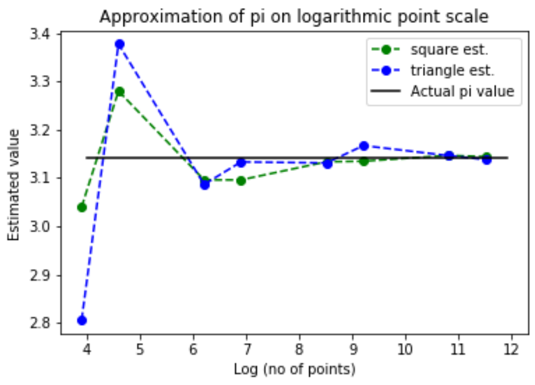
Statistical Simulation and Data Analysis (MTH511A), Autumn 2018
Indian Institute of Technology Kanpur
Programming Assignment Number 1

Student Name: Bhavy Khatri
Roll Number: 150186
Date: November 11, 2018

QUESTION 1

Part 1: Estimating pi using Monte Carlo techniques

We generated a particular number of points uniformly over a square and equilateral triangle. After that we checked that how many points are lying inside the circle inscribed by it. The ratio was used to find the value of π. The following graph was obtained for approximated value of pi vs log(no. of iterations):



No. of Points	Square Estimation	Equilateral Triangle Estimation	Actual value
50	3.04	2.80592231	3.1415926535
100	3.28	3.37749907	3.1415926535
500	3.096	3.08651454	3.1415926535
1000	3.096	3.13327991	3.1415926535
5000	3.1336	3.13120145	3.1415926535
10000	3.1348	3.1670549	3.1415926535
50000	3.14672	3.14616637	3.1415926535
100000	3.14412	3.13800841	3.1415926535

From the table and graph we can easily see that for a given number of points the square estimation beats the triangle estimation. Although both of them converge to the actual value when number of points become large. But why is it so? To answer this question let's look at how we estimated the value of π.

$$\frac{\text{Area of a circle}}{\text{Area of a Square (Triangle)}} = \frac{\text{No of points lying inside the circle}}{\text{Total no. of simulated points}}$$

The above equation works because we are simulating points uniformly over square and triangle.

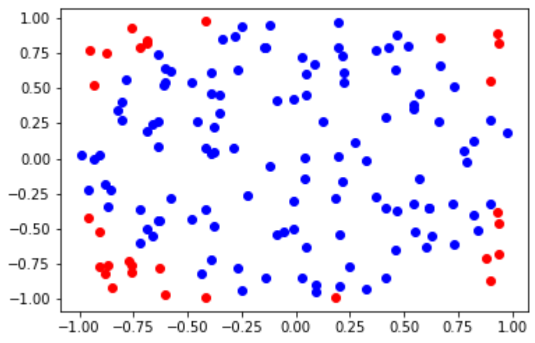
$$\pi_{square} = 4 \times \frac{\text{No of points lying inside the circle}}{\text{Total no. of simulated points in Square}}$$

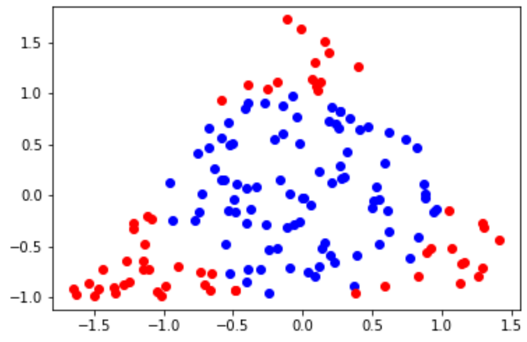
$$\alpha = \frac{\pi_{square}}{4} = \frac{\text{No of points lying inside the circle}}{\text{Total no. of simulated points in Square}}$$

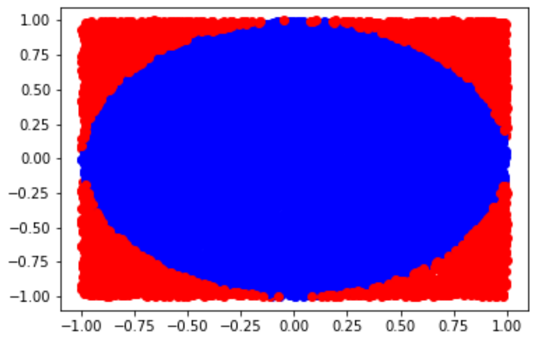
$$\pi_{triangle} = 3\sqrt{3} \times \frac{\text{No of points lying inside the circle}}{\text{Total no. of simulated points in Circle}}$$

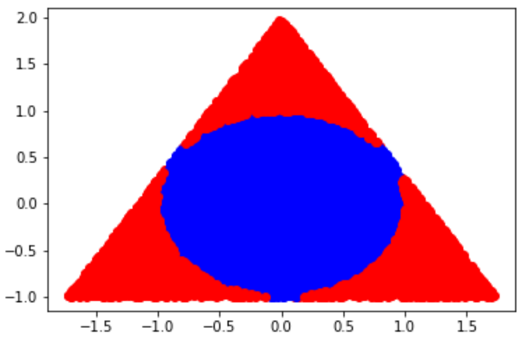
$$\beta = \frac{\pi_{circle}}{3\sqrt{3}} = \frac{\text{No of points lying inside the circle}}{\text{Total no. of simulated points in Circle}}$$

