Statistics 705 Notes 4

Vince Lyzinski

Acknowledgement

Thank you to Eric Slud (UMD) and Minh Tang (NCSU) for their course notes, which I used (and borrowed from) when building these slides!

Random number generation and univariate distributions

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. — John von Neumann (1951)

Anyone who has not seen the above quotation in at least 100 places is probably not very old. — D. V. Pryor (1993)

Random number generators should not be chosen at random. — Donald Knuth (1986)

Background reading

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole. –reference for **runif( )** function in R

There are a plethora of references listed if you type **help(".Random.seed")** into R

True randomness?

In R, the Random Number Generators are only pseudorandom—the numbers aren't random, but should be indistinguishable from truly random numbers

True sources of randomness include

- Atmospheric noise (random.org)
- Thermal noise
- In general, measuring a physical process that we think to be random (and then adjusting for the error of measurement)

What does R do?

Pseudorandom number generation

- Make a deterministic sequence that should be indistinguishable from truly random numbers if we don't know the starting point of the sequence
- How?

Example using runif( )

```
runif(5)
```

```
## [1] 0.8754247 0.1411678 0.1758072 0.9235542 0.1449683
```

```
set.seed(1)
runif(5)
```

```
## [1] 0.2655087 0.3721239 0.5728534 0.9082078 0.2016819
```

```
set.seed(1)
runif(5)
```

```
## [1] 0.2655087 0.3721239 0.5728534 0.9082078 0.2016819
```

LCGs

The simplest and most common pseudorandom number generators: *Linear-Congruential Generators* (**LCG's**):

$$x_{n+1} = a \cdot x_n + b \mod m$$

- a is called the *multiplier*
- b the *addend*
- m the *modulus*, usually close to a word-size, e.g. 2^{32} or $2^{31} - 1$

With carefully chosen a, the period (length of the cycle of the LCG) will be

- m if m is a power of 2
- $m - 1$ if m is prime

LCGs

```
LCG <- function (a, b, m, x0, n) {
  x <- numeric(n)
  # x %% y gives x (mod y)
  for(i in 1:n){
  x[i] <- (a*x0 + b) %% m
  x0 <- x[i]}
  return(x)
}
```

Divide by m if we want Unif[0,1]

LCGs

A good triplet according to *S. K. Park and K. W. Miller, "Random Number Generators: Good Ones are Hard to Find," Transactions of the ACM, Nov. 1988* is

$$m = 2^{31} - 1; \;\; a = 7^5; \;\; b = 0$$

```
# generate 10 from this LCG
LCG(7^5,0,(2^(31)-1),x0=1,10)
```

```
##  [1]       16807  282475249 1622650073  984943658 1144108930
##  [7] 101027544 1457850878 1458777923 2007237709
```

LCGs

There are *famously* bad choices

Ex:

$$a = 65539;\ \ m = 2^{31};\ \ b = 0;\ \ x_0 \text{ odd}$$

Knuth called this "truly horrible!"

- "its very name RANDU is enough to bring dismay into the eyes and stomachs of many computer scientists!" Knuth in *Art of Computer Programming, Vol. 2*

What could go wrong?

Too short of a period

Ex: $a = 13$; $b = 0$; $m = 16$

```
LCG(13,0,16,1,9)
```

```
## [1] 13  9  5  1 13  9  5  1 13
```

period is 4!

What could go wrong?

Ex: $a = 17;\ b = 7;\ m = 16$

```
LCG(17,7,16,1,18)
```

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

period is 16!

What could go wrong?

Full period obtained when $p = m$

By the Hull-Dobell Theorem, this occurs iff

- $gcd(b, m) = 1$
- If $q|m$ and q is prime, then $q|(a - 1)$
- If $4|m$, then $4|(a - 1)$

Back to our ex:

$a = 17;\ b = 7;\ m = 16$

Full period!

What could go wrong?

Ex: *Numerical Recipes* uses

$$m = 2^{32}; \quad a = 1664525; \quad b = 1013904223$$

- gcd(b,m)=1 as b is odd
- 2 is only prime that divides m, and $2|1664524$
- $4|m$ and $\frac{1664524}{4} = 416131$

