Homework 2

This solution has been sourced from student work.

NOTE:Its partly in python, partly in Matlab, using the ipython notebook on the enthought distribution updated to the latest python, and also using, for Matlab Magics, https://github.com/arokem/python-matlab-bridge. If you use the bridge (make sure matlab is in your path) you too can make solutions in this notebook style. Thanks Jean-Remy for the inspiration. Indeed, if you look at that github repo, you could even mix matlab and python. (Or R/Octave/cython and python for your own professional work)

Others can safely ignore the next two cells. They are for setup only.

In [1]:
```
import os
import pymatbridge as pymat
reload(pymat)
from IPython.display import Image
```

In [2]:
```
ip = get_ipython()
pymat.load_ipython_extension(ip)
```

```
Starting MATLAB on http://localhost:53418
 visit http://localhost:53418/exit.m to shut down same
...MATLAB started and connected!
```

Question 1

It is Valentine's day and you decided to design a nice gift for your loved one. Your design is shown in Fig. 1 (unfortunately your aesthetic sensitivities are not great). A sphere of radius r_1 consists of gold and platinum, with densities ρ_g and ρ_p. Gold is located within a cylinder of radius r_2, as illustrated in Fig. 1, and platinum of density ρ_p fills up the rest of the sphere. Write a program that calculates the two moments of inertia of this sphere corresponding to rotation about the z and x axis. The inner cylinder is centered around the z-axis, as also shown in the figure.

Do the calculation for a platinum (21090 kg/m^3) sphere of radius 5 cm with an inner gold (19320 kg/m^3) cylinder of radius 1 cm.

a) Carry out the calculation using Monte Carlo sampling of the moment of inertia integral

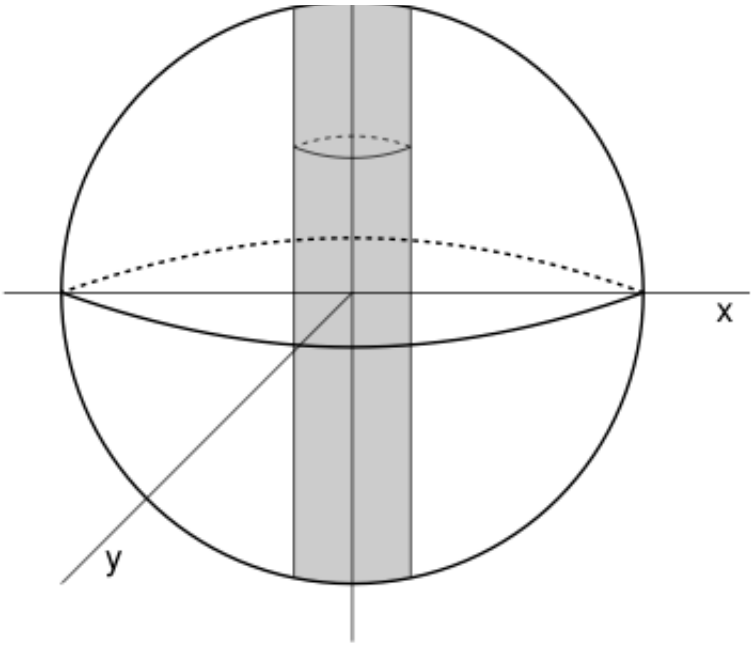
$$I = \int dx \int dz \int dz \rho(x,y,z)\, r^2(x,y,z)$$

where $r(x,y,z)$ is the perpendicular distance of the point (x, y, z) from the axis of rotation. Show the values of I_x and I_z.

b) Do the simulation $N_s = 1000$ times and for each simulation choose N_{mc} such the accuracy is 0.1%. Show how you estimated the accuracy and what is the value of N_{mc}.

In [6]:
```
Image(filename="files/HM2_fig1.png")
```

Out[6]:



The square of the perpendicular distances to the x-axis and the z-axis are:

$r^2|_{x-axis}(x,y,z) = y^2 + z^2$

and

$r^2|_{z-axis}(x,y,z) = x^2 + y^2$

The density is given by:
$rho(x,y,z) = \rho_g \text{ if } x^2 + y^2 \leqslant r_2^2$
$rho(x,y,z) = \rho_p \text{ if } x^2 + y^2 > r_2^2$

Solution (copied almost directly and adapted from AM207_GR_Joey_Goodknight_Stephanie_Valleau_HM2 abd a bit from Bancel_Jean-Remy and Marion et.al.)

We evaluated the moment of inertia with Monte Carlo Integration using $N_s = 1000$ and $N_{mc} = 10^6$ and obtained:

$$\begin{aligned} I_x &= 0.0109960345148 \pm 1.43263727172 \times 10^{-05} \\ I_z &= 0.0110399824561 \pm 1.43818272069 \times 10^{-05} \end{aligned}$$

Note: the code below is not the most efficient and will take a bit of time to compute!

In [9]:
```
import time
import math

import numpy as np

densityOfPlatinum = 21090.00 #kg/m^3
densityOfGold     = 19320.00

platinumSphereRadius = .050 #m
goldCylinderRadius = .010 #m

boxMax = platinumSphereRadius
boxMin = -boxMax
boxLength = boxMax - boxMin
```

```
boxVolume = boxLength**3

numberOfSimulations = 1000
numberOfMonteCarloSamples = 10**6


#define a function of x, y, and z to give us the density
def densityFunctionOfWeirdObject(x, y, z):
    sphericalRadius = (x**2 + y**2 + z**2)**.5
    #if the point lies outside the sphere, automatically output 0
    if  sphericalRadius >  platinumSphereRadius:
        return 0
    else:
        #check to see if the point is within the z-aligned cylinder
        cylinderRadius = (x**2 + y**2)**.5
        if cylinderRadius > goldCylinderRadius:
            #if it is not, return the density of the sphere. otherwise of
the cylinder
            return densityOfPlatinum
        else:
            return densityOfGold


#define the integrand for the rotation about x
def IxIntegrand(x, y, z):
    distFromXSquared = z**2 + y**2
    return distFromXSquared * densityFunctionOfWeirdObject(x, y, z)

#define the integrand for the rotation about x
def IzIntegrand(x, y, z):
    distFromZSquared = x**2 + y**2
    return distFromZSquared * densityFunctionOfWeirdObject(x, y, z)


#make my functions vectorized

vectorizedIxIntegrand = np.vectorize(IxIntegrand)
vectorizedIzIntegrand = np.vectorize(IzIntegrand)

startTime = time.time()
calculatedIx = []
calculatedIz = []

for i in range(numberOfSimulations):
    randomNumbers = (boxMax - boxMin) * np.random.rand(3,
numberOfMonteCarloSamples) + boxMin
    IxIntegrands = vectorizedIxIntegrand(randomNumbers[0],
randomNumbers[1], randomNumbers[2])
    IzIntegrands = vectorizedIzIntegrand(randomNumbers[0],
randomNumbers[1], randomNumbers[2])

    Ix = boxVolume * np.sum(IxIntegrands) / numberOfMonteCarloSamples
    Iz = boxVolume * np.sum(IzIntegrands) / numberOfMonteCarloSamples

    calculatedIx.append(Ix)
    calculatedIz.append(Iz)
    if i %100==0:
```

```
        print "On %d th simulation" % i



averageIx = np.average(calculatedIx)
errorIx = np.std(calculatedIx)

averageIz = np.average(calculatedIz)
errorIz = np.std(calculatedIz)


timeElapsed = time.time() - startTime
print "Ix = %s +/- %s" % (averageIx, errorIx)
print "Iz = %s +/- %s" % (averageIz, errorIz)
print "time elapsed is % seconds" % timeElapsed
```

```
On 0 th simulation
On 100 th simulation
On 200 th simulation
On 300 th simulation
On 400 th simulation
On 500 th simulation
On 600 th simulation
On 700 th simulation
On 800 th simulation
On 900 th simulation
Ix = 0.0109966166929 +/- 1.48713021092e-05
Iz = 0.0110398727794 +/- 1.47109318081e-05
time elapsed is 5419.47790194econds
```

