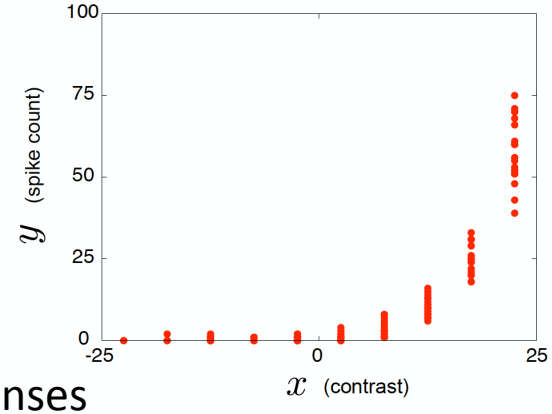


Parameterized by one number: μ

$$\mathrm{P}(y|\mu) = \frac{\mu^y e^{-\mu}}{y!}$$

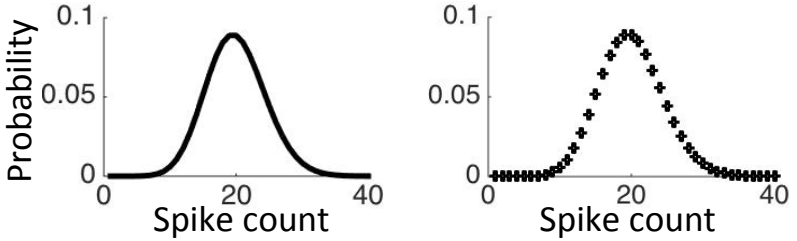
A few nice facts about the Poisson distribution

$$\mathrm{P}(y|\mu) = \frac{\mu^y e^{-\mu}}{y!}$$



Nice fact #1:
Mean of distribution = μ
Variance of distribution = μ
Variance grows with mean – this is commonly seen in neural responses

Nice fact #2:
Distribution is actually only defined for non-negative integers
Now, it's built in that -1 or 1.5 spikes don't make sense

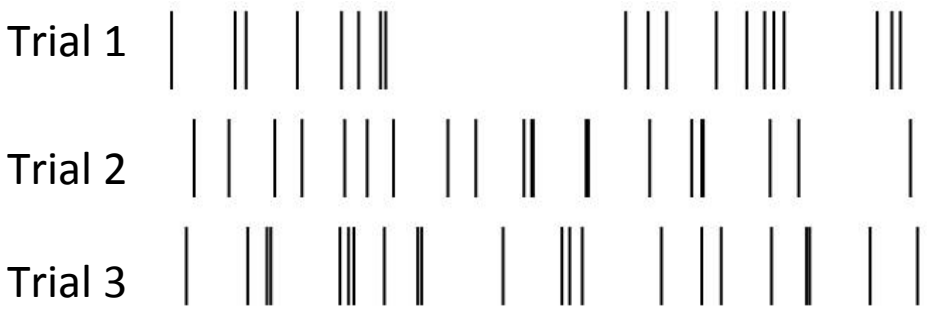


Nice fact #3:
Maps onto a noisy spike train super nicely

Poisson dist. pops up when describing the distribution of the total number of (unlikely) successes that occur from many independent trials

→ For spike trains, there is a small chance that a spike occurs within a small time bin. But string a lot of 'small time bins' together, and (assuming non-zero firing rate) you get Poisson spiking!

Given the data, what's the most likely firing rate?



Assume a Poisson distribution.
Probability of getting y_1 spikes in 1 trial:

$$P(y_1|\mu) = \frac{\mu^{y_1} e^{-\mu}}{y_1!}$$

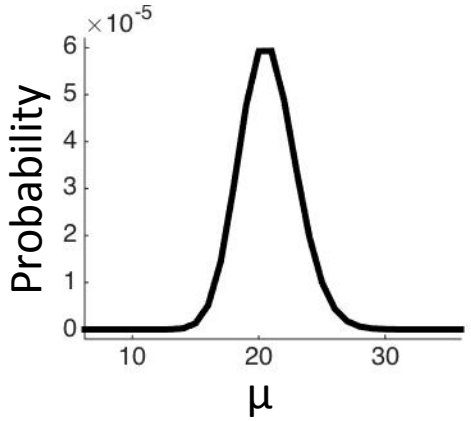
The probability of observing all the data, given μ, is the *product* of individual trial probabilities

$$\mathcal{L}(\mu) = \ P(y|\mu) = P(y_1|\mu)P(y_2|\mu)P(y_2|\mu) = \prod_{i=1}^{3} \frac{\mu^{y_i} e^{-\mu}}{y_i!}$$

Take maximum likelihood approach: for what value of μ is the probability the highest?

One option: brute-force numerical computation – sweep over values of μ, compute probability, find the μ associated with highest probability

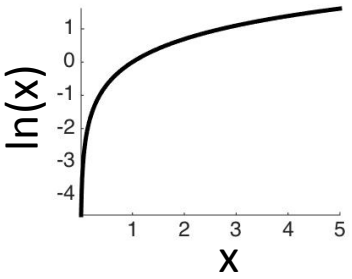
This value of μ is called the maximum likelihood estimate (MLE)



Rob Kass, Statistical Issues in the Analysis of Neuronal Data

Given the data, what's the most likely firing rate?

The analytical method: maximize $\mathcal{L}(\mu) = \mathrm{P}(\mathbf{y}|\mu) = \prod_i^n \frac{\mu^{y_i} e^{-\mu}}{y_i!}$



Maximize log-likelihood

$$\log(\mathcal{L}(\mu)) = \log(\prod_i^n \frac{\mu^{y_i} e^{-\mu}}{y_i!})$$

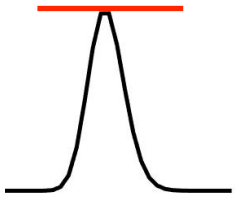
$$\log(\mathcal{L}(\mu)) = \sum_i^n y_i \log(\mu) - \mu - \log(y_i!)$$

Take derivative with respect to μ and set this equal to 0:

$$\frac{\mathrm{d}}{\mathrm{d}\mu}(\log(\mathcal{L}(\mu))) = \sum_i^n \frac{y_i}{\mu} - 1 = 0$$

$$-n + \sum_{n^i}^n \frac{y_i}{\mu} = 0$$

$$\frac{1}{n} \sum_i^{n^i} y_i = \mu$$

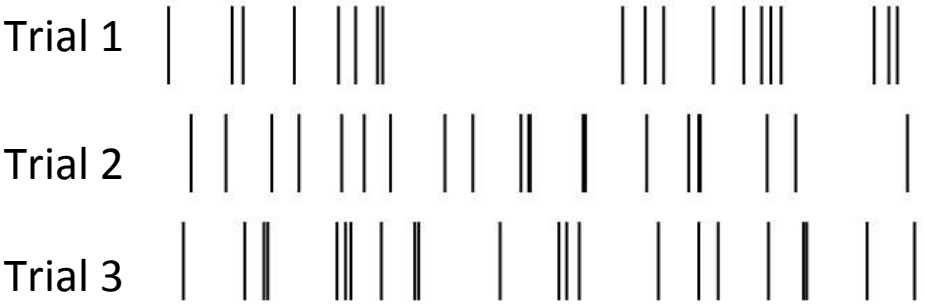


Assuming a Poisson noise model, the maximum likelihood estimate is equal to our intuitive guess from earlier!

