Assignment 3

Christina Bisol

20046973

Q1

Given N pairs of measurements: (x_i, y_i) with negligible uncertainties in x_i and a constant uncertainty, σ_i on the y_i

Part A:

Assume the σ_y corresponds to a standard error. Want to determine the parameters {a,b,c} of the quadratic function $y(x) = a + bx + cx^2$ that best fits. To do so, we will use the chi-squared and minimize it by taking its derivative and setting it to 0:

Chi-squared formula is:

$$X^2 = \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i, \beta))^2}{\sigma_{yi}^2}$$

substituing our values in for the given eqn:

$$X^2 = \sum_{i=1}^{i=N} \frac{(y_i - a - bx_i - cx_i^2)^2}{\sigma_{yi}^2}$$

Now taking the deterivative with respect to a, b, and c we have the following eqns:

$$\frac{dX^2}{da} = \frac{d}{da}\sum_{i=1}^{i=N} \frac{(y_i - a - bx_i - cx_i^2)^2}{\sigma_{yi}^2} = \frac{-2}{\sigma_{yi}^2}\sum_{i=1}^{i=N}(y_i - a - bx_i - cx_i^2)$$
$$\frac{dX^2}{db} = \frac{d}{db}\sum_{i=1}^{i=N} \frac{(y_i - a - bx_i - cx_i^2)^2}{\sigma_{yi}^2} = \frac{-2}{\sigma_{yi}^2}\sum_{i=1}^{i=N}x_i(y_i - a - bx_i - cx_i^2)$$
$$\frac{dX^2}{dc} = \frac{d}{dc}\sum_{i=1}^{i=N} \frac{(y_i - a - bx_i - cx_i^2)^2}{\sigma_{yi}^2} = \frac{-2}{\sigma_{yi}^2}\sum_{i=1}^{i=N}x_i^2(y_i - a - bx_i - cx_i^2)$$

Setting all three eqn's to zero we get:

$$\frac{-2}{\sigma_{yi}^2}\sum_{i=1}^{i=N}(y_i - a - bx_i - cx_i^2) = 0$$
$$\frac{-2}{\sigma_{yi}^2}\sum_{i=1}^{i=N}x_i(y_i - a - bx_i - cx_i^2) = 0$$
$$\frac{-2}{\sigma_{yi}^2}\sum_{i=1}^{i=N}x_i^2(y_i - a - bx_i - cx_i^2) = 0$$

Expanding & simplifying

$$\sum_{i=1}^{i=N} y_i = Na + b\sum_{i=1}^{i=N} x_i + c\sum_{i=1}^{i=N} x_i^2$$
$$\sum_{i=1}^{i=N} x_i y_i = a\sum_{i=1}^{i=N} x_i + b\sum_{i=1}^{i=N} x_i^2 + c\sum_{i=1}^{i=N} x_i^3$$
$$\sum_{i=1}^{i=N} x_i^2 y_i = a\sum_{i=1}^{i=N} x_i^2 + b\sum_{i=1}^{i=N} x_i^3 + c\sum_{i=1}^{i=N} x_i^4$$

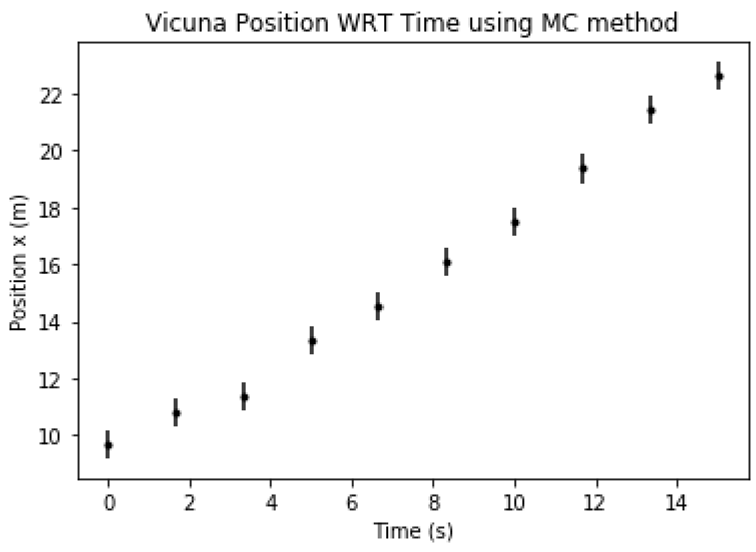
substituting in $s_n \equiv \sum_i^N x_i^n$ and solving we get the desired outcome (verified with an online solver):

