Assignment-10 (Explorations Into Chaos and Number Theory)

By Pranav S Ramanujam
EP23B038
Department of Physics.

1) Chaos (The Lorentz Attractor)

Theory:

The Lorenz attractor is a mathematical model that describes a chaotic, deterministic system exhibiting what is known as the “butterfly effect” (States that the behavior of chaotic systems is highly sensitive to initial conditions, meaning small changes in the starting values like the changes in the flapping of its wings by a butterfly may be responsible for a something like a tornado in the far future).
In the Lorentz Attractor, we deal with the behaviour of Convection currents in a fluid, whose 3 parameters, x[t] (the rate of convective motion), y[t] (the temperature difference between the ascending and descending currents) and z[t] (the distortion (from linearity) of the vertical temperature profile) are related via a set of differential equations:
x’[t]=σ(y[t]-x[t])
y’[t]=x[t]*(ρ-z[t])-y[t]
z’[t]=x[t]*y[t]-β*z[t]
The plots of x[t], y[t] and z[t] greatly vary with even the slightest of changes made in the initial conditions.

Code:

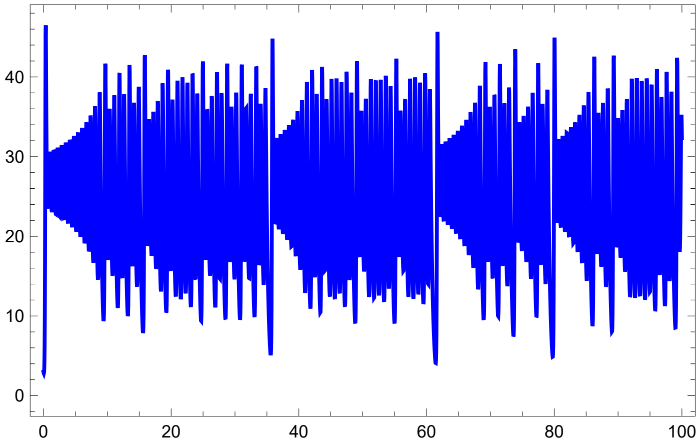
```
In[∘]:= eq1=x'[t]==σ*(y[t] - x[t]);
        eq2=y'[t]==x[t]*(ρ-z[t])-y[t];
        eq3=z'[t]==x[t]*y[t]-β*z[t];
        σ=10;
        ρ=28;
        β=8/3;
        sol=NDSolve[{eq1,eq2,eq3,x[0]==1,y[0]==2,z[0]==3},{x,y,z},{t,0,100}]
```

```
Out[∘]=
{{x → InterpolatingFunction[ Domain: {{0., 100.}}
                             Output: scalar ],

  y → InterpolatingFunction[ Domain: {{0., 100.}}
                             Output: scalar ],

  z → InterpolatingFunction[ Domain: {{0., 100.}}
                             Output: scalar ]}}
```

```
In[∘]:= xFunction[t_]=x[t]/. sol[[1]];
        yFunction[t_]=y[t]/. sol[[1]];
        zFunction[t_]=z[t]/. sol[[1]];
```

```
In[∘]:= plot1=Plot[Evaluate[{z[t]}/.sol],{t,0,100},PlotStyle→Blue,Frame→True] (*Plotting z[t] as a f
```