Note on ML: this is not the only method, but is a great one: unbiased and efficient

Summary: using ML to find mean firing rate

Our data:



The question we asked:
Assuming the firing rate is constant within and across trials, and that the spike counts across trials follow a Poisson distribution, what is the mean firing rate?

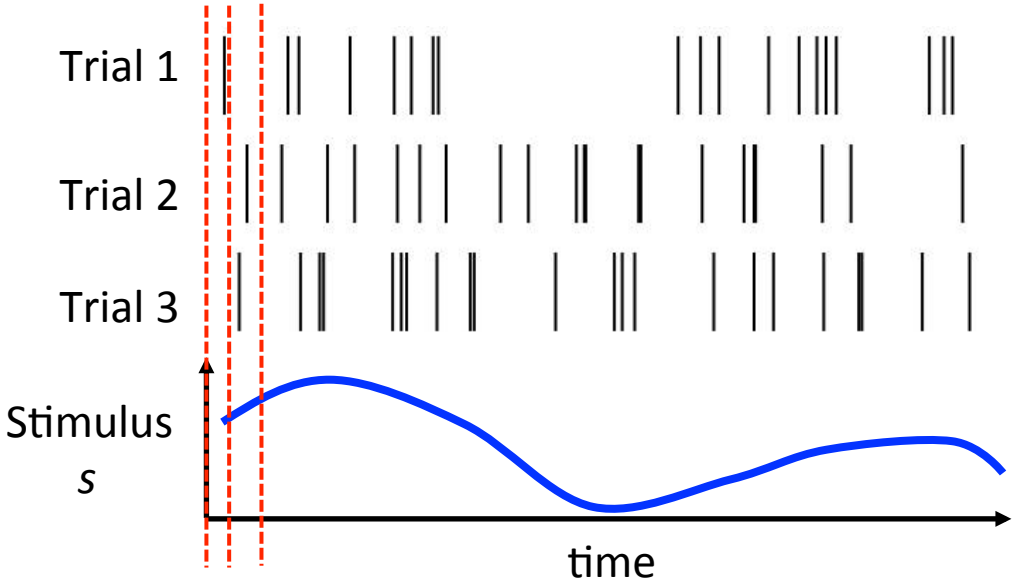
A related (and more general) question: What is the parameter (in this case: μ) that returns a probability distribution that best matches what we see in our data?

To answer this question, we maximized the probability of observing our data given the parameter μ

$$\mathcal{L}(\mu) = P(y|\mu) = P(y_1|\mu)P(y_2|\mu)P(y_2|\mu) = \prod_{i=1}^{3} \frac{\mu^{y_i} e^{-\mu}}{y_i!}$$

We found an analytical expression for μ: $\frac{1}{n}\sum_{i}^{n} y_i = \mu$

Let's step this up a notch…



In previous example, we assumed that the firing rate was constant…. What if firing rate varies?

The firing rate could vary with a stimulus.

New goal: find a model that captures the relationship between the stimulus and spike rate

To do this:

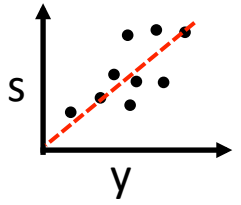
1. Bin across time to get a vector of spike numbers (**y**) and stimulus values across time (**s**)
   1. If < 1 spike/bin, then this vector represents the probability of observing a spike
2. Assume that time bins are conditionally independent, given the stimulus value

$$P(\mathbf{y}|\mathbf{s}) = P(y_1|s_1)P(y_2|s_2)\ldots P(y_m|s_m)$$

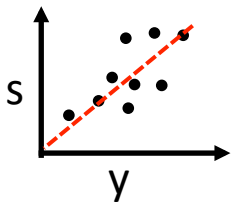
3. Formulate a model for the relationship between stimulus and spiking
   1. First: linear regression (forget about Poisson spiking for a second)

$$y_i = ks_i + b + \epsilon_i$$

Note – index *i* is for a time bin now, NOT a trial. We'll combine trials.



Least squares solution



$$y_1 = ks_1 + b + \epsilon_1$$
$$y_2 = ks_2 + b + \epsilon_2$$
$$\vdots$$
$$y_m = ks_m + b + \epsilon_m$$

ε is a Gaussian noise term: mean 0, var = σ^2

We want to minimize ε, so that our estimate of the spike number is close to the actual spike number, *for every time bin*

The knobs we can turn are *k* and *b*

Note: we have 2 unknowns and *m* > 2 equations, so this is an overdetermined system. Probably no *k* and *b* that satisfy all equations… choose best *k* and *b*

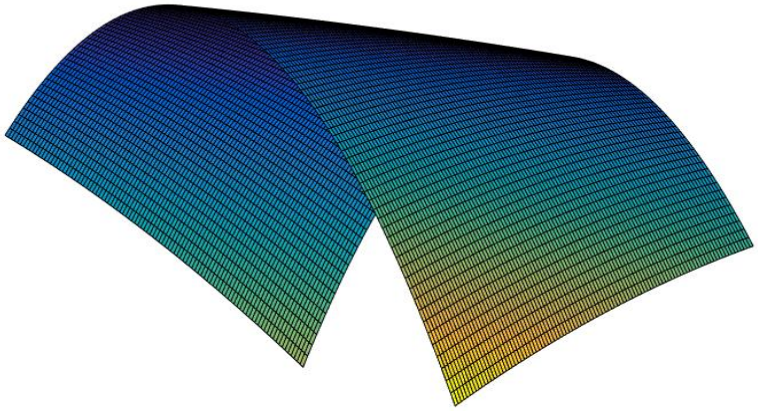
In mathematical terms:

For one time bin:
Minimize $|ks_i + b - y_i|$

This is an easier and equivalent thing to min:
Minimize $(ks_i + b - y_i)^2$

For all time bins:
Minimize $\sum_{i=1}^{m}(ks_i + b - y_i)^2$

One way we can solve this: sweep over all possible b and k values, compute this sum, and then take the k and b values to return the smallest value. But there's a smarter way!



