01.

Celsius and Fahrenheit

The conversion from Fahrenheit to Celsius.

The Mathematical Form

Formula: C = (F-32)/1.8

I created a subroutine to convert Fahrenheit to Celsius.

```
subroutine sub_fun(C,F)
    implicit none
    double precision C
    double precision F
    C = (F-32)/1.8
end subroutine sub_fun
```

CYGNUS

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

DATA

```
implicit none

double precision :: F, C ! fahrenheit and celsius
double precision :: jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec ! avg temp each month
double precision :: j_f_m_C,j_f_m_F, a_m_j_C,a_m_j_F,&
                    j_a_s_C,j_a_s_F, o_n_d_C,o_n_d_F  !avg temp each three months
                                !in celsisus and fahrenheit

!googling data we get
!the average data in fareinheit
jan = 56.5 !68/45:High/Low consecutively
feb = 60.0
mar = 64.5
apr = 72.5
may = 80.5
jun = 90.0
jul = 94.0
aug = 93.0
sep = 86.5
oct = 76.5
nov = 63.5
dec = 56.0
```

```
!calculating the average of each three months in fahrenheit
j_f_m_F = (jan+feb+mar)/3
a_m_j_F = (apr+may+jun)/3
j_a_s_F = (jul+aug+sep)/3
o_n_d_F = (oct+nov+dec)/3
```

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

Recording Data

```fortran
! write the file
open(file="/Users/phihung/Documents/PHY/first/homework/hw_01/temp.dat", unit=10)
    ! call subroutine to do the conversions
    call sub_fun(j_f_m_C,j_f_m_F)
    call sub_fun(a_m_j_C,a_m_j_F)
    call sub_fun(j_a_s_C,j_a_s_F)
    call sub_fun(o_n_d_C,o_n_d_F)
    ! write to the file the data
    write(10,*) j_f_m_C,j_f_m_F
    write(10,*) a_m_j_C,a_m_j_F
    write(10,*) j_a_s_C,j_a_s_F
    write(10,*) o_n_d_C,o_n_d_F
close(10)
end
```

```
15.740741157727980      60.333333333333336
27.222229433648590      81.000000000000000
32.870371241137839      91.166666666666671
18.518519009091737      65.333333333333329
```

