1. Work problem 11 of section 5.3 using at least three numerical methods. Create graphs of time (on the horizon axis) versus velocity (on the vertical axis).

Below is the subroutine function that the three numerical methods (testeuler, extrapall, and the predictor corrector) use for (a):

subroutine computef(neq, t, y, f)

implicit none

integer :: neq

double precision :: f(neq)

double precision, intent(in) :: t, y(neq)

f(1) = (y(1) / t - (y(1) / t)\*\*2)

return

end

Below is the subroutine function that the three numerical methods (testeuler, extrapall, and the predictor corrector) use for (b):

subroutine computef(neq, t, y, f)

implicit none

integer :: neq

double precision :: f(neq)

double precision, intent(in) :: t, y(neq)

f(1) = (((1 + y(1)) / t) - (y(1) / t)\*\*2)

return

end

Below is the subroutine function that the three numerical methods (testeuler, extrapall, and the predictor corrector) use for (c):

subroutine computef(neq, t, y, f)

implicit none

integer :: neq

double precision :: f(neq)

double precision, intent(in) :: t, y(neq)

f(1) = -(y(1) + 1) \* (y(1) + 3)

return

end

Below is the subroutine function that the three numerical methods (testeuler, extrapall, and the predictor corrector) use for (d):

subroutine computef(neq, t, y, f)

implicit none

integer :: neq

double precision :: f(neq)

double precision, intent(in) :: t, y(neq)

f(1) = -(5\*y(1)) + ((5\*t)\*\*2) + (2\*t)

return

end

First method used (testeuler):

program testeuler

implicit none

integer :: i, neq, nsteps

double precision, allocatable, dimension(:,:) :: archive

double precision :: tin, tout, dt, a, b, x, y, xactual, yactual

double precision, allocatable, dimension(:) :: yin, yout, t, f1, f2, u

neq = 1

nsteps = 50

allocate(yin(neq), yout(neq), t(0:nsteps), u(neq), f1(neq), f2(neq), archive(neq, 0:nsteps))

a = 1.0d0

b = 2.0d0

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

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 + f2) / 2.0d0)

archive(:,i+1) = yout

end do

print\*, 'creating "out1"'

open(unit=8, file='q1out1', status='replace')

do i = 0, nsteps

write(8,\*) t(i), archive(1,i)

end do

close(8)

print\*, 'writing pcerror'

open(unit=8, file='pcerror', status='replace')

do i = 0, nsteps

x = archive(1,i)

y = archive(2,i)

xactual = cos(t(i))

yactual = t(i) / (1 + log(t(i)))

write(8,\*) i, sqrt((x - xactual)\*\*2 + (y - yactual)\*\*2)

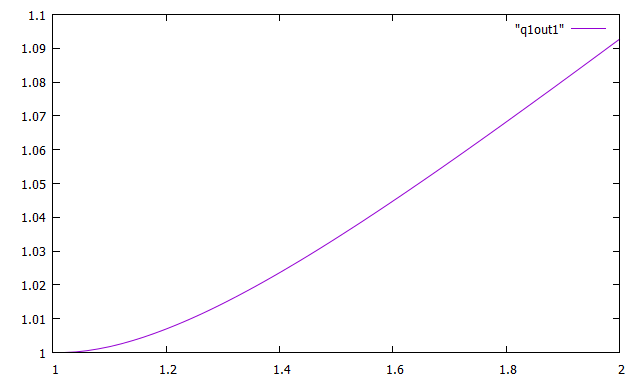
end do

deallocate(yin, yout, archive, t, f1, f2)

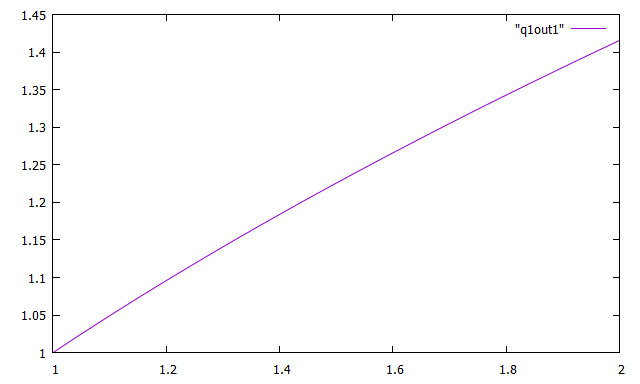
stop

end program testeuler

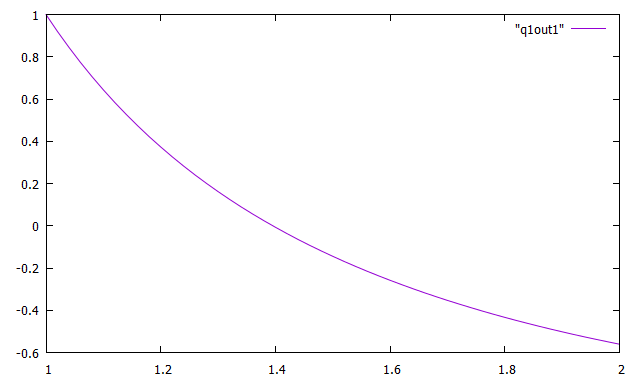
Method 1 used with the subroutine computef for A:



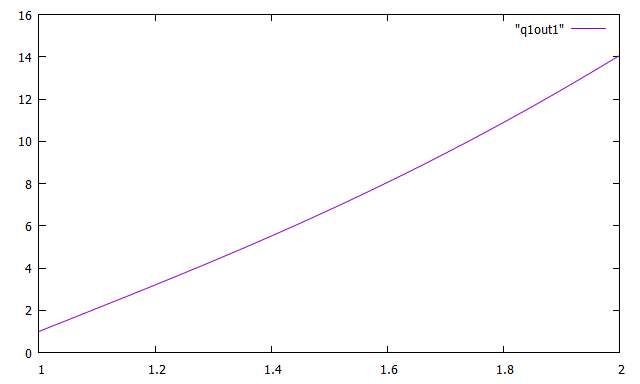
Method 1 used with the subroutine computef for B:



Method 1 used with the subroutine computef for C:



Method 1 used with the subroutine computef for D:



