Assignment 1

Christina Bisol 20046973

Question 1

In [1]:
```
#Question 1a)
#Using the derivative method to calculate the central value & uncertainty
#first I will introduce Sympy to calculate derivates
#This is similar to example 4.3 in the textbook
import sympy as s
s.init_printing()
x,y,sigma_x,sigma_y,sigma_xy=s.symbols('x y \sigma_x \sigma_y \sigma_xy')
#defined the variables as shown in example 4.2 and the covariance for partB
#Now I will define the given function as F to use later:
F=(s.sqrt(x**2+y**2))/s.ln((s.sqrt(x**2+y**2)))
#print out the function by leaving it in the last line
F
```

Out[1]:
$$\frac{\sqrt{x^2 + y^2}}{\log\left(\sqrt{x^2 + y^2}\right)}$$

In [2]:
```
#Now I will define the partial derivaties in the same manner
#first w respect to x and then w respect to y
dFdx=s.diff(F,x)
dFdy=s.diff(F,y)
dFdx #show the derivative function wrt to x
```

Out[2]:
$$\frac{x}{\sqrt{x^2 + y^2}\log\left(\sqrt{x^2 + y^2}\right)} - \frac{x}{\sqrt{x^2 + y^2}\log^2\left(\sqrt{x^2 + y^2}\right)}$$

In [3]:
```
dFdy #show the derivative function wrt to y
```

Out[3]:
$$\frac{y}{\sqrt{x^2 + y^2}\log\left(\sqrt{x^2 + y^2}\right)} - \frac{y}{\sqrt{x^2 + y^2}\log^2\left(\sqrt{x^2 + y^2}\right)}$$

In [4]:
```
#for part a) x & y are not correlated so I will use the quadratic sum given in
 4.22 of the txtbk
uF=s.sqrt((dFdx*sigma_x)**2+(dFdy*sigma_y)**2)
#where uF is the uncertainty of the function
uF #showing the quadratic sum for the error on F
```

Out[4]:

$$\sqrt{\sigma_x^2\left(\frac{x}{\sqrt{x^2+y^2}\log\left(\sqrt{x^2+y^2}\right)}-\frac{x}{\sqrt{x^2+y^2}\log^2\left(\sqrt{x^2+y^2}\right)}\right)^2+\sigma_y^2\left(\frac{}{\sqrt{x^2+}\right.}$$

In [5]:
```
#assignming given values for part a) into the symbols from before
values_a=[(x,11.1),(sigma_x,0.4),(y,5.4),(sigma_y,1.5)]
#and print out the outputs of the central value and uncertainty using the subs
 function
print("1A) f(x,y)={:.2f}+/-{:.2f}".format((F.subs(values_a)),(uF.subs(values_a
))))
```

```
1A) f(x,y)=4.91+/-0.18
```

In [7]:
```
#Now for part b, the values for x, y and their uncertainties are the same
#Using formula 4.24 from the txtbk to account for the covariance
#and define the given covariance
uF_mod=s.sqrt((dFdx*sigma_x)**2+(dFdy*sigma_y)**2+2*dFdx*dFdy*sigma_xy)
uF_mod
```

Out[7]:

$$\sqrt{\begin{aligned}&\sigma_x^2\left(\frac{x}{\sqrt{x^2+y^2}\log\left(\sqrt{x^2+y^2}\right)}-\frac{x}{\sqrt{x^2+y^2}\log^2\left(\sqrt{x^2+y^2}\right)}\right)^2\\&+\sigma_x y\left(\frac{2x}{\sqrt{x^2+y^2}\log\left(\sqrt{x^2+y^2}\right)}-\frac{2x}{\sqrt{x^2+y^2}\log^2\left(\sqrt{x^2+y^2}\right)}\right)\left(\frac{}{\sqrt{x^2+y^2}}\right.\\&+\sigma_y^2\left(\frac{y}{\sqrt{x^2+y^2}\log\left(\sqrt{x^2+y^2}\right)}-\frac{y}{\sqrt{x^2+y^2}\log^2\left(\sqrt{x^2+y^2}\right)}\right)^2\end{aligned}}$$

