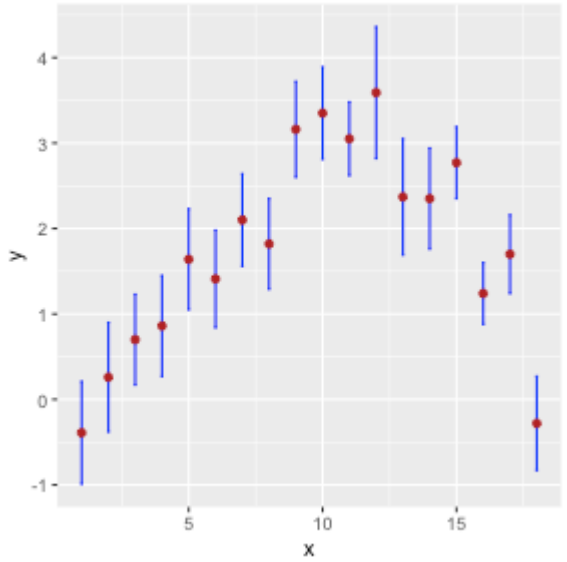
Numerical Optimization

36-600

Example

- We will adopt the same dataset that we analyzed in the last class:

x	1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	11.00	12.00	13.00	14.00	15.00	16.00	17.00	18.00
y	-0.39	0.26	0.70	0.86	1.64	1.41	2.10	1.82	3.16	3.35	3.05	3.59	2.37	2.35	2.77	1.24	1.70	-0.28
e	0.60	0.64	0.53	0.59	0.59	0.57	0.54	0.53	0.56	0.54	0.43	0.77	0.68	0.59	0.42	0.36	0.46	0.55



Optimizing the Coefficients of an Arbitrary Function

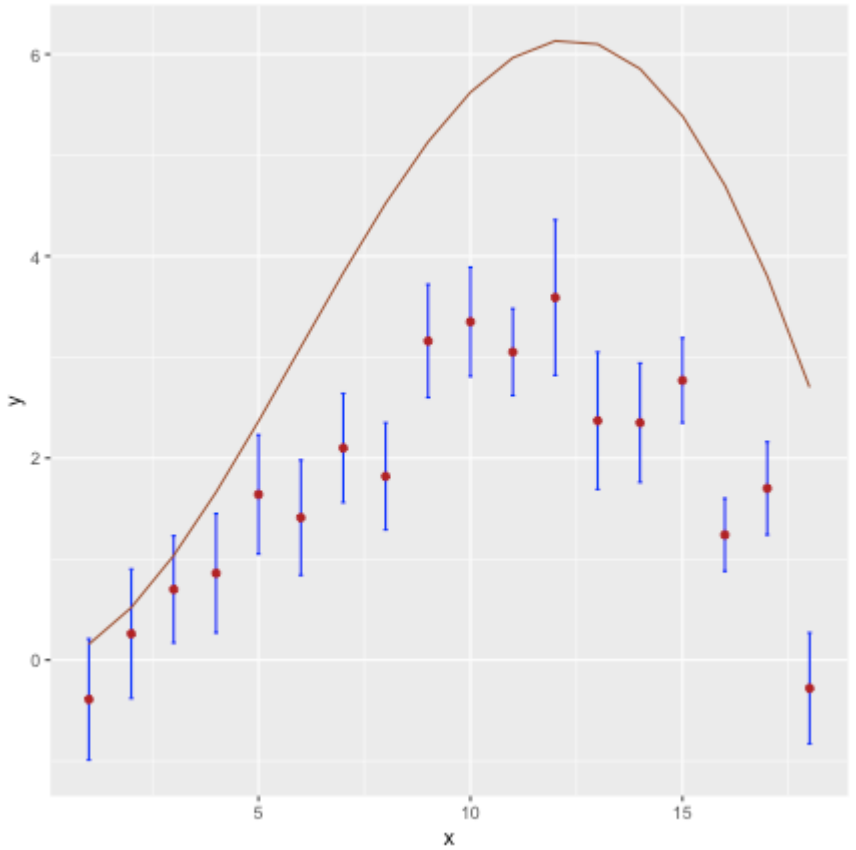
- Suppose you read somewhere that for these data, the underlying model is apparently

$$f(x) = \sin\left(\frac{\pi}{\beta}x\right)x^{\alpha}$$

- α and β are unknown quantities that has to be determined
- OK...let's start by picking what we think might be good values for α and β: 0.75 and 20

