

Nonlinear models

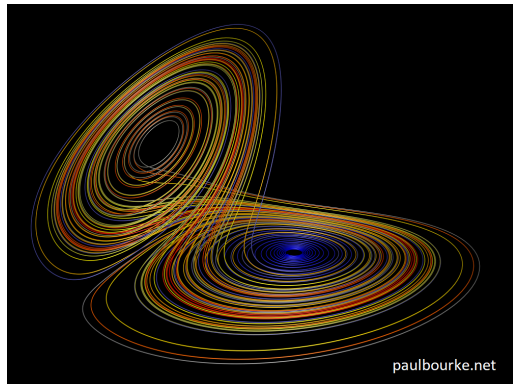
I don't think we're in Kansas anymore

Samuel Robinson, Ph.D.

Oct 13, 2023

Outline

- What are nonlinear models?
- Mechanistic models
 - Some common models
 - Strategies for fitting
- Empirical models
 - Some common models
 - GAMs



The Lorenz System: a classical 3D nonlinear system

What are nonlinear models?

- **Linear models** take the form:

$$\hat{y} = X\beta = b_0 1 + b_1 x_1 \dots + b_i x_i$$
$$y \sim \text{Normal}(\hat{y}, \sigma)$$

- **Nonlinear models** are any kind of model that can't be reduced to this linear (matrix) form:

$$\hat{y}_t = \hat{y}_{t-1} r \left(1 - \frac{\hat{y}_{t-1}}{k} \right)$$
$$\hat{y} = \frac{k n_0 e^{rt}}{k + n_0 (e^{rt} - 1)}$$
$$y \sim \text{Normal}(\hat{y}, \sigma)$$

Two common situations

- ① “I have governing equations for this system, and I want to fit them to my data”
 - e.g. Logistic growth equation, Michaelis-Menten kinematics, Ricker model
- ② “I don't know what equations represent my system, but I need some kind of *smooth* process that describes them”
 - e.g. Changes in organism population over growing season, changes in stock prices over time

Part 1: Mechanistic models

Governing equations

Dynamics of some systems can be described by a set of equations, either in *discrete* or *continuous* time

- Exponential growth: *Discrete time*

$$n_t = n_{t-1}r$$

- Predator prey cycles: *Discrete time*

$$\text{prey}_t = \text{prey}_{t-1}(r_1 - a_1 \text{pred}_{t-1})$$

$$\text{pred}_t = \text{pred}_{t-1}(a_2 \text{prey}_{t-1} - d)$$

- Exponential growth: *Continuous time*

$$\frac{dn}{dt} = nr$$

- Predator prey cycles: *Continuous time*

$$\frac{d\text{prey}}{dt} = r - a_1 \text{pred}$$

$$\frac{d\text{pred}}{dt} = a_2 \text{prey} - d$$

Some other common dynamic models

- Logistic growth

$$n_t = n_{t-1} \left(1 + r \left(1 - \frac{n_{t-1}}{k} \right) \right)$$

- Beverton-Holt model

$$N_t = \frac{R_0 N_{t-1}}{1 + N_{t-1}/M}$$

- Michaelis-Menten

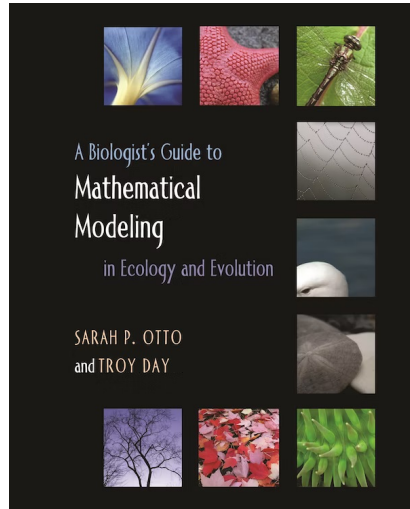
$$\frac{dp}{dt} = \frac{V_{max} a}{K_m + a}$$

- Susceptible-Infected-Recovered (SIR) model

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta IS}{N} \\ \frac{dI}{dt} &= \frac{\beta IS}{N} - \gamma I \\ \frac{dR}{dt} &= \gamma I\end{aligned}$$

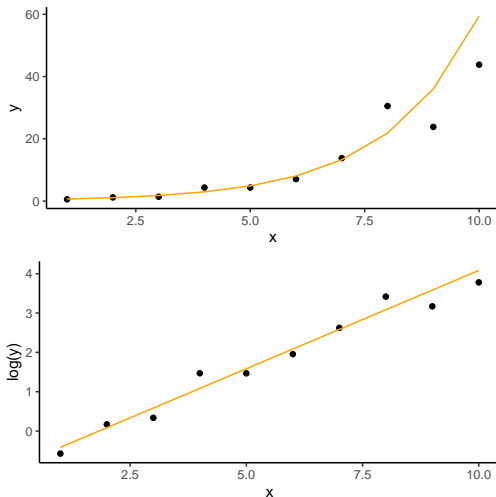
Where do these equations come from?