$$a = \frac{(s_3^2 - s_2 s_4)(s_2 \sum y - s_1 \sum xy) - (s_2^2 - s_1 s_3)(s_3 \sum xy - s_2 \sum x^2 y)}{(s_3^2 - s_2 s_4)(s_2 N - s_1^2) + (s_2^2 - s_1 s_3)^2}$$
$$b = \frac{(s_2^2 - Ns_4)(s_1 \sum y - N \sum xy) - (s_1 s_2 - Ns_3)(s_2 \sum y - N \sum x^2 y)}{(s_2^2 - Ns_4)(s_1^2 - Ns_2) - (s_1 s_2 - Ns_3)^2}$$
$$a = \frac{(s_1 s_2 - Ns_3)(s_1 \sum y - N \sum xy) - (s_1^2 - Ns_2)(s_2 \sum y - N \sum x^2 y)}{(s_1 s_2 - Ns_3)^2 - (s_1^2 - Ns_2)(s_2^2 - Ns_4)}$$

Part B:

Using the Monte Carlo simulation to show vicun running. Assuming time has negligble uncertainty b/n 0 and 15 seconds.

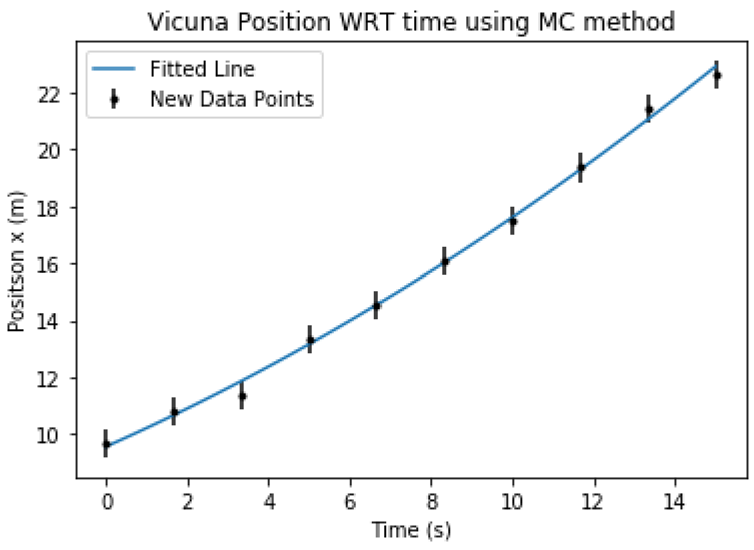
In [22]:
```
#import necessary modules to compute results
import numpy as np
import pylab as pl
import scipy.stats as stats

#defining variables
N=10 #define the number of Monte Carlo "experiments"
x=10 #vicuna started position in meters
sigma_x=0.5 #error in x
#standard eror of position, assumed constant
v=0.5 #start velocity in m/s
a=0.05 #acceleration, assume constant, in m/s^2

tvals=np.linspace(0,15,N) #generate time values

meanx=x+v*tvals+v*a*tvals**2 #expected positions using kinematics
#this is the mean value of x for each t

#normally distributed x values for N experiments
xvals=np.random.normal(meanx,sigma_x,N)

#Plotting the data (position as a function of time) with error bars:
pl.errorbar(tvals,xvals,sigma_x,color='black',marker='.',ls='none')
pl.xlabel('Time (s)')
pl.ylabel('Position x (m)')
pl.title('Vicuna Position WRT Time using MC method')
pl.show()
```



Part C:

Fitting the data with formula from question 1a

In [23]:
```
#s is the sum functvalson as given at the end of part a
#which is substvalstuted into the eq'ns for a,b,c for simplicity
def s(t,n):
    return sum(t**n)
#using s to calculate a when x=tvals and y=meanx:
a = ((s(tvals,3)**2-s(tvals, 2)*s(tvals, 4))*(s(tvals, 2)*sum(xvals) - s(tvals
, 1)*sum(xvals*tvals))    \
- (s(tvals, 2)**2-s(tvals, 1)*s(tvals, 3))*(s(tvals, 3)*sum(tvals*xvals)-s(tva
ls, 2)*sum(xvals*tvals**2))) \
/ ((s(tvals, 3)**2 -s(tvals, 2)*s(tvals, 4))*(s(tvals, 2)*N - s(tvals, 1)**2)
+ (s(tvals, 2)**2 - \
s(tvals, 1)*s(tvals, 3))**2)

#using s to calc b when x=tvals & y=meanx:
b =((s(tvals, 2)**2 -N*s(tvals, 4))*(s(tvals, 1)*sum(xvals)-N*sum(xvals*tvals
))-(s(tvals, 1)*s(tvals, 2)-N*s(tvals, 3)) \
*(s(tvals, 2)*sum(xvals)-N*sum(xvals*(tvals**2)))) / ((s(tvals, 2)**2 - N*s(tv
als, 4))*(s(tvals, 1)**2-N*s(tvals, 2)) \
- (s(tvals, 1)*s(tvals, 2) - N*s(tvals, 3))**2)

#repeate for c:
c =((s(tvals, 1)*s(tvals, 2)-N*s(tvals, 3))*(s(tvals, 1)*sum(xvals)-N*sum(xval
s*tvals)) - (s(tvals, 1)**2-N*s(tvals, 2))\
*(s(tvals, 2)*sum(xvals)-N*sum(xvals*tvals**2)) ) / ( (s(tvals, 1)*s(tvals, 2)
-N*s(tvals, 3))**2 - (s(tvals, 1)**2 \
-N*s(tvals, 2))*(s(tvals, 2)**2 - N*s(tvals, 4))  )

t_array=np.linspace(0,15,100) #array of tvalsme values to graph
x_array=c*t_array**2+b*t_array+a #plugging in this array into the quadratvalsc
#using the coefficients we just calculated as instructed from part A
#to get the corresponding x-values for t_array

#Plottvalsng original data (part b) and the fitted line (using t_array & x_arr
ay)
pl.errorbar(tvals,xvals,sigma_x,color='black',marker='.',label="New Data Point
s",ls='none')
pl.plot(t_array,x_array, label='Fitted Line')
#plottvalsng fitted line in a different colour
pl.xlabel('Time (s)')
pl.ylabel('Positson x (m)')
pl.title('Vicuna Position WRT time using MC method')
pl.legend()
pl.show()
```



