Naren Suri,

Exploratory Data Analysis,

Homework -1

Date : 9/9/2015.

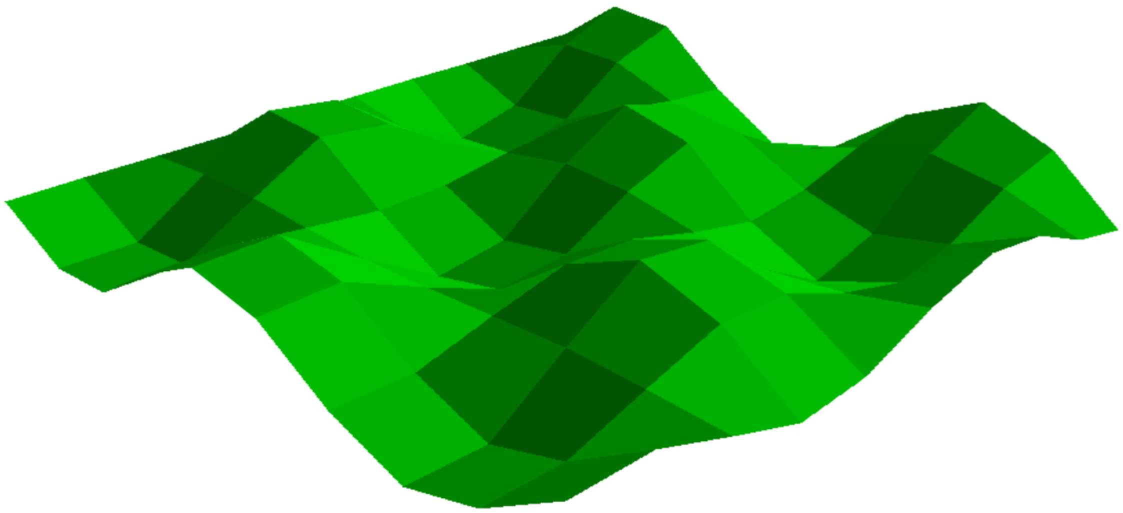
The purpose of the first two problems is to practice using R effectively. Pretend you are teaching a Calculus class and you want to give your students a graphical feeling for the gradient of a bivariate function. Let us take the bivariate function

$$z = f(x, y) = cos(x)sin(y)$$

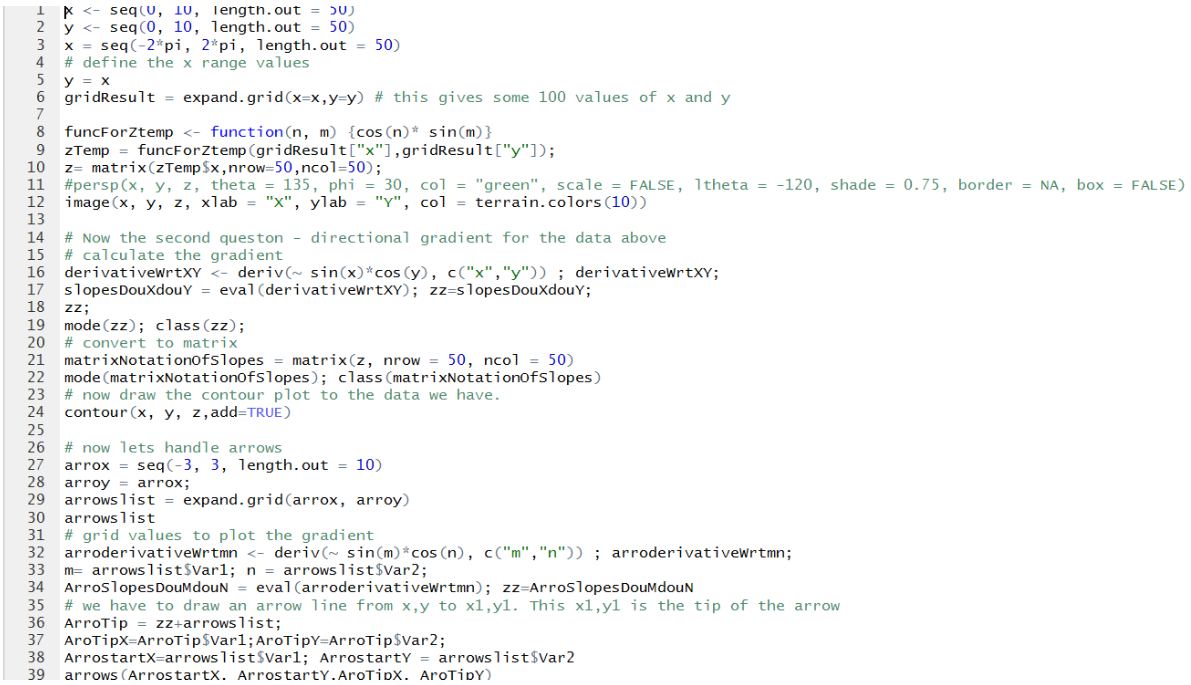
1. Create a function in R which will produce either an image or perspective plot of the bivariate function $f(x,y)$ over a specified x and y range. Hint: You can use the function expand.grid(a, b) in R to create a grid of (x, y) values for which you can create the plot.