- Mostly from literature, sometimes from your own derivations
- Can be derived from causal models, flow diagrams, organismal life cycles
- Math-heavy topic for another class!
If you're interested, I might start with this book:



Fitting nonlinear models: transformations

- Sometimes you can transform your data to approximate nonlinear models
- e.g. $y = b_0 e^{xb_1}$ (Exponential growth)
 - Transformation:
 $\ln(y) = \ln(b_0 e^{xb_1}) =$
 $\ln(b_0) + \ln(e^{xb_1}) = \ln(b_0) + xb_1$
 - Linear model in R: `lm(log(y)~x)`
- This can cause problems because *distances* don't mean the same thing at all ranges of x-values; in general, it's better to use a NLM if you're able to

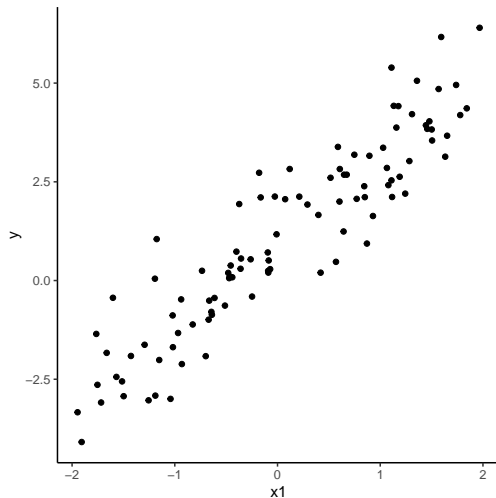


Fitting nonlinear models: simple example

- We have a *pretty good idea* what rules the system is following, and we want to figure out the parameters that it uses
- Simple example: let's start with a simple linear model, where we have 2 parameters b_0 and b_1 that we're looking for

$$\hat{y} = X\beta = b_0 + b_1x_1$$

- We're trying to find the parameters of a line that *most closely* fits our data:

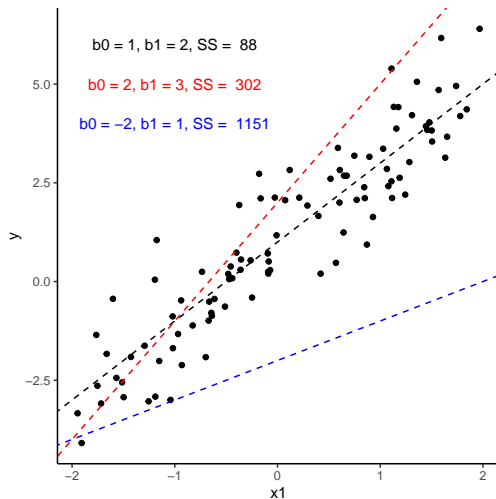


Fitting mechanistic models (cont.)

- How might we define “closest fit” in a mathematical sense?
- One common measure is *sum of squared distances*. This is just the difference between the data and the **line**:

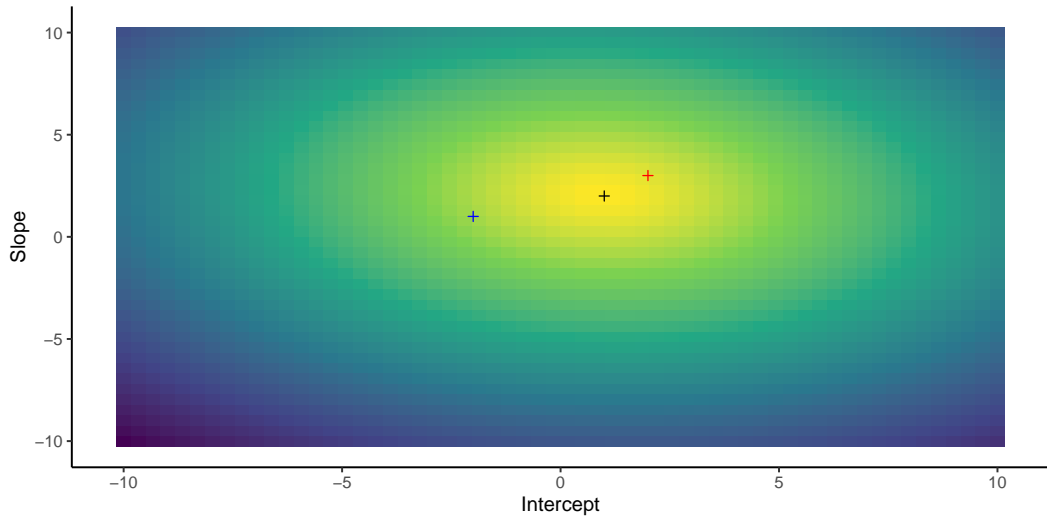
$$S = \sum_{i=1}^N (y_i - (b_0 + b_1 x_i))^2$$

- Here are three “guesses” at the slope and intercept, along with their SS scores. Which one looks to be the best?