Through this comparison of the value of best fit parameters and their expected values we can clearly see that it does agree with the data qualitatively. This is evident in the line of best fit passing through each data point withint the error bars.

Part D:

Fitting the dat from part b to a quadratic ewn using QExpy

In [25]:
```
import qexpy as q #import the QExpy module

#fitting the data to a quadratic using QExpy
x = q.MeasurementArray(xvals, [0.5], name='Position', units='m')
t = q.MeasurementArray(tvals, [0], name='Time', units='s')

#simplifying to make a plot with residuals
fig = q.MakePlot(t, x) #plotting position as a function of t
fig.fit('polynomial2') #specifying quadratic fit
fig.show_residuals = True #display the residuals

fig.show()

#From running the code, Chi2 = 6.57 and ndof = 6
chi = 2.29**0.5
#Find the p value that corresponds to this chi2
p = stats.chi2.sf(chi,6)
print('The p-value that corresponds to the Chi^2 and number of degrees of free
dom obtained from the simulation is {:.2f}'\
      .format(p))
```

```
-----------------Fit results-------------------
Fit of  dataset2  to  degree_2_polynomial
Fit parameters:
dataset2_degree_2_polynomial_fit0_fitpars_par0 = 9.6 +/- 0.2,
dataset2_degree_2_polynomial_fit0_fitpars_par1 = 0.64 +/- 0.07,
dataset2_degree_2_polynomial_fit0_fitpars_par2 = 0.017 +/- 0.004

Correlation matrix: 
[[ 1.    -0.81   0.664]
 [-0.81   1.    -0.963]
 [ 0.664 -0.963  1.   ]]

chi2/ndof = 2.29/6
---------------End fit results----------------



The p-value that corresponds to the Chi^2 and number of degrees of freedom ob
tained from the simulation is 0.96
```

Comment about parameters

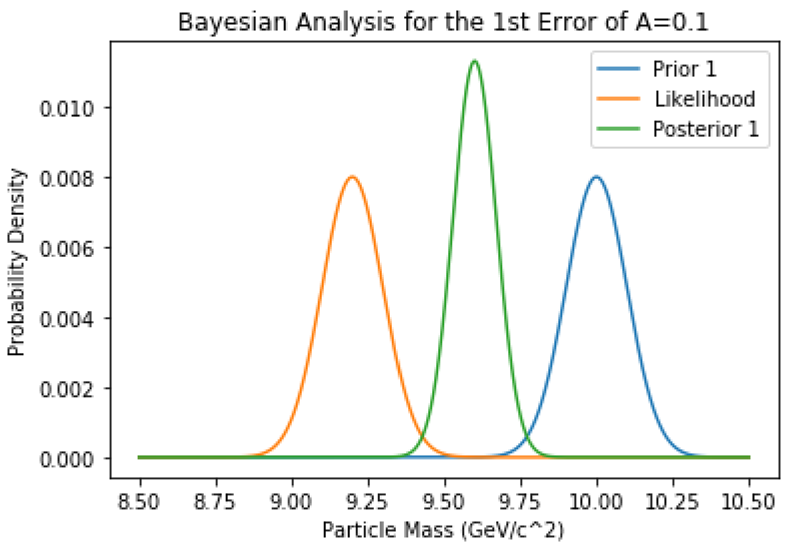
Since the p-value is high (0.96), by defintion the results are significant, which confirms the conclusions in part b (i.e. the curve of best fit being in quadratic form is a good representation of the data).

