Computational Laboratory with Fortran Coding

Expanding Civilization

and

Monte Carlo Integration

Report by: Taimur Ahmad Khan

Introduction

Programming since its proper birth in 1843, has developed in its own world as well as worlds of natural sciences. Common problems in theory and experiment in physics have been tackled by the computational approach. A common example addressed in this report is the Monte Carlo integration, and mathematical sequences.

The first problem executed; *The expanding civilization*, is a mathematical sequence designed to predict and describe the growth of a population and its sustainability with limited resources. Its development or stagnation determined by variables and parameters set and not affected by external factors.

The Monte Carlo integration used here to find the area under a trigonometric curve; between two specific points and to obtain the volume between two spheres.

Numerical calculations and assessment of complicated problems are simplified by programming. In these two problems, a large quantity of data points need to be examined and assessed in a way that are difficult for physicists to do on their own without computational help. For the sequencing, a hundred values for fifty varying initial conditions and two thousand different parameters each are found to give accurate graphical representations of the population rate of growth and population. The Monte Carlo, computes integrals by generating sequences of random events using random input of values.

Small executed lines of code go over solutions in a time more benefitting than physical techniques.

Problem 1: The Sequence

A mathematical expression for this sequence is given by;

(i)

Examining equation (i), two variables; **x** and **a** are looked at. “**a**” describing the rate of development and “**x**” representing the population itself. x takes values between its boundary conditions of 1 and 0. The population maximum and minimum (when the population=0) are represented by these boundary conditions respectively. Looking at the equation one can see that as is related to by a limiting factor or decreasing factor of . This term ensures that when the civilization population reaches maximum and the resources are exhausted, the next step is = 0 or more generally, civilization collapse. For a continuity of the sequence the following bit of code is written;

x = a \*x0\*(1-x0) (1)

x0 = x

In the question, an assumption is made that the convergence point of this sequence comes within a hundred iterations of it running. The “do loop” is used to produce those hundred iterations without repeating the same lines of code.

call random\_number (x0) (2)

do i = 1, 100

x = a\*x0\*(1-x0)

x0 = x

end do

The first loop is made to find the convergence point of this sequence. “call random\_number (x0)” is used to set the initial condition x0 or as a random number between 1 and 0. This value is put in the sequence to result in a value for x or . This entire task is then repeated for a hundred iterations.

The second “do loop” is used to repeat the operation with varying initial conditions all with values in the interval of [0,1]. Fifty versions of the previous procedure are executed with random values set as x0.

do j = 1, 50 (3)

call random\_number (x0)

do i = 1, 100

x = a\*x0\*(1-x0)

x0 = x

end do

write (100, \*) a, x

end do

For each of the fifty initial conditions called at random, a hundred iterations take place for the convergence point. The value for **a** under which these iterations take place along with the numerical value of **x** are given as output. As these values are later needed in a graphical representation, the values are stored in a file called fort.100 that can be extracted and used in the plotting tool xmgrace. The “i” loop is nested within the “j” loop.

Until now, the value of the rate parameter **a** set has been constant. A third ‘do loop’ which nests the above two mentioned is set for two thousand linearly changing values of **a** by an amount ‘VARa’.

do k = 1, 2000 (4)

a = a+VARa

do j = 1, 50

call random\_number (x0)

do i = 1, 100

x = a\*x0\*(1-x0)

x0 = x

end do

write (100, \*) a, x

end do

end do

The increment by which it changed; VARa, is calculated equidistant intervals between the initial and final value of rate parameter **a**. As two thousand iterations are required;

(ii)

Where in this case, N = 2000. Spanning this interval, each of the two thousand values in between the initial and final cycle give way to fifty varying initial conditions for resulting iterations of the convergence of x. These three ‘do loops’ are the basis of this code and the values they produce are graphically plotted with the convergence points of population on the x axis and the values of growth rate which produce them on the y axis. The range of the two thousand points of **a** are changed.