(1b) To pick a value of Nmc which gives an error of 10^{-3}, we can use the formula for the generic error scaling of Monte Carlo to find

$$10^{-3} \sim \frac{1}{\sqrt{N_{mc}}} \implies N_{mc} \sim 10^6$$

. One can get more accurate by calculating the ratio of the variance and the average as a function of monte carlo points and find the number of points for which this ratio is less of equal to 0.1%. In this was one finds $N_{mc} \sim 1700000$ which, as expected is of the same order of magnitude as the estimate from earlier. This number will vary some depending upon the accuracy of program implementation, etc. Note that this is the total number of samples, not just those accepted as being inside the sphere

In [7]:
```
error = 4
Nx = 1657000

#first Ix
while error > .001:
    Nx=Nx+5000 #sufficient large step ensures quick decrease
    randomNumbers = (boxMax - boxMin) * np.random.rand(3, Nx) + boxMin
    IxIntegrands = vectorizedIxIntegrand(randomNumbers[0],
randomNumbers[1], randomNumbers[2])

    avf = np.average(IxIntegrands)
    avfSquared = np.average(map(lambda x: x**2, IxIntegrands))

    error =  ((avfSquared - avf**2) / Nx)**.5 / avf
    print Nx, error
```

```
print "Ix needed %s trials to get to error of less than .1%%" % str(Nx)
```

```
1662000 0.00101914245064
1667000 0.00101829517183
1672000 0.00101714302406
1677000 0.00101565265608
1682000 0.0010139478511
1687000 0.00101209983235
1692000 0.0010105835229
1697000 0.00100974119041
1702000 0.0010081335169
1707000 0.00100589016247
1712000 0.00100397705504
1717000 0.00100435800016
1722000 0.00100244029858
1727000 0.00099979192056
Ix needed 1727000 trials to get to error of less than .1%
```

More generally, one could also find the distribution of N_{mc} which gives a notion of the spread required to achieve 0.1% accuracy. Some folks did this. There was some confusion about where $N_s = 1000$ comes from. Its plucked out of thin air. The actual error estimate comes from the variance formula, running multiple simulations proves this for us and allows us to 'experimentally' calculate the variance. Some folks did that: fabulous! Here I have taken a bit of a short circuit by running the multiple experiments only once in the interest of speed.

Question 2

Again it's Valentine's day and a single Harvard student is looking for a date. Your fellow student is looking for a (fe)male student that is older than 23 years old,
taller than 170cm but shorter than 180cm and weighs less than 160 lbs.

Assume the probability distribution for age (a), weight (w) and height (h) is given by

$$P(a,w,h) = \lambda \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu\right)$$

where $\mathbf{x} = [a,w,h]$,

$\mu = [\mu_a,\mu_w,\mu_h] = [22.2,162,176]$

is a vector representing the mean values , and Σ is the covariance matrix given by:

$$\Sigma = \begin{bmatrix} 2.3 & 1.1 & 4.2 \\ 1.1 & 13.5 & 3.3 \\ 4.2 & 3.3 & 9.2 \end{bmatrix}$$

Calculate the probability of the student finding someone to date.

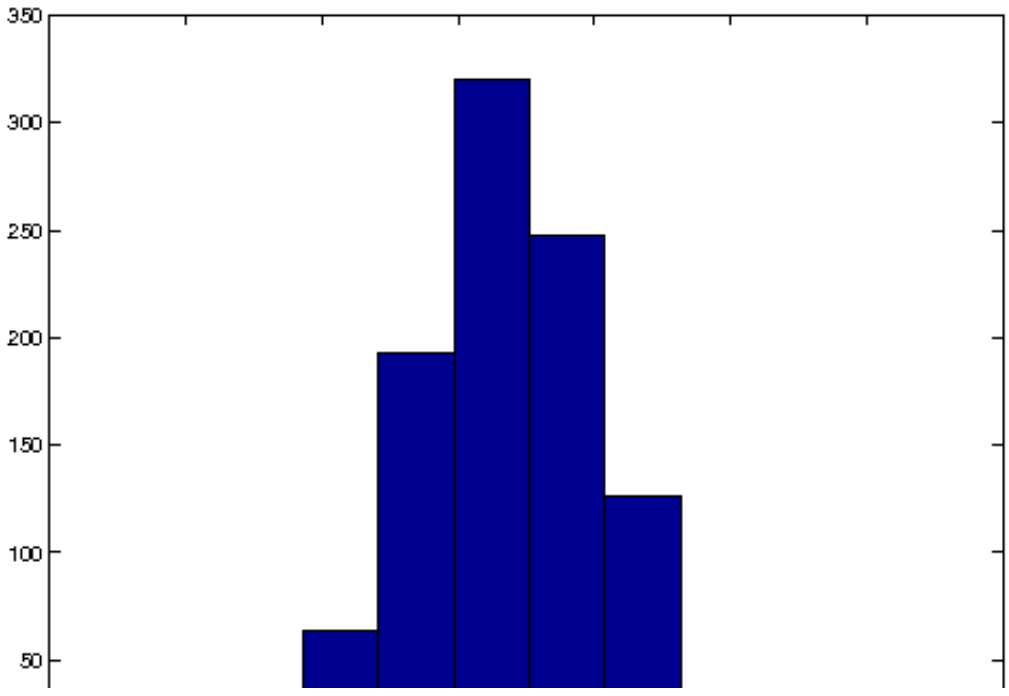
Solution (copied almost directly and adapted from AM207_GR_MARION_DIERICKX_XINYI_GUO_PHILIP_MOCZ_HM2)

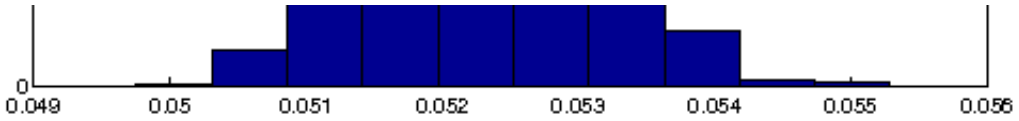
We generate $N = 10^5$ ages, heights, and weights according to the multivariate normal distribution with specified mean vector and covariance matrix using Matlab's built-in mvnrnd function, which returns a $N \times D$ matrix R of random vectors chosen from the multivariate normal distribution with $1 \times D$ mean vector μ and $D \times D$ covariance matrix Σ. The probability of finding a date is then the fraction of people out of N who meet the criteria for age, weight, and height. We repeat this experiment $N_s = 1000$ times, and estimate the error by taking the standard deviation of the resultant matrix.

