Lecture 11: Distributions as Models

36-350

1 October 2014

Previously

- R functions for regression models
- R functions for probability distributions

Agenda

- Distributions from data
- Review of R for theoretical distributions
- Fitting distributions to data
- Checking distributions against data

===

```
data("cats",package="MASS")
```

Cats may be too saccharine (unless we think about where `cats$Hwt` comes from)

Let's Grab some Data

The "pdfetch" package grabs economic and financial information from public-domain sources, e.g., Federal Reserve or Yahoo Finance

```
require(pdfetch)
```

```
## Loading required package: pdfetch
```

```
# Corrections Corp. of America
CXW <- pdfetch_YAHOO("CXW",fields="adjclose",
  from=as.Date("2000-01-01",
  to=as.Date("2014-09-30")))
```

===

```
summary(CXW)
```

```
##      Index                CXW
##  Min.   :2000-01-03   Min.   : 0.69
##  1st Qu.:2003-09-15   1st Qu.: 6.26
##  Median :2007-05-23   Median :13.67
##  Mean   :2007-05-21   Mean   :14.28
##  3rd Qu.:2011-01-27   3rd Qu.:19.47
##  Max.   :2014-10-06   Max.   :36.12
```

```
plot(CXW)
```

If the prison-industrial complex is too grim, refer back to the cats

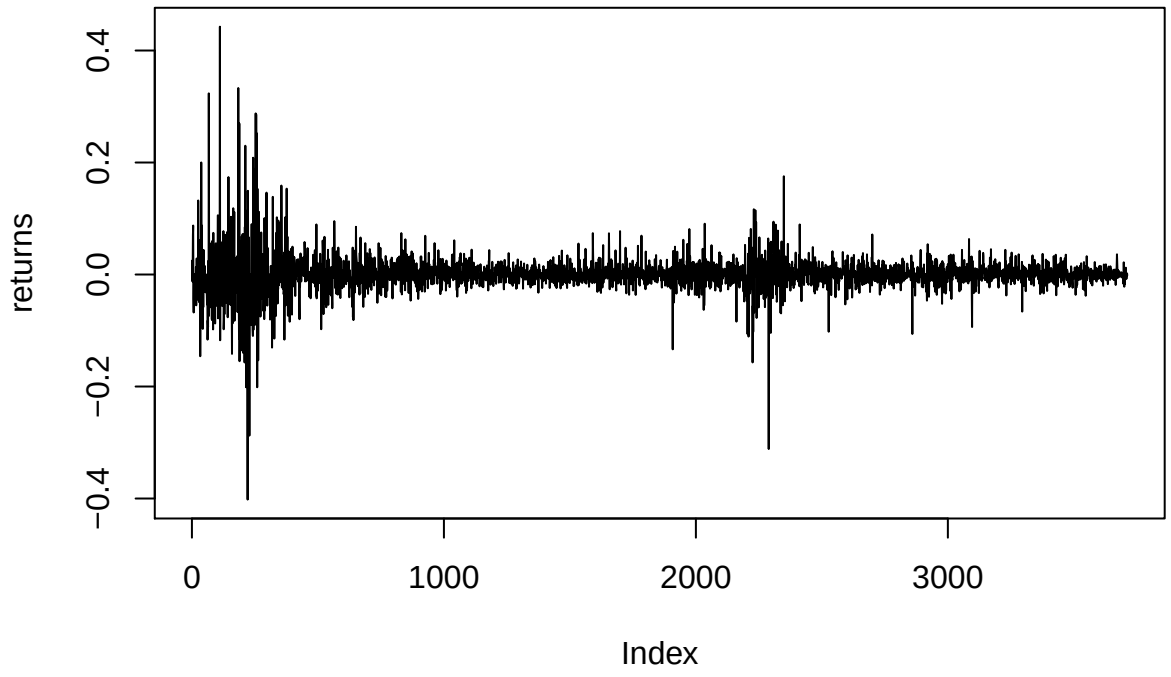
Let's Transform Some Data

The price p_t doesn't matter, what matters are the returns $r_t = \log(p_t/p_{t-1})$

```
returns <- na.omit(as.vector(diff(log(CXW$CXW))))
summary(returns)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4020 -0.0104  0.0000  0.0005  0.0108  0.4430
```

```
plot(returns,type="l")
```



The Data's Distribution

`quantile(x,probs)` calculates the quantiles at `probs` from `x`

```
quantile(returns,c(0.25,0.5,0.75))
```

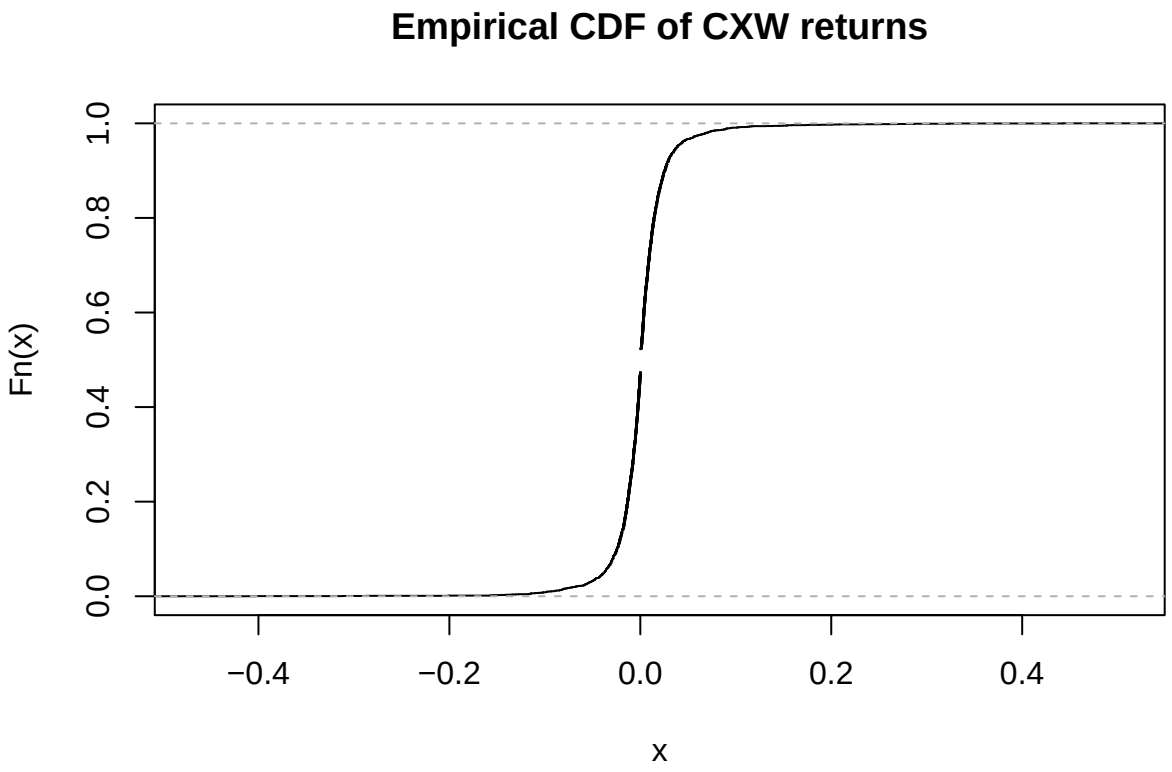
```
##      25%      50%      75%
## -0.01044  0.00000  0.01079
```

Getting distributions from data (cont'd.)