Growing Rate (a)

Population Point (x)

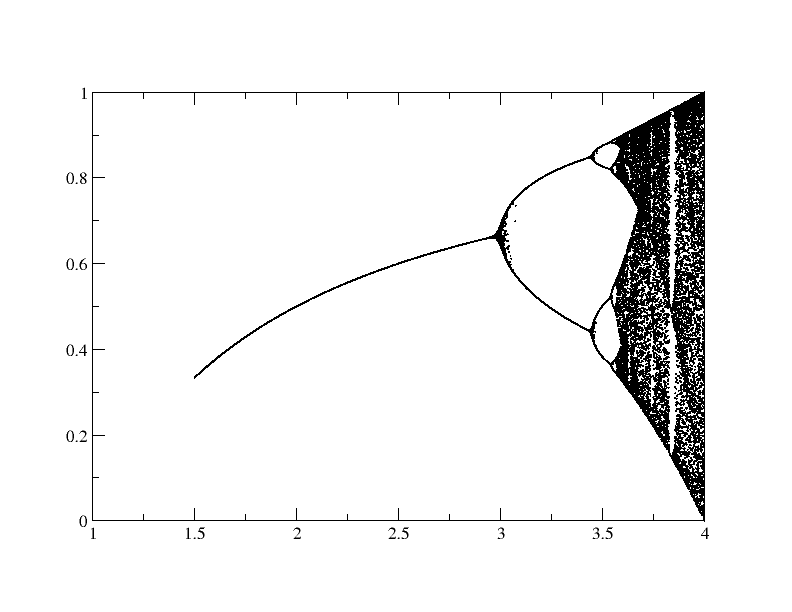
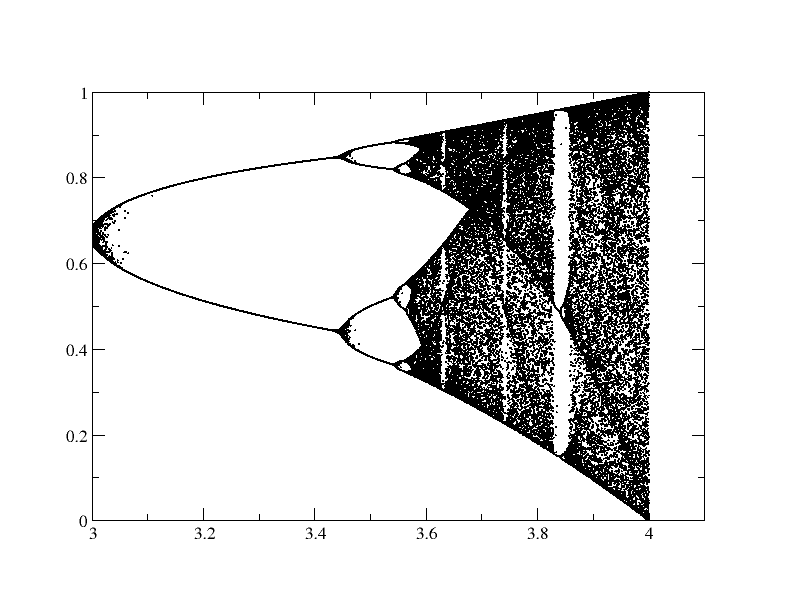


Figure (1)

Interval of “a” [1.5,4]

