Statistics functions

AUTHOR
Karl Gregory

R has many built-in functions for generating realizations of random variables and evaluating probability mass/density functions (pmfs/pdfs), cumulative probability functions (cdfs), and quantile functions.

Random number generation

Before introducing these functions, we wish to comment briefly on the idea of generating random numbers from a computer. It is actually not possible to generate numbers truly at random from a computer (we might ask whether it is possible by *any means at all* to generate random numbers, but we will not stray this far into the philosophical). A computer program will produce output based deterministically on an input, meaning that once the input is known, the output is known. If one can generate a random input, then one can retrieve from a computer a sequence of numbers corresponding to this input: such an input is called a *seed*, and the numbers generated based on the seed can be regarded as *pseudo-random* numbers. Once a seed is fixed, the sequence of pseudo-random numbers a computer will produce is also fixed. Some software will set the seed according to the exact time of day that the software began running, so that each time a user runs the software, different sequences of pseudo-random numbers will be produced.

Many careers have been spent on coming up with better ways to make a computer construct sequences of numbers, based on a seed, which will satisfy tests of randomness. The principal aim has been to generate a sequence of numbers which act like independent realizations from the uniform distribution on the interval $(0, 1)$. We can retrieve such sequence in R with the `runif()` function:

```
runif(50) # generate 50 pseudo-random Uniform(0,1) realizations
```

```
 [1] 0.9308787077 0.4566190194 0.3423241214 0.1388481127 0.5347049315
 [6] 0.3907967699 0.2695951485 0.1702047708 0.3698448180 0.0007702245
[11] 0.7132399790 0.0772755996 0.9763683723 0.9537657339 0.7771071785
[16] 0.5586855146 0.8333425578 0.3559388467 0.2689520475 0.3119368011
[21] 0.2071755244 0.8089807765 0.1426891489 0.6624902359 0.2159810779
[26] 0.0204272333 0.0310420173 0.2935617536 0.4668650182 0.6845511305
[31] 0.2843407975 0.9921871182 0.3674608748 0.2078510467 0.9063579065
[36] 0.6289490701 0.7650181742 0.8604083005 0.5786431504 0.3609595336
[41] 0.4559477731 0.6450185589 0.4616275323 0.4494822051 0.6420732702
[46] 0.5413725451 0.2863077268 0.7698679457 0.4721740680 0.4448345422
```

Once one has obtained a random realization on the interval $(0, 1)$, one can transform it into a realization of a random variable from any other distribution by passing it through the quantile function of that distribution. Specifically, suppose X is a random variable with cdf given by

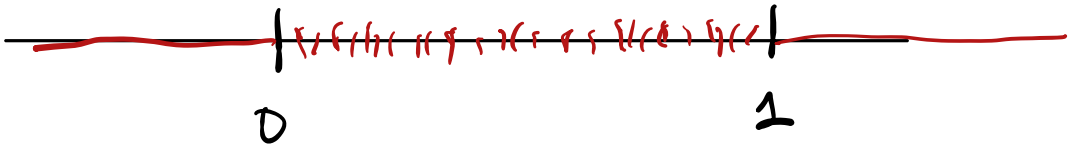
$$F(x) = P(X \leq x), \quad x \in \mathbb{R}.$$

Then the quantile function of X is given by

$$Q(u) = \inf\{x : F(x) \geq u\}, \quad u \in (0, 1),$$

Random number generation.

Main goal: Generate Uniform $(0,1)$ realizations



Random seed.

Say I want to generate $X \overset{\text{has distribution}}{\sim} F$.

$$F(x) = P(X \leq x) \qquad F: \mathbb{R} \to [0,1]$$

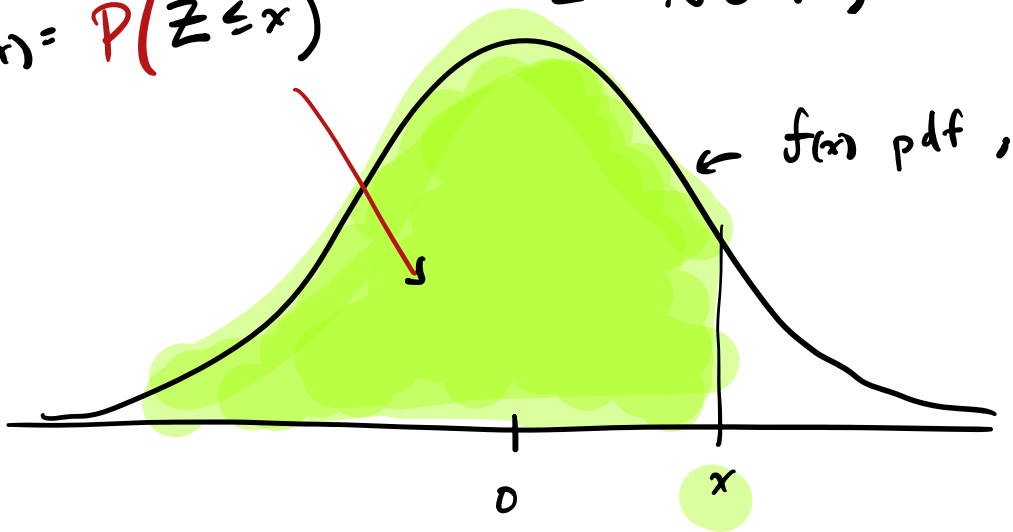
cdf (cumulative dist. function).

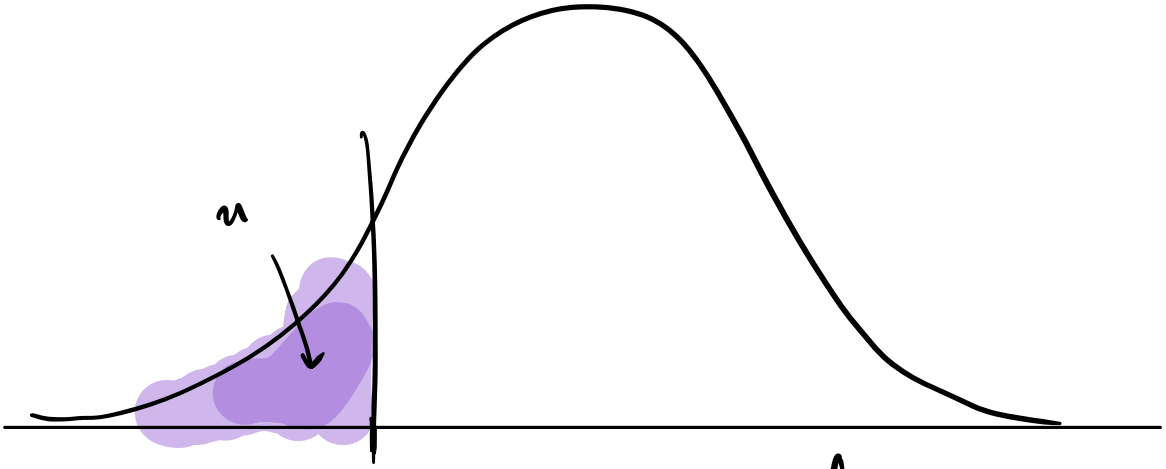
The quantile function (an inverse to the cdf)

$$Q(u) = \boxed{\phantom{F^{-1}(u)}} = F^{-1}(u)$$

$F(x) = P(Z \le x)$ cdf

$Z \sim N(0,1)$





$x = Q(u)$: quantile function

To generate X which has quantile function Q, generate $U \sim \text{Uniform}(0,1)$ and set