The second method used (extrapall):

program extrapall

implicit none

integer :: neq, nsteps, i

double precision, allocatable, dimension(:,:) :: archive

double precision :: tin, tout, dt, a, b, x, y, xactual, yactual

double precision, allocatable, dimension(:) :: yin, yout, t, u, f, fi, fim1, fim2, fim3, fim4, fip1

neq = 1

nsteps = 50

allocate(yin(neq), yout(neq), t(0:nsteps), u(neq), f(neq), archive(neq,0:nsteps))

allocate(fi(neq), fim1(neq), fim2(neq), fim3(neq), fim4(neq), fip1(neq))

a = 1.0d0

b = 2.0d0

dt = (b - a) / dble(nsteps)

print\*, 'starting loop'

do i = 0, nsteps

t(i) = a + dble(i) \* dt

end do

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

do i = 0, nsteps - 1

tin = t(i)

tout = t(i + 1)

yin = archive(:,i)

call extrapolate(neq, tin, tout, yin, yout)

archive(:,i+1) = yout

end do

print\*, 'writing out4'

open(unit=8, file='q1out4', status='replace')

do i = 0, nsteps

write(8,\*) t(i), archive(1,i)

end do

close(8)

print\*, 'writing main4error'

open(unit=8, file='main4error', status='replace')

do i = 0, nsteps

x = archive(1,i)

y = archive(2,i)

xactual = cos(t(i))

yactual = t(i) / (1 + log(t(i)))

write(8,\*) i, sqrt((x - xactual)\*\*2 + (y - yactual)\*\*2)

end do

close(8)

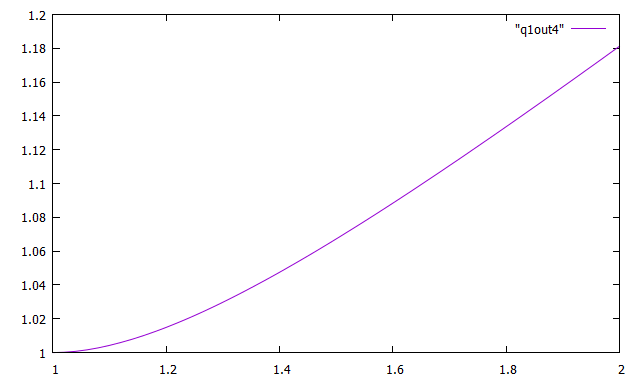
deallocate(yin, yout, u, archive, t, f)

deallocate(fi, fim1, fim2, fim3, fim4, fip1)

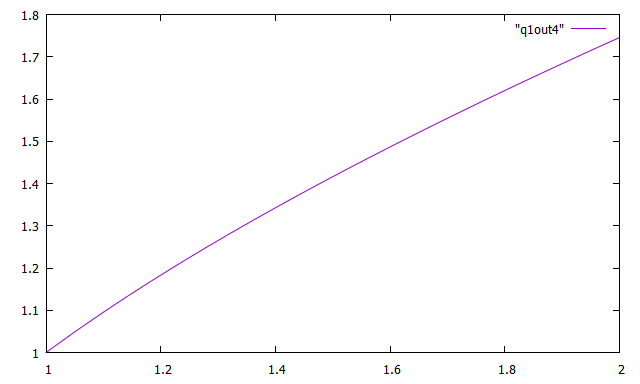
stop

end program extrapall

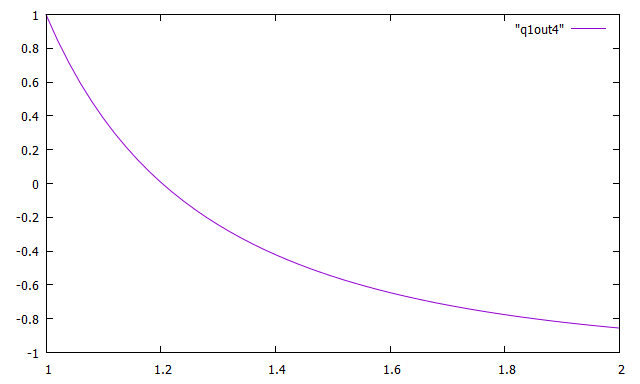
Method 2 used with the subroutine computef for A:



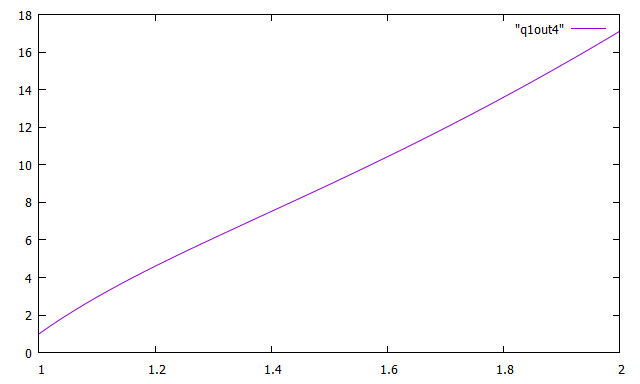
Method 2 used with the subroutine computef for B:



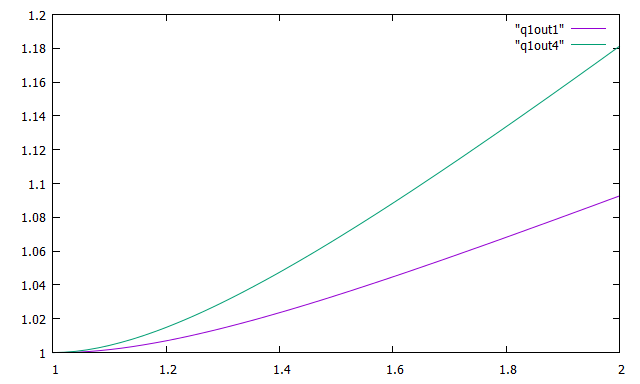
Method 2 used with the subroutine computef for C:



Method 2 used with the subroutine computef for D:



Method 1 and 2 used with the subroutine computef for A:



The third method used (predictor corrector):

program predictorcorrector

implicit none

integer :: i, neq, nsteps

double precision , allocatable, dimension(:) :: yin, yout, t, u, f

double precision , allocatable, dimension(: , :) :: archive

double precision :: dt, a, b, x, y, xactual, yactual

double precision , allocatable, dimension(:) :: fi, fim1, fim2, fim3, fim4, fip1

neq = 1

nsteps = 100

allocate( yin(neq), yout(neq), u(neq), t(0:nsteps), f(neq), archive(neq, 0:nsteps))

allocate(fi(neq), fim1(neq), fim2(neq), fim3(neq), fim4(neq), fip1(neq))

a = 1.0d0

b = 2.0d0

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 /)

call abinitializer(neq, nsteps, t, archive)

do i = 4, nsteps - 1

call computef(neq, t(i), archive(:,i), fi)

call computef(neq, t(i-1), archive(:,i-1), fim1)

call computef(neq, t(i-2), archive(:,i-2), fim2)

call computef(neq, t(i-3), archive(:,i-3), fim3)

call computef(neq, t(i-4), archive(:,i-4), fim4)

u = archive(:, i) + dt / 720.0d0 \* (1901.0d0 \* fi -2774.0d0\*fim1 + 2616.0d0\* fim2 - 1274.0d0\*fim3 + 251.0d0 \* fim4)

call computef(neq, t(i+1), u, fip1)

archive(:, i+1) = archive(:,i) + dt / 720.0d0 \* (251.0d0 \* fip1 + 646.0d0\*fi - 264.0d0\*fim1 + 106.0d0\*fim2 - 19.0d0 \* fim3)

end do

print\*, 'writing "aopc"'

open(unit=8, file ='aopc', status='replace')

do i = 0, nsteps

write(8,\*) t(i), archive(1,i)

end do

close(8)

print\*, 'writing "aopcerror"'

open(unit = 8, file = 'aopcerror', status = 'replace')

do i = 0, nsteps

x = archive(1, i)

y = archive(2, i)

xactual = cos(t(i))

yactual = t(i) / ( 1 + log(t(i)))

write(8,\*) i, SQRT((x - xactual)\*\*2 + (y - yactual)\*\*2)

end do

deallocate(yin, yout, u, archive, t, f)

deallocate(fi, fim1, fim2, fim3, fim4, fip1)

stop

end program predictorcorrector

subroutine abinitializer(neq, nsteps, t, archive)

implicit none

integer :: nsteps

integer, intent(in) :: neq

double precision, intent(in) :: t(0:nsteps)

double precision :: yout(neq), tin, tout, yin(neq), 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)

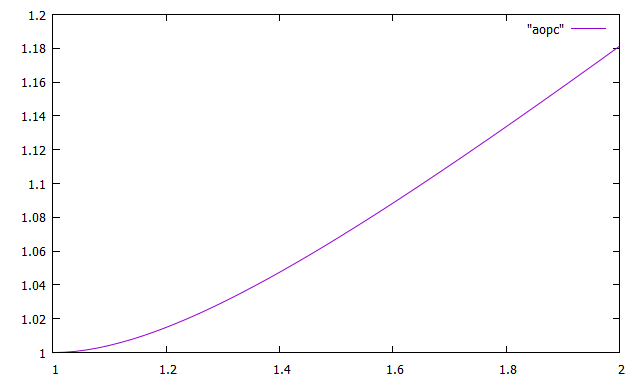
call extrapolate(neq, tin, tout, yin, yout)

archive(:,4) = yout

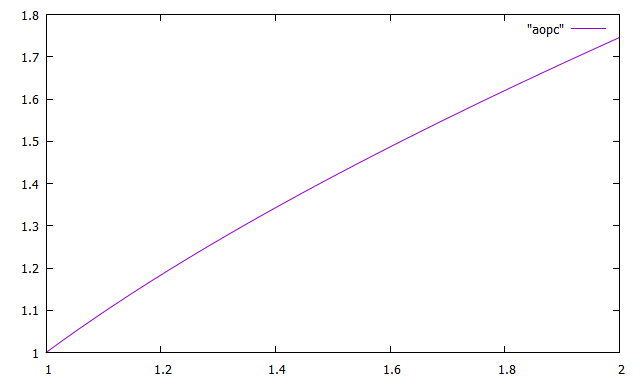
return

end

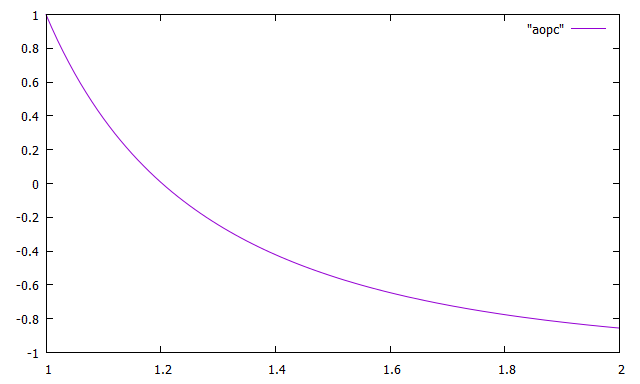
Method 3 used with the subroutine computef for A:



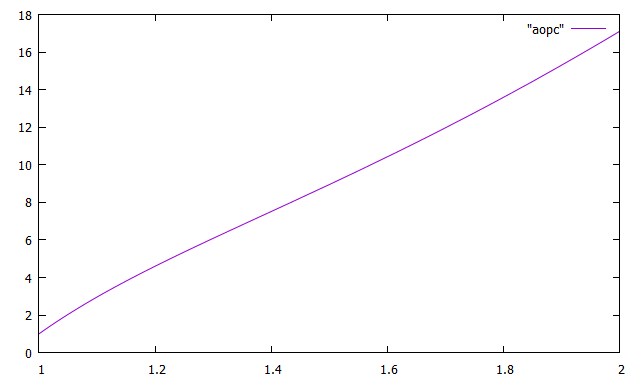
Method 3 used with the subroutine computef for B:



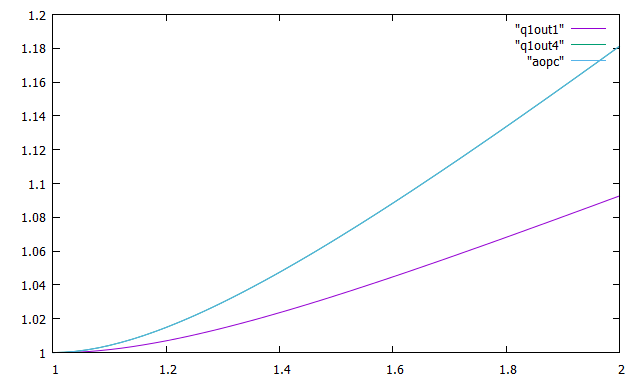
Method 3 used with the subroutine computef for C:



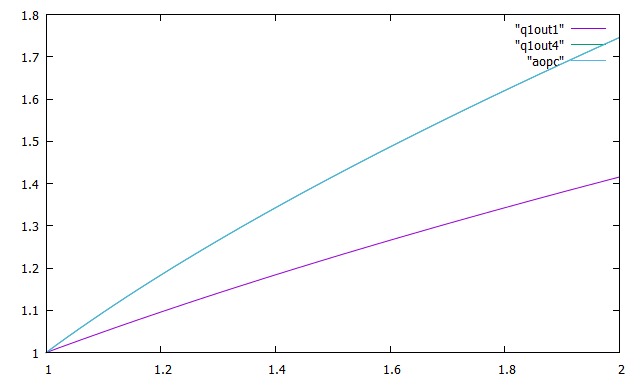
Method 3 used with the subroutine computef for D:



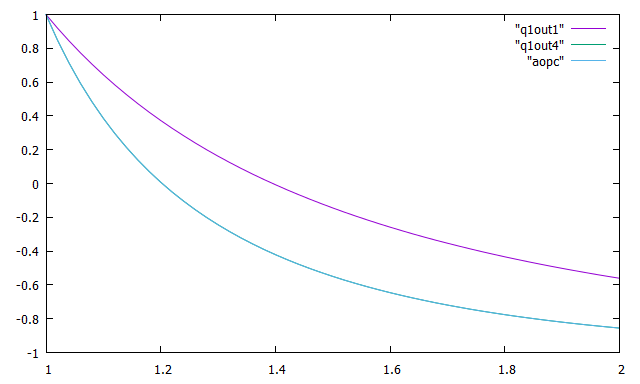
Method 1, 2, and 3 used with the subroutine computef for A:



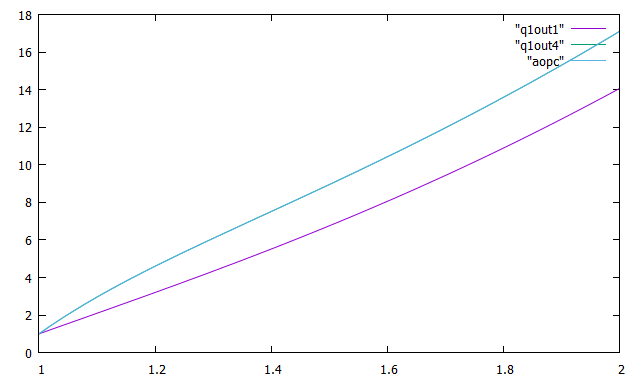
Method 1, 2, and 3 used with the subroutine computef for B:



Method 1, 2, and 3 used with the subroutine computef for C:



Method 1, 2, and 3 used with the subroutine computef for D:



1. Solve problem 6 of section 5.4 using at least two numerical schemes. RK4 need not be one of them.

Method 1 Explanation

For the first numerical scheme, I used euler which is the least accurate. This program uses the subroutine computef. First, I declare the variables, initialize them, and then allocate the memory needed. The main driver program sets the variables and then calls the subroutine computef. This subroutine computes the right-hand side of the differential equation and it indexes from one to neq where the array of right sides is f. In the computef I set f(1) to equal to exp(y(1)). After it completes this subroutine it then returns to the main driver program. It then opens or creates the file “out1” and writes the data to it. Afterwards, it closes the file, opens or creates a new file for the margin of errors and writes that data into the new file. At the end of the main driver program it deallocates the memory allocated to prevent a memory leak and then closes the last remaining file. The resulting graphs follow the snippets of code.

Method 2 Explanation

For the second numerical scheme, I used extrapall. This program uses the subroutines: extrapolate, euler, and computef. The first thing done in the main driver program is the declaration of the variables, initialization of the variables declared, and then allocation of the memory that will be needed. Then within a loop the driver program sets the variables and passes this information to the extrapolate subroutine. Extrapolate computes the estimate by then passing the variables received to the subroutine euler, through the form of a loop. The euler subroutine then takes that data, after altering tin and yout in a loop, and sends it to the computef subroutine. The subroutine computef computes the right-hand side of the differential equation and it indexes from one to neq where the array of right sides is f. In the computef I set f(1) to equal to exp(y(1)). After it completes this subroutine it then returns to the main driver program, opens or creates the file “out4” and writes the data to it. Afterwards, it closes the file, opens or creates a new file, called “main4error”, for the margin of errors and writes that data into that file. At the end of the main driver program it deallocates the memory that was allocated and closes the last remaining file. The resulting graphs follow the snippets of code.

Below are the subroutine methods that both numerical schemes use. The subroutines include extrapolate, euler, and computef.

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

nsteps = 1

allocate(table(0:nrich,0:nrich,neq))

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

subroutine euler(neq, tin, tout, nsteps, yin, yout)

implicit none

integer :: i

integer, intent(in) :: neq, nsteps

double precision, intent(in) :: tin, tout, yin(neq)

double precision :: dt, f(neq), yout(neq)

yout = yin

dt = ((tout - tin) / dble(nsteps))

do i = 0, nsteps - 1

call computef(neq, tin + dble(i) \* dt, yout, f)

yout = (yout + dt \* f)

end do

return

end

subroutine computef(neq, t, y, f)

implicit none

integer :: neq

double precision :: f(neq)

double precision, intent(in) :: t, y(neq)

f(1) = exp(y(1))

return

end

Below is the main program for the first method used, euler:

program euler

implicit none

integer :: neq, nsteps, i

double precision, allocatable, dimension(:,:) :: archive

double precision :: tin, tout, dt, a, b, x, y, xactual, yactual

double precision, allocatable, dimension(:) :: yin, yout, t, f1, f2, u

neq = 1

a = 0.0d0

b = 0.2d0

nsteps = 20

dt = (b - a) / dble(nsteps)

allocate(yin(neq), yout(neq), t(0:nsteps), u(neq), f1(neq), f2(neq), archive(neq,0:nsteps))

do i = 0, nsteps

t(i) = a + dble(i) \* dt

end do

archive(:,0) = 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 + f2) / 2.0d0

archive(:,i+1) = yout

end do

print\*, 'creating "out1"'

open(unit=8, file='out1', status='replace')

do i = 0, nsteps

write(8,\*) t(i), archive(1,i)

end do

close(8)

print\*, 'writing pcerror'

open(unit=8, file='pcerror', status='replace')

do i = 0, nsteps

x = t(i)

y = archive(1,i)

xactual = cos(t(i))

yactual = sin(t(i))

write(8,\*) i, sqrt((x-xactual)\*\*2 + (y-yactual)\*\*2)

