GSCP Portfolio

- **Olivia J. Clark**

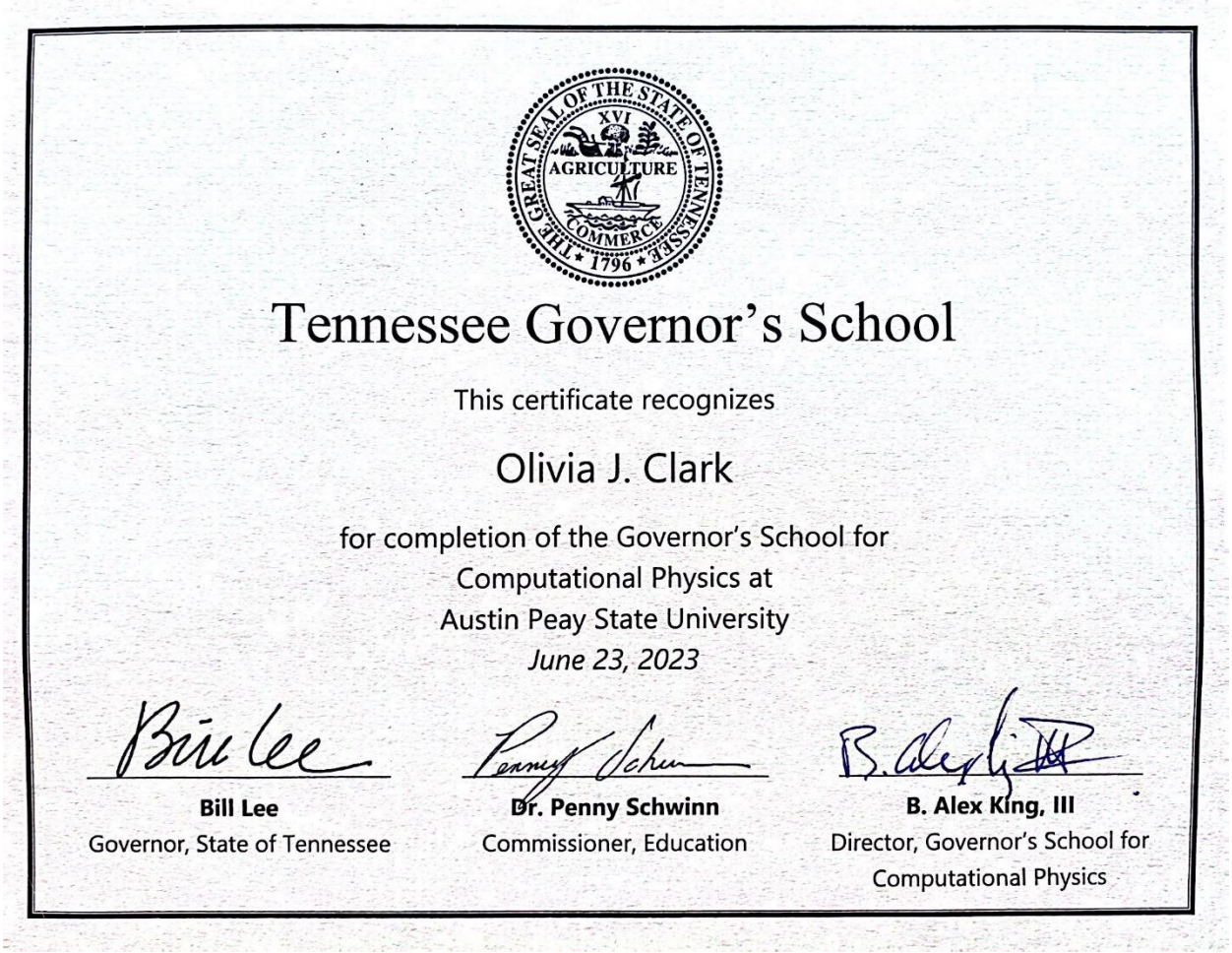
June 2023

Governor's School for Computational Physics

Austin Peay State University – Clarksville, TN

Acknowledgements:

First, I'd like to thank the coordinators and professors involved in this program for investing in the education of young, ambitious students across Tennessee. I think I can speak for the other 30 students in this group and plenty before in saying that the opportunities and experience received through the Governor's School have planted a seed in the progress of my development that I hope to cultivate into my future in academia.



Tennessee Governor's School

This certificate recognizes

Olivia J. Clark

for completion of the Governor's School for
Computational Physics at
Austin Peay State University
June 23, 2023

Bill Lee
Governor, State of Tennessee

Dr. Penny Schwinn
Commissioner, Education

B. Alex King, III
Director, Governor's School for
Computational Physics

Coordinator Sherry Bagwell – BagwellS@apsu.edu

Professor Alexander King – KingA@apsu.edu

Professor Daniel Mayo – mayod@apsu.edu

Preface

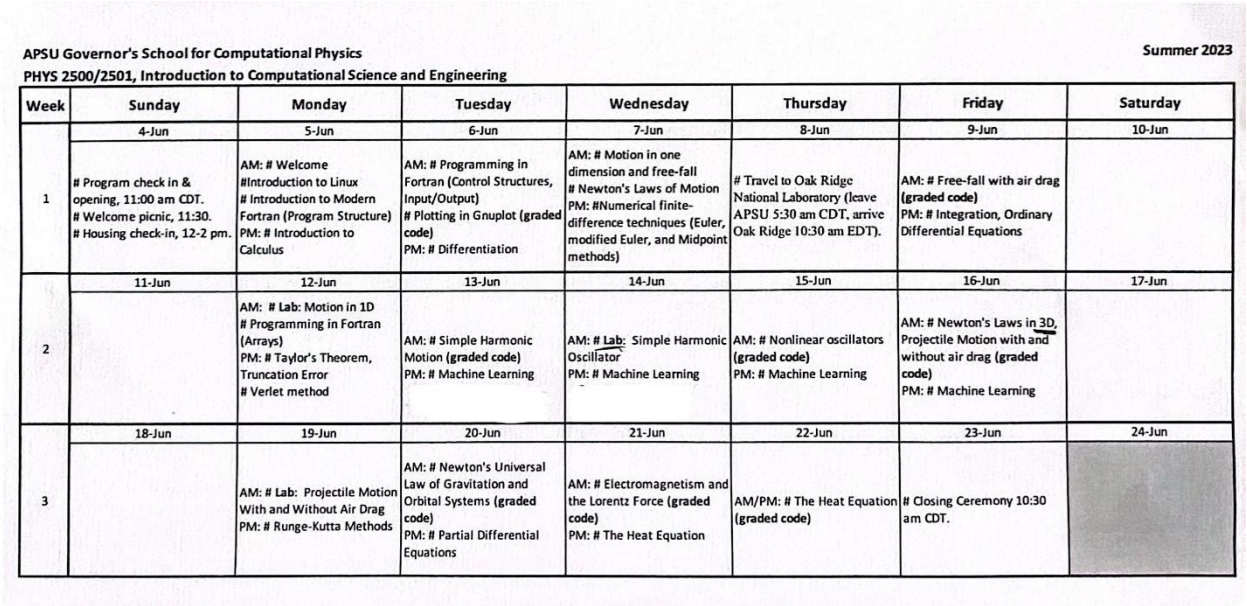
Tennessee Governor's Schools are selective state programs offering striving students with opportunities for accelerated summer learning in a given field. The Governor's School for Computational Physics at Austin Peay State University is a 3-week program serving as an introduction and practice of computational science and engineering. Here, students maintain a rigorous schedule of focused lectures, assignments, and lab work. Additional to coursework, students at GSCP are given opportunities for various lab and facility tours as well as educational field trips and exciting scientific demos. It provided me with a standard on which to base my future university experience on along with a strong foundation for what I plan to continue to study.

