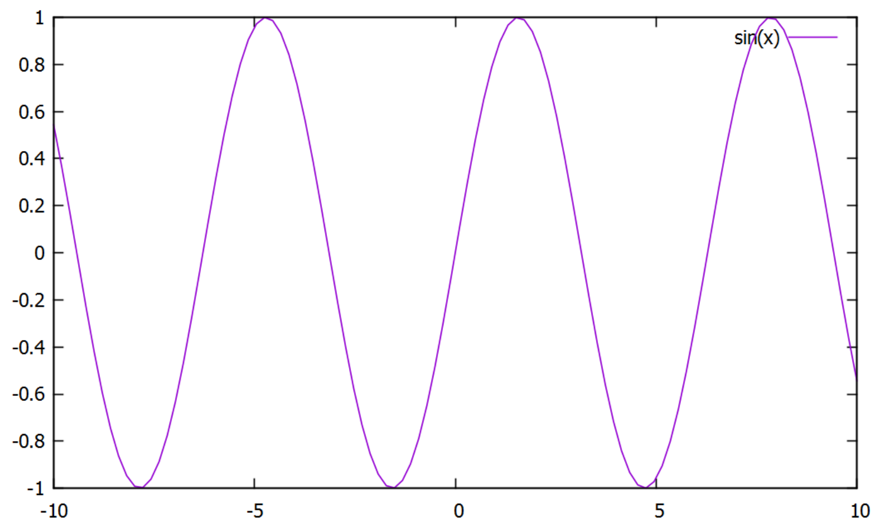
Yanes Garcia

Mathematics 4670

Gnuplot: The Euler, Heun, Midpoint, and Adams Bashforth Methods

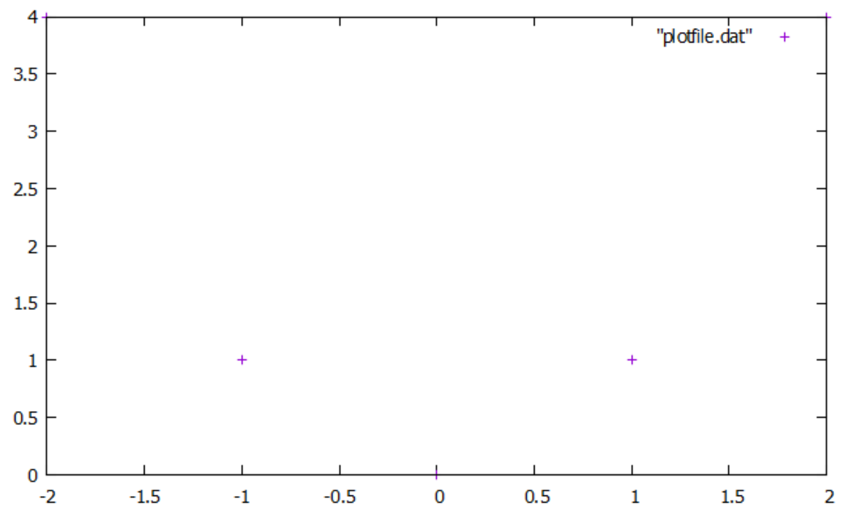
1. For the first question, I downloaded and installed gnuplot. Below, I include the graph of sin(x);



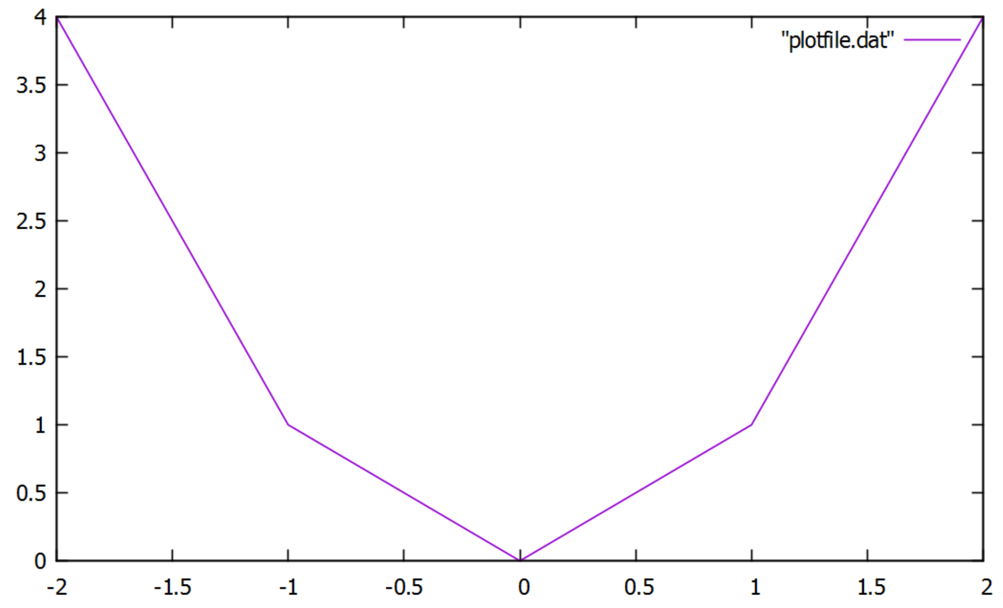
Also, I made up a small data file. The points are:

```
-2   4
-1   1
 0   0
 2   4
```

This file produces the following graph:



Below, I have the same graphic but with lines.



2. In this question, the three numerical methods that I chose were the Euler, Heun and the Midpoint method. I am going to use these three numerical methods to solve

$$\frac{dy}{dt} = te^{3t} - 2y$$

for 0 <= t<=2. For each method, I obtain 100 data points.

The code for Euler's method is the following one:

```
!main Euler
program testeuler
```

```
implicit none
integer :: nsteps, i
integer:: neq
double precision, allocatable, dimension(:, :) :: archive
double precision, allocatable, dimension(:) ::  yin, yout, t, f1
double precision :: a, b, tin, dt
double precision :: computef
real::f
neq=1
nsteps=100
print*, 'nsteps= ', nsteps
allocate(t(0: nsteps), archive(neq, 0:nsteps), yin(neq), yout(neq))

a = 0.0d0
b = 2.0d0
dt = (b-a)/dble(nsteps)

do i=0, nsteps
t(i) = a + dble(i)* dt
end do

archive (:,0) = (/0.0d0/)
yin = archive(:,0)
do i=0, nsteps-1
        tin=t(i)
        yin= archive(1,i)
        f= computef(tin, yin)
        yout = yin + dt*f
      archive(:, i+1) =yout
end do
print*, 'Creating " euler" '
open(unit=8, file= ' euler', status='replace')
do i=0, nsteps
        write(8, *) t(i), archive(:, i)
```

```
end do
close(8)
deallocate(yin, yout,  t, f1, archive)
stop
end

double precision function computef(t, y) result(f1)
implicit none
double precision:: t, y
real :: f1
f1= t*EXP(3*t)-2*y
return
end
```

