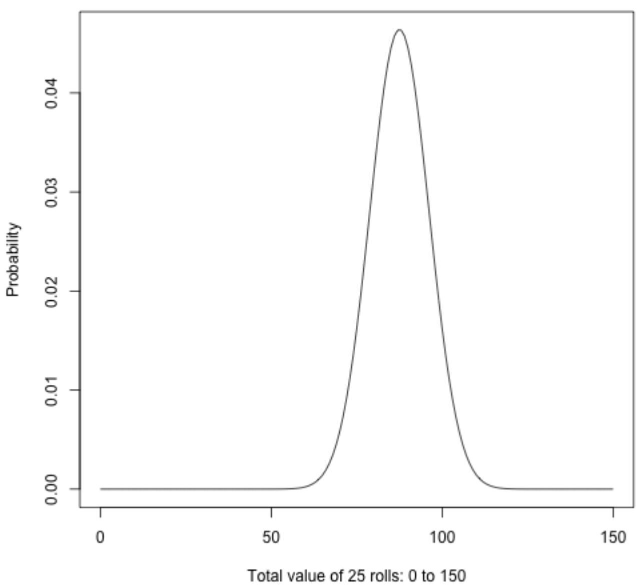
```
#Navdeep Gill
#Assignment 4

#Convolution function

conv <- function(x,y)
{ # function starts

  #Error Checking
  if ( !is.vector(x) )  { stop("parameter must be a vector") }
  if ( !is.numeric(x) ) { stop("parameter must be numeric") }
  if ( !is.vector(y) )  { stop("parameter must be a vector") }
  if ( !is.numeric(y) ) { stop("parameter must be numeric") }

  #Get relevant lengths of vectors x & y
  n = length(x)
  m = length(y)

  #Make a new sequence a where new.x is a vector of x flipped from n:1 and the rest are 0's
  new.x = c(x[n:1],rep(0,(m-1)))

  #Make a new sequence new.y where new.y is a vector of 0's and then y
  new.y = c( rep(0,(n-1)),y )

  #Get relevant lengths k (only need to get of a since length(new.x) = length(new.y))
  k = (length(new.x))

  #Allocate vector for convolution
  conv.vec = numeric(k)

  #Start loop to calculate convolution
  for (i in 1:k)
  { #i
    conv.vec[i]=sum(new.x*new.y)

    #Re allocate new.x for next iteration in loop. This is essentially shifting, getting the
overlap, multiplying, and adding
    #across the overlap.
    new.x = c(0,new.x)
    new.x = new.x[1:k]
  }#i
  return(conv.vec)
} # function ends

#####################################################################################################


#1

#Let Y be the sum of 25 throws of a die. Use your conv() function to calculate the pdf of Y.
Using
#only the pdf of Y, calculate E[Y] and Var(Y). You can do this using the definition of
expectation
#and variance. Plot the pdf of Y. Calculate P( 79 <= Y <= 96) and also P( 70 <= Y <= 105).

#Define two vectors for convolution function
rolling = c(0,rep(1/6,6))
dice = c(0, rep(1/6,6))

#Convolve 25 rolls of a die. So, if X1 = one roll of a die, then Y = X1+X2+...+X25
  for (i in 1:24)
  {
    rolling = conv(rolling, dice)
  }
  rolling
```

```
##   [1] 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
##   [8] 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
##  [15] 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
##  [22] 0.000e+00 0.000e+00 0.000e+00 0.000e+00 3.517e-20 8.793e-19 1.143e-17
##  [29] 1.029e-16 7.202e-16 4.177e-15 2.088e-14 9.247e-14 3.697e-13 1.354e-12
##  [36] 4.594e-12 1.457e-11 4.350e-11 1.230e-10 3.309e-10 8.504e-10 2.096e-09
##  [43] 4.966e-09 1.135e-08 2.505e-08 5.355e-08 1.110e-07 2.236e-07 4.382e-07
##  [50] 8.364e-07 1.557e-06 2.828e-06 5.019e-06 8.712e-06 1.480e-05 2.461e-05
##  [57] 4.011e-05 6.410e-05 1.005e-04 1.547e-04 2.337e-04 3.469e-04 5.061e-04
##  [64] 7.259e-04 1.024e-03 1.421e-03 1.941e-03 2.609e-03 3.453e-03 4.501e-03
##  [71] 5.778e-03 7.310e-03 9.111e-03 