



Mittuniversitetet

MID SWEDEN UNIVERSITY

Institution of Information Systems and -Technology (IST)

Laboration 2

Monte Carlo Simulations

(MA069G, Mathematical Modelling 6hp)

Eric Johansson (erjo2002@student.miun.se)
Can Kupeli (caku2002@student.miun.se)
Samuel Greenberg (sagr1908@student.miun.se)

Friday 30th September, 2022

Supervisor:

Cornelia Schiebold

Magnus Eriksson

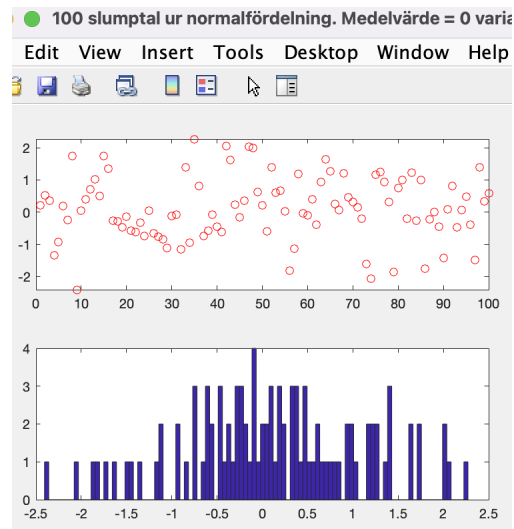
Suproakash Hazra

Contents

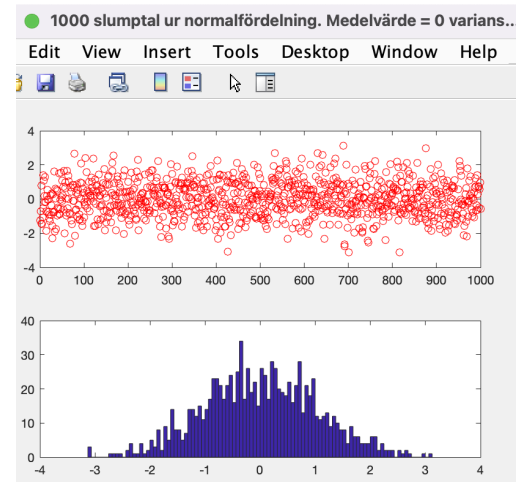
Question 1	2
(a)	2
(b)	3
(c)	4
(d)	4
Question 2	5
(a)	5
(b)	5
(c)	6
(d)	6
(e)	7
Question 3	9
(a)	9
(b)	10
(c)	10
Question 4	11

Question 1

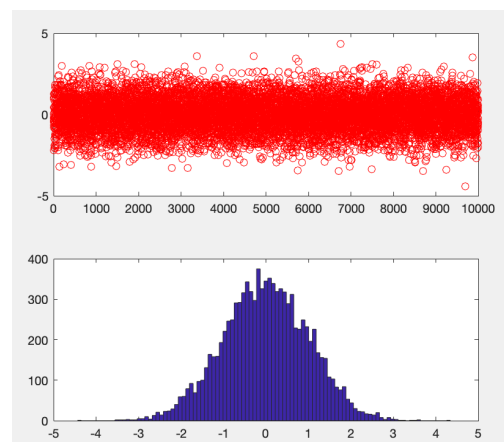
(a)