I came into GSCP having taken physics and calculus, but with no previous computer science experience. I've left with a sense that the three are almost inseparable, and with a desire to see more work done towards their synthesis. Due to the accelerated pace of the program, I've felt this way about many aspects of what I learned there- feeling as though I had drank from the firehose of computational science- leaving with a desire to retain and expand upon the skills I gained there. I still find myself looking to my old notebooks to satisfy my curiosity and guide my current studies. The given coursework did not include the creation of a portfolio, however, considering the inspiration that it ensued, I feel that it's appropriate to formulate the foundations of my education in this realm. I hope to continue this practice and to be able to look back at this portfolio and my GSCP experience as the first step among many in this field.

Contents:

Introduction

Our coursework at GSCP consisted of 4.5 hours of class time per day, the first block being a 3 hour session of computer science and physics and the second as our math class which spanned from an intro to calculus to series expansions and partial differential equations, all with a block in the middle to explore concepts in machine learning. Provided with our laptops operating on Ubuntu Linux, our computer science lectures were structured around coupling the basics of our new machines with programming in Fortran 95. We learned and practiced essential programming concepts in order to maintain the sophistication and efficiency that's required in our physical simulations. Our physics course carried us throughout calculus-based mechanics, ending off with a lesson in electromagnetism. The course syllabus was designed as follows:



APSU Governor's School for Computational Physics **Summer 2023**

PHYS 2500/2501, Introduction to Computational Science and Engineering

Week	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	4-Jun	5-Jun	6-Jun	7-Jun	8-Jun	9-Jun	10-Jun
1	# Program check in & opening, 11:00 am CDT. # Welcome picnic, 11:30. # Housing check-in, 12-2 pm.	AM: # Welcome #Introduction to Linux # Introduction to Modern Fortran (Program Structure) PM: # Introduction to Calculus	AM: # Programming in Fortran (Control Structures, Input/Output) # Plotting in Gnuplot (graded code) PM: # Differentiation	AM: # Motion in one dimension and free-fall # Newton's Laws of Motion PM: #Numerical finite-difference techniques (Euler, modified Euler, and Midpoint methods)	# Travel to Oak Ridge National Laboratory (leave APSU 5:30 am CDT, arrive Oak Ridge 10:30 am EDT).	AM: # Free-fall with air drag (graded code) PM: # Integration, Ordinary Differential Equations	
	11-Jun	12-Jun	13-Jun	14-Jun	15-Jun	16-Jun	17-Jun
2		AM: # Lab: Motion in 1D # Programming in Fortran (Arrays) PM: # Taylor's Theorem, Truncation Error # Verlet method	AM: # Simple Harmonic Motion (graded code) PM: # Machine Learning	AM: # Lab: Simple Harmonic Oscillator PM: # Machine Learning	AM: # Nonlinear oscillators (graded code) PM: # Machine Learning	AM: # Newton's Laws in 3D, Projectile Motion with and without air drag (graded code) PM: # Machine Learning	
	18-Jun	19-Jun	20-Jun	21-Jun	22-Jun	23-Jun	24-Jun
3		AM: # Lab: Projectile Motion With and Without Air Drag PM: # Runge-Kutta Methods	AM: # Newton's Universal Law of Gravitation and Orbital Systems (graded code) PM: # Partial Differential Equations	AM: # Electromagnetism and the Lorentz Force (graded code) PM: # The Heat Equation	AM/PM: # The Heat Equation (graded code)	# Closing Ceremony 10:30 am CDT.	

Course syllabus for GSCP 2023

And needless to say, we were able to pack a lot of learning content into our 3-week block.

The course is aimed at the synthesis of the three subjects into a study of computational physics. Computational science is the application of computational, numerical techniques in solving scientific problems. I find it worth noting that techniques of computational physics were a strong factor in the initial development of computer technology and allow us to solve problems that wouldn't have otherwise been possible, creating a new science in doing so. These problem-solving capabilities often take the guise of approximation methods that allow us to solve complex problems much easier and faster. One tradeoff, it seems, is in accuracy versus efficiency. However, as our computers and approximation methods advance, we see more and more so how the best of both worlds can be achieved. The methods used throughout this portfolio will serve as an example of this feat of technology and the mathematics that exemplify it, while aiming to create sophisticated representations of physical phenomena in doing so.

The following portfolio serves as a compilation of my coursework completed at the GSCP. Aside from our first assignment, I've divided the contents by the physical problems being studied. Each assignment alloted a physical phenomenon that we were tasked with simulating based on the approximation methods previously described. These simulations are aimed at creating a useful and accurate description of the object being simulated. For our purposes, this generally means representing the evolution of the objects state or trajectory over time for a given physical scenario. For each assignment, I will give a description of the new challenges presented physically, mathematically, and computationally, as well as the code I've written and the simulation it produces.

Fourier Approximation:

In hindsight, I'd like to comment on the elegance of our first assignment as an intimidating, yet conquerable example of the power of the field that we were being introduced to. A Fourier Series approximation is an advanced numerical approximation technique that approximates a curve using an infinite series of sine and cosine functions. These functions are by nature periodic, oscillatory, and wavelike. However, the Fourier series tells us that- with enough computational power to approximate an "infinite" series- any curve can be approximated no matter how non-wavelike. Our challenge was to demonstrate the power of this approximation by producing code that creates 1) a jagged, "sawtooth" piecewise function, and 2) a Fourier approximation of the sawtooth function for comparison.

