Distributions

James M. Flegal

Agenda

- Random number generation
- Distributions in R
- Distributional Models
- Moments, generalized moments, likelihood
- Visual comparisons and basic testing

Random number generation

- How does R get "random" numbers?
  - It doesn't, instead it uses pseudorandom numbers that we hope are indistinguishable from random numbers
  - Pseudorandom generators produce a deterministic sequence that is indistinguishable from a true random sequence if you don't know how it started
  - Why? Truly random numbers are expensive

Random number generation

- Pseudorandom generators produce a sequence of Uniform$(0, 1)$ random variates
- Linear congruential generator is one of the oldest and best-known pseudorandom number generator algorithms
- Other distributions usually based such a sequence
- Example: If $U \sim \text{Unif}(0, 1)$ then $-\beta \ln(1 - U) \sim \text{Exp}(\beta)$
- More on this later . . .

Example: `runif()`

```
runif(1:10)
```

```
##  [1] 0.44083639 0.64647956 0.67560056 0.49333730 0.61593355 0.92339429
##  [7] 0.45539437 0.17013010 0.09595561 0.84816697
```

```
set.seed(10)
runif(1:10)
```

```
##  [1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597 0.22543662
##  [7] 0.27453052 0.27230507 0.61582931 0.42967153
```

```
set.seed(10)
runif(1:10)
```

```
##  [1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597 0.22543662
##  [7] 0.27453052 0.27230507 0.61582931 0.42967153
```

Linear congruential generator

- Easily implemented and fast
- Modulo arithmetic via storage-bit truncation
- Defined as

$$X_{n+1} = (aX_n + c) \mod m$$

where X is the sequence of pseudorandom values

Linear congruential generator

```
seed <- 10
new.random <- function (a=5, c=12, m=16) {
out <- (a*seed + c) %% m
seed <<- out
return(out)
}
out.length <- 20
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random()
variates
```

```
##  [1] 14  2  6 10 14  2  6 10 14  2  6 10 14  2  6 10 14  2  6 10
```

- Unfortunately, this sequence has period 8

Linear congruential generator

```
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a=131, c=7, m=16)
variates
```

```
##  [1]  5  6  9  2 13 14  1 10  5  6  9  2 13 14  1 10  5  6  9  2
```

```
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a=129, c=7, m=16)
variates
```

```
##  [1]  9  0  7 14  5 12  3 10  1  8 15  6 13  4 11  2  9  0  7 14
```

Linear congruential generator

```
variates <- rep (NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a=1664545, c=1013904223, m=2^32)
variates
```

```
##  [1] 1037207853 2090831916 4106096907  768378826 3835752553 1329121000
##  [7] 2125006663 2668506502 3581687205 2079234980 2067291011 2197025090
## [13] 3748878561 2913996384  758844863 4029469438 2836748829 1458315036
## [19] 2399149563 2766656186
```

How to distinguish non-randomness?

- Look at period
- Missing some values
- Proper distribution in the limit
- Autocorrelation
- Gets harder for higher dimensions

Distributions

- Beta, Binomial, Cauchy, Chi-Square, Exponential, F, Gamma, Geometric, Hypergeometric, Logistic, Log Normal, Negative Binomial, Normal, Poisson, Student t, Studentized Range, Uniform, Weibull, Wilcoxon Rank Sum Statistic, Wilcoxon Signed Rank Statistic
- Parameters of these distributions may not agree with textbooks
- Lots of distributions, but they all work the same way

Distributions

Every distribution that R handles has four functions. There is a root name, for example, the root name for the normal distribution is `norm`. This root is prefixed by one of the letters