$$X = Q(U).$$

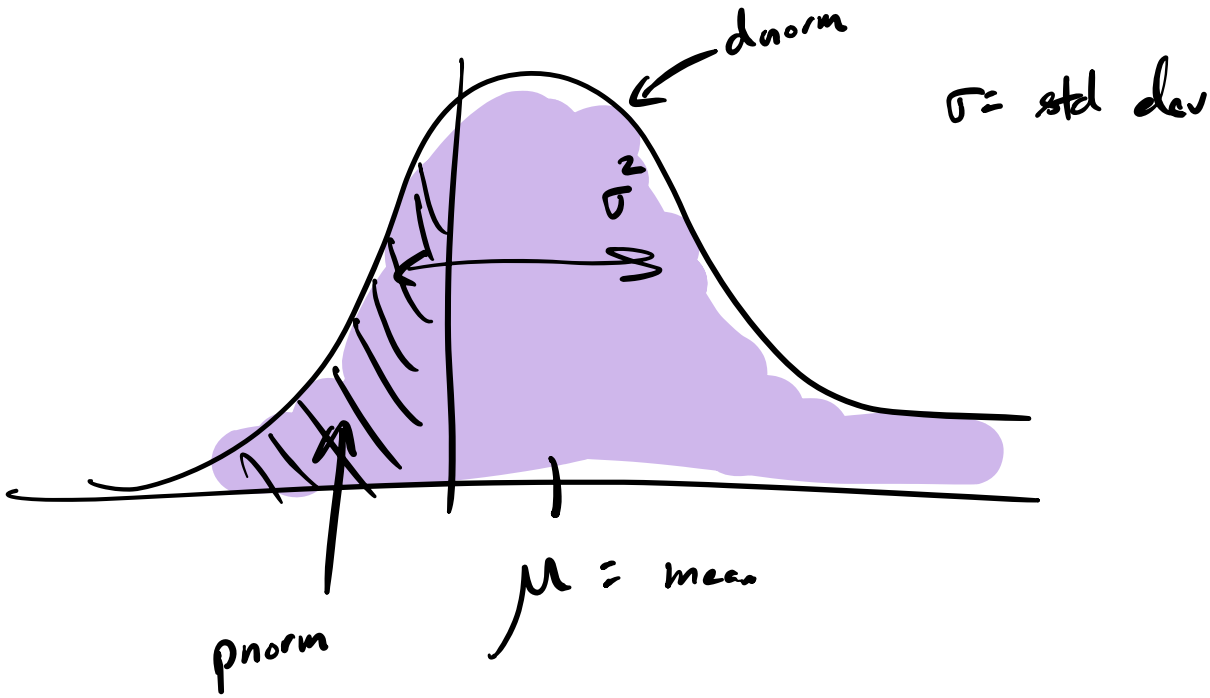
Logistic distribution:

$$F(x) = \frac{e^x}{1+e^x}$$

$$Q(u) = \log\left(\frac{u}{1-u}\right)$$

To generate $X \sim$ logistic dist. Generate $U \sim Unif(0,1)$.

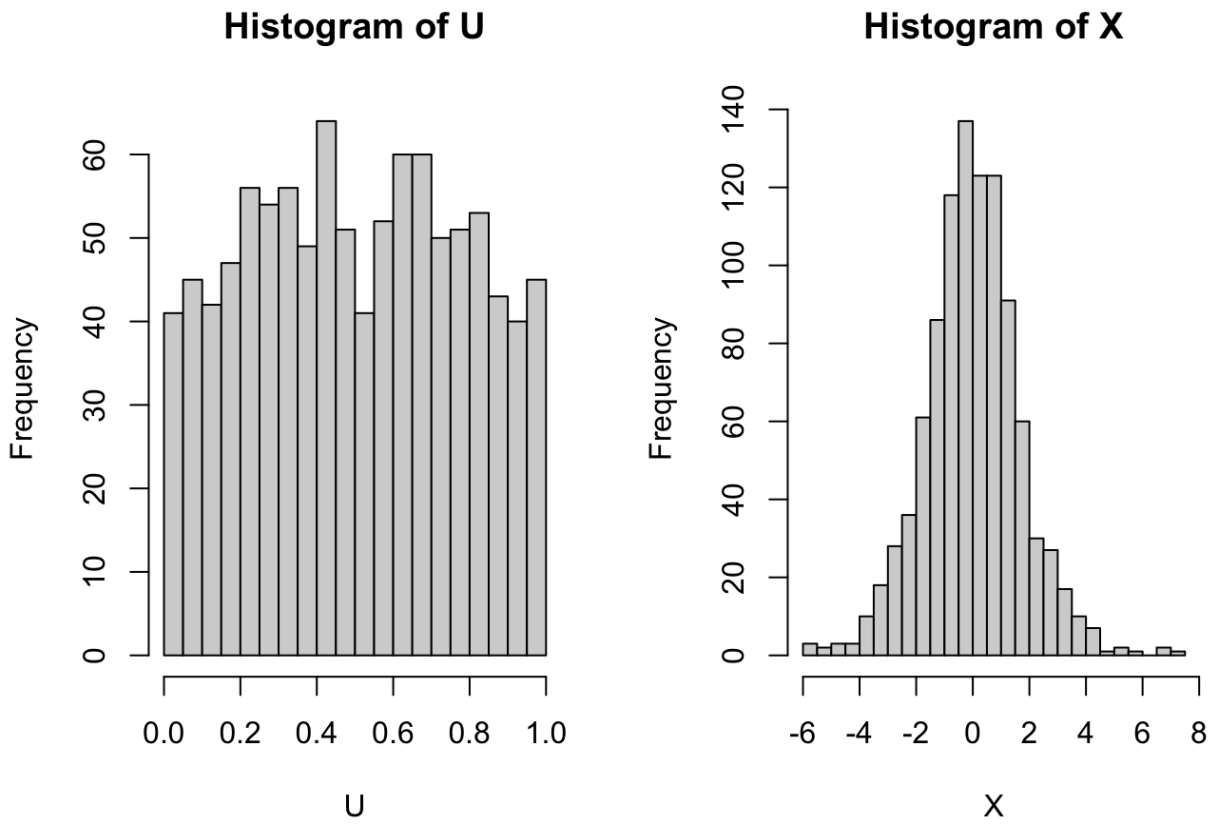
then set $X = \log\left(\frac{U}{1-U}\right)$.



where $Q(u)$ is simply the inverse of $F(x)$ when the latter is continuous and strictly increasing. So to generate a realization of X, we generate a realization U from the uniform distribution on $(0,1)$ and set $X = Q(U)$.

For example, to generate a realization of a random variable X having the logistic cdf $F(x) = e^x/(1-e^x)$, we can generate a realization U from the uniform distribution on $(0,1)$ and set $X = Q(U)$, where $Q(u) = \log(u/(1-u))$ for $u \in (0,1)$.

```
U <- runif(1000)
X <- log(U/(1-U)) # logistic distribution quantile function

par(mfrow=c(1,2)) # put next two plots side-by-side
hist(U,breaks = 20)
hist(X,breaks = 20)
```



When generating realizations of random variables in R, this is what is happening in the background; the generation begins with the generation of pseudo-random realizations from the uniform distribution on $(0,1)$.

The rabbit hole of how such sequences of pseudo-random uniform realizations are generated is very deep, and we will not peer further into it than we have done in the above paragraphs. If interested see Chapter 2 of Robert and Casella (2004).

Functions for generating random variables

For each of several different distributions, R provides four functions: one for generating random realizations, one for evaluating the pmf or pdf (according to whether the distribution is for a discrete or continuous random variable), one for evaluating the cdf, and one for evaluating the quantile function. These four functions take the form `rxxx()`, `dxxx()`, `pxxx()`, and `qxxx()`, respectively, where `xxx` is replaced by an abbreviation for a distribution. Here are two examples, one continuous and one discrete:

- normal distribution: `rnorm()`, `dnorm()`, `pnorm()`, and `qnorm()`.
- binomial distribution: `rbinom()`, `dbinom()`, `pbinom()`, and `qbinom()`.

Each set of functions takes a different set of optional arguments depending on the distribution. For example, for the normal distribution the mean and standard deviation can be specified and for the binomial distribution the number of Bernoulli trials and the success probability of each trial can be specified.