Note that $\alpha > \beta$ that means we the accepted number of points in case of square is greater than that of circle. That's the reason we are getting better estimate in case of square.
The plots for 100 and 10000 points are as follows:





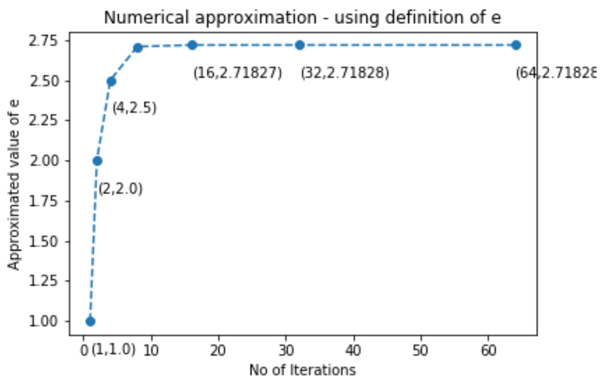


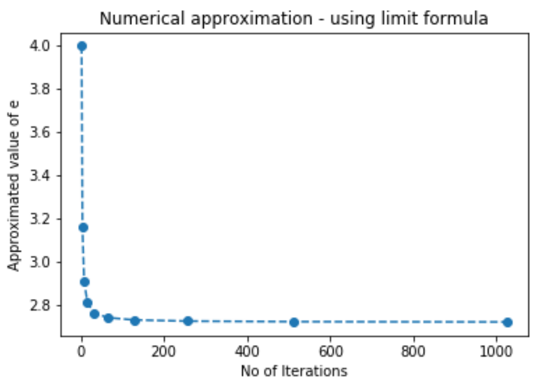


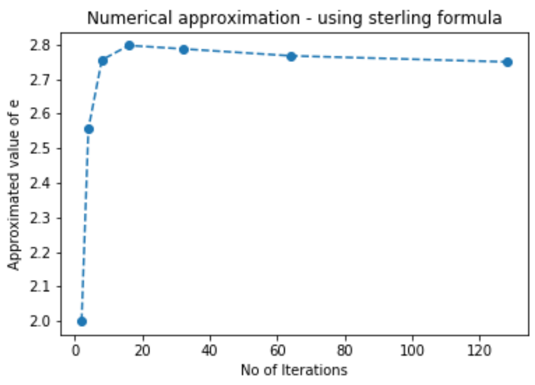
Part 1: Estimating e using Numerical and Stochastic Approximations
Numerical Approximations
The three numerical techiques were used for estimating e.

1. Using definition of e, $e = \sum_{n=1}^{\infty} \frac{1}{n!}$
2. $\frac{1}{e} = lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n$
3. Using sterling formula $e = lim_{n \to \infty} \frac{n}{\sqrt[n]{n!}}$

The curves were obtained as follows:







Stochastic Approximations
Method 1: Using exponential distribution
When we want to find estimation of some quantity then one of the basic ideas in stochastic approximation is to find a Random variable whose expectation is equal to the the quantity that we want to estimate. Then, we will use **Weak Law of Large Number** which says that after having large number of samples then sample mean will converge to the actual mean.

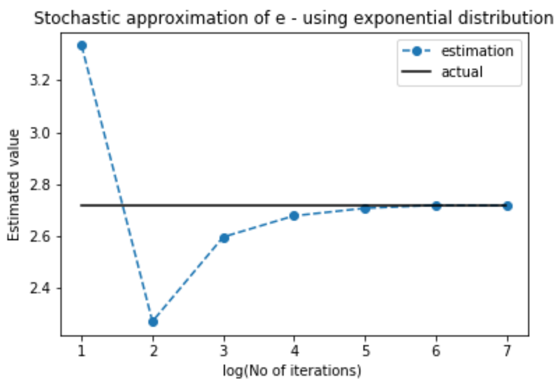
Now, we will apply same idea as mentioned above on the exponential distribution to get the estimate of e. The exponential distribution is given by:

$$f(x) = \begin{cases} e^{-x} & x > 0 \\ 0 & otherwise \end{cases}$$

Note tha $P(X > 1) = \frac{1}{e}$, if we define a new r.v. $Y = \mathcal{I}(X > 1)$ then it is easy to see that $E(Y) = \frac{1}{e}$.
Algorithm:

1. Generate $U_1, U_2, \dots U_N \sim U(0, 1)$.
2. Using probability integral transform i.e. $X_i = -log(1-U_i) \forall i = 1 \dots N$, generate $X_1, \dots X_N \sim Exp(1)$.
3. Define $Y_i = \mathcal{I}(X_i > 1), \forall i = 1 \dots N$.
4. Compute $B_n = \frac{1}{N}\sum_{i=1}^{N} Y_i$. By WLLN it will converge assymptotically in probability to $\frac{1}{e}$.
5. By central mapping $\frac{1}{B_N}$ will converge to e.