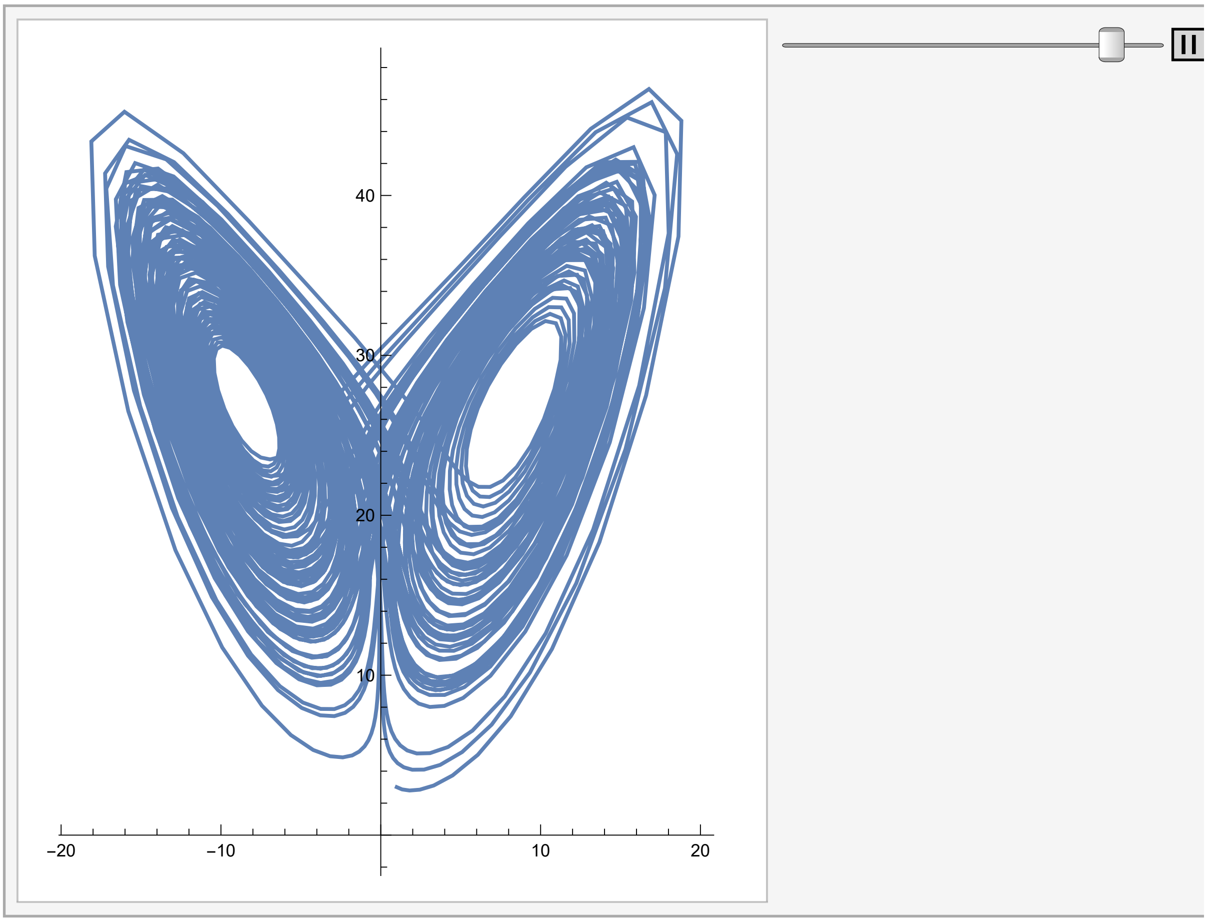
Out[∘]=



In[•]:=

```
animation1=Table[ParametricPlot[{xFunction[t],zFunction[t]},{t,0,tmax}],{tmax,1/10,100,1}];
ListAnimate[animation1] (*Making an animation of the parametric plot of z[t] vs x[t]*)
```

Out[•]=



In[•]:=

```
sol2=NDSolve[{eq1,eq2,eq3,x[0]==100001/10^5,y[0]==200001/10^5,z[0]==300001/10^5},{x,y,z},{t,0,
conditions very slightly*)
```

Out[•]=

```
{{x → InterpolatingFunction[ Domain: {{0., 100.}}
                             Output: scalar ],

  y → InterpolatingFunction[ Domain: {{0., 100.}}
                             Output: scalar ],

  z → InterpolatingFunction[ Domain: {{0., 100.}}
                             Output: scalar ]}}
```

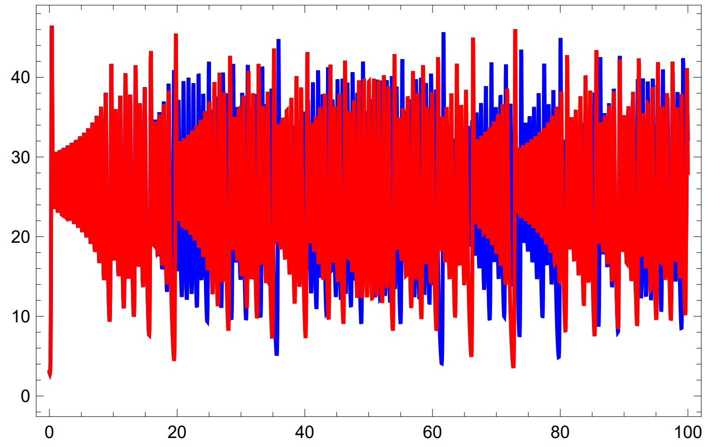
In[•]:=

```
zFunction2[t_]=z[t]/. sol2[[1]];   (*storing z[t] after the change of initial conditions as zFu
xFunction2[t_]=x[t]/. sol2[[1]];
plot2=Plot[zFunction2[t],{t,0,100},PlotStyle→Red];
```

In[∘]:=

```
Show[plot1,plot2] (*A plot of z[t] before (blue) and after (red) the change of initial condit
```

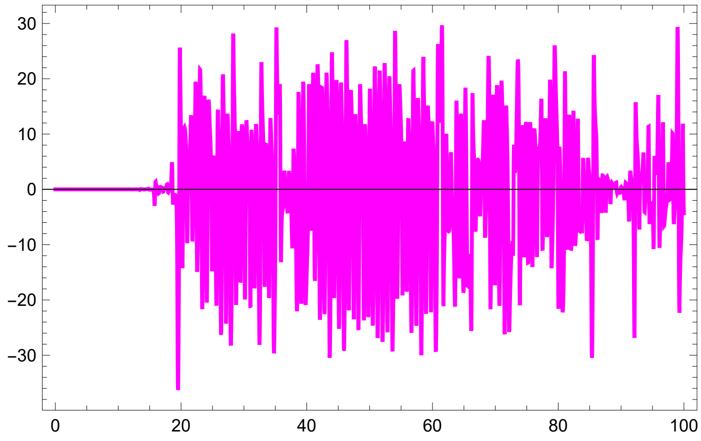
Out[∘]=



In[∘]:=

```
Plot[zFunction2[t]-zFunction[t],{t,0,100},PlotStyle→Magenta, Frame→True]
```