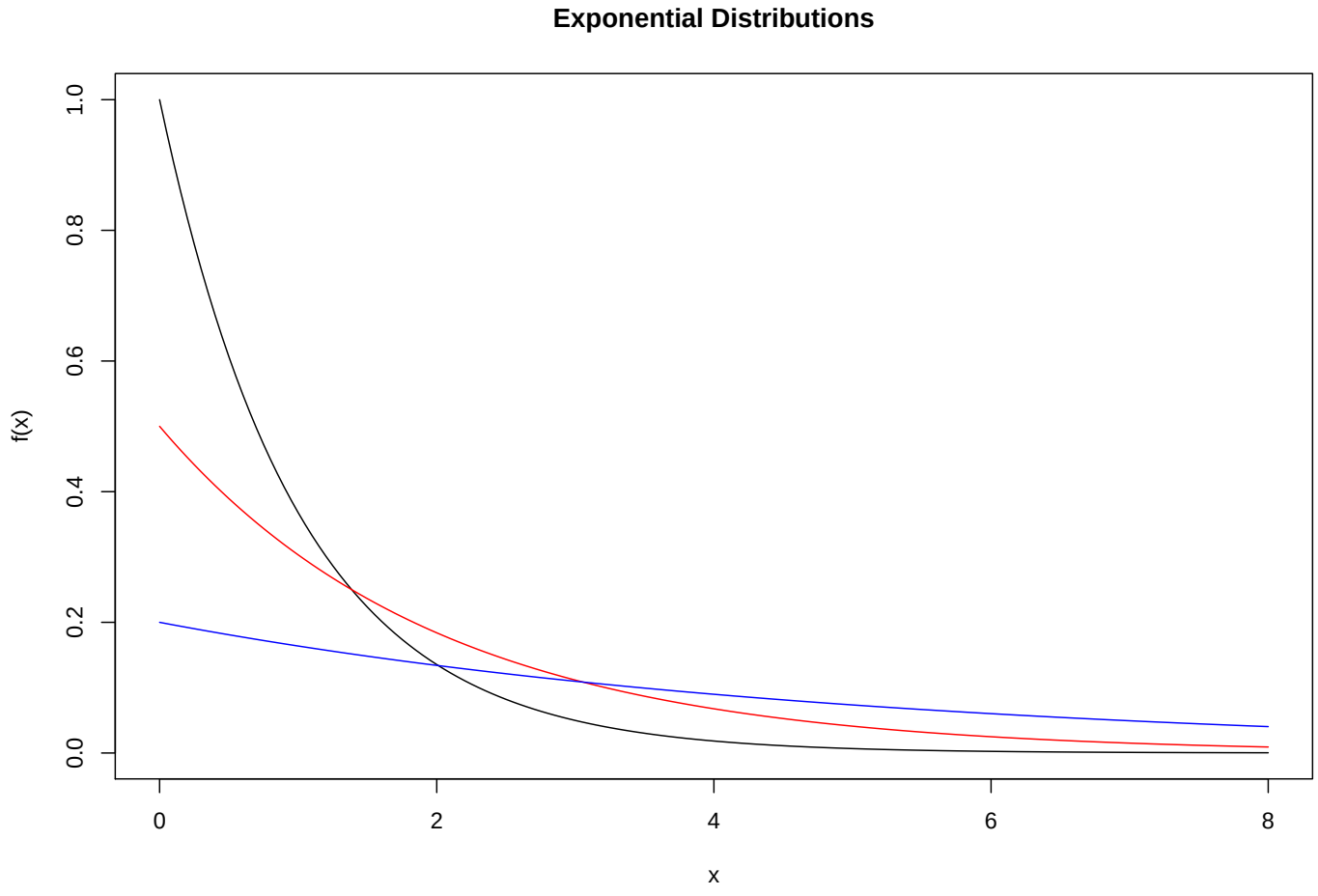
- `p` for "probability", the cumulative distribution function (c. d. f.)
- `q` for "quantile", the inverse c. d. f.
- `d` for "density", the density function (p. f. or p. d. f.)
- `r` for "random", a random variable having the specified distribution

Distributions

- Beta, Binomial, Cauchy, Chi-Square, Exponential, F, Gamma, Geometric, Hypergeometric, Logistic, Log Normal, Negative Binomial, Normal, Poisson, Student t, Studentized Range, Uniform, Weibull, Wilcoxon Rank Sum Statistic, Wilcoxon Signed Rank Statistic
- `beta`, `binom`, `cauchy`, `chisq`, `exp`, `f`, `gamma`, `geom`, `hyper`, `logis`, `lnorm`, `nbinom`, `norm`, `pois`, `t`, `tukey`, `unif`, `weibull`, `wilcox`, `signrank`
- Exponential distribution has `pexp`, `qexp`, `dexp`, and `rexp`