Least squares solution

Goal: minimize $\sum_{i=1}^{m}(ks_i + b - y_i)^2$

Let's re-formalize in matrix-vector notation:

$$\begin{aligned} y_1 &= ks_1 + b + \epsilon_1 \\ y_2 &= ks_2 + b + \epsilon_2 \\ &\vdots \\ y_m &= ks_m + b + \epsilon_m \end{aligned} \qquad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = k \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{bmatrix} + b \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} s_1 & 1 \\ s_2 & 1 \\ \vdots & \vdots \\ s_m & 1 \end{bmatrix} \begin{bmatrix} k \\ b \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{bmatrix} \qquad y = \mathbf{S}\theta + \epsilon$$

Now, the minimization problem:

Minimize: $||\epsilon||^2$ → Minimize: $||\mathbf{S}\theta - y||^2$ Recall this is just a vector!

Expand: $||\mathbf{S}\theta - y||^2 = (\mathbf{S}\theta - y)^T(\mathbf{S}\theta - y)$

Least squares solution

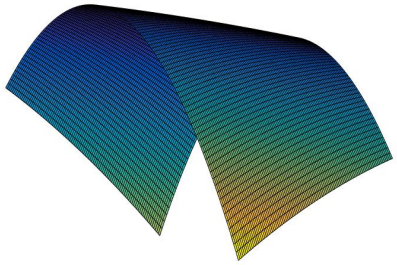
Expand:

$$= (\mathbf{S}\theta - y)^T(\mathbf{S}\theta - y)$$

$$= \theta^T\mathbf{S}^T\mathbf{S}\theta - 2y^T\mathbf{S}\theta + y^Ty$$

Take gradient with respect to parameters, and set this equal to 0:

$$2\mathbf{S}^T\mathbf{S}\theta - 2\mathbf{S}^Ty = 0$$



Gradient will be of form: $\begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}k} \\ \frac{\mathrm{d}}{\mathrm{d}b} \end{bmatrix}$

Normal equations:

$$\mathbf{S}^T\mathbf{S}\theta = \mathbf{S}^Ty$$

$$\theta = (\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^Ty$$

This is great! We can find the right values of *k* and *b* by just computing the expression on the right hand side of the equation!

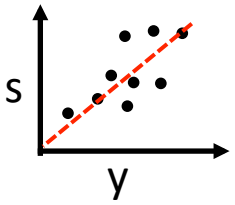
It's not always this easy to find the minimum of a function. Many times, you have to use numerical methods – more on this later…

In MATLAB: for this problem, you can also use `theta = polyfit(y,s)`

Relationship between least-squares and ML

Use this model: $y_i = ks_i + b + \epsilon_i$ ε is a Gaussian noise term: mean 0, var = σ^2

Original goal: minimize $\sum_{i=1}^{m}(ks_i + b - y_i)^2$



We could have come at this a different way, like we did for the mean spike rate problem, by maximizing a log-likelihood function

With the way we've said it so far, we would use a *Gaussian* noise model, not a Poisson

Gaussian distribution: $P(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(\mu-y)^2}{2\sigma^2}}$

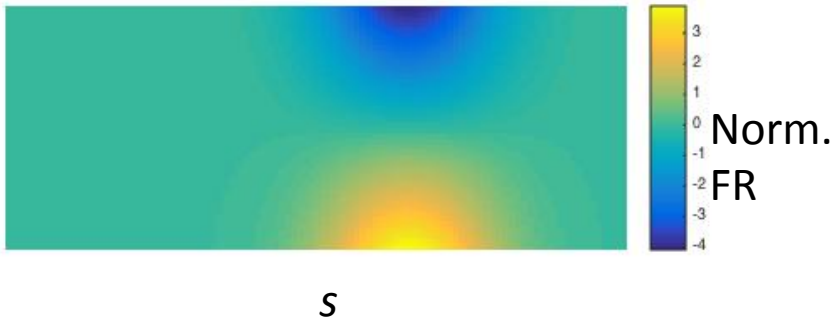
Probability distribution with our model: $P(y|s,k,b,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \prod_{i=1}^{m} e^{\frac{-(ks_i+b-y_i)^2}{2\sigma^2}}$

If we take the log of this expression: $\log(P(y|s,k,b,\sigma^2)) = \sum_{i=1}^{m} \log(\frac{1}{\sqrt{2\pi\sigma^2}}) - \frac{(ks_i+b-y_i)^2}{2\sigma^2}$

The least squares solution is the same as the maximum likelihood solution! $\log(P(y|s,k,b,\sigma^2)) = \sum_{i=1}^{m} a - c(ks_i+b-y_i)^2$

Linear model with two stimuli

What if instead, there were multiple stimuli: *s* and *r*



$$y_1 = ks_1 + lr_1 + b + \epsilon_1$$
$$\vdots$$
$$y_m = ks_m + lr_m + b + \epsilon_m$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = k \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} + l \begin{bmatrix} r_1 \\ \vdots \\ r_m \end{bmatrix} + b \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_m \end{bmatrix}$$

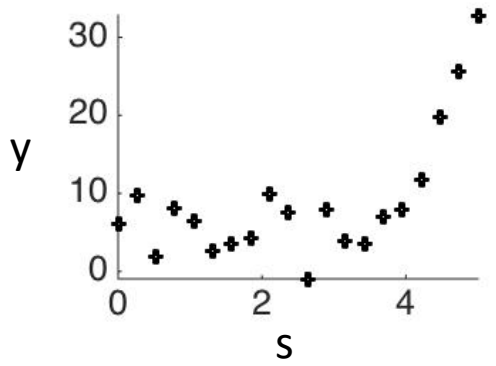
$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} s_1 & r_1 & 1 \\ \vdots & \vdots & \vdots \\ s_m & r_m & 1 \end{bmatrix} \begin{bmatrix} k \\ l \\ b \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_m \end{bmatrix}$$