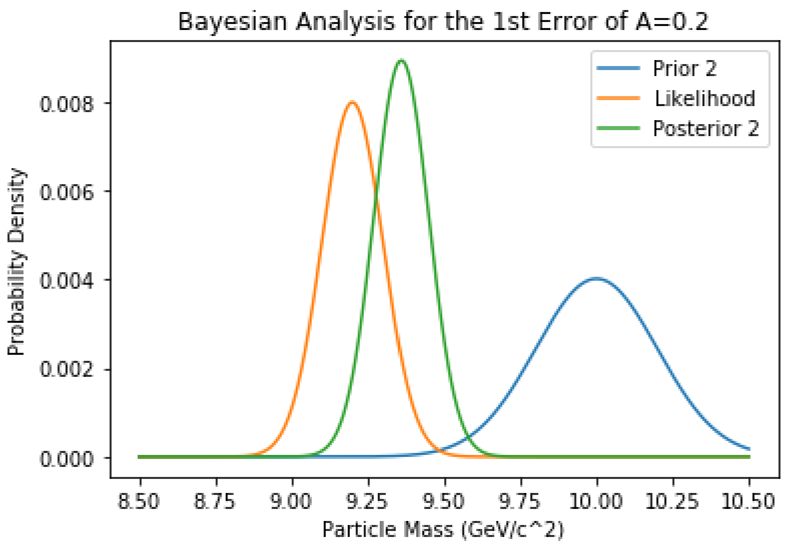
Q2 Bayesian Analysis of particle masses

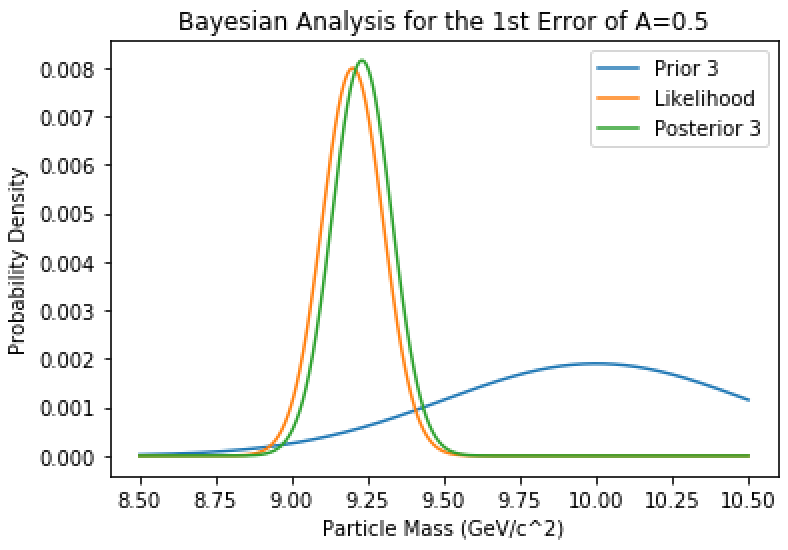
Part A

Plotting the normalized daata

In [4]:
```
#introducing the variables
ma=10.0 #mass of particle in GeV/c^2
mb=9.2 #mass of particle in GeV/c^2 from repeated experiment

#defining the given uncertainties
e1=0.1
e2=0.2
e3=0.5
eb=e1 #error on mass of experiment b

#define array to hold mean values of the mass for either experiment
n=np.linspace(8.5,10.5,1000)


#Assume the prior prob for the mass is given by normal distribution
#using the given formula (massA=n,muA=given mA,sigma=e):
Pp1 = stats.norm.pdf(n,ma,e1) #for first error
Pp2 = stats.norm.pdf(n,ma,e2) #for second error
Pp3 = stats.norm.pdf(n,ma,e3) #for third error

#normalizing the priors
Pp1=Pp1/Pp1.sum()
Pp2=Pp2/Pp2.sum()
Pp3=Pp3/Pp3.sum()

#Likelihood is also normally distributed:
Plike=stats.norm.pdf(n,mb,eb)
plike=Plike/Plike.sum()
#only have one error for mass B so only have to compute once
#and will use for each error on A

#Posterior=P(a)P(b)
post1=Pp1*plike
post2=Pp2*plike
post3=Pp3*plike

#normalizing
post1=post1/post1.sum()
post2=post2/post2.sum()
post3=post3/post3.sum()

#plotting with regards to the first error of 0.1
pl.plot(n,Pp1, label='Prior 1')
pl.plot(n, plike, label='Likelihood')
pl.plot(n, post1, label='Posterior 1')
pl.xlabel('Particle Mass (GeV/c^2)')
pl.ylabel('Probability Density')
pl.legend()
pl.title("Bayesian Analysis for the 1st Error of A=0.1")
pl.show()

#plotting with regards to the second error of 0.2
pl.subplot()
pl.plot(n,Pp2, label='Prior 2')
pl.plot(n, plike, label='Likelihood')
pl.plot(n, post2, label='Posterior 2')
pl.xlabel('Particle Mass (GeV/c^2)')
pl.ylabel('Probability Density')
pl.legend()
pl.title("Bayesian Analysis for the 1st Error of A=0.2")
pl.show()

#plotting with regards to the third error of 0.5
pl.subplot()
pl.plot(n,Pp3, label='Prior 3')
pl.plot(n,plike, label='Likelihood')
pl.plot(n, post3, label='Posterior 3')
pl.xlabel('Particle Mass (GeV/c^2)')
pl.ylabel('Probability Density')
pl.legend()
pl.title("Bayesian Analysis for the 1st Error of A=0.5")
pl.show()
```