`ecdf()` - *e* mpirical *c* umulative *d* istribution *f* unction; no assumptions but also no guess about distribution between the observations

In math, ECDF is often written as $\widehat{F}$ or $\widehat{F}_n$

```
plot(ecdf(returns), main="Empirical CDF of CXW returns")
```



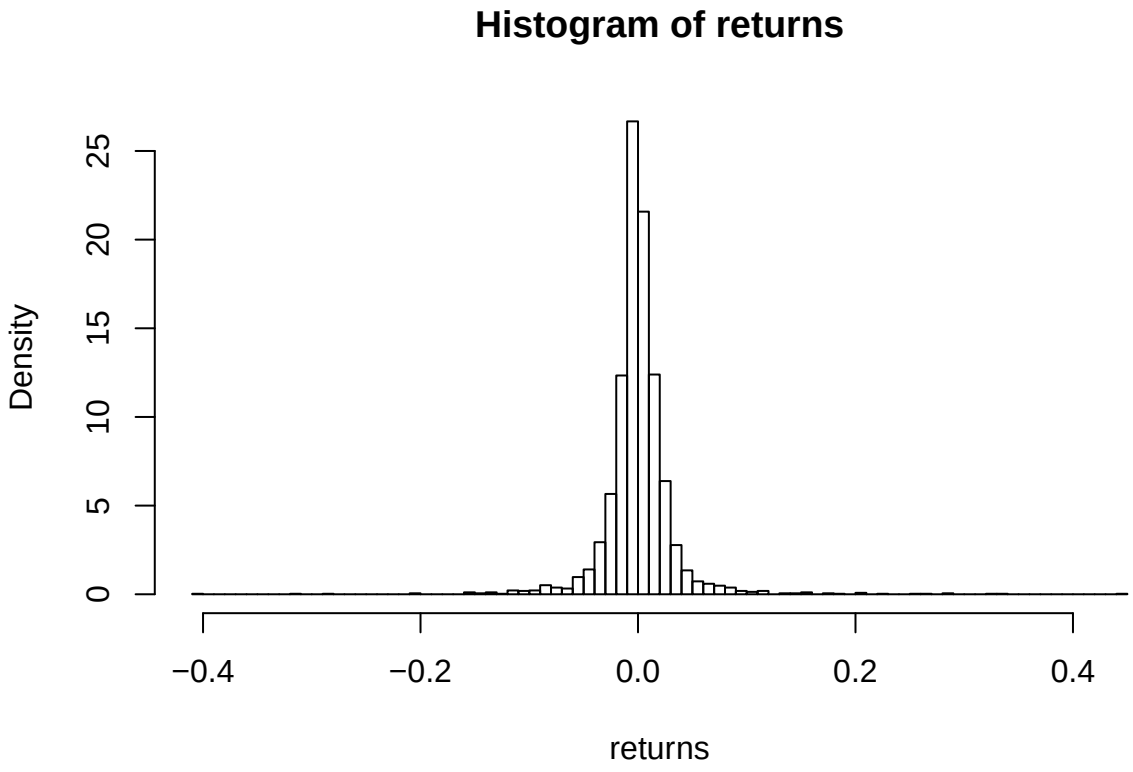
Conceptually, `quantile` and `ecdf` are inverses to each other

Getting Probability Densities from Data

`hist(x)` calculates a histogram from x - divide the data range up into equal-width bins and *count* how many fall into each bin - *Or* divide bin counts by (total count)*(width of bin), and get an estimate of the probability density function (pdf)
Produces plot as a default side-effect

===

```
hist(returns,n=101,probability=TRUE)
```



Getting probability densities from data (cont'd.)

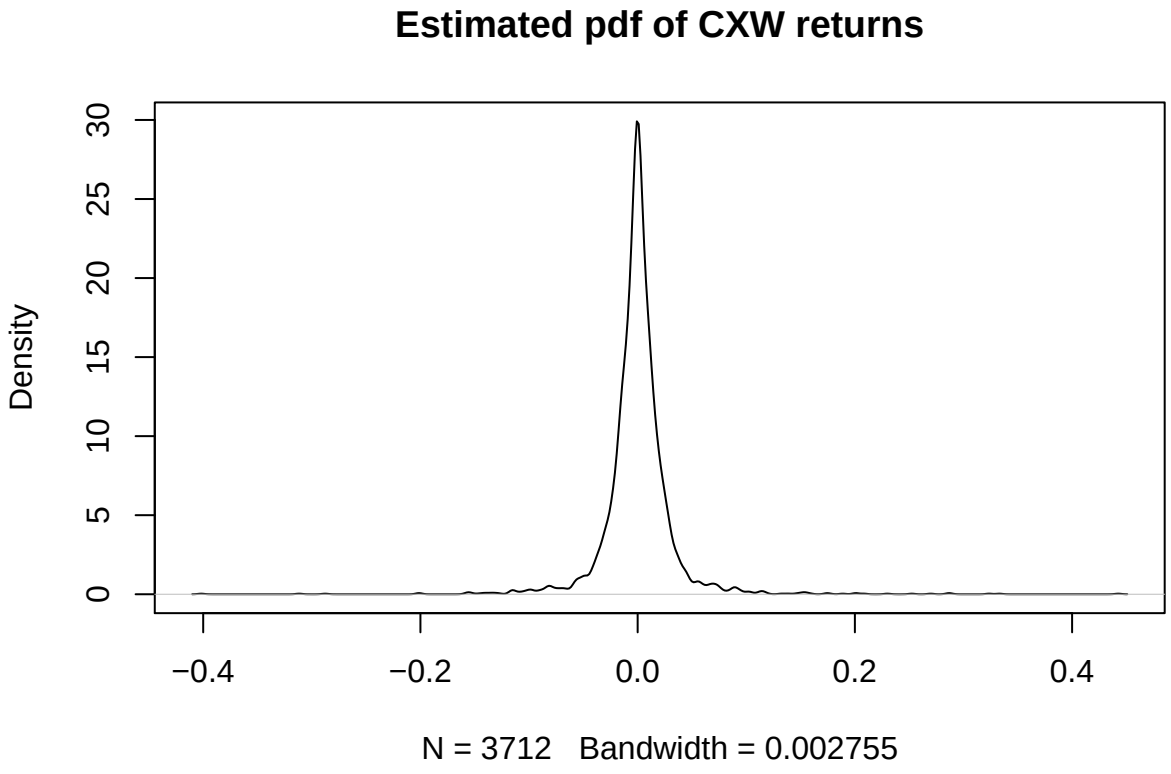
`density(x)` estimates the density of `x` by counting how many observations fall in a little window around each point, and then smoothing
"Bandwidth" ≈ width of window around each point
Technically, a "kernel density estimate"
Take 36-402 in the spring for much more