In [10]:
```
%%matlab
```

```
clear all
clc

Sigma = [2.3  1.1  4.2;
         1.1 13.5  3.3;
         4.2  3.3  9.2];
mu = [22.2 162 176];

Ntrial = 1000;
Nsample = 10^5;
prob_date = zeros(Ntrial,1);
for i = 1:Ntrial
    % generate random multivariate sample
    x = mvnrnd(mu,Sigma, Nsample);
    % pick out eligible dates
    possible_dates = x(x(:,1)>23 & x(:,2)<160  & x(:,3)>170 & x(:,3)<180,:
);
    [m,n] = size(possible_dates);
    prob_date(i) = m/Nsample;
end

mprob = mean(prob_date)
eprob = std(prob_date)
figure
hist(prob_date)
```

```
mprob =

    0.0524


eprob =

   6.9891e-04
```



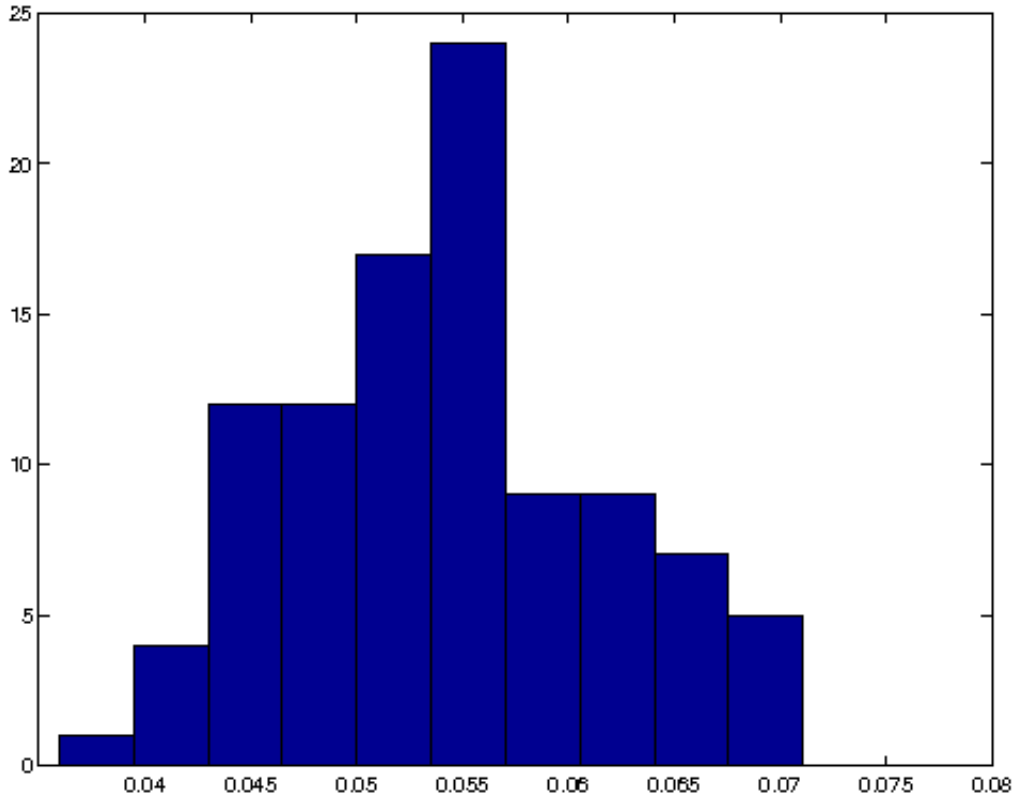


We confirm the above result by an alternative approach, namely using (vanilla) Monte Carlo integration of the probability distribution over the acceptable ranges. For proper normalization we need $\lambda = \frac{1}{(2\pi)^{3/2}|det\Sigma|^{1/2}}$. We choose finite cut-offs for the acceptable range of dates that are many standard deviations away from the mean: namely a max age of 100 and a min weight of 100. The code generates 100 trials each with $N = 10^5$ sampling points. (Note: this will take very long to run)

In [11]:
```
%%matlab
clear all
clc
format long

Sigma = [2.3  1.1  4.2;
         1.1 13.5  3.3;
         4.2  3.3  9.2];

mu = [22.2 162 176]';

lambda = 1/((2*pi)^(3/2)*det(Sigma)^(1/2));

P = @(x) lambda*exp(-(1/2)*(x-mu)'*(Sigma\(x-mu)));

x_min = [23 100 170]';
x_max = [100 160 180]';
% we choose finite cut-offs for acceptable ranges that are multiple std
% deviations away from the mean.

V = (x_max(1) - x_min(1))*(x_max(2) - x_min(2))*(x_max(3) - x_min(3));

Ns = 100;
Nmc = 10^5;

y = zeros(Nmc,1);
prob_date = zeros(1,Ns);
for i = 1:Ns
    for j = 1:Nmc
        x = (x_max-x_min).*rand(3,1) + x_min;
        y(j) = lambda*exp(-(1/2)*(x-mu)'*(Sigma\(x-mu)));
    end

    prob_date(i) = V*mean(y);
end

probability = mean(prob_date)
error = std(prob_date)
figure
hist(prob_date)
```

```
probability =
```

```
    0.054091285722894

error =

    0.007206973774563
```



It can be seen the answer is toughly 5.3% with an error of about .1%. This error could have been calculated by obtaining the variance on the integral

Question 3

Generate a series of random numbers that follow the Lorentzian/Cauchy distribution

$$P(x) = \begin{bmatrix} 0 & x < 0 \\ \frac{1}{2\text{atan}(5)} \frac{1}{(x-5)^2+1} & 0 \leq x \leq 10 \\ 0 & x > 10 \end{bmatrix}$$

I. Use inverse transform starting from a uniform distribution.
II. Use the rejection method to produce the same distribution.

For both methods plot a histogram of the values.

Solution (copied almost directly and adapted from AM207_Levary_Ramani_Tarigopula_HM2)

The inverse transform method exploits the fact that $F^{-1}(r) = t$ where F^{-1} is the inverse CDF of P and r is as

usual a number generated from the uniform distribution on $[0,1]$. Observing that the given distribution is already normalized, we have

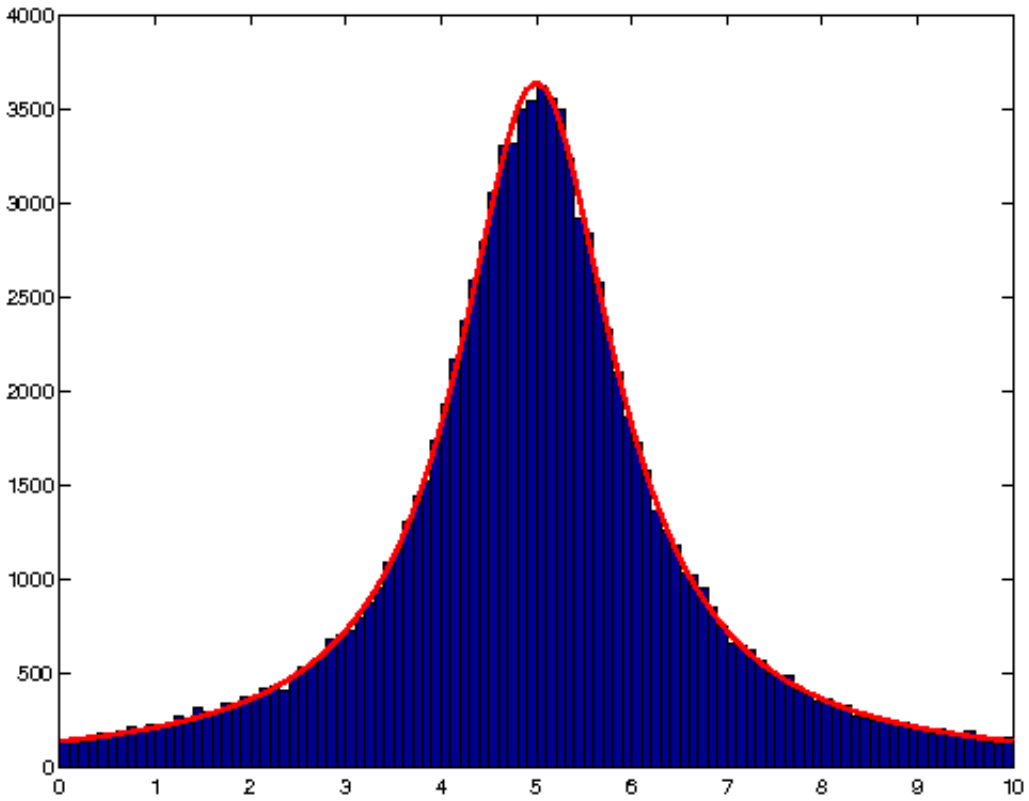
$$F(t) = \frac{1}{2atan(5)} \int_0^t \frac{1}{(x-5)^2+1} = \frac{1}{2atan(5)} \left(-atan(5-t) + atan(5)\right) = r$$