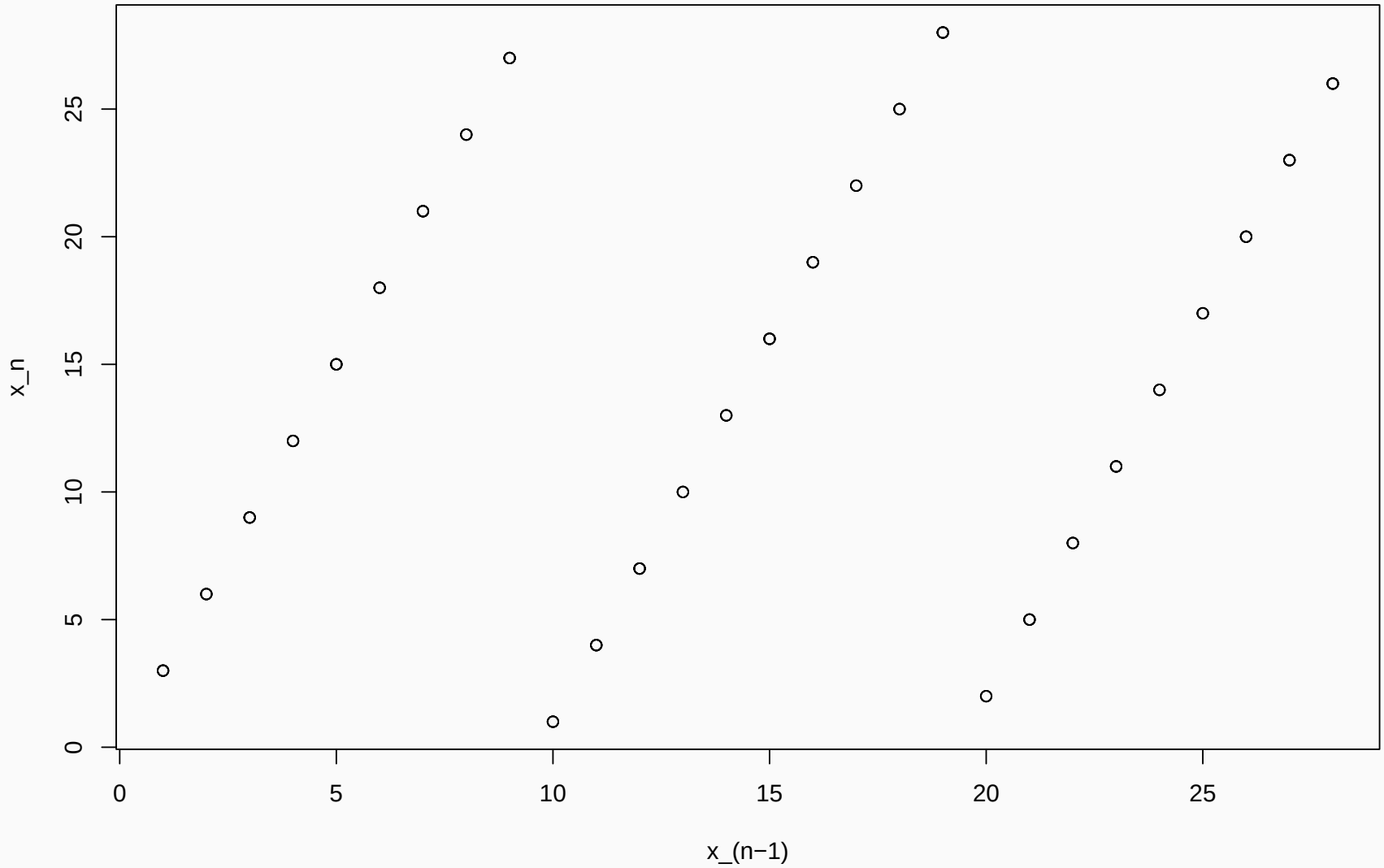
What could go wrong?

Other main issue is a number-theoretic property spotted by G. Marsaglia in a famous article ("Random numbers fall mainly in planes", 1968 Proc. Nat. Acad. Sci.):

- LCGs result in sequences which fall along hyperplanes in some number of dimensions; at most (but sometimes much less than) $\sqrt{m}$.
- There are several tests of randomness (mentioned e.g. by Knuth) which test how finely spaced these hyperplanes are (e.g., "spectral test" of Coveyou - Macpherson)
- Idea: further apart planes are, the worse the LCG
- RANDU fails the spectral test in dimension 3

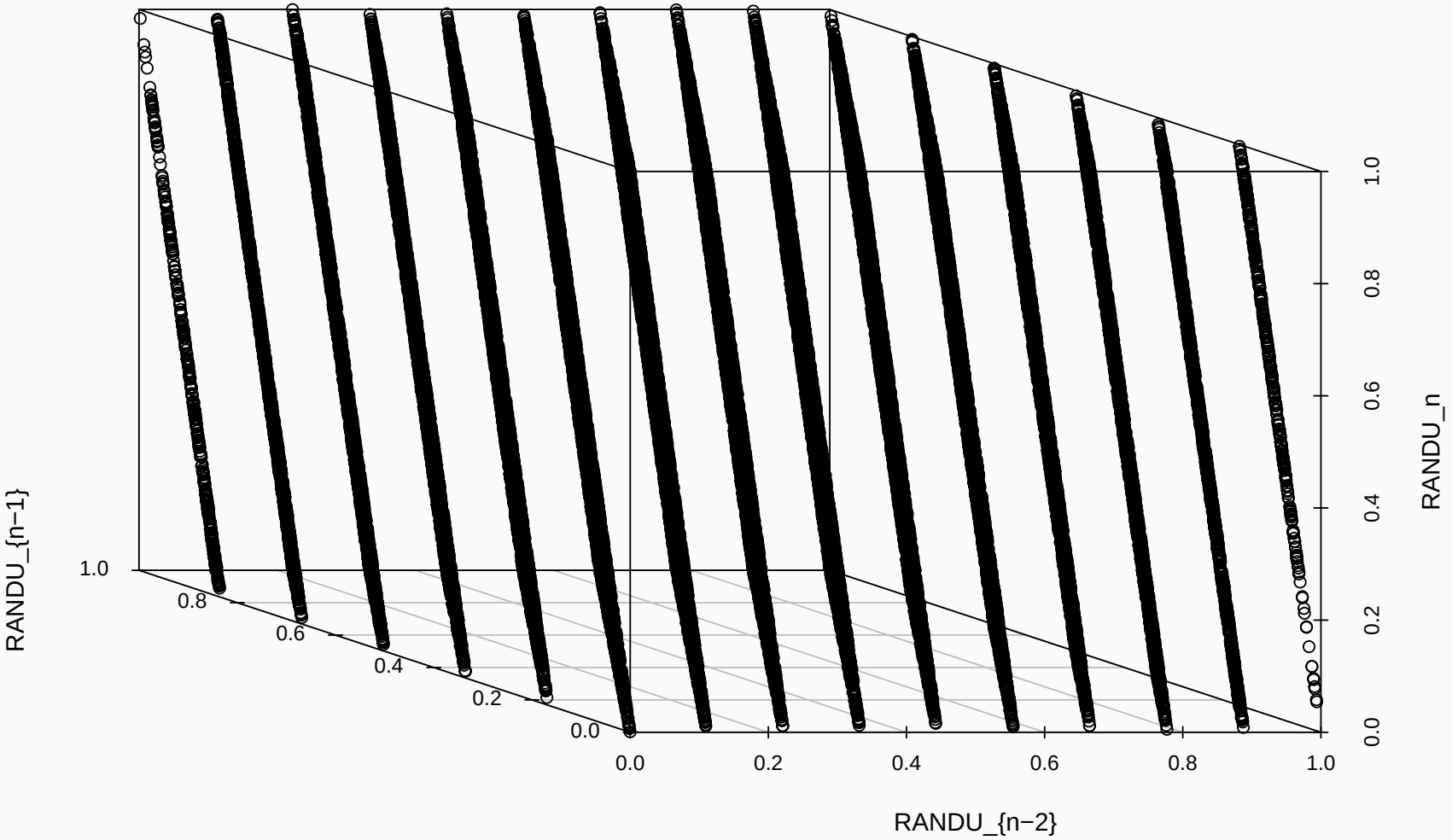
What could go wrong?