Remember: `density()` is an *estimate* of the pdf, not The Truth

`density` returns a collection of x, y values, suitable for plotting

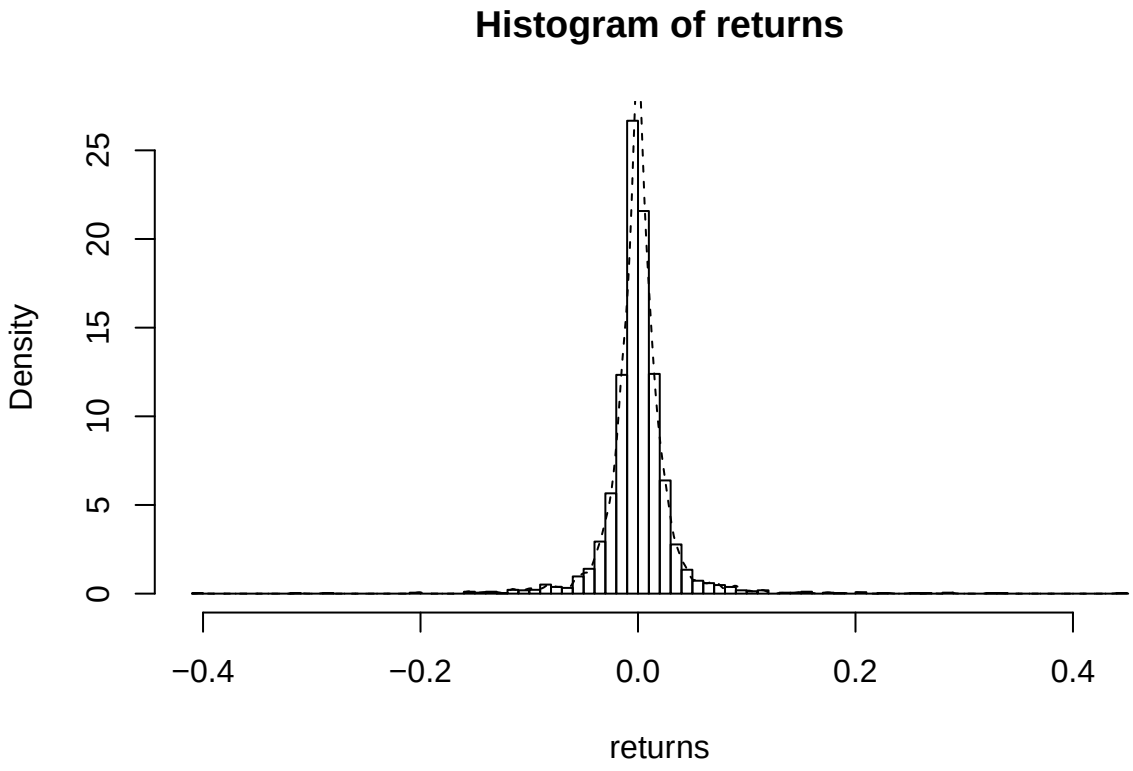
===

```
plot(density(returns),main="Estimated pdf of CXW returns")
```



===

```
hist(returns,n=101,probability=TRUE)
lines(density(returns),lty="dashed")
```



Getting distributions from data (cont'd.)

`table()` - tabulate outcomes, most useful for discrete spaces; remember to normalize if you want probabilities

```
plot(table(cats$Sex)/nrow(cats),ylab="probability")
```

Who Cares About the Distribution of the Data?

- Overly detailed: every single observation recorded as a separate tick
  - *Too much information*
- The exact set of samples would never repeat if we re-ran things anyway
  - *That information is wrong*
- Try to *summarize* what will *generalize* to other situations
  - Use a model, remember the model's parameters

R commands for distributions

- `d`*foo* = the probability *d* ensity (if continuous) or probability mass function of *foo* (pdf or pmf)
- `p`*foo* = the cumulative *p* robability function (CDF)
- `q`*foo* = the *q* uantile function (inverse to CDF)
- `r`*foo* = draw *r* andom numbers from `foo` (first argument always the number of draws)

`?Distributions` to see which distributions are built in

If you write your own, follow the conventions

Examples

```
dnorm(x=c(-1,0,1),mean=1,sd=0.1)
```

```
## [1] 5.521e-87 7.695e-22 3.989e+00
```

```
pnorm(q=c(2,-2)) # defaults to mean=0,sd=1
```

```
## [1] 0.97725 0.02275
```

```
dbinom(5,size=7,p=0.7,log=TRUE)
```

```
## [1] -1.147
```

```
qchisq(p=0.95,df=5)
```

```
## [1] 11.07
```

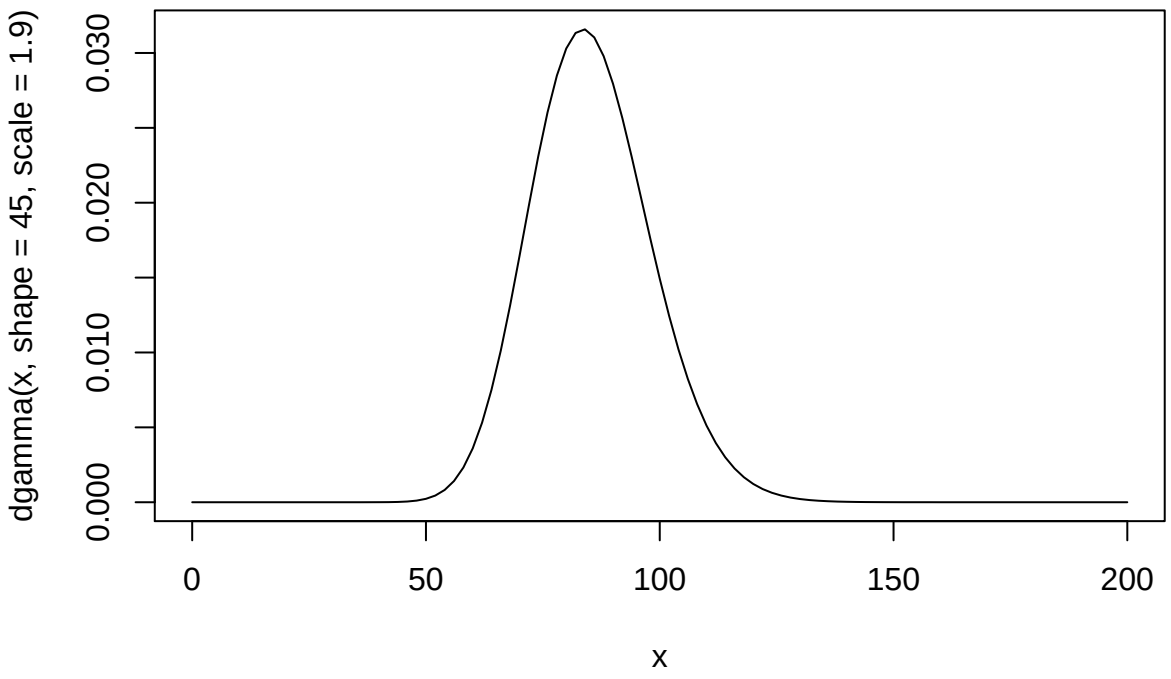
```
rt(n=4,df=2)
```

```
## [1] -3.4438 -0.7827 -1.0216 -0.3335
```