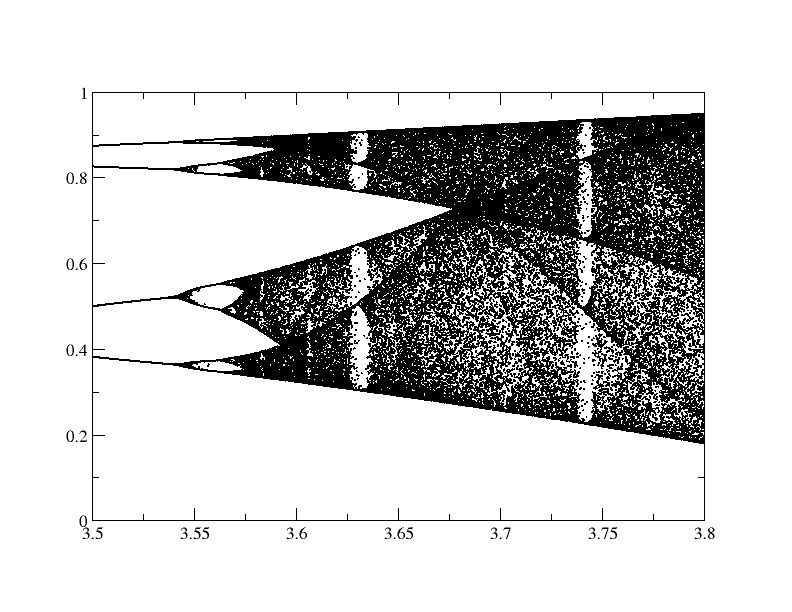


Population Point (x)

Growing Rate (a)

Figure (2)

Interval of “a” [3,4]



Population Point (x)

Growing Rate (a)

Figure (3)

Interval of “a” [3.5,3.8]

Analysing the graphics, for the value of growth rate = 1.5, there is one population point (convergence point). The intervals of **a** for which there are multiple values of population points are **a** ≥ 3. From **a** = 1 to **a** = 3, there are singular values for population points or convergence points. As soon as the growth rate crosses the value of three, many values of **x** are assigned to the one value of **a**. There is a constant increase in population size as growth rate increases. These multiple convergence points show instability as its increasing with its limited resources. When **a** tends to 4, the graphic spans the entire vertical axis with the population points at both maximum and minimum values of **x**, showing both a rise to maximum and then immediate civilisation collapse.

The graphs are continuations of each other at ranging from **a** [1.5,4] and the second and third graph showing more detailed sections within this range. There properties are the same and can be referred to as *self-similar* graphics. An overall trend is shown at different specific sections of this sequence.

Problem 2: The Monte Carlo

The basis of this algorithm comes from probability of success and failure. This algorithm computes integrals using generation and manipulation of random numbers. Imagine a two-dimensional circle whose area is to be found. Let the circle be inside an imaginary shape, in this case a square of side equal to the diameter of the circle in question. If one throws stones inside the square and counts the total number of hits with the total number of hits inside the circle, using the formula;

(iii)