The sawtooth function was defined as:

$$f(x) = \begin{cases} (x), 0 \leq x < \pi \\ (x - 2\pi), \pi < x \leq 2\pi \end{cases}$$

And the Fourier Series states:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \, cos(nx) + \sum_{n=1}^{\infty} b_n \, sin(nx)$$

Where, the summation in indexed based on n and coefficients a_n and b_n are used to describe the amplitude and phase of the oscillation and, in our case, are

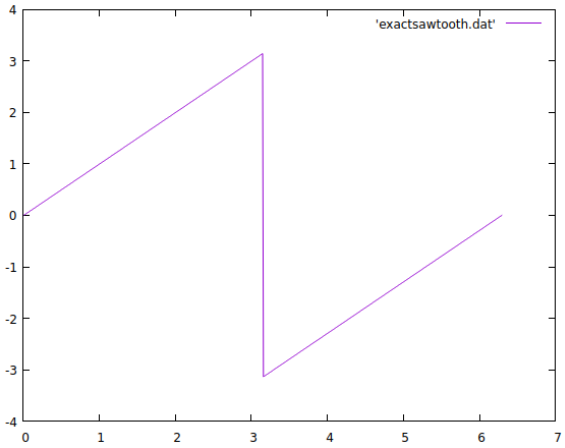
$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(s)\, cos(ns)\, ds \text{ and } b_n = \frac{1}{\pi}\int_0^{2\pi} f(s)\sin(ds)$$

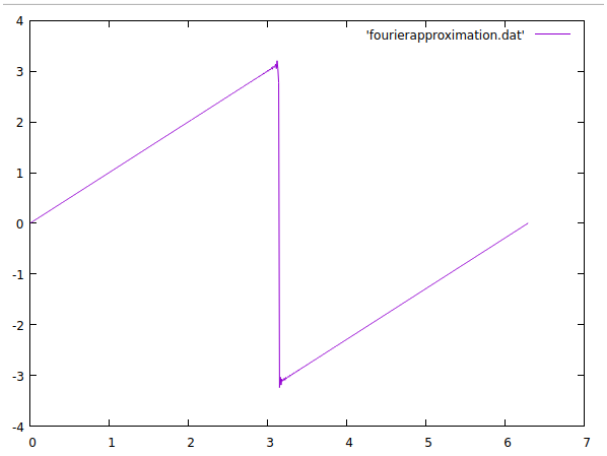
The code is as follows:

```
program exactsawtooth

implicit none

real(8):: y, x, pi
integer(4):: i

pi = 4*atan(1.0)
x=0.0

open(unit=100, file='exactsawtooth.dat')
write(100,*) "# x-values        y-values"

do while (x<=2.0*pi)
if(x<=pi) then
y=x
elseif (x>pi) then
y=x-(2.0*pi)

endif

x=x+0.01


write(100,*) x,y

enddo

close(100)

end program exactsawtooth
```

```
program fourierapproximation

implicit none

real(8):: x, y, pi
integer(4):: n
pi = 4.0*atan(1.0)

open(unit=200, file='fourierapproximation.dat')

write(200,*)   '# x-values       y-values'

x=0.0

do while (x<=2.0*pi)

y=0.0

do n=1,1000
y=y+((((-1.0)**(n+1.0))*2.0)/n)*sin(n*x)
enddo

write(200,*) x, y
x=x+0.01

enddo

end program fourierapproximation
```

Where we use loops to iterate through the n index, evolving y-values based on small x-increments and writing the values in a datafile each time the loop is completed in order to collect coordinates for a complete picture of each function. The flow of the Sawtooth program is controlled uisng the “do while” statements that function the same as the branches of the piecewise function, whereas the Fourier code applies to all values within the original domain. The graphs produced are almost identical:



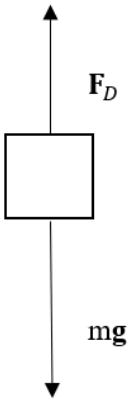


Demonstrating the approximation power of the Fourier Series.

Freefall:

Of course, our introduction to physics was in the exploration of the kinematics equations in 1D. Here, we examined a freefall scenario in conjunction with learning Euler's approximation methods. Whereas with our Fourier assignment we exposed how to approximate a curve in somewhat of a pure mathematical experiment, here we saw how our approximations can be applicable to curves with a physical meaning.

Our product was to be a simulation of freefall describing the position, velocity, and acceleration of the particle using Euler's 1st and 2nd methods while accounting for air drag. This particular problem aims to describe an object with a mass of 145 grams that is thrown downward at a speed of 3.8 m/s from a height of 57 meters. Yielding:



$$\sum F = F_D - mg \Rightarrow \sum a = \frac{F_D}{m} - g$$

where $F_D = kv^2$ and $k = \frac{1}{2} C_D \rho A$ which is determined to be

$$\approx 0.00132 \text{ kg/m}$$

We use both Euler's 1st Method, stated as:

$$y_1 = y_0 + h \cdot f(x_0, y_0)$$

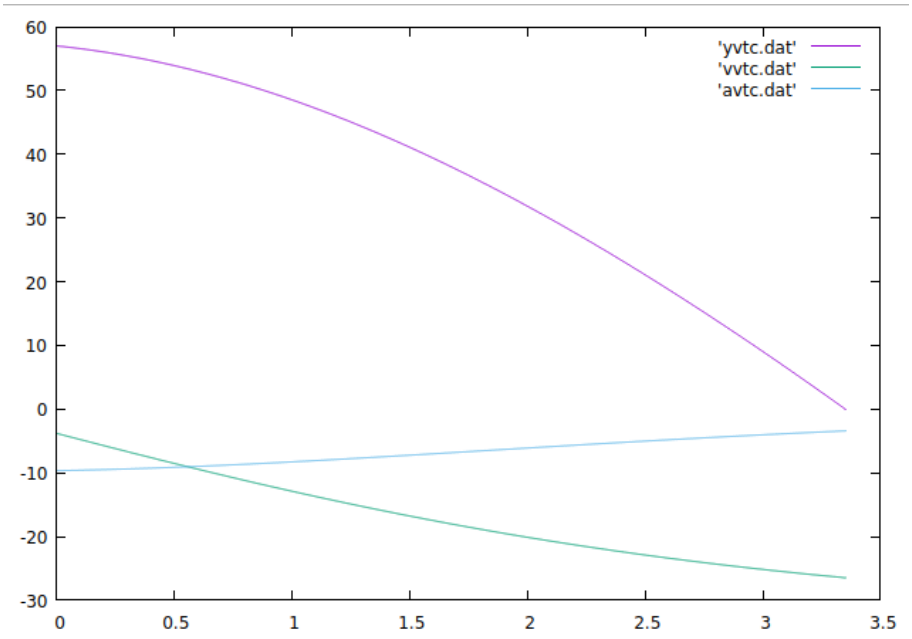
And Euler's Improved Method:

$$y_1 = y_0 + h \cdot \left(\frac{f(x_0, y_0) + f(x_1, y_1)}{2} \right)$$

Where f is the derivative function of y with respect to x and h is the given x-increment, meaning that each next point along the curve is determined by its current value as well as the value of its derivative. Both methods demonstrate how we can approximate values for velocity and position based on the objects acceleration, but the comparison here demonstrates how greater accuracy can be achieved through the use of more complex approximation methods. Taking the average of new and old slopes removes room for error in our approximation. Implementing Euler's Method with the initial conditions of the system and what we know about freefall motion allows the program to be written as:

```
1  program cfreefall
2  implicit none
3  real(8):: y, v, t, a, h, g
4  real(8):: ys, vs, ts, as
5  real (8):: k, m
6
7  open(unit=120, file='yvtc.dat')
8  write(120,*) '#t-values, y-values'
9  open(unit=220, file='vvtc.dat')
10 write(220,*) '#t-values, v-values'
11 open(unit=320, file='avtc.dat')
12 write(320,*) '#t-values, a-values'
13
14 h= 0.001
15 y= 57.0
16 v= -3.8
17 t= 0.0
18 g= -9.8
19 k= 0.00132 !k= 0.5*dc*csa*p
20 m= .145
21
22 do while (y>=0)
23
24 vs=v
25 v=v+a*h
26 a=((k*(v**2))/m)+g
27 y=y+((vs+v)/2)*h
28
29 write(120,*) t, y
30 write(220,*) t, v
31 write(320,*) t, a
32 t=t+h
33
34 enddo
35 endprogram cfreefall
```

Where Euler's 1st method is used to approximate the velocity "v", and Euler's 2nd is used to find the position, y, based on the average of "v" and "vs". Here, we use a "do while" loop to tell the program to stop once the object hits the ground. The following graph is produced:



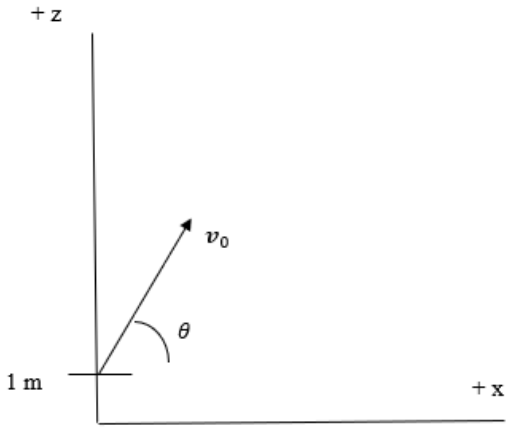
Freefall Position, Velocity, and Acceleration versus Time

Where, since air drag is accounted for, we see a decrease in the magnitude of the acceleration over time as the object approaches terminal velocity. Therefore, we're able to conclude some validity in our simulation despite a lack of experimental comparison.

Projectile Motion:

Stepping our freefall scenario up a notch, we introduce 2D projectile motion with the same approximation scheme. We were tasked with representing the evolution of the objects state over time, a 3D plot of the object's trajectory, and to determine the maximum height, range, and time of flight of the projectile for any initial angle and velocity. For the sake of this example, the

initial angle and velocity are 45 degrees and 20 m/s, and the object is launched from an initial position of 1 m **k**. The background is as follows:



For a given initial velocity and angle, we can decompose the velocity vector into it's individual components with

$$V_x = V_0 \cos\theta \; ; \; V_y = V_0 \sin\theta$$

With these components and the fact that there is no acceleration in the horizontal direction, we can find the time it takes to reach the maximum height using the 1st kinematics equation:

$$v_f = v_0 - gt \Rightarrow t_{up} = \frac{v_y}{g}$$

Where the final velocity in the y-direction at the top of the arc is zero. Coupling this with the 3rd kinematics equation allows us to find the maximum height of the projectile,

$$v_f^2 = v_0^2 + 2gh \Rightarrow h_{max} = \frac{v_y^2}{2g}$$

Using the time of flight to the top and exploiting the symmetry of the motion, 2nd kinematics equation allows us to solve for the range of the projectile,

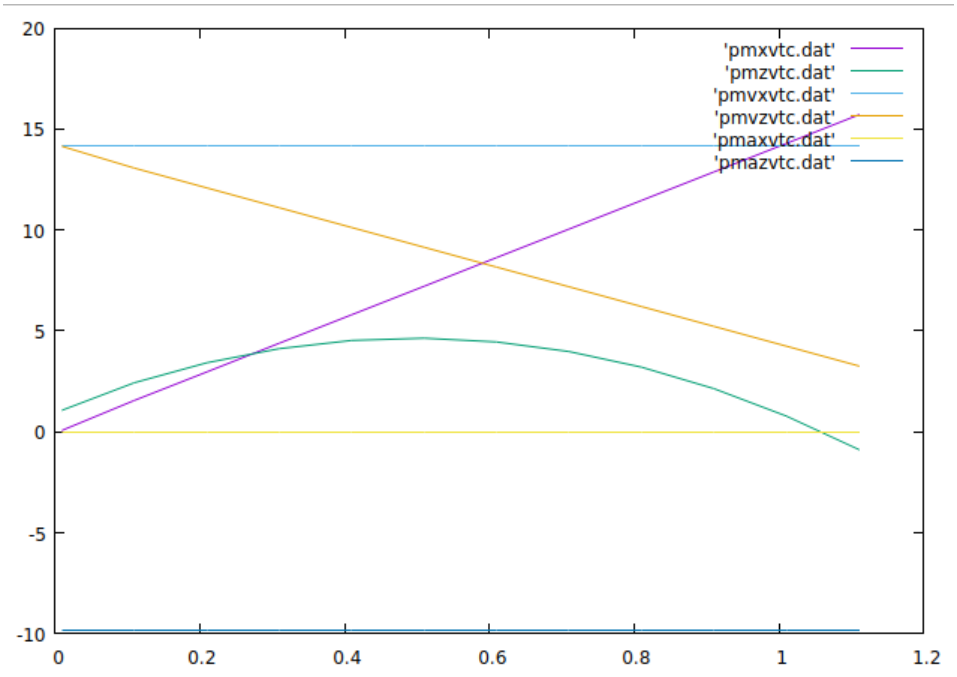
$$\Delta x = v_x t$$

We will use the same combination of Euler's methods as our freefall problem for our mathematical background, however, the increased dimensionality requires new computational techniques. The code is as follows:

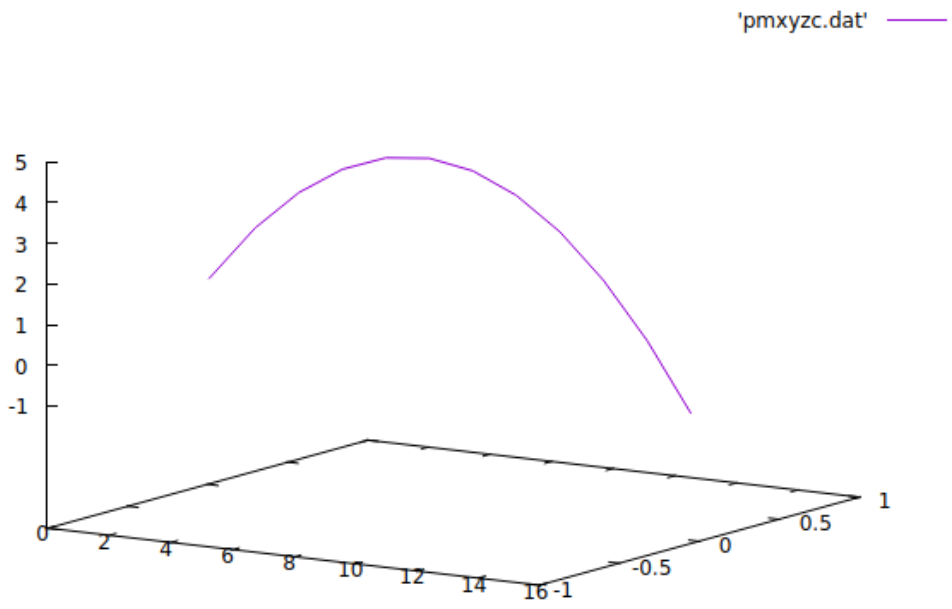
```
program projectilemotionc
implicit none

real(8):: pi, theta, g, h, Vox, Voz, t, Vo, Roz, x, zmax, m,  k, vold
real:: r(3), v(3), a(3)

pi=4*atan(1.0)
g=9.8
h=0.1
m=0.145
k=0.00132
write(*,*) 'the initial angle is'
read(*,*) theta
write(*,*) 'the initial velocity is'
read(*,*) Vo

theta= (theta*pi)/180
! 1=x; 2=y; 3=z
Vox= Vo*cos(theta)
Voz=Vo*sin(theta)
Roz= 1.0
!initial z-position=1.0m
r=0.0
v=0.0
a=0.0
t=0.01

open(unit=100, file='pmxvtc.dat')
open(unit=200, file='pmzvtc.dat')
open(unit=300, file='pmvxvtc.dat')
open(unit=400, file='pmvzvtc.dat')
open(unit=500, file='pmaxvtc.dat')
open(unit=600, file='pmazvtc.dat')
open(unit=700, file='pmxzc.dat')
!splot
open(unit=800, file='pmxyzc.dat')

```

```
37
38 do while (r(3)>=0.0)
39
40 vold=v(1)
41 v(1)= Vox + a(1)*t
42 r(1)= ((v(1)+vold)/2.0)*(t)
43 a(1)= a(1)!-(k*(v(1)**2))
44
45 vold=v(3)
46 v(3)= Voz + a(3)*t
47 r(3)= Roz+((vold+v(3))/2.0)*(t)+(0.5*(a(3)*t*t))
48 a(3)= (-g)!-(k*(v(3)**2))
49
50 write(100,*) t, r(1)
51 write(200,*) t, r(3)
52 write(300,*) t, v(1)
53 write(400,*) t, v(3)
54 write(500,*) t, a(1)
55 write(600,*) t, a(3)
56 write(700,*) r(1), r(2)
57 !splot
58 write(800,*) r(1), r(2), r(3)
59
60 t=t+h
61
62 enddo
63
64 zmax= Roz+((Voz)*(t/2))+(0.5*(a(3)*((t/2)**2)))
65
66 write(*,*) 'The range of the projectile is', r(1)
67 !write(*,*) r(1)
68 write(*,*) 'The time of flight of the projectile is', t
69 !write(*,*) t
70 write(*,*) 'The maximum height reached by the projectile is', zmax
71 !write(*,*) zmax
72
73 end program projectilemotionc
```

Where I initialize the state of the object (position, velocity, and acceleration) each as 3-dimensional arrays in order to represent the motion in an xyz plane. The x plane is represented by the first column of each array, the y in the second, and z in the third. This allows us to treat each component of the motion separately while simultaneously allowing for a more complete graphical representation. In this case, there is no y-component of the objects motion, however, accounting for y-values in our array allows us to make a 3D plot of the trajectory as well as 2D plots of the position, velocity, and acceleration over time shown below.



2D Projectile Plot



3D Projectile Plot

SHM:

Simple harmonic motion is defined by oscillation around an equilibrium where the restorative force is proportional to the displacement from equilibrium. Here, we examined the case of a mass oscillating on a spring where we were tasked with simulating the evolution of the masses position, velocity, and acceleration as well as kinetic, potential, and total energies as a function of time. First, we'll consider the spring scenario. If we take the position to be described by the oscillatory function,

$$x(t) = C_1\, cos(\omega t) + C_2\, sin(\omega t)$$

Where C_1 and C_2 are constants describing the amplitude and $\omega = 2\pi f$, accounting for the frequency of the oscillation. We can derive the velocity as:

$$\dot{x}(t) = -\omega C_1\, sin(\omega t) + \omega C_2\, cos(\omega t)$$

And the acceleration as

$$\ddot{x}(t) = -\omega^2 C_1\, cos(\omega t) - \omega^2 C_2\, sin(\omega t)$$

Here, the acceleration can be simplified as

$$\ddot{x}(t) = -\omega^2\big(x(t)\big)$$

Which, when coupled with Hooke's Law in the absence of external forces, yields

$$F_{sp} = -kx = ma = -\omega^2\big(x(t)\big)m$$

Where k is the spring constant. This allows us to solve for ω as :