Ex: $a = 3,\ b = 0,\ m = 29$



What could go wrong?

Ex: RANDU



Chi-squared testing for randomness

To test the randomness of a given sequence, we can use the χ^2 goodness of fit test for the mulitnomial distribution

Used to assess the equidistribution of the non-overlapping K-tuples

$$(x_n, \ldots, x_{n+K-1}), \quad n = 0, K, 2K, \ldots$$

by tabulating the counts of these K-tuples falling in sets $A \subset [0,1]^K$ of the form

$$\prod_{i=1}^{K} [a_i/L, (a_i+1)/L)$$

for $a_i \in \{0, 1, \ldots, L-1\}$

Chi-squared testing for randomness

Idea: Divide the unit K-cube into L^K equal sized bins, let p_i denote the probability of an observation falling into bin i

Test

$$H_0 : p_i = \frac{1}{L^K} \text{ for all i; } H_a : p_i \neq \frac{1}{L^K} \text{ for some i}$$

Use Pearson's χ^2 test:

$$\text{under } H_0, \ T = \sum_i \frac{(n_i - np_i)^2}{np_i} \sim (approx.)\chi^2_{L^K - 1}$$

Chi-squared testing for randomness

A quick R code is given by (*ncoord* corresponds to K and *nquant* corresponds to L)

```
FitNtupl<- function(ncoord, nquant, indata)
{
  ## assumes block of pseudo-random uniform[0,1)
  ## numbers in  indata; to be tested for fit based on
  ## empirically generated contingency-table of nquant
  ## equal-length intervals in each of ncoord
  ## consecutive coordinates
  ntup = length(indata) %/% ncoord
  idata = c(matrix(trunc(nquant * indata - 1e-11),
             ncol = ncoord) %*% nquant^(0:(ncoord - 1))) + 1
  cellexp = ntup/(nquant^ncoord)
  cells = table(idata)
```

Chi-squared testing for randomness

code con...

```
diagind = 1 + (0:(nquant - 1)) * sum(nquant^(0:(ncoord - 1)))
# overall balance
chistat = sum((cells - cellexp)^2)/cellexp
# fraction falling in diagonal cells
# test for coord agreement
diagstat = (sum(cells[diagind]) - nquant *
              cellexp)^2/(nquant *cellexp *
              (1 - ((cellexp * nquant)/ntup)))
list(chisq = chistat, pval = 1 - pchisq(chistat,
       nquant^ncoord - 1), diagstat = diagstat,
       diagPval = 1-pchisq(diagstat, 1), CountTbl = cells)
}
```

Chi-squared testing for randomness

```
set.seed(123)
x <- FitNtupl(2,4,runif(1.e4))
x$chisq
```

```
## [1] 14.4576
```

```
x$pval
```

```
## [1] 0.4911446
```

```
x$diagstat
```

```
## [1] 0.05226667
```

```
x$diagPval
```

```
## [1] 0.819165
```

Chi-squared testing for randomness

Try RANDU

```
x <- LCG(65539,0,2^(31),1,2e5)
x <- x/(2^(31))
unlist(FitNtupl(2,10,x))[1:4]
```

```
##       chisq        pval    diagstat    diagPval
## 101.1480000   0.4211794   0.2250000   0.6352563
```

No wonder it was used so much!

Fails more complex tests (ex: spectral test)

Random numbers/distributions in R

For a variety of distributions, R has built in pseudogenerators for

- **r**: generate RV from the given distribution
- **d**: density functions (for continuous distributions) or probability mass functions (for discrete distributions)
- **p**: cumulative distribution function $F(x) = \mathbb{P}(X \leq x)$
- **q**: quartile function, $Q(u) = F^{-1}(u)$ (For discrete distributions the quantile is the smallest integer m such that $F(m) \geq u$.)

Ex: For the normal distribution, calls are **rnorm**, **dnorm**, **pnorm**, **qnorm**

Random numbers/distributions in R

```
rnorm(5, mean=0,sd=1)
```

```
## [1] -0.4941739  1.1275935 -1.1469495  1.4810186  0.9161912
```

```
dnorm(1, mean=0,sd=1)
```

```
## [1] 0.2419707
```

```
pnorm(1, mean=0,sd=1)
```

```
## [1] 0.8413447
```

```
qnorm(0.84, mean=0,sd=1)
```

```
## [1] 0.9944579
```

```
qnorm(pnorm(1, mean=0,sd=1), mean=0,sd=1)
```

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

Random numbers/distributions in R

Table 5.1: S function names and parameters for standard probability distributions.