Solving for t we find

$t = 5 + \tan(2r(atan(5)) - atan(5))$

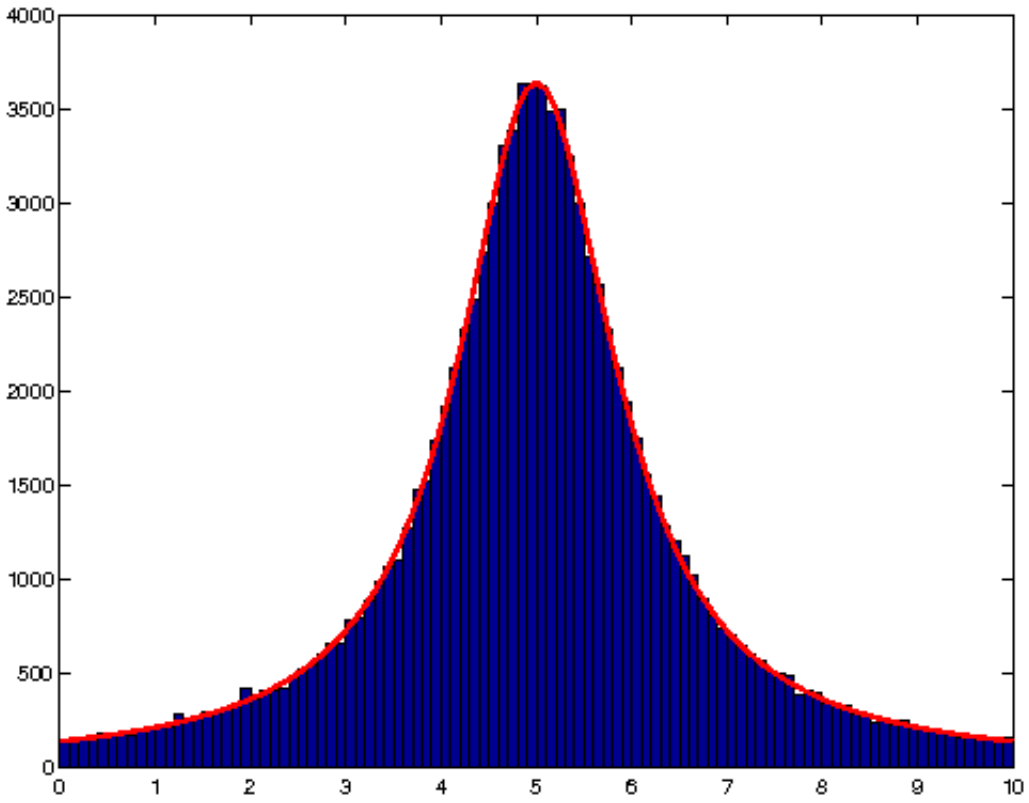
Note that sampling r on the range $[0,1]$ gives values of $t \in [0,10]$ as desired. 10^5 random numbers are generated using the inverse transform method.

In [9]:
```
%%matlab
clear all
f = @(x) 1/(2*atan(5))./((x-5).^2+1);
N=10^5;
invx=rand(1,N);
invy=5+tan(2*atan(5).*invx-atan(5));
figure
num_bins=100;
hist(invy,num_bins);
hold on;
x_sample = 0:0.1:10;
plot(x_sample,N*f(x_sample)*range(x_sample)/num_bins,'r','linewidth',2);
hold off;
```



For the rejection method, we note by inspection that the maximum of $P(x)$ occurs at $x = 5$ yielding a value of $P(x) = \frac{1}{2atan(5)}$. We thus generate random ordered pairs of numbers from the 2-D uniform distribution on the grid $[0,10] \times [0, \frac{1}{2atan(5)}]$. The rejection method is applied to sample of the order of 10^5 random numbers from the given distribution

In [7]:
```
%%matlab
f = @(x) 1/(2*atan(5))./((x-5).^2+1);
N=10^5;
x=10*rand(1,5*N);
const=1/(2*atan(5));
y=const*rand(1,5*N);
accept=y<=const*1./((x-5).^2+1);
temp=x(accept);
figure
num_bins=100;
hist(temp(1:N),num_bins);
hold on;
x_sample = 0:0.1:10;
plot(x_sample,N*f(x_sample)*range(x_sample)/num_bins,'r','linewidth',2);
hold off;
```



Question 4

Generate a random sample from a two dimensional distribution using the rejection method. Estimate the number of objects you accept (N_+) and reject (N_-).

$$P(x,y) = \exp\left(-(x^2 + \cos(x)\, y^2)\right)$$

for $-2 < x < 2$
and $-2 < y < 2$.

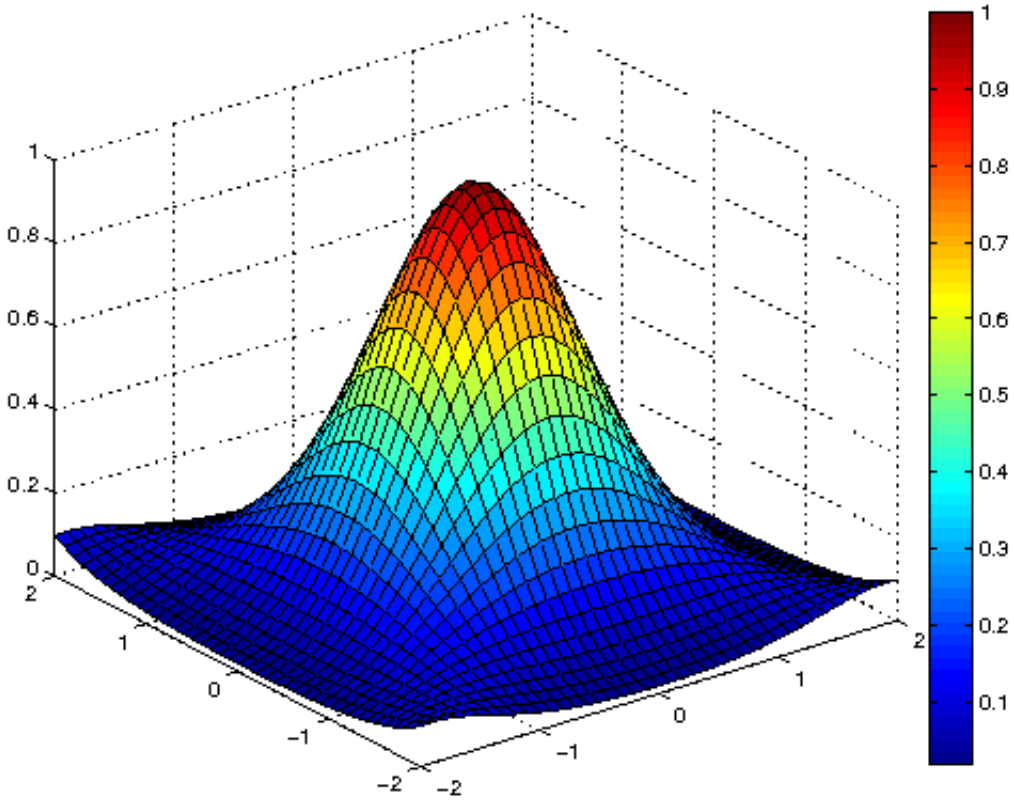
Think of at least two ways of improving the ratio of N_+/N_- and implement it. By how much has this method improved the ratio ?

