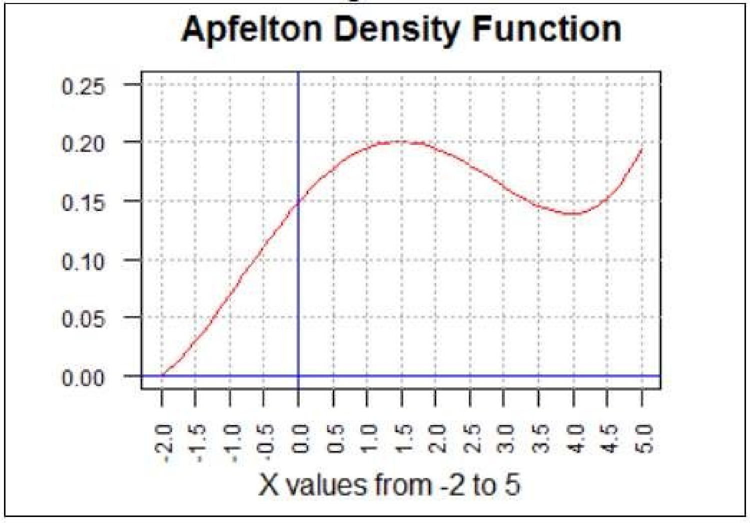
Topic 11c: Probability: Continuous Cases

Some word of warning is due here. This section presents useless material in the sense the Apfelton and Blumenkopf distributions are a figment of my imagination. The goal here is to demonstrate probability distributions for continuous function. We can see what they are like, how to use them, how to read values from tables, and how to use R functions to get values. In future topics we will encounter real continuous probability distributions and then we can apply all that we learn here to those functions.

Point One: Continuous probability distributions are different from discrete probability distributions. In the discrete case we might ask P(X = 2) but in a continuous case the probability that X is equal to any specific value is 0. All we can do is to ask for the probability that X is less than a value, or X is greater than a value, or X is between two values, or X is less than one value or it is greater than a different value. A small consequence of this distinction is that in these situations $P(X<2) = P(X\leq 2)$ since the $P(X=2)=0$.

Here is a graph of the Apfelton probability distribution.

Figure 1



The really important points here are that the function is defined for x values in the interval from -2 to 5 and that the area under the curve, and above the x-axis, from -2 to 5 is exactly 1. Therefore, for any value, say 3, between -2 and 5, inclusive, we can take the area under the curve and to the left of the line y=3 and make that area the $P(X<3)$.

Look at the Apfelton distribution link on the web page.

With the probability table available we can solve all four types of problems:

$P(X < 3) =$

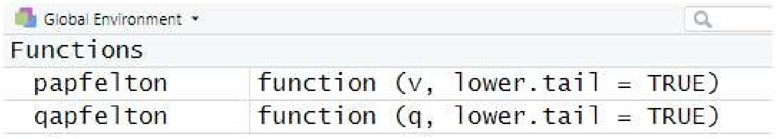
$P(-1.45 < X < 1.63) =$

$P(X > 3.4) =$

$P(X \leq 0.54 \text{ or } X \geq 2.04) =$

Now, instead of using the table, use the papfelton() function that we can load into our R envirionment.

```
     # first load the papfelton() function
     #
     #  this will actually loaded both papfelton() and
     #  qapfelton()
source("../apfelton.R")
```



```
Functions
  papfelton      function (v, lower.tail = TRUE)
  qapfelton      function (q, lower.tail = TRUE)
```

```
papfelton( 3 )           # solves P(X<3)
1 - papfelton( 3.4 )     # solves P(X>3.4)
     # or, using the alternative approach
papfelton( 3.4, lower.tail=FALSE)  # also solves P(X>3.4)
papfelton( 1.63 ) - papfelton( -1.45 ) #solves P(-1.45<X<1.63)
papfelton(0.54) + (1-papfelton(2.04))  #solves P(X<0.54 or X>2.04)
     # or, using the alternative approach
papfelton(0.54)+papfelton(2.04,lower.tail=FALSE)
```

```
>      #  Now solve the problems
> papfelton( 3 )           # solves P(X<3)
[1] 0.6957328
> 1 - papfelton( 3.4 )     # solves P(X>3.4)
[1] 0.2421646
>      # or, using the alternative approach
> papfelton( 3.4, lower.tail=FALSE)  # also solves P(X>3.4)
[1] 0.2421646
> papfelton( 1.63 ) - papfelton( -1.45 ) #solves P(-1.45<X<1.63)
[1] 0.4340971
> papfelton(0.54) + (1-papfelton(2.04))  #solves P(X<0.54 or X>2.04)
[1] 0.7073376
>      # or, using the alternative approach
> papfelton(0.54)+papfelton(2.04,lower.tail=FALSE)
[1] 0.7073376
```

So far we have seen problems where we are given a value, x, and then we are asked to find the probability such as getting that value or less, P(X≤x). We can solve this using the table of probabilities, or by using papfelton().