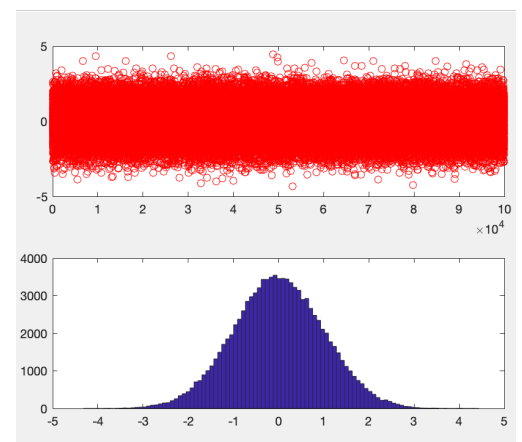
(a) 100 Randomly generated numbers



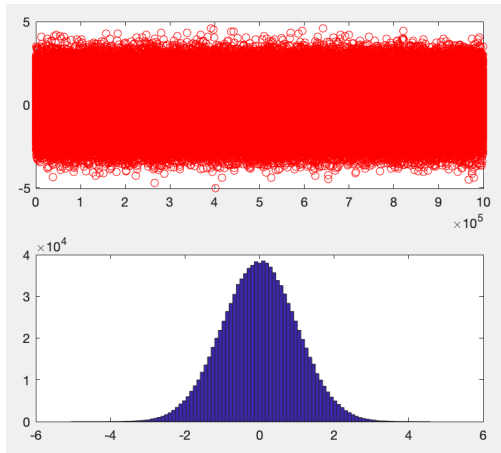
(b) 1000 Randomly generated numbers



(a) 10000 Randomly generated numbers



(b) 100000 Randomly generated numbers



(a) 1000000 Randomly generated numbers

A screenshot of a MATLAB code editor window titled 'Editor - /Users/ericjohansson/school/mat...'. It shows two tabs: 'q3b.m' and 'mcconv.m'. The 'q3b.m' tab is active and contains the following code:

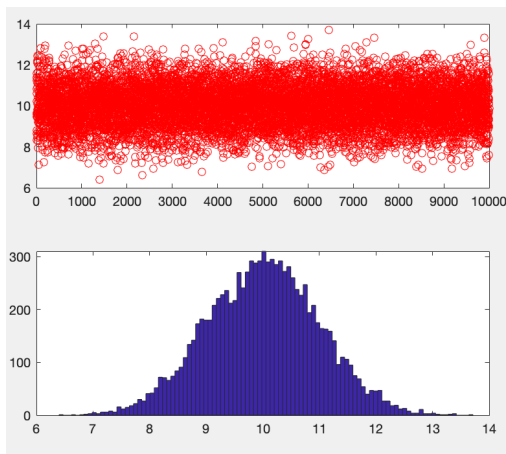
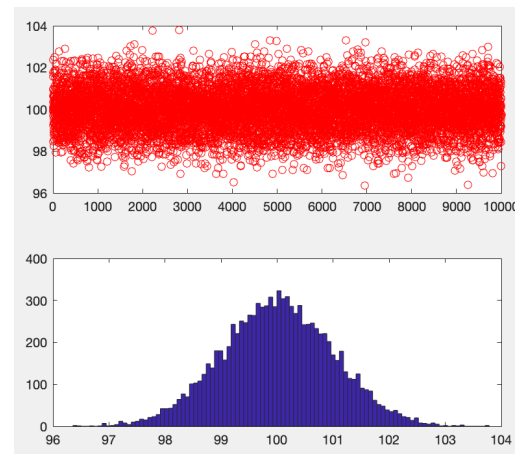
```
1 DemoLikformf(100,1)
2
```

A green checkmark is visible in the right margin next to line 1.

(b) Code for question 1

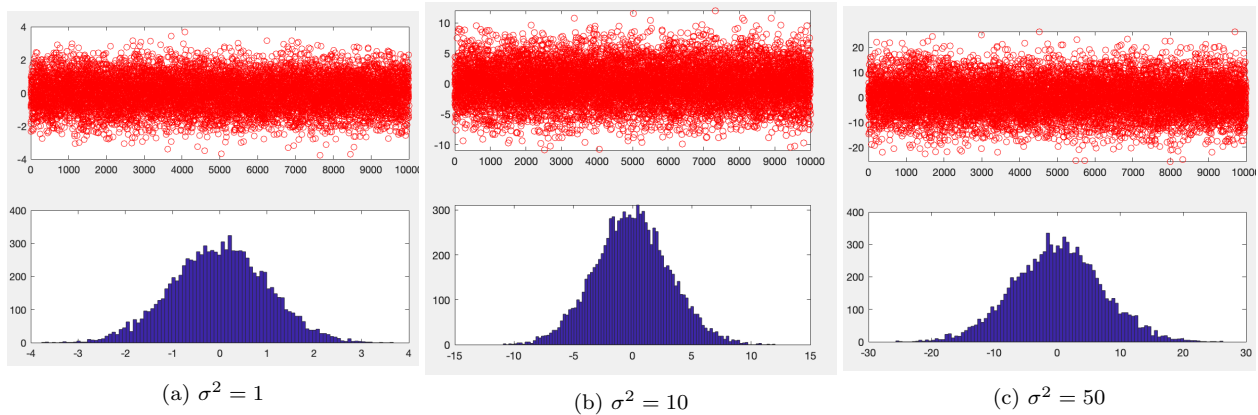
As N increases the distribution becomes more alike a Normal distribution.