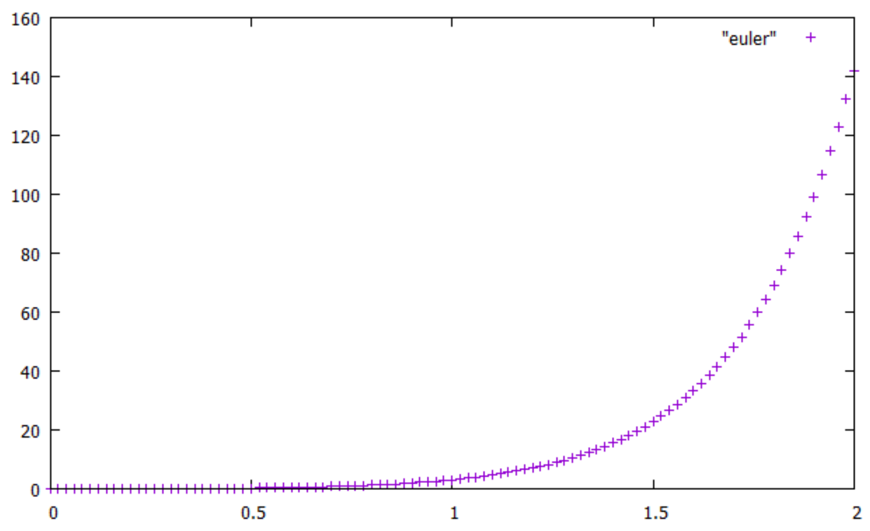
I'll show only the relevant part of the output.

```
0.00000000000            0.00000000000
2.000000000000E-02         0.00000000000
4.000000000000E-02        4.247346147895E-04
6.000000000000E-02        1.309742741287E-03
8.000000000000E-02        2.694013826549E-03
```

And so on and so forth. I am going to show the last numbers on the sequence which is created when I solve the above equation.

```
1.92000000000            106.667783260
1.94000000000            114.587247981
1.96000000000            123.078273982
1.98000000000            132.181265315
2.00000000000            141.939437923
```

Below, we have the graph for the Euler's method is



The code for Midpoint method is the following:

```
!main Midpoint
program testmidpoint
implicit none
integer :: nsteps, i
integer:: neq
double precision, allocatable, dimension(:, :) :: archive
double precision, allocatable, dimension(:) ::  yin, yout, t
double precision :: a, b , tin, dt
double precision :: computef
real::f, f1

neq=1
nsteps=100
print*, 'nsteps= ', nsteps
allocate(t(0: nsteps), archive(neq, 0:nsteps), yin(neq), yout(neq))
a = 0.0d0
b = 2.0d0
dt = (b-a)/dble(nsteps)

do i=0, nsteps
 t(i) = a + dble(i)* dt
end do
```

```
archive (:,0) = (/0.0d0/)
yin = archive(:,0)

do i=0, nsteps-1
    tin=t(i)
    yin= archive(1,i)
    f= computef(tin, yin)
    f1=computef(tin+(dt/2),yin+(dt/2)*f)
    yout = yin + dt*f1
       archive(:, i+1) =yout
end do

print*, 'Creating "midpoint" '
open(unit=8, file= 'midpoint', status='replace')
do i=0, nsteps
write(8, *) t(i), archive(:, i)
end do
close(8)
deallocate(yin, yout,  t, archive)
stop
end

double precision function computef(t, y) result(f1)
implicit none
double precision:: t, y
real :: f1
f1= t*EXP(3*t)-2*y
return
end
```

I'll show only the relevant part of the output.

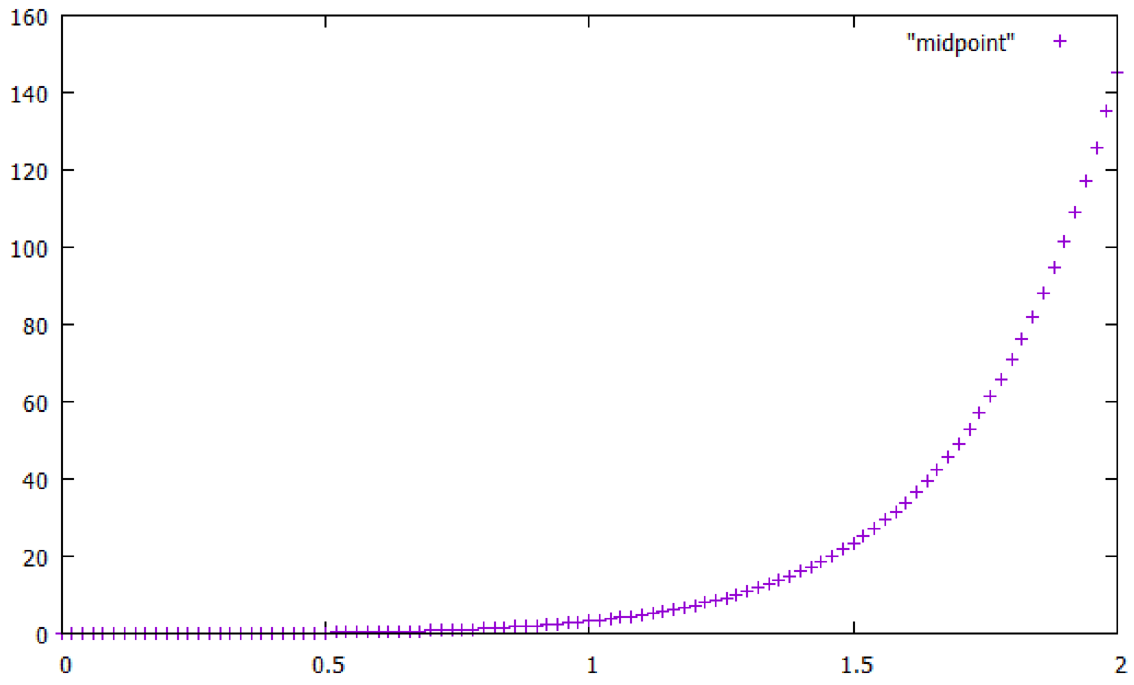
```
0.00000000000          0.00000000000
2.000000000000E-02       2.060909010470E-04
```

```
4.000000000000E-02        8.460220135748E-04
6.000000000000E-02        1.956652272493E-03
8.000000000000E-02        3.578367549926E-03
0.100000000000        5.755351651460E-03
```

And so on and so forth. I am going to show the last numbers on the sequence that is formed when I solve the above equation.

```
1.94000000000        117.278067478
1.96000000000        125.961417697
1.98000000000        135.270229953
2.00000000000        145.248654025
```