end do

deallocate(yin, yout, archive, t, f1, f2)

stop

end program euler

Below is the main program for the second method used, extrapall:

program extrapall

implicit none

integer :: neq, nsteps, i

double precision, allocatable, dimension(:,:) :: archive

double precision :: tin, tout, dt, a, b, x, y, xactual, yactual

double precision, allocatable, dimension(:) :: yin, yout, t, u, f, fi, fim1, fim2, fim3, fim4, fip1

neq = 1

a = 0.0d0

b = 0.2d0

nsteps = 20

dt = (b - a) / dble(nsteps)

allocate(yin(neq), yout(neq), t(0:nsteps), u(neq), f(neq), archive(neq,0:nsteps))

allocate(fi(neq), fim1(neq), fim2(neq), fim3(neq), fim4(neq), fip1(neq))

print\*, 'starting loop'

do i = 0, nsteps

t(i) = a + dble(i) \* dt

end do

archive(:,0) = 1.0d0

do i = 0, nsteps - 1

tin = t(i)

tout = t(i + 1)

yin = archive(:,i)

call extrapolate(neq, tin, tout, yin, yout)

archive(:, i + 1) = yout

end do

print\*, 'writing out4'

open(unit=8, file='out4', status='replace')

do i = 0, nsteps

write(8,\*) t(i), archive(1,i)

end do

close(8)

print\*, 'writing main4error'

open(unit=8, file='main4error', status='replace')

do i = 0, nsteps

x = t(i)

y = archive(1,i)

xactual = cos(t(i))

yactual = sin(t(i))

write(8,\*) i, sqrt((x-xactual)\*\*2 + (y-yactual)\*\*2)

end do

close(8)

deallocate(yin, yout, u, archive, t, f)

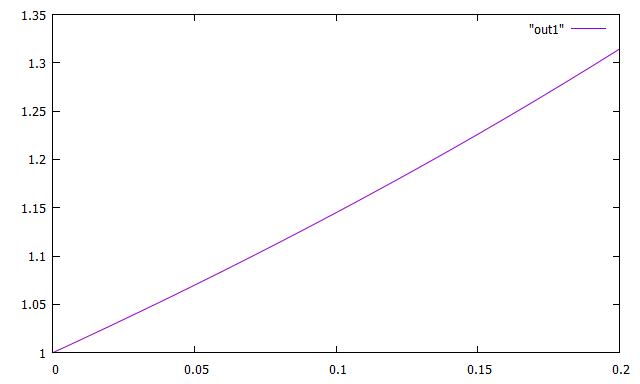
deallocate(fi, fim1, fim2, fim3, fim4, fip1)

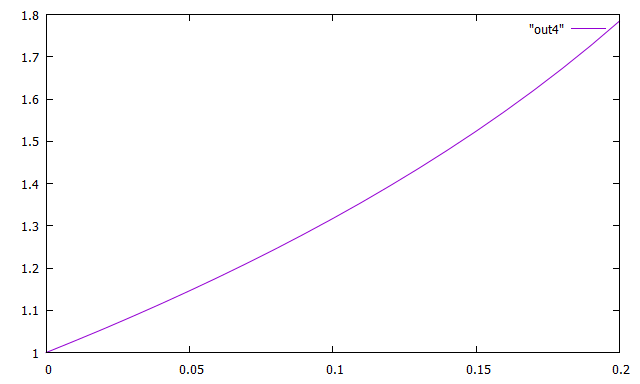
stop

end program extrapall

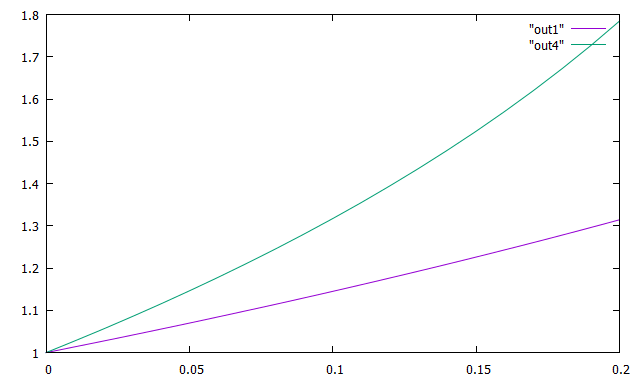
Below are the graphs of the methods used (euler and extrapall) along with a graph of them together.

Graph of the euler method:



Graph of the extrapall method: 

Graph of the euler and extrapall methods together (displaying the margin of error):