Solution (copied almost directly and adapted from AM207_OVERVELDE_HM2.pdf)

In order to draw samples from this distribution, we create a box around the maximum and minimum values of x, y and $P(x;y)$. Then, we draw random samples $(x_i;y_i;P_i)$ from this box and reject samples if $P(x_i;y_i) < P_i$. We fi

nd that for $N = 2 * 10^4$ samples we have $\frac{N+}{N-} = 0.46/1.54$, or about 30%

In [3]:
```
%%matlab
%Part 1: Plot function to integrate with Monte Carlo

clear all
clc

x=-2:0.1:2;
y=-2:0.1:2;

[xm,ym]=meshgrid(x,y);

P=exp(-(xm.^2+cos(xm).*ym.^2));

figure(1)
surf(xm,ym,P), colorbar
```



In [4]:
```
%%matlab
%Get acceptance ratio
```

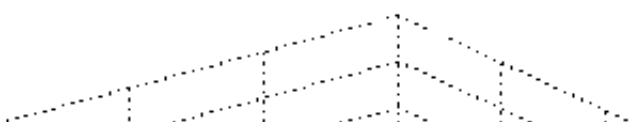
```
xmin=-2; xmax=2; ymin=-2; ymax=2; Pmin=0; Pmax=1;
VBox=(xmax-xmin)*(ymax-ymin)*(Pmax-Pmin);
iter=0;
dN=1; N=2e4;
Naccept=0;
Nreject=0;

for iter=1:1000
    x1=(xmax-xmin)*rand(N,1)+xmin;
    y1=(ymax-ymin)*rand(N,1)+ymin;
    P1=(Pmax-Pmin)*rand(N,1)+Pmin;
    Pt=exp(-(x1.^2+cos(x1).*y1.^2));
    Naccept(iter)=sum(P1<=Pt);
    Nreject(iter)=sum(P1>Pt);
end
mean(Naccept)/mean(Nreject)
std(Naccept)
```

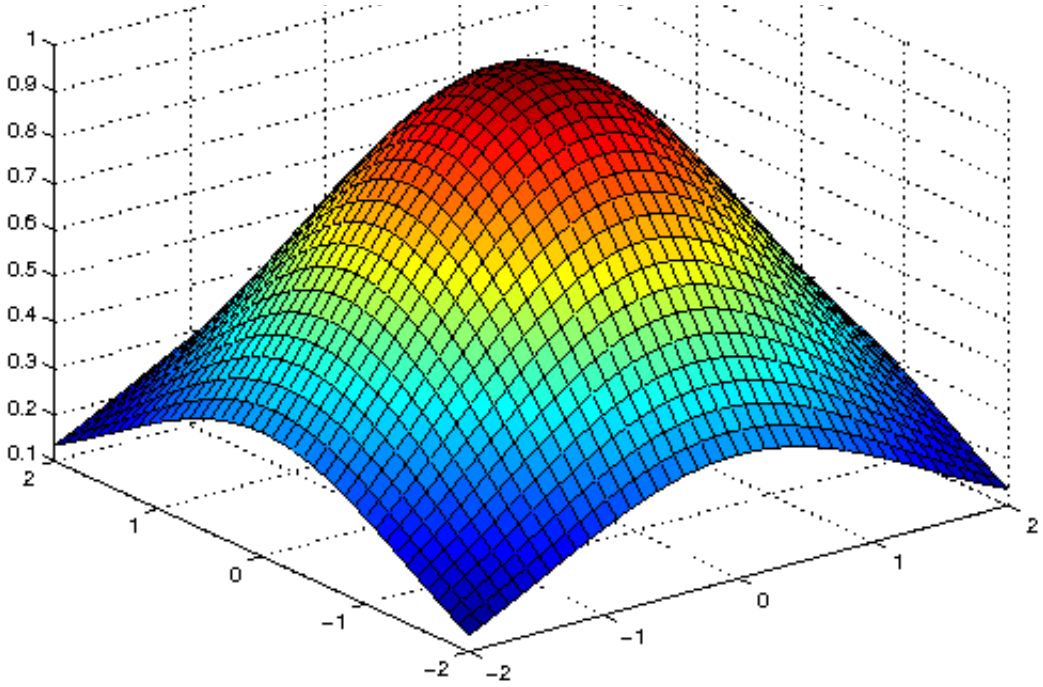
```
ans =

    0.2974


ans =

   57.3792
```

Rejection Method on Steroids as an alternative

We use
$P_c(x;y) = \exp(-(x^2+y^2)/4)$
as the distribution function. The function that needs to be integrated and the function used for the distribution is shown below. Now, we have reduced the volume to draw from and thus we reduce the number of samples that will be rejected

In [5]:
```
%%matlab
%Plot covering function for rejection method on steroids

clear all, clc;
dx=0.1; dy=0.1;
x=-2:dx:2;
y=-2:dy:2;

[xm,ym]=meshgrid(x,y);


Pa=exp(-(xm.^2+ym.^2)/4);       %Covering Distribution function
figure
surf(xm,ym,Pa)
```





In [6]:
```
%%matlab
%Results rejection method on steroids
clear all, clc;

xmin=-2; xmax=2; ymin=-2; ymax=2; Pmin=0; Pmax=1;
iter=0;
Naccept=0;
Nreject=0;

Ni=2e4

for iter=1:1000
    U1=rand(Ni,1)*exp(1/2)*(1-exp(-4));

    U2=rand(Ni,1);
    r=sqrt(-2*log(1-U1/exp(1/2)));
    max(max(r));
    theta=2*pi*U2;
    x1=r.*cos(theta);
    y1=r.*sin(theta);
    I=and(abs(x1)<2,abs(y1)<2);
    N=sum(I);
    x1=x1(I);
    y1=y1(I);
    P1=rand(N,1);
    Pt=exp(-(x1.^2+cos(x1).*y1.^2));
    Naccept(iter)=sum(P1<=Pt);
    Nreject(iter)=sum(P1>Pt)+Ni-N;
end

mean(Naccept)/mean(Nreject)
std(Naccept)
```

```
Ni =
```

```
        20000


ans =

    0.5666


ans =

   70.9301
```