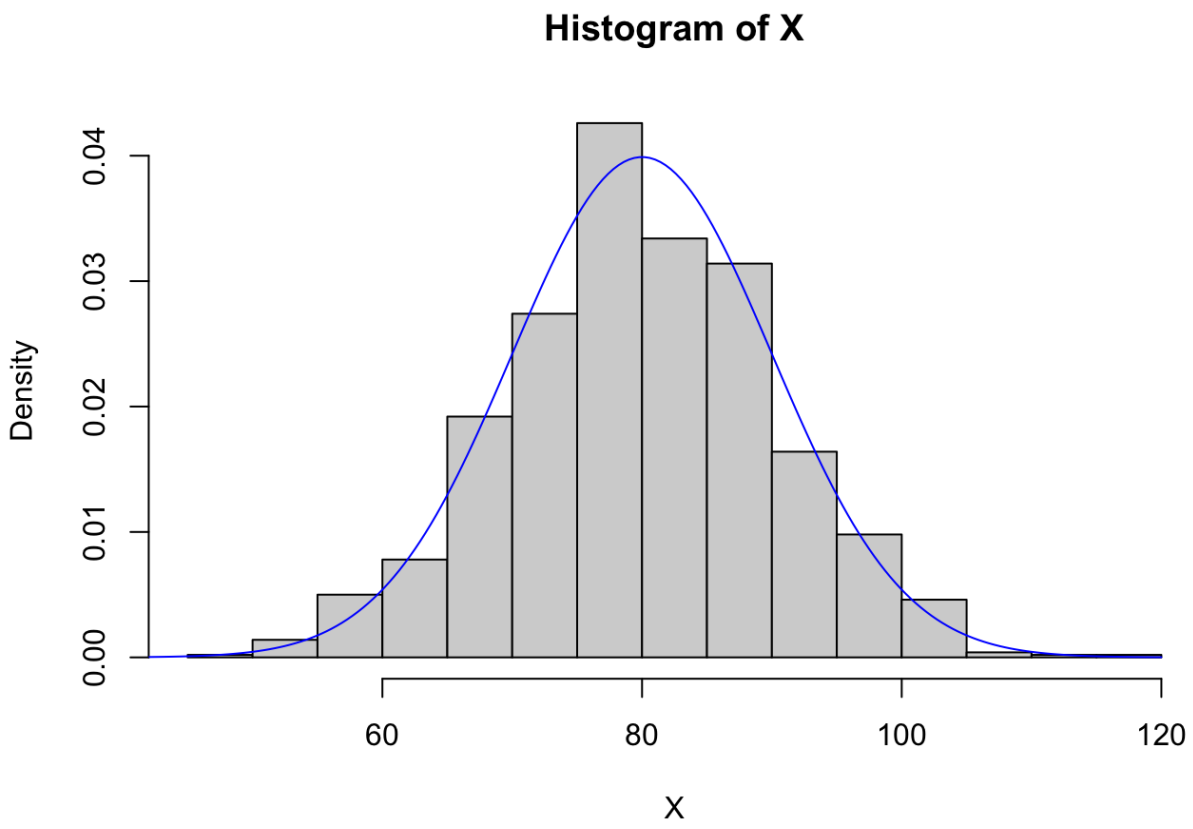
To see a list of all the distributions included in R, execute this command:

```
?distributions
```

Normal distribution functions

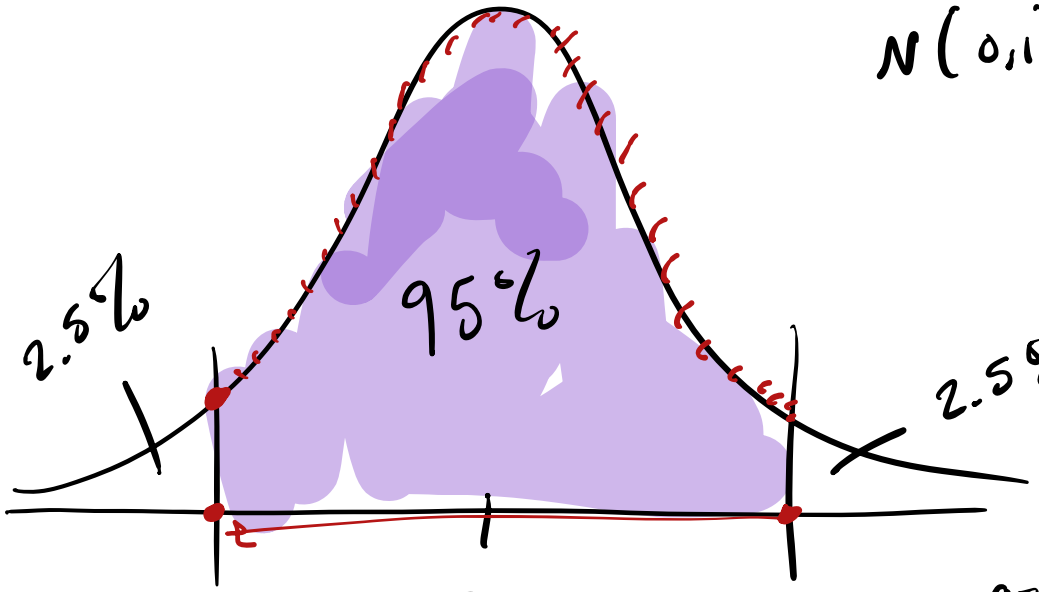
Here we illustrate using the `rnorm()` and `dnorm()` functions:

```
set.seed(1) # you don't need to do this, but if you want to get the same "random" numbers each

# rnorm() function to generate random realizations
n <- 1000
mu <- 80
sigma <- 10
X <- rnorm(n,mean = mu,sd = sigma) # specify mean and standard deviation

# make a histogram of the realizations
hist(X,freq = F, breaks = 20) # set freq = F so it will have the height of a density

# overlay pdf with dnorm() function evaluated at a sequence of x values
x <- seq(mu - 4*sigma, mu + 4*sigma,length=200)
lines(dnorm(x,mu,sigma)~x, col = "blue")
```

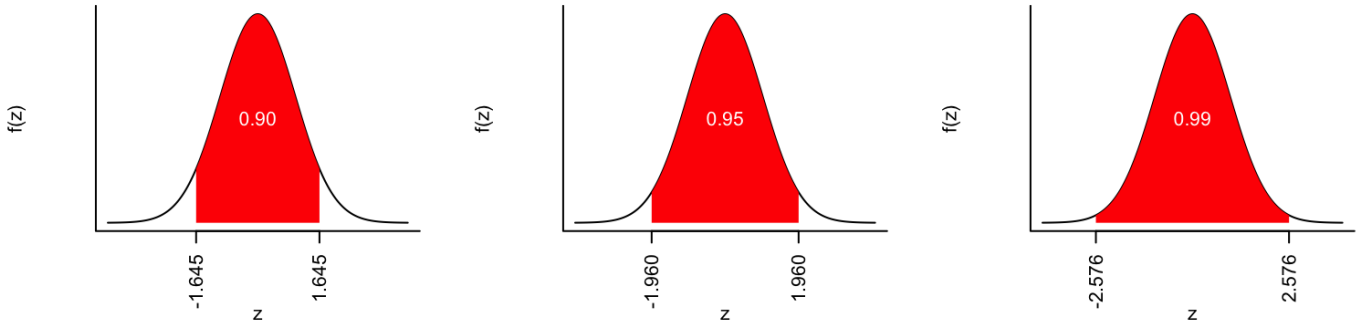


Here we use `dnorm()` as well as `qnorm()` to make some cool plots of the standard normal pdf.

```
# make three plots in one row
par(mfrow=c(1,3))

# make a sequence of z values
z <- seq(-4,4,length=200)

# define some upper quantiles
u <- c(0.05,0.025,0.005)

# for each value in u, shade the region between the upper and lower quantiles
for(j in 1:length(u)){

  qu <- qnorm(c(u[j],1-u[j]))
  plot(dnorm(z)~z,
       type = "l",
       ylab = "f(z)",
       bty = "l",
       xaxt = "n",
       yaxt = "n")

  axis(side = 1, las = 2, at = qu, labels = sprintf(qu,fmt="%.3f"))

  x <- c(qu[1],seq(qu[1],qu[2],length=100),qu[2])
  y <- c(0,dnorm(seq(qu[1],qu[2],length=100)),0)
  polygon(x, y, col = "red",border = "NA")
```


N(0,1)
2.5%
95%
2.5%
-1.96
0
1.96
= 97.5th percentile
= qnorm(0.975)
qnorm(0.025)

```
  text(x = 0, y = 0.5*dnorm(0), label = sprintf(1-2*u[j],fmt="%.2f"), col = "white")

}
```



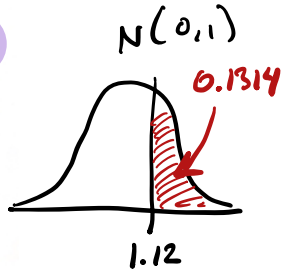
Here we use the `pnorm()` function to reproduce this z-table from Mohr, Wilson, and Freund (2021):

Table A.1 Table of the Standard Normal Distribution — Probabilities exceeding Z are given in the body of the table.