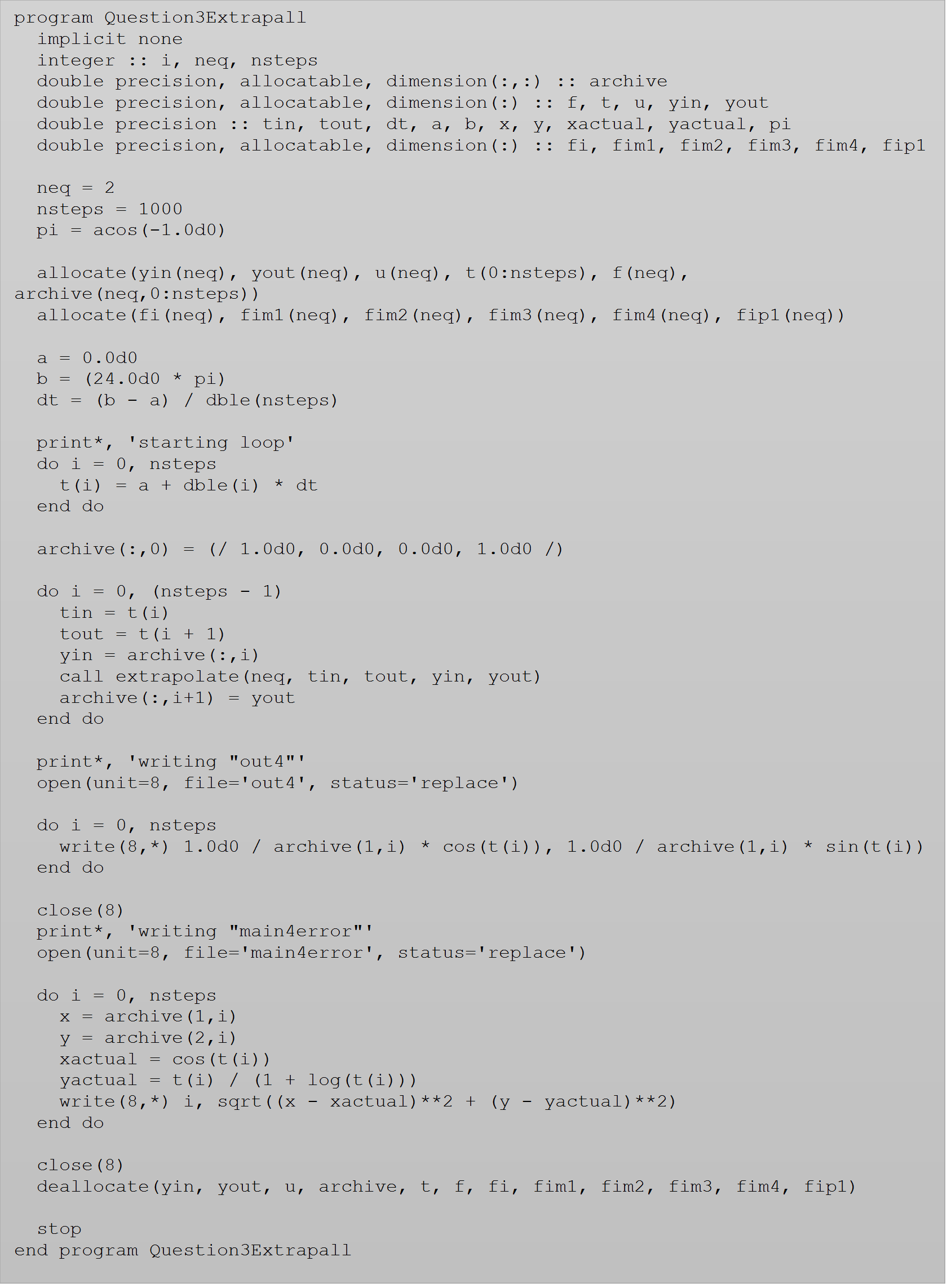
1. Another formulation of the orbit problem is to use polar coordinates. Variables *u* = 1/*r* and so

In these variables the differential equation of motion is

Where *A* is a constant. For circular orbits the radius will be constant, and so will *u*. So, in our simple problem,

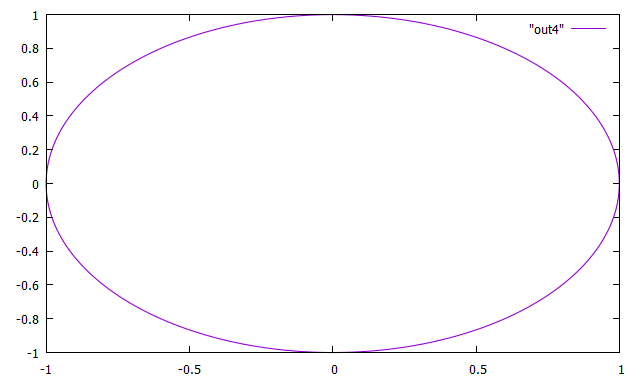
The initial condition will be *u*(0) = 1, *u’*(0) = 0. Solve this problem for (that is, 12 orbital periods). Then experiment with the initial value of *u’* and get a nice elongated ellipse. Draw graphs, and explain fully what you’ve done.

For this problem, the main driver program uses the subroutines: extrapolate, euler, and computef. First, I declare and initialize the variables (b is the 12 orbital periods), allocate memory for select variables, and set the initial state. Afterwards, the program loops through nsteps minus one times and sets variables and then sends those variables to the subroutine extrapolate. Extrapolate computes the estimate by then passing the variables received to the subroutine euler, through the form of a loop. The euler subroutine then takes that data and sends it to the computef subroutine. The subroutine computef computes the right hand side of the differential equation and it indexes from one to neq where the array of right sides is f. In the computef I set f(1) to equal y(2) and then f(2) to equal the negative of y(1) plus one. After the loop, in which all of the nested calls to subroutines, completes it writes that data into a file called “out4”, closes that file, and then writes the margin of error into a separate file called “main4error”. At the end of the driver it then closes the second file and deallocates the memory. For this problem, I kept the double precision allocatable dimension variable t as the independent variable. I also set the reciprocal of the radius, a, to u for the circular orbit. I originally used the condition of zero and then tried different values for the initial dt and the results follow the code snippet below.

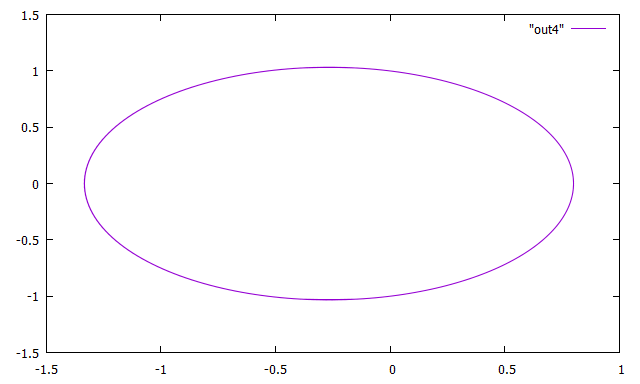


Below is the resulting graph of when we change the initial state, archive, to

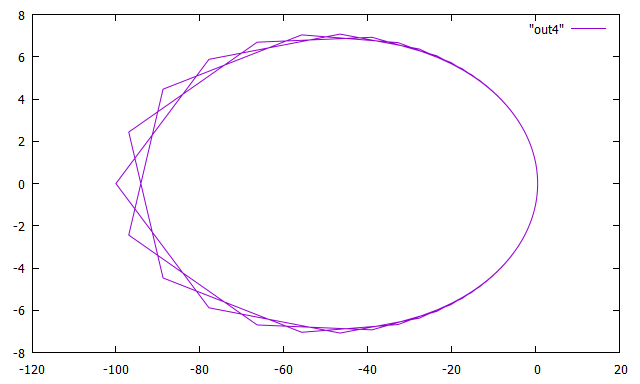
(/ 1.0d0, 0.0d0, 0.0d0, 1.0d0 /):