After running the algoithm for **N = [10, 100, 1000, 10000, 100000, 1000000, 10000000]** the estimated values are [**3.33333333, 2.27272727, 2.5974026 , 2.67881061, 2.70797227, 2.71958619, 2.71850678**]. We get the following plot:

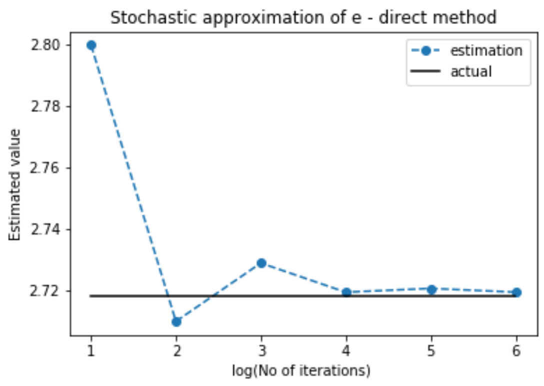


Method 2 - Direct Estimation:
Algorithm:

- Generate $U_1 \ldots U_n \overset{i.i.d}{\sim} Unif(0,1)$
- Define $S_n = U_1 + \ldots + U_n, n \geq 1$
- Define $T = inf\{n : S_n > 1\}$. Note that $support(T) = \{2, 3, 4, \ldots\}$

1. Generate $T_1, T_2, \ldots T_m$ iid random variable as described above.
2. By Weak Law of Large Numbers $\frac{1}{m}\sum_{i=1}^{m} T_i \overset{p/a.s.}{\longrightarrow} E(T_1) = e$. Note that $E(T) = e$

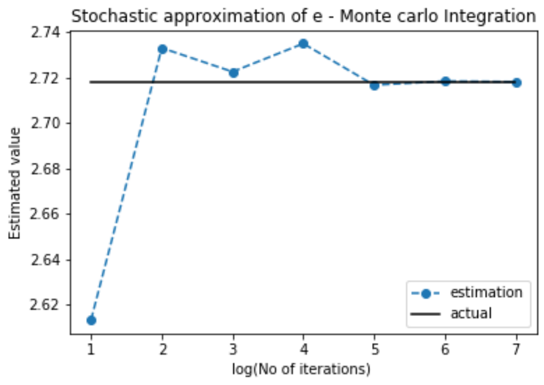
After running the algoithm for **N = [10, 100, 1000, 10000, 100000, 1000000]** the estimated values are **[2.8 , 2.71 , 2.729 , 2.7195 , 2.72074 , 2.719521]**. We get the following plot:



Method 3 - Using Monte carlo integration:
Note that $\alpha = \int_1^2 \frac{1}{x}dx = \log_e 2$. Therefore, we can estimate e as follows $e = 2^{\frac{1}{\alpha}}$. We will use monte carlo integration to achieve the same.
Algorithm:

1. Generate $X_1, \ldots X_m \overset{iid}{\sim} U(1,2)$.
2. Compute the estimator of integral $\hat{\alpha} = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{X_i}$.
3. Return $2^{\frac{1}{\hat{\alpha}}}$

After running the algoithm for **N** = [**10, 100, 1000, 10000, 100000, 1000000, 10000000**] the estimated values are [**2.61328166, 2.73311734, 2.72263031, 2.73503453, 2.71676695, 2.71847457, 2.7182204**]. We get the following plot:

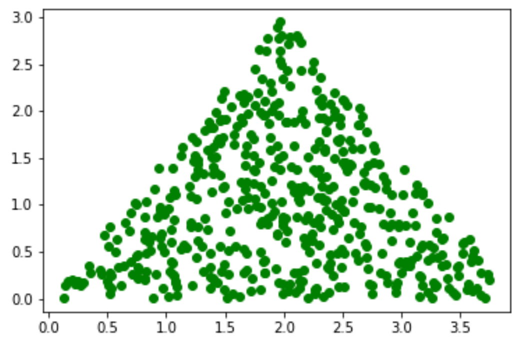


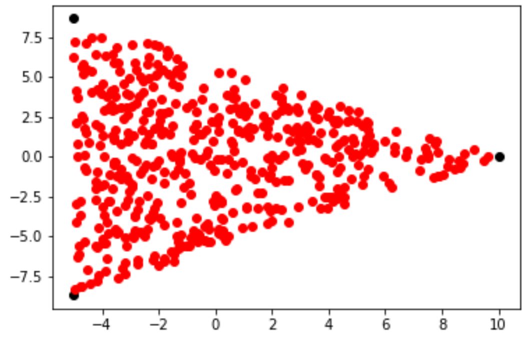
Statistical Simulation and Data Analysis (MTH511A), Autumn 2018
Indian Institute of Technology Kanpur
Programming Assignment Number 1

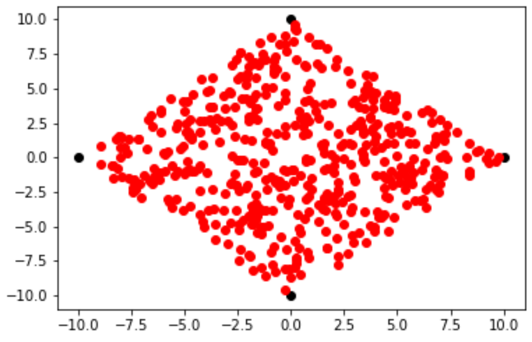
Student Name: Bhavy Khatri
Roll Number: 150186
Date: November 11, 2018