Map of fitting surface

We can try this for a whole bunch of intercepts and slopes:



Getting R to do this

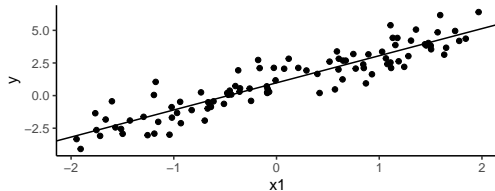
- It's pretty clear where the best intercept and slope is, but how do we get R to do this?
- First, we need a function that returns SS given a set of parameters:

```
#Function to calculate SS  
ssFun <- function(B,xdat,ydat){  
  sum((ydat - (B[1] + B[2]*xdat))^2)  
}
```

- Next, we use the `optim` function to find the intercept and slope values that return the minimum value of SS. How did it do? (Actual values: $b_0:1$, $b_1:2$)

```
#Starts at 0,0 and "looks around" from there  
optim(par = c(0,0) , fn = ssFun)
```

```
## $par  
## [1] 0.9769795 2.0779779  
##  
## $value  
## [1] 86.78819  
##  
## $counts  
## function gradient  
##      67      NA  
##  
## $convergence  
## [1] 0  
##  
## $message  
## NULL
```



General framework

Here are some simple rules for fitting models:

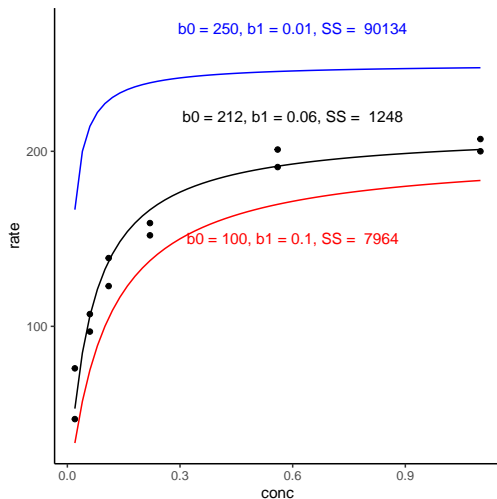
- ① Think about how your system works. What rules do you think your system follows?
- ② Write down these rules as equations, with **parameters** that control the system at time t
 - Some differential ($\frac{dx}{dt}$) equations can sometimes be solved by hand
 - Otherwise you need to use an ODE solver (fme in R)
- ③ Come up with an *objective function* that describes the differences between predictions and actual data
- ④ Get R to find parameters that *minimize* the objective function
- ⑤ See how well your model predicted your data:
 - Are all of your parameters *identifiable* from your data?
 - Do you need to go back to step 1?

Fitting mechanistic models: nonlinear example

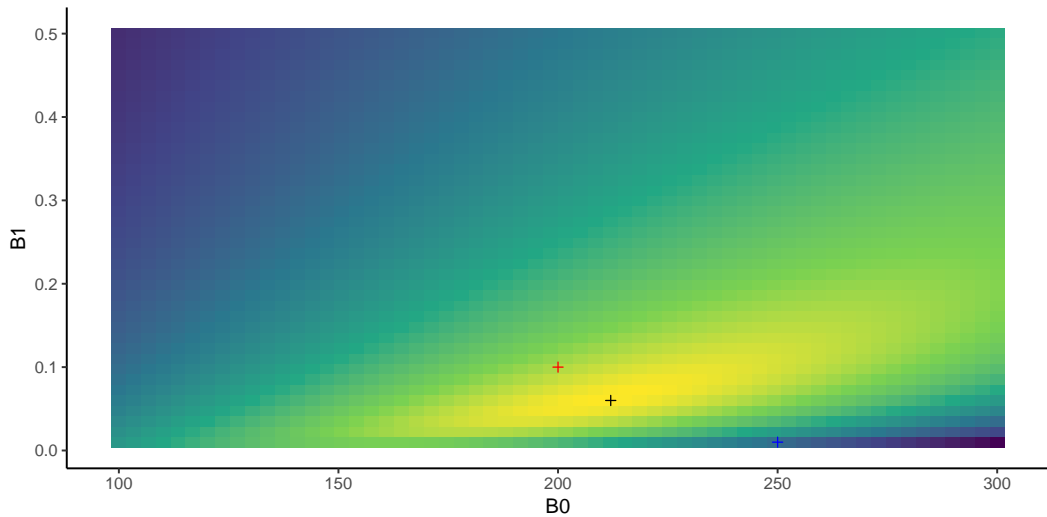
- Let's move on to a nonlinear model (Michaelis-Menten), where we also have 2 parameters b_0 and b_1 that we're looking for

$$\hat{y} = \frac{b_0 x_1}{b_1 + x_1}$$

- Again, we're trying to find the parameters of a nonlinear curve that *most closely* fits our data:



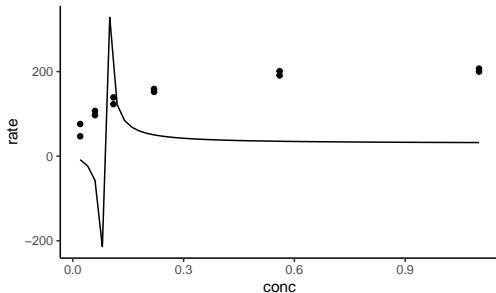
Nonlinear example (cont.)