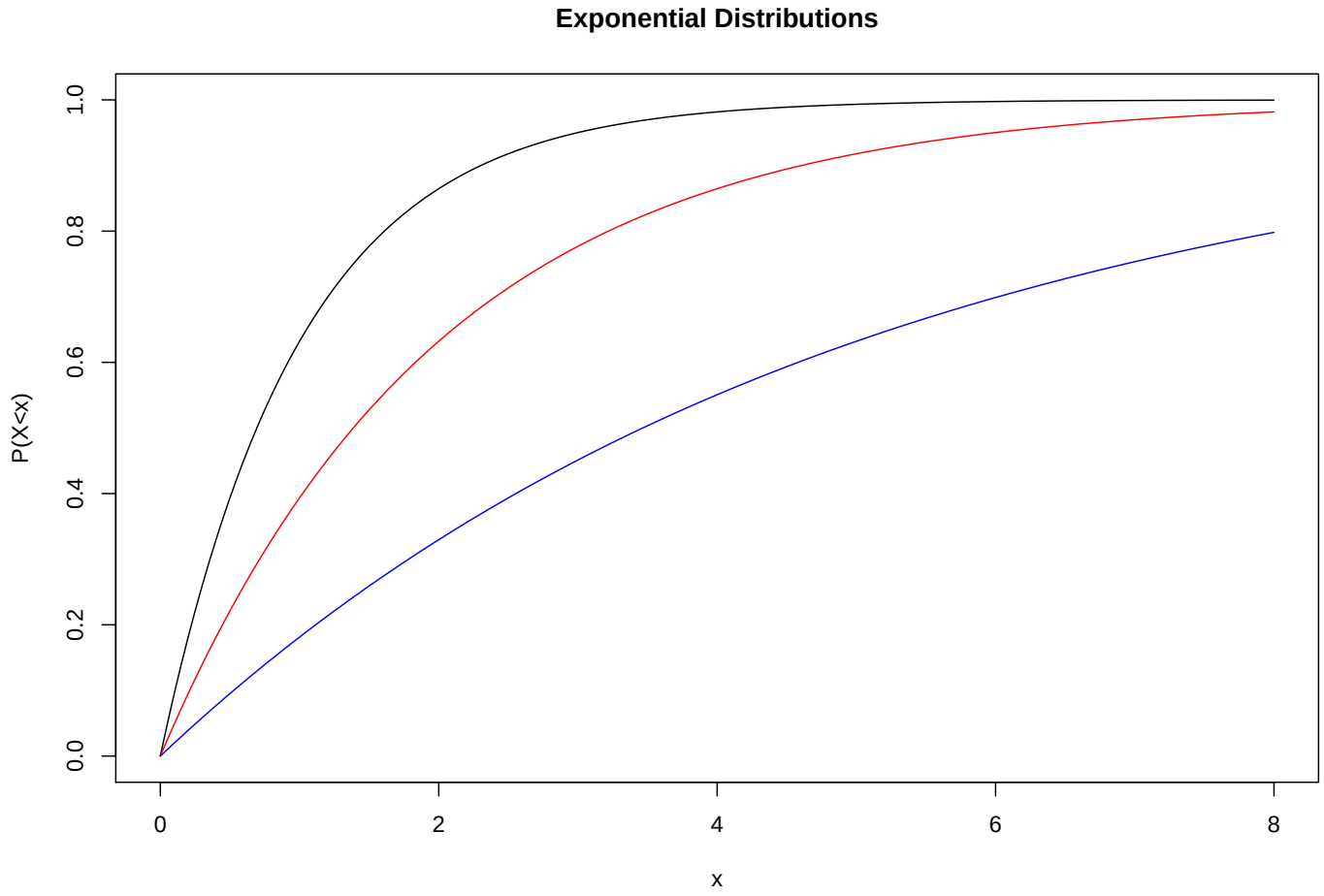
Exponential distribution

```
this.range <- seq(0, 8, .05)
plot (this.range, dexp(this.range), ty="l", main="Exponential Distributions",
      xlab="x", ylab="f(x)")
lines (this.range, dexp(this.range, rate=0.5), col="red")
lines (this.range, dexp(this.range, rate=0.2), col="blue")
```



Exponential distribution

```
this.range <- seq(0, 8, .05)
plot (this.range, pexp(this.range), ty="l", main="Exponential Distributions",
      xlab="x", ylab="P(X<x)")
lines (this.range, pexp(this.range, rate=0.5), col="red")
lines (this.range, pexp(this.range, rate=0.2), col="blue")
```