Part B

Determining corresponding values of the mass & uncertainty that combines both experiments

In [5]:
```
#defining constants:
i=0
i_max=0
maxm=0

#find mean & stddev
def mAnds(post,sigma): #takes in posterior and error
    i=0
    i_max=0
    maxm=0


    for item in post:
        if item> maxm:#comparing each item with prev max to find mean
            maxm=item
            i_max=i
        i=i+1
    summ=post[i_max]
    k=0

    while summ<0.68: #to find standard dev with 68% confidence
        k=k+1
        summ=summ+post[i_max+k]
        summ=summ+post[i_max-k]
    #print out the outputs of this function
    print("For prior error of {:.1f}, Mean of mass = {:.2f} +/- {:.2f}".format
(sigma,n[i_max],n[i_max]-n[i_max-k]))
#Getting the mean and standard deviation on each posterior for the given error
s using the function above
mAnds(post1,e1)
mAnds(post2,e2)
mAnds(post3,e3)
```

```
For prior error of 0.1, Mean of mass = 9.60 +/- 0.07
For prior error of 0.2, Mean of mass = 9.36 +/- 0.09
For prior error of 0.5, Mean of mass = 9.23 +/- 0.10
```

Part C

Comparing the results

In [28]:
```
#For the first error A=0.1
#using Chi-squared
X1=(mb*eb**(-2)+ma*e1**(-2))/(e1**(-2)+eb**(-2))
eX1=(e1**(-2)+eb**(-2))**(-0.5)

#Print results:
print("When the error on the prior is 0.1 we have {:.2f} +/- {:.2f}".format(X1
,eX1))

#Repeat for when error A=0.2:
X2=(mb*eb**(-2)+ma*e2**(-2))/(e2**(-2)+eb**(-2))
eX2=(e2**(-2)+eb**(-2))**(-0.5)
print("When the error on the prior is 0.2 we have {:.2f} +/- {:.2f}".format(X2
,eX2))

#Repeat for when error A=0.5
X3=(mb*eb**(-2)+ma*e3**(-2))/(e3**(-2)+eb**(-2))
eX3=(e3**(-2)+eb**(-2))**(-0.5)
print("When the error on the prior is 0.5 we have {:.2f} +/- {:.2f}".format(X3
,eX3))
```

```
When the error on the prior is 0.1 we have 9.60 +/- 0.07
When the error on the prior is 0.2 we have 9.36 +/- 0.09
When the error on the prior is 0.5 we have 9.23 +/- 0.10
```

These values in 2c agree with the values calculated in 2b :)

Q3 analysing data from uniform circular motion

Part A

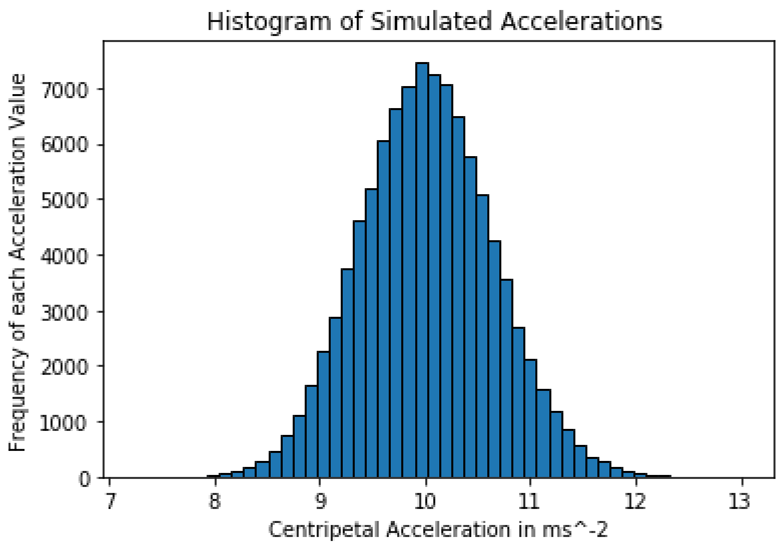
Using Monte Carlo method to propagate errors