Below, we have the graph for the Midpoint method is



The code for Heun's method is the following:

```
!main Heun
program testheun
implicit none
integer :: nsteps, i
integer:: neq
double precision, allocatable, dimension(:, :) :: archive
double precision, allocatable, dimension(:) ::  yin, yout, t
double precision :: tin, dt, a, b
double precision :: computef
real::f,f1,f2
neq=1
nsteps=100
print*, 'nsteps= ', nsteps
allocate(t(0: nsteps), archive(neq, 0:nsteps), yin(neq), yout(neq))
a = 0.0d0
b = 2.0d0
dt = (b-a)/dble(nsteps)

do i=0, nsteps
   t(i) = a + dble(i)* dt
end do
archive (:,0) = (/0.0d0/)
yin = archive(:,0)
do i=0, nsteps-1
   tin=t(i)
   yin= archive(1,i)
   f= computef(tin, yin)
   f2=computef(tin+(dt/3),yin+(dt/3)*f)
   f1=computef(tin+(2*dt/3),yin+(2*dt/3)*f2)
   yout = yin + (dt/4)*(f+3*f1)
     archive(:, i+1) =yout
end do
print*, 'Creating " heun " '
open(unit=8, file= ' heun', status='replace')
do i=0, nsteps
write(8, *) t(i), archive(:, i)
end do
close(8)
deallocate(yin, yout,  t, archive)
stop
end
```

```
double precision function computef(t, y) result(f1)
implicit none
double precision:: t, y
real :: f1
f1= t*EXP(3*t)-2*y
return
end
```

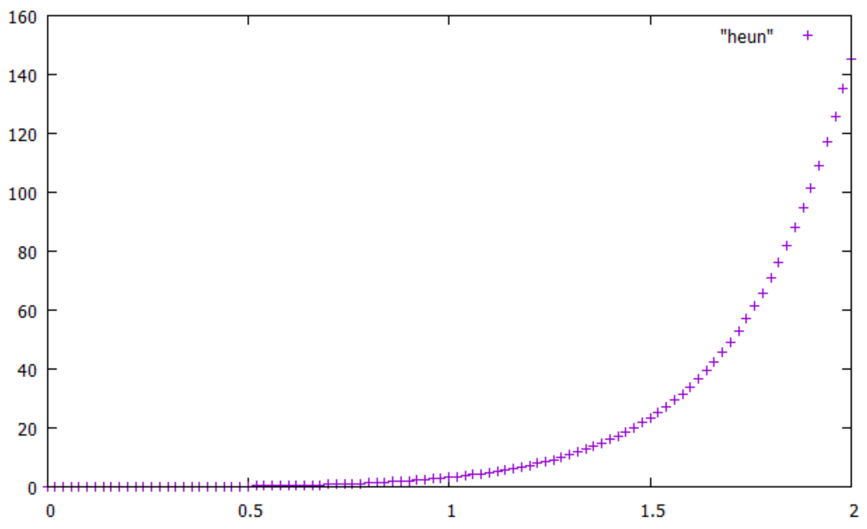
I’ll show only the relevant part of the output.

```
0.00000000000           0.00000000000
2.000000000000E-02        2.054416202009E-04
4.000000000000E-02        8.447134122252E-04
```

And so on and so forth. I am going to show the last numbers on the sequence which is created when I solve the above equation.

```
1.90000000000           101.615622096
1.92000000000           109.168482356
1.94000000000           117.266883846
1.96000000000           125.949414516
1.98000000000           135.257348933
2.00000000000           145.234832454
```

Below, we have the graph for the Heun’s method is



3. In this question, we analyze an orbit problem, and for this reason, we are going to see some of the behaviors of Kepler's laws, which are the following ones:

- Planets move around the Sun in ellipses, with the Sun at one focus
- As the planet moves along its path, a line joining the planet to the Sun sweeps out equal areas in equal times.
- For any two planets, the ratio of the squares of their periods of revolution about the Sun is the same as the ratio of the cubes of their mean distances from the Sun.

We are going to use the Euler's method to solve our orbit problem. In this one, the initial x velocity is zero and the initial y velocity is one.

The Euler's method code for solve this orbit problem is the following one:

```
!subroutine Euler method
subroutine euler(neq, tin, tout, nsteps, yin, yout)
implicit none
integer, intent(in) :: neq, nsteps
double precision, intent(in) :: tin, tout, yin(neq)
double precision :: yout(neq), f(neq), dt
integer :: i

dt=(tout - tin) / dble(nsteps)
yout = yin
do i=0, nsteps-1
   call computef(neq, tin+dble(i)*dt, yout, f)
   yout = yout + dt*f
end do
return
end

!extrapolated Euler
subroutine extrapolate(neq, tin, tout, yin, yout)
implicit none
integer, intent(in) :: neq
integer :: i, k, nrich, nsteps
```

```
double precision :: yout(neq), bot
double precision, intent(in) :: tin, tout, yin(neq)
double precision, allocatable, dimension(:,:,:) :: table
nrich=10
allocate (table(0:nrich, 0:nrich, neq))
nsteps=1
do i=0, nrich
   call euler(neq, tin, tout, nsteps, yin, yout)
   table(i,0,:) = yout
   nsteps = nsteps*2
end do
do k=1, nrich
   bot=2.0d0**k - 1.0d0
   do i = k, nrich
           table(i,k,:)= table(i,k-1,:) + (table(i,k-1,:)-table(i-1, k-1,:))/bot
   end do
end do
yout= table(nrich, nrich, :)
return
end

!Adams Bashforth methods
subroutine abinitialzer(neq, nsteps, t, archive)
implicit none
integer, intent(in) :: neq
integer :: i, k, nrich, nsteps
double precision :: yout(neq), bot
double precision :: tin, tout, yin(neq)
double precision, intent(in) :: t(0:nsteps)
double precision :: archive( neq, 0: nsteps)

tin=t(0)
yin= archive(:, 0)
tout = t(1)
```

```
call extrapolate(neq, tin, tout, yin, yout)
archive(:, 1) = yout

tin=t(1)
yin= archive(:, 1)
tout = t(2)
call extrapolate(neq, tin, tout, yin, yout)
archive(:, 2) = yout

tin=t(2)
yin= archive(:, 2)
tout = t(3)
call extrapolate(neq, tin, tout, yin, yout)
archive(:, 3) = yout

tin=t(3)
yin= archive(:, 3)
tout = t(4)
call extrapolate(neq, tin, tout, yin, yout)
archive(:, 4) = yout
return
end

!right hand side function
subroutine computef(neq, t, y, f)
implicit none
integer :: neq
double precision, intent(in) :: t, y(neq)
double precision :: f(neq), d

d=(y(1) ** 2.0d0 + y(2)**2.0d0) ** 1.5d0

f(1) = y(3)
f(2) = y(4)
```

```
f(3) = -y(1)/d
f(4) = -y(2)/d
return
end

!main Euler method 1
program testeuler
implicit none
integer :: neq, nsteps, i
double precision, allocatable, dimension(:) :: yin, yout, t, f1, f2, u
double precision, allocatable, dimension(:,:) :: archive
double precision :: tin, tout, dt, a, b, x, y, xactual, yactual
neq=4
nsteps=10000
allocate( yin(neq), yout(neq), t(0: nsteps), u(neq), f1(neq), f2(neq), archive( neq,
    0:nsteps))

a = 0.0d0
b = 3.0d1
dt = (b-a)/dble(nsteps)

do i=0, nsteps
 t(i) = a + dble(i)*dt
end do

archive (:,0) = (/1.0d0, 0.0d0, 0.0d0, 1.0d0/)
yin = archive(:,0)

do i=0, nsteps-1
    tin=t(i)
    tout=t(i+1)
    yin= archive(:,i)
    call computef(neq, tin, yin, f1)
    yout = yin + (tout-tin)*f1
    archive(:, i+1) = yout
end do
print*, 'Creating "orbit" '
open(unit=8, file= 'orbit', status='replace')

do i=0, nsteps
    write(8, *) archive(1,i), archive(2, i)
end do

close(8)
deallocate(yin, yout, archive, t, f1, f2)
stop
end
```