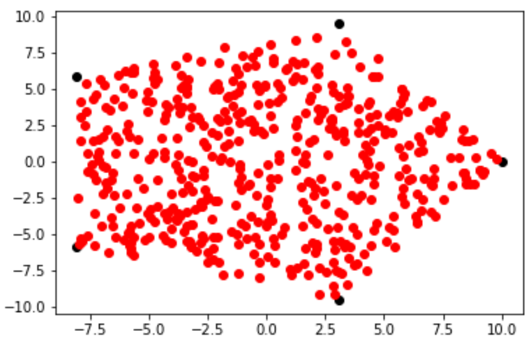
QUESTION
2

Part-1: Generating observations using direct geometric approach and alias method
The following observations were obtained:

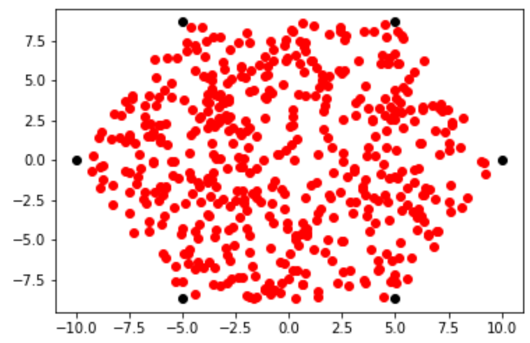




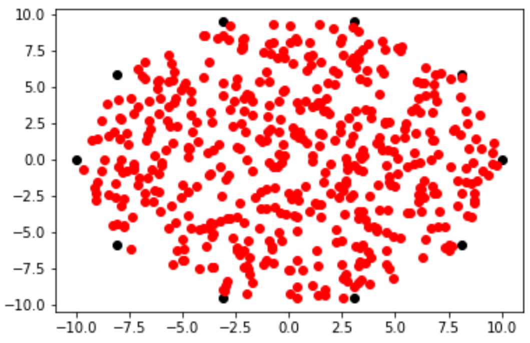




10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Statistical Simulation and Data Analysis (MTH511A), Autumn 2018
Indian Institute of Technology Kanpur
Programming Assignment Number 1

Student Name: Bhavy Khatri
Roll Number: 150186
Date: November 11, 2018

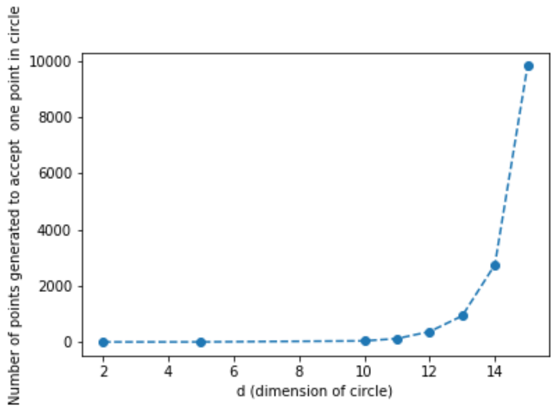
QUESTION 3

Aim: To Generate points inside the d-dimensional circle.

Part 1: Using Accept Reject

Idea: Generate a d-dimensional cube using d uniform(-1, 1) random numbers. After that check the condition whether it lies inside the circle or not. If it lies inside the circle then accept it otherwise reject it.

I simulated around 100 points for **d=[2, 5, 10, 11, 12, 13, 14, 15]** and the following graph was obtained:



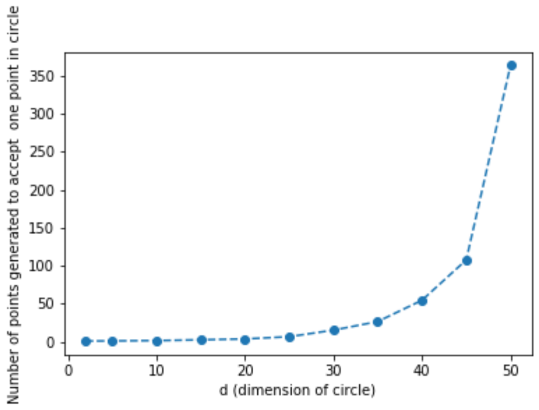
d (dimension of a circle)	No. of points to generate a single point inside a circle
2	1.4
5	7.54
10	384.1
11	1189.32
12	3662.61
13	9276.96
14	27464.29
15	98154.31

Now interestingly, to generate from $d = 25$, it was taking a lot of time. So, I restricted myself to generate upto 15th dimension circle. This is highly inefficient method for generating points from higher dimension circle **(Curse of Dimensionality)**.

Part 2 MCMC:
Algorithm:

1. Start with any point $Z_0 = (X_0, \ldots, X_d)$ inside the circle of d-dimension. Most easy choice is to choose all zeros.
2. Generate $W \sim Q = [X_1 - 0.2, X_1 + 0.2], \ldots, [X_d - 0.2, X_d + 0.2]$, where Q is the proposal distribution.