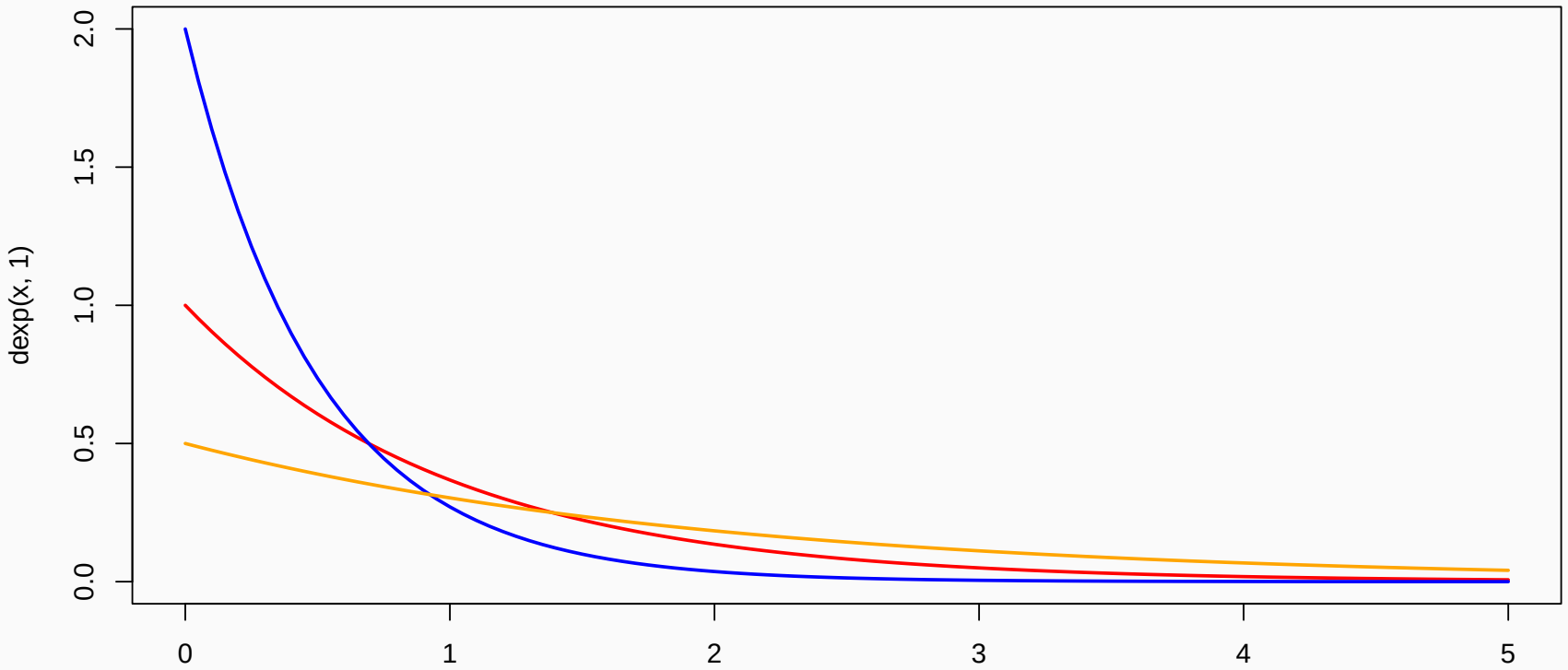
Distribution	S name	Parameters
beta	`beta`	`shape1, shape2`
binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
chi-squared	`chisq`	`df`
exponential	`exp`	`rate`
F	`f`	`df1, df2`
gamma	`gamma`	`shape, rate`
geometric	`geom`	`prob`
hypergeometric	`hyper`	`m, n, k`
log-normal	`lnorm`	`meanlog, sdlog`
logistic	`logis`	`location, scale`
negative binomial	`nbinom`	`size, prob`
normal	`norm`	`mean, sd`
Poisson	`pois`	`lambda`
T	`t`	`df`
uniform	`unif`	`min, max`
Weibull	`weibull`	`shape, scale`
Wilcoxon	`wilcox`	`m, n`

Example: Exponential distribution

```r
dexp(0:4,1) # rate * e^{-rate* x}
```

```
## [1] 1.00000000 0.36787944 0.13533528 0.04978707 0.01831564
```

```r
x <- (0:100)/20
plot(x,dexp(x,1),lwd=2,col="red",type='l',ylim=c(0,2))
points(x,dexp(x,2),lwd=2,col="blue",type='l')
points(x,dexp(x,0.5),lwd=2,col="orange",type='l')
```