In this approach our ratio has improved to 56%

Using a discrete stratification method as a rejection on steroids

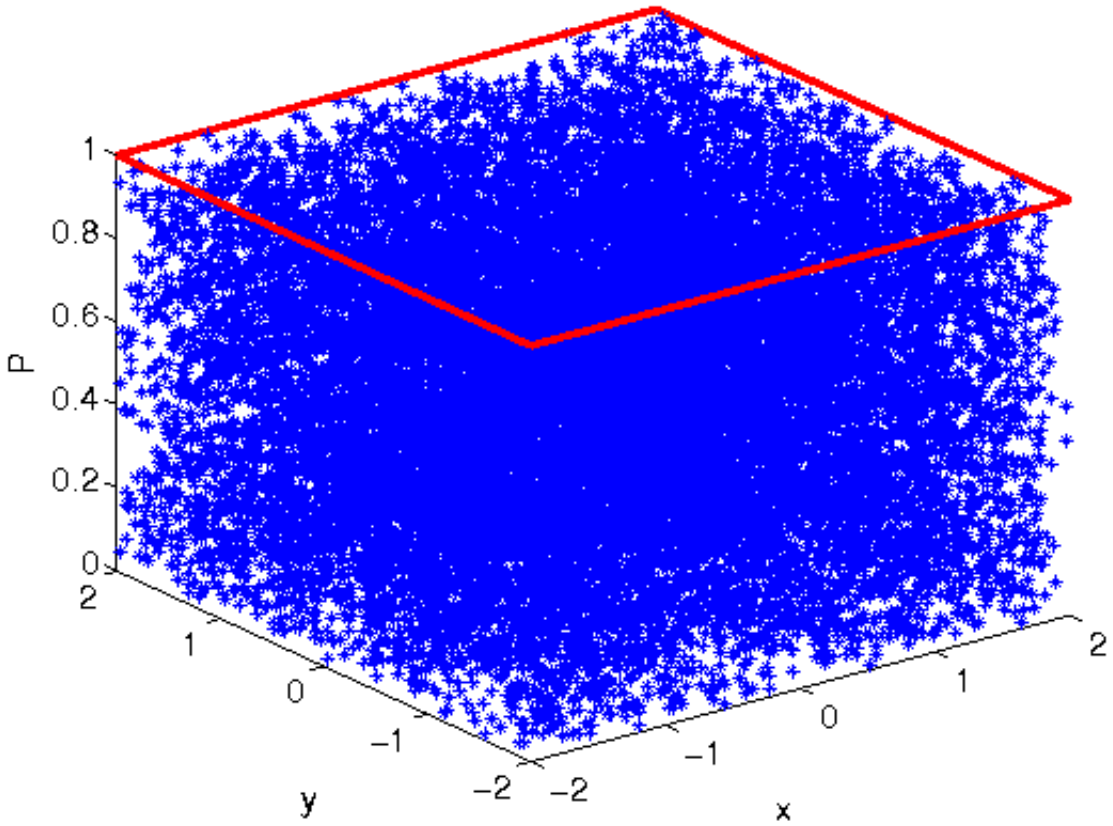
Let's say we both divide the x and y axis in d pieces. For each of these pieces we apply a rejection method, but now with altered distribution for the probabilities $P_i = U(0, P_{max})$. We thus change the limit depending on the function $P(x,y)$ value. Since this function does not have many local optima, we can find the P_{max} in each subdomain by looking at it's 4 corners and by looking at the center of the cell. We then make boxes of fixed size by ceiling the P_{max} value to multiples of $dP = 0.1$). Thus, for example, for P ranging from 0 to 1 we can have 11 discrete values. We use a total of $N = 2 * 10^4$ samples in the domain, and we divide these samples over the $\frac{d^2}{dP}$ pieces of our box. We now take $d = 1, 2, 4, 5, 10, 20$. Finally, we reject the samples for which $P_i > P(x_i, y_i)$. In the last figure the $N+$ and $N-$ values are given for the used d. Clearly, using more cells increases the function fit and thus reduces the ratio between accepted and rejected samples. The standard deviaton is shown in the plot by the errorbar and is found by repeating the simulation 1000 times. Note further that for $d = 1, 2$ we have the rejection method.

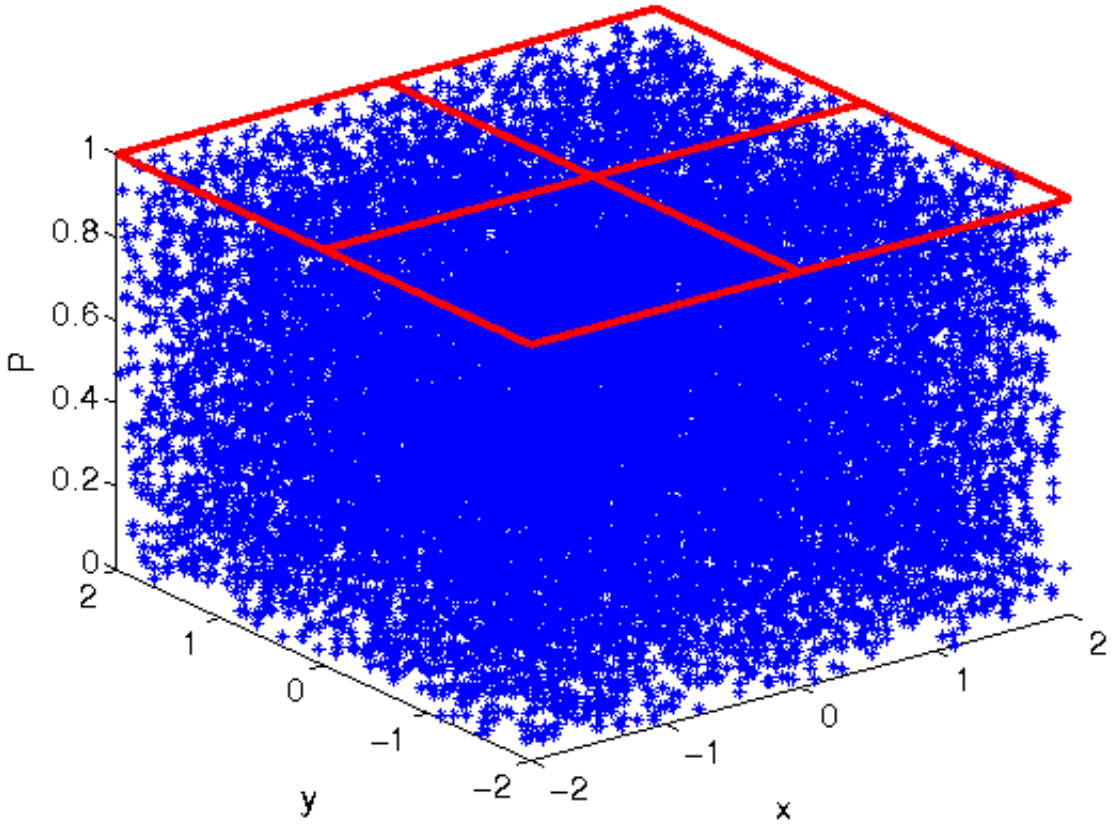
In [7]:
```
%%matlab
%Part 3: Plot integration method discrete rejection method on steroids

clear all, clc;

xmin=-2; xmax=2; ymin=-2; ymax=2; Pmin=0; Pmax=1;

rep=0;
d1=[1,2,4,5,10,20];
for d=d1
    rep=rep+1;
    d
    dP=0.1;
    dx=(xmax-xmin)/d;
    dy=(ymax-ymin)/d;

    figure,hold on
    %Set up mesh and maximum values
    for i=1:d
        for j=1:d
            M(i,j).x=dx*[(i-1) i i (i-1) i-1/2]+xmin;
            M(i,j).y=dy*[(j-1) (j-1) j j j-1/2]+ymin;
            M(i,j).P=exp(-(M(i,j).x.^2+cos(M(i,j).x).*M(i,j).y.^2));
            M(i,j).Pmax=ceil(max(M(i,j).P/dP))*dP;
            line([M(i,j).x(1:4) M(i,j).x(1)],[M(i,j).y(1:4)
M(i,j).y(1)],ones(5,1)*M(i,j).Pmax,'color','r','linewidth',3)
```