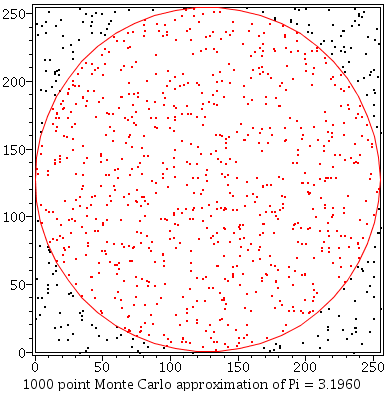
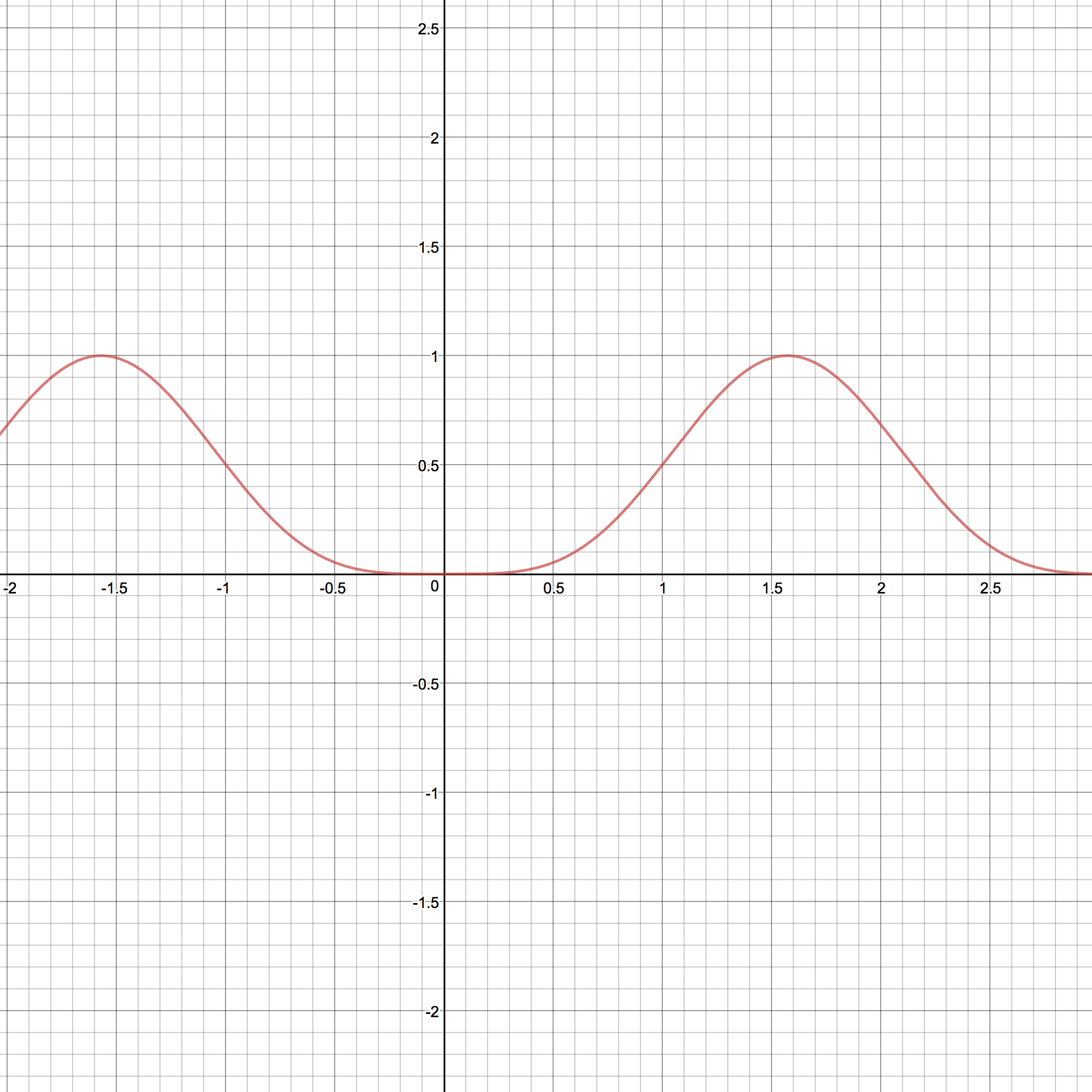


Figure (4)

In this case, the total area is equal to the area of the square. It sets a counter and measures the probability of hits and misses and using a coordinate system to limit the location of integral needed, calculates from the range of the limit whether the stone hit the area required in the integral. For volume, an added coordinate range with limits can be used to signify three dimensions and a three dimensional integral required. It is used in the calculations of volume and area as it gives a good approximate answer to the actual integral solution.

Looking at the first part of problem 2, this algorithm is adapted to a new shape. The curve of in the limit from x = 0 and x = π/2;

 Figure (5)

Y

1 Units

X

π/2 Units

The area under the red curve is the required Monte Carlo integral, the curve can be placed in an imaginary shape of a rectangle with sides 1 and π/2 units. The number of hits (stones randomly hitting the area) both total and within the integral are set to 0 and a value of pi is set.