Out[∘]=



2) Approximating Feigenbaum’s Constant Using the Logistic Map:

Theory:

The Logistic Map is a Recursive relation which predicts the population of a species in the next year, when given the current population of that species as input.
The Logistic Map is given by the relation:
X_{N+1}=r * X_N * (1 - X_N)
where X_{N+1} is the ratio of the next year’s population and the maximum possible population of the species, and X_N is the ratio of the current population with respect to the same. Hence X_N and X_{N+1} always lie between 0 and 1. ‘r’ is the rate of breeding. r=0 implies no breeding at all. The value of ‘r’ can be between 0 and 4 only. ‘r’ cannot exceed 4 as otherwise, for certain initial values of X_N, the value of X_{N+1} may exceed 1.

It was observed that for 0<=r<1, after many years, the population tends to zero.
for 1<=r<3, after many years, the population tends to a constant value.
for 3<=r<3.45, after many years, the population oscillates back and forth between two values.
for 3.45<=r<3.54, after many years, the population oscillates between four values.
for 3.54<=3.r<3.68, after many years, the population oscillates between eight values and so on.
From here onwards, till we reach r=4, as we continue to increase the value of ‘r’, bifurcations keep occurring and it finally ends up in chaos, in which, the population after many years don’t really follow any pattern and vary randomly (This is used today for random number generation). As we still keep increasing ‘r’, there is a small region where the population after many years oscillates between 3 ,6 ,9 ...... and so on values, but this also ends up in chaos ultimately.
The ratio of the difference between the values of ‘r’ in successive bifurcations was found to be constant and is known as the Feigenbaum’s Constant ≃ 4.669.

Code:

```
In[•]:= Clear["Gloabal*`"]
```

```
In[•]:= population[n_]:=Module[{f},RecurrenceTable[{f[i+1]==r*(f[i])*(1-f[i]),{f[0]==0.4}},f,{i,0,n-
```

```
In[•]:= popultn={};
        rvals={};
        For[r=1,r<4,r+=1/500,If[FindTransientRepeat[population[10000],5]⟦2⟧=={},Continue[]];
        AppendTo[rvals,r];AppendTo[popultn,FindTransientRepeat[population[10000],10]⟦2⟧]]

         (*Finding the population after many years for values of 'r' ranging from  1 to 4, but only for
        periodicity is shown after many years (As "FindTransientRepeat[population[10000],5][[2]]" ret
        repeating after many years, and I have imposed the condition "If[FindTransientRepeat[populatio
        which means the chaos part will be skipped*)
```

```
In[•]:= rvals2={};
        popultn2={};
        For[r=36/10,r<4,r+=1/500,AppendTo[rvals2,r];AppendTo[popultn2,Take[population[10000],-10]]]
        the chaotic values of population after many years for values of 'r' from 3.6 to 4*)
```

```
rvals3={};
popultn3={};
For[r=0,r<1,r+=1/500,AppendTo[rvals3,r];AppendTo[popultn3,0]]
 (*Creating a separate list to store values of population
after long time for r less than 1*)
```

In[∘]:=

```
result={};
result2={};
result3={};
For[j=1,j≤Length[rvals],j++,For[k=1,k≤Length[popultn⟦j⟧],k++,AppendTo[result,{rvals⟦j⟧,popul
For[l=1,l≤Length[rvals2],l++,For[m=1,m≤Length[popultn2⟦l⟧],m++,AppendTo[result2,{rvals2⟦l⟧,p
result3=Transpose[{rvals3,popultn3}];
FinalResult=Join[result,result2,result3] (*The final set of points of the populations after ve
from 0 to 4*)
```