Get R to do it

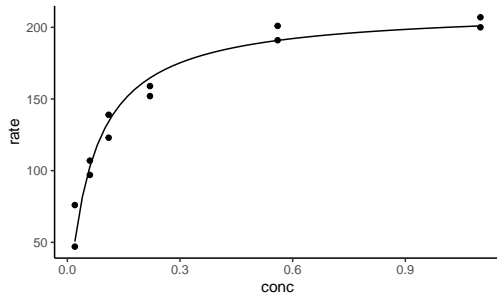
Mind your starting parameters!

```
## $par
## [1] 29.4800715 -0.0910235
##
## $value
## [1] 196776.4
##
## $counts
## function gradient
##      195      NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```



Better to start with more “realistic” values

```
## $par
## [1] 212.68974107  0.06412427
##
## $value
## [1] 1195.449
##
## $counts
## function gradient
##      81      NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```



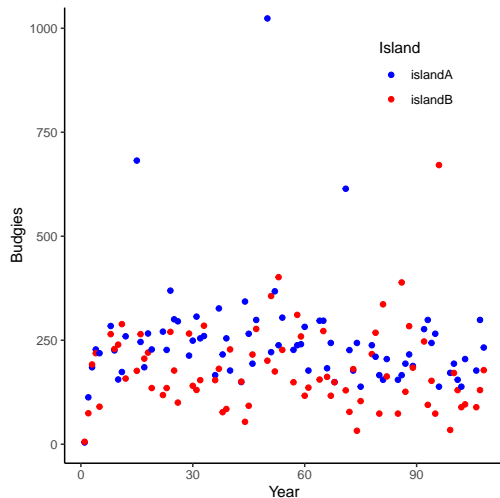
First challenge: logistic growth

The logistic growth model

$n_t = n_{t-1}r(1 - \frac{n_{t-1}}{K})$ has the definite solution:

$$n(t) = \frac{Kn_0e^{rt}}{K + n_0(e^{rt} - 1)}$$

- Write an *objective function* that takes a vector of parameters $([n_0, K, r])$ as its first input, plus time steps t and a vector of n values
- Get R to fit a logistic growth model to a dataset of *budgie* numbers (found [here](#))
- Does it look like the K value (carrying capacity) differs much between budgies on different islands?



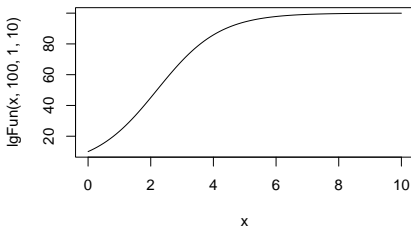
Logistic growth results

- First we create a function that calculates the logistic growth curve at time t

```
lgFun <- function(t,K,r,n0){  
  (K*n0*exp(r*t))/(K+n0*(exp(r*t)-1))}
```

- curve is handy for showing what a set of function output along x will look like. Looks like it works!

```
curve(lgFun(x,100,1,10),from=0,to=10)
```



- Next we make the objective function, which is the sum of squared differences between y_{dat} and the logistic growth function. B is now a single vector that contains K , r , and n_0 :

```
ssFunLG <- function(B,xdat=bDat$year,ydat){  
  sum((lgFun(xdat,B[1],B[2],B[3])-ydat)^2)}
```

- Unfortunately, this gives weird answers (negative starting values). This is actually just a flat line:

```
opt1 <- optim(c(100,1,1),ssFunLG,ydat=bDat$is1  
opt1$par
```

```
## [1]      245.686237      6.434686 -11263.870767
```

Logistic growth results (cont.)

- Let's try writing the objective function again, but now we'll scale the n_0 parameter ($B[3]$ below) to be on the *log* scale. This prevents it from going below zero:

```
ssFunLG2 <- function(B,xdat=bDat$year,ydat) sum((lgFun(xdat,B[1],B[2],exp(B[3]))-ydat)^2)
```

- This runs, and starting values (n_0) are now in log-units. This looks better, but since n_0 is so low, it might just be best to set it at a value close to zero:

```
opt2 <- optim(c(100,1,1),ssFunLG2,ydat=bDat$islandA)
opt2$par
```

```
## [1] 250.644893    6.569662   -7.823500
```

- One last re-write of the objective function! Now we set n_0 to 0.01, which is close to zero, and put it in the place of $B[3]$:

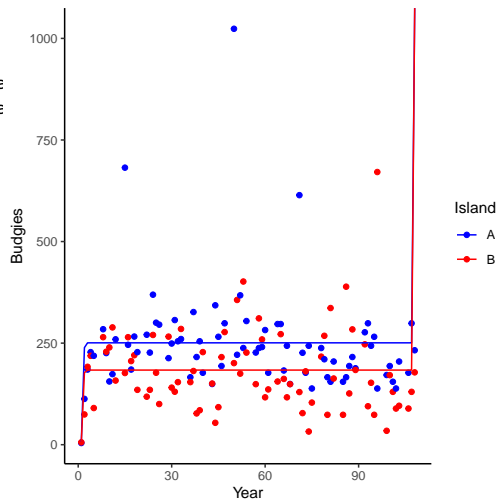
```
ssFunLG3 <- function(B,xdat=bDat$year,ydat) sum((lgFun(xdat,B[1],B[2],0.01)-ydat)^2)
```

Logistic growth results (cont.)

- Now we fit a model for each island group.

```
opt3 <- optim(c(100,1),ssFunLG3,ydat=bDat$isla  
opt4 <- optim(c(100,1),ssFunLG3,ydat=bDat$isla
```

- Looks like reasonable values:
[1] 250.71918 6.57206
[1] 183.664368 6.572059
- Now we can plot the results: it looks like Island A has a higher K value!



How do you get SEs on parameters?