(b)

(a) $\mu = 10$ (b) $\mu = 100$

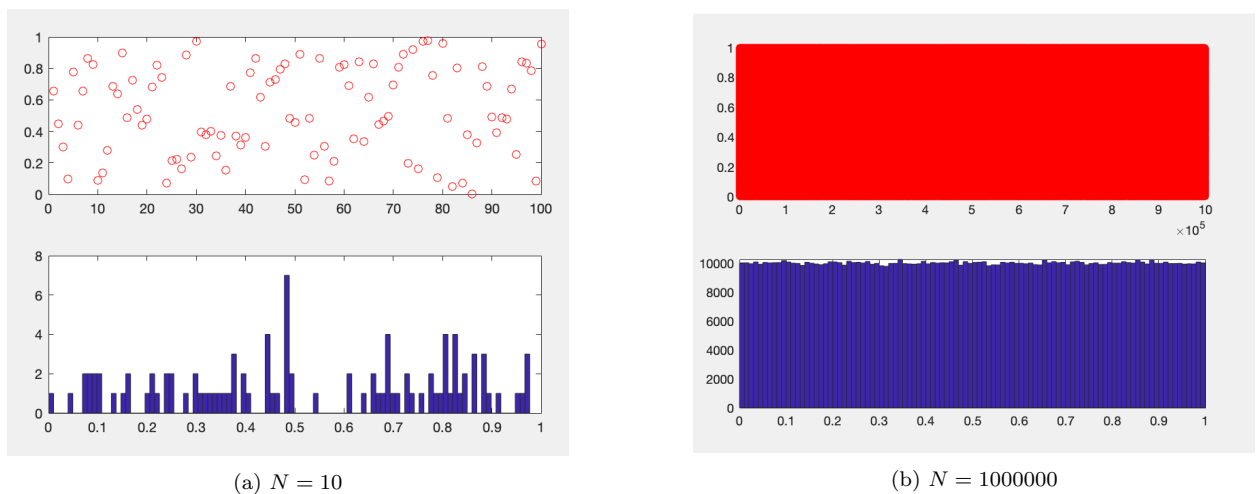
We can see that the distribution is about the same and the main difference is that they are centered around different values.

(c)



Once again the form of the distribution is mostly the same, however we can see now that the values extend more to the side as the variance increases. It doesn't increase by five times when comparing 10 to 50 because the standard deviation is $\sqrt{\text{Variance}}$. Since 50 is $5 \cdot 10$ it means that the standard deviation will increase by $\sqrt{5}$ times which is pretty accurate with the results.

(d)



We can see that as $\lim_{N \rightarrow \infty}$ the distribution becomes more like a uniform distribution.