How can we solve a problem such as "Find the x value such that P(X≤x)=0.35?" We can find that answer by using the table "backwards."

Find y such that P(X≤y) = 0.35. Answer: y = ____________

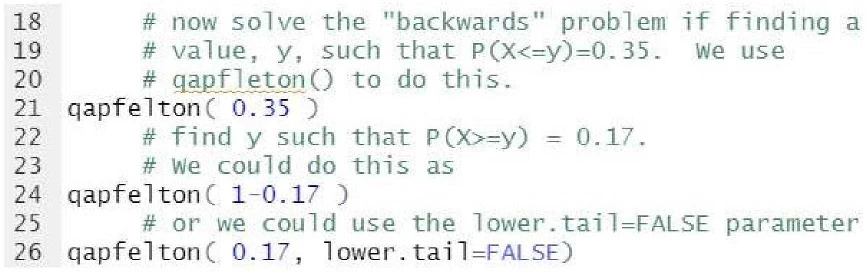
Between 1.16 and 1.17.
Probably close to 1.166.

Find y such that P(X≥y) = 0.17. To do this we have to remember that if P(X≥y)=0.17 then P(X<y) must be 1.0 - 0.17 = 0.83. Then we can use the table to find y such that P(X<y)=0.83. That y is the answer to the original problem.

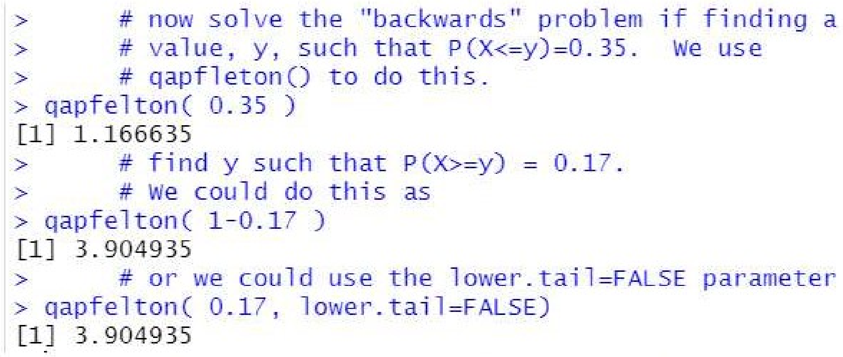
y = _______________

Between 3.90 and 3.91, almost certainly close to 3.905.

We could use papfelton() to find these answers by doing a successive approximation. That would be very inefficient for us, though not bad for a computer. Fortunately, we have a different function, qapfelton(), that will do this "backwards" problem for us. Here is how we can solve the two problems using qapfelton().

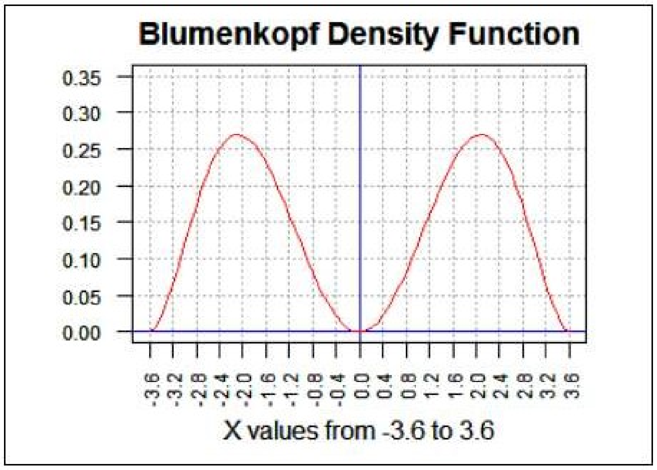


```
      # now solve the "backwards" problem if finding a
      # value, y, such that P(X<=y)=0.35.  We use
      # qapfleton() to do this.
qapfelton( 0.35 )
      # find y such that P(X>=y) = 0.17.
      # We could do this as
qapfelton( 1-0.17 )
      # or we could use the lower.tail=FALSE parameter
qapfelton( 0.17, lower.tail=FALSE)
```



```
>      # now solve the "backwards" problem if finding a
>      # value, y, such that P(X<=y)=0.35.  We use
>      # qapfleton() to do this.
> qapfelton( 0.35 )
[1] 1.166635
>      # find y such that P(X>=y) = 0.17.
>      # We could do this as
> qapfelton( 1-0.17 )
[1] 3.904935
>      # or we could use the lower.tail=FALSE parameter
> qapfelton( 0.17, lower.tail=FALSE)
[1] 3.904935
```

As part of this introduction we move onto the Blumenkopf probability distribution. It has the following graph.