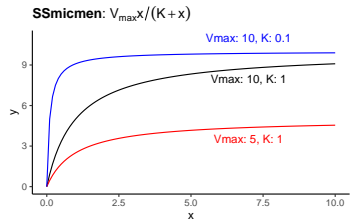
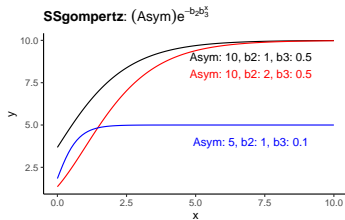
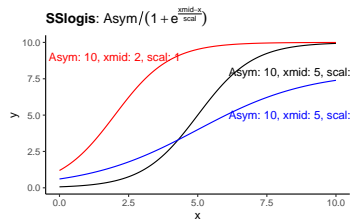
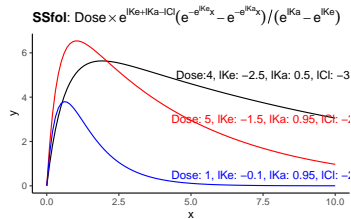
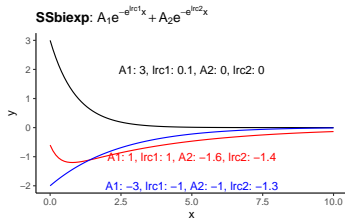
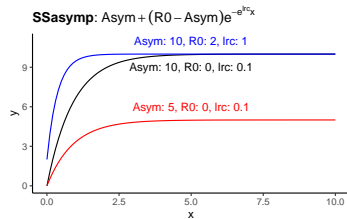
- Easy way: bootstrapping
 - Re-samples the data and re-fits the model a bunch of times
- Medium way: Monte Carlo sampling or *likelihood profiling*
 - Tests how the objective function changes around the final parameters, and uses this to calculate SEs and p-values for your parameters
- Hard way: calculate Hessian of the objective function (serious math)

I don't waaaaaant to!



Good news: someone already did the scary math for you!

A bunch of popular nonlinear models are already available in R:

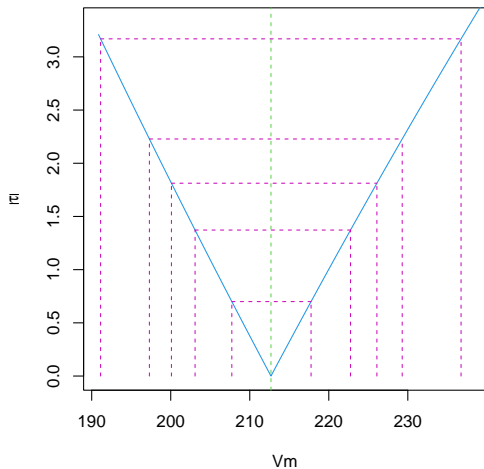


Even better news: nls can easily do likelihood profiles

Here's the Michaelis-Menten model from before:

```
##  
## Formula: rate ~ SSmicmen(conc, Vm, K)  
##  
## Parameters:  
##      Estimate Std. Error t value Pr(>|t|)  
## Vm 2.127e+02  6.947e+00  30.615 3.24e-11 ***  
## K  6.412e-02  8.281e-03   7.743 1.57e-05 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 10.93 on 10 degrees of freedom  
##  
## Number of iterations to convergence: 0  
## Achieved convergence tolerance: 1.929e-06
```

Likelihood profile for V_m



Second challenge

Try fitting the same budgies data using `nls` instead of `optim`, and test whether the difference in `K` values between islands is significant!

Syntax for `nls`:

```
#Define the function yourself
nls(y ~ (K*0.01*exp(r*x))/(K+0.01*(exp(r*x)-1)), data = dat, start = list(K=1,r=1))

#Use a preset function
nls(y ~ lgFun(x,K,r,0.01), data=dat,start=list(K=1,r=1)) #Uses a function
```

Note: unless you use one of the self-starting (SS) models, you have to provide reasonable *starting values* for `nls`

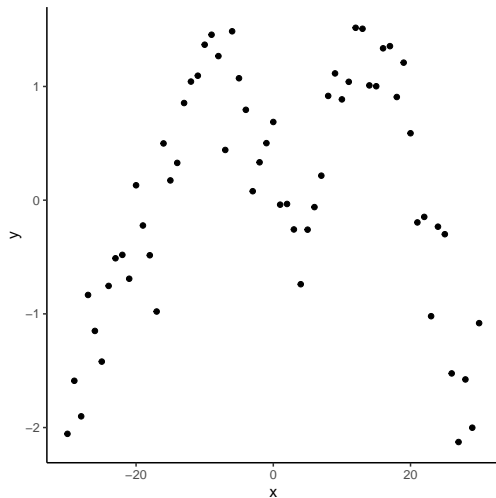
Second challenge results

```
nmod1 <- nls(islandA ~ lgFun(year,Kval,rval,0.001), ##  
             data=bDat,start=list(Kval=250,rval=6.5), ## Formula: islandA ~ lgFun(year, Kval, rval, 0.001)  
             ## Parameters:  
             ##      Estimate Std. Error t value Pr(>|t|)  
             ## Kval 250.654      14.275  17.559 < 2e-16 ***  
             ## rval   6.107       1.005   6.077 4.52e-08 ***  
             ## ---  
             ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
             ##  
             ## Residual standard error: 124.4 on 76 degrees of freedom  
             ##  
             ## Number of iterations to convergence: 3  
             ## Achieved convergence tolerance: 1.619e-06  
             ##  
             ## Formula: islandB ~ lgFun(year, Kval, rval, 0.001)  
             ## Parameters:  
             ##      Estimate Std. Error t value Pr(>|t|)  
             ## Kval 183.631      11.350   16.18 < 2e-16 ***  
             ## rval   5.872       1.116    5.26 1.29e-06 ***  
             ## ---  
             ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
             ##  
             ## Residual standard error: 98.93 on 76 degrees of freedom  
             ##  
             ## Number of iterations to convergence: 3  
             ## Achieved convergence tolerance: 6.935e-07
```