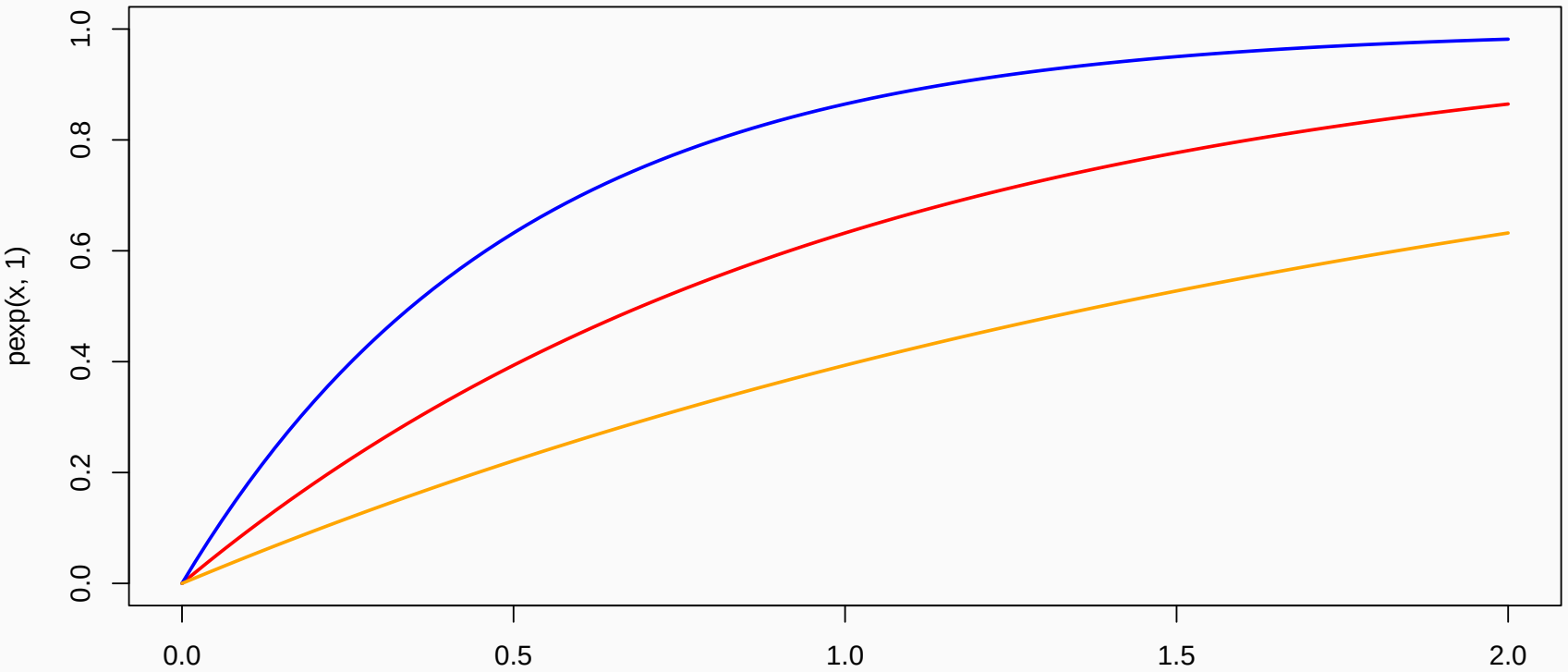


Example: Exponential distribution

```
pexp(0:4,1) # 1 - e^{-rate* x}
```

```
## [1] 0.0000000 0.6321206 0.8646647 0.9502129 0.9816844
```

```
x <- (0:100)/50
plot(x,pexp(x,1),lwd=2,col="red",type='l',ylim=c(0,1))
points(x,pexp(x,2),lwd=2,col="blue",type='l')
points(x,pexp(x,0.5),lwd=2,col="orange",type='l')
```



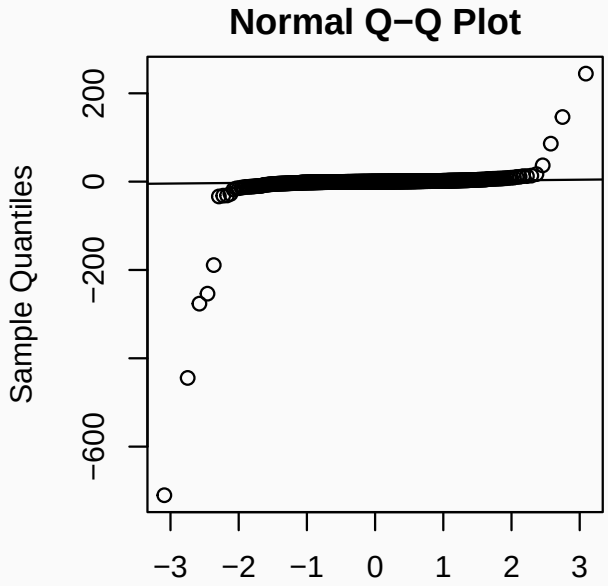
Random numbers/distributions in R

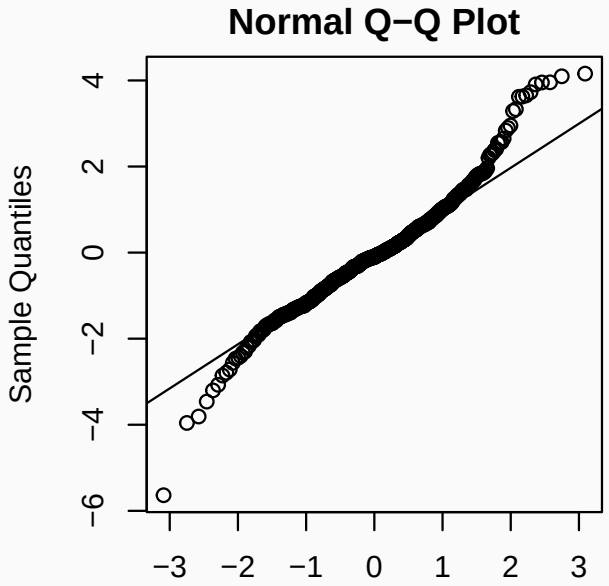
Ex: Q-Q plots

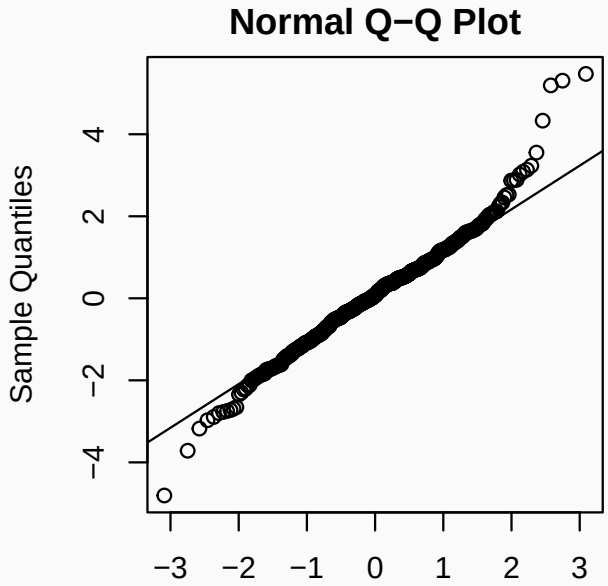
```
par(mfrow=c(2,2),mar=c(2, 2, 2, 2))
dof <- c(1,5,10,100)
for(i in 1:4){
x <- rt(250, df = dof[i])
par(pty = "s")
qqnorm(x); qqline(x)}
```