Fitting distributional models

- Method of moments
- Match other summary statistics
- Maximize likelihood estimation

Method of moments: Closed form

- Pick enough moments to identify the parameters, i.e. at least 1 moment per parameter
- Write moment equations in terms of the parameters, e.g. the gamma distribution

$$\mu = as \text{ and } \sigma^2 = as^2$$

- Solve the system of equations (by hand)

$$a = \mu^2/\sigma^2 \text{ and } s = \sigma^2/\mu$$

- Write a function

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

Method of moments: 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 your function

Applies more generally

- Can match other data summaries, e.g. the median or ratios of quantiles
- If you can't solve exactly, set up a discrepancy function and minimize it

Maximum Likeihood

- Usually think of parameters as fixed and consider the probability of different outcomes, $f(x|\theta)$ with θ constant and x changing
- Likelihood of a parameter value, i.e. $L(\theta)$ treats this a function of θ
- Search over values of θ for the one that makes the data most likely
- Results in maximum likelihood estimate (MLE)

Maximum Likeihood

- Suppose $x_1, \ldots, x_n$ are i.i.d., then

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

- Tends to be computationally unstable since it may contain lots of small numbers
- Instead consider the log likelihood

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

- In pseudo-code

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

What to do with the likelihood?

- Maximize it!
- Sometimes we can do the maximization by hand via calculus
  - Gaussian, Poisson, Pareto, ...
- More generally we'll consider numerical optimization
  - Include a minus sign when using a minimization function

Why use an MLE?

- Usually consistent: converges on the truth as we get more data
- Usually efficient: converges on the truth at least as fast as anything else
- Some settings where the maximum isn't well-defined (e.g. x_{min} for a Pareto)
- Sometimes data is too aggregated or mangled to use an MLE

MLEs for univariate distributions

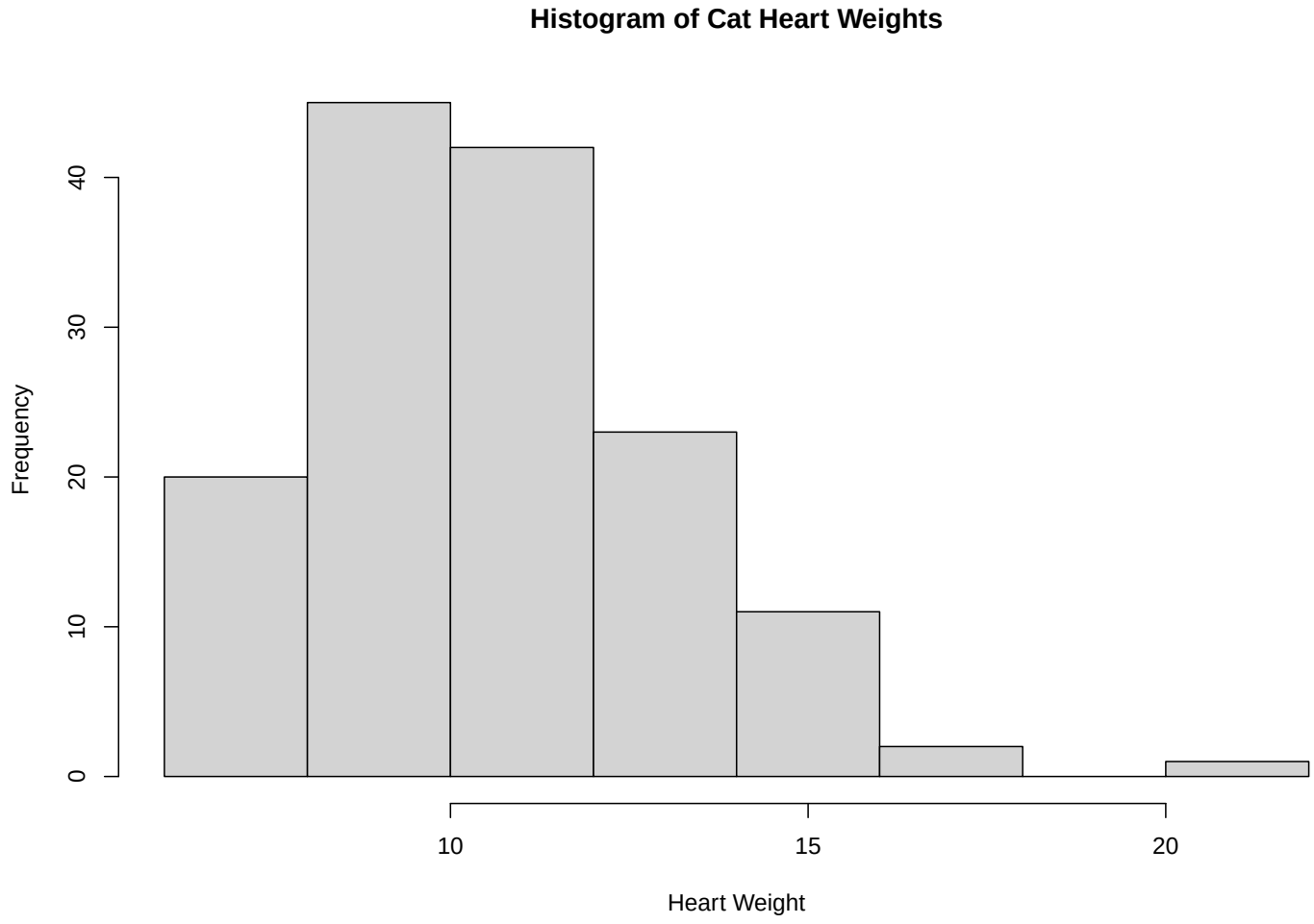
- `fitdistr` in `MASS` package
- Knows most standard distributions, but can also handle arbitrary probability density functions
- Starting value for the optimization is optional for some distributions, required for others (including user-defined densities)
- Returns parameter estimates and standard errors from large-n approximations (so use cautiously)