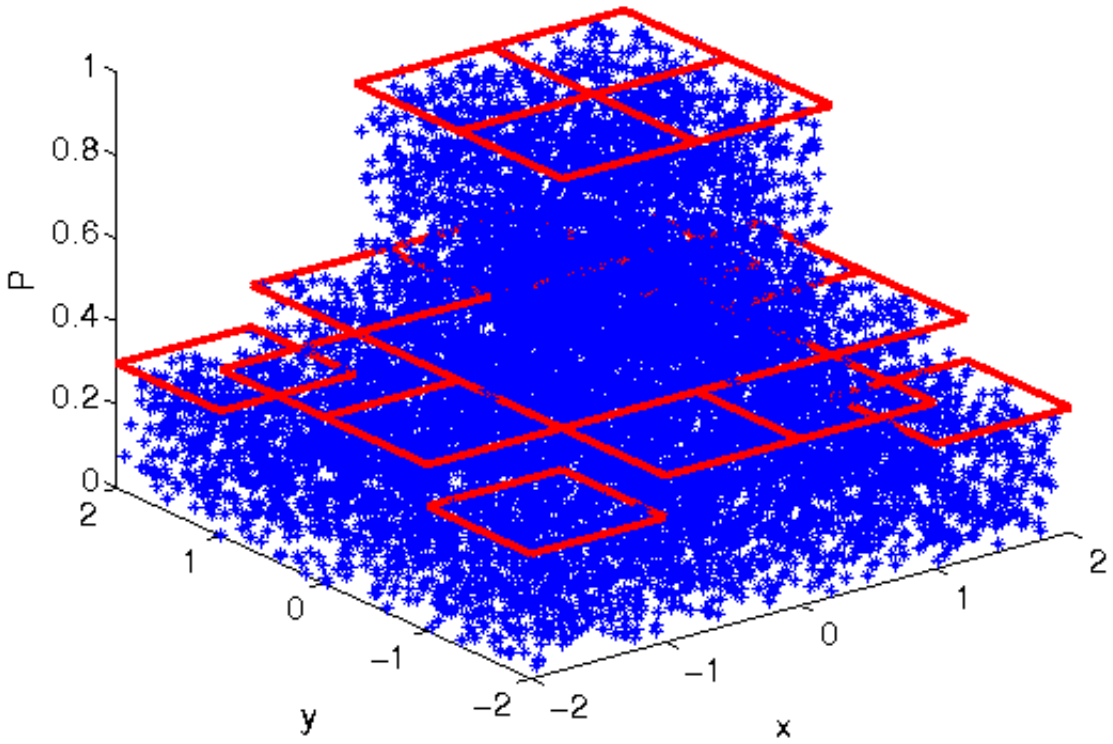
```
        end
    end

    N=2e4;

    dPN=dP*N/d^2;
    dN=1;
    Naccept=0;
    Nreject=0;
    for i=1:d
        for j=1:d
            Ni=round(M(i,j).Pmax/dP*dPN/dN);
            x1=(M(i,j).x(2)-M(i,j).x(1))*rand(Ni,1)+M(i,j).x(1);
            y1=(M(i,j).y(3)-M(i,j).y(1))*rand(Ni,1)+M(i,j).y(1);
            P1=(M(i,j).Pmax-Pmin)*rand(Ni,1)+Pmin;
            Pt=exp(-(x1.^2+cos(x1).*y1.^2));
            Naccept=Naccept+sum(P1<=Pt);
            Nreject=Nreject+sum(P1>Pt);
            plot3(x1,y1,P1,'*','color','b')
        end
    end

    xlabel('x','fontsize',16)
    ylabel('y','fontsize',16)
    zlabel('P','fontsize',16)
    set(gca,'fontsize',16)
    view(3)
    drawnow;
end
```

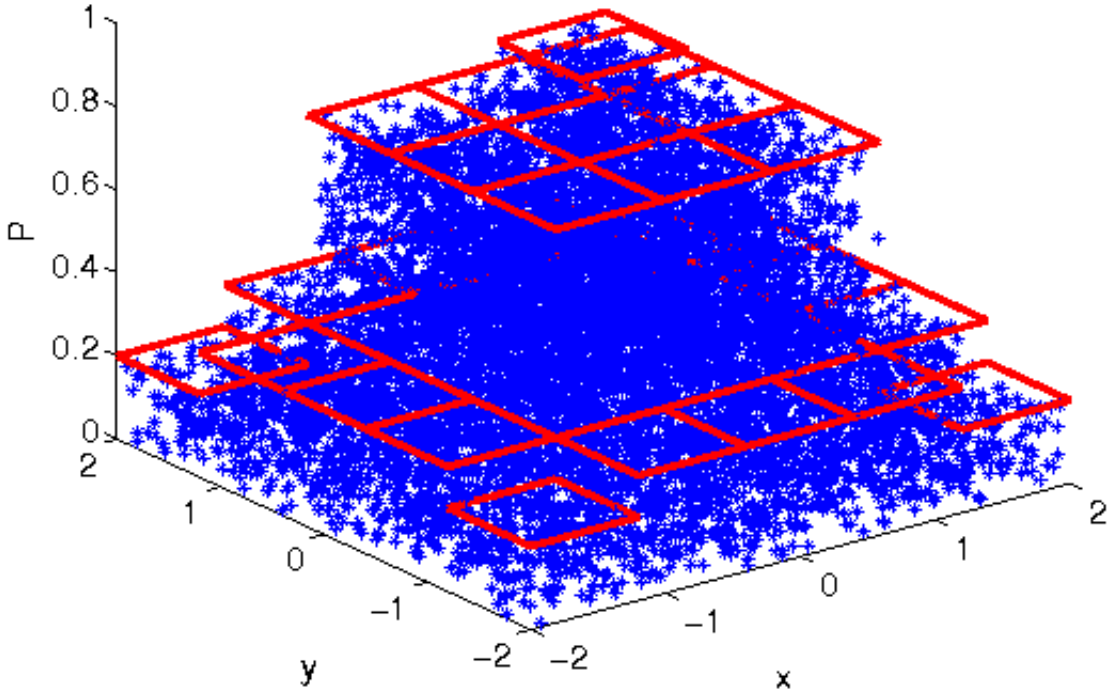
```
d =

     1


d =

     2


d =

     4


d =

     5


d =

    10


d =
```