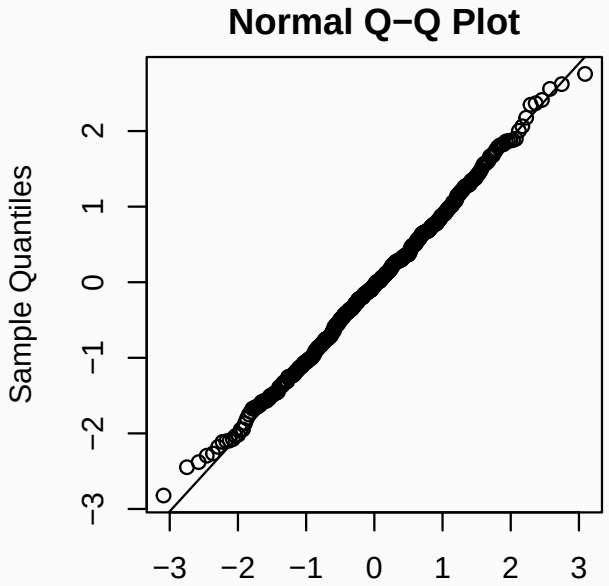
The greater spread of the extreme quantiles for the data is indicative of a long- tailed distribution.

Random numbers/distributions in R









Random numbers/distributions in R

Ex: MLE (more on this later!)

```
x <- rnorm(300,0,3)
nll <- function(m,s) -sum(dnorm(x, mean = m, sd = s, log = TRUE))
library(MASS)
fitdistr(x, "normal")
```

```
##       mean          sd
##    0.1793423   2.8662987
##  (0.1654858) (0.1170162)
```

Random numbers/distributions in R

Ex: MLE

```
library(stats4)
M1 <- mle(nll, start=list(m=-5, s=1), method="L-BFGS-B")
M2 <- mle(nll, start=list(m=-5, s=1), method="CG")
summary(M1)
```

```
## Maximum likelihood estimation
##
## Call:
## mle(minuslogl = nll, start = list(m = -5, s = 1), method = "L
##
## Coefficients:
##    Estimate Std. Error
## m 0.1793422  0.1654859
## s 2.8662992  0.1170161
##
## -2 log L: 1483.176
```

Random numbers/distributions in R

Ex: MLE

```
summary(M2)
```

```
## Warning in sqrt(diag(object@vcov)): NaNs produced
```

```
## Maximum likelihood estimation
##
## Call:
## mle(minuslogl = nll, start = list(m = -5, s = 1), method = "C
##
## Coefficients:
##   Estimate Std. Error
## m 143.9835   54.15994
## s 979.6245        NaN
##
## -2 log L: 4689.765
```

Generating random data

Some important commands are

- **sample(n)** select a random permutation of $1, \cdots, n$
- **sample(x)** randomly permute the elements of x
- **sample(x, replace = T)** generate a bootstrap sample of x
- **sample(x, n)** sample n items from x without replacement
- **sample(x, n, replace = T)** sample n items from x with replacement
- **sample(x, n, replace = T, prob=p)** probability sample of n items from x with replacement

Data summaries

If x is a data vector, some important commands are (note, if missing data, can often add **na.rm=TRUE** as an argument)

- **mean(x)** computes the mean of x
- **sd(x)** computes the sample standard deviation of x (note, uses $\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$)
- **var(x)** computes the sample variance (variance-covariance matrix when given a matrix input)
- **median(x)** computes the median of x
- **quantile(x)** quantiles at $(0, 0.25, 0.5, 0.75, 1)$ by default; interpolates via

$$\begin{aligned} quantile(x,p) =& [1-(p(n-1)-\lfloor p(n-1)\rfloor)]x_{(1+\lfloor p(n-1)\rfloor)} \\ &+ [p(n-1)-\lfloor p(n-1)\rfloor]x_{(2+\lfloor p(n-1)\rfloor)} \end{aligned}$$

- **summary(x)** gives *min*, *Q1*, *median*, *mean*, *Q3*, and *max* of a vector

Classical statistical tests

Many classical univariate (and multivariate!) tests are included in R

Examples include:

`binom.test`	`chisq.test`	`cor.test`	`fisher.test`
`friedman.test`	`kruskal.test`	`mantelhaen.test`	`mcnemar.test`
`prop.test`	`t.test`	`var.test`	`wilcox.test`
`chisq.gof`	`ks.gof`		

with many, many more (Google is your friend here!)