$$\omega = \sqrt{\frac{k}{m}}$$

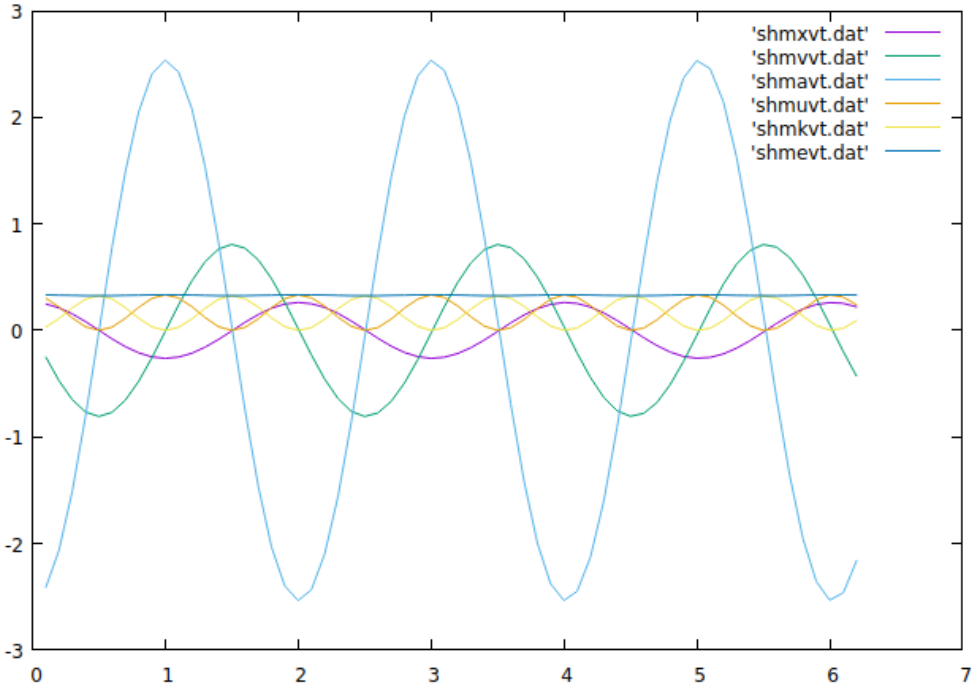
Setting t = 0 in our first two equations, allows us to determine that $C_1 = x_0$ and that $C_2\omega = v_0$ and therefore $C_2 = \frac{v_0}{\omega}$, allowing us to conclude that

$$x(t) = x_0 cos(\omega t) + \frac{v_0}{\omega} sin(\omega t)$$

By now, we have reduced ω to known constants, simplified our C_1 and C_2 terms to allow for them to be evaluated throughout our simulation, and simplified a function for the object's acceleration. Our derived expressions for the objects position and velocity won't be needed for the simulation to take place and instead, values will be determined using Euler's 1st and 2nd approximation methods. Assuming the mass and spring constant to equal one, and determining the number of indexes based on the period of the motion, the program is as follows:

```
program shmmidpt
implicit none

integer(4):: i, n
real(8), dimension(:), allocatable:: x,v,a,k,u,e,t
real(8):: m, c, tau, pi, h, w

open(unit=100, file= 'shmxvt.dat')
open(unit=200, file= 'shmvvt.dat')
open(unit=300, file= 'shmavt.dat')
open(unit=400, file= 'shmuvt.dat')
open(unit=500, file= 'shmkvt.dat')
open(unit=600, file= 'shmevt.dat')

m=1.0
 c=1.0
a=1.0
w=sqrt(c/m)
h=0.1
pi=4*atan(1.0)
tau=2*pi*sqrt(m/c)
!nh=number of timestops, number of values -1 (initial)
n=floor(10*tau/h)

allocate (x(n), v(n), a(n), k(n), u(n), e(n), t(n))

t(1)=0.0
x(1)=1.0
v(1)=0.0
u(1)=(1/2)*c*x(1)**2
k(1)=(1/2)*m*v(1)**2
e(1)=k(1)+u(1)

do i= 1,(n-1)

!Program x, ... arrays
!(i) refers to array values programmed from do i= ...

a(i+1)=-(w**2)*x(i)
v(i+1)=v(i)+h*a(i)
x(i+1)=x(i)+h*((v(i)+v(i+1))/2)
t(i+1)=t(i)+h

!write (unit,*) x(i+1) ...

write(100,*) t(i), x(i)
write(200,*) t(i), v(i)
write(300,*) t(i), a(i)

u(i+1)=(0.5)*c*x(i+1)**2
k(i+1)=(0.5)*m*v(i+1)**2
e(i+1)=k(i+1)+u(i+1)

write(400,*) t(i), u(i)
write(500,*) t(i), k(i)
write(600,*) t(i), e(i)


enddo

endprogram shmmidpt
```

Where here, we use allocatable arrays to list and store values based on their index rather than dynamically updating vector values like we did with projectile motion. This approach is useful for lists or arrays whose size may need to be updated with the simulation rather than preset. Values for the mass's energy are updated using formulas, $KE = \frac{1}{2}mv^2$ and $PE = \frac{1}{2}kx^2$, rather than using an approximation method, and are displayed in the plot:



Position, Velocity, Acceleration, KE, PE, and Total Energy versus Time

Yielding the oscillatory pattern that we would expect to see based on the background presented, while conserving the total mechanical energy of the system.

Orbital Motion:

As somewhat of a concluding synthesis of the problems presented previously, we were assigned to solve the orbital motion problem. Our goal was to represent the trajectory of a simplified, circular orbit of a point mass. We simplify the mass of the satellite to equal one, whereas the big mass is equal to $\frac{4\pi^2}{G}$, allowing us to work easily in astronomical units.

This simulation was made using the Runge-Kutta 4th order approximation method which takes a weighted average of the slopes at four points, balancing computational speed and accuracy, yielding a very low relative error.

RK4 is stated as:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = f(x_n, y_n) \quad ; \quad k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \quad ; \quad k_4 = f(x_n + h, y_n + hk_3)$$