1.

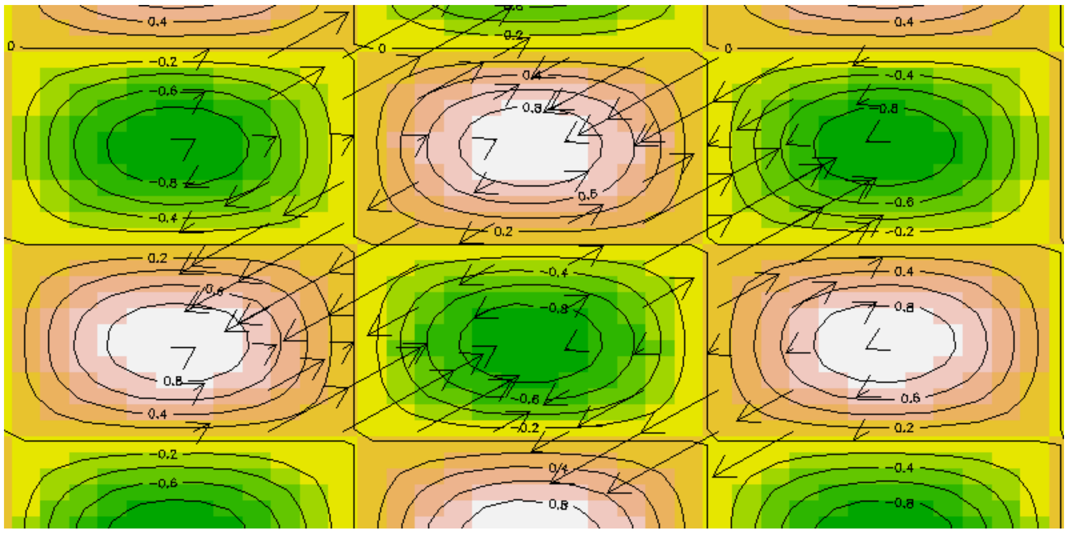
```
# First Question starts here. plot a perspective 3d gaph with some random x and y values, using expand.grid to feed the perspective plot
#Generate a random gid of x,y and prepare the grid values.
# then we will send these values to the function f(x,y) = cos(x)*sin(y) = Z.
#store the return of f() in to Z, as stated above
# The persp, ?persp, tells you that it takes x,y(your inputs to function f) and Z (your resultant of f(x,y), and limit, etc..)
# use various args in the persp(), to draw a 3-D plot.
# Now, getting in to the action
x <- seq(0, 10, length.out = 10)
y <- seq(0, 10, length.out = 10)
gridResult = expand.grid(x=x,y=y) # this gives some 100 values of x and y
gridResult
class(gridResult) # its a dataframe, so accessing it as gridResult["x"] will give a data frame. But,
# if you want to retrieve the values as vectors use, $ sign, gridResult$x , or gridResult[,1]
#mode(gridResult[,1]); class(gridResult[,1])

#The grid columns can be accessed using the gridResult["name of column"]
#example("persp"), it helps you to write a function and how to use it in the persp()
funcForZtemp <- function(n, m) {cos(n)* sin(m)}
# calculate Z for persp using the function created above
zTemp = funcForZtemp(gridResult["x"],gridResult["y"]);
z= matrix(zTemp$x,nrow=10,ncol=10);
# now we got the z, and we are good to plot the perspective using the x,y,z over some x,y,z limited range.
# lets draw a perspective, example("persp") helps.
#par(mfrow=c(1,2));

persp(x, y, z, theta = 135, phi = 30, col = "green", scale = FALSE, ltheta = -120, shade = 0.75, border = NA, box = FALSE)
```



2. Now create some R code which would allow the user to overlay some arrows on your image plot which point in the direction of the gradient of $f(x, y)$ for a sequence of test points $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$ For this exercise you can use the deriv() function in R along with the expand.grid(a, b) function.



```
x <- seq(0, 10, length.out = 50)
y <- seq(0, 10, length.out = 50)
x = seq(-2*pi, 2*pi, length.out = 50)
# define the x range values
y = x
gridResult = expand.grid(x=x,y=y) # this gives some 100 values of x and y

funcForZtemp <- function(n, m) {cos(n)* sin(m)}
zTemp = funcForZtemp(gridResult["x"],gridResult["y"]);
z= matrix(zTemp$x,nrow=50,ncol=50);
#persp(x, y, z, theta = 135, phi = 30, col = "green", scale = FALSE, ltheta = -120, shade = 0.75, border = NA, box = FALSE)
image(x, y, z, xlab = "X", ylab = "Y", col = terrain.colors(10))

# Now the second queston - directional gradient for the data above
# calculate the gradient
derivativeWrtXY <- deriv(~ sin(x)*cos(y), c("x","y")) ; derivativeWrtXY;
slopesDouXdouY = eval(derivativeWrtXY); zz=slopesDouXdouY;
zz;
mode(zz); class(zz);
# convert to matrix
matrixNotationOfSlopes = matrix(z, nrow = 50, ncol = 50)
mode(matrixNotationOfSlopes); class(matrixNotationOfSlopes)
# now draw the contour plot to the data we have.
contour(x, y, z,add=TRUE)

# now lets handle arrows
arrox = seq(-3, 3, length.out = 10)
arroy = arrox;
arrowslist = expand.grid(arrox, arroy)
arrowslist
# grid values to plot the gradient
arroderivativeWrtmn <- deriv(~ sin(m)*cos(n), c("m","n")) ; arroderivativeWrtmn;
m= arrowslist$Var1; n = arrowslist$Var2;
ArroSlopesDouMdouN = eval(arroderivativeWrtmn); zz=ArroSlopesDouMdouN
# we have to draw an arrow line from x,y to x1,y1. This x1,y1 is the tip of the arrow
ArroTip = zz+arrowslist;
AroTipX=ArroTip$Var1;AroTipY=ArroTip$Var2;
ArrostartX=arrowslist$Var1; ArrostartY = arrowslist$Var2
arrows(ArrostartX, ArrostartY,AroTipX, AroTipY)
```