N=10000 (5)

Nhit=0

Ntot=0

pi= acos(dble(-1))

Let the area of the rectangle be equal to π/2 units squared as the product of its lengths. As done in the Monte Carlo code, two numbers in the interval of [0,1] are called at random.

call random\_number (r) (6)

call random\_number (s)

Limits are required to be set on the two axes as a function of the random numbers called. For the area of the circle,

x= 2\*r - 1 (7)

y= 2\*s - 1

if (x\*\*2 + y\*\*2 < 1) then

Nhit=Nhit+1

endif

Ntot=Ntot+1

so that the values of x and y come in the range and domain of [-1,1] with values of r and s having a limited interval of [0,1]. The two boundary values satisfy the solution. These parameters were followed by a condition to generate a certain output if met and if not met. This condition was set by using the ‘if else’ statement. This condition if met means that the random thrown stone has hit the required area. Therefore, number of hits (Nhit) count is increased by one. Ntot or total hits inside and outside of the circle has no relation with this statement and will increase by one even if condition is not met.

For the case of this trigonometric function, since the random input is in the range [0,1], the following parameters;

x=r\*(pi/2) (8)

y=s\*(1)

allow the range of x and y to not exceed the limits of [0, π/2] and [0,1] respectively. Using the same type of ‘if else’ statement as used in the area of the circle, to complete the parameter that if the y coordinate magnitude is less than the value of at that point then it is within the limits of the boundary of the integral and will be counted as a hit inside the integral shape and the count of Nhit will be increased by one. This condition’s results do not affect the value of Ntot or total hits within the rectangular region.

if ((sin(x))\*\*4>y) then (9)