In [8]:
```
#asssigning values to variable for part b
values_b=[(x,11.1),(sigma_x,0.4),(y,5.4),(sigma_y,1.5),(sigma_xy,1.5)]
print("1b) f(x,y)={:.2f}+/-{:.2f}".format((F.subs(values_b)),(uF_mod.subs(valu
es_b))))
```

```
1b) f(x,y)=4.91+/-0.32
```

In [9]:
```
#Part c uses the same method as in part A but w/ different values
#Assigning values for part c
values_c=[(x,11.1),(sigma_x,4.0),(y,5.4),(sigma_y,0.15)]
print("1c) f(x,y)={:.2f}+/-{:.2f}".format((F.subs(values_c)),(uF.subs(values_c
))))
```

```
1c) f(x,y)=4.91+/-0.86
```

In [11]:
```
import qexpy as q #import QExPy for part d
q.set_sigfigs(2) #adjusting number of significant figures to 2
```

In [12]:
```
#definig the given function so I can use multiple times for part d
def f(x,y):
    F=(q.sqrt(x**2+y**2))/q.log((q.sqrt(x**2+y**2)))
    return F
```

In [15]:
```
#1d) I will use Measurement to find the central value and uncertainties
#for part a there is no covariance so I will just use the measurement
x_q=q.Measurement(11.1,0.4)
y_q=q.Measurement(5.4,1.5)
q.set_error_method("Derivative") #ensure QExPy uses derivative method
#now I can print the output for part A using the QExpy for the derivative meth
od
print(f(x_q,y_q))
```

```
4.91 +/- 0.18
```

In [16]:
```
#Now setting the covariance and using the same measuremets as before
x_q1=q.Measurement(11.1,0.4)
y_q1=q.Measurement(5.4,1.5)
x_q1.set_covariance(y_q1,1.5)
q.quick_MC=True
print(f(x_q1,y_q1))
```

```
4.91 +/- 0.32
```

In [17]:
```
#Now changing the variables for part c using QExPy
x_q2=q.Measurement(11.1,4.0)
y_q2=q.Measurement(5.4,0.15)
print(f(x_q2,y_q2))
```

```
4.91 +/- 0.86
```

In [18]:
```
print("1d) The values using QExPy are equal to the values from parts a-c")
```

```
1d) The values using QExPy are equal to the values from parts a-c
```

Question 2

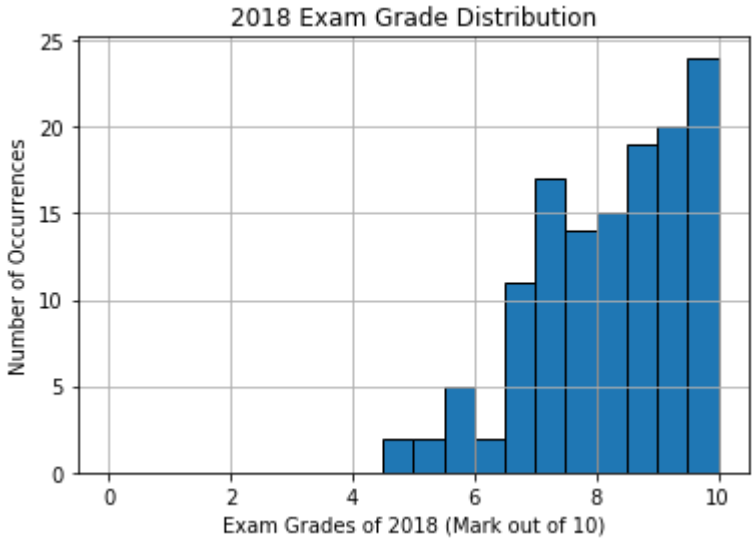
In [19]:
```
import csv
import numpy as np #to determine the statistical
with open('grades2018.csv') as file:
    csvReader=csv.reader(file, delimiter=',')
    #introduce each column of the file as empty arrays
    assignment=[]
    quiz=[]
    exam=[]
    lineCount=0 #starting the counter at 0
    for row in csvReader:
        if lineCount>0:#to ignore the headers
            #the append will add the values into the array
            assignment.append(float(row[0]))
            quiz.append(float(row[1]))
            exam.append(float(row[2]))
            lineCount+=1
        else:
            lineCount+=1
#Now that I have all the data as floats in one array I can calculate the mean,
 standard deviation, and error on the mean
#I need numpy module so i will import it and then use the print function to ou
tput these desire outputs
exam
N=len(exam)#length of the array
Emean=np.mean(exam)
Estd=np.std(exam,ddof=1)#ensures we use N-1, not N when dividing in the formul
a
Emerror=Estd/np.sqrt(N)
print("The mean is {:.2f}+/{:.2f} and the standard deviation is {:.2f}".format
((Emean),(Emerror),(Estd)))
```

```
The mean is 7.98+/0.11 and the standard deviation is 1.28
```