3. Compute $\alpha = min\left(1, \frac{\mathcal{I}(\|W\|<1)}{\mathcal{I}(\|X_t\|<1)}\right)$.
4. Generate $U \sim U(0,1)$.
5. If $0 < u < \alpha$ then $Z_{t+1} = W$. In this case Z_{t+1} will be new point selected randomly uniformly from the circle. When $\alpha < U < 1$ then $Z_{t+1} = Z_t$.
6. Repeat until you get the desired number of points.



d (dimension of a circle)	No. of points to generate a single point inside a circle
2	1.14
5	1.28
10	1.72
15	2.76
20	3.89
25	6.85
45	107.71
50	363.42

Clearly, MCMC beats accept reject by a high margin in higher dimension. So, the next question arises why did MCMC beat Accept-Reject?
How did MCMC overcome the curse of dimensionality?
The low efficiency for accept-reject in high dimensions is because information about past acceptances is not retained. In MCMC, we used the information about our current sample to propose the next sample. Note that the density of the uniform distribution over a p-sphere is given by:

$$f(\hat{x}) = \frac{\Gamma(d/2+1)}{\pi^{p/2}}\mathcal{I}(\|x\| < 1)$$

In MCMC, we didn't care about the coeffecient as it cancelled in the accept-stay step.

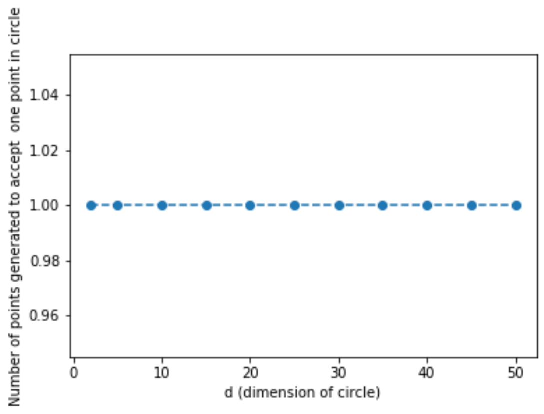
Part 3: Using Symmetricity
Consider unit circle of dimension d. We will use polar co-ordinates in this case. Let, R denote the distance of random point from the origin, then $P(R \leq r) = r^d$. Note that this is a valid CDF. We can use probability integral transform to generate from the above distribution. After fixing the r we will try to generate from the surface of radius R. For that we will use the following fact that if $X_1, \dots X_d$ are generated from independent normal distribution then,

$$\frac{(X_1, \dots, X_d)}{\sqrt{X_1^2 + \dots X_d^2}}$$

will give a point on the surface of unit sphere. For generating a point on the sphere of radius R $= U^{\frac{1}{d}}$, we will just consider,

$$U^{\frac{1}{d}}\frac{(X_1,\ldots,X_d)}{\sqrt{X_1^2+\ldots X_d^2}}$$

We get the following curves and table:



d (dimension of a circle)	No. of points to generate a single point inside a circle
2	1.0
5	1.0
10	1.0
15	1.0
20	1.0
25	1.0
45	1.0
50	1.0

Note that this is better than MCMC and accept reject as there is no rejection or staying at the same state involved. The algorithm always move forward. Every point generated will be uniformly distributed inside the circle of dimension d.

Statistical Simulation and Data Analysis (MTH511A), Autumn 2018
Indian Institute of Technology Kanpur
Programming Assignment Number 1

Student Name: Bhavy Khatri
Roll Number: 150186
Date: November 11, 2018

QUESTION 4

Aim: To minimize real valued function of two variables using deterministic and Monte Carlo algorithm.

$f(x, y) = (x\sin(20y) + y\sin(20x))^2\cosh(\sin(10x)x) + (x\cos(10y) - y\sin(10x))^2\cosh(\cos(20y)y)$

Reason that (0, 0) is the global minimum:
Note that cos hyperbolic function is always positive and square is also always positive. This implies $f(x, y) \geq 0$. Note that function takes the value 0 at $(0, 0)$ which means (0, 0) is a global minimum.

Deterministic Algorithm: Gradient Descent
Algorithm:

- Choose learning rate neta = 0.03.

1. Choose (x_0, y_0) = (0.5, 0.5).
2. Update $x^k = x^{k-1} - neta \times \frac{\partial f}{\partial x}|_{x_{k-1},y_{k-1}}$.
3. Similarly update $y^k = y^{k-1} - neta \times \frac{\partial f}{\partial y}|_{x_k,y_{k-1}}$.
4. Repeat until convergence.

After Running the algorithm for 1000 iterations and with the initial value (0.5, 0.5), it converges to the local minima (-0.1790 0.4235). Note that this is not the global minima. We will see in case of simulated annealing with the same initial condition function will converge to global minima.

Note:

- To implement the gradient descent algorithm, I had to compute the gradient with respect to each variable. At this point I had two options:
  - Use finite difference (Numerical Method) i.e. $f'(x) = \frac{f(x+h)-f(x)}{h}$.
  - Use symbolic differentiation.

  In case of numerical differentiation, I was getting the overflow error because of the presence of cos hyperbolic function because when it start learning and the point become more than 2 then it just exponentially burst the function value which become harder to handle. That's why symbolic differentiation is preferred.