Tenths Place of Z is Given in the Row Label, and Hundredths Place of Z is in the Column Headings.

Z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.4960	0.4920	0.4880	0.4840	0.4801	0.4761	0.4721	0.4681	0.4641
0.1	0.4602	0.4562	0.4522	0.4483	0.4443	0.4404	0.4364	0.4325	0.4286	0.4247
0.2	0.4207	0.4168	0.4129	0.4090	0.4052	0.4013	0.3974	0.3936	0.3897	0.3859
0.3	0.3821	0.3783	0.3745	0.3707	0.3669	0.3632	0.3594	0.3557	0.3520	0.3483
0.4	0.3446	0.3409	0.3372	0.3336	0.3300	0.3264	0.3228	0.3192	0.3156	0.3121
0.5	0.3085	0.3050	0.3015	0.2981	0.2946	0.2912	0.2877	0.2843	0.2810	0.2776
0.6	0.2743	0.2709	0.2676	0.2643	0.2611	0.2578	0.2546	0.2514	0.2483	0.2451
0.7	0.2420	0.2389	0.2358	0.2327	0.2296	0.2266	0.2236	0.2206	0.2177	0.2148
0.8	0.2119	0.2090	0.2061	0.2033	0.2005	0.1977	0.1949	0.1922	0.1894	0.1867
0.9	0.1841	0.1814	0.1788	0.1762	0.1736	0.1711	0.1685	0.1660	0.1635	0.1611
1.0	0.1587	0.1562	0.1539	0.1515	0.1492	0.1469	0.1446	0.1423	0.1401	0.1379
1.1	0.1357	0.1335	0.1314	0.1292	0.1271	0.1251	0.1230	0.1210	0.1190	0.1170
1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985
1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0721	0.0708	0.0694	0.0681
1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
1.8	0.0359	0.0351	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183
2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0048
2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
2.9	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
3.2	0.0007	0.0007	0.0006	0.0006	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005
3.3	0.0005	0.0005	0.0005	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0003
3.4	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002
3.5	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002

Example: $P(Z > 2.94) = 0.0016$; $P(Z < -2.94) = 0.0016$; $P(1.91 < Z < 2.33) = 0.0281 - 0.0099 = 0.0182$.
On the TI-84 calculator, $P(Z > 2.94)$ = normcdf(2.94,1E99,0,1); $P(1.91 < Z < 2.33)$ = normcdf(1.91,2.33,0,1) = 0.0182.
This table produced using Microsoft Excel with entries = 1-NORM.S.DIST(z,1).



1 - pnorm(1.12)

```
cols <- seq(0,0.09,by=0.01)
rows <- seq(0,3.5,by=0.1)

ncols <- length(cols)
nrows <- length(rows)
```

```r
tab <- matrix(NA,nrows,ncols)
for(i in 1:nrows)
  for(j in 1:ncols){

    z <- rows[i] + cols[j]
    tab[i,j] <- round(1 - pnorm(z),4) # area under the curve to the right of z, rounded to fou

  }

# name the columns and rows
colnames(tab) <- cols
rownames(tab) <- rows

tab
```

```
        0   0.01   0.02   0.03   0.04   0.05   0.06   0.07   0.08   0.09
0   0.5000 0.4960 0.4920 0.4880 0.4840 0.4801 0.4761 0.4721 0.4681 0.4641
0.1 0.4602 0.4562 0.4522 0.4483 0.4443 0.4404 0.4364 0.4325 0.4286 0.4247
0.2 0.4207 0.4168 0.4129 0.4090 0.4052 0.4013 0.3974 0.3936 0.3897 0.3859
0.3 0.3821 0.3783 0.3745 0.3707 0.3669 0.3632 0.3594 0.3557 0.3520 0.3483
0.4 0.3446 0.3409 0.3372 0.3336 0.3300 0.3264 0.3228 0.3192 0.3156 0.3121
0.5 0.3085 0.3050 0.3015 0.2981 0.2946 0.2912 0.2877 0.2843 0.2810 0.2776
0.6 0.2743 0.2709 0.2676 0.2643 0.2611 0.2578 0.2546 0.2514 0.2483 0.2451
0.7 0.2420 0.2389 0.2358 0.2327 0.2296 0.2266 0.2236 0.2206 0.2177 0.2148
0.8 0.2119 0.2090 0.2061 0.2033 0.2005 0.1977 0.1949 0.1922 0.1894 0.1867
0.9 0.1841 0.1814 0.1788 0.1762 0.1736 0.1711 0.1685 0.1660 0.1635 0.1611
1   0.1587 0.1562 0.1539 0.1515 0.1492 0.1469 0.1446 0.1423 0.1401 0.1379
1.1 0.1357 0.1335 0.1314 0.1292 0.1271 0.1251 0.1230 0.1210 0.1190 0.1170
1.2 0.1151 0.1131 0.1112 0.1093 0.1075 0.1056 0.1038 0.1020 0.1003 0.0985
1.3 0.0968 0.0951 0.0934 0.0918 0.0901 0.0885 0.0869 0.0853 0.0838 0.0823
1.4 0.0808 0.0793 0.0778 0.0764 0.0749 0.0735 0.0721 0.0708 0.0694 0.0681
1.5 0.0668 0.0655 0.0643 0.0630 0.0618 0.0606 0.0594 0.0582 0.0571 0.0559
1.6 0.0548 0.0537 0.0526 0.0516 0.0505 0.0495 0.0485 0.0475 0.0465 0.0455
1.7 0.0446 0.0436 0.0427 0.0418 0.0409 0.0401 0.0392 0.0384 0.0375 0.0367
1.8 0.0359 0.0351 0.0344 0.0336 0.0329 0.0322 0.0314 0.0307 0.0301 0.0294
1.9 0.0287 0.0281 0.0274 0.0268 0.0262 0.0256 0.0250 0.0244 0.0239 0.0233
2   0.0228 0.0222 0.0217 0.0212 0.0207 0.0202 0.0197 0.0192 0.0188 0.0183
2.1 0.0179 0.0174 0.0170 0.0166 0.0162 0.0158 0.0154 0.0150 0.0146 0.0143
2.2 0.0139 0.0136 0.0132 0.0129 0.0125 0.0122 0.0119 0.0116 0.0113 0.0110
2.3 0.0107 0.0104 0.0102 0.0099 0.0096 0.0094 0.0091 0.0089 0.0087 0.0084
2.4 0.0082 0.0080 0.0078 0.0075 0.0073 0.0071 0.0069 0.0068 0.0066 0.0064
2.5 0.0062 0.0060 0.0059 0.0057 0.0055 0.0054 0.0052 0.0051 0.0049 0.0048
2.6 0.0047 0.0045 0.0044 0.0043 0.0041 0.0040 0.0039 0.0038 0.0037 0.0036
2.7 0.0035 0.0034 0.0033 0.0032 0.0031 0.0030 0.0029 0.0028 0.0027 0.0026
2.8 0.0026 0.0025 0.0024 0.0023 0.0023 0.0022 0.0021 0.0021 0.0020 0.0019
2.9 0.0019 0.0018 0.0018 0.0017 0.0016 0.0016 0.0015 0.0015 0.0014 0.0014
3   0.0013 0.0013 0.0013 0.0012 0.0012 0.0011 0.0011 0.0011 0.0010 0.0010
3.1 0.0010 0.0009 0.0009 0.0009 0.0008 0.0008 0.0008 0.0008 0.0007 0.0007
3.2 0.0007 0.0007 0.0006 0.0006 0.0006 0.0006 0.0006 0.0005 0.0005 0.0005
3.3 0.0005 0.0005 0.0005 0.0004 0.0004 0.0004 0.0004 0.0004 0.0004 0.0003
3.4 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0002
3.5 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002
```