$$y = \mathbf{S}\theta + \epsilon$$

This is great! Same equation as before applies

Linear model, as a nonlinear function of stimuli

Maybe the stimulus-response relationship was something more complicated…



Definitely not a linear function of s

Fit a function that is nonlinear in the stimulus, but linear in the coefficients

$$\hat{y}_i = a_2 s_i^2 + a_1 s_i + a_0$$ We could fit a 2-degree polynomial

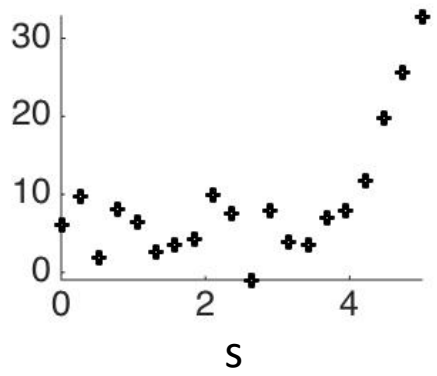
$$\hat{y}_i = a_3 s_i^3 + a_2 s_i^2 + a_1 s_i + a_0$$ A 3-degree polynomial

$$\hat{y}_i = a_k s_i^k + \ldots + a_2 s_i^2 + a_1 s_i + a_0$$ An m-degree polynomial

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} s_1^k & \ldots & s_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ s_m^k & \ldots & s_m & 1 \end{bmatrix} \begin{bmatrix} a_k \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}$$

Still, it's the same formulation

Model comparison



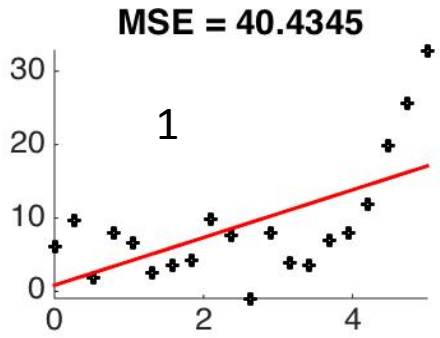
$$y_i = a_2 s_i^2 + a_1 s_i + a_0$$

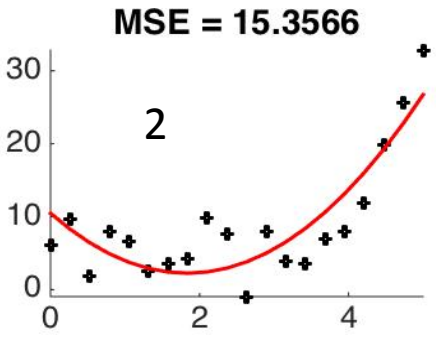
$$y_i = a_3 s_i^3 + a_2 s_i^2 + a_1 s_i + a_0$$

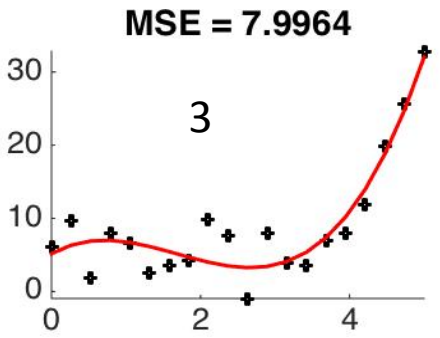
$$y_i = a_4 s_i^4 + a_3 s_i^3 + a_2 s_i^2 + a_1 s_i + a_0$$

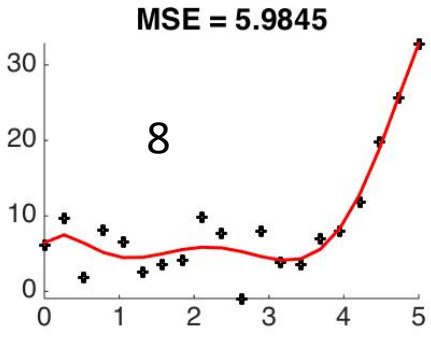
All possible models

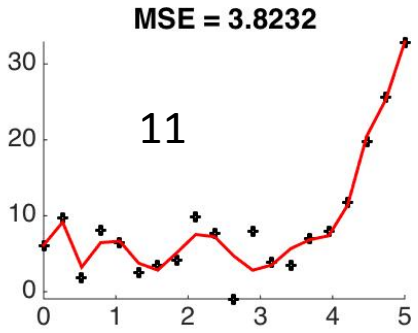
Which one is the best?