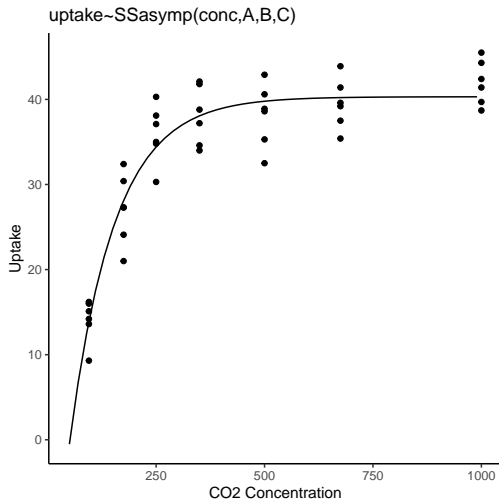
Part 2: Empirical models

Empirical smoothing

- Sometimes we don't know the specific rules that govern your system, but we want to know the *general shape*
 - e.g. population changes across time or space, temperature across seasons
- We want something that can give us *general predictions* across the range of your data without actually dealing with the underlying process
- Solution: “empirical” smoothing



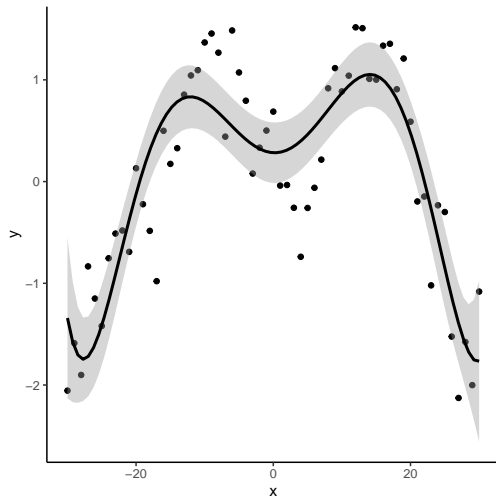
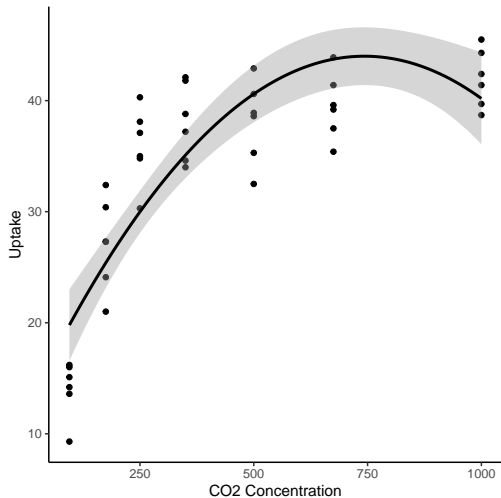
“Guess the family”



- Sometimes you can use a preset nonlinear family that looks “similar enough” to your data
- e.g. SSlogis, SSweibull
- See also: “Transformations” slide from first section

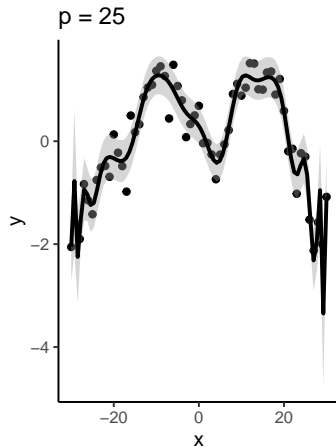
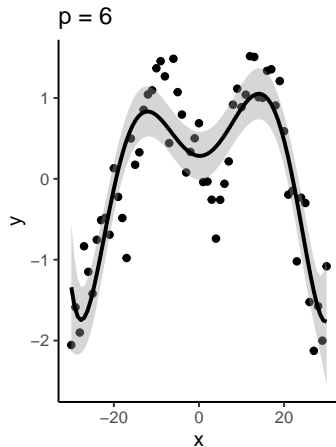
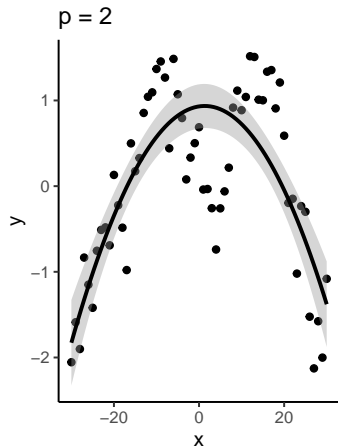
Polynomial smoothing

If the pattern is “wiggly”, you can use polynomials:



Problems with polynomials

- How many orders of polynomials do you use? Limited to discrete values
- Polynomial models don't do well outside of the range of prediction, especially at the edges of your data