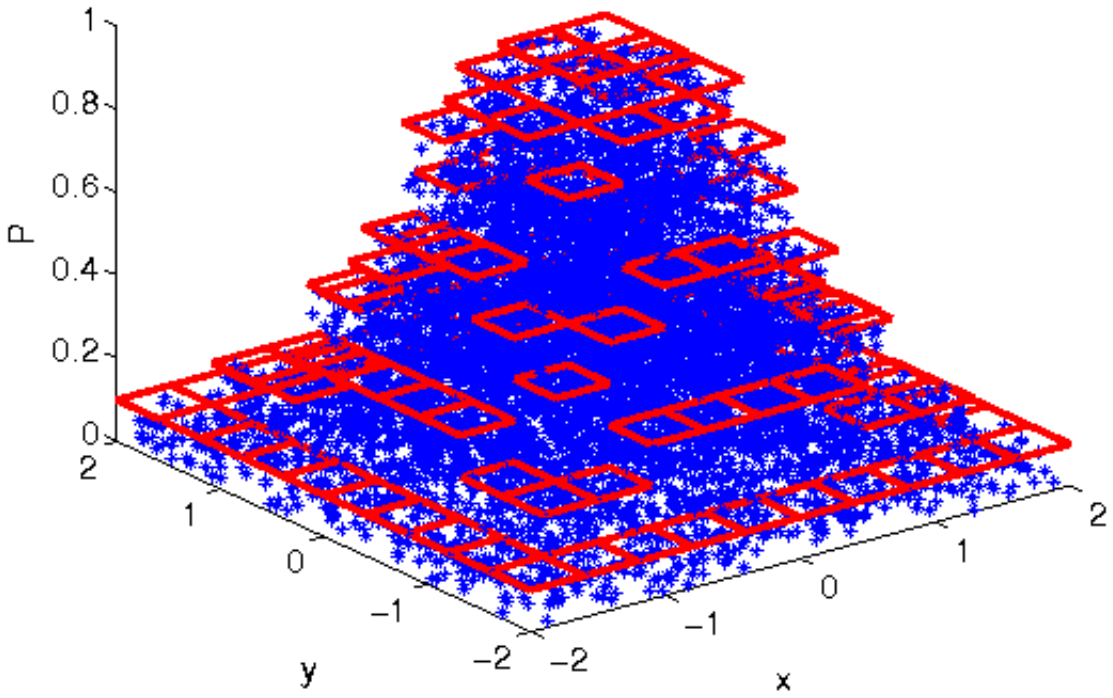
20

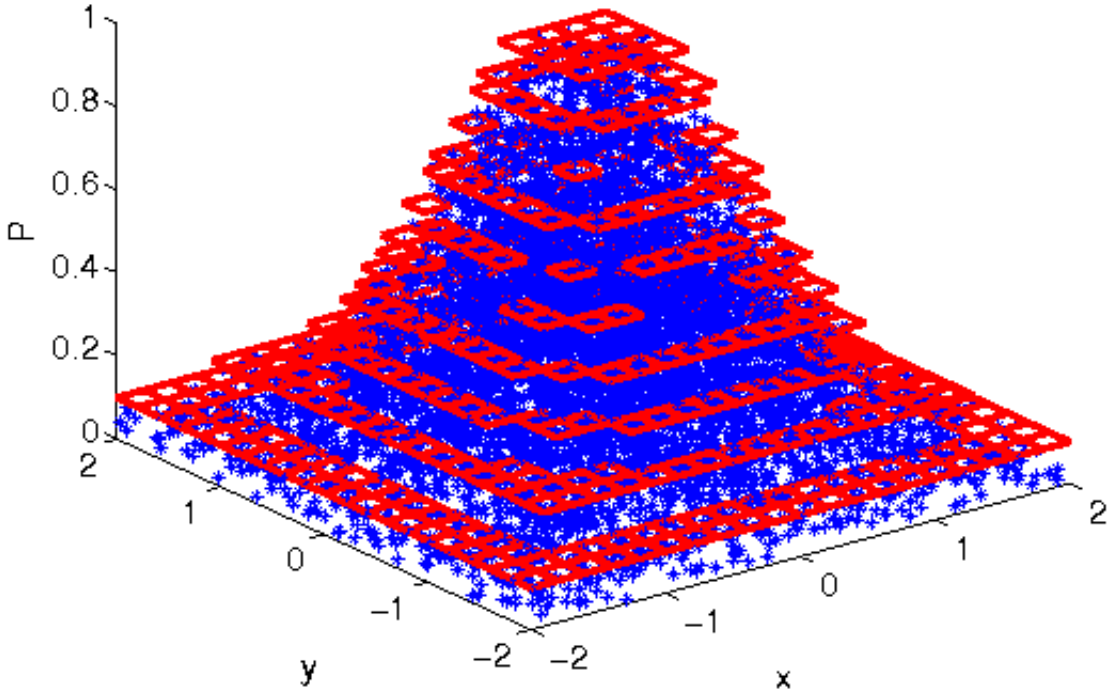





P
1
0.8
0.6
0.4
0.2
0
2
1
0
−1
−2
y
−2
−1
0
1
2
x


P
1
0.8
0.6
0.4
0.2
0
2
1
0
−1
−2
y
−2
−1
0
1
2
x



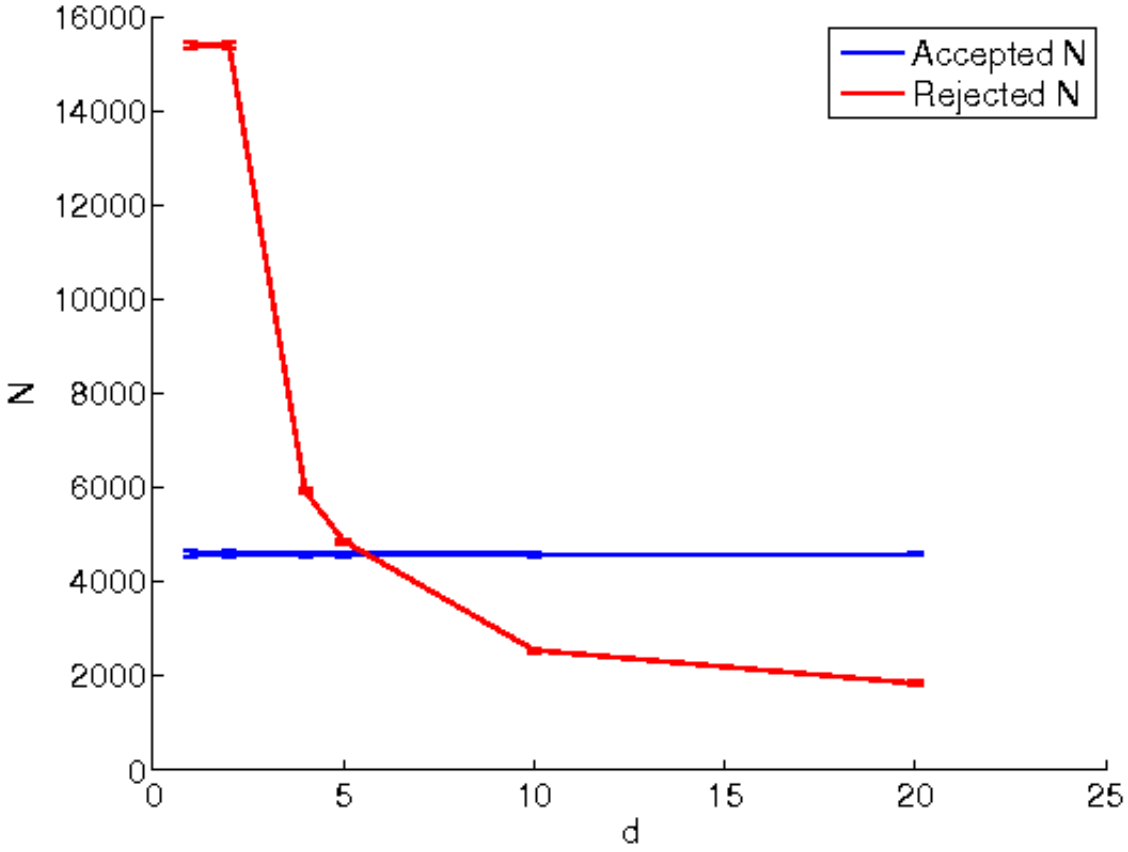


In [8]:
```
%%matlab
%Results integration method discrete rejection method on steroids
```

```
clear all, clc;

xmin=-2; xmax=2; ymin=-2; ymax=2; Pmin=0; Pmax=1;

rep=0;
d1=[1,2,4,5,10,20];
for d=d1
    rep=rep+1;
    d
    dP=0.1;
    dx=(xmax-xmin)/d;
    dy=(ymax-ymin)/d;
    %Set up mesh and maximum values
    for i=1:d
        for j=1:d
            M(i,j).x=dx*[(i-1) i i (i-1) i-1/2]+xmin;
            M(i,j).y=dy*[(j-1) (j-1) j j j-1/2]+ymin;
            M(i,j).P=exp(-(M(i,j).x.^2+cos(M(i,j).x).*M(i,j).y.^2));
            M(i,j).Pmax=ceil(max(M(i,j).P/dP))*dP;
        end
    end

    N=2e4;

    dPN=dP*N/d^2
    dN=1;
    for iter=1:100
        Naccept=0;
        Nreject=0;
        for i=1:d
            for j=1:d
                Ni=round(M(i,j).Pmax/dP*dPN/dN);
                x1=(M(i,j).x(2)-M(i,j).x(1))*rand(Ni,1)+M(i,j).x(1);
                y1=(M(i,j).y(3)-M(i,j).y(1))*rand(Ni,1)+M(i,j).y(1);
                P1=(M(i,j).Pmax-Pmin)*rand(Ni,1)+Pmin;
                Pt=exp(-(x1.^2+cos(x1).*y1.^2));
                Naccept=Naccept+sum(P1<=Pt);
                Nreject=Nreject+sum(P1>Pt);
            end
        end

        Na(iter,rep)=Naccept;
        Nr(iter,rep)=Nreject;
    end
end

figure,hold on
errorbar(d1,mean(Na),std(Na),'b','linewidth',2)
errorbar(d1,mean(Nr),std(Nr),'r','linewidth',2)
set(gca,'fontsize',16)
xlabel('d','fontsize',16)
ylabel('N','fontsize',16)
legend('Accepted N','Rejected N')
```

```
d =

     1
```

```
     1

dPN =

        2000

d =

     2

dPN =

   500

d =

     4

dPN =

   125

d =

     5

dPN =

    80

d =

    10

dPN =

    20

d =

    20


dPN =

     5
```


Accepted N
Rejected N
N
d
0
2000
4000
6000
8000
10000
12000
14000
16000
0
5
10
15
20
25