Displaying Probability Distributions

`curve` is very useful for the `d`, `p`, `q` functions:

```
curve(dgamma(x,shape=45,scale=1.9),from=0,to=200)
```



N.B.: the `r` functions aren't things it makes much sense to plot

How Do We Fit Distributional Models to the Data?

- Match moments (mean, variance, etc.)
- Match other summary statistics
- Maximize the likelihood

Method of Moments (MM), Closed Form

- Pick enough moments that they **identify** the parameters
  - At least 1 moment per parameter; algebraically independent
- Write equations for the moments in terms of the parameters
  e.g., for gamma
  $$\mu = as \ , \ \sigma^2 = as^2$$
- Do the algebra by hand to solve the equations
  $$a = \mu^2/\sigma^2 \ , \ s = \sigma^2/\mu$$
- Code up the formulas (did this in lab 3)

```
gamma.est_MM <- function(x) {
  m <- mean(x); v <- var(x)
  return(c(shape=m^2/v, scale=v/m))
}
```

MM, Numerically

- Write functions to get moments from parameters (usually algebra)
- Set up the difference between data and model as another function

```
gamma.mean <- function(shape,scale) { return(shape*scale) }
gamma.var <- function(shape,scale) { return(shape*scale^2) }
gamma.discrepancy <- function(shape,scale,x) {
  return((mean(x)-gamma.mean(shape,scale))^2 + (var(x)-gamma.mean(shape,scale))^2)
}
```

- Minimize it

More Generally...

- Nothing magic about moments
- Match other data summaries, say the median
  - Or even more complicated things, like ratios of quantiles
  - You did this in lab
- If you can't solve exactly for parameters from the summaries, set up a discrepancy function and minimize it
  - You are doing this in the HW
- The summaries just have to converge on population values

Maximum Likeihood

- Usually we think of the parameters as fixed and consider the probability of different outcomes, $f(x;\theta)$ with θ constant and x changing
- **Likelihood** of a parameter value = $L(\theta)$ = what probability does θ give to the data?
  - For continuous variables, use probability density
  - $f(x;\theta)$ but letting θ change while data constant
  - *Not* the probability of θ, if that even makes sense
- **Maximum likelihood** = guess that the parameter is whatever makes the data most likely
- Most likely parameter value = **maximum likelihood estimate** = **MLE**

Likelihood in Code

- With independent data points x_1, x_2, x_n, likelihood is

$$L(\theta) = \prod_{i=1}^{n} f(x_i;\theta)$$

- Multiplying lots of small numbers is numerically bad; take the log:

$$\ell(\theta) = \sum_{i=1}^{n} \log f(x_i;\theta)$$

- In pseudo-code:

```
loglike.foo <- function(params, x) {
  sum(dfoo(x=x,params,log=TRUE))
}
```

(will see how to do this for real later in class)

What Do We Do with the Likelihood?

- We maximize it!
- Sometimes we can do the maximization by hand with some calculus
  - For Gaussian, MLE = just match the mean and variance
  - For Pareto, MLE $\widehat{a} = 1 + 1/\overline{\log(x/x_{\min})}$
- Doing numerical optimization
  - Stick in a minus sign if we're using a minimization function

Why Use the MLE?

- Usually (but not always) *consistent*: converges on the truth as we get more data
- Usually (but not always) *efficient*: converges on the truth at least as fast as anything else
- There are some parameters where the maximum isn't well-defined (e.g. $x_{\min}$ for a Pareto)
- Sometimes the data is too aggregated or mangled to use the MLE (as with the income data in lab 5)

fitdistr

MLE for one-dimensional distributions can be done through `fitdistr` in the `MASS` package

It knows about most the standard distributions, but you can also give it arbitrary probability density functions and it will try to maximize them
A starting value for the optimization is optional for some distributions, required for others (including user-defined densities)

Returns the parameter estimates and standard errors
SEs come from large-n approximations so use cautiously

fitdistr examples

Fit the gamma distribution to the cats' hearts:

```
require(MASS)
```

```
## Loading required package: MASS
```

```
fitdistr(cats$Hwt, densfun="gamma")
```

```
##    shape     rate
##   20.300    1.910
##  ( 2.373) ( 0.226)
```

Returns: estimates above, standard errors below

fitdistr Examples (cont'd.)

Fit the Students-t distribution to the returns:

```
fitdistr(returns,"t")
```

```
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
```

```
##         m           s          df
##   0.0001797   0.0126275   1.8792569
##  (0.0002676) (0.0003036) (0.0783154)
```

Here parameters are location (m), scale (s) and degrees of freedom (very heavy tails)

fitdistr Examples (cont'd.)