3. For the next two problems use Data from UREDA page 31, problem 2(b), 2(c).

   (a) Give stem-and-leaf displays for each of those batches of data. Choose one dataset to present your display by hand using two lines per stem (justify whether this is suggested by the rules), and the other dataset using R function using different values for "scale".

   (b) Briefly describe the distributions from your displays (shape, symmetry, outliers, etc).

# stem and leaf

StemAndLeaf = c(88,66,71,63,101,55,76,49,63,38,91,79,41,36,73,55,42,49,50,90,51)

- stem and leaf plots are one way of repesneting the frequency of the elements in the data
- we can use the frequency tables, relative frequencies ec, but stem does the same in a different way

code:

stem(StemAndLeaf)

stem(StemAndLeaf, scale = 1, width = 110)

```
output : > stem(StemAndLeaf, scale = 1, width = 110)

  The decimal point is 1 digit(s) to the right of the |

   2 | 68
   4 | 12990155
   6 | 3361369
   8 | 801
  10 | 1
```

# 2 - stem plot

stem.leaf(StemAndLeaf,m=2)

stemLeaf2 = c(0.12,0.15,0.15,0.10,0.13,0.15,0.14,0.08,0.11,0.09,0.14,0.09,0.13,0.14,0.12,0.16,0.15,0.13,0.12,0.12,0.09)

stem(stemLeaf2)

stem(stemLeaf2, scale =0.5,width=25)

stem(stemLeaf2, scale =0.2,width=25)

stem(stemLeaf2, scale =1,width=25)

stem(stemLeaf2, scale =10,width=25)

output:

```
> stemLeaf2 = c(0.12,0.15,0.15,0.10,0.13,0.15,0.14,0.08,0.11,0.09,0.14,0.09,0.13,0.14,0.1
> stem(stemLeaf2)

  The decimal point is 2 digit(s) to the left of the |

   8 | 0000
  10 | 00
  12 | 0000000
  14 | 0000000
  16 | 0
This is of no real use, because it doesn't help in either fastly calculating mean, mode,
At least.

> stem(stemLeaf2, scale =0.5,width=25)

  The decimal point is 1 digit(s) to the left of the |

  0 | 8999
  1 | 012222333444
  1 | 55556

This shape is right skewed a bit.

> stem(stemLeaf2, scale =0.2,width=25)

  The decimal point is 1 digit(s) to the left of the |

  0 | 8999
  1 | 01222233344445
For this data, this plot gives no important information.
Plot gives stem size decently, but not very clear in explaining the data nature.
And always, data points those are very close are to be scaled up to see their real nature


> stem(stemLeaf2, scale =1,width=25)

  The decimal point is 2 digit(s) to the left of the |

   8 | 0000
  10 | 00
  12 | 0000000
  14 | 0000000
  16 | 0
This gives no value to the statistician. Always analyse the behavior of your data and
Then fix the better parameters that suits your data.

> stem(stemLeaf2, scale =10,width=25)

  The decimal point is 3 digit(s) to the left of the |

   80 | 0
   82 |
   84 |
   86 |
   88 |
   90 | 000
   92 |
   94 |
   96 |
   98 |
  100 | 0
  102 |
  104 |
  106 |
  108 |
  110 | 0
  112 |
  114 |
  116 |
  118 |
  120 | 0000
  122 |
  124 |
  126 |
  128 |
  130 | 000
  132 |
  134 |
  136 |
  138 |
  140 | 000
  142 |
  144 |
  146 |
  148 |
  150 | 0000
  152 |
  154 |
  156 |
  158 |
  160 | 0
```

This is no different than having the original data. Which gives no value to the reader.

Plotted many graphs to see how parameters impact their behaviour.

```
> #2-stem plot
```

stem.leaf

#Manual way: as ased in question

Since stem and leaf plots are to represent frequency, its always better to choose the decent number of buckets which can define the behavior of the data.