Sidenote

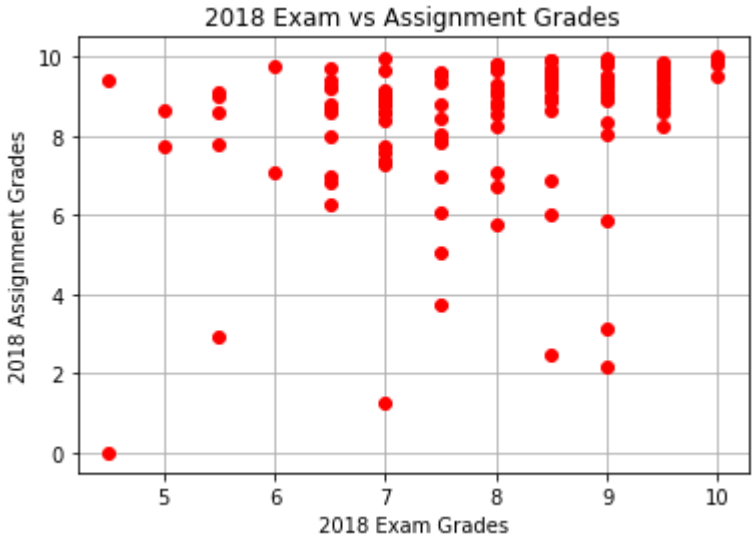
I did not know about this skiprows function so I made a for loop I also did not know that I could make arrays with python and access an entire row or column with ":" hence the for loop again

In [20]:
```
#2b) referring to example 5.3 in the textbook
import pylab as pl #need to make the histogram
pl.hist(exam,bins=np.linspace(0,10,21),edgecolor='black')
#making the bins 0.5 width and have edges of black
pl.xlabel("Exam Grades of 2018 (Mark out of 10)")
pl.ylabel("Number of Occurrences") #axis titles
pl.title("2018 Exam Grade Distribution") #graph title
pl.grid() #adds a grid
pl.show() #outputs the histogram
```



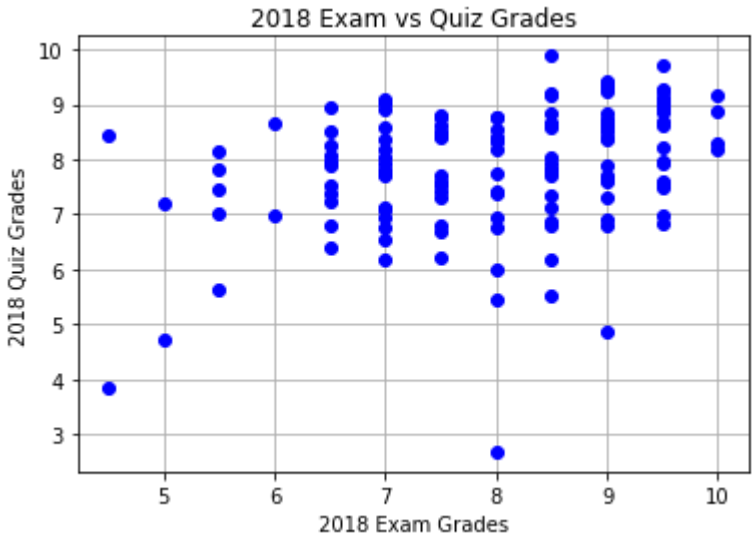
In [21]:
```
#2c) Creating scatter plots
#We already have the data for the quizzes and assignments from the for loop in
 part a
#first is the exam grades vs the assignments
pl.plot(exam,assignment,'ro') #plots the points as red points
pl.xlabel("2018 Exam Grades")
pl.ylabel("2018 Assignment Grades")
pl.title("2018 Exam vs Assignment Grades")
pl.grid()#grid on
pl.show
```