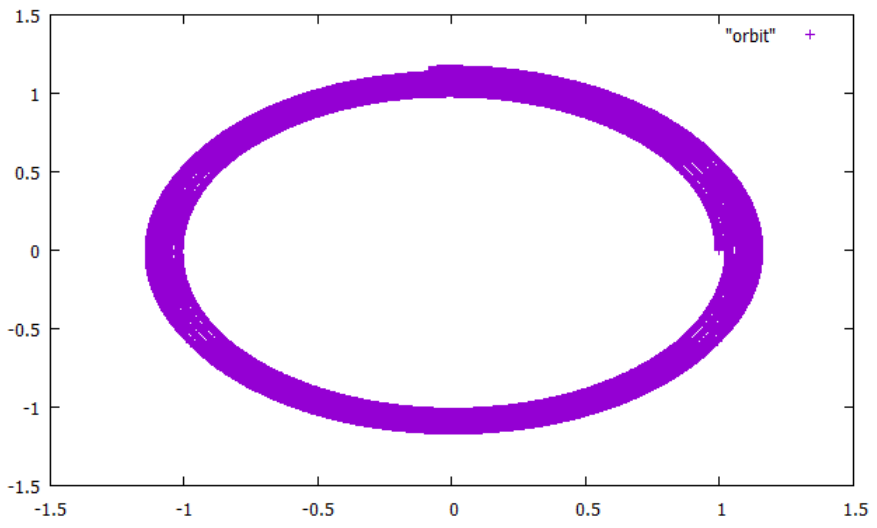
I'll show only the relevant part of the output.

```
 1.00000000000          0.00000000000
 1.00000000000         3.000000000000E-03
0.999991000000          6.000000000000E-03
0.999973000121          8.999973000364E-03
0.999946000567          1.199989200219E-02
```

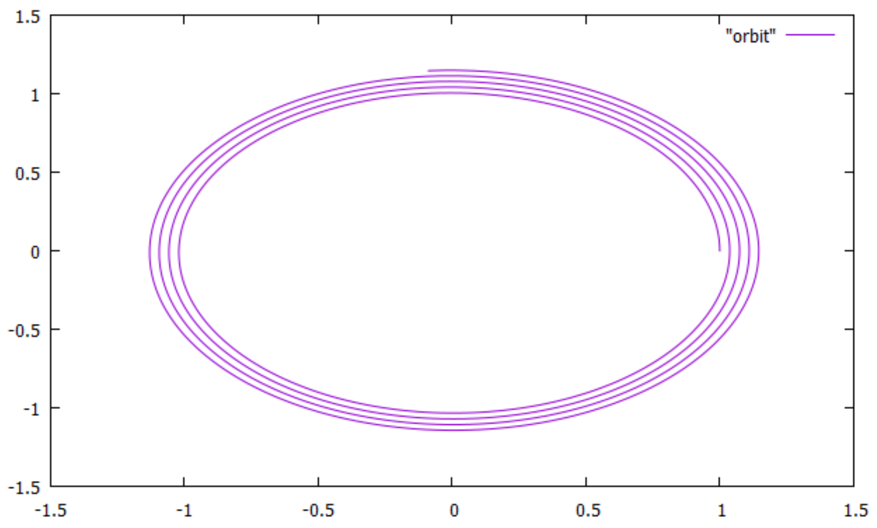
And so on and so forth. I am going to show the last numbers on the sequence.

```
-6.461718861886E-02          1.14720586488
-6.741903294813E-02          1.14705913561
-7.022049392073E-02          1.14690560026
-7.302155492922E-02          1.14674525997
-7.582219936970E-02          1.14657811591
-7.862241064193E-02          1.14640416929
-8.142217214947E-02          1.14622342136
-8.422146729974E-02          1.14603587341
-8.702027950415E-02          1.14584152678
```

The program produces the next graph,



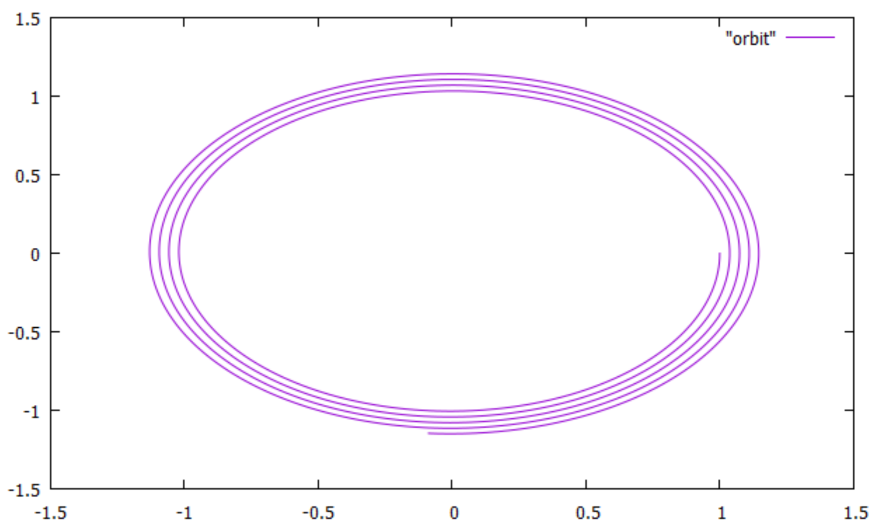
Under, I have the same graphic but with lines. It helps us to see better the orbits.



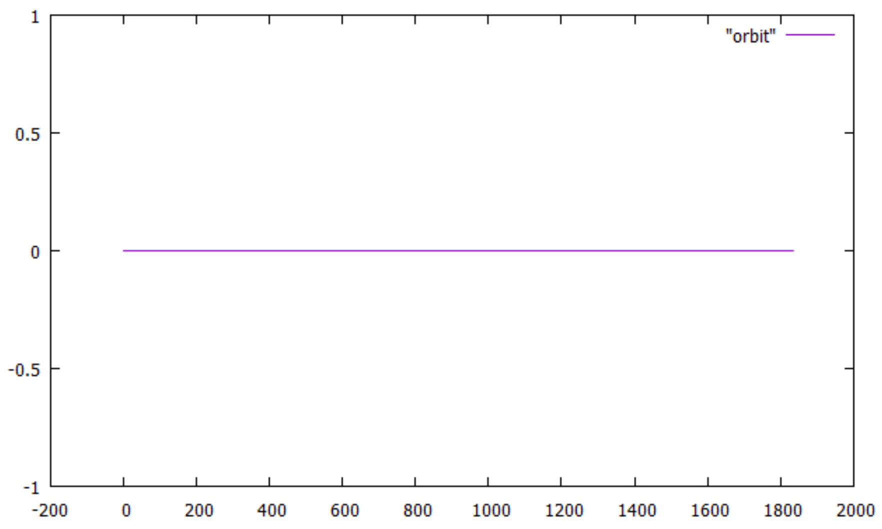
In the above problem, the initial x velocity is zero and the initial y velocity is one. However, we can change the initial y velocity to other values to obtain some interesting results. To change the y velocity, we must change the forth value of the following expression:

archive (:,0) = (/1.0d0, 0.0d0, 0.0d0, 1.0d0/)

I am going to plot all the following files with lines. Using plot “file_name” with lines. When we change the 1.0d0 to -1.0d0, we can see that both of these graphs are similar, but they are not the same because they are inverse. This means that the negative sign only affects to the sense of the orbit.