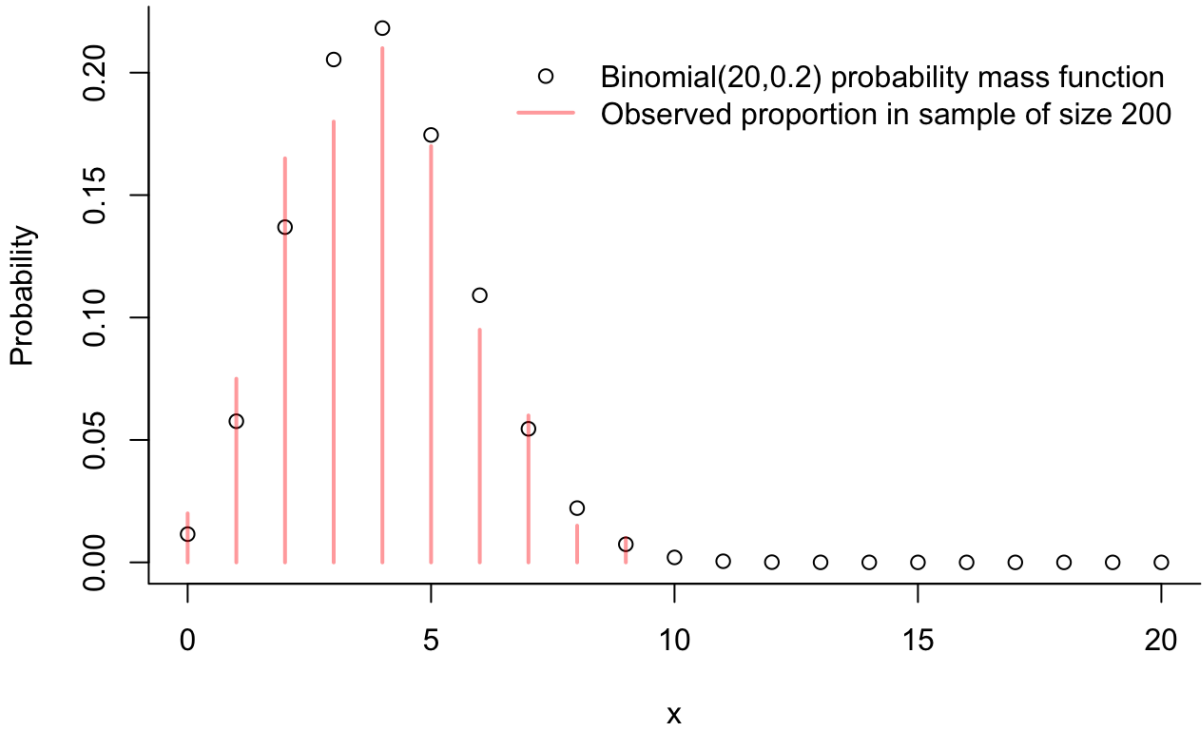
Binomial distribution functions

Here we illustrate the `rbinom()` and `dbinom()` functions.

```
N <- 200 # draw this many realizations
n <- 20 # number of Bernoulli trials for the Binomial random variable
p <- 1/5 # success probability
X <- rbinom(N,n,p)

# the table() function can be useful for summarizing observations of a discrete random variabl
table(X)
```

```
X
 0  1  2  3  4  5  6  7  8  9
 4 15 33 36 42 34 19 12  3  2
```

```
x <- 0:n
plot(dbinom(x,n,p)~x,
     bty = "l",
     xlab = "x",
     ylab = "Probability")

points(table(X)/N, col = rgb(1,0,0,.4))

# Next two lines help you find coordinates relative to the plotting window.
# Useful if you want to position a legend irrespectively of the axes.
xpos <- grconvertX(.4, from="ndc", to = "user") # 40% percent from the left
ypos <- grconvertY(.8, from="ndc", to = "user") # 80% percent from the bottom

legend(x = xpos,
       y = ypos,
       pch = c(1,NA),
       lty = c(NA,1),
       lwd = c(NA,2),
       col = c("black",rgb(1,0,0,.4)),
       legend = c(paste("Binomial(",n,",",p,") probability mass function",sep=""),
                  paste("Observed proportion in sample of size ",N,sep="")),
       bty = "n")
```



Here we illustrate the `pbinom()` function in a program which makes some plots for visualizing the normal approximation to the binomial distribution. Recall that for $X \sim \text{Binomial}(n, p)$ we have

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}.$$

For large n, the central limit theorem gives

$$X \overset{\text{approx}}{\sim} \text{Normal}(np, np(1-p)),$$

where the approximation improves as $n \to \infty$. Now, we would like to use the Normal distribution to approximate the probabilities $P(X = x)$, but since the Normal distribution is for a continuous random variable, it will assign probability zero to any single value x. To get around this, we note that for the binomial random variable X, which is discrete, we have

$$P(X = x) = P(x - 1/2 < X < x + 1/2)$$

for every integer-valued x, so we will approximate the probability $P(X = x)$ with the probability assigned by the $\text{Normal}(np, np(1-p))$ distribution to the interval $(x - 1/2, x + 1/2)$. That is, we will use the approximation

$$P(X = x) \approx \begin{cases} \Phi\left(\frac{x+1/2-np}{\sqrt{np(1-p)}}\right), & x = 0 \\ \Phi\left(\frac{x+1/2-np}{\sqrt{np(1-p)}}\right) - \Phi\left(\frac{x-1/2-np}{\sqrt{np(1-p)}}\right), & x > 0, \end{cases}$$

where Φ denotes the cdf of the $\text{Normal}(0, 1)$ distribution. Likewise, we will approximate the cumulative distribution

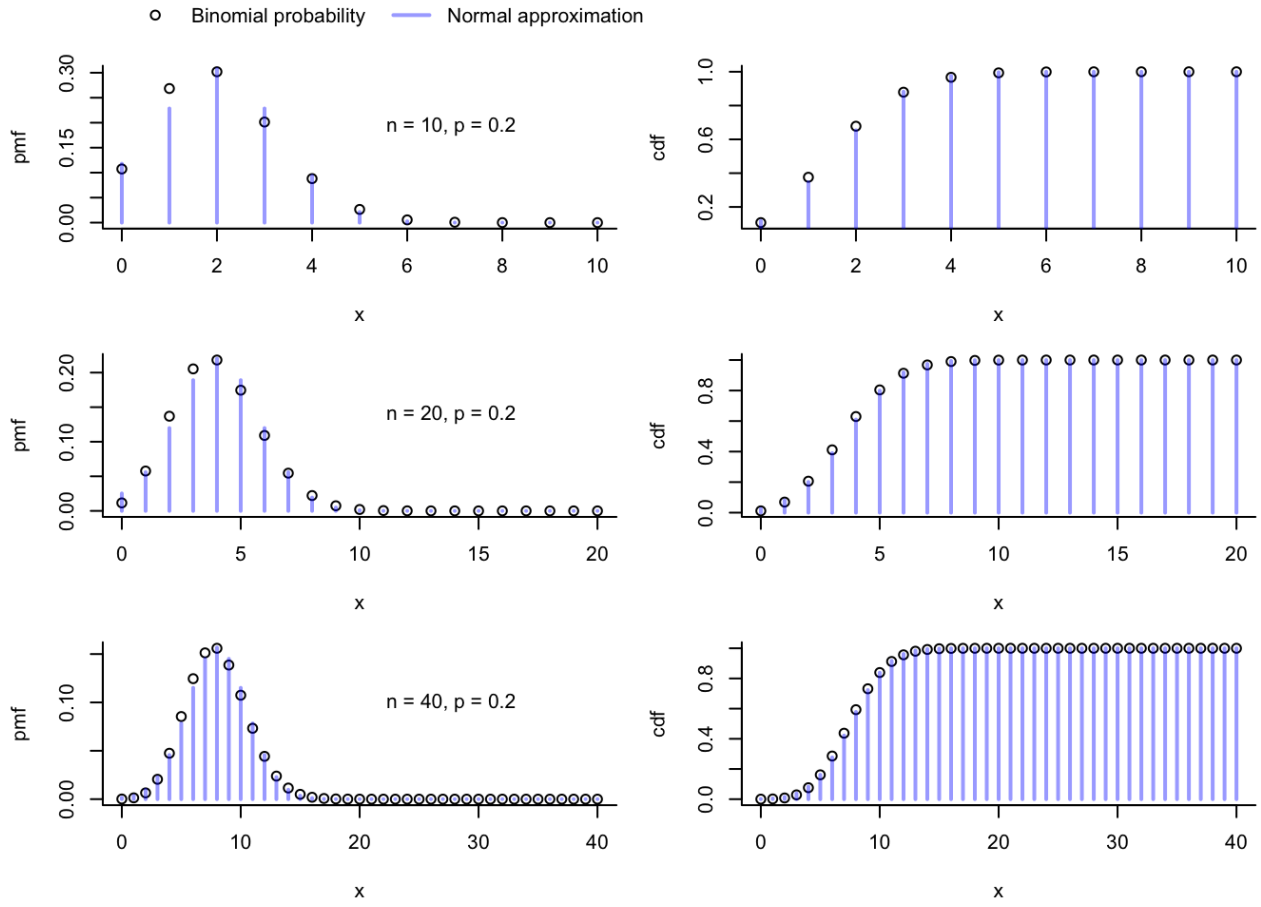
function of the $\text{Binomial}(n,p)$ distribution as