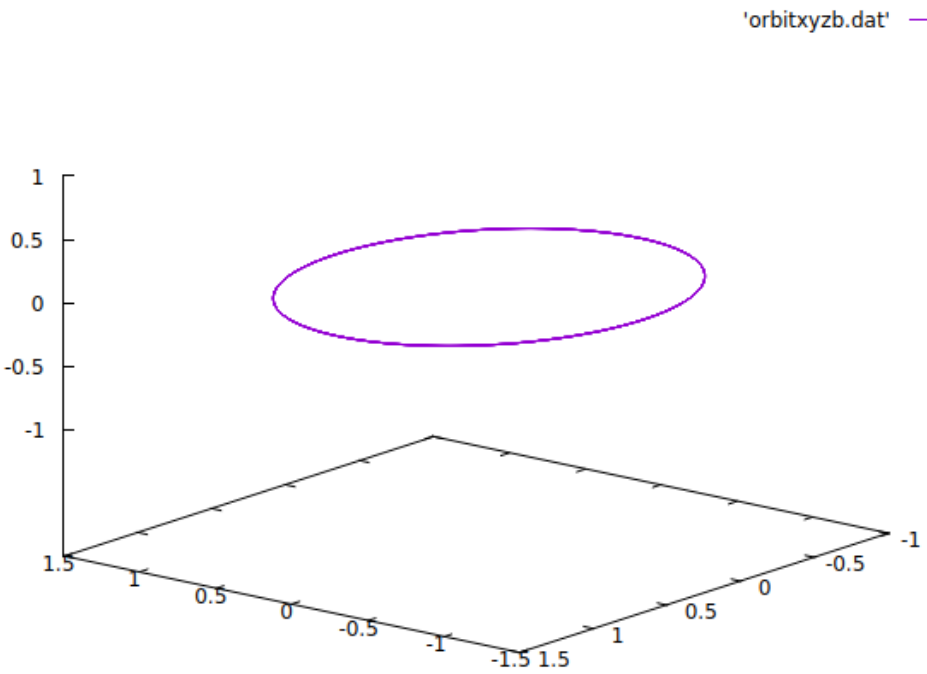
The formulation depicts how RK4 essentially evaluates the slopes halfway through a given x interval, hence the $\frac{h}{2}$ terms, yielding more powerful estimates that are then weighted accordingly in the final evaluation of the function. Due to the complexity required for RK4, here we will introduce the use of arrays to hold values for k as well as “state” and “temp” vectors for efficiency. The temp vector will be evaluated based on the state (position and velocity), and will hold the temporary, half-incremented values required for the approximation technique.

The initial mass and unit conditions described previously, as well as given initial position as r = 1.0 AU **i** and velocity as v = 1.0 AU/yr **i** + 2π AU/yr **j** allow the simulation to be written as:

```
program orbitb
implicit none

real(8):: state(6), r(3), v(3), a(3)
real(8):: k1(6), k2(6), k3(6), k4(6), temp(6)
real(8):: K, P, E, x, y, z, t, h, pi, GM, m1
integer(4):: i

open(unit=100, file='orbitkvtb.dat')
open(unit=200, file='orbitpvtb.dat')
open(unit=300, file='orbitevtb.dat')
open(unit=400, file='orbitxyzb.dat')

pi= 4*atan(1.0)
h= 0.01
GM= 4*(pi**2.0)
m1= 1.0
r=0
r(1)=1.0
v=0
v(1)=1.0
v(2)=2.0*pi
state(1:3)= r
state(4:6)= v
do i=1,1000

!k1= d/dx(r0,v0)

a= -GM/((norm2(r))**3)*r
k1(1:3)= v
k1(4:6)= a

!k2=d/dx[r0,v0]+h/2*k
temp=state+(0.5*h*k1)
a= -GM/((norm2(temp(1:3)))**3)*temp(1:3)
```

```
k2(1:3)= temp(4:6)
k2(4:6)= a

!k3=d/dx[r0,v0]+h/2*k
temp=state+(0.5*h*k2)
a= -GM/((norm2(temp(1:3)))**3)*temp(1:3)
k3(1:3)= temp(4:6)
k3(4:6)= a

!k4=d/dx[r0,v0]+h*k
temp=state+(h*k3)
a= -GM/((norm2(temp(1:3)))**3)*temp(1:3)
k4(1:3)= temp(4:6)
k4(4:6)= a

!r(1:3)= r+((h/6)*(k1+(2*k2)+(2*k3)+k4))
state=state+((h/6)*(k1+(2*k2)+(2*k3)+k4))

r=state(1:3)
v=state(4:6)

K=(0.5)*m1*((norm2(v))**2)
P=-(GM*m1)/(norm2(r))
E=K+P
a= -GM/((norm2(r))**3)*r
write(100,*) t, K
write(200,*) t, P
write(300,*) t, E
write(400,*) r(1),r(2),r(3)

t=t+h

enddo

endprogram orbitb
```

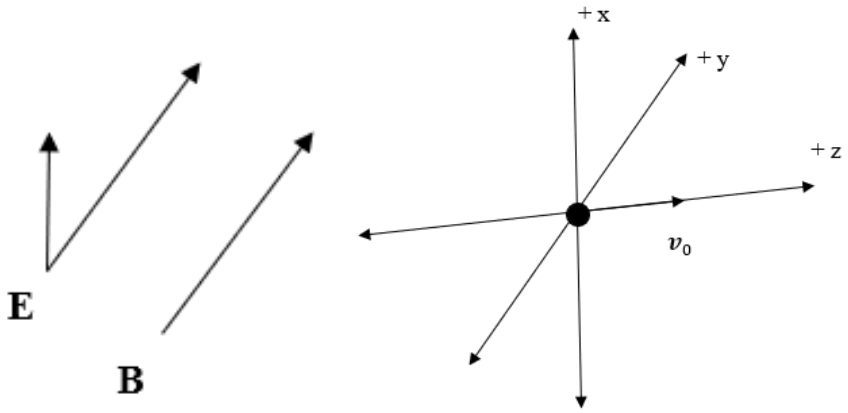
Producing the circular trajectory hypothesized:



3D Orbit Trajectory

Lorentz Force:

As a fitting concluding assignment, we were introduced to concepts of electromagnetism via the Lorentz Force which describes the sum of the forces on a test charge that's moving through both an electric and magnetic field. We aimed to represent the 3D trajectory of one such particle, moving with an initial speed of **v** = 1.0 m/s **k** through an fields of strength **E** = 0.01 N/C **j** + 0.1 N/C **k** and **B** = 0.1 N/C **k**. In our case, we simplified the mass and charge to be equal to one.