User-defined density:

```
dpareto <- function(x,exponent,xmin,log=FALSE) {
  f <- (exponent-1)/xmin * (x/xmin)^(-exponent)
  f <- ifelse(x<xmin,NA,f)
  if(!log) { return(f) } else (return(log(f)))
}
# Fit pareto to large absolute returns
  # Parameters given outside the "start" list are fixed
fitdistr(abs(returns)[abs(returns)>0.05], densfun=dpareto,
         start=list(exponent=2.5), xmin=0.05)
```

```
## Warning: one-dimensional optimization by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly
```

```
##   exponent
##   2.898
##  (0.122)
```

Checking Your Estimator

- simulate, then estimate; estimates should converge as the sample grows

```
gamma.est_MM(rgamma(100,shape=19,scale=45))
```

```
## shape scale
## 22.89 36.51
```

```
gamma.est_MM(rgamma(1e5,shape=19,scale=45))
```

```
## shape scale
## 18.89 45.27
```

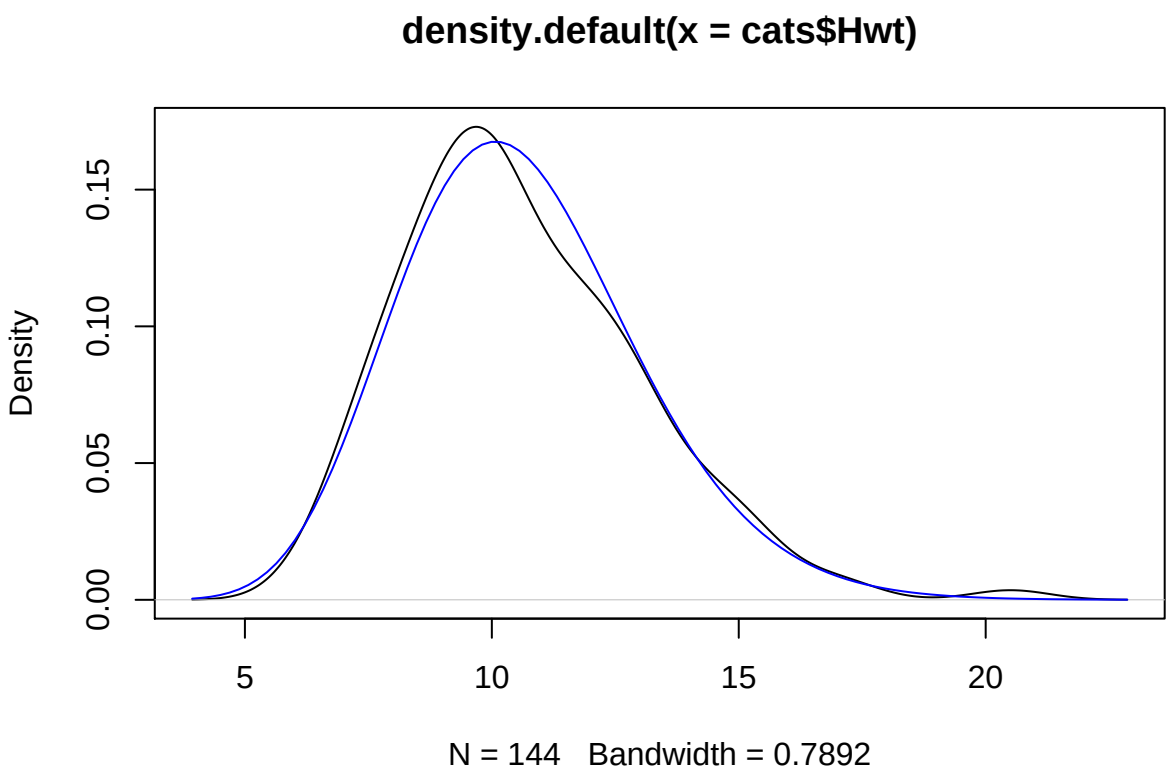
```
gamma.est_MM(rgamma(1e6,shape=19,scale=45))
```

```
## shape scale
## 19.01 44.99
```

Checking the Fit

Use your eyes: Graphic overlays of theory vs. data

```
plot(density(cats$Hwt))
cats.gamma <- gamma.est_MM(cats$Hwt)
curve(dgamma(x,shape=cats.gamma["shape"],scale=cats.gamma["scale"]),add=TRUE,col="blue")
```



Checking the Fit (cont'd.)

- Calculate summary statistics *not* used in fitting, compare them to the fitted model

```
# Really bad and good days for CXW shareholders, per model:
qnorm(c(0.01,0.99),mean=mean(returns),sd=sd(returns))
```

```
## [1] -0.07586  0.07691
```

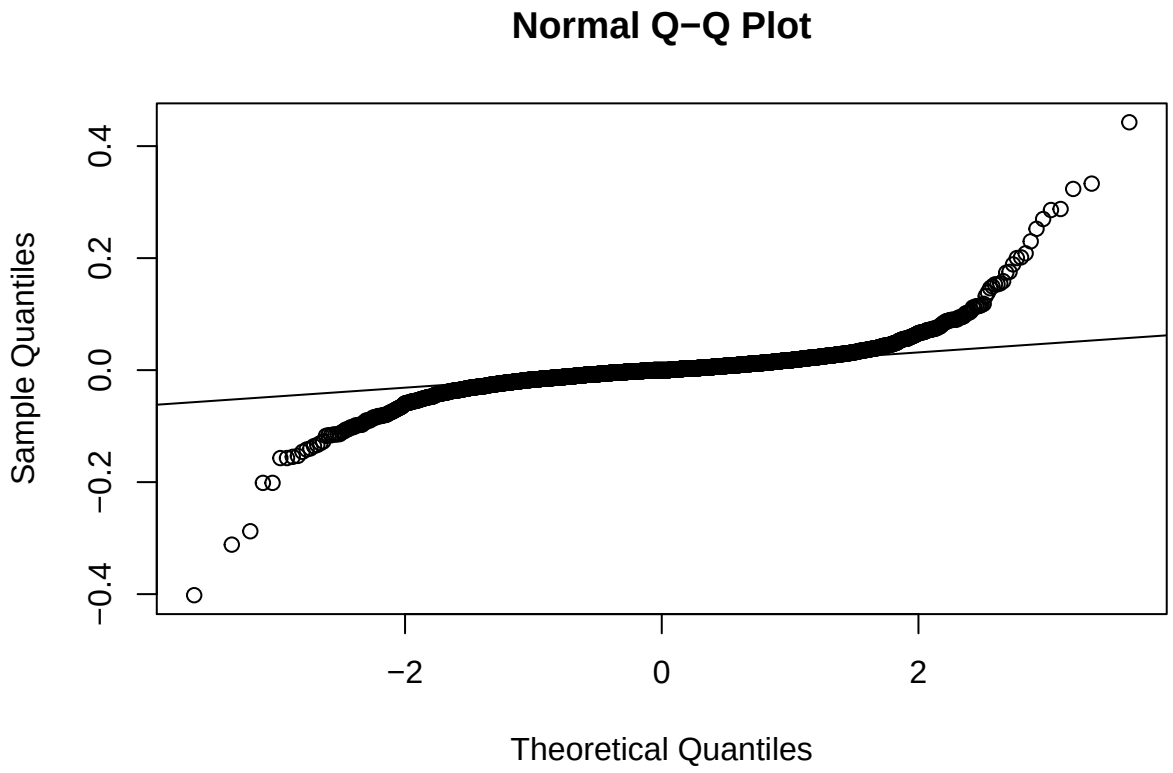
```
# As it happened:
quantile(returns,c(0.01,0.99))
```

```
##       1%      99%
## -0.09372  0.09414
```