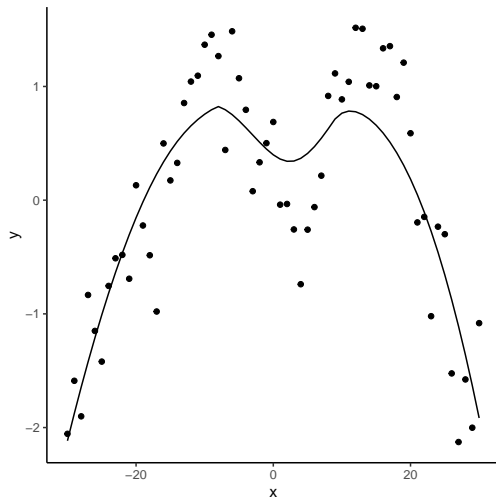


LOESS smoothers

- LOESS (LOcal regrESSion) fits simple polynomial model at *each* data point,
- Similar to a moving-average smoother (“window” of nearest N data points)

Problems:

- Computation-heavy: fits a weighted model for every data point
- Require a fair bit of data to get good predictions, sensitive to outliers
- Similar to polynomials, doesn't do well outside the range of the data



GAM: Generalized Additive Models

- Additive models are a hybrid linear model that use *basis functions* to approximate “wiggly” data
- Uses random effects to penalize curves in order to avoid overfitting (i.e. “just wiggly enough”)
- The `mgcv` package can deal with a large range of additive models, from a large range of distributions (count data, presence/absence, survival, categorical, and more)
- This package is useful for a wide variety of things, and it’s definitely worth learning

How do GAMs work?

Penalized GAMs are actually a *random effects model*, and take the form:

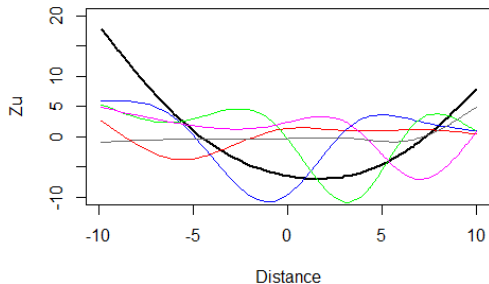
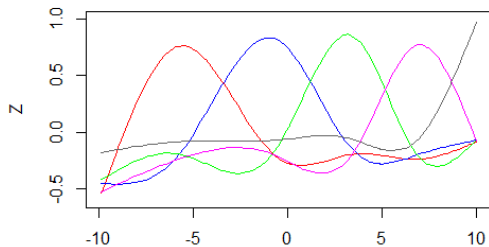
Prediction = Fixed Effect + Random Effect

$$\hat{y} = X\beta + Zu$$

$$y \sim \text{Normal}(\hat{y}, \sigma)$$

$$u \sim \text{Normal}(0, \lambda S)$$

- Creates *basis functions* across the range of data stored in columns of Z
- Finds values u
- λS penalty term: selects for optimal “wiggleness”



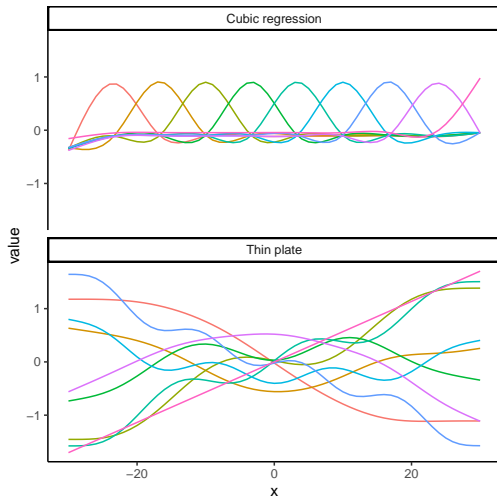
GAM example

Let's see how this works on a dataset:

```
#Fits a GAM using a cubic regression basis  
gmod1 <- gam(y~s(x,bs='cr'),data=vdat)
```

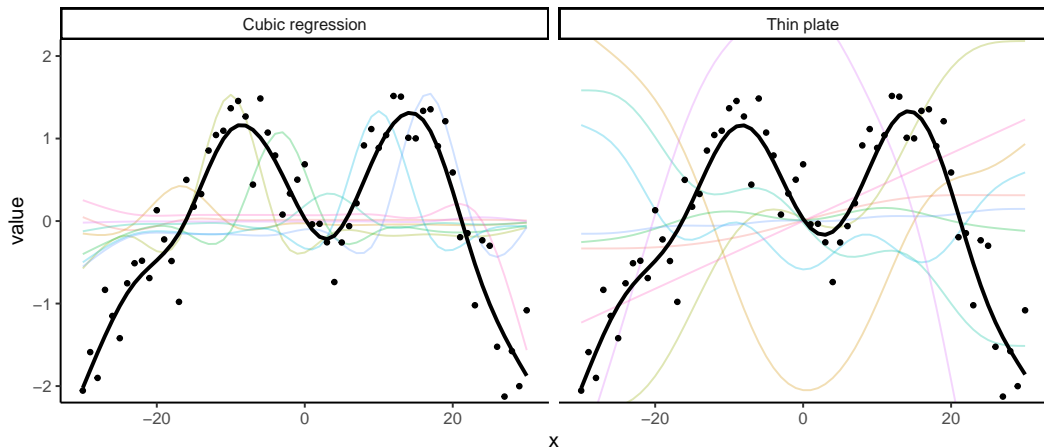
```
#Thin-plate spline basis (default)  
gmod2 <- gam(y~s(x,bs='tp'),data=vdat)
```

- There are a bunch of different basis functions available (see `?smooth.terms`) from `mgcv`, but `cr` and `tp` are common



GAM example (cont.)