1.119e-02 1.356e-02 1.618e-02 1.905e-02
##  [78] 2.211e-02 2.532e-02 2.858e-02 3.184e-02 3.498e-02 3.791e-02 4.054e-02
##  [85] 4.277e-02 4.452e-02 4.573e-02 4.634e-02 4.634e-02 4.573e-02 4.452e-02
##  [92] 4.277e-02 4.054e-02 3.791e-02 3.498e-02 3.184e-02 2.858e-02 2.532e-02
##  [99] 2.211e-02 1.905e-02 1.618e-02 1.356e-02 1.119e-02 9.111e-03 7.310e-03
## [106] 5.778e-03 4.501e-03 3.453e-03 2.609e-03 1.941e-03 1.421e-03 1.024e-03
## [113] 7.259e-04 5.061e-04 3.469e-04 2.337e-04 1.547e-04 1.005e-04 6.410e-05
## [120] 4.011e-05 2.461e-05 1.480e-05 8.712e-06 5.019e-06 2.828e-06 1.557e-06
## [127] 8.364e-07 4.382e-07 2.236e-07 1.110e-07 5.355e-08 2.505e-08 1.135e-08
## [134] 4.966e-09 2.096e-09 8.504e-10 3.309e-10 1.230e-10 4.350e-11 1.457e-11
## [141] 4.594e-12 1.354e-12 3.697e-13 9.247e-14 2.088e-14 4.177e-15 7.202e-16
## [148] 1.029e-16 1.143e-17 8.793e-19 3.517e-20
```

```
# Plot of roll
plot(0:150,rolling,type="l", ylab="Probability", xlab="Total value of 25 rolls: 0 to 150")
```




```
# Below is used to find expectation of the convolution
x = rolling
k = length(x)

expectation <- function(x)
{
  for (i in 1:k)
  {
    x[i] = (x[i]*(i-1))
  }
  return(x)
}
expected = expectation(x)
expected.value = sum(expected)
expected.value
```

```
## [1] 87.5
```

```
#Below is used to find variance of convolution
v = rolling
k = length(v)
variation <- function(v)
{
  for (i in 1:k)
  {
    v[i] = (v[i]*(i-1)^2)
  }
  return(v)
}
variant = variation(v)
var.conv = sum(variant)
variance = var.conv - (expected.value)^2
variance
```

```
## [1] 72.92
```

```
#Before we find probabilities, we need the standard deviation
standard.dev = sqrt(variance)
standard.dev
```

```
## [1] 8.539
```

```
# P( 79 <= Y <= 96)

prob1 = pnorm(96, mean=expected.value, sd=standard.dev)

prob2 = pnorm(79, mean=expected.value, sd=standard.dev)
actualprob = prob1 - prob2
actualprob
```

```
## [1] 0.6805
```

```
#P( 70 <= Y <= 105)

prob3 = pnorm(105, mean=expected.value, sd=standard.dev)
prob4 = pnorm(70, mean=expected.value, sd=standard.dev)
actualprob2  = prob3 - prob4
actualprob2
```

```
## [1] 0.9596
```

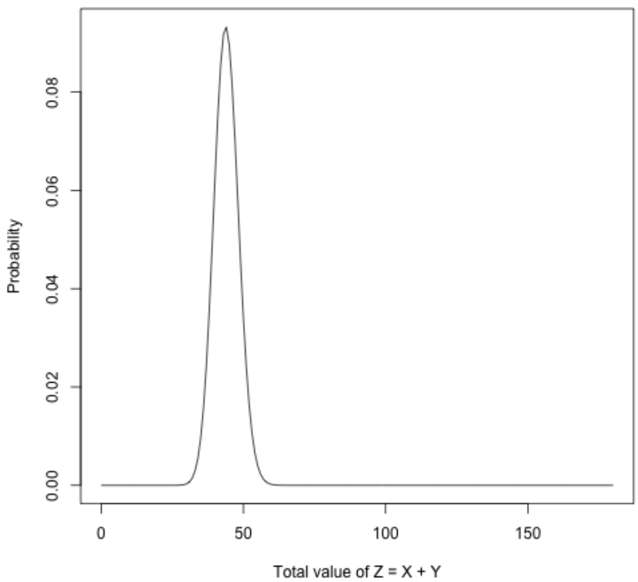
```
####################################################################################################

#2

#X ~ binom(40, 0.8) and Y ~poisson(12). X and Y are independent. Z = X + Y. Use your conv()
#function to calculate the pdf of Z. Using the pdf of Z you calculated, calculate E[Z] and
Var(Z).
#Plot the pdf of Z. Calculate P( Z <= 40 ). Since Y is Poisson, the support of Z is semi-
infinite.
#When convolving X and Y, it will suffice (actually more than suffice) to truncate the pdf of Y
at
#200.