The bucketing should make our job easy to count the frequencies

# 88,66,71,63,101,55,76,49,63,38,91,79,41,36,73,55,42,49,50,90,51

Stem : Leaf

2 | 68

```
 4 | 12990155

 6 | 3361369

 8 | 801

10 | 1
```

2. Designing 2 stem model

   StemAndLeaf = c(88,66,71,63,101,55,76,49,63,38,91,79,41,36,73,55,42,49,50,90,51)
   stem(StemAndLeaf,scale=4)

   output :

```
 3  |  68
 4  |  12
 4  |  99
 5  |  01
 5  |  55
 6  |  33
 6  |  6
 7  |  13
 7  |  69
 8  |
 8  |  8
 9  |  01
 9  |
10  |  1
```

B. Can we modify the second stem plot?

The decimal point is 1 digit(s) to the left of the |

```
Stem  |  leaf

0    | 8999

1    | 012222333444

1    | 55556
```

This split makes it easy to count the frequencies, considering the decimal point location.

B. discuss about the Shapes of the stem leaf graphs:

(Shape, symmetry, outliers, etc

1. # Stem : Leaf

```
   2 | 68
   4 | 12990155
   6 | 3361369
   8 | 801
  10 | 1
```

This is more right skewed data.

In addition, this is not exactly symmetric. It looks more like a bell curve, a distribution curve.

The 10 value, as it was repeated only once it don't really affect the final frequency calculations.

3. Plot 2 shape

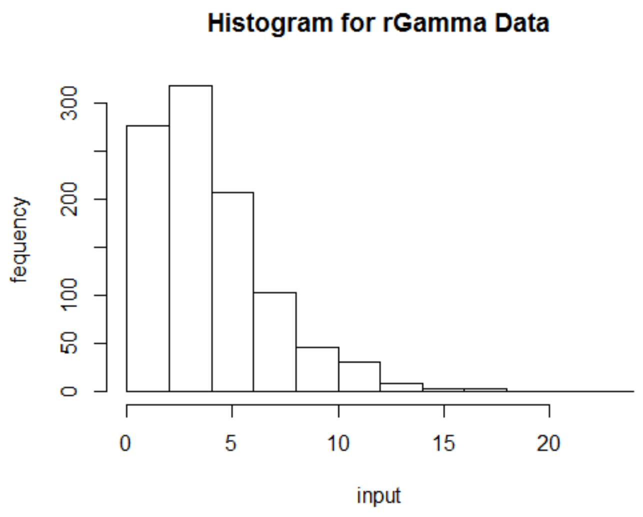
```
Stem   leaf
0    | 8999
1    | 012222333444
1    | 55556
```

This is more like symmetric, though it's not exactly symmetric in shape.

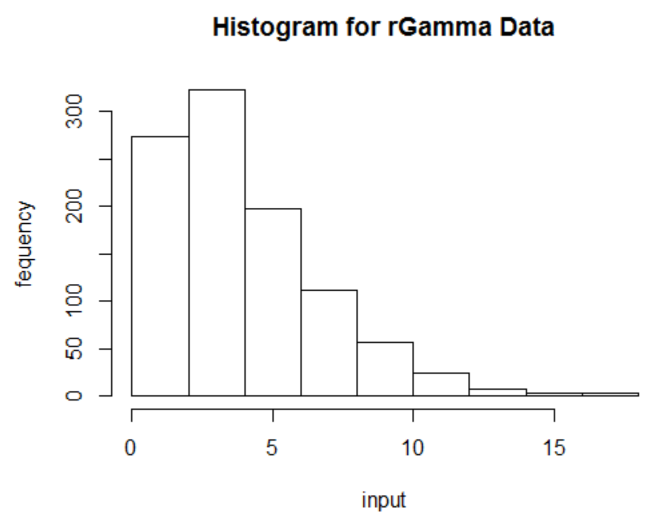
It has sharper center hill. Right skewed compared to let, but not to a big scale.

4. Generate 1000 data from Gamma distributions (R: rgamma()) with two different sets of parameter values picked by you. For each of the generated datasets
   (a) Produce a plot of the exact density function using the curve() function in R.
   (b) Produce QQ plots of the generated data and interpret. Your interpretation might include symmetry, shape, heavy/light tail(s), location of the median, etc.
   (c) Produce histogram plots of the data.
   (d) Produce non-parametric density plots using the density() function for three different kernel functions. Which kernel function do you think best represents the data?

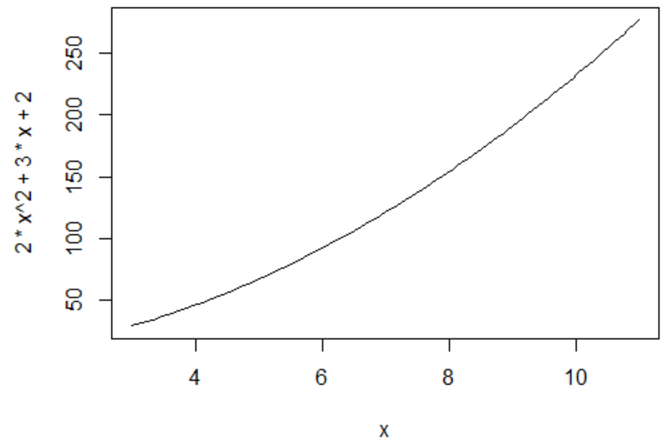
```
rgamma1 = rgamma(1000,shape=2,rate=0.5)

rgamma1


rgamma2 = rgamma(1000,shape=2,scale=2)


hist(rgamma1,main ="Histogram for rGamma Data",xlab = "input", ylab="fequency")
```