Out[21]: <function matplotlib.pyplot.show(*args, **kw)>



In [22]:
```
#Now for the exam grades vs quiz grades
pl.plot(exam,quiz,'bo') #plots the points as blue points
pl.xlabel("2018 Exam Grades")
pl.ylabel("2018 Quiz Grades")
pl.title("2018 Exam vs Quiz Grades")
pl.grid()#grid on
pl.show
```

Out[22]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [23]:
```
#2d) Creating a function to compute the covariance using the code from example
 5.2 in the textbook
def Correlation(a,b):
    #compiling the file's data into arrays:
    ai=np.array(a)
    bi=np.array(b)
    #first we need the means
    mean1=ai.mean()
    mean2=bi.mean()
    #then standard deviations
    std1=ai.std()#do NOT adjust for N-1 bc this is for the entire class, not a
 sample
    std2=bi.std()
    #Now covariances:
    cov=((ai-mean1)*(bi-mean2)).sum()/(len(a)-1)
    CF=cov/std1/std2
    return CF
```

In [24]:
```
#2d) Determining the correlation factor of these graphs:
#Applying the covariance function for the exam and assignments, and then exam
 and quiz:
EA=Correlation(exam,assignment)
EQ=Correlation(exam,quiz)
print("2d) The correlation factors between the Exams and Assignments and the E
xams and Quizzes, respectively, are:")
print("{:.2f} and {:.2f}".format((EA),(EQ)))
```

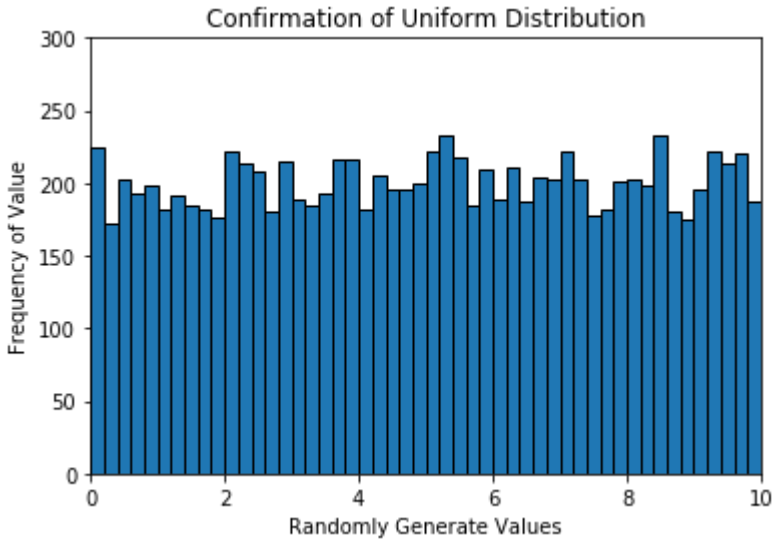
```
2d) The correlation factors between the Exams and Assignments and the Exams a
nd Quizzes, respectively, are:
0.29 and 0.34
```

Since the quizzes and exams have the greater correlation, the quizzes are a better prediction for the exam grades. This can be accounted for students helping. each other out on assignments, but quizzes and exams are individually done.