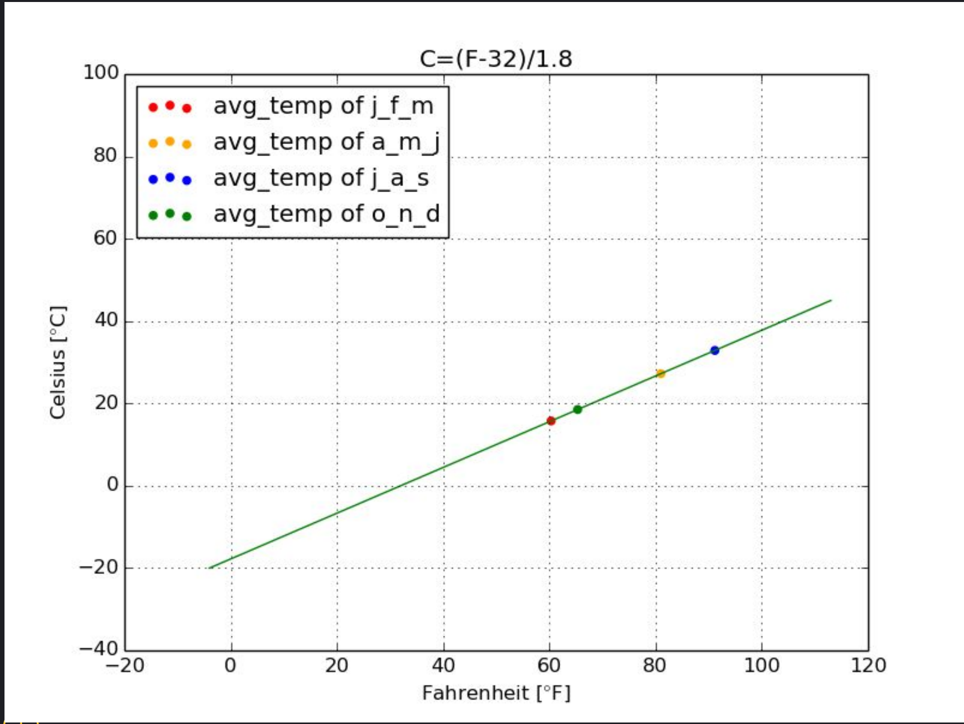
003-1040559 1250 00

Python Plotting

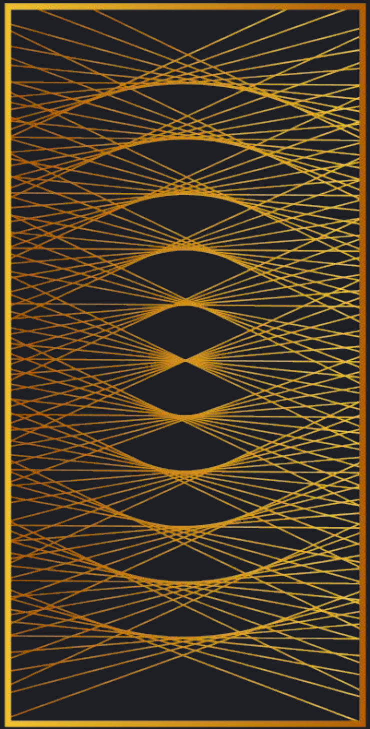
Use the data generated from fortran to plot the relationship between celsius and fahrenheit.

```python
#import necessary libs
import matplotlib.pyplot as plt
import numpy as np

#load data
data = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/temp.dat')

#unload data
celsius = data[:,0] # load first column
fahrenheit = data[:,1] # load second colum

#plot the line
plt.plot(celsius,fahrenheit)

#plot the points
plt.scatter(fahrenheit[0],celsius[0],color = 'red',label='avg_temp of j_f_m' )
plt.scatter(fahrenheit[1],celsius[1],color = 'orange',label='avg_temp of a_m_j' )
plt.scatter(fahrenheit[2],celsius[2],color = 'blue',label='avg_temp of j_a_s' )
plt.scatter(fahrenheit[3],celsius[3],color = 'green',label='avg_temp of o_n_d' )

#label
plt.xlabel(r'Fahrenheit [$\degree$F]')
plt.ylabel(r'Celsius [$\degree$C]')

#plot frrom -20C->-4F to 45C->113
y = [-20, 45]
x = [-4, 113]
plt.plot(x,y)

#title of graph
plt.title('C=(F-32)/1.8')

#add legend
plt.legend(loc='upper left')
plt.grid(True)

plt.show()
```

Plotting



Conclusion:
The relationship between Celsius and Fahrenheit is a linear relationship. The graph shows that a_m_j has the lowest temperature while j_a_s has the highest temperature.



02.

Folium of Descartes

Generate data: x^3+y^3-3axy=0

For a=2,3,4

The Mathematical Form

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

```
implicit none !so it doesnt assign random variables to be integers
! function to be generated is x^3 + y^3 -3axy = 0
! converting function to polar coordinate
! r = 3acos(theta)sin(theta)/[cos^3(theta)+sin^3(theta)]
double precision :: step, start
integer :: n !for looping
integer, parameter :: a1= 2, a2=3, a3=4 !a-values
real(kind=8) :: pi
integer, parameter :: n_max = 1000 ! number of theta points to be generated
double precision :: theta(n_max), r(n_max) ! array for storing data points

! set default values (theta, r)
theta = 0.0 !angle
r = 0.0 !radius

! find n_max points from -pi/6 to pi/1.5
pi=4.D0*DATAN(1.D0) !calculate pi values
start = -pi/6 !start from -pi/6
step = ((pi/1.5)+(pi/6)) / n_max !the value of one step

! assign the values (theta, r)
do n=1,n_max ! loop through all coordinates
    theta(n) = start ! take theta value
    start = start + step ! update theta value
    r(n) = (3*cos(theta(n))*sin(theta(n))) / ((cos(theta(n)))**3 + (sin(theta(n)))**3)  ! take radius value
enddo
```