Out[∘]=

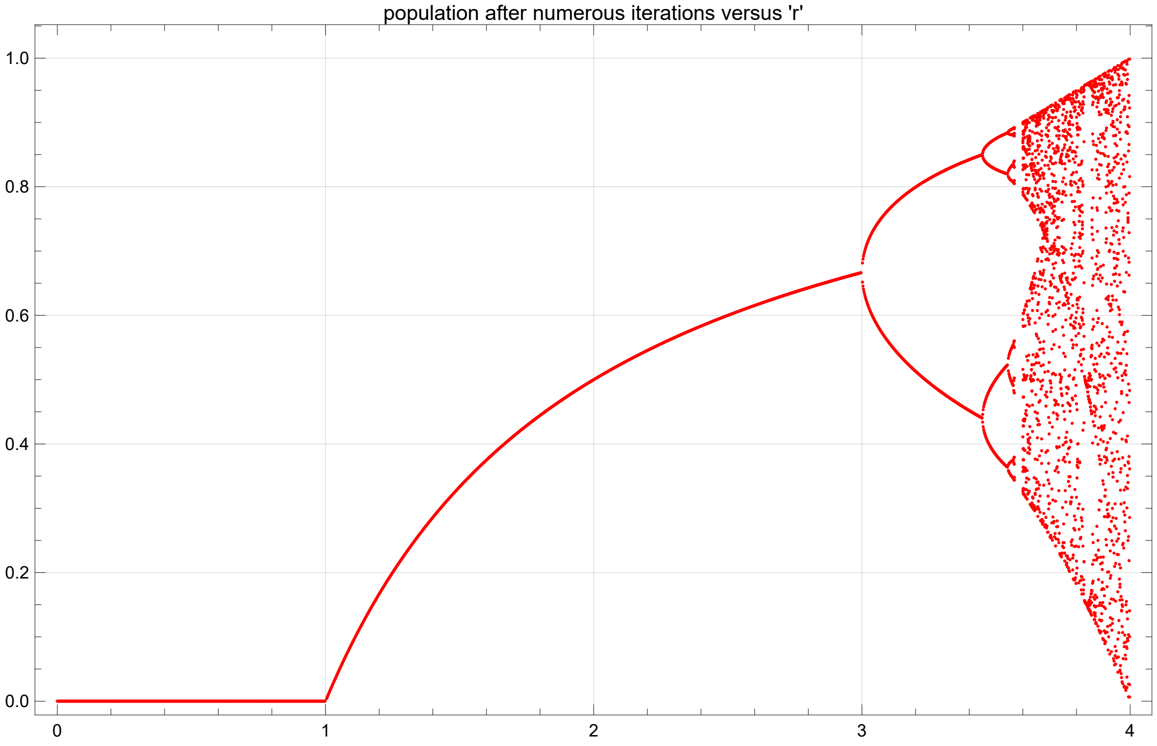
```
{{251/250, 0.00398406}, {503/500, 0.00596421}, {126/125, 0.00793651}, {101/100, 0.00990099}, {253/250, 0.0118577},
 {507/500, 0.0138067}, {127/125, 0.015748}, {509/500, 0.0176817}, {51/50, 0.0196078}, {511/500, 0.0215264},
 {128/125, 0.0234375}, {513/500, 0.0253411}, {257/250, 0.0272374}, {103/100, 0.0291262}, {129/125, 0.0310078},
 {517/500, 0.032882}, … 5226 …, {121/125, 0}, {97/100, 0}, {243/250, 0}, {487/500, 0}, {122/125, 0}, {489/500, 0},
 {49/50, 0}, {491/500, 0}, {123/125, 0}, {493/500, 0}, {247/250, 0}, {99/100, 0}, {124/125, 0}, {497/500, 0}, {249/250, 0}, {499/500, 0}}
```

Size in memory: 0.7 MB | + Show more | Show all | Iconize ▾ | Store full expression in notebook

In[∘]:=

```
ListPlot[FinalResult,PlotStyle→Red,Frame→True,GridLines→Automatic,PlotLabel→"population after
```

Out[∘]=



In[ ]:=

```
(*To approximate the Feigenbaum Constant*)
index1;
index2;(*To find the index of 'r' in rvals at which each of the bifurcations occur*)
index3;
For[i=100,i≤Length[popultn],i++,If[(Length[popultn⟦i⟧]==2) && (Round[ popultn⟦i⟧⟦1⟧,0.001]≠Ro
index1=i-1;Break[],Continue[]]]
(*(Length[popultn⟦i⟧]==2) as thats where the bifurcation from one to two branches begins,
(Round[ popultn⟦i⟧⟦1⟧,0.001]≠Round[popultn⟦i⟧⟦2⟧,0.001]), I have done the rounding off as, du
at places where I am supposed to get a single value for the population after long time, I get
at the eight decimal or so, which is irrelevent to us
Likewise for the following two loops*)
For[i=100,i≤Length[popultn],i++,If[Length[popultn⟦i⟧]==4 &&  (Round[popultn⟦i⟧⟦1⟧,0.001]≠Roun
index2=i-1;Break[],Continue[]]]
For[i=100,i≤Length[popultn],i++,If[Length[popultn⟦i⟧]==8 &&  (Round[popultn⟦i⟧⟦1⟧,0.001]≠Roun
index3=i-1;Break[],Continue[]]]
```

```
N[rvals⟦index2⟧]
N[rvals⟦index1⟧]   (*The values of 'r' where the bifurcations occur*)
N[rvals⟦index3⟧]
FeigenConst=N[(rvals⟦index2⟧-rvals⟦index1⟧)/(rvals⟦index3⟧-rvals⟦index2⟧)] (*By the definiti
```