Question 3

In [25]:
```
#3a) Importing the given code:
#this uses the same code as 2b
import math
xa=np.random.uniform(0,10,10000)
#defining the histogram values as indicated in the question
n,bins,patches=pl.hist(xa,bins=50,edgecolor='black') #distinguishing each bin
 by making the edgecolors black
#giving axis labels
pl.xlabel("Randomly Generate Values")
pl.ylabel("Frequency of Value")
pl.title("Confirmation of Uniform Distribution") #title
pl.axis([0,10,0,300]) #axis size
#I chose not to use the grid because it was not helpful in reading the graph
pl.show #display the graph
```

Out[25]: <function matplotlib.pyplot.show(*args, **kw)>



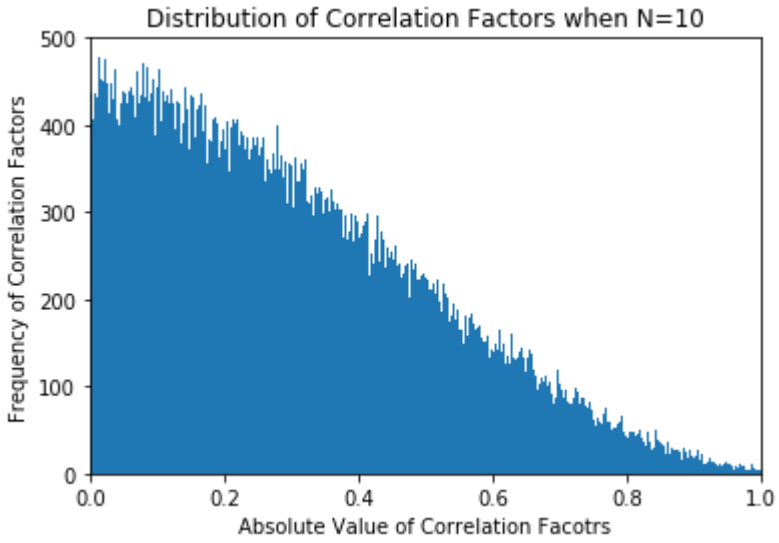
3a) Clearly this is a unifromly distributed histogram.

In [36]:
```
#3b) Generating the random using a for loop
import random
#making this a function so I can use for part c
def genrand():
    xlist=[] #introducing empty lists
    ylist=[]
    for i in range(10): #for N=10
        x=random.uniform(0,1) #generating the random numbers
        y=random.uniform(0,1)
        xlist.append(x) #append the random values into the lists
        ylist.append(y)
        i=+1
    cFactor=Correlation(xlist,ylist) #using the correlation function from befo
re
    return cFactor
#using this function for part b to give the correlation factor:
print("The correlation factor between the two data sets is {:.2f}".format(genr
and()))
```

```
The correlation factor between the two data sets is -0.10
```

In [37]:
```
#3c) Using the function in part b in a for loop to get 100 000 values
clist=[]
for j in range(100000):
    c=genrand()
    clist.append(abs(c))#taking absolute value of the correlation factor outpu
tted by genrand
    j=+1
#Using same process as creating a histogram as before:
#plotting the array clist (n):
n,bins,patches=pl.hist(clist,bins=500)#500 was chosen arbitrarily given 50 was
 used for 10 000 iterations
#not distinguishing each bin because it will be messy
pl.xlabel("Absolute Value of Correlation Facotrs")
pl.ylabel("Frequency of Correlation Factors")
pl.title("Distribution of Correlation Factors when N=10") #title
pl.axis([0,1,0,500]) #axis size
#I chose not to use the grid because it was not helpful in reading the graph
pl.show
```

Out[37]: <function matplotlib.pyplot.show(*args, **kw)>



In [38]:
```
#3d) Finding the value along the x-axis which 95% of the values lies
#need to repeat with with 10% so will make another function
def function(n,array):
    array.sort() #sort the list in ascending order
    #this will calculate the area from left to right
    a=int(((n*len(array))-1)) #create a marker that is 95% of all inputs
    return array[a]
#95% of the inputs are before the a-th term in the sorted list
#applying the function for the given percentages
ans_95=function(0.95,clist) #for threshold of 5%
ans_90=function(0.90,clist) #for threshold of 10%
print("95% of the data lies before x={:.3f},(90% thershold),".format(ans_90))
print("and 90% of the data lies before x={:.3f} (95% threshold)".format(ans_95
))
```

```
95% of the data lies before x=0.614,(90% thershold),
and 90% of the data lies before x=0.707 (95% threshold)
```