Example: Cat heart weights

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.2
```

```
data("cats",package="MASS")
hist(cats$Hwt, xlab="Heart Weight", main="Histogram of Cat Heart Weights")
```



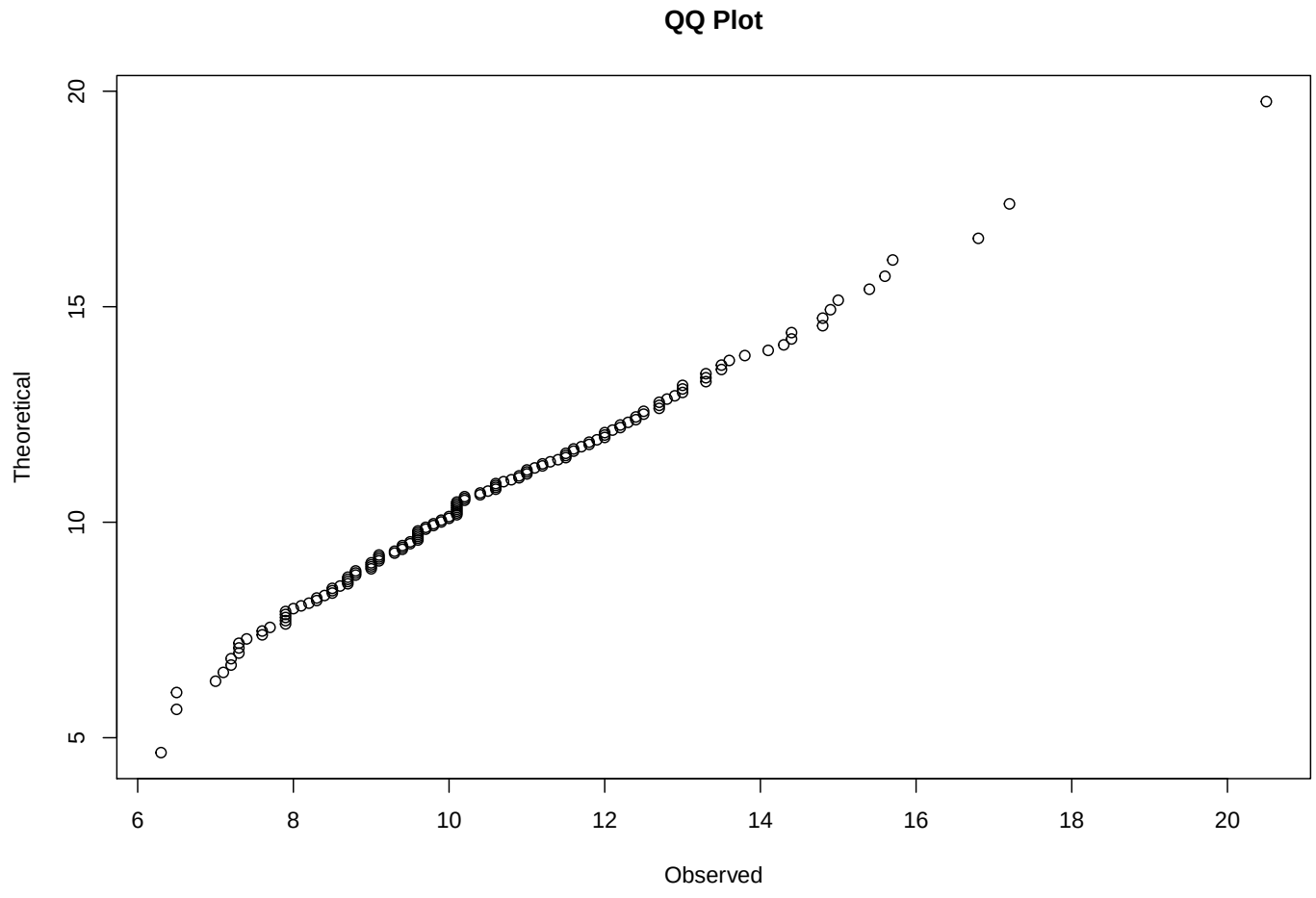
Example: Cat heart weights

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