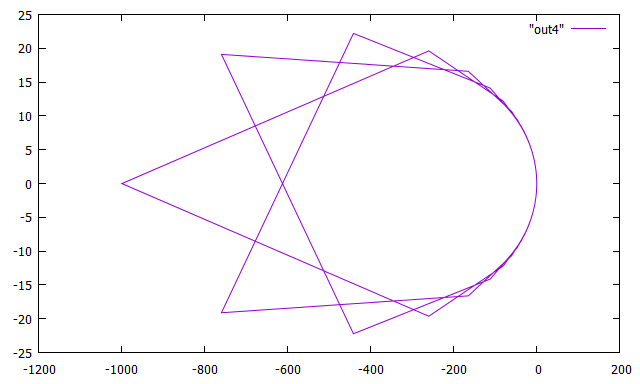
When we change the initial state, archive, to (/ 1.25d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:



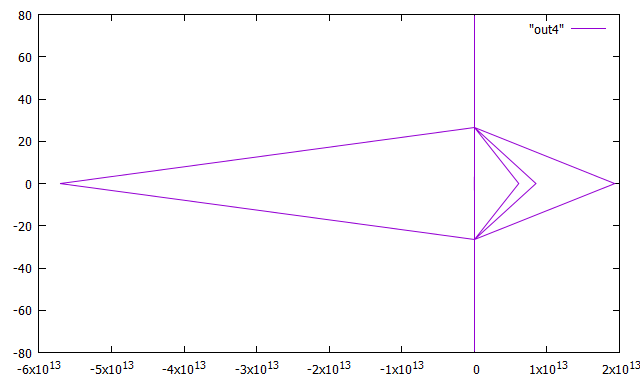
When we change the initial state, archive, to (/ 1.99d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:



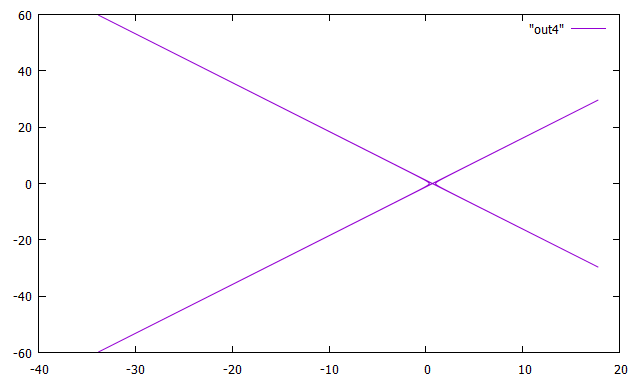
When we change the initial state, archive, to (/ 1.999d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:



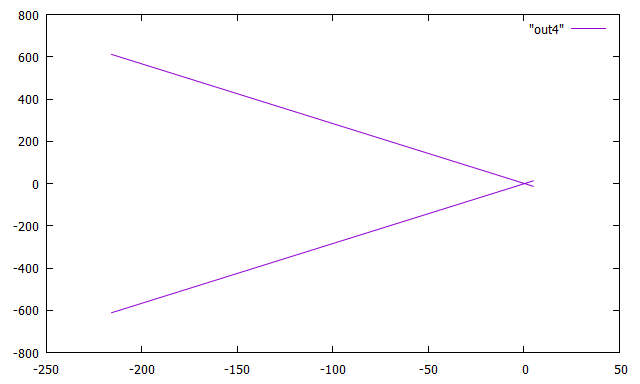
When we change the initial state, archive, to (/ 2.0d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:



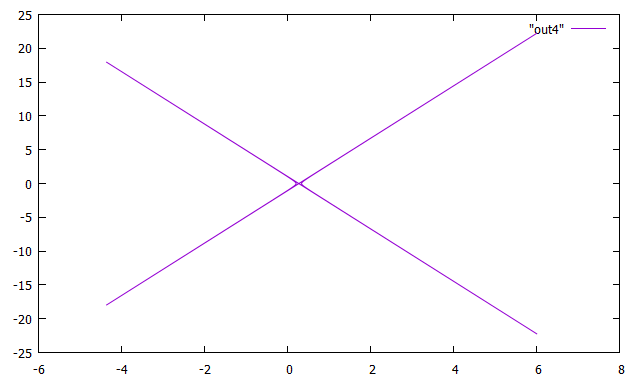
When we change the initial state, archive, to (/ 3.0d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:

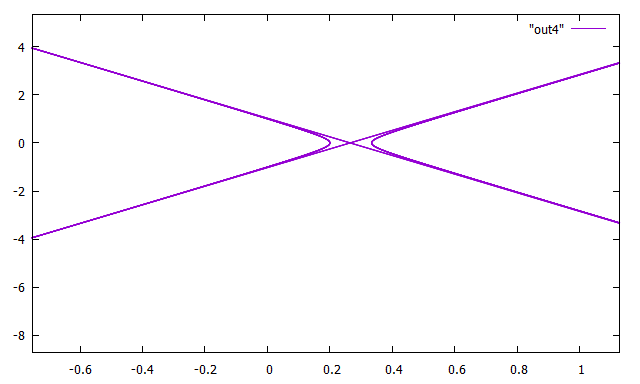


When we change the initial state, archive, to (/ 4.0d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:



When we change the initial state, archive, to (/ 5.0d0, 0.0d0, 0.0d0, 1.0d0 /) we receive:





1. (Extra Credit) Repeat problem three using this new equation. What effect is there on the orbit?

For this problem, I added a double precision constant variable, constb, which I set to (1 / 100). I then changed the double precision variable b to (48.0d0 \* pi) and then (72.0d0 \* pi) to see the change. For the new equation I added the Bu2, ((constb\*u)\*\*2), to the yin within the loop. When b was changed to (48.0d0 \* pi) you could see in the graph that the y points stayed the same. However, the x variables changed, they stretched (graph included). When the b was changed to (72.0d0 \* pi) something interesting happens. The points are similar to the (48.0d0 \* pi) but the orbit loops multiple times (graph and zoomed in graph included). There is a similar effect at (96.0d0 \* pi) except there seems to be a decrease in loops when compared with the (72.0d0 \* pi) graph.