Quantile-Quantile (QQ) Plots

- Plot theoretical vs. actual quantiles
- *or* plot quantiles of two samples against each other
- Ideally, a straight line when the distributions are the same
- `qqnorm`, `qqline` are specialized for checking normality

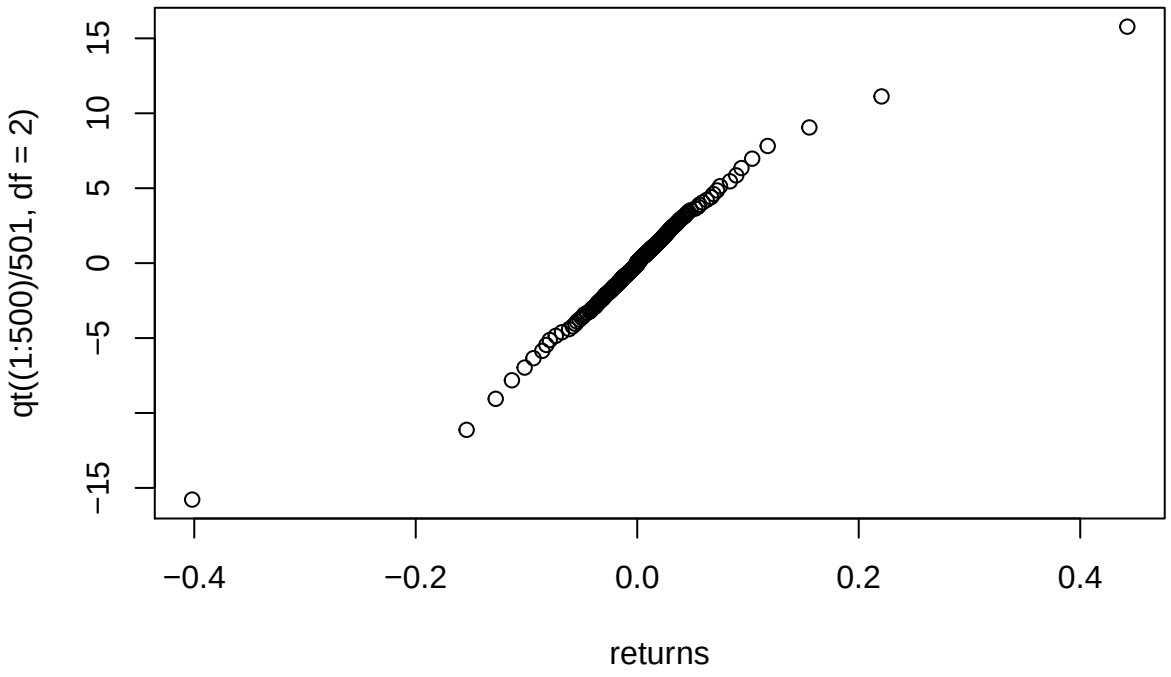
```
qqnorm(returns); qqline(returns)
```



QQ Plots (cont'd)

- 'qqplot(x,y) will do a Q-Q plot of one vector against another

```
qqplot(returns,qt((1:500)/501,df=2))
```

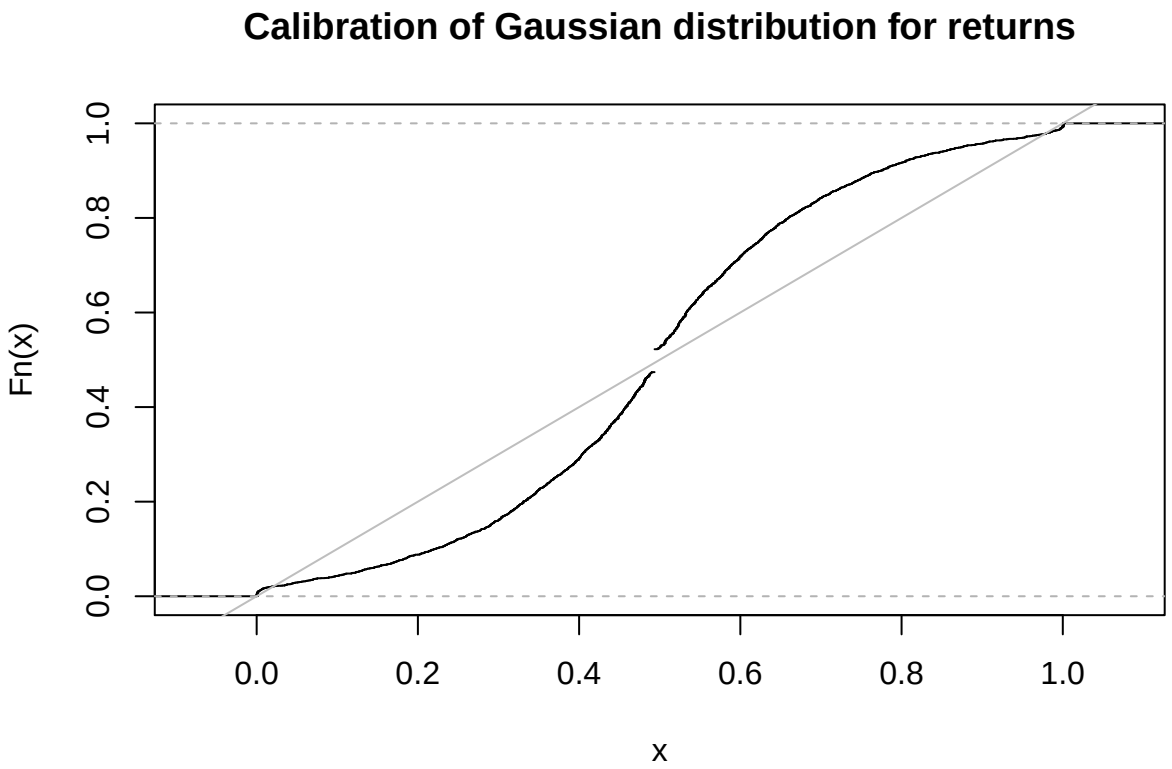


Calibration Plots

- If the distribution is right, 50% of the data should be below the median, 90% should be below the 90th percentile, etc.
- Special case of **calibration** of probabilities: events with probability p% should happen about p% of the time, not more and not less
- We can look at calibraton by calculating the (empirical) CDF of the (theoretical) CDF and plotting
  - Ideal calibration plot is a straight line up the diagonal
  - Systematic deviations are a warning sign

Making a Calibration Plot

```
plot(ecdf(pnorm(returns, mean=mean(returns), sd=sd(returns))),
     main="Calibration of Gaussian distribution for returns")
abline(0,1,col="grey")
```