```
##      shape         rate
##   20.2998092    1.9095724
##  ( 2.3729250) ( 0.2259942)
```

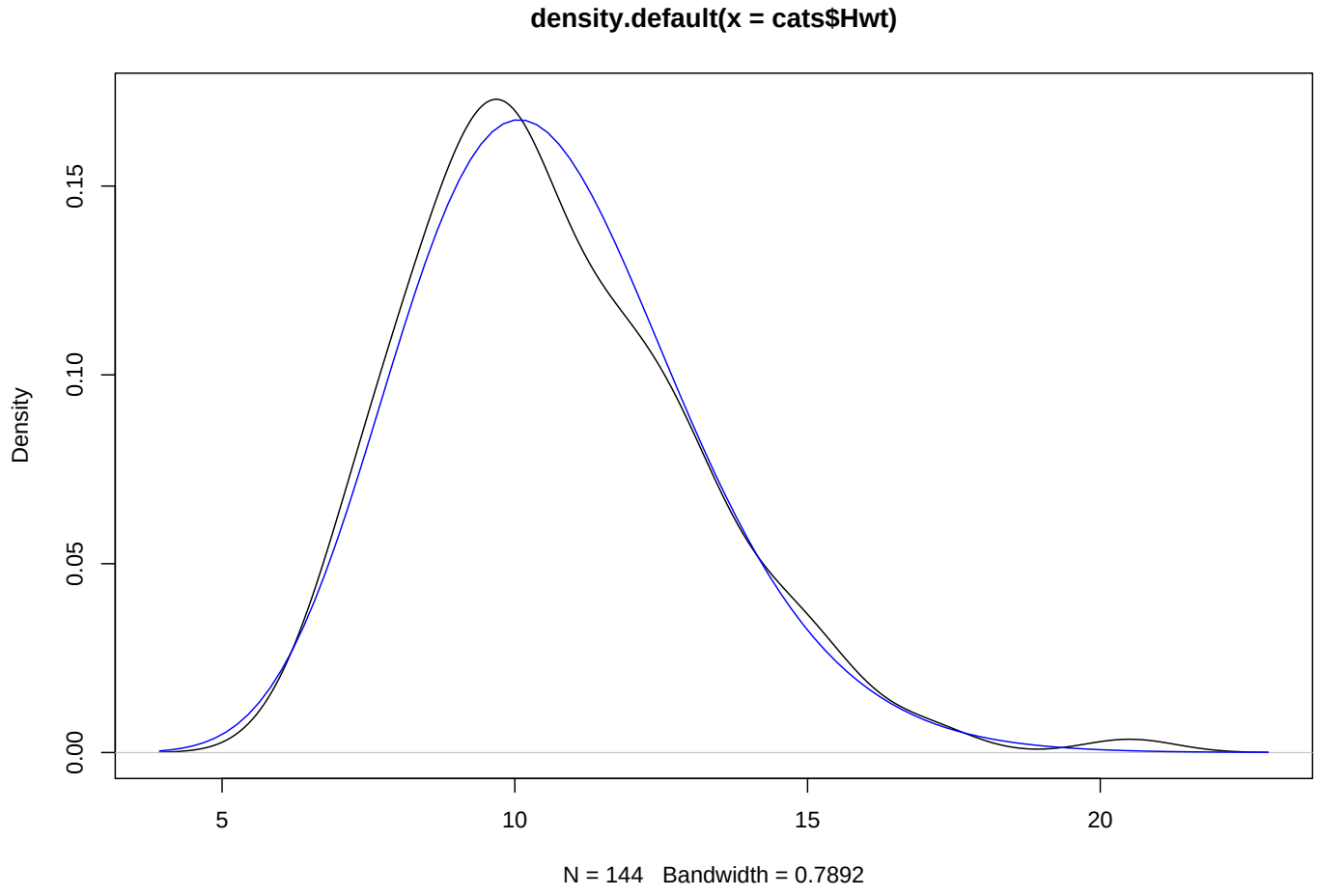
Checking the fit