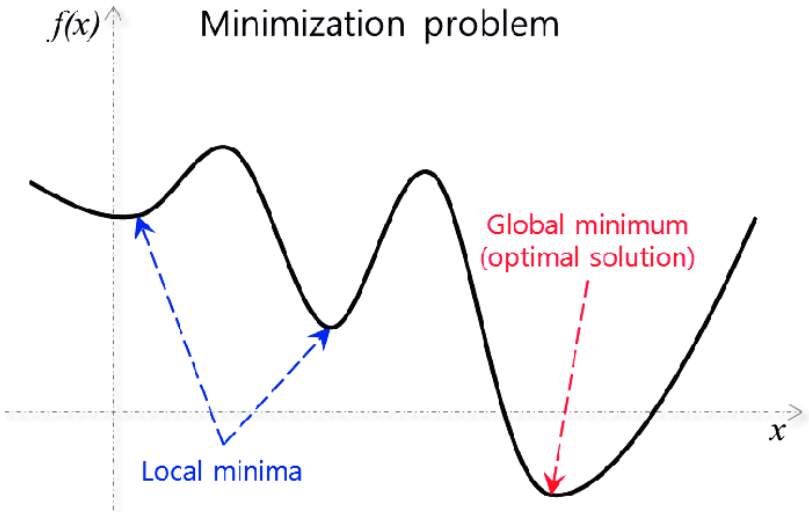
```
mod.x <- 1:18 ; mod.y <- sin(pi*x/20)*x^0.75
ggplot(data=df,mapping=aes(x=x,y=y)) +
  geom_errorbar(aes(ymin=y-e, ymax=y+e), width=.1,co
  geom_line(mapping=aes(x=mod.x,y=mod.y),color="oliv
```

- Not bad, but not quite...how do we determine $\hat{\alpha}$ and $\hat{\beta}$?



Digression: Optimization

- We want to code an *optimizer*, a piece of code that determines the optimal values for α and β
- In the figure, x represents the quantity being optimized (for us, α), and $f(x)$ represents some function of x (the *objective function*) that quantifies how good the fit is
- You want to determine the value of x that (hopefully globally) minimizes the objective function, in a computationally efficient manner
- So, what might the objective function $f(x)$ be in practice?



Digression: Optimization

- A workhorse objective function is real-life applications is the *chi-square function*:

$$\chi^2 = \sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{\hat{\sigma}_i^2} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \frac{1}{\hat{\sigma}_i^2}$$

  - $\hat{y}_i$ is the predicted value for y_i
  - $\hat{\sigma}_i^2$ is the estimated variance (square of the standard deviation, i.e., square of the estimated uncertainty)

- Look at the way we write χ^2: it is the sum of the squared errors, weighted…and the weight is the inverse-variance weighting we discussed in the last class
  - so in the context of our data, we can write χ^2 as

$$\chi^2 = \sum_{i=1}^{n} w_i (y_i - g(x_i))^2 = \sum_{i=1}^{n} \frac{(y_i - g(x_i))^2}{e_i^2} = \sum_{i=1}^{n} \frac{(y_i - \sin(x_i \pi/\beta) x_i^{\alpha})^2}{e_i^2}$$

Digression: Optimization

- What are some of the useful properties of χ^2?
  - because $|y_i - \hat{y}_i|$ should be $\approx e_i$ for a good model, we can take the optimized value of χ^2 and divide by $n - p$ (where p is the number of free parameters in $g(x)$...here, $p = 2$) and check to see if the value is approximately 1
  - we can utilize the χ^2 goodness-of-fit test to see if the optimized model provides an acceptable fit to the data
- The p-value for the GoF test is

```
1 - pchisq(chi2.min,n-p) # chi2.min is value of chi2 for optimized parameters
```

Code the Optimizer

- The function `optim()` is a workhorse optimization function within `R`
- Here's how we'd use it:

```
fit.fun <- function(par,data)
{
  x <- data$x; y <- data$y; e <- data$e
  return( sum((y-sin(x*pi/par[2])*x^par[1])^2/e^2) )
}
par <- c(0.75,20)                  # initial guesses for alpha, beta
op.out <- suppressWarnings(optim(par,fit.fun,data=df))
op.out$value                       # the minimum chi-square value
```

```
## [1] 14.04143
```

```
op.out$par                         # the estimated parameter values
```

```
## [1]  0.4745909 18.7649516
```