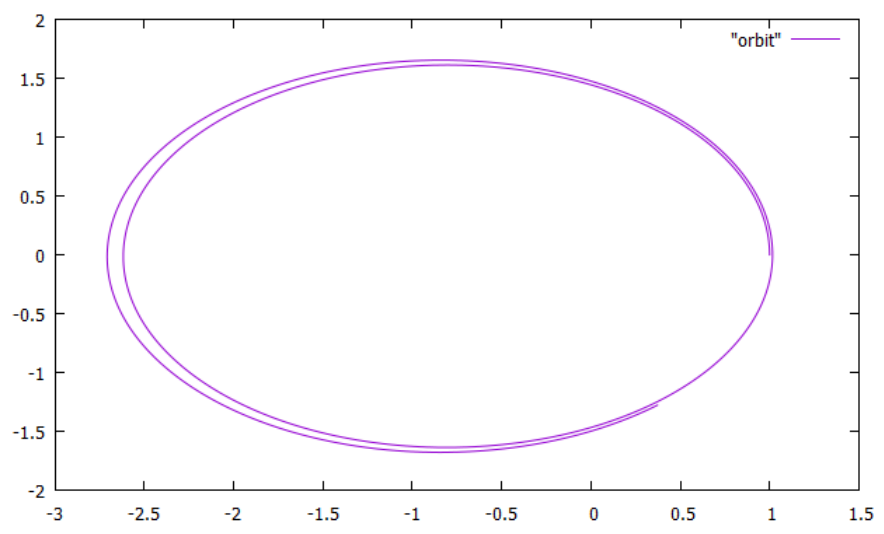
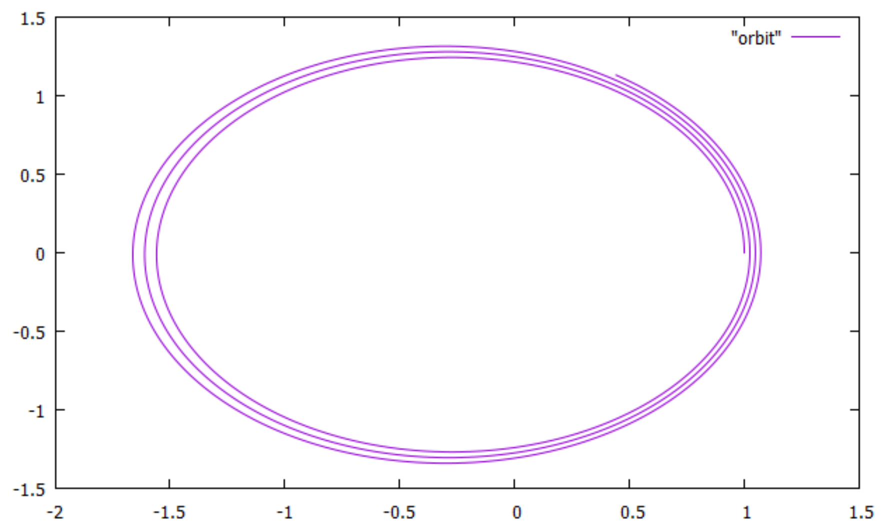
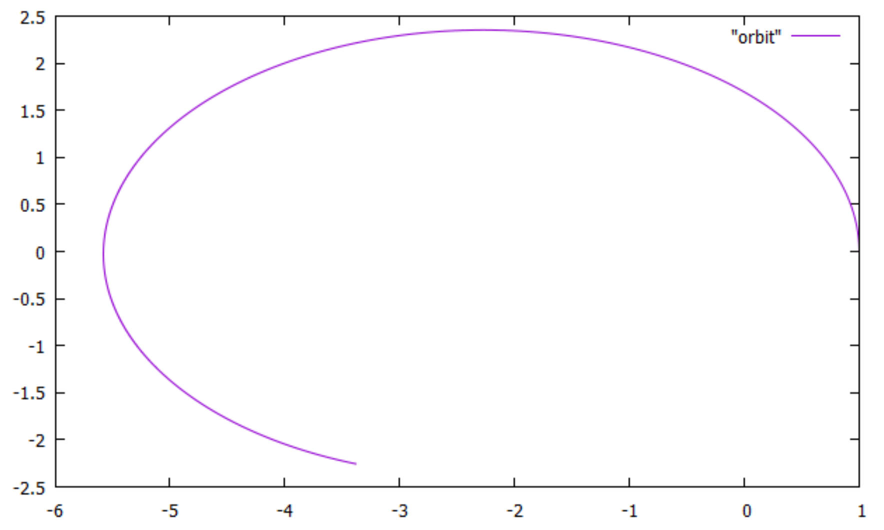