Out[ ]=

```
3.448
```

Out[ ]=

```
2.996
```

Out[ ]=

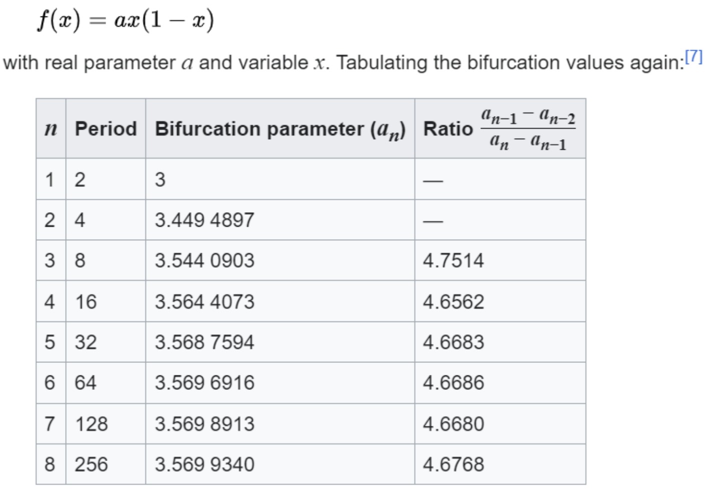
```
3.542
```

Out[ ]=

```
4.80851
```

Actually, the Feigenbaum's constant is the limiting value of this ratio as we do it again and again (The ratio converges to 4.669....). I have found out the ratio only for the the transition from period=2 to period = 4, period=4 to period=8, which should ideally be 4.7514.

I am attaching a snippet from Wikipedia below:



$$f(x) = ax(1-x)$$

with real parameter a and variable x. Tabulating the bifurcation values again:[7]

n	Period	Bifurcation parameter (a_n)	Ratio $\frac{a_{n-1}-a_{n-2}}{a_n-a_{n-1}}$
1	2	3	—
2	4	3.449 4897	—
3	8	3.544 0903	4.7514
4	16	3.564 4073	4.6562
5	32	3.568 7594	4.6683
6	64	3.569 6916	4.6686
7	128	3.569 8913	4.6680
8	256	3.569 9340	4.6768

Here a is the bifurcation parameter, x is the variable. The values of a for which the period doubles (e.g. the largest value for a with no period-2 orbit, or the largest a with no period-4 orbit), are a1, a2 etc.

In[∘]:=

3) Pi Hiding in Prime Regularities:

Theory:

In this section, I will be trying to approximate the value of π by a method which involves counting the number of lattice points inside a circle centered at origin. I will be incorporating a new method to easily compute the number of lattice points inside the circle.

Code:

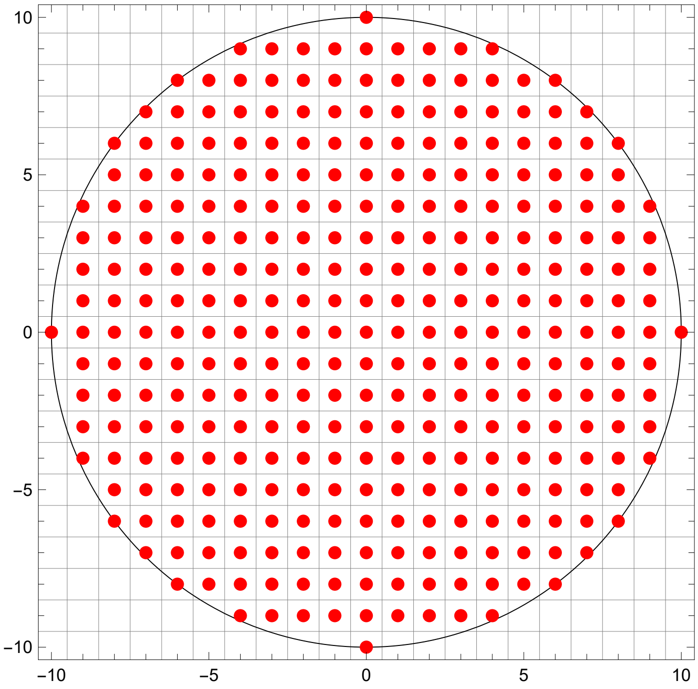
In[1]:=

```
Clear["Global`*"]
```

In[2]:=

```
(*Consider the following circle centered at origin*)
r=10;
latticePoints = Select[Tuples[Range[-Ceiling[r], Ceiling[r]], 2], Norm[#] ≤ r &];
plot1=Graphics[{Circle[{0, 0}, r],Red, PointSize[0.02], Point[latticePoints]},Frame→True,Grid
{i, -19/2, 19/2, 1}],Table[i,
{i, -19/2, 19/2, 1}]}]
```