Code the Optimizer

```
fit.fun <- function(par,data)
{
  x <- data$x; y <- data$y; e <- data$e
  return( sum((y-sin(x*pi/par[2])*x^par[1])^2/e^2) )
}
par    <- c(0.75,20)                  # initial gues
op.out <- suppressWarnings(optim(par,fit.fun,data=df
op.out$value # the minimum chi-square value
op.out$par   # the estimated parameter values
```

- In the call to `optim()`, the first two arguments are `par` (a vector of parameter values, which here is of length 2 and represents α and β) and `fit.fun`
- `fit.fun()` itself expects `par` as its first argument; however, for the function to actually work, we need to pass in a data frame which contains x, y, and e...
- ...so back in the call to `optim()`, we need to explicit say what the variable `data` represents
  - we tack on a third argument, `data=df`, where `df` is the data frame defined earlier that has the values of x, y, and e
- Finally, note how we make an initial guesses for α and β (`par <- c(0.75,20)`)
  - the optimizer works fine with these guesses, but beware that in more complicated problems we run the risk of not finding the global minimum for χ^2 if our initial guess is "bad"

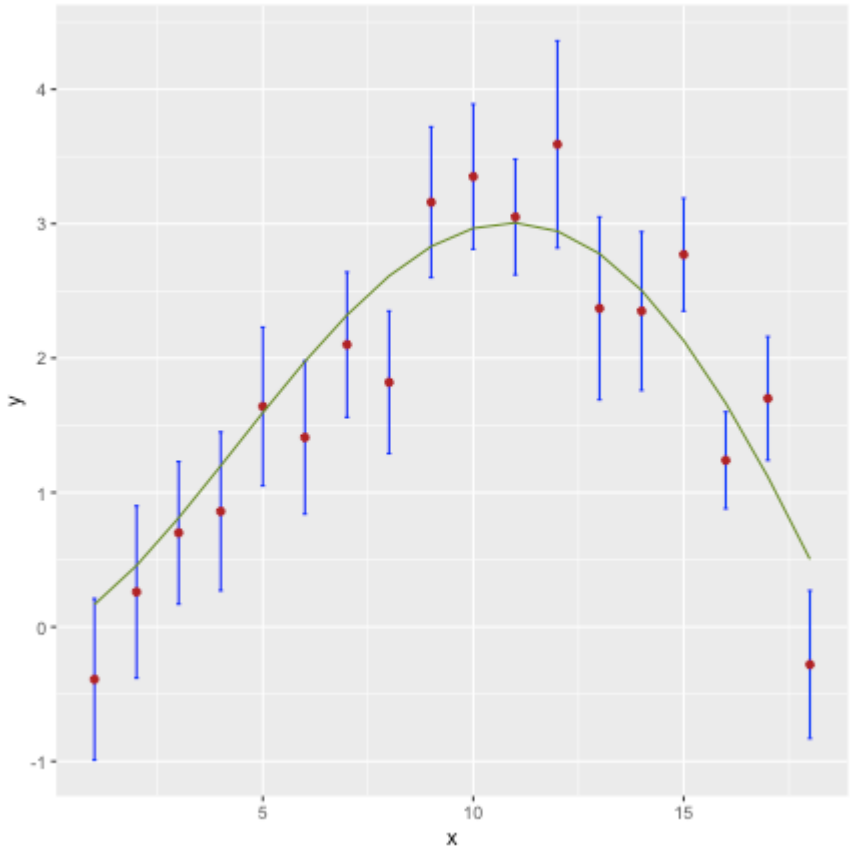
Is This an Acceptable Model?

- $\chi^2/(n-p) = \chi^2/(n-1) = 14.041/16 \approx 0.875 \Rightarrow$ Yes
- We run the chi-square GOF test:

```
1 - pchisq(op.out$value,nrow(df)-1)
```

```
## [1] 0.6641673
```

- The p-value is 0.66
    - we fail to reject the null hypothesis that our model is an acceptable one



Wait...Let's Look at That Model Again

- "Wait," you say...
  - "...not all the blue error bars overlap the model...that's an indication the model is bad, right?"
- No...
  - the uncertainties as being estimates of the standard deviation: given x, the sampling of y has this amount of uncertainty
  - if the normal distribution governs the uncertainty, then we expect $\approx$ 68% of the blue lines to overlap the model
  - if we doubled the length of the blue lines, then we would expect $\approx$ 95% of the blue lines to overlap the model...etc.
- What we observe: 13 of 18 blue lines overlap the model, or 72.2%
  - given the small sample size, this is completely consistent with an expected 68% overlap rate