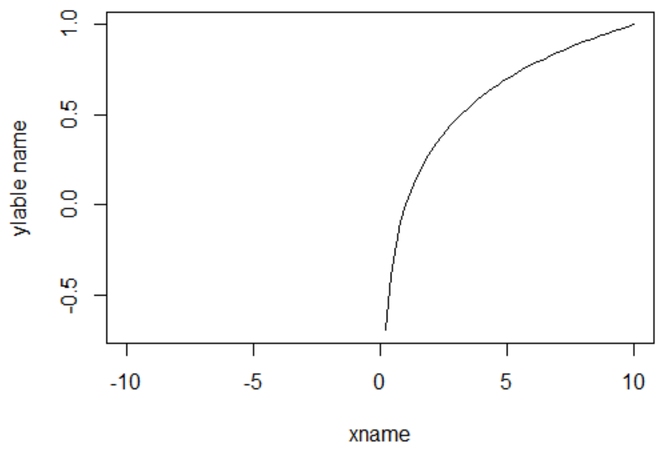
```
hist(rgamma2,main ="Histogram for rGamma Data",xlab = "input", ylab="fequency")
```



```
#-------------------------------------------------

curve(2*x^2+3*x+2, from = 11, to = 3)
```



```
curve(log10(x), from = -10, to = 10, add = FALSE, type = "l", xlab = "xname", ylab = "ylable name")
```

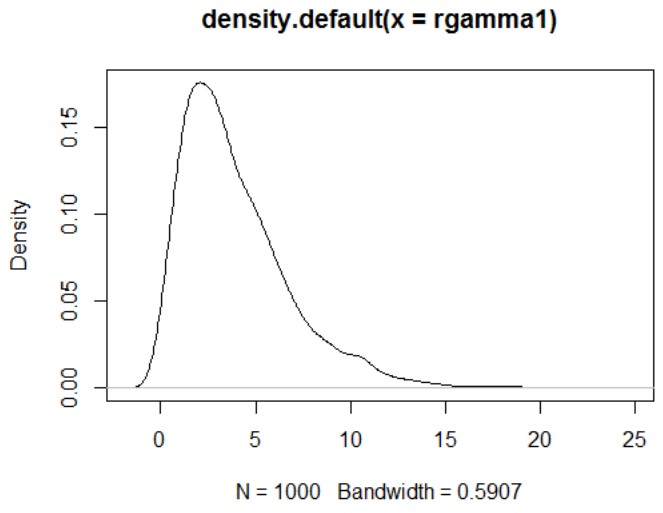


#---

# plotting the Kernel Density fnction

d <- density(rgamma1) # returns the density data

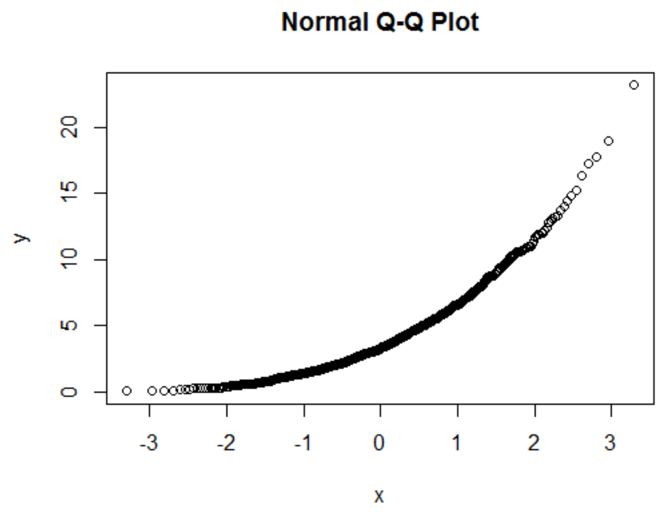
plot(d) # plots the results



#---

# QQ Plot

qqnorm(rgamma1,xlab = deparse(substitute(x)),ylab = deparse(substitute(y)))



This QQ plot for the data rgamma is actually giving clues about how well the data is distributed.

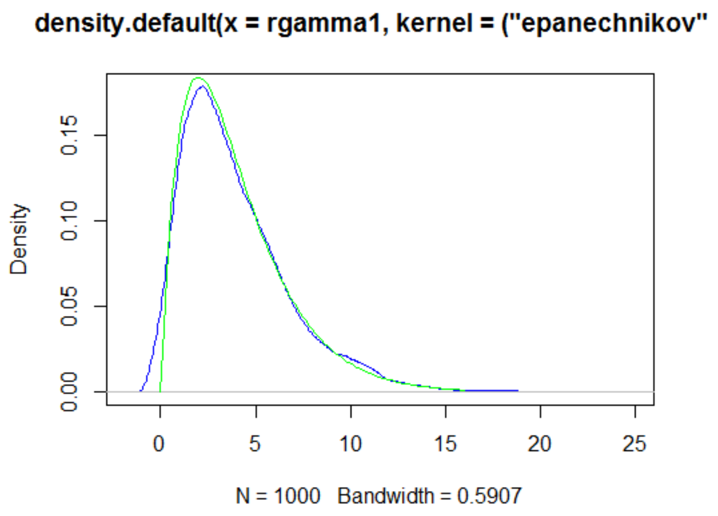
This has the heavy right tail behavior, we see the curve on the right side was going up, in different to the general normal distribution behavior.