```
cats.gamma <- gamma.est_MM(cats$Hwt)
qqplot(cats$Hwt, qgamma(ppoints(500), shape=cats.gamma["shape"],
      scale=cats.gamma["scale"]), xlab="Observed", ylab="Theoretical",
      main="QQ Plot")
```



Checking the fit

```
plot(density(cats$Hwt)) # more on this later
curve(dgamma(x,shape=cats.gamma["shape"],scale=cats.gamma["scale"]),add=TRUE,col="blue")
```

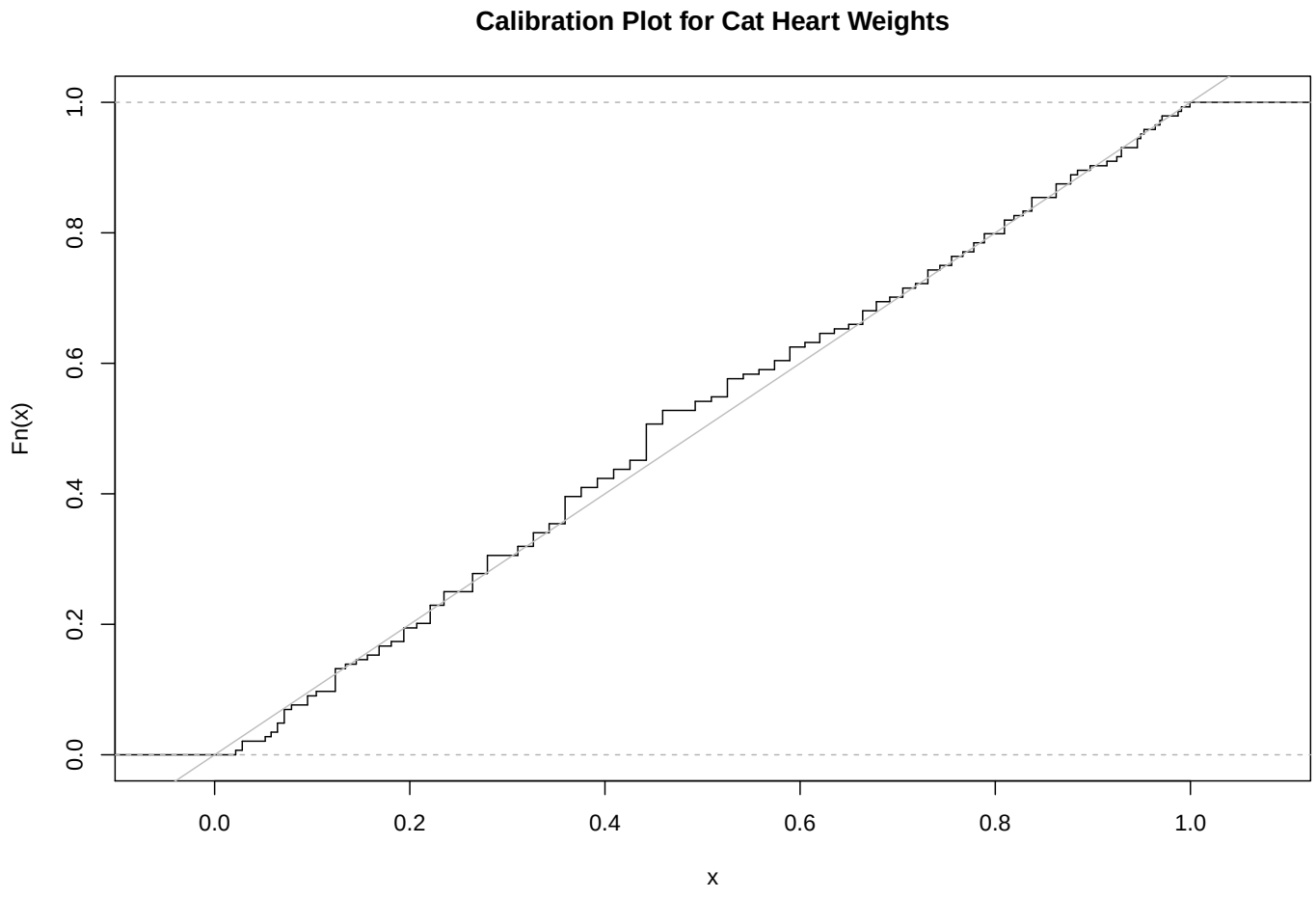


Calibration plots

- 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
- Can look at calibration 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

Calibration plots