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

Record Data

Two forms of coordinates are then generated for a=2,3,4.

```
!record the data for polar values (theta, r*2,r*3,r*4)
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/polar_coordinates.dat', unit=10)
do n=1, n_max
    write(10,*) theta(n), r(n)*2, r(n)*3, r(n)*4
enddo
close(unit=10)

!record the data for cartesian values (x1,y1)
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/cartesian_coordinates_1.dat', unit=11)
do n=1, n_max
    write(11,*) r(n)*2*cos(theta(n)), r(n)*2*sin(theta(n))! x = rcos(theta), y=rsin(theta) a=2
enddo
close(unit=11)
!record the data for cartesian values (x2,y2)
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/cartesian_coordinates_2.dat', unit=11)
do n=1, n_max
    write(11,*) r(n)*3*cos(theta(n)), r(n)*3*sin(theta(n))! x = rcos(theta), y=rsin(theta) a=3
enddo
close(unit=11)
!record the data for cartesian values (x3,y3)
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/cartesian_coordinates_3.dat', unit=11)
do n=1, n_max
    write(11,*) r(n)*4*cos(theta(n)), r(n)*4*sin(theta(n))! x = rcos(theta), y=rsin(theta) a=4
enddo
close(unit=11)
end
```

003-10405

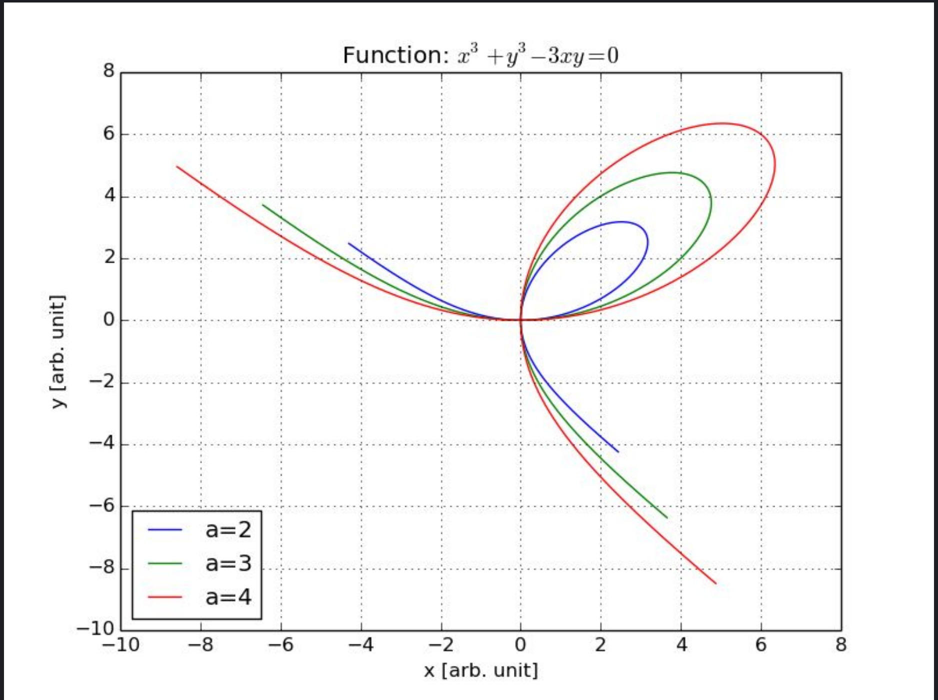
Plotting

I plot three figures on one

```python
#import necessary libs
import matplotlib.pyplot as plt
import numpy as np

#unpack data
data1 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/cartesian_coordinates_1.dat')
data2 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/cartesian_coordinates_2.dat')
data3 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/cartesian_coordinates_3.dat')
#unload (x1,y1), (x2,y2), (x3,y3)
x_1 = data1[:,0]
y_1 = data1[:,1]
x_2 = data2[:,0]
y_2 = data2[:,1]
x_3 = data3[:,0]
y_3 = data3[:,1]

#plot 3 graphs
plt.plot(x_1,y_1,label='a=2')