# plotting with all the 6 different kernal options

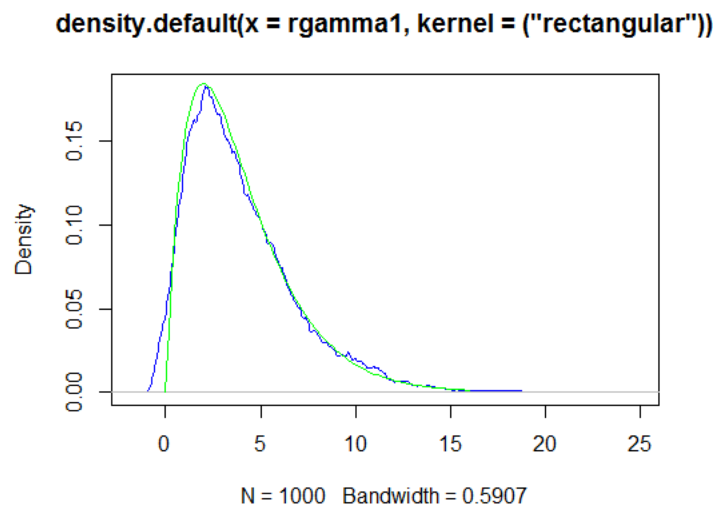
# --- kernal = epanechnikov

plot(density(rgamma1, kernel = ("epanechnikov")), col = "blue")

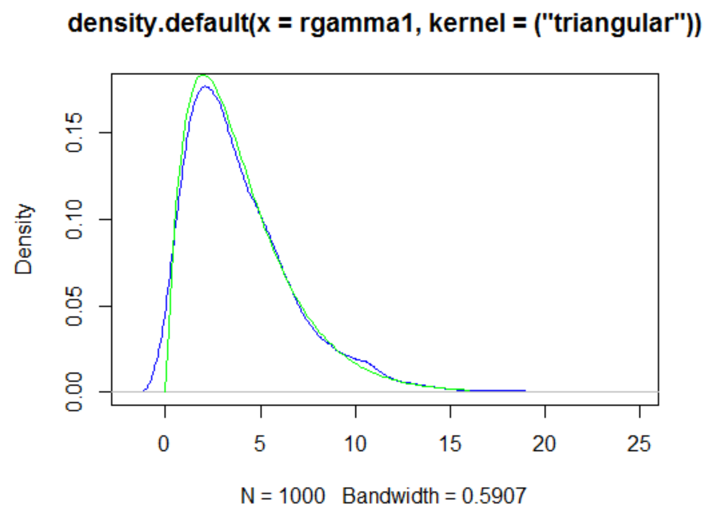
curve(dgamma(x,shape=2,rate=0.5), from = 0, to = 16, add = TRUE, col = "green")



```
# --- kernal = rectangular

plot(density(rgamma1, kernel = ("rectangular")), col = "blue")

curve(dgamma(x,shape=2,rate=0.5), from = 0, to = 16, add = TRUE, col = "green")
```

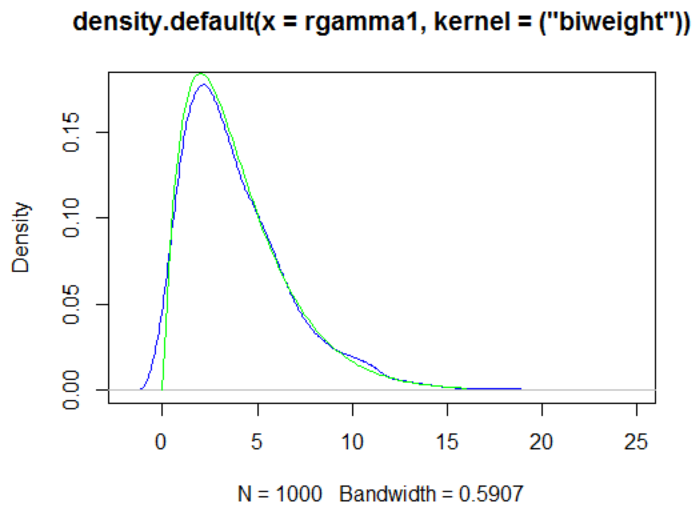


```
# --- kernal = triangular

plot(density(rgamma1, kernel = ("triangular")), col = "blue")

curve(dgamma(x,shape=2,rate=0.5), from = 0, to = 16, add = TRUE, col = "green")
```