```
plot(ecdf(pgamma(cats$Hwt, shape=cats.gamma["shape"], scale=cats.gamma["scale"])),
     main="Calibration Plot for Cat Heart Weights", verticals = T, pch = "")
abline(0,1,col="grey")
```



Calibration plots

Exercise: Write a general function making a calibration plot that inputs a data vector, cumulative probability function, and parameter vector

Kolmogorov-Smirnov test

- How much "error" is acceptable in a QQ plot or calibration plot?
- Alternatively, consider the biggest gap between theoretical and empirical CDF:

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

- Useful because D_{KS} has the same distribution if the theoretical CDF is fixed and correct
- Also works for comparing the empirical CDFs of two samples, to see if they came from the same distribution

Kolmogorov-Smirnov test

```
ks.test(cats$Hwt, pgamma, shape=cats.gamma["shape"], scale=cats.gamma["scale"])
```

```
## Warning in ks.test(cats$Hwt, pgamma, shape = cats.gamma["shape"], scale =
## cats.gamma["scale"]): ties should not be present for the Kolmogorov-Smirnov test

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  cats$Hwt
## D = 0.068637, p-value = 0.5062
## alternative hypothesis: two-sided
```

- Caution: Solution is 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
- Could estimate using (say) 80% of the data, and then check the fit on the remaining 20%

Kolmogorov-Smirnov test

- Can also test whether two samples come from same distribution

```
x <- rnorm(100, mean=0, sd=1)
y <- rnorm(100, mean=.5, sd=1)
ks.test(x, y)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x and y
## D = 0.28, p-value = 0.0007873
## alternative hypothesis: two-sided
```

Chi-squared test

- Compare an actual table of counts to a hypothesized probability distribution

```
M <- as.table(rbind(c(762, 327, 468), c(484, 239, 477)))
dimnames(M) <- list(gender = c("F", "M"), party = c("Democrat","Independent", "Republican"))
M
```

```
##        party
## gender Democrat Independent Republican
##      F      762         327        468
##      M      484         239        477
```

```
Xsq <- chisq.test(M)
Xsq
```

```
##
##  Pearson's Chi-squared test
##
## data:  M
## X-squared = 30.07, df = 2, p-value = 2.954e-07
```

Chi-squared test

- 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 test for continuous distributions

- `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
```

- Use these for a χ^2 test

Chi-squared test for continuous distributions

```
# 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 in chisq.test(c(0, cats.hist$counts, 0), p = p): 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
##  0.854616
```

- Don't need to run `hist` first; can also use `cut` to discretize

Chi-squared test for continuous distributions

- Better ways to do this
  - Loss of information from discretization
  - Lots of work just to use `chisq.test()`
- Try `ks.test`, "bootstrap" testing, "smooth tests of goodness of fit", ...

Summary

- Random number generation
- Distributions in R
- Parametric distributions are models
- Methods of fitting; moments, generalized moments, likelihood
- Methods of checking; visual comparisons, other statistics, tests, calibration