plt.plot(x_2,y_2,label='a=3')
plt.plot(x_3,y_3,label='a=4')

#label x and y axis
plt.xlabel('x [arb. unit]')
plt.ylabel('y [arb. unit]')

#title
plt.title(r'Function: $x^{3} + y^{3} - 3xy = 0$')


plt.legend(loc='lower left')
plt.grid(True)

plt.show()
```

Conclusion

The three functions are from the same tree family. The problem is easier to approach in polar form.



T

1

03

Gregory Series

$$\pi = 4 \cdot \lim_{N \to \infty} \left(\sum_{n=1}^{N} \frac{(-1)^{n+1}}{2 \cdot n - 1} \right)$$

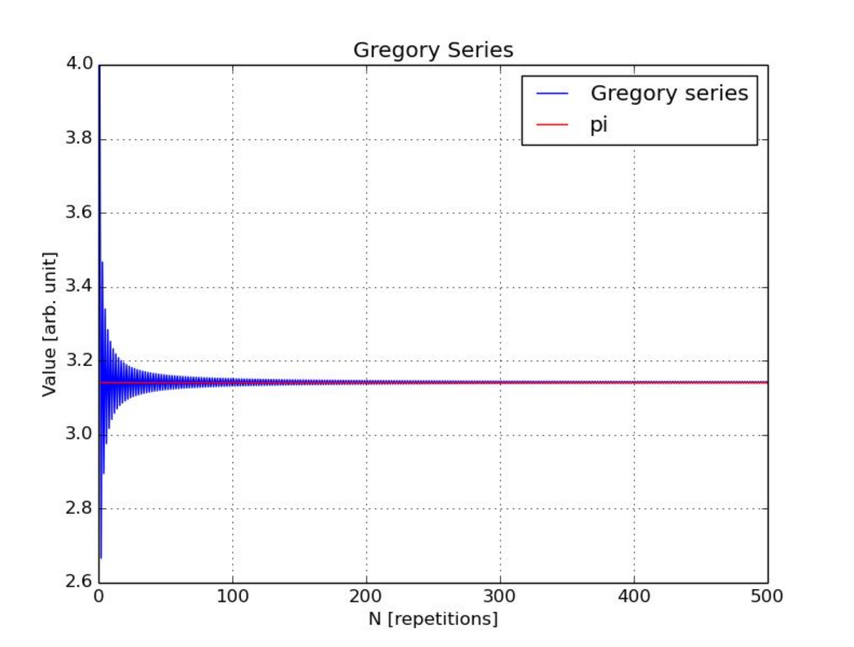
Do loop for gregory convergence (inner loop)
and record value for each N (outer loop)

```fortran
implicit none

integer, parameter :: n_max = 500  ! N values
integer :: n_2, n_1
double precision :: value ! the pi values recorded

! open the file to record
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/gregory_convergence.dat', unit=11)
do n_1=1, n_max !calculate convergence from N = 1 to 500
    value = 0.0 ! reset value each loop
    do n_2=1, n_1 !calculate the series for each N value
        value = value + (((-1)**(float(n_2)+1)) / (2*float(n_2) - 1)) !gregory series
    end do
    value = value * 4.0 ! times 4 from outside
    write(11,*) n_1, value ! write the N and the pi value recorded
end do
close(unit=11)

end
```

```python
import matplotlib.pyplot as plt
import numpy as np

#unload data
data = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/gregory_convergence.dat')

# unpack value
n = data[:,0]
pi = data[:,1]

#define y=pi function
def pi_function(n):
    return n*0 + np.pi
# record on k
k = pi_function(n)

# plot the repetition vs
plt.plot(n,pi)
plt.xlabel('N [repetitions]')
plt.ylabel('Value [arb. unit]')

plt.plot(n,k, color='red')

plt.title(r'Gregory Series')

plt.legend(['Covergence'])
plt.grid(True)

plt.show()
```