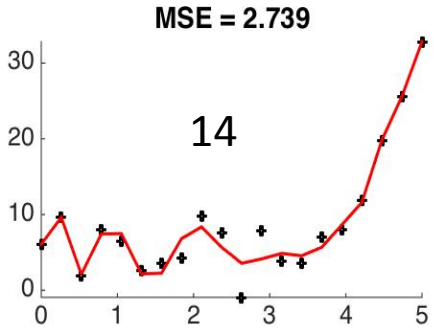






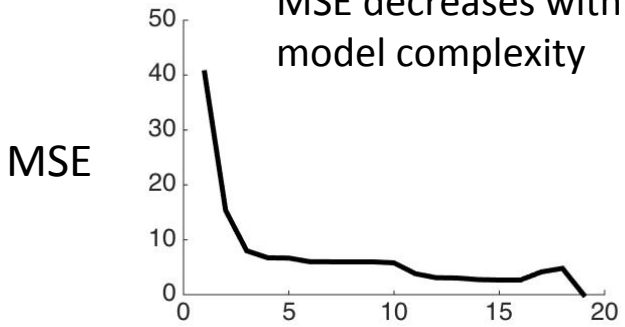


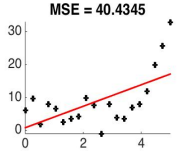




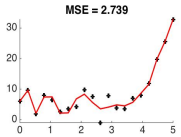
Over-fitting

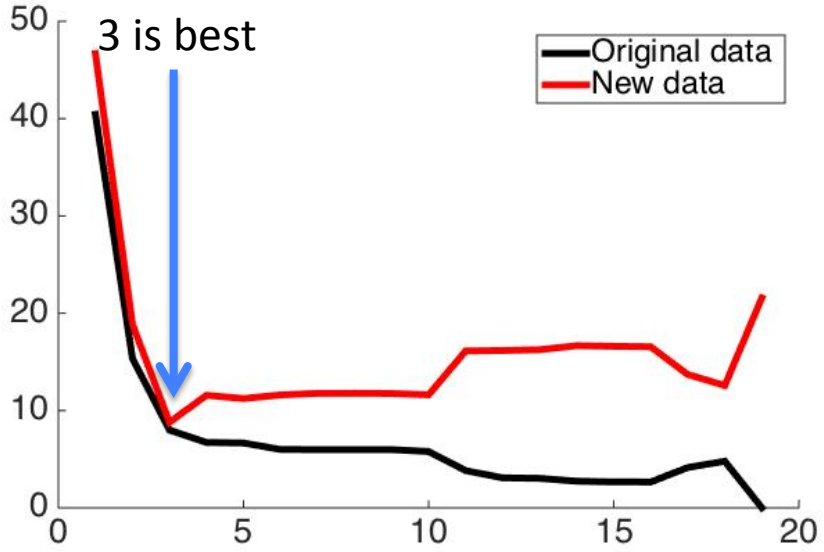
MSE decreases with model complexity



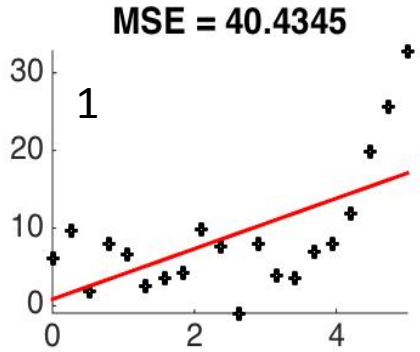


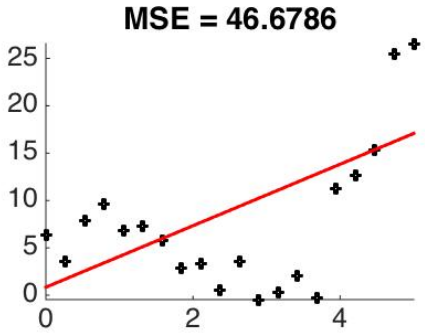
Degree of polynomial

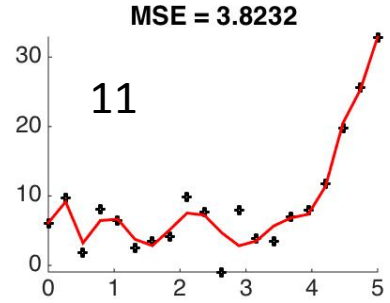


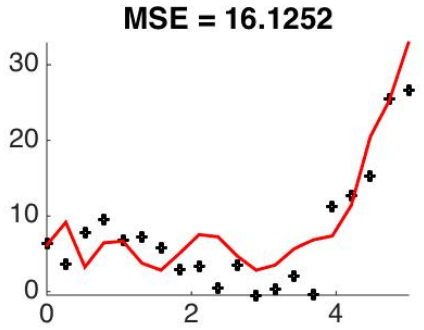


But what happens if I test this model on a different set of data?









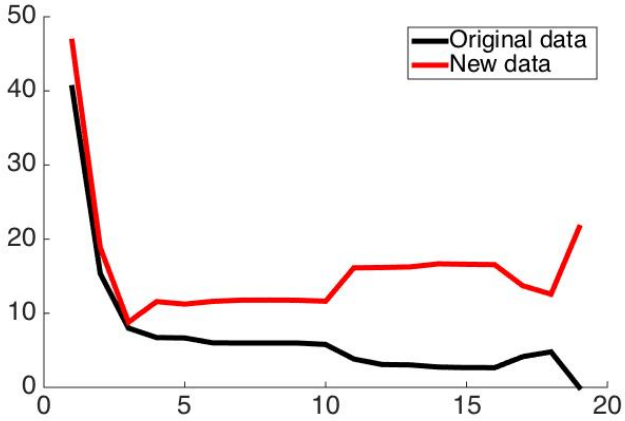
The complex model really suffers (MSE quadruples) compared to simpler model – it overfits

Simpler model generalizes better, but MSE of complex model is still better…

Which model 'generalizes' the best? We want to say things that are true for all experiments, not just ours

Cross-validation

Model-evaluation technique in which you choose best model based on performance on held-out data



The way you do it without collecting extra data:

1. Divide data into a 'training set' and a 'testing set'
   Usually 80-90% / 20-10%
2. Fit model(s) using the training data set
3. Use the parameters from step 2 to find the expected fit for testing data
4. Record measure of model fit on testing data

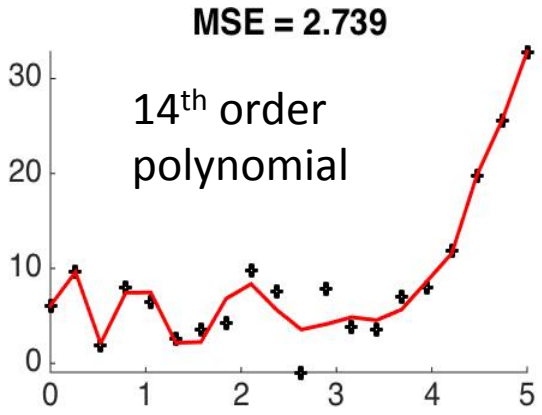
$$y_1 = \mathbf{S}_1\theta$$
$$y_2 = \mathbf{S}_2\theta$$

Sometimes (often), you want to divide your data into 5-10 pairs of training sets and testing sets [Specific example: choose 10% to set aside, fit on remaining 90%. Then, choose another 10% to set aside, fit on remaining 90%. For this example, you repeat this procedure 10 times.]

A note on measures of model fit:
Here, I used MSE, in continuing with the least-squares theme. BUT THIS IS NOT THE ONLY WAY

- Log-likelihood $\log(P(y|s,k,b,\sigma^2)) = \frac{1}{\sqrt{2\pi\sigma^2}}\sum_{i=1}^{m} -(ks_i + b - y_i)^2 + \log(2\sigma^2)$
- Correlation
- Variance explained – incorporate squared error from a 'mean model'

Over-fitting and data/parameter ratio



This model is complex:
Underlying structure maybe complex
Difficult to tell when considering the small number of data points…

20 data points and 15 parameters - not enough data!

When fitting models, # data points should be *at least* double the # of parameters
Something like an order of magnitude is probably more realistic
Depends on firing rate, noise in data

What do you do if you aren't sure you have enough data and you can't get more?