Example

```r
# using the sleep data set
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

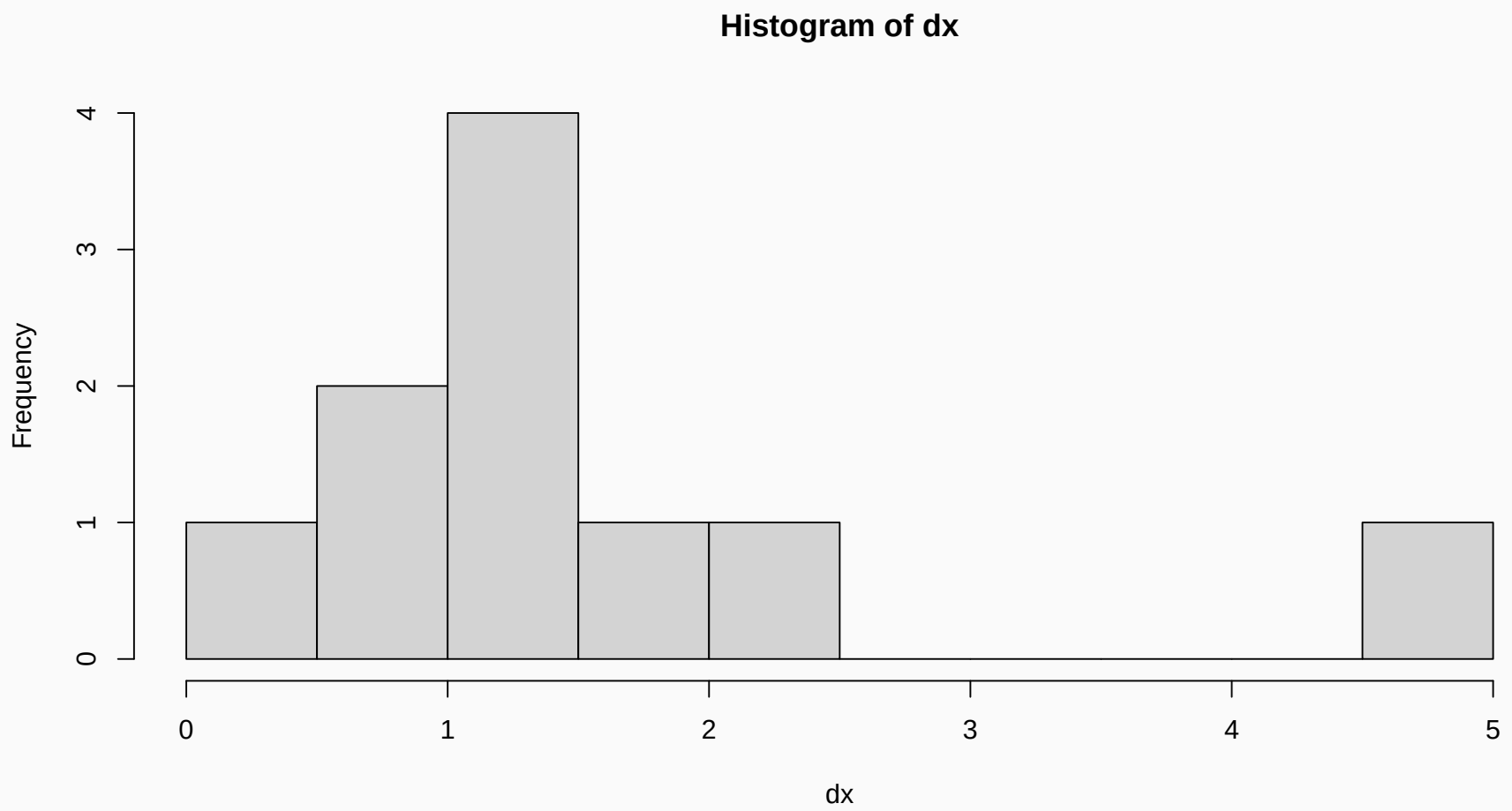
```r
data(sleep)
head(sleep)
```

```
##   extra group ID
## 1   0.7     1  1
## 2  -1.6     1  2
## 3  -0.2     1  3
## 4  -1.2     1  4
## 5  -0.1     1  5
## 6   3.4     1  6
```

```r
x1 <- sleep[sleep$group==1, 1]
x2 <- sleep[sleep$group==2, 1]
dx <- x2 - x1
```

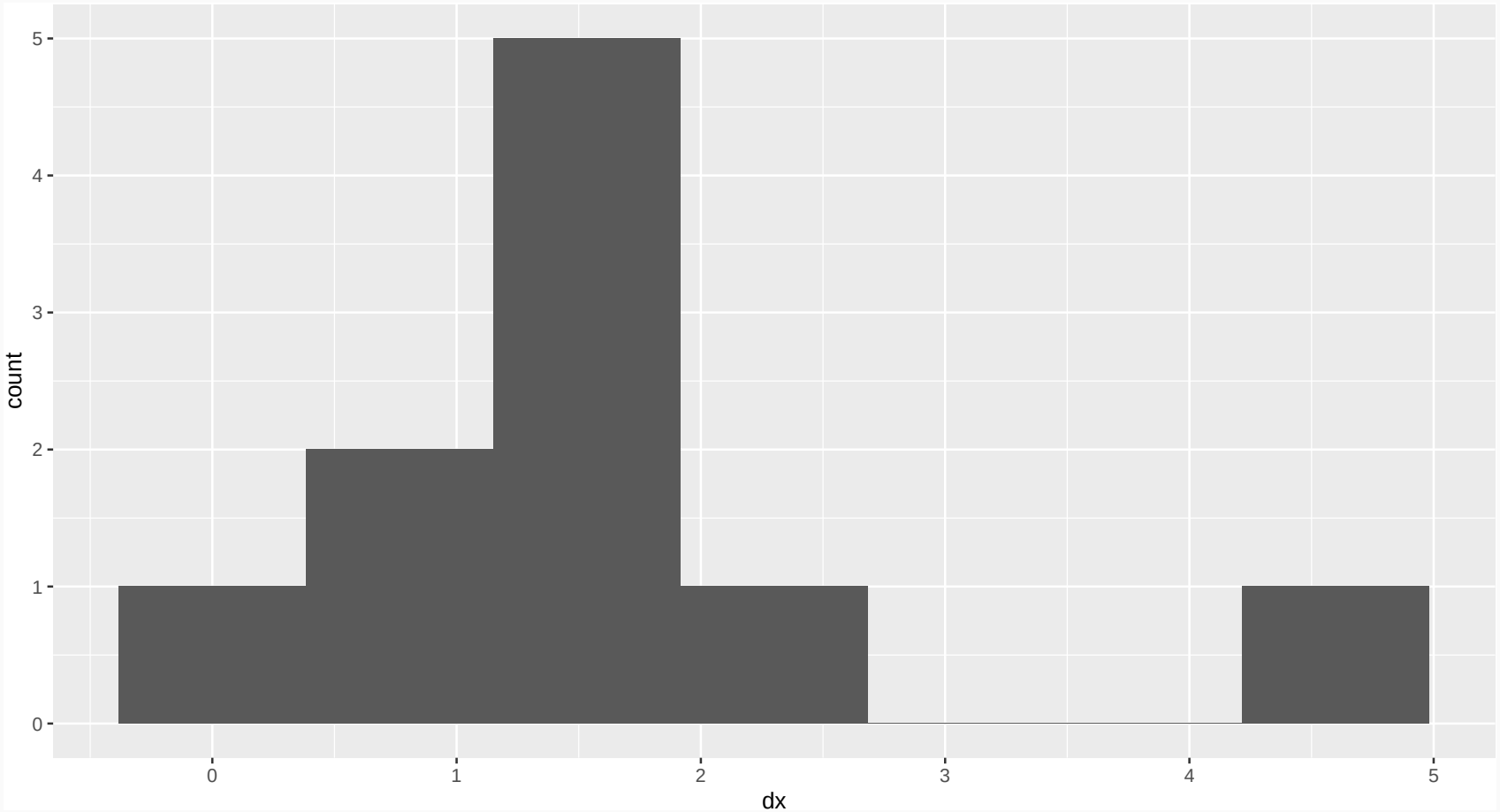
Example