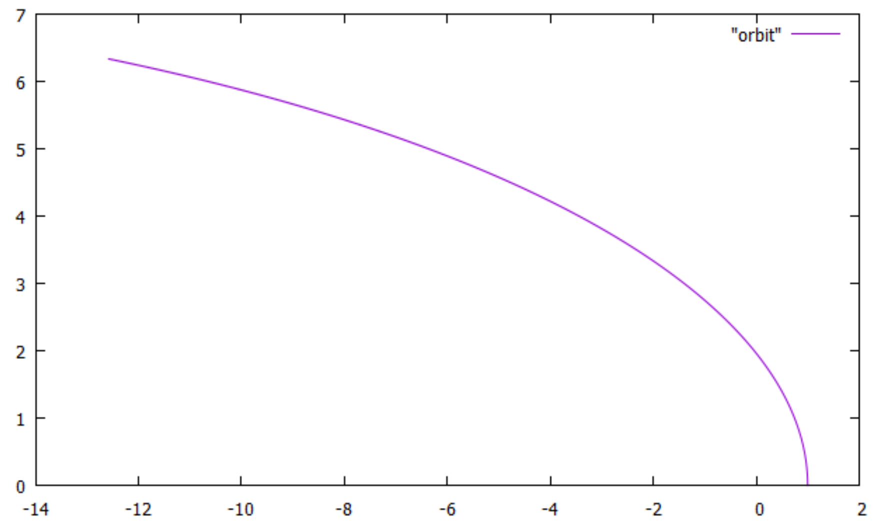
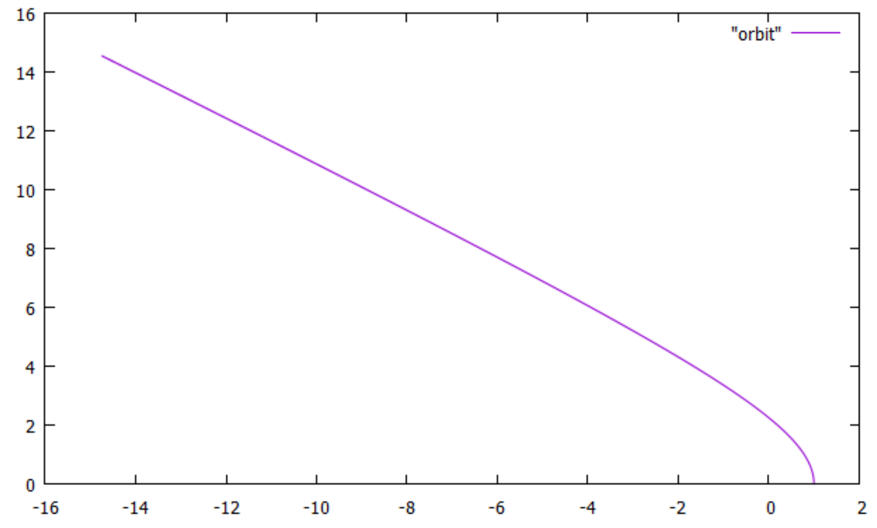
If we set the y velocity to 0, we get a horizontal line. It happens because both the y velocity and the x velocity are zero.



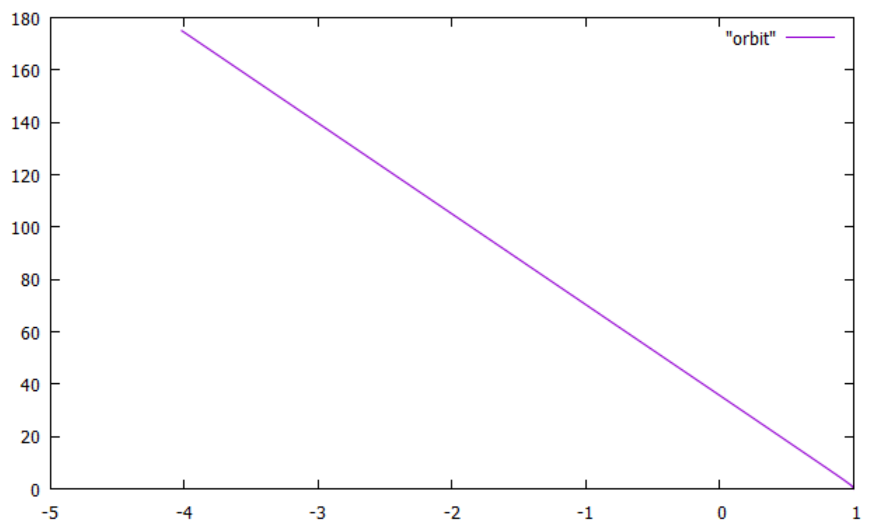
If we increase the y velocity, we get the next graphs:

Y velocity	Graph

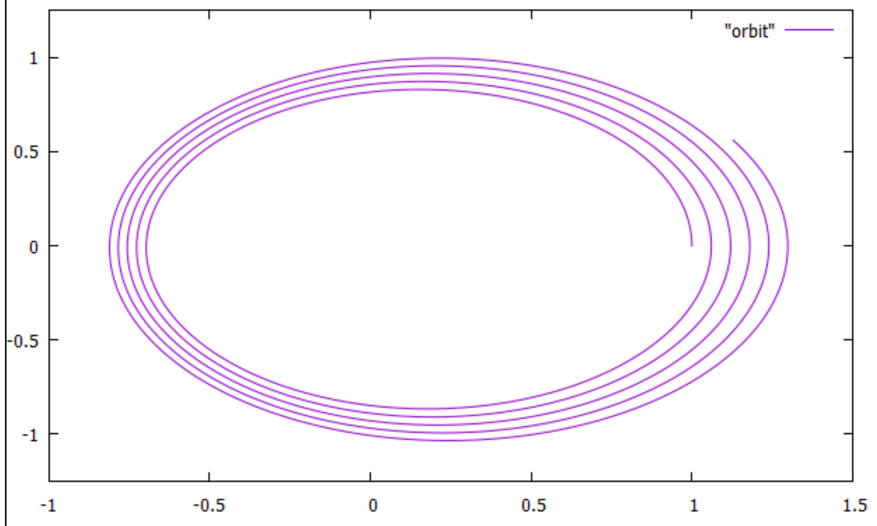
1.1	
1.2	
1.3	

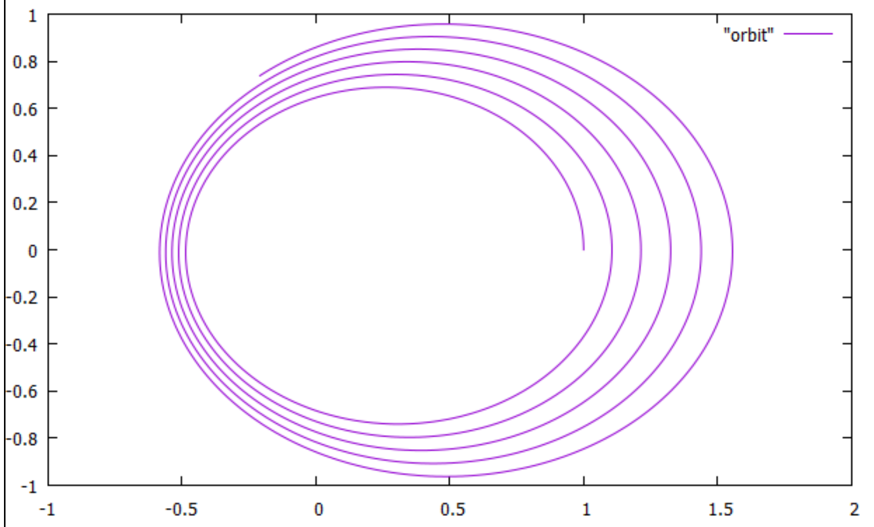
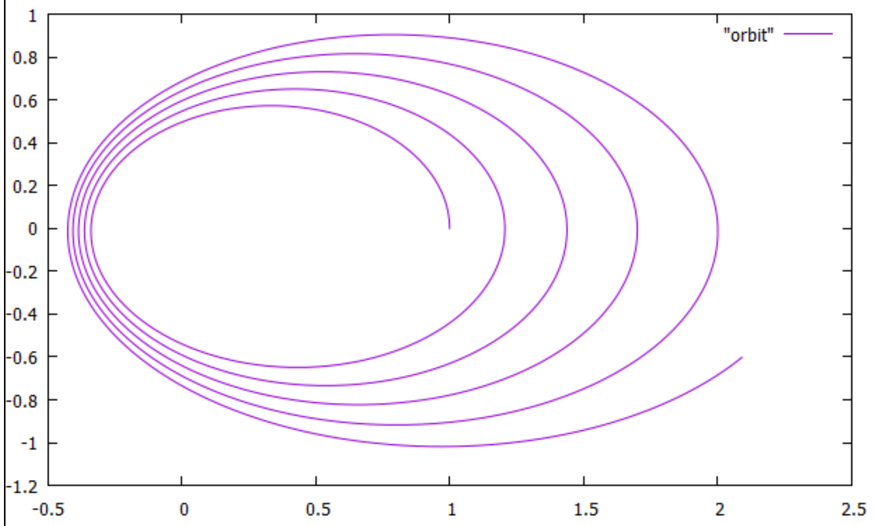
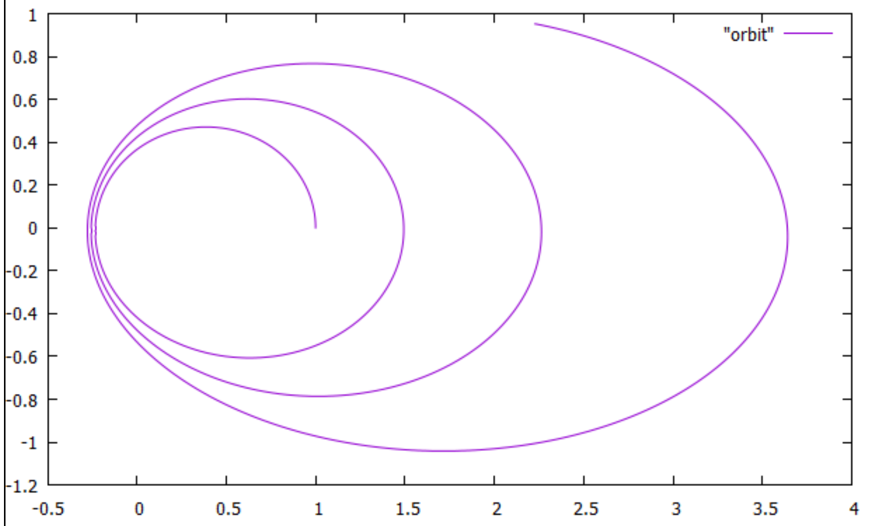
1.4	
1.5	

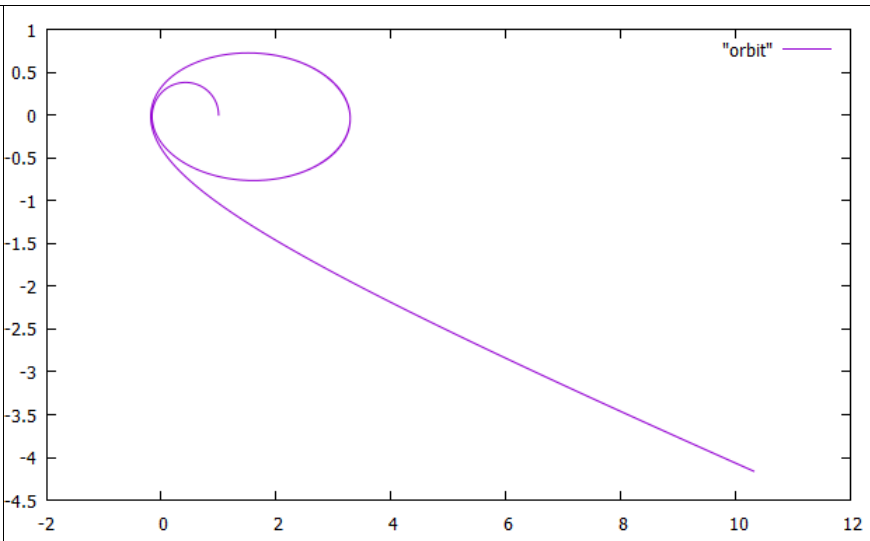
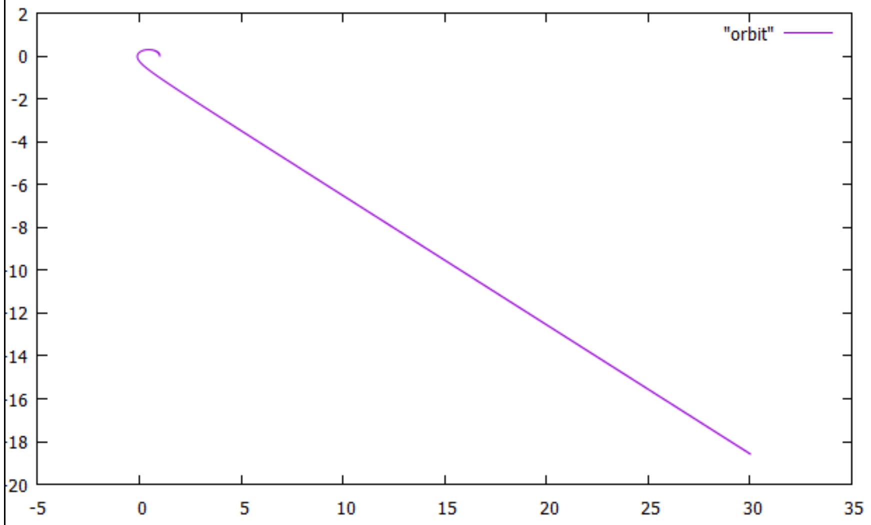
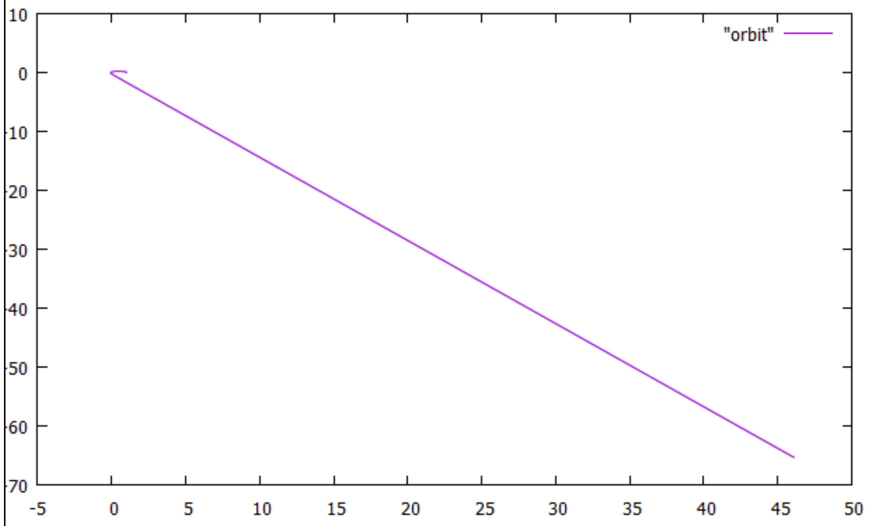
With the above graphs, we can see that when the y velocity is incremented, the orbit loses its form. The incrementing of y velocity produces that the orbit becomes a kind of straight line with negative slope. For example, if we set the y velocity to 6.0, we get the following graph.