$$P(X \leq x) \approx \Phi\Big(\frac{x+1/2-np}{\sqrt{np(1-p)}}\Big)$$

These approximations improve as $n \to \infty$.

```
p <- 1/5 # success probability
nn <- c(10,20,40) # select a few sample sizes

# set up plotting so the the plots will appear in a 3 by 2 grid
par(mfrow=c(3,2),
    mar=c(4.1,4.1,1.1,1.1), # adjust the margin of each plot so the plots don't get squished
    oma = c(0,0,2,0)) # make a top outer margin in which to place a legend

# loop through the three sample sizes in nn
for(i in 1:3){

  # set sample size for this step in the loop
  n <- nn[i]
  x <- 0:n

  # obtain normal approximations to cdf and pmf
  x_correct <- 0:n + 0.5 # use the continuity correction
  pn <- pnorm(x_correct,n*p,sd = sqrt(n*p*(1-p))) # cdf
  dn <- c(pn[1],diff(pn)) # pmf by differencing. Check ?diff

  # make plots
  plot(dbinom(x,n,p)~x,
       bty = "l",
       xlab = "x",
       ylab = "pmf")

  points(dn~x, col = rgb(0,0,1,.4), type = "h", lwd = 2)

  xpos <- grconvertX(.7,from="nfc",to="user")
  ypos <- grconvertY(.7,from="nfc",to="user")

  text(x = xpos,
       y = ypos,
       labels = paste("n = ",n,", p = ",p,sep=""))

  plot(pbinom(x,n,p)~x,
       bty = "l",
       xlab = "x",
       ylab = "cdf"
       )

  points(pn~x, col = rgb(0,0,1,.4), type = "h", lwd = 2)

}

xpos <- grconvertX(.1,from="ndc",to="user")
```

```
ypos <- grconvertY(1,from="ndc",to="user")
legend(x = xpos,
       y = ypos,
       horiz = T, # give legend a horizontal orientation
       pch = c(1,NA),
       lty = c(NA,1),
       lwd = c(NA,2),
       col = c("black",rgb(0,0,1,.4)),
       legend = c("Binomial probability",
                  "Normal approximation"),
       bty = "n",
       xpd = NA) # xpd = NA is sometimes need to make something appear in the outer margin
```



You may experiment with the `qbinom()` function on your own!

Practice

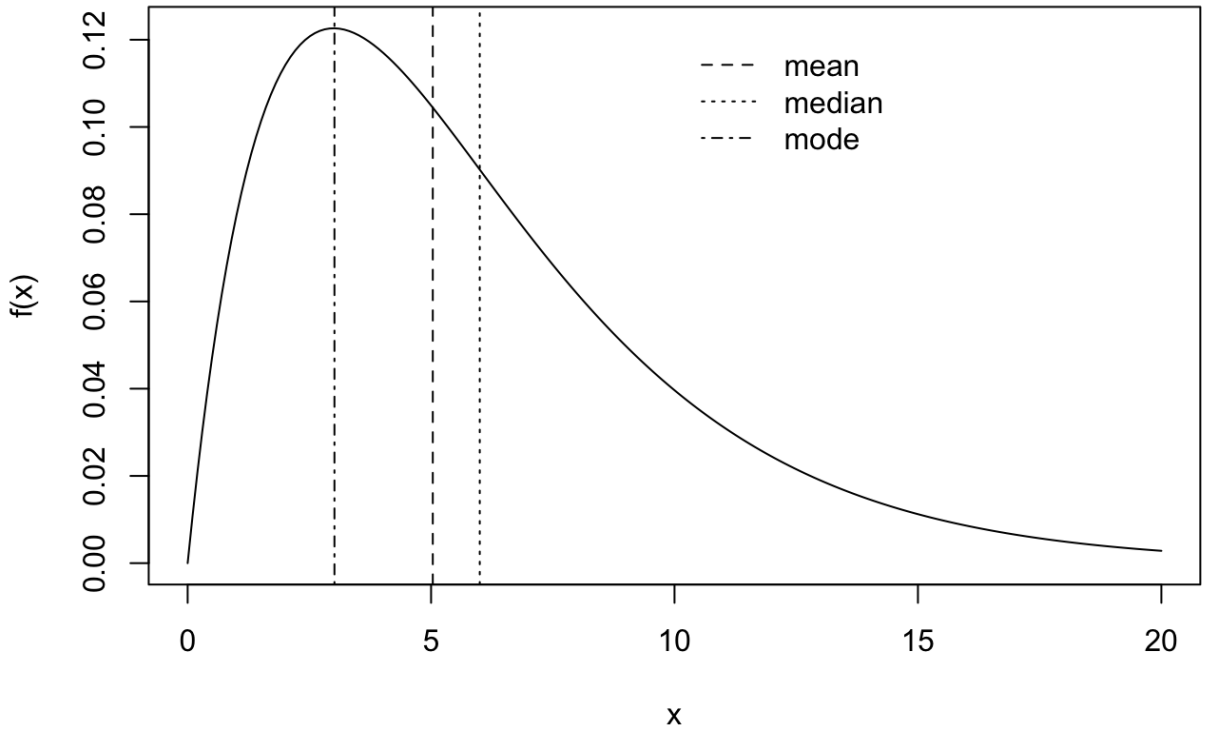
Practice writing code and anticipating the output of code with the following exercises.

Write code

1. Make a plot like the one below of a gamma distribution pdf. The pdf of the $\text{Gamma}(\alpha, \beta)$ distribution is given by

$$f(x, \alpha, \beta) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-x/\beta}, \quad x > 0$$

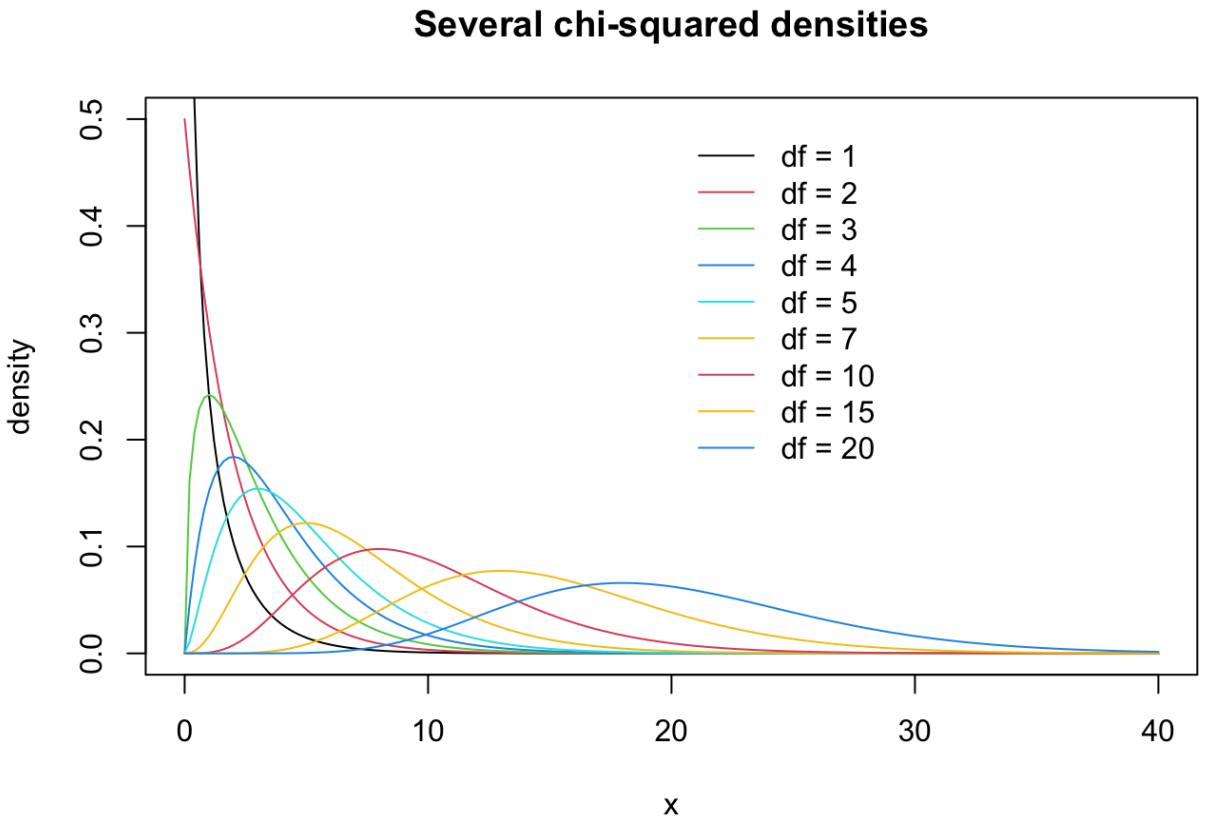
for a shape parameter α and a scale parameter β; the expected value of a random variable X having this density is $\mathbb{E}X = \alpha\beta$. The gamma distribution functions in R are `rgamma()`, `dgamma()`, `pgamma()`, and `qgamma()`. Run `?dgamma()` to make sure you understand how to put in the arguments. Choose your own values of the shape and scale parameters α and β. Put vertical lines at the mean, the median (the median is the 0.5 quantile of the distribution), and the mode (the value of x for which $f(x, \alpha, \beta)$ is greatest) as shown.