Out[4]=



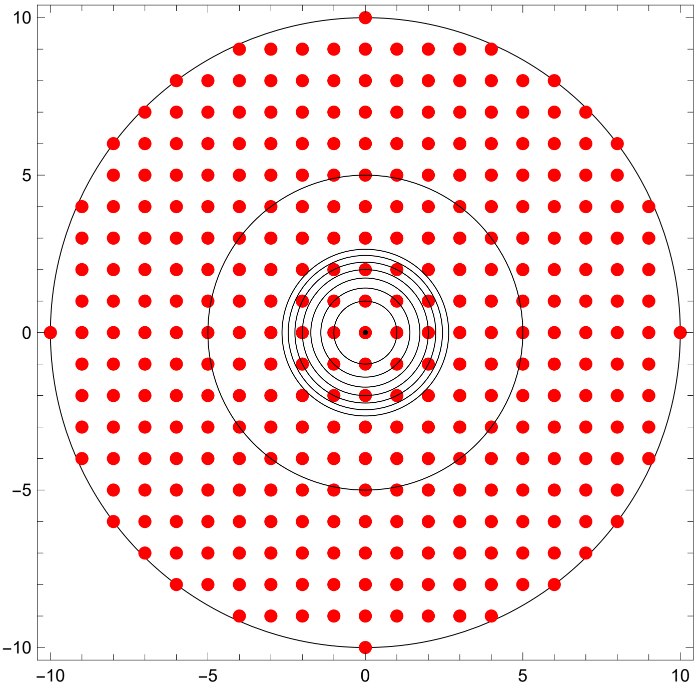
Its Clearly seen that more or less, each of the lattice points corresponds to a unit of area of the circle.

Hence we can say:
No of Lattice Points ≃ Area of the Circle (which is = πR^2)
And the error is minimised as the value of ‘R’ is increased.

In[5]:=

```
plot2=Graphics[{Circle[{0, 0}, r],Red, PointSize[0.02], Point[latticePoints]},Frame→True];
plot3=Graphics[{Circle[{0,0},Sqrt[1]]}];
plot4=Graphics[{Circle[{0,0},Sqrt[2]]}];
plot5=Graphics[{Circle[{0,0},Sqrt[3]]}];
plot6=Graphics[{Circle[{0,0},Sqrt[4]]}];
plot7=Graphics[{Circle[{0,0},Sqrt[0]]}];
plot8=Graphics[{Circle[{0,0},Sqrt[5]]}];
plot9=Graphics[{Circle[{0,0},Sqrt[6]]}];
plot10=Graphics[{Circle[{0,0},Sqrt[7]]}];
plot11=Graphics[{Circle[{0,0},Sqrt[25]]}];
Show[plot2,plot3,plot4,plot5,plot6,plot7,plot8,plot9,plot10,plot11]
```

Out[15]=



As seen above, all of the lattice points lie on some or the other circle of radius N^1/2 (where N goes from 0 to R^2), but some circles are such that they don’t have any lattice points lying on them. Therefore, the number of lattice points = Sum of lattice points on all such circles.

Now to find the number of lattice points lying on a circle of radius N^1/2, we essentially just need to find the number of lattice points whose coordinates are (a,b) such that a^2 + b^2 = N.
Hence we can think that finding all pairs (a,b) such that a^2 + b^2=N is equivalent to finding all complex numbers z=a+bi such that
zẑ=N (We are essentially just finding the number of Gaussian Integers which are at a distance of N^1/2 from the origin)
To achieve this, we do the following:

In[ ]:=

Factorise N into its Prime Gaussian Factors:
Lets say N=25 (We are finding the number of lattice points on the circle of radius (25)^1/2

```
In[16]:= n=25;
         FactorInteger[n, GaussianIntegers → True]

Out[17]=
{{-1, 1}, {1 + 2 i, 2}, {2 + i, 2}}
```

Here we can ignore the ‘-1’ as we can consider it as I^2 and multiply this inside (1+2i)^2 to get (2-i)^2, hence we have
25=z1^2 * z2^2 where z1 (2-i) and z2 (2+i) are complex conjugates of each other.
Now if we divide these into 2 columns and multiply the elements in each column, we get:

z1 z2
z1 z2
3 - 4i 3 + 4i

Which are indeed complex conjugates of each other, hence 3-4i qualifies as one of the gaussian integers we were looking for.
By performing all the permutations of this, ie:

z2 z1
z1 z2

and

z2 z1
z2 z1

we obtain ‘5’ and 3+4i also as part of the set of gaussian integers we were looking for.
To obtain the final number of all such Gaussian integers, we need to multiply the number of elements we already have by 4, as we’ll obtain more Gaussian integers satisfying the criteria by multiplying the gaussian numbers in the product by ‘i’ and ‘-i’ or ‘-1’ and ‘-1’ or
‘-i’ and ‘i’.
Eg: we currently have : (3+4i), (3-4i), 5
(3+4i)*(3-4i)=25 --> (3i-4)*(-3i-4)=25 Hence we can add 3i-4 to our set as z OverHat[z]=25 where z=3i-4.
similarly we get -5, -3i-4, -4i-3, 5i, -5i, 3i+4, 4-3i, -3+4i, -4+3i.
total=12
which is the same we get by 3*4=12. (no of permutations*4)
Hence by this, we get that 12 Gaussian integers lie on the circle with radius=25^1/2.
But when we introduce a prime factor of the form of 4n+3, eg:3

```
In[18]:= n=75;
         FactorInteger[n, GaussianIntegers → True] (*Once again we can ignore the '-1'*)

Out[19]=
{{-1, 1}, {1 + 2 i, 2}, {2 + i, 2}, {3, 1}}
```

here we cannot divide the prime Gaussian factors into 2 columns and multiply them to get a complex number ‘z’ such that z z(bar)=75
as 3 cannot be factorised further.
Hence there is no Gaussian Integer lying on the circle with radius 75^1/2.
However if 3 appears more than once:

In[20]:=

```
n=45;
FactorInteger[n, GaussianIntegers → True] (*We can ignore -i also the same way we ignored -1
```

Out[21]=

```
{{-ⅈ, 1}, {1 + 2 ⅈ, 1}, {2 + ⅈ, 1}, {3, 2}}
```

Now both the columns will have 3 in them, so we can multiply them to find ‘z’ such that z z[bar]=45 a total of 3*4=12 points will lie on this

Hence summing up everything till now:
If we want to find the number of Lattice points on a circle of radius N^1/2 (provided that 2 isn’t a prime factor of N), we need to first factorise ‘N’ into its prime natural number factors (they will be odd only as we are considering that N is odd).
1) If the total power of a factor of the form 4n+1 is ‘m’, then it will give m+1 permutations when we arrange them in the form of a column and multiply
eg: 5^3 = (2+i)^3*(2-i)^3:
2+i   2-i   2-i 2+i
2+i   2-i   2+i 2-i
2+i   2-i   2+i 2-i   and so on will give 4 permutations and each of them will give a z such that z z[bar]=N
2) If the factor is of the form of 4n+3 (eg:3), then it will give 1 permutation if its power is even and zero permutations if its power is odd (as we need to split its powers equally between the two columns)
Then the final number of permutations is equal to the number of permutations contributed by each of the prime factors, and the number of Lattice points on the circle is obtained by multiplying 4 to this.
Eg: N=3^4 * 5^3 * 13^2
number of permutations= 1* (3+1) * (2+1)=12---> number of lattice points=4*12=48.

Now finally we have the case where ‘2’ can be a factor

In[22]:=

```
n=10;
FactorInteger[n, GaussianIntegers → True] (*We can again ignore the '-1'*)
```

Out[23]=

```
{{-1, 1}, {1 + ⅈ, 2}, {1 + 2 ⅈ, 1}, {2 + ⅈ, 1}}
```

This is basically 10=(1+i)*(1-i)*(2+i)*(2-i)
now our columns will be
1+i   1-i
2+i   2-i
and one may think that since we can swap 1+i and 1-i, the presence of ‘2’ as a factor will lead to the doubling of the number of permutations, thereby doubling the number of lattice points lying on that circle. But that is incorrect, as swapping (1+i) and (1-i) and then finding the product has the same effect as multiplying the number obtained in the product by i or -i, and this is already taken into consideration when we used to multiply the number of permutations by 4. Hence the factor of

‘2’ or any power of ‘2’ has no effect on the number of lattice points on the circle.
Eg: circles of rad=5^1/2, 10^1/2, 20^1/2, 40^1/2, 80^1/2...................... all have 8 lattice points lying on them.

So now we can build upon our precious conclusion:
-> for every prime factor = ‘2’, we multiply the number of permutations by ‘1’
->for every prime factor of the form 4n+1 with an exponent of ‘m’, we multiply the number of permutations by ‘m+1’
->for every prime factor of the form 4n+3 with an exponent of ‘m’, we multiply the number of permutations by ‘1’ if m%2==0 and by ‘0’ if m%2==1

So finally, to mathematically make sense out of everything done till now, we define the function χ[n] (“chi”)

```
In[24]:= χ[t_]= Piecewise[{{1,Mod[t,4]==1},{-1,Mod[t,4]==3},{0,Mod[t,2]==0}}];
```

This function is defined such because, any prime factor of the form 4n+1 can be split into 2 complex Gaussian Integers which are conjugate to each other.
Any prime factor of the form 4n+3 cannot be split further.
Any prime factor of the form 2n does not make a difference to the number of Lattice points on the circle.
And a special property of Chi is its multiplicative property, i.e, χ[n] * χ[m] = χ[m * n]

```
In[25]:= n=2^2 * 3^4 * 5^3;
```

By our previous logic, the number of lattice points on the circle of radius n^1/2 = 4*(1)*(1)*(3+1)=16
this can also be obtained by:
no of Lattice points = 4* (χ[2^0] + χ[2^1] + χ[2^2]) * (χ[3^0] + χ[3^1] + χ[3^2] + χ[3^3] + χ[3^4]) * (χ[5^0] + χ[5^1] + χ[5^2] + χ[5^3])
=4* (1+0+0)* (1-1+1-1+1)* (1+1+1+1)=16
another way to verify this method would be, if we took the 4n+3 type factor with odd exponent, we will end up getting
1-1+1-1+1-1................+1-1=0 in the product, which makes sense as the number of lattice points in that case must be =0.