If we decrease the y velocity, we get the following graphs:

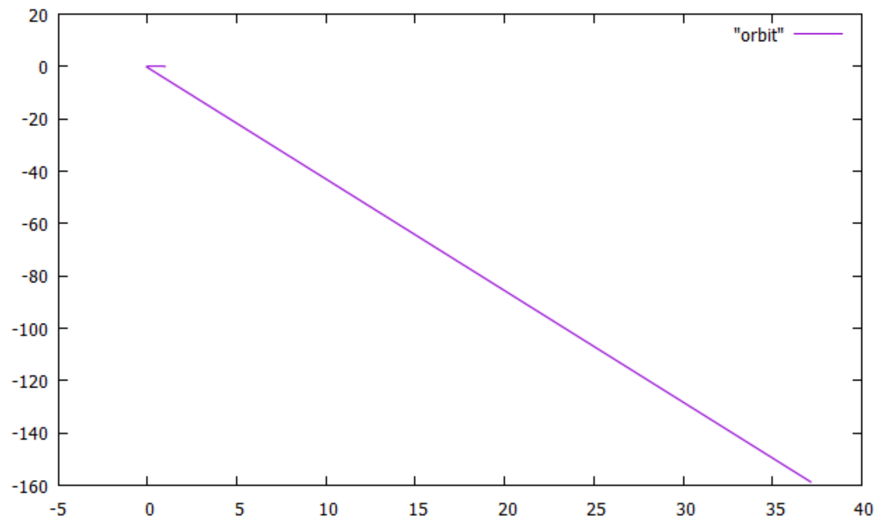
Y velocity	Graph
0.9	

0.8	
0.7	
0.6	

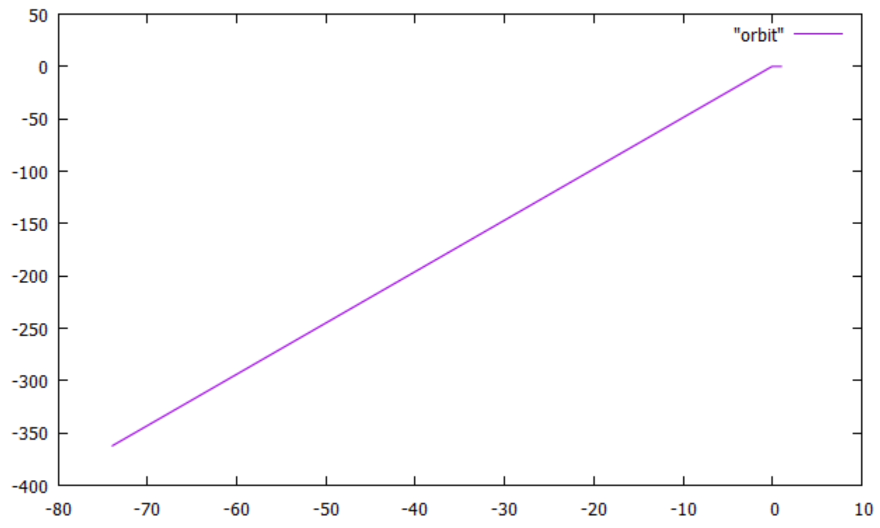
0.5	
0.4	
0.3	

We can appreciate that the lower the y velocity, the more separated are the orbits. The decreasing of y velocity produces that the orbit loses its form. When the y velocity is slow, the orbit becomes a kind of straight line.

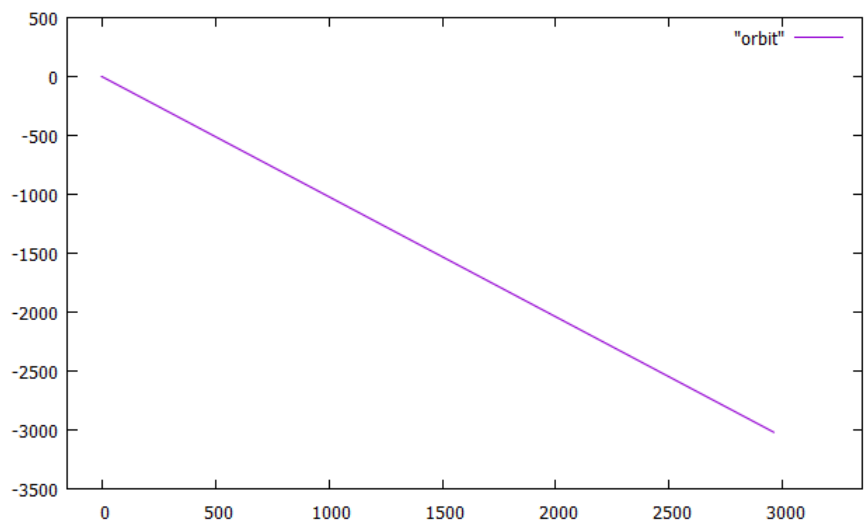
I must mention that when the y velocity is between 0.0 and 0.2, we can find interesting facts. For example, the graph when the y velocity is 0.2 is a straight line with negative slope.



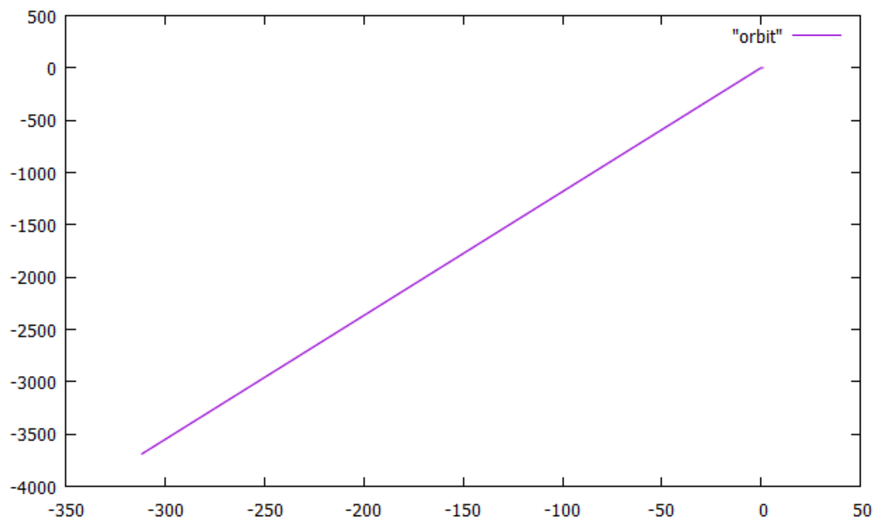
And, when the y velocity is 0.1, the graph is a straight line with positive slope.



This interesting behavior also happens when the y velocity is 0.02 and 0.03. The graph when the y velocity is 0.02 is a straight line with negative slope.



However, when the y velocity is 0.03, the graph is a straight line with positive slope.


"orbit"