2. Make the plot below showing the pdf of the chi-squared distribution for several values of the degrees of freedom parameter. The chi-squared pdf is given by

$$f(x, \nu) = \frac{1}{\Gamma(\nu) 2^{\nu/2}} x^{\nu/2-1} e^{-x/2}, \quad x > 0$$

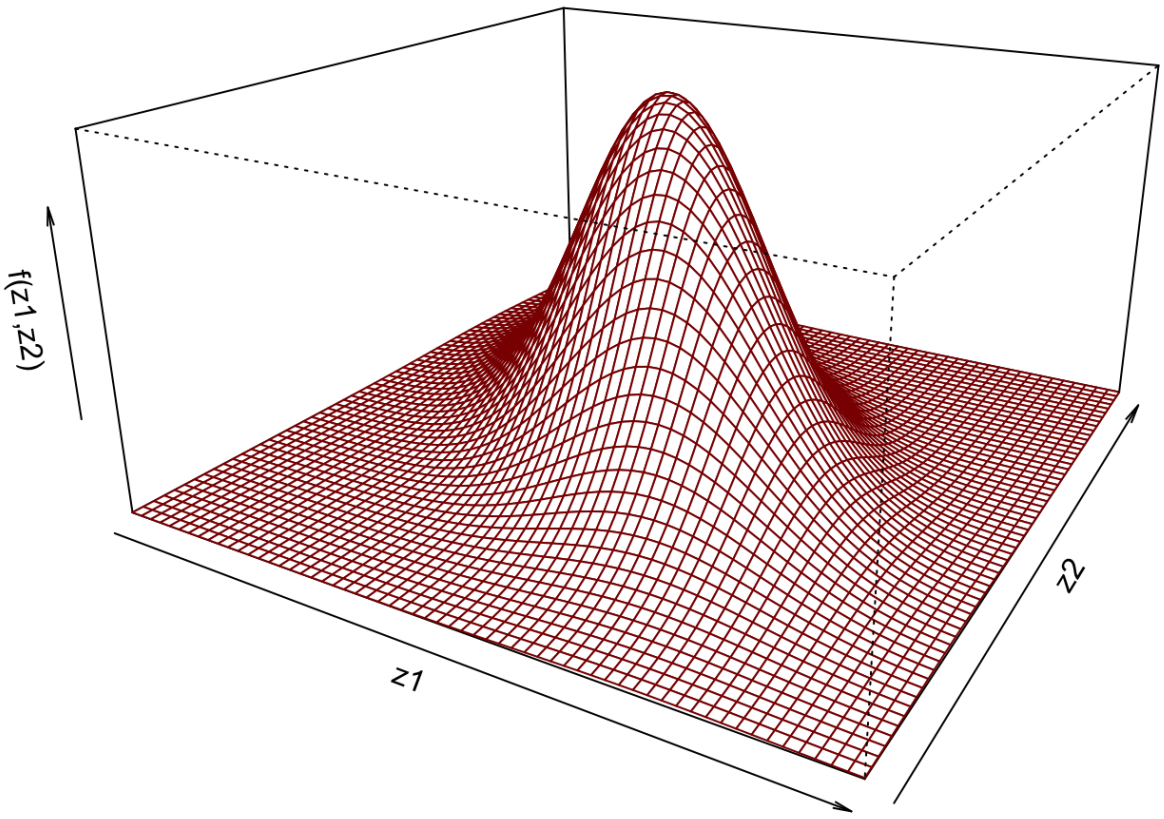
where $\nu > 0$ is the degrees of freedom parameter and $\Gamma(\cdot)$ is the gamma function. The chi-squared pdf can be evaluated with the R function `dchisq()`. Write a for loop to put all the curves on the plot (note that you can put numbers in for `col=`)!



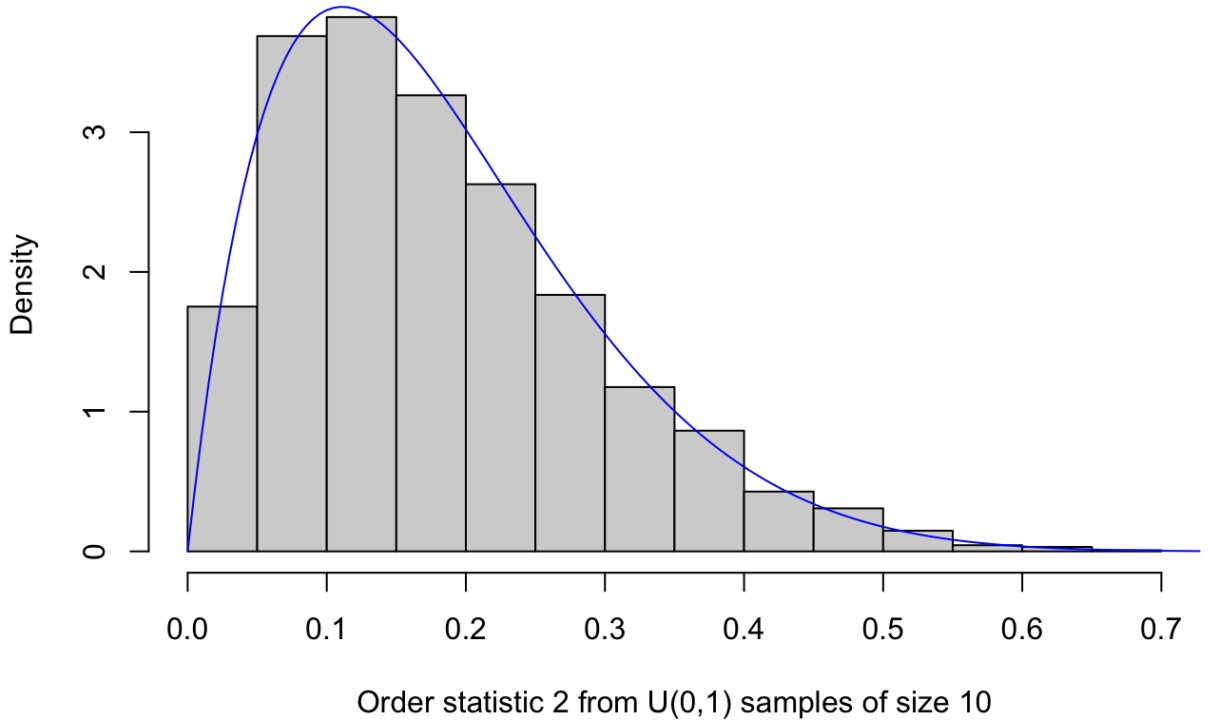
3. The standard bivariate normal probability density function is given by

$$f(z_1, z_2, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left(\frac{z_1^2 - 2\rho z_1 z_2 + z_2^2}{2(1-\rho^2)}\right), \quad z_1, z_2 \in \mathbb{R},$$

where $\rho \in (-1, 1)$ is the correlation parameter. Make a plot like the one below of this function (you will have to define your own function and evaluate it over a grid of points) with your own choice of the parameter ρ. You can use the `persp()` function to create the 3-d plot.