Nhit=Nhit+1

endif

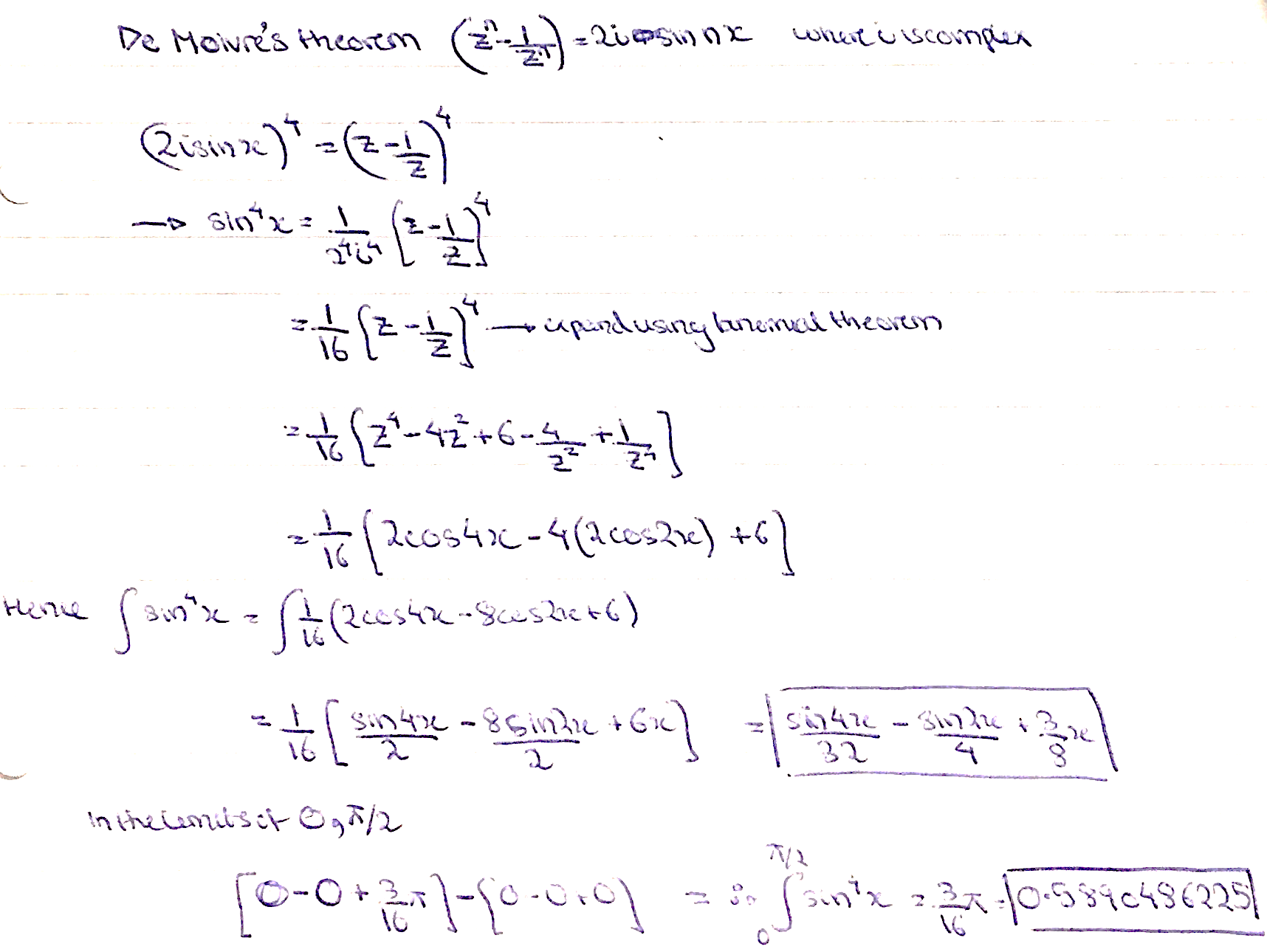
Ntot=Ntot+1

These conditions are nested within a loop from 1 to N which in this case is ten thousand. After the necessary iterations take place, the formula for area as seen in equation (iii) is modified to fit this particular problem;

dble(Nhit)/dble(Ntot))\*(pi/2) (10)

where π/2 is the area of the rectangle. The area found from the ratios is an approximation to the integral of the trigonometric function in the limit [0, π/2]. The obtained estimate is area = 0.59093357814024006 . The actual answer computed with written calculation of the integral of is written below;

Figure (6)



In the Monte Carlo the error allowed is given by where N is the number of iterations for which hits are recorded. The error allowed in this problem, with 10,000 trials is equal to 0.01 The difference between the values of the integral through the Monte Carlo algorithm and normal integration is which is less than the error allowed showing that the approximation was successful.

The second part of this problem required the volume between two spheres of radius 1 units and 0.75 units. The lightly shaded region in the diagram below is the volume required and computed in the following lines of code.

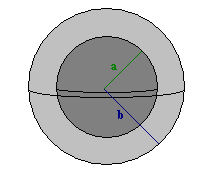


Figure (7) Concentric Spheres.

The framing box set is a cube of side 2 units as the limits of x,y, and z; [-1,1] have a difference of 2 units. As the volume requires three dimensions, an additional parameter for an additional axis is modified in the code.

call random\_number (r) (11)

call random\_number (s)

call random\_number (t)

The third parameter t along with s and r become functions of the axes to set limits on the space required. Notice the parameters are similar to that of a circle which required limits of [-1,1] from inputs ranging from [0,1]. The conditional parameter, again nested in an ‘if else’ statement follows. From polar coordinates, (the sum of the magnitudes of points on the three axes squared) is equal to the radial distance squared. If the sum of the squares of the values of x,y and z are less than the radius 1 squared and greater than the radius 0.75 squared, then these points come in between the large and small sphere and satisfy the volume in question.