Include assumptions about data
Avoid spuriously high parameter values (in this case, coefficients of polynomial)

Original goal: minimize $\sum_{i=1}^{m}(ks_i + b - y_i)^2$

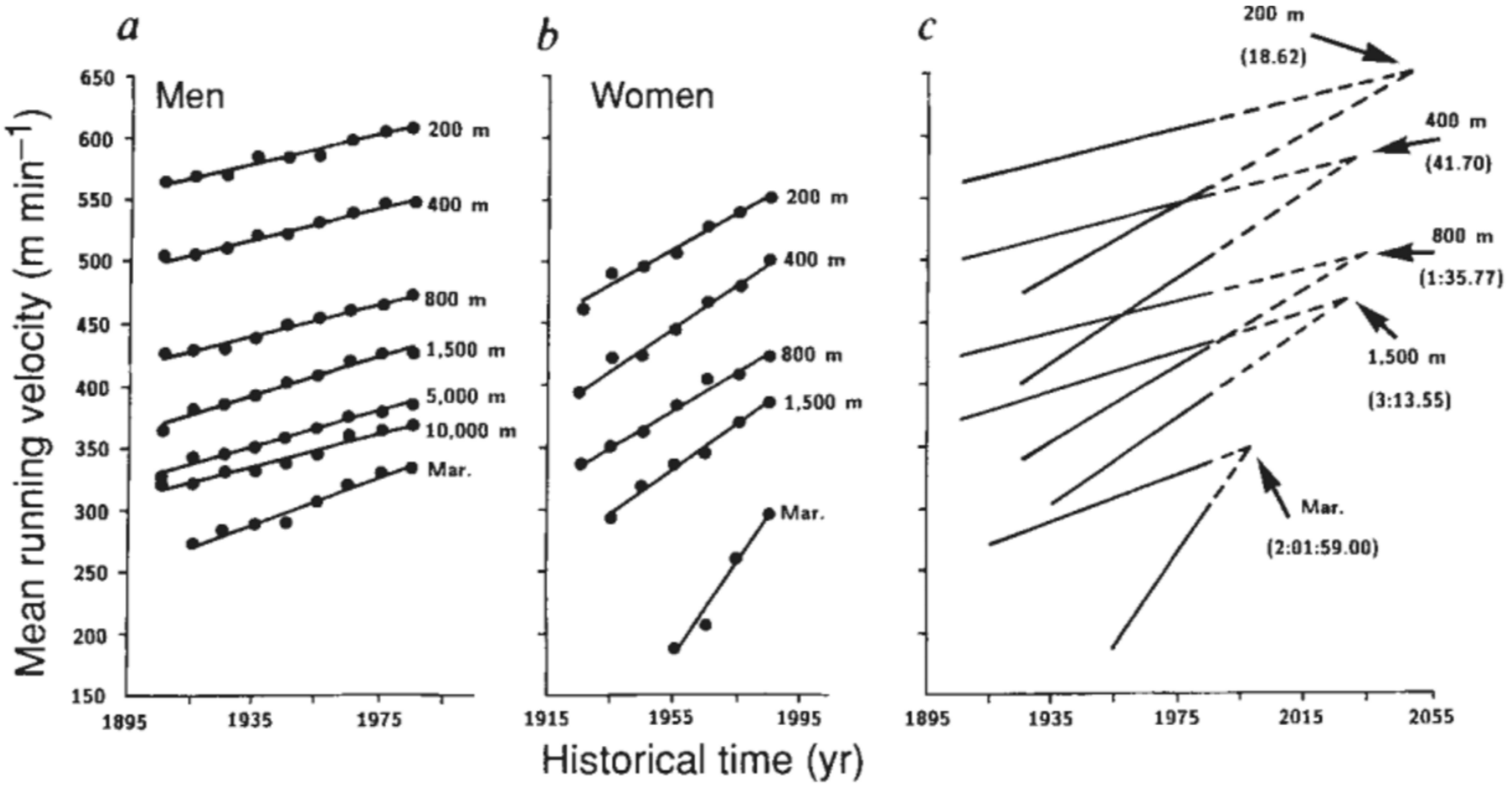
$+|k| + |b|$

$+\alpha|k| + \beta|b|$

$+\alpha k^2 + \beta b^2$

[Bayes' rule interpretation of regularizers]

Be careful with model interpretation…

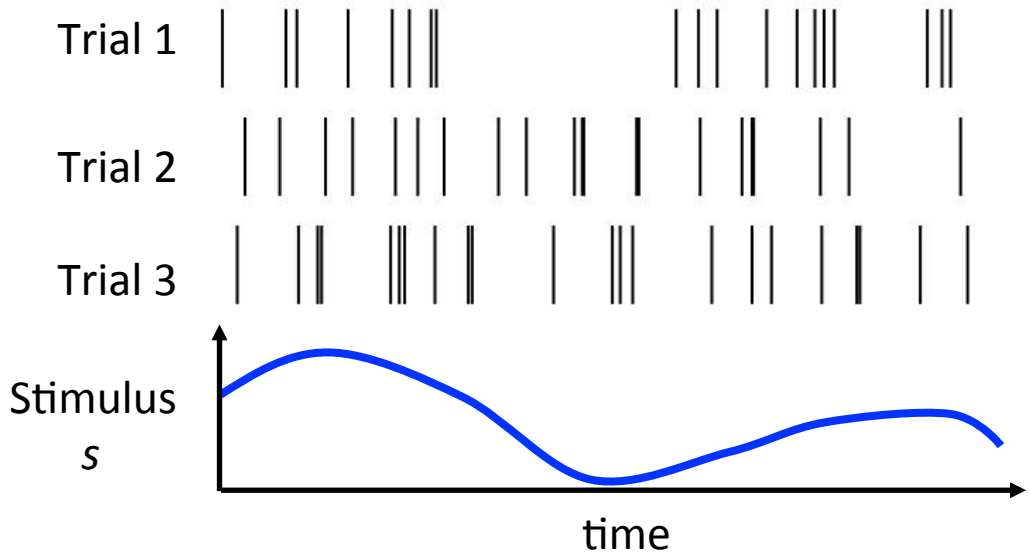


In 2095, women marathoners will overtake women 200m runners
In 6419, women marathoners will reach the speed of sound

Whipp et al 1992, Nature

Linear-Poisson model

The 2nd-to-last model we will consider:



Last model, we used least squares to find parameters to a linear model

This was equivalent to finding the maximum likelihood with a Gaussian noise model:

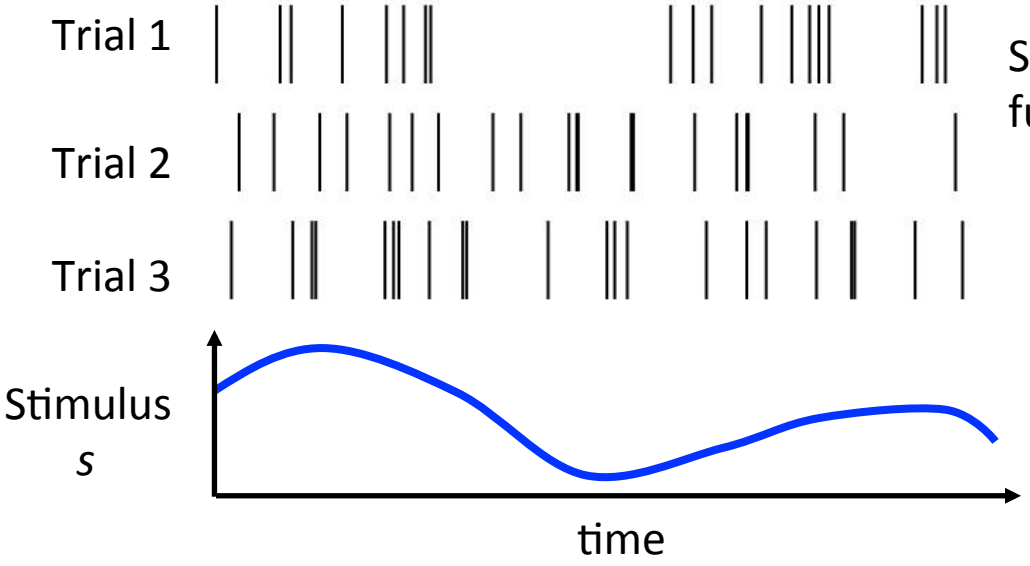
$$P(y|s,k,b,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \prod_{i=1}^{m} e^{\frac{-(ks_i+b-y_i)^2}{2\sigma^2}}$$

(At least) two things are that great about this model:

1. Spiking tends to be Poisson, not Gaussian
    1. Variance increases with the mean
2. There's nothing stopping firing rate values from going negative

Let's fix these things by using a Poisson noise distribution!

Linear-Poisson model



Spike rate is a linear function of stimulus:

$$\hat{y}_i = \theta s_i$$

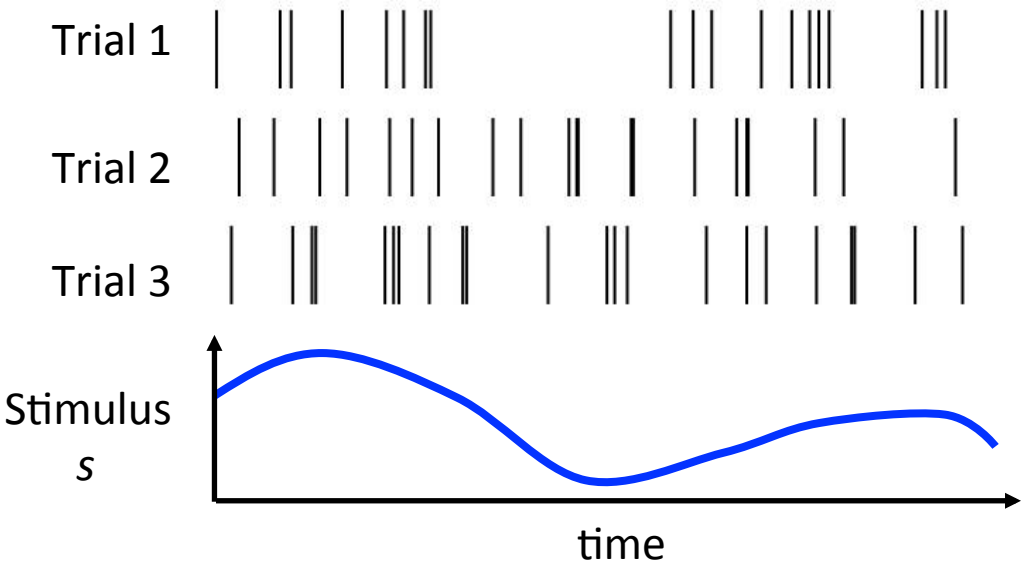
(dropped intercept term – transform firing rate for this to be true)

Then, with a Poisson model:

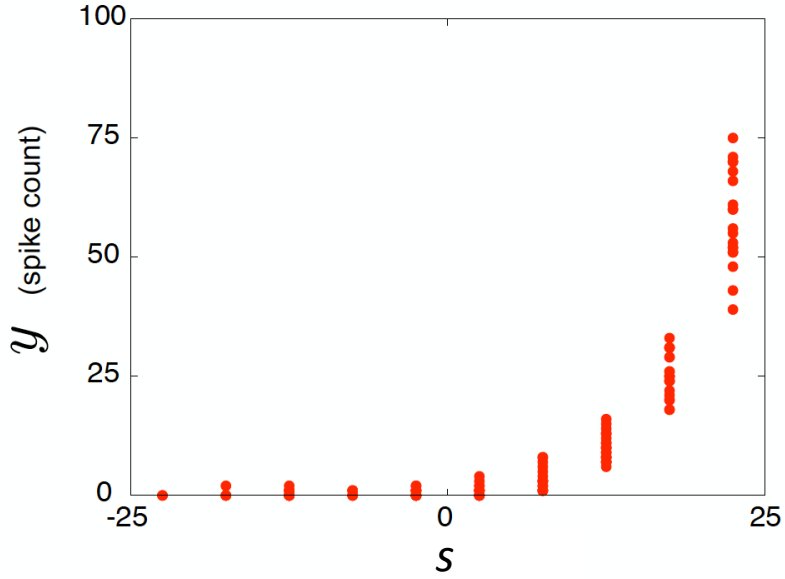
$$P(y_i|s_i,\theta) = \frac{(\theta s_i)^{y_i} e^{-\theta s_i}}{y_i!}$$

Homework: analytically find the parameter value that maximizes log-likelihood
(Steps: take log, take derivative with respect to theta, set this equal to 0, then solve!)