#Define binomial(40,.8) and poisson(12)
n = 40
p = 0.8
lamda = 12
x = dbinom(0:n,n,p) # pdf of values over support
y = dpois(0:140,12)

#Get convolution for Z = x + y
Z = conv(x,y)

# Below is used to find expectation of the convolution
x = Z
k = length(x)
expectation <- function(x)
{
  for (i in 1:k)
  {
    x[i] = (x[i]*(i-1))
  }
  return(x)
}
expected = expectation(x)
expected.value = sum(expected)
expected.value
```

```
## [1] 44
```

```
#Below is used to find variance of convolution
v = Z
k = length(v)
variation <- function(v)
{
  for (i in 1:k)
  {
    v[i] = (v[i]*(i-1)^2)
  }
  return(v)
}
variant = variation(v)
var.conv = sum(variant)
variance = var.conv - (expected.value)^2
variance
```

```
## [1] 18.4
```

```
#Plot of pdf,Z
plot(0:180,Z,type="l", ylab="Probability", xlab="Total value of Z = X + Y")
```




```
#Before we find probabilities, we need the standard deviation
standard.dev = sqrt(variance)
standard.dev
```

```
## [1] 4.29
```

```
#Find probabilit Z<=40
prob.z = pnorm(40, mean=expected.value, sd=standard.dev)
prob.z
```

```
## [1] 0.1755
```

```
#####################################################################################################

#3
#Go to the Coupon6.R script that is included in the Blackboard Course Materials. It starts with a
#function that simulates the number of times sample is called on a vector of length 3 until all
#permutations of (1,2,3) are generated. Change the code to find the number of times sample
#must be called to generate all the permutations of (1,2,3,4). Making simple changes such as this
#to existing code is a job skill.

############
#
#    coupon6 function
#
#    Specialized to 6 coupons
#
#    This code is not good for a general number of coupons
#    It gives an idea of one kind of strategy that may work
#    very well in other applications
#
#    Simulates the distribution of number of coupons bought
#    to acquire at least one of each coupon type.

#    Another interpretation is: how many permutations from sample()
#    do we compute before we have computed all 6 permuations?
#############

coupon6 <- function(n)   # n is number of simulations
{ # begin coupon 6

  t = 0   # number of coupons bought
  v = numeric(313)
  #   We represent the sample as a base 4 number
  #   Each permutation of (1,2,3,4) represents a coupon
  #  (1,2,3,4) maps to 64*1 + 16*2 + 4*3 +4 => v[112]
  #  (1,2,4,3) maps to 64*1 + 16*2 + 4*4 +3 => v[115]
  #  (1,4,2,3) maps to 64*1 + 16*4 + 4*2 +3 => v[139]
  #  (1,4,3,2) maps to 64*1 + 16*4 + 4*3 +2 => v[142]
  #  (1,3,4,2) maps to 64*1 + 16*3 + 4*4 +2 => v[130]
  #  (1,3,2,4) maps to 64*1 + 16*3 + 4*2 +4 => v[124]
  #  (2,3,1,4) maps to 64*2 + 16*3 + 4*1 +4 => v[184]
  #  (2,3,4,1) maps to 64*2 + 16*3 + 4*4 +1 => v[193]
  #  (2,4,3,1) maps to 64*2 + 16*4 + 4*3 +1 => v[205]
  #  (2,4,1,3) maps to 64*2 + 16*4 + 4*1 +3 => v[199]
  #  (2,1,4,3) maps to 64*2 + 16*1 + 4*4 +3 => v[163]
  #  (2,1,3,4) maps to 64*2 + 16*1 + 4*3 +4 => v[160]
  #  (3,1,2,4) maps to 64*3 + 16*1 + 4*2 +4 => v[220]
  #  (3,1,4,2) maps to 64*3 + 16*1 + 4*4 +2 => v[226]
  #  (3,4,1,2) maps to 64*3 + 16*4 + 4*1 +2 => v[262]
  #  (3,4,2,1) maps to 64*3 + 16*4 + 4*2 +1 => v[265]
  #  (3,2,4,1) maps to 64*3 + 16*2 + 4*4 +1 => v[241]
  #  (3,2,1,4) maps to 64*3 + 16*2 + 4*1 +4 => v[232]
  #  (4,1,2,3) maps to 64*4 + 16*1 + 4*2 +3 => v[283]
  #  (4,1,3,2) maps to 64*4 + 16*1 + 4*3 +2 => v[286]
  #  (4,3,1,2) maps to 64*4 + 16*3 + 4*1 +2 => v[310]
  #  (4,3,2,1) maps to 64*4 + 16*3 + 4*2 +1 => v[313]
  #  (4,2,3,1) maps to 64*4 + 16*2 + 4*3 +1 => v[301]
  #  (4,2,1,3) maps to 64*4 + 16*2 + 4*1 +3 => v[295]


  p = c(4^3,4^2,4,1) #Add 4^3 for base 4
  #Redefine vector s based on above
  s = 
c(112,115,124,130,139,142,160,163,184,193,199,205,220,226,232,241,262,265,283,286,295,301,310,313)

  num.bought = numeric(n)
  for ( i in 1:n )
  { # for
    # each iteration of the for loop is one trial
    t    = 0           # number of coupons bought
    v[s] = c(rep(0,6)) # so far no coupon bought
    while ( prod(v[s]) == 0 ) # true while one coupon not obtained yet
    { # while
      t = t + 1
      x = sample(c(1,2,3,4),4,replace = F)
      v[sum(p*x)] = 1
    } # while
    num.bought[i] = t
  } # for
  return(num.bought)
} # end coupon 6

n = 5000
b = coupon6(n)  # n trials
print(mean(b))
```

```
## [1] 91.01
```

```
dev.new()
plot(table(b)/n, main = "Simulation PDF for\n obtaining 6 coupons") # simulation pdf
```