4. Fix some integers $n \geq 1$, $k \in \{1, \ldots, n\}$, and S, where S is large (say at least a thousand). Then generate S samples of size n from the $\mathrm{Uniform}(0, 1)$ distribution and on each sample retrieve the kth order statistic—this is the value appearing in position k when the vector is sorted in increasing order. You can do the above with with a for loop or in some other way. Then make a histogram of these kth order statistic values. On top of the histogram, overlay the pdf of the $\mathrm{Beta}(k, n - k + 1)$ distribution (can use `dbeta()` function). For $n = 10$, $k = 2$, and $S = 5000$, my plot looks like this:



Read code

1. Explain what this function will do:

```
rdbexp <- function(n,lambda = 3){

  sgn <- sample(c(-1,1),n,replace=T)
  x <- rexp(n,lambda)
  val <- sgn * x
  return(val)

}
```

2. If $Z_1, \ldots, Z_k$ are independent $\text{Normal}(0, 1)$ random variables, then $\sum_{i=1}^{k} Z_k^2$ will have the chi-squared distribution with degrees of freedom k. With this beautiful truth in mind, explain what the following code will do. What would you write as a comment after each line of code to explain what it is doing?

```
df <- 6
S <- 5000
X <- numeric(S)
for(s in 1:S){

  Z <- rnorm(df)
  X[s] <- sum(Z**2)
```

```
}

hist(X,freq = F,breaks = 50)
x <- seq(0,30,length=200)
lines(dchisq(x,df)~x,col = "blue")
```

3. Consider drawing n marbles from a bag containing N marbles, M of which are red, and counting the number of red marbles you draw. This random number follows a probability distribution called the hypergeometric distribution. If the number of marbles N in the bag is increased to a very large number, and if the number of red marbles M is increased proportionally with N, the distribution of the number of red marbles you draw from the bag in your handful of n becomes very close to the binomial distribution with the number of Bernoulli trials equal to n and the success probability given by M/N. Study the code below and see if you can explain what each line is doing. What would you write if you were to add a comment to each line of code?

```
p <- 1/2
N <- 50*2**(0:7)
M <- floor(N*p)
n <- 5

tab <- matrix(0,length(N),n+1)
for(i in 1:length(N)){

  tab[i,] <- dhyper(0:n,m = M[i], n = N[i] - M[i], k = n)

}

tab <- rbind(tab,dbinom(0:n,n,p))
rownames(tab) <- c(N,Inf)
colnames(tab) <- 0:n
tab
```

```
              0         1         2         3         4          5
50   0.02507599 0.1492618 0.3256622 0.3256622 0.1492618 0.02507599
100  0.02814225 0.1529470 0.3189108 0.3189108 0.1529470 0.02814225
200  0.02969161 0.1546438 0.3156646 0.3156646 0.1546438 0.02969161
400  0.03046975 0.1554579 0.3140723 0.3140723 0.1554579 0.03046975
800  0.03085962 0.1558567 0.3132837 0.3132837 0.1558567 0.03085962
1600 0.03105475 0.1560540 0.3128912 0.3128912 0.1560540 0.03105475
3200 0.03115236 0.1561522 0.3126955 0.3126955 0.1561522 0.03115236
6400 0.03120118 0.1562011 0.3125977 0.3125977 0.1562011 0.03120118
Inf  0.03125000 0.1562500 0.3125000 0.3125000 0.1562500 0.03125000
```

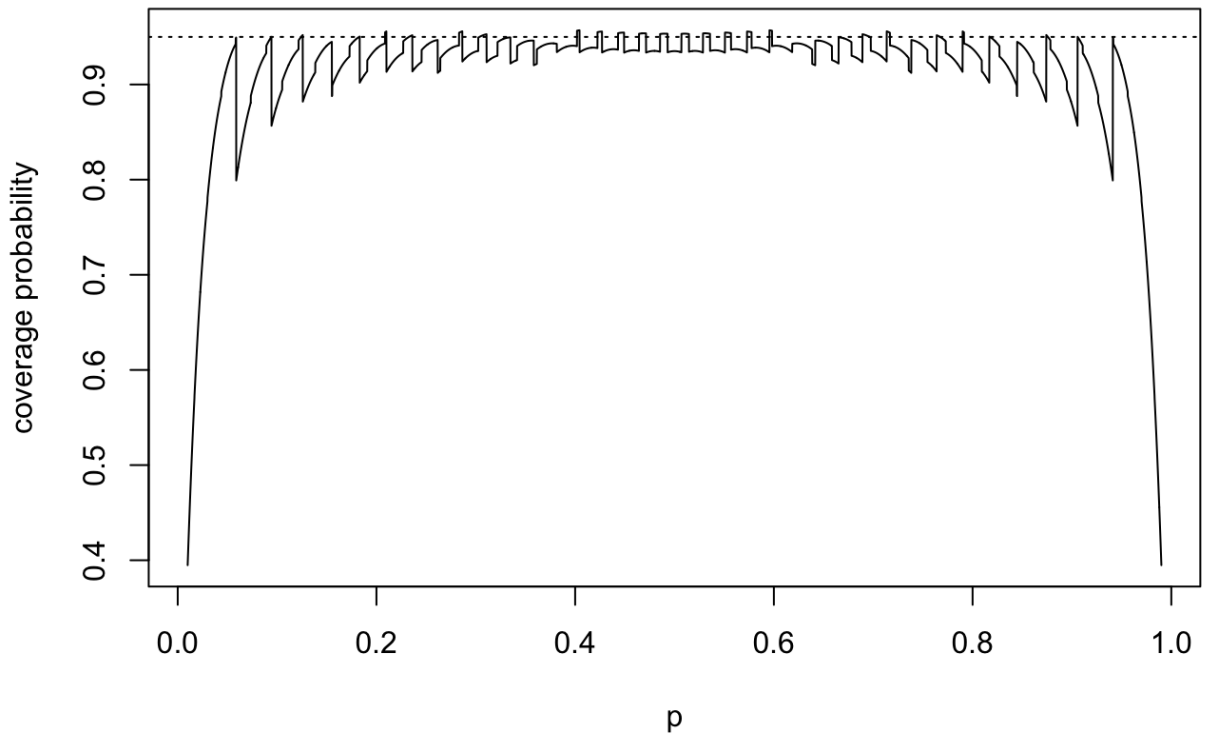
4. If $X_1, \ldots, X_n$ is a random sample from a $\text{Bernoulli}(p)$ distribution, the Wald-type $(1-\alpha) \times 100\%$ confidence interval for p is the interval

$$\hat{p}_n \pm z_{\alpha/2} \sqrt{\frac{\hat{p}_n(1-\hat{p}_n)}{n}},$$

where $\hat{p}_n = Y/n$, with $Y = X_1 + \cdots + X_n$, and $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal distribution. For a given sample size n and significance level α, the code below computes, over a sequence of values

of p the exact probability that this interval will contain the true value of p. It uses the fact that Y has the $\text{Binomial}(n, p)$ distribution. Read the code and try writing a comment for each line to explain what it is doing.

```
n <- 50
alpha <- 0.05
y <- 0:n
z_val <- qnorm(1-alpha/2)

py <- y/n
spy <- sqrt(py*(1-py)/n)
up <- py + z_val * spy
lo <- py - z_val * spy

p <- seq(0.01,0.99, by = 0.0001)
cp <- numeric(length(p))
for(i in 1:length(p)){

  ind <- (lo <= p[i]) & (up >= p[i])
  cp[i] <- sum(dbinom(y[ind],n,p[i]))

}

plot(cp ~ p,
     type = "l",
     ylab = "coverage probability")
abline(h = 1-alpha, lty = 3)
```



References

Mohr, Donna L, William J Wilson, and Rudolf J Freund. 2021. *Statistical Methods*. Academic Press.
Robert, C. P., and G. Casella. 2004. *Monte Carlo Statistical Methods*. Springer Verlag.