Plotting

I plot repetitions vs value of pi



Conclusion

The series converges quickly and almost about the exact value around N=200

04

Function Series

Create a data set for each N=1,10,100,1000,10000

Given N
Inner loop: find the value at a single x value
Outer loop: find all x values

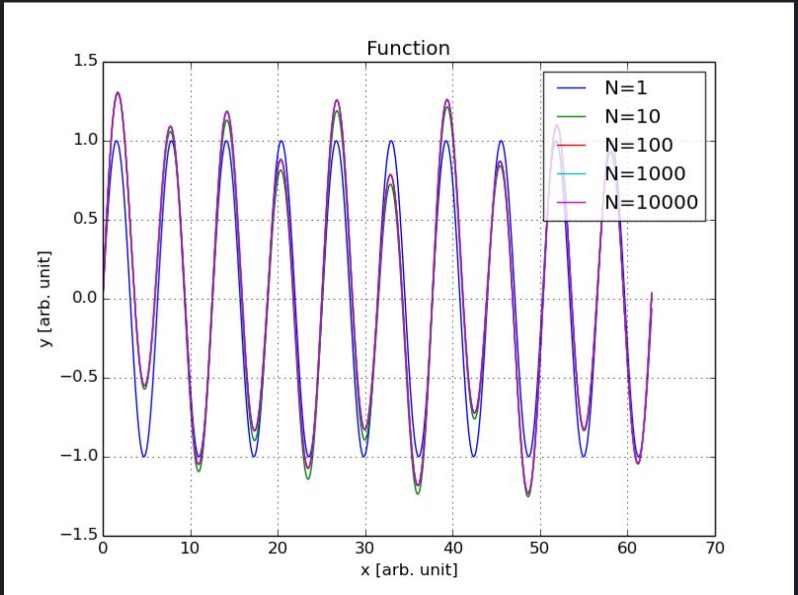
```fortran
implicit  none
double precision :: pi
integer :: Nt=1000,N=1000 ! Nt:number of x values, N: iterations for sigma
double precision :: dx ! x_step
integer :: n_1, n_2
double precision :: x, y ! functions
double precision :: function, x_current


pi=4.D0*DATAN(1.D0)! create pi

dx = 20*pi / float(Nt) !find dx between 0 to 20*pi

!default value
function = 0.0
x_current = 0.0

open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/convergence_1000.dat',unit=10)
do n_1 = 1, Nt ! loop through to calculate all points
    x = x_current ! assign the x value
    function = 0.0 ! reset value for y

    do n_2 = 1, N !use looping thr sigma to find a single point f(x)
        function = function + sin(x/n_2)/(n_2**2) !find the f(x) value
    end do

    x_current = x_current + dx ! update the x-value
    y = function ! paste to y
    write(10,*) x , y ! record data
end do
close(unit=10)
end
```