Question 2

(a)

header

```
mean_y1 = mean(floor(1+6*rand(1,10.^3)));
mean_y2 = mean(floor(1+6*rand(1,10.^4)));
mean_y3 = mean(floor(1+6*rand(1,10.^5)));
```

(a) Code for 2a

Name ▲	Value
mean_y1	3.5240
mean_y2	3.4854
mean_y3	3.5045

(b) Results for 2a

Yes as N increases the resulting average converges to 3.5

(b)

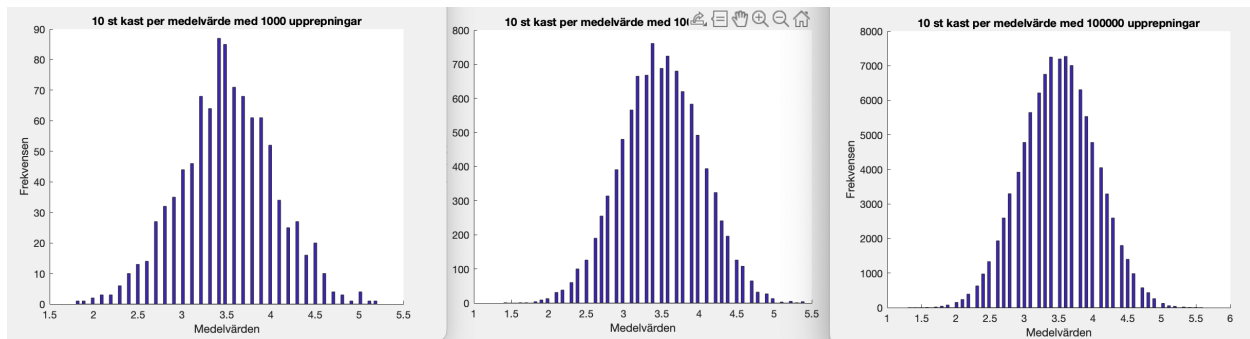


Figure 8: Results for 2b

We can see that it becomes more like a normal distribution as we repeat the test more times

(c)

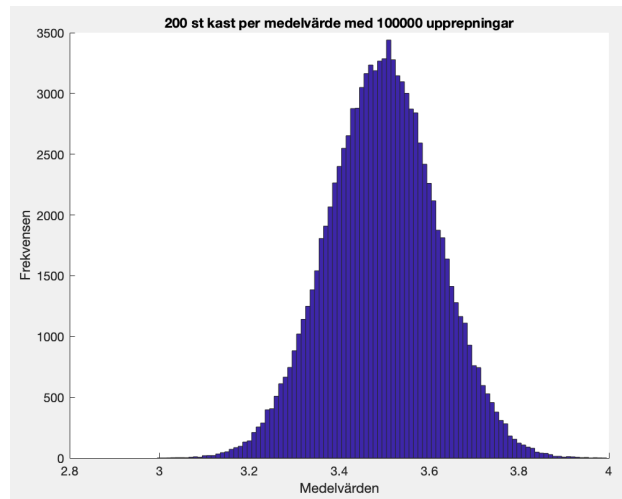


Figure 9: Results for 2c

It's possible to observe that the variance of this result is lower than that of question 2b. This is logical since we take the mean of more throws before adding them. This should create fewer outliers and thus lower the variance.

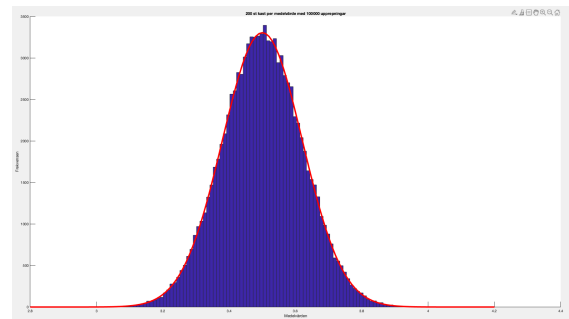
(d)

```
clear
clc
format long

n = 200; %antal kast per medelvärde
N = 10^5; %antal upprepningar
for i=1:length(N)
    figure;
    hold on
    r = zeros(1,N(i));
    for j=1:N(i)
        y = floor(1+6*rand(1,n));
        r(j) = mean(y);
    end
    hist(r,100); % Rita ett histogram med 100 intervall
    title([num2str(n), ' st kast per medelvärde med ', num2str(N(i)), ' upprepningar'])
    xlabel('Medelvärden')
    ylabel('Frekvensen')
    shg
end

sigma = sqrt(35/(12*n));
mu = 3.5;
x = 2.8:0.01:4.2;
f = 1/(sigma*sqrt(2*pi))*exp(-((x-mu).^2)/(2*sigma.^2));
plot(x,1000*f, 'Color','red','LineWidth',4);
```

(a) Code for 2d



(b) Results for 2d

The magnitude of the function f is dependent on n so we need to multiply the function to make it fit our histogram.