Stochastic Algorithm: Simulated Annealing
We want to minimize $f(x, y)$ which is equivalent to maximizing $\frac{1}{f(x,y)}$. Let's call it $h(x, y)$.

$$\begin{aligned}(x^*, y^*) &= argmin_{(x,y)} f(x, y)\\ &= argmax_{(x,y)} h(x, y) = \frac{1}{f(x, y)}\\ &= exp\left(\frac{h(x, y)}{T}\right), T > 0\end{aligned}$$

Algorithm:

- Consider proposal probability distribution to be bivariate normal with co-variance matrix equal to $\mathbf{I}_{2\times 2}$ identity matrix . We can choose any probability distribution with infinite support. Mean which represents the location of the distribution will be decided later in the iteration.
- Fix T=1.

1. Start with initial value (x_0, y_0).
2. Generate $\mathcal{G}' \sim \mathcal{N}(0, \mathbf{I})$.
3. Compute $\mathcal{G} = (x_{k-1}, y_{k-1})^T + \mathcal{G}'$, using location scale transform.
4. Simulate $U \sim U(0, 1)$.
5. If $0 < U < min\left(1, exp\left(\frac{h(\mathcal{G}) - h(x_{k-1}, y_{k-1})}{T}\right)\right)$, then update $(x_k, y_k) = \mathcal{G}$. Otherwise don't do anything.

After implementing the above algorithm and taking the initial value as **(0.5, 0.5)**, we get **(0.00040, 0.0102)** after 2000 iterations. It reached very near to the global minimum.

Statistical Simulation and Data Analysis (MTH511A), Autumn 2018
Indian Institute of Technology Kanpur
Programming Assignment Number 1

Student Name: Bhavy Khatri
Roll Number: 150186
Date: November 11, 2018

QUESTION 5

Fischer Scoring:

EM Algo:
Note that n_a, n_b, n_{ab}, n_o are the observed data. Frequency of alleles A, B and O is given by p_a, p_b, p_o .But there is also unobserved data, that's why the complete data is given by - $(n_{aa}, n_{ao}, n_{bb}, n_{bo}, n_{ab}, n_o)$. Their corresponding allele frequency is given by $p_a^2, 2p_ap_o, 2p_ap_b, p_b^2, 2p_bp_o, p_o^2$. Also $n = n_{aa} + n_{ao} + n_{bb} + n_{bo} + n_{ab} + n_o$ .Then log likelihood function is given by:

$$\begin{aligned} l_n(p) &= n_{aa}\log(p_a^2) + n_{bb}\log(p_b^2) + n_o\log(p_o^2) + n_{ab}\log(2p_ap_b) \\ &+ n_{ab}\log(2p_ap_b) + n_{ab}\log(2p_ap_b) + \log\left(\frac{n!}{n_{aa}!n_{ao}!n_{bb}!n_{bo}!n_{ab}!n_o!}\right) \end{aligned}$$

In **expectation step**, we estimate the expectation of latent quantities. Note that, $n_{aa} + n_{ao} = n_a$, therefore,

$$n_{aa}|n_a \sim Bin\left(n_a, \frac{p_a^2}{p_a^2 + 2p_ap_o}\right)$$

Therefore the expectation is given by:

$$n_{aa}^{(k)} = \mathbb{E}(n_{aa}|n_{obs}) = n_a\frac{p_a^{(k)^2}}{p_a^{(k)^2} + 2p_a^{(k)}p_o^{(k)}}$$

Similarly,

$$\begin{aligned} n_{ao}^{(k)} &= \mathbb{E}(n_{ao}|n_{obs}) = n_a\frac{2p_a^{(k)}p_o^{(k)}}{p_a^{(k)^2} + 2p_a^{(k)}p_o^{(k)}} \\ n_{bb}^{(k)} &= \mathbb{E}(n_{bb}|n_{obs}) = n_b\frac{2p_b^{(k)}p_o^{(k)}}{p_b^{(k)^2} + 2p_b^{(k)}p_o^{(k)}} \\ n_{bo}^{(k)} &= \mathbb{E}(n_{bo}|n_{obs}) = n_b\frac{2p_b^{(k)}p_o^{(k)}}{p_b^{(k)^2} + 2p_b^{(k)}p_o^{(k)}} \end{aligned}$$

while n_ab, n_o will remain the same that is

$$\begin{aligned} \mathbb{E}(n_{ab}|n_{obs}) &= n_{ab} \\ \mathbb{E}(n_o|n_{obs}) &= n_o \end{aligned}$$

In the maximization step, we will maximize the likelihood function. Since, now we know the estimates of latent variable, therefore estimation of allele frequency will be easier. Also, since $p_a + p_b + p_o = 1$, it will be constrained optimization problem. Solving using lagrange multiplier we get,

$$\begin{aligned}
p_a^{(k+1)} &= \frac{2n_{aa}^{(k)} + n_{ao}^{(k)} + n_{ab}^{(k)}}{2n} \\
p_b^{(k+1)} &= \frac{2n_{bb}^{(k)} + n_{bo}^{(k)} + n_{ab}^{(k)}}{2n} \\
p_o^{(k+1)} &= \frac{n_{ao}^{(k)} + n_{bo}^{(k)} + 2n_{o}^{(k)}}{2n}
\end{aligned}$$

Now, taking $n_a, n_b, n_{ab} and n_o =$ **25, 50, 100, 150** and different value of p we get the following results.