x= -1 + 2\*r (12)

y= -1 + 2\*s

z= -1 + 2\*t

V = x\*\*2 + y\*\*2 + z\*\*2

if (1\*\*2>V .and. 0.75\*\*2<V) then

Nhit=Nhit+1

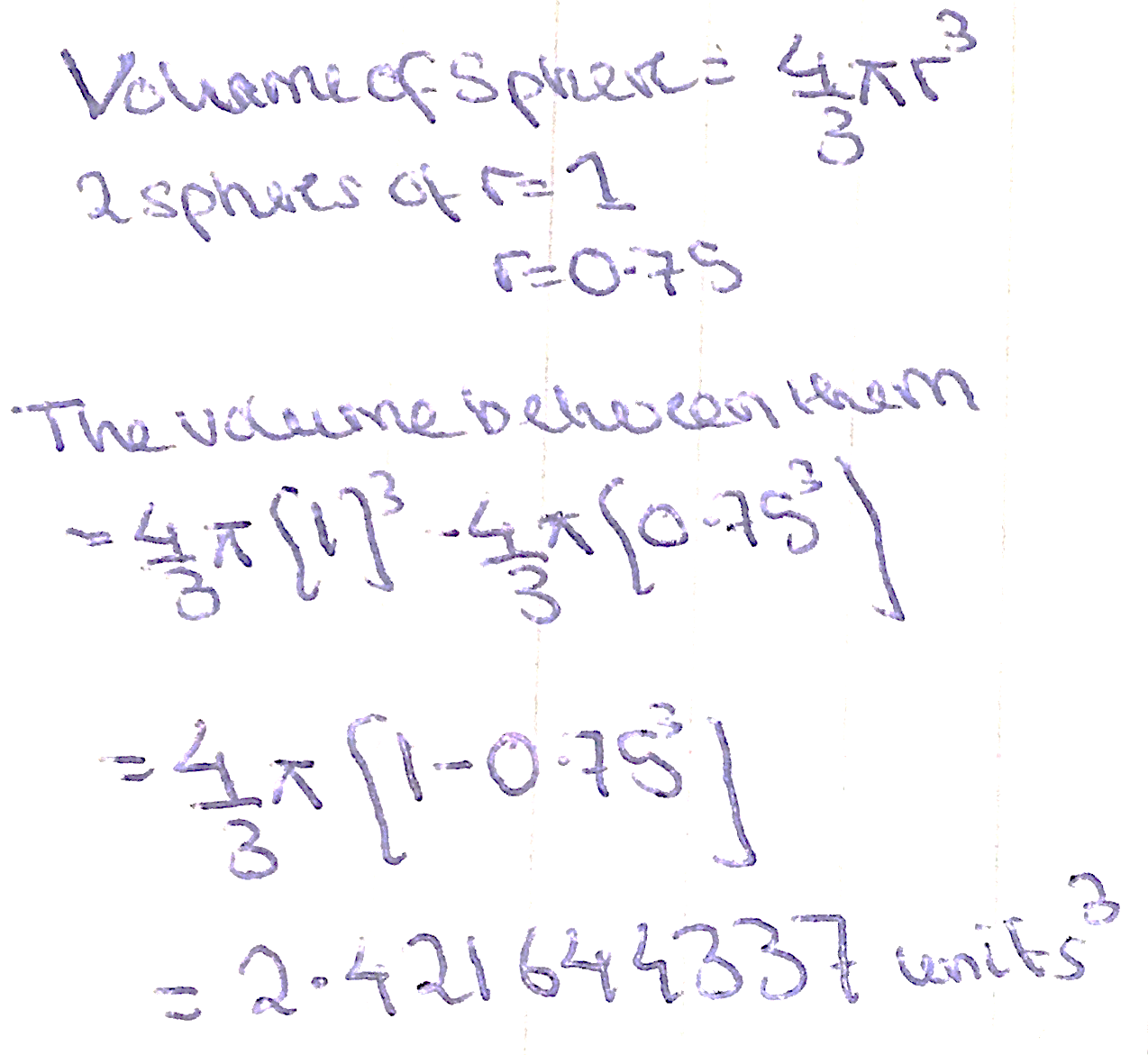
endif

Ntot=Ntot+1

When satisfied, the counter on hits within required volume will increase by one as will the total hits which is not bound by the condition set and will increase nonetheless. The volume is calculated using the ratio of the two volumes instead of areas used in the previous code;