(e)

```

header
loops = 1000;
p = [5, 1, 1, 1, 1, 5];

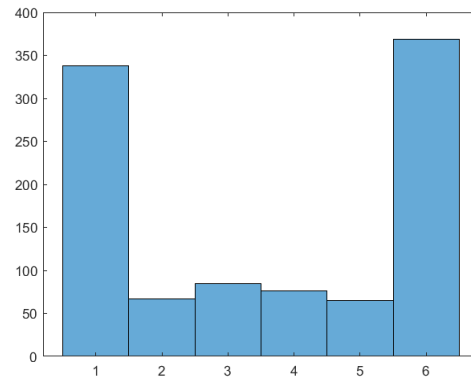
p = p / sum(p);
F = cumsum(p);

counter = 1;
inverse_sampler = zeros(1, loops);
while(counter <= loops)
    u = rand(1);
    inverse_sampler(counter) = sum(u > F) + 1;
    counter = counter + 1;
end

histogram(inverse_sampler)

```

(a) Code for 2e



(b) Histogram for 2e

p =

0.357142857142857 0.071428571428571 0.071428571428571 0.071428571428571 0.071428571428571 0.357142857142857

Figure 12: Probability of p

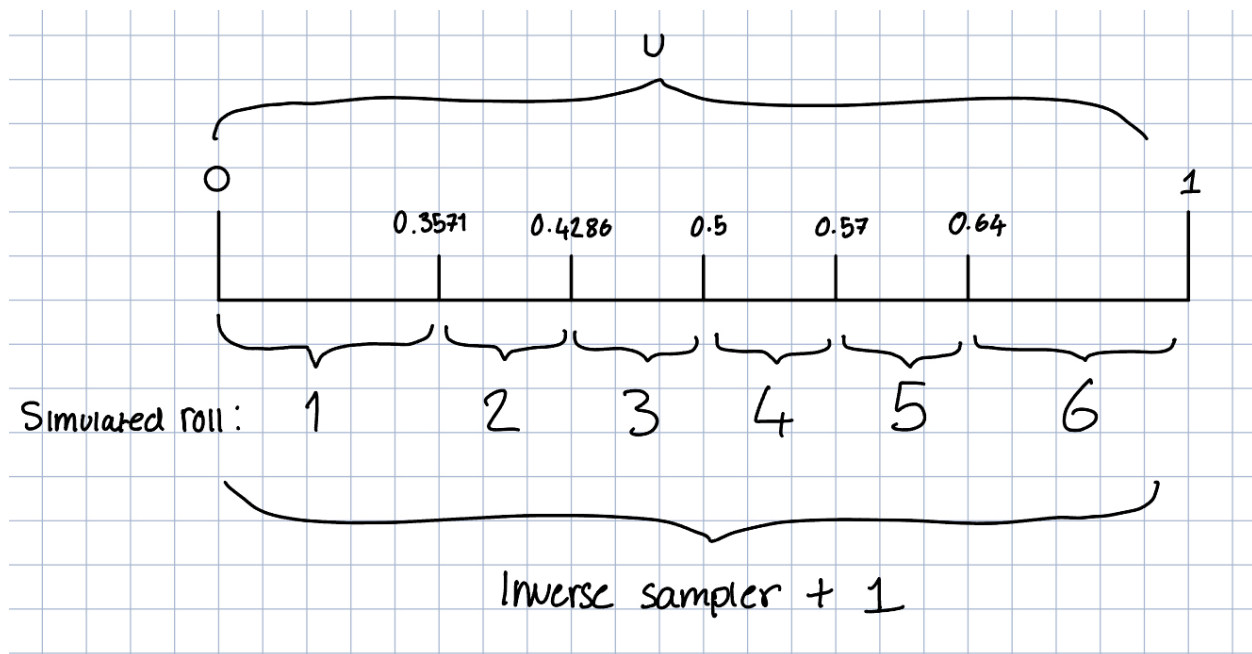


Figure 13: Visualisation of the sampled cumulative distribution function





Comparing the probabilities for the different outcomes of p with the histogram we can see that they match up well. Where $p(1)$ and $p(6)$ is roughly 5 times greater than $p(2)$, $p(3)$, $p(4)$, $p(5)$. Looking at their y-values in the histogram they are about a fifth of the height at 1 and 6.