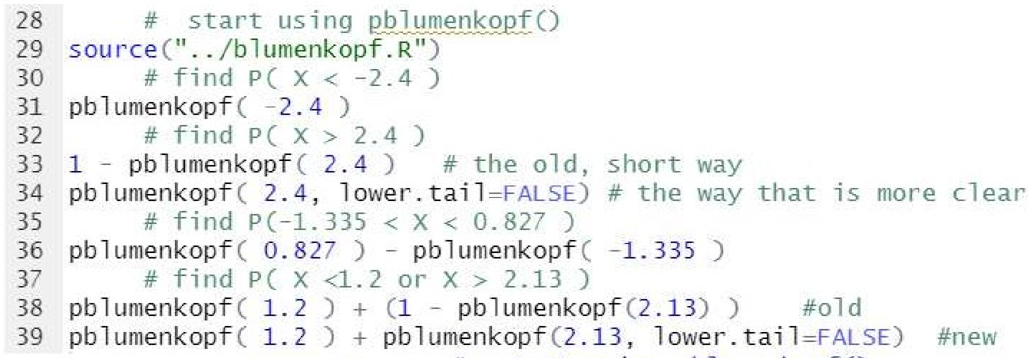
Here we see that the Blumenkopf distribution is defined for x being between -3.6 and 3.6. The area under the curve and above the x-axis is 1.0. Again, the P(X<-2.4) is the area under the curve and to the left of -2.4. Also, and this is an important distinction, the Blumenkopf distribution is symmetric. Thus, the P(X<-2.4) = P(X>2.4). We have a table of values for the Blumenkopf distribution (see the web page link). And, fortunately, we can load the pblumenkopf() and qblumenkopf() functions.