program extrapall

implicit none

integer :: i, neq, nsteps

double precision, allocatable, dimension(:,:) :: archive

double precision, allocatable, dimension(:) :: f, t, u, yin, yout

double precision :: tin, tout, dt, a, b, x, y, xactual, yactual, pi, constb

double precision, allocatable, dimension(:) :: fi, fim1, fim2, fim3, fim4, fip1

neq = 2

nsteps = 1000

pi = acos(-1.0d0)

allocate(yin(neq), yout(neq), u(neq), t(0:nsteps), f(neq), archive(neq,0:nsteps))

allocate(fi(neq), fim1(neq), fim2(neq), fim3(neq), fim4(neq), fip1(neq))

a = 0.0d0

b = (48.0d0 \* pi)

constb = (1/100)

dt = (b - a) / dble(nsteps)

print\*, 'starting loop'

do i = 0, nsteps

t(i) = a + dble(i) \* dt

end do

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

do i = 0, (nsteps - 1)

tin = t(i)

tout = t(i + 1)

yin = archive(:,i) + (constb\*u)\*\*2

call extrapolate(neq, tin, tout, yin, yout)

archive(:,i+1) = yout

end do

print\*, 'writing "out4"'

open(unit=8, file='out4', status='replace')

do i = 0, nsteps

write(8,\*) 1.0d0 / archive(1,i) \* cos(t(i)), 1.0d0 / archive(1,i) \* sin(t(i))

end do

close(8)

print\*, 'writing "main4error"'

open(unit=8, file='main4error', status='replace')

do i = 0, nsteps

x = archive(1,i)

y = archive(2,i)

xactual = cos(t(i))

yactual = t(i) / (1 + log(t(i)))

write(8,\*) i, sqrt((x - xactual)\*\*2 + (y - yactual)\*\*2)

end do

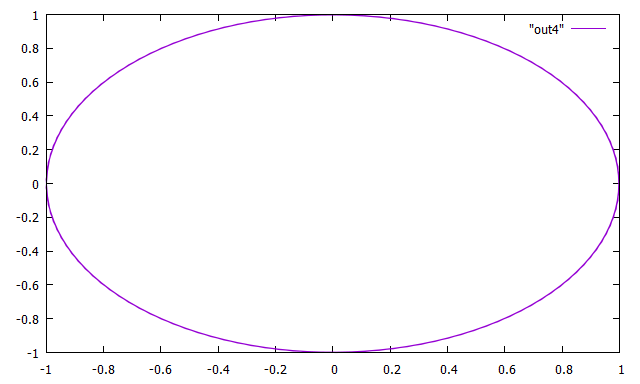
close(8)

deallocate(yin, yout, u, archive, t, f, fi, fim1, fim2, fim3, fim4, fip1)

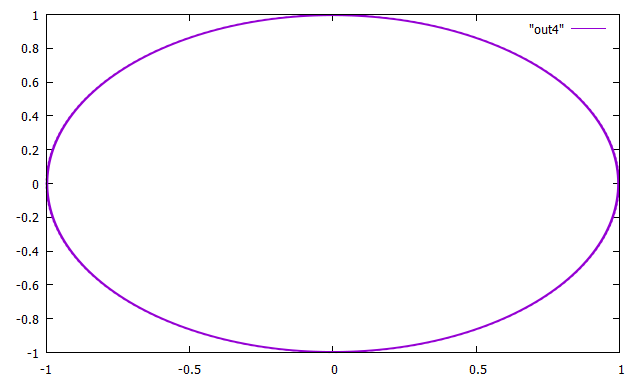
stop

end program extrapall

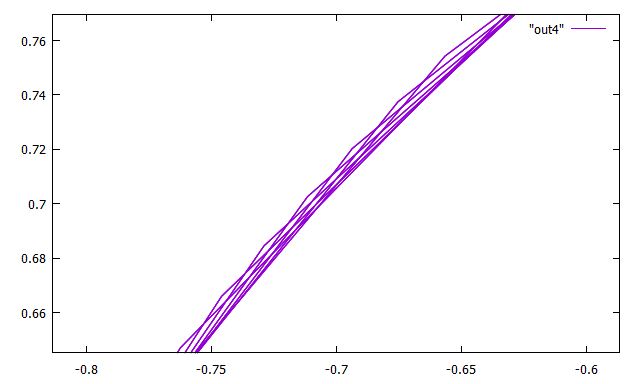
When b = (48.0d0 \* pi):



When b = (72.0d0 \* pi):



Zoomed in:



1. (Extra Credit) Try to determine the moment when the surface of the Earth touches the surface of the Sun. After that moment, the equation of motion will no longer be correct due to impact, friction, mass ejection from the Sun, the Earth getting vaporized and so on. If the program runs beyond this point it will likely crash. You’ll want to stop when the distance from the center of the Sun to the center of the Earth is about 434,000 miles, which in our units is about 4.6 x 10-3. Use at least two methods, both should be highly accurate.

After changing the initial position and using the formula (radius of the Sun + radius of the Earth = moment of collusion) and speed which is equivalent to over the period which is also equivalent to one year, the moment or time, t, of impact is 64 days. The program stops at 0.0046 (4.6 x 10-3 astronomical units) so it doesn’t crash.

program extrapall

implicit none

integer :: i, neq, nsteps

double precision, allocatable, dimension(:,:) :: archive

double precision, allocatable, dimension(:) :: f, t, u, yin, yout

double precision :: tin, tout, dt, a, b, x, y, xactual, yactual, pi

double precision, allocatable, dimension(:) :: fi, fim1, fim2, fim3, fim4, fip1

neq = 2

nsteps = 1000

pi = acos(-1.0d0)

allocate(yin(neq), yout(neq), u(neq), t(0:nsteps), f(neq), archive(neq,0:nsteps))

allocate(fi(neq), fim1(neq), fim2(neq), fim3(neq), fim4(neq), fip1(neq))

a = 0.0d0

b = (24.0d0 \* pi)

dt = (b - a) / dble(nsteps)

print\*, 'starting loop'

do i = 0, nsteps

t(i) = a + dble(i) \* dt

end do

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

do i = 0, (nsteps - 1)

tin = t(i)

tout = t(i + 1)

yin = archive(:,i)

call extrapolate(neq, tin, tout, yin, yout)

archive(:,i+1) = yout

end do

print\*, 'writing "out4"'

open(unit=8, file='out4', status='replace')

do i = 0, nsteps

write(8,\*) 1.0d0 / archive(1,i) \* cos(t(i)), 1.0d0 / archive(1,i) \* sin(t(i))

end do

close(8)

print\*, 'writing "main4error"'

open(unit=8, file='main4error', status='replace')

do i = 0, nsteps

x = archive(1,i)

y = archive(2,i)

xactual = cos(t(i))

yactual = t(i) / (1 + log(t(i)))

write(8,\*) i, sqrt((x - xactual)\*\*2 + (y - yactual)\*\*2)

end do

close(8)

deallocate(yin, yout, u, archive, t, f, fi, fim1, fim2, fim3, fim4, fip1)

stop

end program extrapall