To generate a continuous random variable with a density function, you would take the cumulative sum of the density function and then find the inverse of it. Lastly compute it as $X = F^{-1}(u)$.

The sampled cumulative distribution function creates a tool that simulates the probabilities of each roll, where their probability corresponds to a percentage of the sample. Due to the inverse sampler being equal to every element it passes in the vector it will be one value too short and will therefore require an increment.

Question 3

(a)

	mu1d100	1.7694
	mu1d1000000	1.7728
	mu2d100	3.0995
	mu2d1000000	3.1414

(a) Answer to question 3a

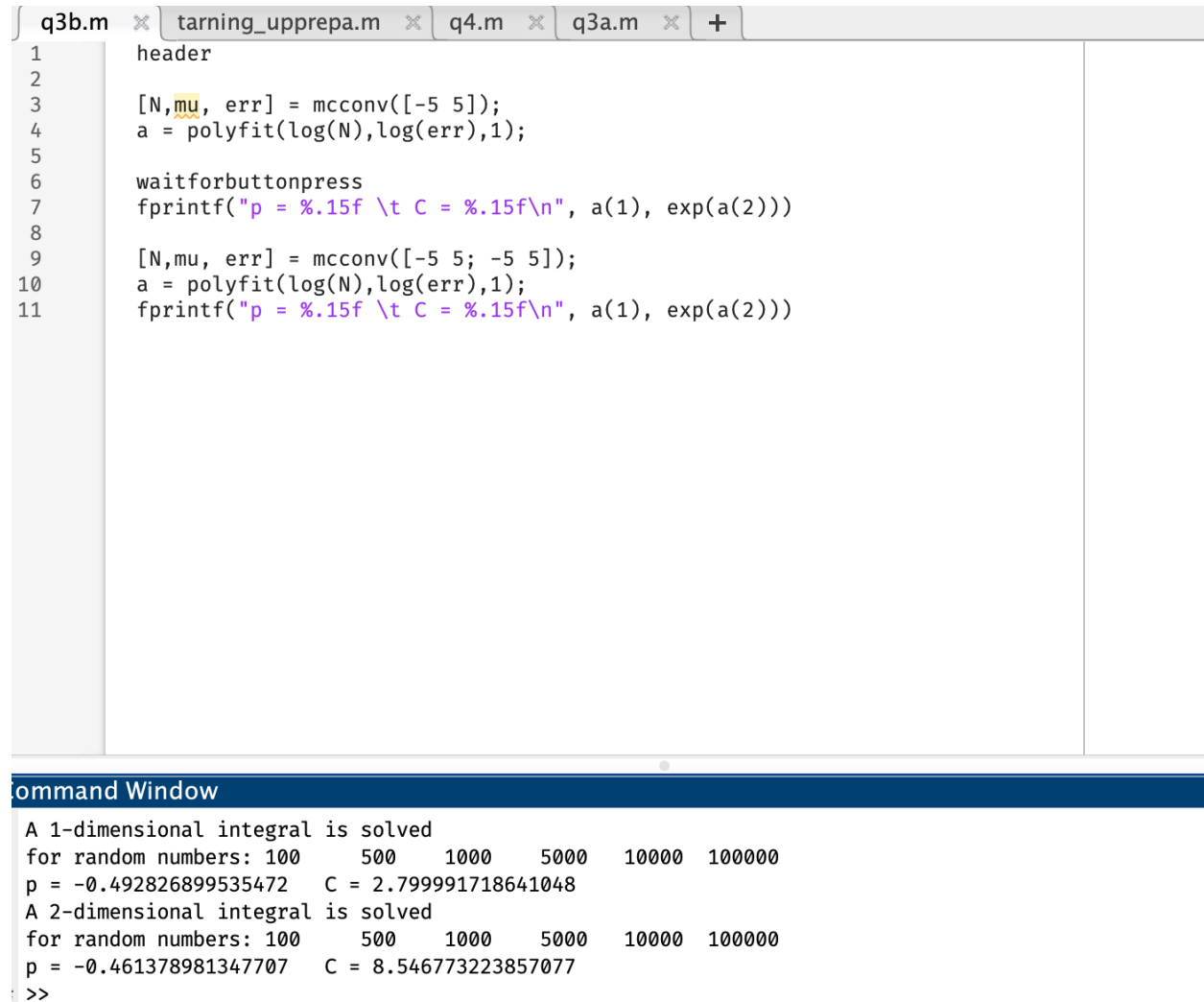
```
header

[N1d100,mu1d100,err1d100]=mcconv([-5 5],100);
waitforbuttonpress
[N1d1000000,mu1d1000000,err1d1000000]=mcconv([-5 5],1000000);
waitforbuttonpress
[N2d100,mu2d100,err2d100]=mcconv([-5 5; -5 5],100);
waitforbuttonpress
[N2d1000000,mu2d1000000,err2d1000000]=mcconv([-5 5; -5 5],1000000);
```