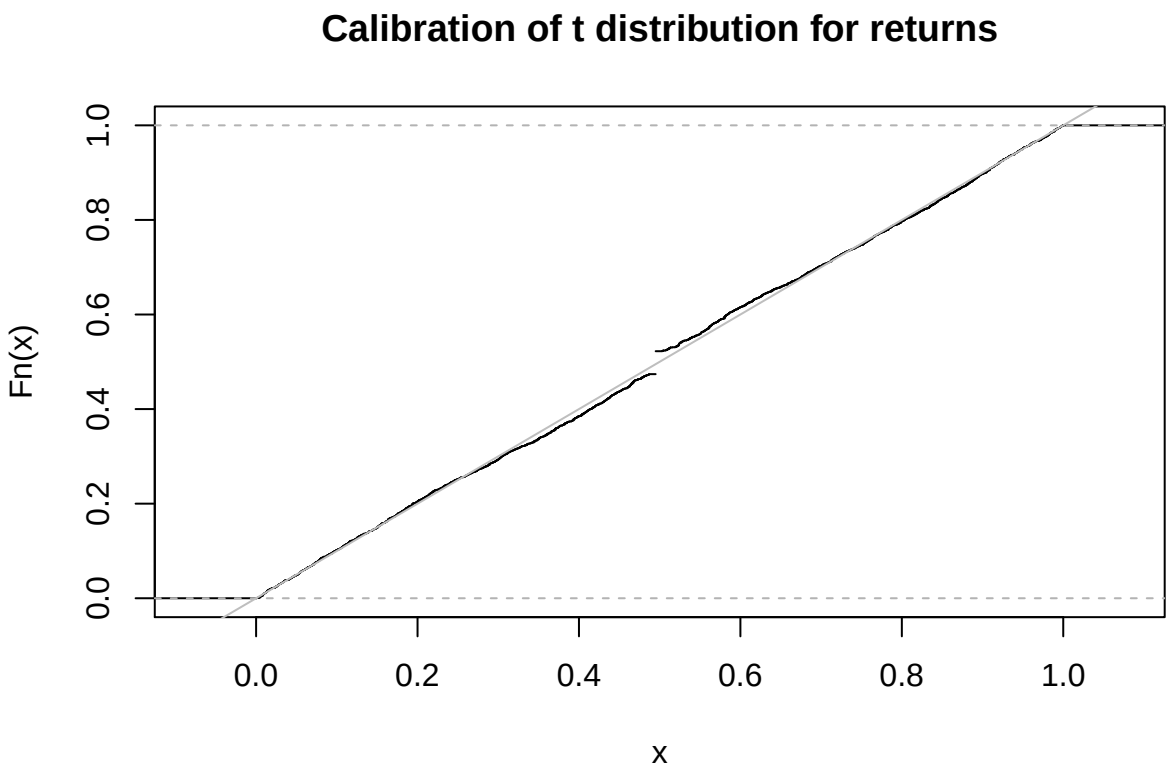
Again, way too many large changes (in either direction)

Calibration Plots (cont'd.)

```
CXW.t <- coefficients(fitdistr(returns,"t"))
```

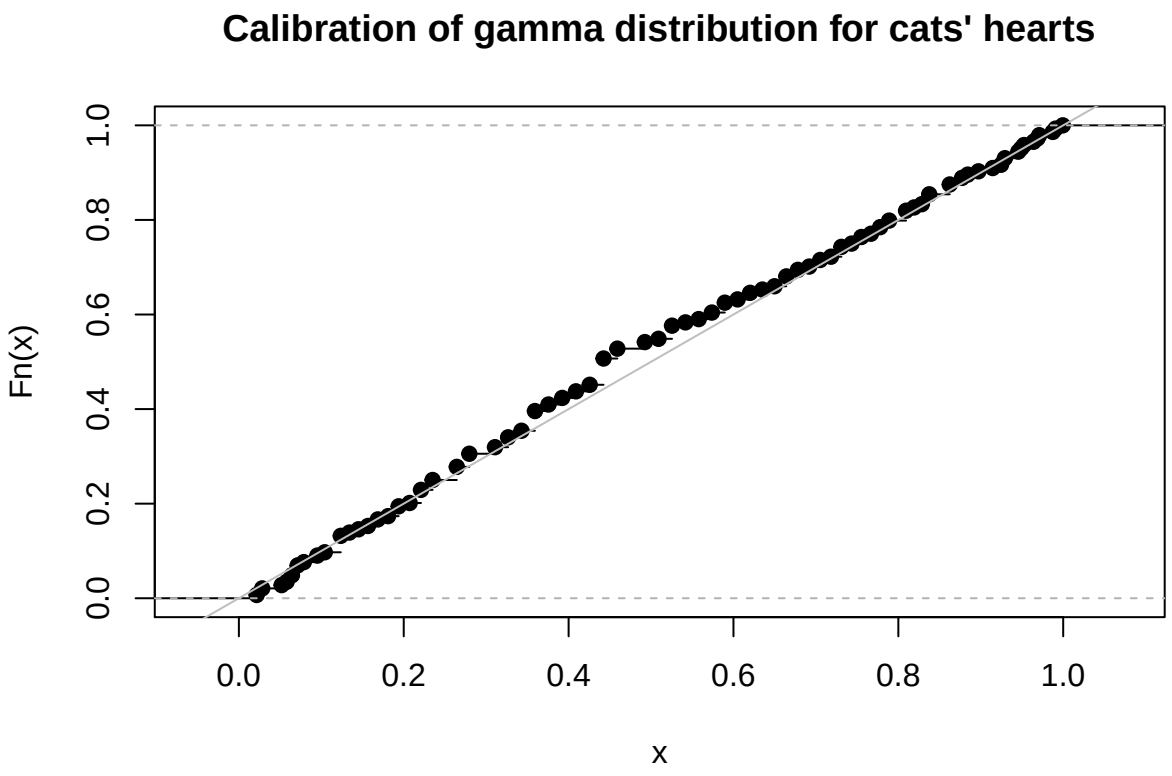
```
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
```

```
plot(ecdf(pt((returns-CXW.t[1])/CXW.t[2], df=CXW.t[3])),
     main="Calibration of t distribution for returns")
abline(0,1,col="grey")
```



Calibration Plots (cont'd.)

```
plot(ecdf(pgamma(cats$Hwt, shape=cats.gamma["shape"], scale=cats.gamma["scale"])),
     main="Calibration of gamma distribution for cats' hearts")
abline(0,1,col="grey")
```



Calibration Plots (cont'd.)