In [7]:
```
#define the given variables & convert to SIU
m=20.0/1000 #mass (convert from g to kg)
em=0.2/1000 #error on the mass (converted from g to kg)
r=10.0/100 #radium converted from cm to m
er=0.2/100 #error on the radius converted from cm to m
v=1.00 #speed in m/s
ev=0.03 #error on the speed in m/s

#Calculating acceleration and uncertaitiies with derivatives
a=(v**2)/r #centripetal acceleartion

#determining its partial derivatives to calc its error
da_dv=(2**v)/r #derivative of centripetal accelw.r.t. velocity
da_dr=(-1*v**2)/(r**2) #derivativae of centripetal accell w.r.t. velocity

ea=((da_dv*ev)**2+(da_dr*er)**2)**0.5 #error on centripetal acceleration

print('For comparison later, using error propagation, the centripetal accelera
tion is \n{:.2f} +/- {:.2f} ms^-2' \
      .format(a,ea))
```

```
For comparison later, using error propagation, the centripetal acceleration i
s
10.00 +/- 0.63 ms^-2
```

In [8]:
```
#Now Using the Monte Carlo Metho as exemplified in example 10.8 of the txtbk
N=100000 #number of samples
mmc=np.random.normal(m,em,N) #N mass values in kg
rmc=np.random.normal(r,er,N) #N radius values in m
vmc=np.random.normal(v,ev,N) #N velcoity values in m/s

#using thes simulated values to determine the centripetal acceleartion:
amc=(vmc**2)/rmc

#making the histogram with 50 bins
pl.hist(amc, bins=50, edgecolor='black')
pl.xlabel("Centripetal Acceleration in ms^-2")
pl.ylabel("Frequency of each Acceleration Value")
pl.title("Histogram of Simulated Accelerations ")

#Finding the central value and error:
acv=amc.mean() #central value of simulated acceleration
eacv=amc.std() #error on the central values of the simulated acceleration

print("Using the Monte Carlo method, the centripetal acceleration in {:.2f} +/
- {:.2f}".format(acv,eacv))
pl.show()
```

```
Using the Monte Carlo method, the centripetal acceleration in 10.02 +/- 0.63
```