```
# visualize the difference
hist(dx,breaks=7)
```



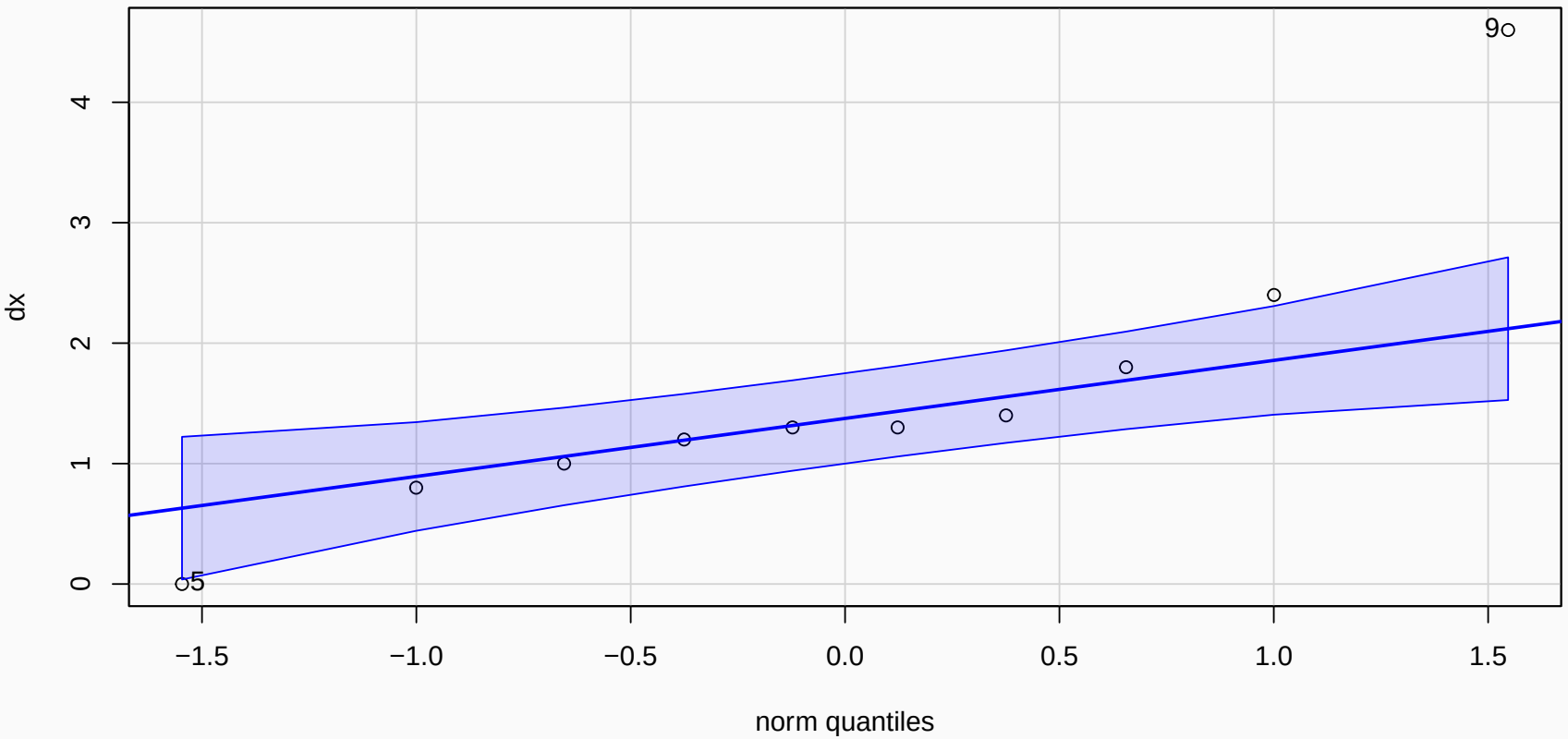
Example

```
# visualize the difference
dx.df <- data.frame(dx = sleep$extra[sleep$group == 2] -
                         sleep$extra[sleep$group == 1])
ggplot(dx.df, aes(x = dx)) + geom_histogram(bins=7)
```



Example

```
# visualize the difference
qqp(dx)
```




```
## [1] 9 5
```

("confidence envelope is based on the SEs of the order statistics of an independent random sample from the comparison distribution")

Example

```r
# paired t-test
tp <- t.test(x1,x2, paired = TRUE,
             alternative = "less")
tp
```

```
##
##  Paired t-test
##
## data:  x1 and x2
## t = -4.0621, df = 9, p-value = 0.001416
## alternative hypothesis: true mean difference is less than 0
## 95 percent confidence interval:
##        -Inf -0.8669947
## sample estimates:
## mean difference
##           -1.58
```

Example

(warnings suppressed)

```
# Wilcoxon ('Mann-Whitney' test)
wp <- wilcox.test(x2,x1, paired = TRUE,
                  alternative = "greater")
wp
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x2 and x1
## V = 45, p-value = 0.004545
## alternative hypothesis: true location shift is greater than 0
```