Linear-Nonlinear-Poisson model



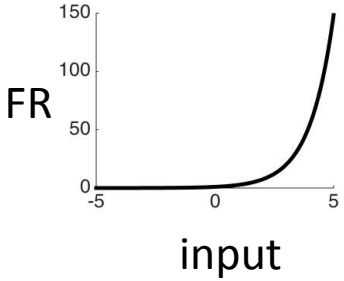
Plotting the stimulus value against spike counts:



Mean spike rate is a nonlinear function of stimulus: $\hat{y}_i = f(\theta s_i)$

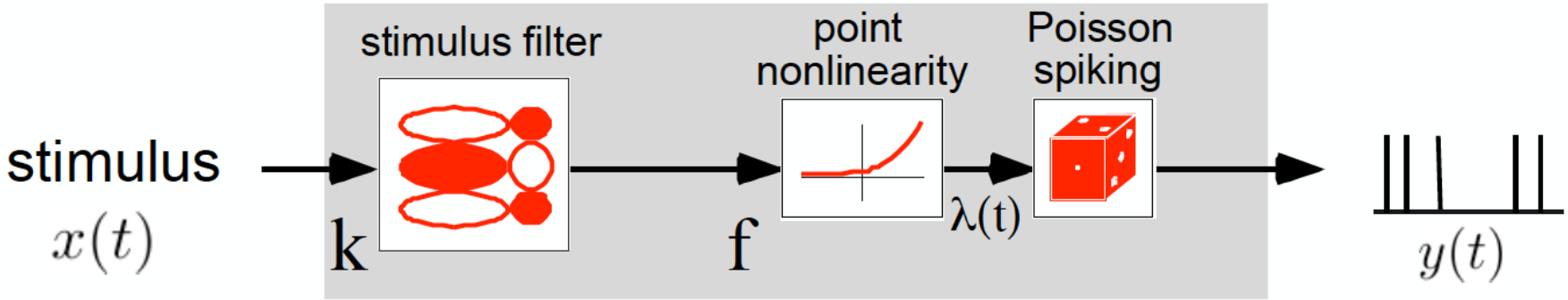
One common nonlinearity: $\hat{y}_i = e^{\theta s_i}$

"Soft-threshold"



This nonlinearity is nice because it actually guarantees that the best parameter is easily found

Interpretation of Linear-Nonlinear-Poisson model



$$\hat{y}_i = e^{\theta s_i} \qquad P(y|\mu) = \prod_i \frac{\mu^{y_i} e^{-\mu}}{y_i!} \quad \rightarrow \quad P(y|e^{\theta s_i}) = \prod_i \frac{(e^{\theta s_i})^{y_i} e^{-e^{\theta s_i}}}{y_i!}$$

Taking the log like we always do:

$$\log(P(y|e^{\theta s_i})) = \sum_i y_i \log(e^{\theta s_i}) - e^{\theta s_i} - \log(y_i!)$$

Simplify:

$$= \sum_i y_i \theta s_i - e^{\theta s_i} - \log(y_i!)$$

Take the derivative and set it equal to 0:

$$\sum_i y_i s_i - e^{\theta s_i} s_i = 0 \qquad \sum_i y_i s_i = \sum_i e^{\theta s_i} s_i$$

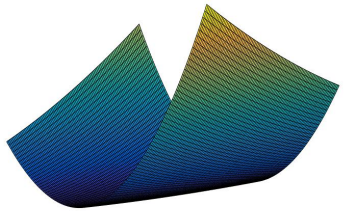
Using [LNP] models in the real world

Turn to numerical approaches to maximize log-likelihood!

Some of the previous problems are also best solved with a built-in optimizer.

WHY? Dividing matrices is a tricky business.
Doing things with big matrices is a tricky business.

So, where does the math end and the MATLAB/Python start?



Usually: still need to pick a noise model, formulate the relationship between firing rate and stimulus, and form equation for log-likelihood
Then: use built-in optimizers to find minimum of negative log-likelihood!
These optimizers start somewhere, and scale down the function to the min.

Let's do this for the LNP model:

Maximize: $\sum_i y_i \theta s_i - e^{\theta s_i}$.

Minimize: $-\sum_i y_i \theta s_i - e^{\theta s_i}$

(Put expression into vector notation)

`MATLAB: minimize` $\mathrm{sum}[\exp(\theta * s) - (y. * s) * \theta]$

```
[theta] = fminunc(@(theta)
myFun(theta,stim,y),init);
```

A note about numerical optimization

```
MATLAB: [theta] = fminunc(@(theta) myFun(theta,stim,y),init);
```

Need to write a function that takes parameter, stimulus, and vector of spike counts, and returns the negative log likelihood. Usually, these functions are faster if the gradient and Hessian (2nd derivatives) are also returned.

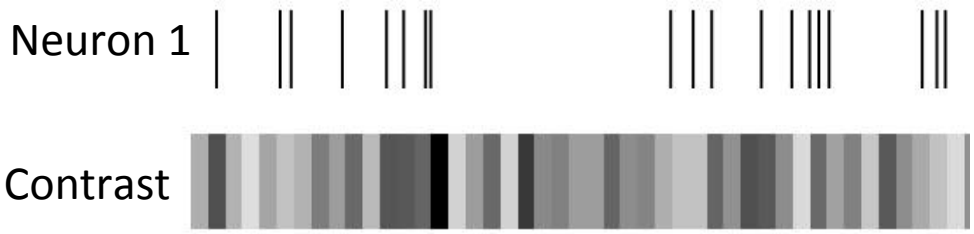
Remember, this is what we want to do:

Minimize: $-\sum_i y_i \theta s_i - e^{\theta s_i}$

```
function [f, df, hessian] = myFun(theta, stim, y)
    r = exp(stim * theta); % estimated rate given the current value of theta
    f = sum(r-y.*(stim*theta)); % the function we want to minimize
    df = stim'*r - stim'*y; % the derivative of this function with respect to theta
    hessian = stim'*(stim.*r) % the second derivative
return
```

Will put example code on the website

Real-world example with LNP



Perform an experiment by flashing squares of different contrast

Spiking at current time depends on contrast before spike – synaptic transmission is not instantaneous!

Mathematical formulation:

Let's bin according to the paradigm – 1 bin/frame
Let's say the spiking in bin *i* depends on the contrasts that happened 0-50 bins before that spike, and the neuron has some mean spiking rate independent of contrast

$$\hat{y}_i = e^{\theta_0 c_{i-0} + \theta_1 c_{i-1} + \theta_2 c_{i-2} + \theta_3 c_{i-3} + \theta_m}$$
$$\hat{y}_i = e^{c_i^T \theta}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_m \end{bmatrix} \quad c_i = \begin{bmatrix} c_{i-0} \\ c_{i-1} \\ c_{i-2} \\ c_{i-3} \\ 1 \end{bmatrix}$$

So, in the last problem we had: maximize $\sum_i y_i \theta s_i - e^{\theta s_i}$

Now, we have: maximize $\sum_i y_i c_i^T \theta - e^{c_i^T \theta}$

In order to plug this into an optimizer, we have to reformulate this into a sum of vector elements

$$C = \begin{bmatrix} c_1^T \\ \vdots \\ c_m^T \end{bmatrix}$$

`MATLAB: minimize` $\mathrm{sum}[\exp(C\theta) - y. * (C\theta)]$