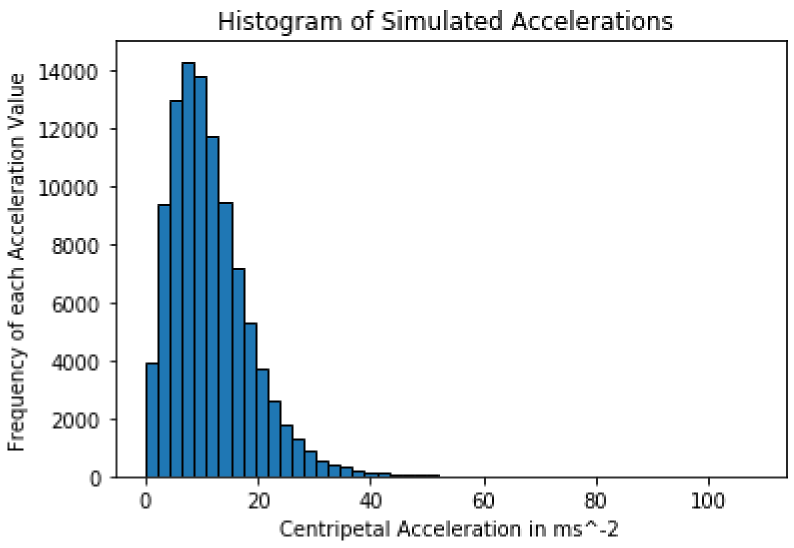


Part B

Same as A but increasing the uncertainty by a factor of 10

In [9]:
```
#copy and pasting the code from part a
#but multiplying the initial variables by 10

#define the given variables & convert to SIU & mult by 10
m=20.0/1000 #mass (convert from g to kg)
em=0.2/100 #error on the mass (converted from g to kg)
r=10.0/100 #radium converted from cm to m
er=0.2/10 #error on the radius converted from cm to m
v=1.00 #speed in m/s
ev=0.30 #error on the speed in m/s

#Calculating acceleration and uncertaitiies with derivatives
a=(v**2)/r #centripetal acceleartion

#determining its partial derivatives to calc its error
da_dv=(2**v)/r #derivative of centripetal accelw.r.t. velocity
da_dr=(-1*v**2)/(r**2) #derivativae of centripetal accell w.r.t. velocity

ea=((da_dv*ev)**2+(da_dr*er)**2)**0.5 #error on centripetal acceleration

print('For comparison later, using error propagation, the centripetal accelera
tion is \n{:.2f} +/- {:.2f} ms^-2' \
      .format(a,ea))

#Now Using the Monte Carlo Metho as exemplified in example 10.8 of the txtbk
N=100000 #number of samples
mmc=np.random.normal(m,em,N) #N mass values in kg
rmc=np.random.normal(r,er,N) #N radius values in m
vmc=np.random.normal(v,ev,N) #N velcoity values in m/s

#using thes simulated values to determine the centripetal acceleartion:
amc=(vmc**2)/rmc

#making the histogram with 50 bins
pl.hist(amc, bins=50, edgecolor='black')
pl.xlabel("Centripetal Acceleration in ms^-2")
pl.ylabel("Frequency of each Acceleration Value")
pl.title("Histogram of Simulated Accelerations ")

#Finding the central value and error:
acv=amc.mean() #central value of simulated acceleration
eacv=amc.std() #error on the central values of the simulated acceleration

print("Using the Monte Carlo method, the centripetal acceleration in {:.2f} +/
- {:.2f}".format(acv,eacv))
pl.show()
```

```
For comparison later, using error propagation, the centripetal acceleration i
s
10.00 +/- 6.32 ms^-2
Using the Monte Carlo method, the centripetal acceleration in 11.42 +/- 7.15
```



Q4 Monte Carlo to evaluate complex integrals

Part A

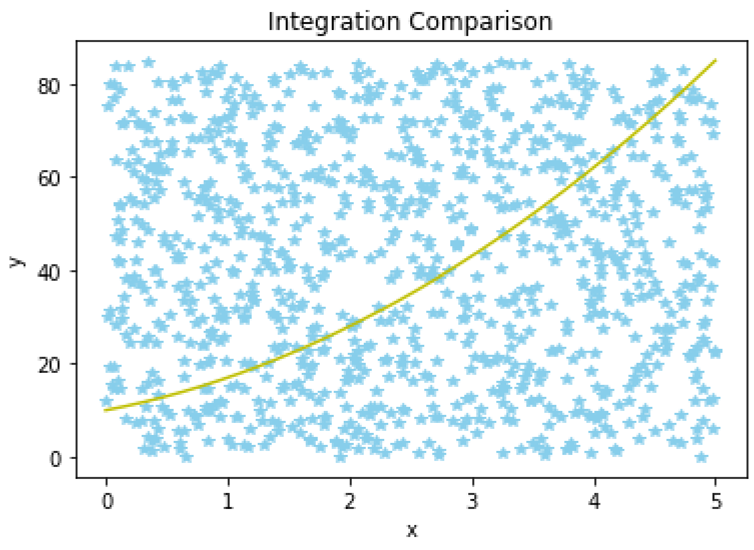
Evaluating the inetgral of $g(x) = 1$ over the area defined by $f(x) = 10 - 5x + 2x^2$ and the x-axis overthe interval of $[0, 5]$ Our bounds are $0 \leq x \leq 5$ and $0 \leq y \leq 85$ and will use the given equation

$$F = V\frac{1}{N}\sum_{1}^{N} f'(x_i)$$

In [10]:
```
#Following the given MC strategy with help from
#http://docs.scipy.org/goc/scipy/reference/tutorial/integrate.html
import scipy.integrate as intg
#introducing the given function as a python function
def f(x):
    y=10+5*x+2*x**2
    return y
#Bounds on V by setting a max and min value on x and y are given as
xmin=0
xmax=5
ymin=0
ymax=85
#Determining the rectangular volume of V using thes points and g(x)=1
V=(xmax-xmin)*(ymax-ymin)*1

#computing the integral of f analytically for comparison later
integral=intg.quad(f,xmin,xmax)
#showing the result
print('The intgeral of the given function computed analytically is {:.2f}'.for
mat(abs(integral[0])))

#Now using MC method using the same bounds as before

#another function to produce randomly generated numbers as in example 10.1 fro
m the txtbk
def LCG(a=16807,b=0,c=2147483647):
    L=(a*LCG.seed+b)%c
    LCG.seed=L
    return L
#initialize the seed
LCG.seed=10135

#Generate N random values for x & y in the bounds of V above
#I used example 10.2 in the textbook as a template

N=1000 #arbitrarily chose 1000 random points
#generating an empty array of size N to store x & y values
x=np.zeros(N)
y=np.zeros(N)
g=np.zeros(N) #generating an empty array of size N to store the values

#using the LCG function to fill the arrays with random numbers
for i in range(N):
    x[i]=(LCG()/N)%5 #used modulo 5 so it will be between 0 & 5
    y[i]=(LCG()/N)%85 #used modulo 85 so y values will be

#Ensuring y points are below the curve so analys g'(x)
    if y[i] <= f(x[i]):
        g[i]=1 #g'(x_i)=1 as desired
    else:
        g[i]=0 #otherwise it is 0

#Now computing the average value of g' as the given desired integral
F=(V/N)*sum(g) #how many points that lie between the urve and the x-axis

#showing the results
```

```
print('The value of the integral evaluated using the Monte Carlo Method is {:.
2f}'.format(F))

xvals= np.linspace(0,5,N) #Creating N x values from x=0 to x=5

#plotting the points to ensure matches Figure (1) in the question
pl.plot(x,y,'*',color='skyblue')
pl.plot(xvals, f(xvals), 'y') #plots xvals to show the line
pl.xlabel('x')
pl.ylabel('y')
pl.title('Integration Comparison')

pl.show()
```

```
The intgeral of the given function computed analytically is 195.83
The value of the integral evaluated using the Monte Carlo Method is 196.35
```