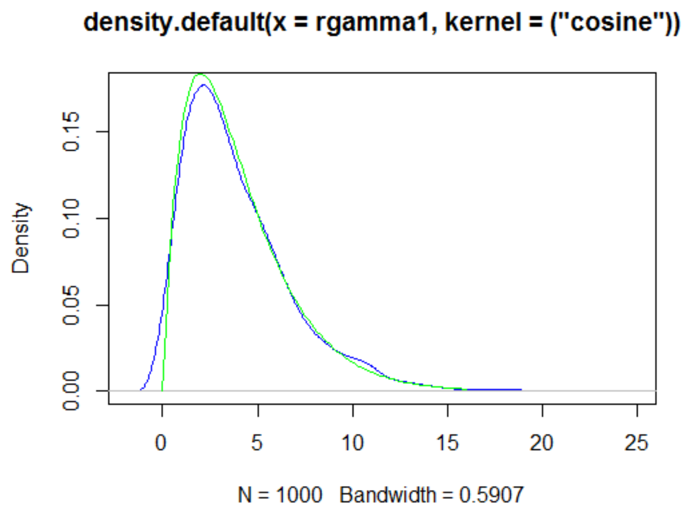


```
# --- kernal = biweight
```

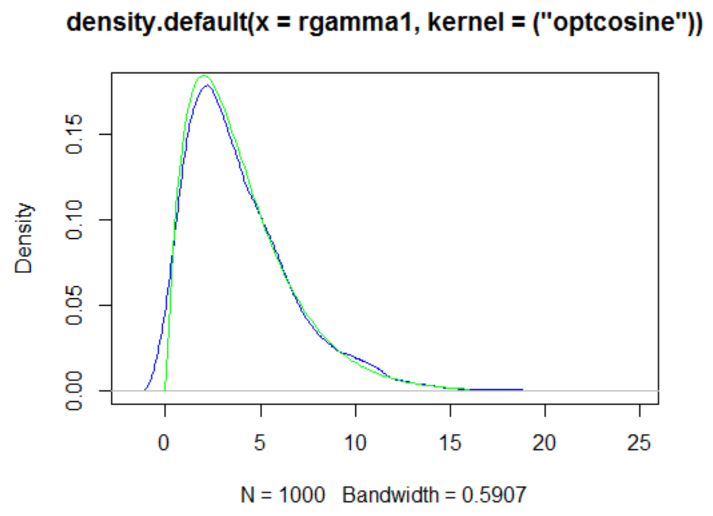
```
plot(density(rgamma1, kernel = ("biweight")), col = "blue")

curve(dgamma(x,shape=2,rate=0.5), from = 0, to = 16, add = TRUE, col = "green")
```



```
# --- kernal = cosine

plot(density(rgamma1, kernel = ("cosine")), col = "blue")

curve(dgamma(x,shape=2,rate=0.5), from = 0, to = 16, add = TRUE, col = "green")
```



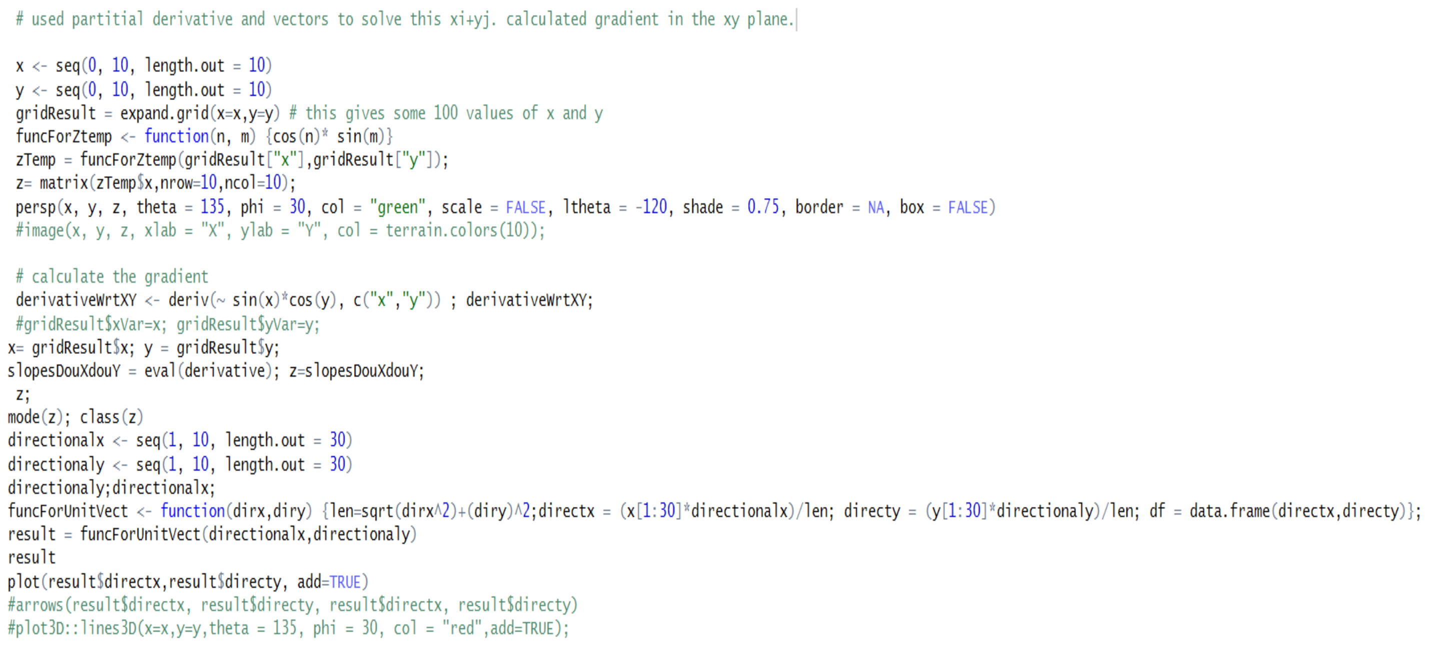
```
# --- kernal = optcosine

plot(density(rgamma1, kernel = ("optcosine")), col = "blue")

curve(dgamma(x,shape=2,rate=0.5), from = 0, to = 16, add = TRUE, col = "green")
```