Plotting

I plot 5 different N values to see the convergence

```python
import matplotlib.pyplot as plt
import numpy as np

#load data
data1 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/convergence_1.dat')
data10 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/convergence_10.dat')
data100 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/convergence_100.dat')
data1000 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/convergence_1000.dat')
data10000 = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/convergence_10000.dat')

#unpack data
x_1 = data1[:,0]
y_1 = data1[:,1]
x_10 = data10[:,0]
y_10 = data10[:,1]
x_100 = data100[:,0]
y_100 = data100[:,1]
x_1000 = data1000[:,0]
y_1000 = data1000[:,1]
x_10000 = data10000[:,0]
y_10000 = data10000[:,1]

plt.plot(x_1,y_1, label='N=1')
plt.plot(x_10,y_10, label='N=10')
plt.plot(x_100,y_100, label='N=100')
plt.plot(x_1000,y_1000, label='N=1000')
plt.plot(x_10000,y_10000, label='N=10000')
plt.xlabel('x [arb. unit]')
plt.ylabel('y [arb. unit]')

legend = plt.legend()
frame = legend.get_frame()
frame.set_alpha(0.8) #set tranparency

plt.title(r'Function')

#plt.legend()
plt.grid(True)
```

Conclusion



Purple shows the closest to the convergence while other lines are trying to converge to the purple line

01
02
03
04
05
06

01 02 03 04 05 06

05

Matrix Multiplication

$$A=\begin{pmatrix}1&2&3\\3&2&1\\1&1&1\end{pmatrix},B=\begin{pmatrix}1&1&1\\1&2&3\\3&2&1\end{pmatrix},$$