Iteration	p_a	p_b	p_c
0	0.33333	0.33333	0.33333
1	0.20512	0.25641	0.30769
2	0.20192	0.25339	0.31391
3	0.20166	0.25288	0.31467
4	0.20164	0.25281	0.31477
5	0.20163	0.25280	0.31478
6	0.20163	0.25280	0.31478
7	0.20163	0.25280	0.31478
8	0.20163	0.25280	0.31478
9	0.20163	0.25280	0.31478
10	0.20163	0.25280	0.31478

Iteration	p_a	p_b	p_c
0	0.2	0.3	0.5
1	0.19871	0.24852	0.32199
2	0.20137	0.25218	0.31566
3	0.20160	0.25272	0.31489
4	0.20163	0.25279	0.31479
5	0.20163	0.25280	0.31478
6	0.20163	0.25280	0.31478
7	0.20163	0.25280	0.31478
8	0.20163	0.25280	0.31478
9	0.20163	0.25280	0.31478
10	0.20163	0.25280	0.31478

Iteration	p_a	p_b	p_c
0	0.8	0.1	0.1
1	0.22307	0.25641	0.28974
2	0.20299	0.25436	0.31186
3	0.20175	0.25305	0.31442
4	0.20164	0.25284	0.31473
5	0.20163	0.25281	0.31477
6	0.20163	0.25280	0.31478
7	0.20163	0.25280	0.31478
8	0.20163	0.25280	0.31478
9	0.20163	0.25280	0.31478
10	0.20163	0.25280	0.31478

Iteration	p_a	p_b	p_c
0	0.25	0.25	0.5
1	0.2	0.24615	0.32307
2	0.20139	0.25198	0.31584
3	0.20160	0.25270	0.31492
4	0.20163	0.25279	0.31480
5	0.20163	0.25280	0.31478
6	0.20163	0.25280	0.31478
7	0.20163	0.25280	0.31478
8	0.20163	0.25280	0.31478
9	0.20163	0.25280	0.31478
10	0.20163	0.25280	0.31478

Note that how taking different initial values of p doesn't have any impact on the final estimated allele frequency.

Statistical Simulation and Data Analysis (MTH511A), Autumn 2018
Indian Institute of Technology Kanpur
Programming Assignment Number 1

Student Name: Bhavy Khatri
Roll Number: 150186
Date: November 11, 2018

QUESTION 6

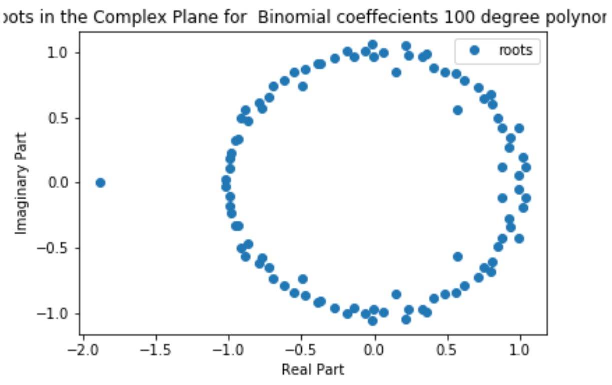
Observations:

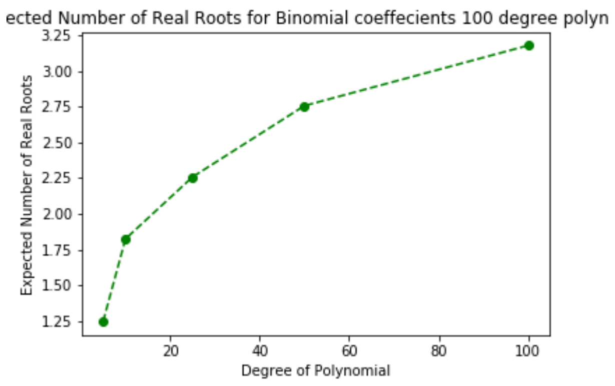
- For each of the distribution it was seen that for large n , **expected number of real roots** was approaching to absolute value **4**.
- In case of large n, it was also seen that most of the roots were complex.
- When graph of roots was plotted in the complex plane a wonderful result was obtained. For each of the the distribution, **roots were symmetrically aligned in the circle of radius one.** It also means they were approximately becoming equal to the nth root of unity i.e. $e^{i\frac{2\pi}{n}}$.

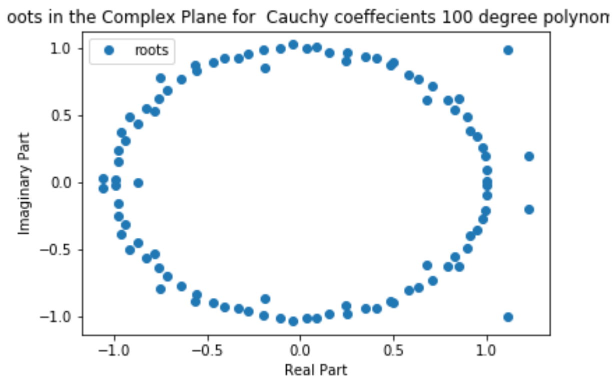
Note:

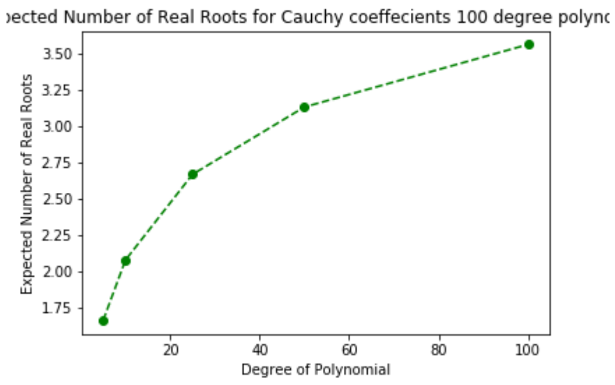
- Points from the Binomial, Normal, Cauchy and Exponential were generated using alias method, Box muller transform, Probability Integral Transform and Probability Integral Transform again respectively.
- For calculating $\mathcal{E}(A_n)$, WLLN was used that is random variable A_in was drawn about 5000 times for each n and then sample mean was approximated as true mean.
- Roots of the polynomial were calculated using the **np.roots** module of the python numpy library. Although, I am aware of **Muller Algorithm** for finding all roots of the polynomial but due to time constraint I was not able to do so.

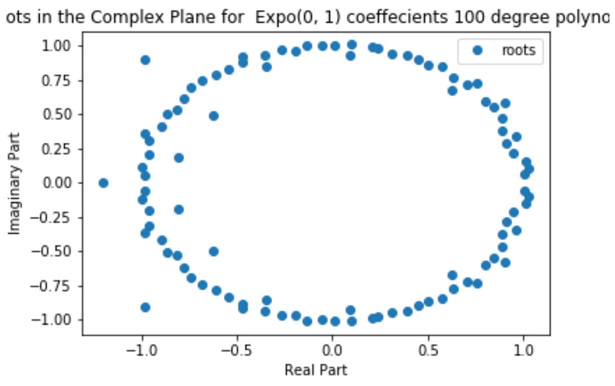
The following plots were obtained:

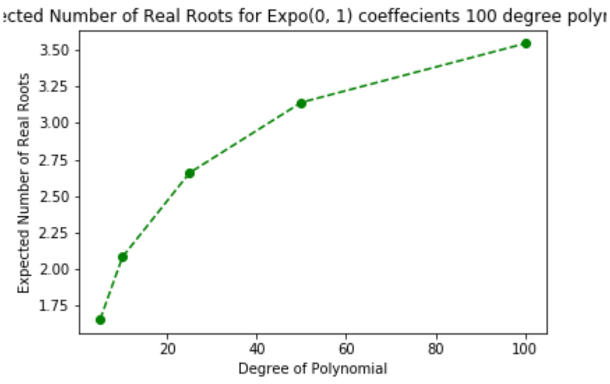



ected Number of Real Roots for Binomial coeffecients 100 degree polyn
Expected Number of Real Roots
Degree of Polynomial
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
20
40
60
80
100

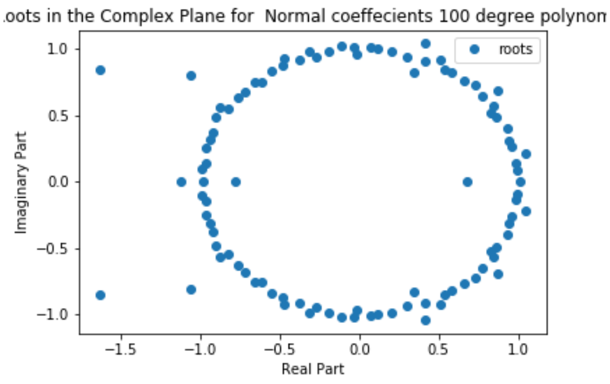

oots in the Complex Plane for Cauchy coeffecients 100 degree polynon
roots
Imaginary Part
Real Part
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


›ected Number of Real Roots for Cauchy coeffecients 100 degree polync
Expected Number of Real Roots
Degree of Polynomial
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
20
40
60
80
100

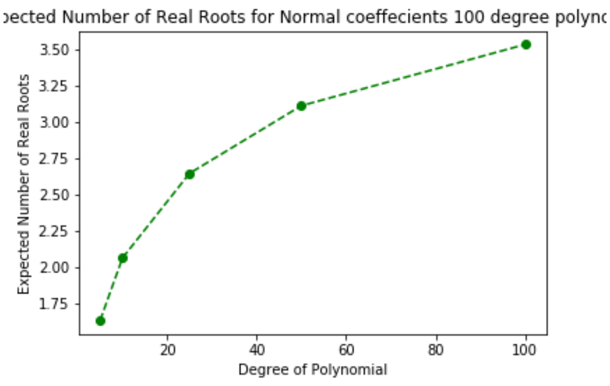

ots in the Complex Plane for Expo(0, 1) coeffecients 100 degree polync
roots
Imaginary Part
Real Part
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0



ected Number of Real Roots for Expo(0, 1) coeffecients 100 degree poly
Expected Number of Real Roots
Degree of Polynomial
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
20
40
60
80
100



oots in the Complex Plane for Normal coeffecients 100 degree polynor
roots
Imaginary Part
Real Part
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0



pected Number of Real Roots for Normal coeffecients 100 degree polync
Expected Number of Real Roots
Degree of Polynomial
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
20
40
60
80
100