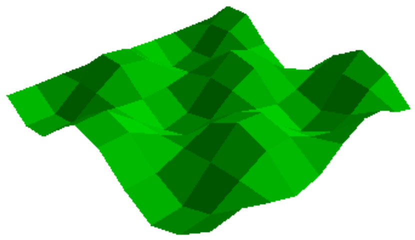
# the 6 different functions has the slight observable difference for the small set of data i have taken.

-

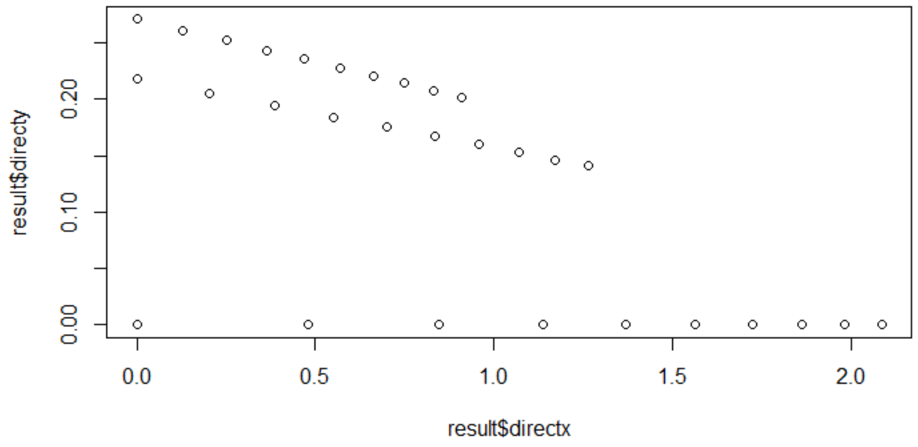
Tried 2nd question in a mathematical way to explore. Need to join the two plots below. Idea, taken pd for the equation. Then taking gradient on a 3d plot = infinite possible ways of drawing tangent. So drawn tangents with respect to x and y, the resultant vector is the z vector, which is the gradient at that particular point in that particular direction. So tried to implement that with own function, yet to discuss with prof to refine it further.



```
 # used partial derivative and vectors to solve this xi+yj. calculated gradient in the xy plane.

 x <- seq(0, 10, length.out = 10)
 y <- seq(0, 10, length.out = 10)
 gridResult = expand.grid(x=x,y=y) # this gives some 100 values of x and y
 funcForZtemp <- function(n, m) {cos(n)* sin(m)}
 zTemp = funcForZtemp(gridResult["x"],gridResult["y"]);
 z= matrix(zTemp$x,nrow=10,ncol=10);
 persp(x, y, z, theta = 135, phi = 30, col = "green", scale = FALSE, ltheta = -120, shade = 0.75, border = NA, box = FALSE)
 #image(x, y, z, xlab = "X", ylab = "Y", col = terrain.colors(10));

 # calculate the gradient
 derivativeWrtXY <- deriv(~ sin(x)*cos(y), c("x","y")) ; derivativeWrtXY;
 #gridResult$xVar=x; gridResult$yVar=y;
x= gridResult$x; y = gridResult$y;
slopesDouXdouY = eval(derivative); z=slopesDouXdouY;
 z;
mode(z); class(z)
directionalx <- seq(1, 10, length.out = 30)
directionaly <- seq(1, 10, length.out = 30)
directionaly;directionalx;
funcForUnitVect <- function(dirx,diry) {len=sqrt(dirx^2)+(diry)^2;directx = (x[1:30]*directionalx)/len; directy = (y[1:30]*directionaly)/len; df = data.frame(directx,directy)};
result = funcForUnitVect(directionalx,directionaly)
result
plot(result$directx,result$directy, add=TRUE)
#arrows(result$directx, result$directy, result$directx, result$directy)
#plot3D::lines3D(x=x,y=y,theta = 135, phi = 30, col = "red",add=TRUE);
```



I have computed the gadient using partial derivatives. Partial-f/dx ivector + partial-f/dy jvector. Computed the gradiant. The obtained gradient at the direction of random points chosen in the algorithm above.   I have drawn contor, but then want to visualize the 3d image with the grdients, Now I want to join the two graphs an indicate the arrows to the direction they were drawn. I have to discuss this model further with the prof.


result$directy
result$directx