Challenge: Write a general function for making a calibraton plot, taking a data vector, a cumulative probability function, and a parameter vector

Kolmogorov-Smirnov Test

- How much should the QQ plot or calibration plot wiggle around the diagonal?
- Answer a different question...
- Biggest gap between theoretical and empirical CDF:

$$D_{KS} = \max_x \left| F(x) - \widehat{F}(x) \right|$$

- Useful because D_{KS} always has the same distribution *if* the theoretical CDF is fixed and correct, and K+S calculated this back in the day
- Also works for comparing the empirical CDFs of two samples, to see if they came from the same distribution

KS Test, Data vs. Theory

```
ks.test(returns,pnorm,mean=0,sd=0.0125)
```

```
## Warning: ties should not be present for the Kolmogorov-Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  returns
## D = 0.0849, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

- More complicated (and *not* properly handled by built-in R) if parameters are estimated
  - Estimating parameters makes the fit look *better* than it really is, so it doesn't help save the model when it gets really rejected (like this one is)

KS Test, Data vs. Theory (cont'd.)

Hack: Estimate using (say) 90% of the data, and then check the fit on the remaining 10%

```
train <- sample(1:length(returns),size=round(0.9*length(returns)))
CWX.t_train <- coefficients(fitdistr(returns[train],"t"))
```

```
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
## Warning: NaNs produced
```

```
returns.test_standardized <- (returns[-train]-CWX.t_train[1])/CWX.t_train[2]
ks.test(returns.test_standardized,pt,df=CWX.t_train[3])
```

```
## Warning: ties should not be present for the Kolmogorov-Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  returns.test_standardized
## D = 0.0566, p-value = 0.1861
## alternative hypothesis: two-sided
```

=== - Can also test whether two samples come from same distribution

```
n <- length(returns)
half <- round(n/2)
ks.test(returns[1:half], returns[(half+1):n])
```

```
## Warning: p-value will be approximate in the presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  returns[1:half] and returns[(half + 1):n]
## D = 0.0598, p-value = 0.002618
## alternative hypothesis: two-sided
```

Chi-Squared Test for Discrete Distributions

Compare an actual table of counts to a hypothesized probability distribution

e.g., as many up days as down?

```
up_or_down <- ifelse(returns > 0, 1, -1)
# 1936 down days, 1772 up days
chisq.test(table(up_or_down),p=c(1/2,1/2))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  table(up_or_down)
## X-squared = 7.423, df = 1, p-value = 0.006438
```

Chi-Squared Test: Degrees of Freedom

- The p-value calculated by `chisq.test` assumes that all the probabilities in p were *fixed*, not estimated from the data used for testing, so df = number of cells in the table -1
- If we estimate q parameters, we need to subtract q degrees of freedom

Chi-Squared Test for Continuous Distributions

- Divide the range into bins and count the number of observations in each bin; this will be `x` in `chisq.test()`
- Use the CDF function `p` *foo* to calculate the theoretical probability of each bin; this is `p`
- Plug in to `chisq.test`
- If parameters are estimated, adjust

Chi-Squared for Continuous Data (cont'd.)

- `hist()` gives us break points and counts:

```
cats.hist <- hist(cats$Hwt,plot=FALSE)
cats.hist$breaks
```

```
## [1]  6  8 10 12 14 16 18 20 22
```

```
cats.hist$counts
```

```
## [1] 20 45 42 23 11  2  0  1
```

Chi-Squared for Continuous Data (cont'd.)

Use these for a χ^2 test:

```
# Why the padding by -Inf and Inf?
p <- diff(pgamma(c(-Inf,cats.hist$breaks,Inf),shape=cats.gamma["shape"],
                 scale=cats.gamma["scale"]))
# Why the padding by 0 and 0?
x2 <- chisq.test(c(0,cats.hist$counts,0),p=p)$statistic
```

```
## Warning: Chi-squared approximation may be incorrect
```

```
# Why +2? Why -length(cats.gamma)?
pchisq(x2,df=length(cats.hist$counts)+2 - length(cats.gamma))
```

```
## X-squared
##    0.8546
```

Don't need to run `hist` first; can also use `cut` to discretize (see `?cut`)

Chi-Squared for Continuous Data (cont'd.)

- This is all a bit old-school
  - Loss of information from discretization
  - Lots of work just to use χ^2
- Try e.g. `ks.test` with an independent test set
- Or try a "smooth test" (take 36-402)

Misc. Tests for Misc. Distributions

- Too many to list

More Advanced and Formal Tests

- "bootstrap" testing (36-402)
- "smooth tests of goodness of fit" (also 36-402)

Summary

- Visualizing and computing empirical distribution
- Parametric distributions are models
- Methods of fitting: moments, generalized moments, likelihood
- Methods of checking: visual comparisons, other statistics, tests, calibration

Aside: Some Math for MM and GMM

- Parameter θ is a p-dimensional vector, true value $= \theta^*$
- Introduce $q \geq p$ **functionals** $g_1, \ldots g_q$, which we can calculate either from the parameter θ or from the data $x_{1:n}$
- *Assume* that for reach i, $g_i(x_{1:n}) \to g_i(\theta^*)$
- *Define*

$$\widehat{\theta}_{GMM} = \text{argmin}_\theta \sum_{i=1}^{q} (g_i(\theta) - g_i(x_{1:n}))^2$$

Math for MM and GMM (cont'd.)

- Shouldn't be hard to believe that $\widehat{\theta}_{GMM} \to \theta^*$
- But why give equal attention to every functional?
  - More weight on the more-precisely-measured functionals
  - More weight on the more-sensitive-to-θ functionals
  - Less weight on partially-redundant functionals
- *Abbreviate* $g(\theta)$ for $(g_1(\theta), \ldots g_q(\theta))$, and likewise $g(x_{1:n})$, so

$$\widehat{\theta}_{GMM} = \text{argmin}_\theta (g(\theta) - g(x_{1:n}))^T (g(\theta) - g(x_{1:n}))$$

- Generalize by introducing any positive-definite matrix Ω:

$$\widehat{\theta}_{GMM} = \text{argmin}_\theta (g(\theta) - g(x_{1:n}))^T \Omega (g(\theta) - g(x_{1:n}))$$

Math for MM and GMM (cont'd.)

- Optimal Ω turns out to be the variance matrix of $g(\theta^*)$
- Iterative approximation: start with no weighting, estimate that variance matrix, re-do the estimate with weights, etc.

Aside: Some Math for the MLE

- More convenient to work with the mean log likelihood:

$$\Lambda(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log f(X_i;\theta)$$

- This is a sample average so the law of large numbers applies:

$$\Lambda(\theta) \to \mathbf{E}[\Lambda(\theta)] = \lambda(\theta)$$

- The true parameter has higher average log-likelihood than anything else: if $\theta \neq \theta^*$

$$\theta \neq \theta^* \;\Rightarrow \lambda(\theta) < \lambda(\theta^*)$$

- Some extra conditions are needed for

$$\widehat{\theta}_{MLE} \to \theta^*$$