P(X< -2.4) =

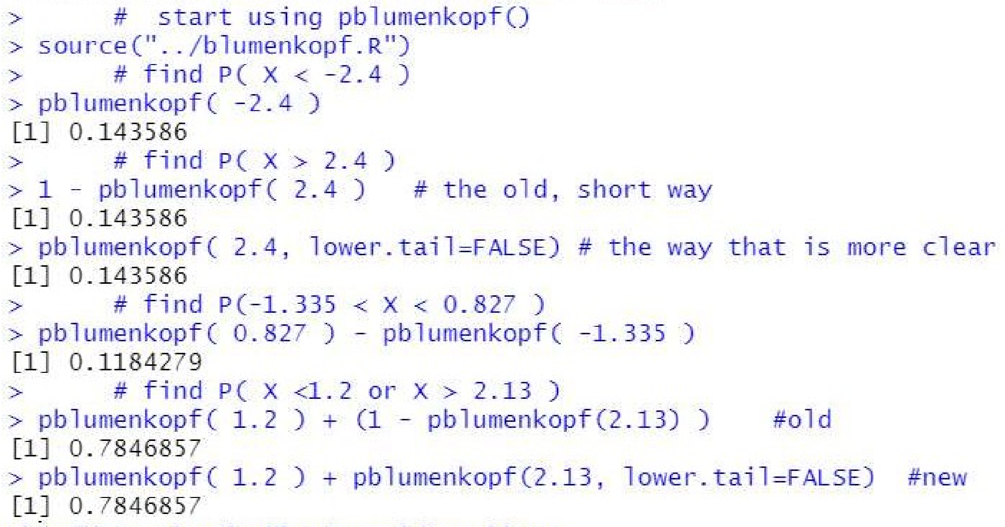
P(X> 2.4) =

P(-1.335 < X < 0.827) =

P(X < 1.2 or X > 2.13) =



```
     #  start using pblumenkopf()
source("../blumenkopf.R")
     # find P( X < -2.4 )
pblumenkopf( -2.4 )
     # find P( X > 2.4 )
1 - pblumenkopf( 2.4 )   # the old, short way
pblumenkopf( 2.4, lower.tail=FALSE) # the way that is more clear
     # find P(-1.335 < X < 0.827 )
pblumenkopf( 0.827 ) - pblumenkopf( -1.335 )
     # find P( X <1.2 or X > 2.13 )
pblumenkopf( 1.2 ) + (1 - pblumenkopf(2.13) )    #old
pblumenkopf( 1.2 ) + pblumenkopf(2.13, lower.tail=FALSE)  #new
```



```
>      #  start using pblumenkopf()
> source("../blumenkopf.R")
>      # find P( X < -2.4 )
> pblumenkopf( -2.4 )
[1] 0.143586
>      # find P( X > 2.4 )
> 1 - pblumenkopf( 2.4 )   # the old, short way
[1] 0.143586
> pblumenkopf( 2.4, lower.tail=FALSE) # the way that is more clear
[1] 0.143586
>      # find P(-1.335 < X < 0.827 )
> pblumenkopf( 0.827 ) - pblumenkopf( -1.335 )
[1] 0.1184279
>      # find P( X <1.2 or X > 2.13 )
> pblumenkopf( 1.2 ) + (1 - pblumenkopf(2.13) )    #old
[1] 0.7846857
> pblumenkopf( 1.2 ) + pblumenkopf(2.13, lower.tail=FALSE)  #new
[1] 0.7846857
```

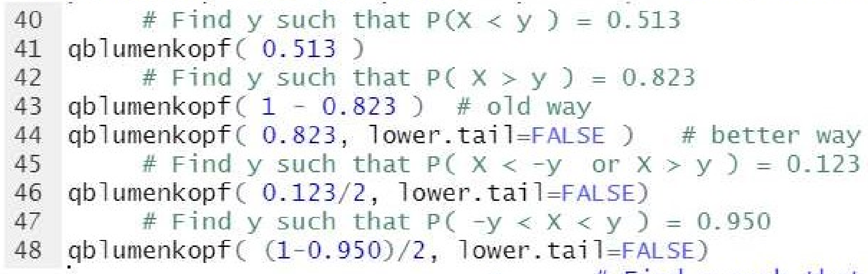
And we can use qblumenkopf( ) to solve the "backwards" problems.

Find y such that P(X < y) =0.513:

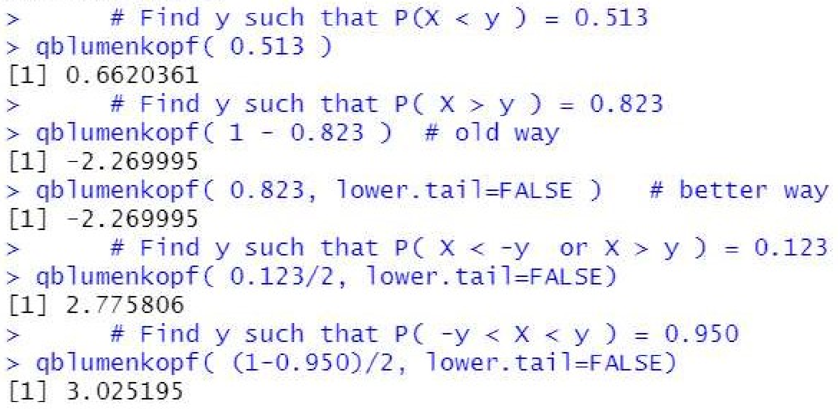
Find y such that P(X > y) = 0.823:

Find y such that P(X < -y or X > y) = 0.123:

Find y such that P(-y < X < y) = 0.950:



```
     # Find y such that P(X < y ) = 0.513
qblumenkopf( 0.513 )
     # Find y such that P( X > y ) = 0.823
qblumenkopf( 1 - 0.823 )  # old way
qblumenkopf( 0.823, lower.tail=FALSE )   # better way
     # Find y such that P( X < -y  or X > y ) = 0.123
qblumenkopf( 0.123/2, lower.tail=FALSE)
     # Find y such that P( -y < X < y ) = 0.950
qblumenkopf( (1-0.950)/2, lower.tail=FALSE)
```



```
>      # Find y such that P(X < y ) = 0.513
> qblumenkopf( 0.513 )
[1] 0.6620361
>      # Find y such that P( X > y ) = 0.823
> qblumenkopf( 1 - 0.823 )  # old way
[1] -2.269995
> qblumenkopf( 0.823, lower.tail=FALSE )   # better way
[1] -2.269995
>      # Find y such that P( X < -y  or X > y ) = 0.123
> qblumenkopf( 0.123/2, lower.tail=FALSE)
[1] 2.775806
>      # Find y such that P( -y < X < y ) = 0.950
> qblumenkopf( (1-0.950)/2, lower.tail=FALSE)
[1] 3.025195
```