When you multiply basis functions by the coefficients (u) and add them:



The additive model follows the *general* shape of the data well!

More GAM things

- The ever-useful `ggpredict` can make partial effects plot for GAMs, and `DHARMa` is useful for checking assumptions (if you're using non-normal distributions)
- Smoothing is automated in `mgcv`, but the number of basis functions (`k`) isn't. `gam.check()` can help you check whether you might need to increase `k`
- If you have very few `x` values, or the pattern isn't very wiggly, then you might need a *lower* `k` value

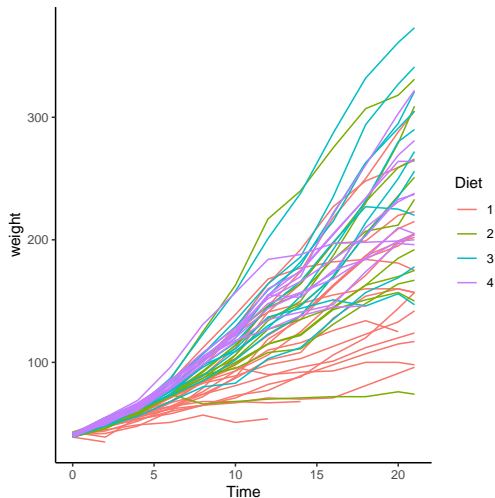
- You can also add non-smooth terms and random intercepts:

```
gmod3 <- gam(y ~ a + b + s(t) + s(site,bs='re'))
```

- If you want “interactions” with smooths, you have to use the `by` argument:
`s(t,by=myFactor)`
- Versions of `cr` and `tp` with extra penalization: `cs` and `ts`. This can remove terms *completely* from the model
- If `x` values are “circular” (e.g. days of the year), you can use *circular* splines (`cc`)

Third challenge

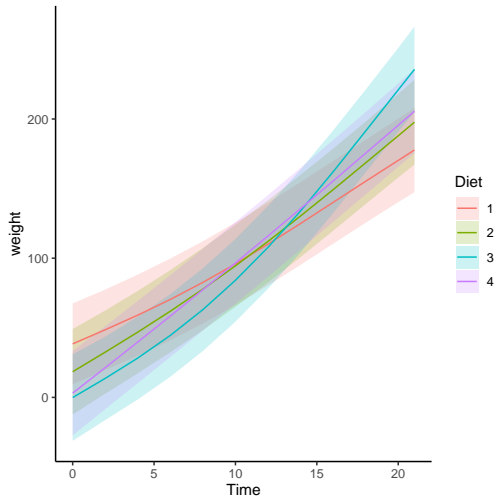
- Try fitting a GAM to the first individual Chick in the `ChickWeight` dataset (included with R)
- Check whether the number of basis functions is appropriate. Does your answer change if you use more or less?
- If you're feeling adventurous, try fitting multiple individual chicks using a *random intercept*, and see if the Diet parameter changes chick growth!



Third challenge results

```
cmod <- gam(weight ~ s(Time,by=Diet)+  
             s(Chick,bs='re'),data=ChickWeight)  
summary(cmod)
```

```
##  
## Family: gaussian  
## Link function: identity  
##  
## Formula:  
## weight ~ s(Time, by = Diet) + s(Chick, bs = "re")  
##  
## Parametric coefficients:  
##             Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 120.677      3.982    30.3   <2e-16 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Approximate significance of smooth terms:  
##             edf Ref.df      F p-value  
## s(Time):Diet1  2.203  2.733 265.87 <2e-16 ***  
## s(Time):Diet2  1.932  2.401 283.15 <2e-16 ***  
## s(Time):Diet3  2.536  3.144 386.85 <2e-16 ***  
## s(Time):Diet4  1.429  1.737 478.17 <2e-16 ***  
## s(Chick)       45.554 49.000  14.34 <2e-16 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## R-sq.(adj) =  0.881   Deviance explained = 89.2%  
## GCV = 666.18   Scale est. = 603.19    n = 578
```



To do: final nonlinear modeling challenge

- For those of you who have nonlinear or “GAMmy” data... time to update your models. If not...
- Fit the latter ChickWeight data using a GAM and a nonlinear mixed effects model (`nlmer`). How does Diet change chick growth?
 - For a family of nonlinear model that are commonly used for growth curves, try `SSlogis` or `SSgompertz` for starters
 - `nlmer` is similar to `lme4` but can have intercepts OR nonlinear parameters as random variables, specified as a *three-part model*.
 - e.g. `nlmer(weight ~ SSgompertz(Time,Asym,b2,b3) ~ (Asym|Chick), startVec = c(50,1,1))` (You also need a vector of starting values)
- How does this compare to a “standard” linear mixed effects model? Does an `lmer` model get the same answers, and does it meet the modelling assumptions?
- **There are multiple ways to do this!** Try a couple ways and see what works.