The Lorentz force equation tell us that:

$$F_L = q(E + v \times B) \Rightarrow a_L = \frac{q(E + v \times B)}{m}$$

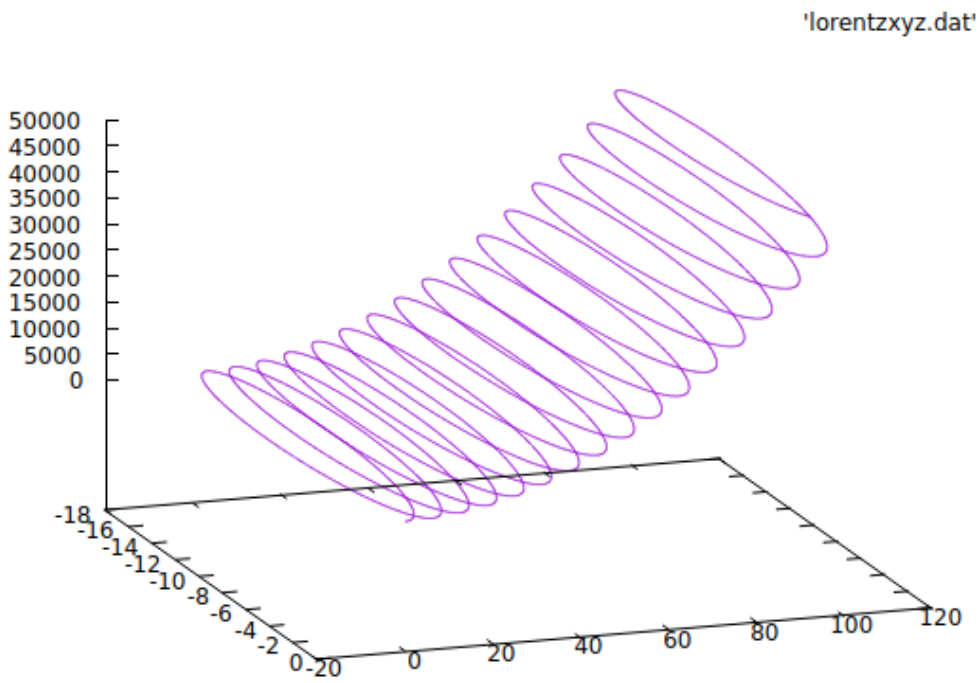
Which, as we've previously shown, when coupled with the initial conditions of the system is sufficient to describe and evolve the particle's motion with time. Here, we apply the RK4 method, representing the particle in three dimensions, utilizing allocable arrays as previously shown. The program is as follows:

```
program lorentz
implicit none

real(8):: state(6), r(3), v(3), a(3), Fe(3), Bv(3)
real(8):: E(3), B(3), k1(6), k2(6), k3(6), k4(6), temp(6)
real(8):: x, y, z, t, h, pi, m, q
integer(4):: i

open(unit=100, file='lorentzxyz.dat')

pi= 4*atan(1.0)
h= 0.01
m= 1.0
q=1.0
r=0
v=0
v(1)=1.0
E=0
E(2)=0.01
E(3)=0.1
B=0
B(3)=0.1
a(1)= (q/m)*(E(1)+((v(2)*B(3))-(v(3)*B(2))))
a(2)= (q/m)*(E(2)-((v(1)*B(3))-(v(3)*B(1))))
a(3)= (q/m)*(E(3)+((v(1)*B(2))-(v(2)*B(1))))

do i=1,100000

!k1= d/dx(r0,v0)

state(1:3)= r
state(4:6)= v
a(1)= (q/m)*(E(1)+((v(2)*B(3))-(v(3)*B(2))))
a(2)= (q/m)*(E(2)-((v(1)*B(3))-(v(3)*B(1))))
a(3)= (q/m)*(E(3)+((v(1)*B(2))-(v(2)*B(1))))
```

```
k1(1:3)= v
k1(4:6)= a

!k2=d/dx[r0,v0]+h/2*k
temp=state+(0.5*h*k1)
r= temp(1:3)
v= temp(4:6)
a(1)= (q/m)*(E(1)+((v(2)*B(3))-(v(3)*B(2))))
a(2)= (q/m)*(E(2)-((v(1)*B(3))-(v(3)*B(1))))
a(3)= (q/m)*(E(3)+((v(1)*B(2))-(v(2)*B(1))))
k2(1:3)= temp(4:6)
k2(4:6)= a

!k3=d/dx[r0,v0]+h/2*k
temp=state+(0.5*h*k2)
r= temp(1:3)
v= temp(4:6)
a(1)= (q/m)*(E(1)+((v(2)*B(3))-(v(3)*B(2))))
a(2)= (q/m)*(E(2)-((v(1)*B(3))-(v(3)*B(1))))
a(3)= (q/m)*(E(3)+((v(1)*B(2))-(v(2)*B(1))))
k3(1:3)= temp(4:6)
k3(4:6)= a

!k4=d/dx[r0,v0]+h*k
temp=state+(h*k3)
r= temp(1:3)
v= temp(4:6)
a(1)= (q/m)*(E(1)+((v(2)*B(3))-(v(3)*B(2))))
a(2)= (q/m)*(E(2)-((v(1)*B(3))-(v(3)*B(1))))
a(3)= (q/m)*(E(3)+((v(1)*B(2))-(v(2)*B(1))))
k4(1:3)= temp(4:6)
k4(4:6)= a

!RK4: y1=y0+(h/6)*(k1+(2*k2)+(2*k3)+k4)
state=state+((h/6)*(k1+(2*k2)+(2*k3)+k4))
```

```
state=state+((h/6)*(k1+(2*k2)+(2*k3)+k4))

r(1:3)=state(1:3)
v(1:3)=state(4:6)

write(100,*) r(1),r(2),r(3)

t=t+h

enddo

end program lorentz
```

Producing the satisfying 3D helical trajectory below.



3D Lorentz particle Trajectory

INSTITUTION CREDIT -Top-

Term: Summer Session 2023

Academic Standing:

Subject	Course	Level	Title	Grade	Credit Hours	Quality Points	R
PHYS	2500	UG	Intro Computatnl Sci and Engr	A	3.000	12.00	
PHYS	2501	UG	Intro Computr Sci and Engr Lab	A	1.000	4.00	

Term Totals (Undergraduate)

	Attempt Hours	Passed Hours	Earned Hours	GPA Hours	Quality Points	GPA
Current Term:	4.000	4.000	4.000	4.000	16.00	4.0000
Cumulative:	4.000	4.000	4.000	4.000	16.00	4.0000

Unofficial Transcript

TRANSCRIPT TOTALS (UNDERGRADUATE) -Top-

	Attempt Hours	Passed Hours	Earned Hours	GPA Hours	Quality Points	GPA
Total Institution:	4.000	4.000	4.000	4.000	16.00	4.0000
Total Transfer:	0.000	0.000	0.000	0.000	0.00	0.0000
Overall:	4.000	4.000	4.000	4.000	16.00	4.0000
	Attempt Hours	**Passed Hours**	**Earned Hours**	**GPA Hours**	**Quality Points**	**GPA**
Institution Combined:	4.000	4.000	4.000	4.000	16.00	4.0000
Transfer Combined:	0.000	0.000	0.000	0.000	0.00	0.0000
Overall Combined:	4.000	4.000	4.000	4.000	16.00	4.0000

Unofficial Transcript

2023 GSCP Unofficial Transcript

Access to the material presented in this portfolio are made available to the public via my Github repository, found at https://github.com/ojordanc2005/GSCP.git .