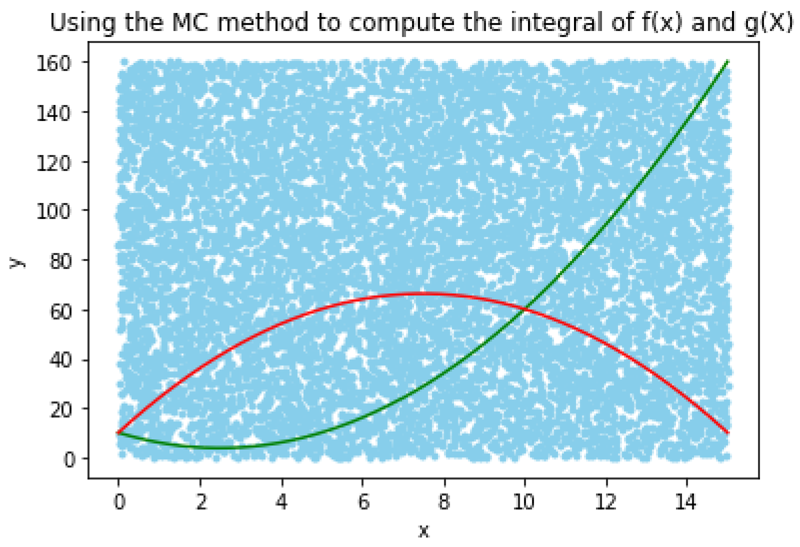


Part B

Considering the area between $f(x) = 10 - 5x + x^2$ and $g(x) = 10 + 15x - x^2$ over the interval $0 \leq x \leq 15$ Want the centre of mass, and its mass assuming a constant surface density of 1.

In [21]:
```
#definining the given functions
def fb(x): #called fb to distinguish b/n the function in part a
    return 10-5*x+x**2
def g(x):
    return 10+15*x-x**2
#because the bottom function becomes the top function after the point of inter
section
#we need this pt because will have two seperate integrals
#Through physical calculations this is determined to be:
k=10 #point of intersection of the two functions

#setting the bounds
xmin=0
xmax=15 #given

ymax=160
ymin=0 #from analysing the graph

#to verify these bouds desmo.com was used

#initialize the seed from the LCG function before
LCG.seed=10135

#Generating 10,000 points for x and y:
N=10000

#initiating empty arrays for x and y values as before:
x=np.zeros(N)
y=np.zeros(N)
G=np.zeros(N)

for i in range(N):
    x[i]=(LCG()/N)%15 #getting x values to be in range of 0 & 15
    y[i]=(LCG()/N)%160 #getting y values to be in range of 0 and 160

    #Determine pts w/in the fish:
    if x[i]<k: #for the head
        if y[i]>fb(x[i]) and y[i]<g(x[i]):
            G[i]=1 #make G 1 if it is in range
        else:
            G[i]=0 #otherwise G is 0
    if x[i]>k: #for the tail
        if y[i]>g(x[i]) and y[i]<fb(x[i]):
            G[i]=1 #make G 1 if it is in range
        else:
            G[i]=0 #otherwise G is 0

#Plotting the data points
pl.plot(x,y,'.',color='skyblue')

#Plotting the two functions
xline=np.linspace(xmin,xmax,N)
pl.plot(xline,fb(xline),'g')
pl.plot(xline,g(xline),'r')
pl.xlabel('x')
pl.ylabel('y')
pl.title('Using the MC method to compute the integral of f(x) and g(X)')


#Calculating the total mass:
M=2400*(1/N)*sum(G)
print('The mass of the fish is {:.2f}'.format(M))

#Determining centre of mass
xvals=0
yvals=0
for i in range(N):
    if G[i]==1:
        xvals=xvals+x[i]
        yvals=yvals+y[i]

Xcm=xvals/sum(G)
Ycm=yvals/sum(G)

print('The centre of mass of the fish is ({:.2f},{:.2f})'.format(Xcm,Ycm))
```

```
The mass of the fish is 654.00
The centre of mass of the fish is (9.18,56.47)
```


Using the MC method to compute the integral of f(x) and g(X)
y
x