For this scenario, our N is 10 and the 95% and 90% correlate with the 5% and 2.5% columns in the table from the textbook, respectively. This is because the data analyzed is for an absolute value around the correlation value hence the doubling of percentage. Given that the confidence level for 95% and 90% are 0.704 and 0.61 respectively, these values are greater than those in table 5.1 for N=10. Therefore the correlation is significant.

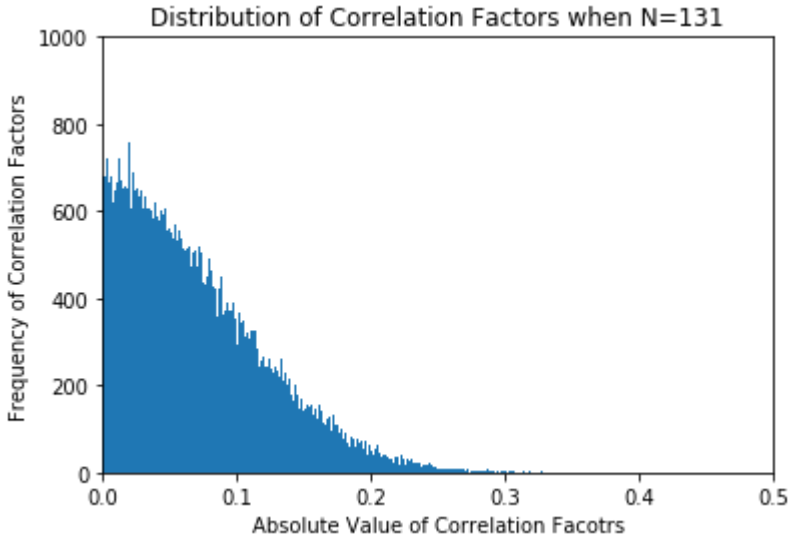
In [41]:
```
#3e) Analyzing the significance of the correlation factor for Q2
#using a similar method as part C by generating random values
k=0 #initiating a variable
#for the number of occurences of the correlation factor greater than 0.34
corrF=[] #defining an empty arrary to store the correlation values
for i in range (100000): #for loop to have multiple values to compare
    random1=np.random.uniform(0,10,131) #generate 131 random numbers b/n 1 & 1
0
    random2=np.random.uniform(0,10,131) #repeat so can compare the correlation
 factors
    corrF.append(Correlation(random1,random2))
    #adding the correlation factors into the empty array
    #uses the Correlation function from before
    if abs(corrF[i])>0.34:
#testing which absolute value of the correlation factor is greater than 0.34
        k+=1 #adds one to how many times the correlation factor is greater tha
n 0.34
#Calculating the probability of the correlation factor being greater than 0.34
probability=(k/100000)*100 #number of values/total values taken (as a percent)

#Showing these results in a histogram:
n,bins,patches=pl.hist(corrF,bins=500)
pl.xlabel("Absolute Value of Correlation Facotrs")
pl.ylabel("Frequency of Correlation Factors")
pl.title("Distribution of Correlation Factors when N=131") #title
pl.axis([0,0.5,0,1000]) #axis size
pl.show

print("A correlation Factor greater than 0.34 when N=131 for 100000 trials i
s.")
k
```

```
A correlation Factor greater than 0.34 when N=131 for 100000 trials is.
```

Out[41]: 5



In [44]:
```
print("The probability of finding a correlation factor greater than 0.34 for N
=131 is")
print("{:.3f}%".format(probability))
```

```
The probability of finding a correlation factor greater than 0.34 for N=131 i
s
0.005%
```

Therefore there is a less than 1% chance the 131 pairs of generated numbers be correlated more than a factor of 0.34. Hence, the correlation factor between the Quizzes and Exams is greatly significant since we know the chances of that number appearing randomly is very slim.