(dble(Nhit)/dble(Ntot))\*(dble(8)) (13)

Where 8 is the volume of the framing box (cube) of side 2 units. The volume computed is 2.42240001 . The exact answer is given by;



The error calculated in the same way before results in the same 0.01 being the allowed value and calculated difference of . The calculated difference is less than the error allowed showing again a successful approximation. The errors from the two parts of this problem are different from each other when the power of the Monte Carlo algorithm comes from the fact that its error is the same and will not change according to any dimensional changes (changes from 2D area to 3D volume). A regular mesh of points would in itself be small three-dimensional blocks to build up the interval. Although the answer computed may be more accurate, the programming required to implement it would be very complex whereas in the case of our code, only one more parameter had to be added to the simple circle problem to find the volume. The Monte Carlo works better for problems involving more than six variables. When the number of measurements is large, it will give a better approximation to the result than mesh of points or periodic blocks.

Conclusion

In this course, the programming language Fortran was taught to develop a computational resource within students. As mentioned in the introductions, the practical use of computing in branches of natural science, has increased. Two mathematical problems, that require complex calculations and data gathering has been toned down to simple lines of code. A sequence to predict and examine population growth with resources available. A random generation of numbers to calculate the correct value of area and volume of a given space. These two problems were some of the few computed in this course along with the Fibonacci sequence and various probabilistic mathematic expressions. With its parallel development with physics, this programming module will give access to a better understanding of this course.

Appendices

Problem 1 Code

program problemone

implicit none

real (8) :: x0,x,a,VARa

integer (4) :: i,j,k

a=3.5

VARa= (3.8-3.5)/2000.0

do k=1,2000

a=a+VARa

do j=1,50

call random\_number (x0)

do i=1,100

x=a\*x0\*(1-x0)

x0=x

end do

write (100,\*) a,x

end do

end do

end program

Problem 2 Code Part 1

program montecarlointegration

implicit none

real (8) :: pi,r,s

real (8) :: x,y

integer (4) :: Nhit, Ntot, i, N

N=10000

Nhit=0

Ntot=0

pi= acos(dble(-1))

do i= 1,N

call random\_number (r)

call random\_number (s)

x=r\*(pi/2)

y=s

if ((sin(x))\*\*4>y) then

Nhit=Nhit+1

endif

Ntot=Ntot+1

end do

write(\*,\*) 'Ntot =', Ntot

write(\*,\*) 'Nhit =', Nhit

write(\*,\*) 'Area =', (dble(Nhit)/dble(Ntot))\*(pi/2)

end program

Problem 2 code Part 2

program montecarlosphere

implicit none

real (8) :: r,s,t,V

real (8) :: x,y,z

integer (4) :: Nhit, Ntot, i, N

N=10000

Nhit=0

Ntot=0

do i= 1,N

call random\_number (r)

call random\_number (s)

call random\_number (t)

x= -1 + 2\*r

y= -1 + 2\*s

z= -1 + 2\*t

V = x\*\*2 + y\*\*2 + z\*\*2

if (1\*\*2>V .and. 0.75\*\*2<V) then

Nhit=Nhit+1

endif

Ntot=Ntot+1

end do

write(\*,\*) 'Ntot =', Ntot

write(\*,\*) 'Nhit =', Nhit

write(\*,\*) 'Volume =', (dble(Nhit)/dble(Ntot))\*(dble(8))

end program

Bibliography

<https://www.mapleprimes.com/maplesoftblog/95508-Monte-Carlo-Pi>

<https://www.physicsforums.com/threads/electrostatics-potential-between-and-outside-concentric-spheres.538712/>

<https://en.wikipedia.org/wiki/Self-similarity>