(b) Code for question 3a

Yes in 2D it converges to $\sqrt{\pi}$ while in 3D it converges to π . Since we're going to 2D it's realistic for the value to be the square of the one in 1D. Just like when going from a line to a square. And this is true in our case since $1D^2 = \sqrt{\pi}^2 = \pi = 2D$.

(b)



The figure shows a MATLAB script editor with a file named 'q3b.m' and a Command Window below it. The script contains two identical blocks of code, each performing a Monte Carlo integration for a 1D and a 2D integral. The Command Window shows the output of these calculations, displaying the estimated value of p and the constant C for different sample sizes N (100, 500, 1000, 5000, 10000, 100000).

```

1 header
2
3 [N,mu, err] = mcconv([-5 5]);
4 a = polyfit(log(N),log(err),1);
5
6 waitforbuttonpress
7 fprintf("p = %.15f \t C = %.15f\n", a(1), exp(a(2)))
8
9 [N,mu, err] = mcconv([-5 5; -5 5]);
10 a = polyfit(log(N),log(err),1);
11 fprintf("p = %.15f \t C = %.15f\n", a(1), exp(a(2)))

```

Command Window

```

A 1-dimensional integral is solved
for random numbers: 100    500    1000    5000    10000    100000
p = -0.492826899535472    C = 2.799991718641048
A 2-dimensional integral is solved
for random numbers: 100    500    1000    5000    10000    100000
p = -0.461378981347707    C = 8.546773223857077
>>

```

Figure 15: Question 3b

We can see that p does not change very much depending on the dimension. However C increases a lot when increasing dimension as you can see in Figure 15, almost tripled.

(c)

Monte Carlo is more efficient in higher dimensions than the Trapezoid method. This is because of the *Curse of Dimensionality* which states that when dimensionality increases the amount of data needed grows exponentially with the dimensionality.

The Trapezoid method needs 5^d points where d is the dimension.

Question 4

```
header
cumwinloss = 0;
day = 1;

while day < 366
    bs = blackjacksim(100);
    cumwinloss = cumwinloss + bs(end);
    day = day+1;
end
if cumwinloss > 0
    fprintf("Total yearly earnings is %d\n", cumwinloss)
else
    fprintf("Total yearly losses is %d\n", cumwinloss)
end
```

Figure 16: Code for question 4

Running this script usually yields a cumulative return of around -8000 to +2000. However, more often than not it ends up being a net loss for the year.