Now we can take this one step further:
let n=45, so the number of lattice points on the circle with radius 45^1/2 will be equal to
number of lattice points = 4* (χ[3^0] + χ[3^1] + χ[3^2])* (χ[5^0] + χ[5^1])
=4* (χ[1] + χ[3] + χ[5] + χ[9] + χ[15] + χ[45]) (If we expand it and use the multiplicative property)
=4* (Σ χ[t] over all the prime factors ‘t’ of ‘n’)

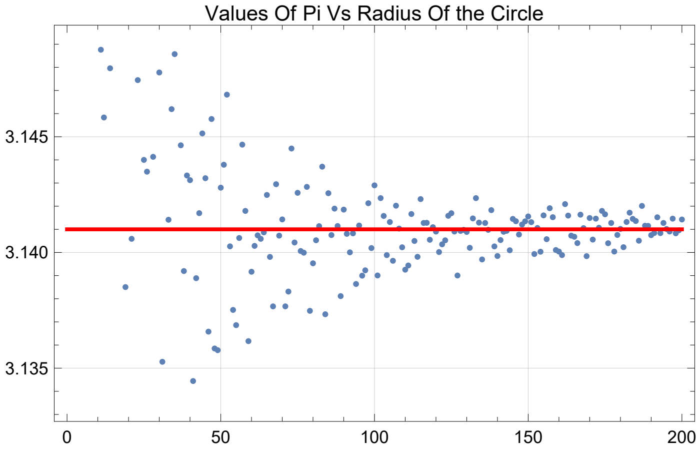
Now trying to approximate the value of Pi using everything above. (note: the error we get when we calculate the value of Pi using this method reduces as the radius of the circle we consider increases.

```
listValsPi={};
listValsRadius={};
LatticePts=0;
For[radius=1,radius≤200,radius++,For[k=1,k≤(radius*radius),k++;ListFactors=Divisors[k];For[l=
LatticePts+=4*χ[ListFactors⟦l⟧];l++]];AppendTo[listValsPi,(LatticePts+1)/(radius*radius)];La
AppendTo[listValsRadius,radius]]
(* 1)radius goes from 1 to 200 as I want to show the variation in the value of Pi calculated v
   2)'k' goes from 1 to radius^2 as the number of Lattice Points in the circle of radius='radi
   Lattice Points on the circles of radius 1, root(2), root(3), root(4),.........,root(radius
   3)The job of the last loop is to find the sum of χ[t] for all 't' which are divisors of k,
   the number of lattice points on it is equal to the summation of the χ values for all the di
   4) While appending LatticePts to listValsPi, I have added '1' to it, as, by the method of s
   divisors for all the values ranging from root(1) to root (radius^2), we are missing the Lat
   I have added one there.*)
listValsPi;
listValsRadius;
```

In[65]:=

```
PiVsRadius=Transpose[{listValsRadius,listValsPi}];
g[x_]=3141/1000;
Plot1=ListPlot[PiVsRadius,Frame→True,GridLines→Automatic,PlotLabel→"Values Of Pi Vs Radius Of
Plot2=Plot[g[x],{x,0,200},PlotStyle→Red];
Show[Plot1,Plot2]
```

Out[69]=



Its Clearly visible that as the radius of the circle gets bigger, the values tend to Pi.

By this, we can say that the value of Pi calculated by our algorithm will have zero error when the radius of the circle tends to Infinity.

When the radius of the circle is tends to Infinity:

Number if Lattice Points:

for the circle of radius 1	: 4* χ[1]
for the circle of radius Root[2]	: 4* (χ[1] + χ[2])
for the circle of radius Root[3]	: 4* (χ[1] + χ[3])
for the circle of radius Root[4]	: 4* (χ[1] + χ[2] + χ[4])
for the circle of radius Root[5]	: 4* (χ[1] + χ[5])
for the circle of radius Root[6]	: 4* (χ[1] + χ[2] + χ[3] + χ[6])

for the circle of radius Root[7]   : 4* (χ[1]   +   χ[7])

for the circle of radius Root[8]   : 4* (χ[1] + χ[2]   +   χ[4]   +   χ[8])

And the pattern continues till radius is equal to Root[R^2]...........

so let R be the radius of that circle and R-> ∞

we get:

Area = Number of Lattice Points

Pi * R^2 = R^2 * 4* (χ[1] + χ[2]/2 + χ[3]/3 + χ[4]/4 + χ[5]/5 + χ[6]/6 + χ[7]/7 + χ[8]/8........................)

(As the number of times χ[n] appears can be approximated to be R^2/n)

=>Pi /4 = (χ[1] + χ[2]/2 + χ[3]/3 + χ[4]/4 + χ[5]/5 + χ[6]/6 + χ[7]/7 + χ[8]/8........................)

=>Pi = 4* (1 -1/3 + 1/5 - 1/7 + 1/9 - 1/11.................... and so on till ∞)

The above is a well known Series which Converges to Pi

References:

1) https://www.youtube.com/watch?v=NaL_Cb42WyY&t=283s (A video by the youtube channel “3Blue1Brown”)

2) https://en.wikipedia.org/wiki/Logistic_map

3) https://reference.wolfram.com/language/ref/GaussianIntegers.html?view=all

4) https://www.youtube.com/watch?v=ETrYE4MdoLQ

5) https://en.wikipedia.org/wiki/Feigenbaum_constants

6) https://reference.wolfram.com/language/ref/FindTransientRepeat.html

7) https://marksmath.org/visualization/LorenzExperiment/