$$C_1=A\cdot B,$$

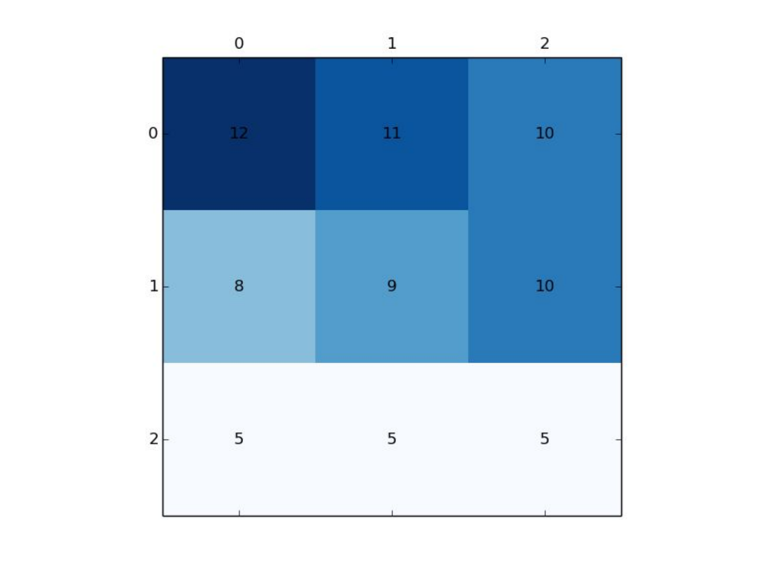
$$C_2=[A,B]\doteq A\cdot B-B\cdot A.$$

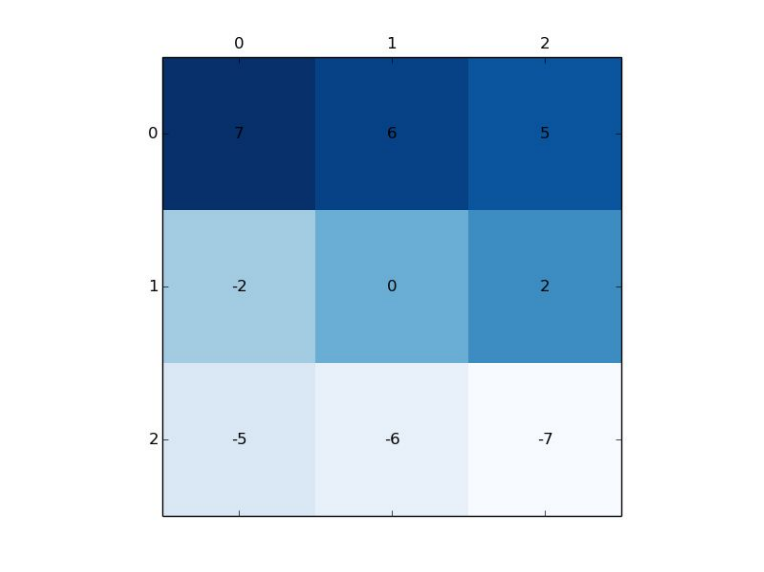
```fortran
implicit none
integer, parameter :: array1_size1=3,array1_size2=3 !dimension for A
double precision A(array1_size1,array1_size2)
integer, parameter :: array2_size1=3,array2_size2=3 !dimension for B
double precision :: B(array2_size1,array2_size2), T(array1_size1,array2_size2), D(array1_size1,array2_size2)
double precision :: C(array1_size1,array2_size2) !if we calculate A1xA2 => dimensions of C are [array1_size1 x array
                                                 !note that array1_size2=array2_size1 [otherwise you cannot calculate
double precision tmp
integer n,n1,n2

!quick check to ensure everything is correct

if(array1_size2.ne.array2_size1)then
  write(*,*) 'check A and B dimensions!'
  stop !incorrect dimensions, program is terminated
endif

!define the matrix A and B
A = reshape([1,2,3,&
             3,2,1,&
             1,1,1],[array1_size1, array1_size2])
B = reshape([1,1,1,&
             1,2,3,&
             3,2,1],[array2_size1, array2_size2])

do n1=1,array1_size1
  do n2=1,array2_size2
    tmp=0.0
    do n=1,array1_size2 !or array2_size1 'cause they
      tmp=tmp+A(n1,n)*B(n,n2) !
    enddo
    C(n1,n2)=tmp
  enddo
enddo

!try the builtin function
T=matmul(A,B)
D=matmul(B,A)
D=T-D !the commutator

open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/matrix_multiplication.dat',unit=10)
write(10,*) "Matrix multiplication of A and B gives"
write(10,*) "using logic and loops"
write(10,*) C(3,3), C(2,3), C(1,3)
write(10,*) C(3,2), C(2,2), C(1,2)
write(10,*) C(3,1), C(2,1), C(1,1)
write(10,*) "using built-in functions"
write(10,*) T(3,3), T(2,3), T(1,3)
write(10,*) T(3,2), T(2,2), T(1,2)
write(10,*) T(3,1), T(2,1), T(1,1)
!do n1=1,array1_size1
 ! write(10,*) (C(n1,n2),n2=1,array2_size2)
!enddo
write(10,*) "The commutator of A and B gives"
write(10,*) D(3,3), D(2,3), D(1,3)
write(10,*) D(3,2), D(2,2), D(1,2)
write(10,*) D(3,1), D(2,1), D(1,1)
close(unit=10)


end
```

Conclusion:

We are able to calculate the product and commutator using matmul and loops

```
Matrix multiplication of A and B gives
using logic and loops
   12.000000000000000        11.000000000000000        10.000000000000000
   8.0000000000000000        9.0000000000000000        10.000000000000000
   5.0000000000000000        5.0000000000000000        5.0000000000000000
using built-in functions
   12.000000000000000        11.000000000000000        10.000000000000000
   8.0000000000000000        9.0000000000000000        10.000000000000000
   5.0000000000000000        5.0000000000000000        5.0000000000000000
The commutator of A and B gives
   7.0000000000000000        6.0000000000000000        5.0000000000000000
  -2.0000000000000000        0.0000000000000000        2.0000000000000000
  -5.0000000000000000       -6.0000000000000000       -7.0000000000000000
```





06

Projectile Motion

```fortran
implicit none
double precision :: t_final   !total time of propagation
double precision, parameter :: dt=1.0d-1            !time step in seconds
integer :: Nt!=int(t_final/dt)            !total number of time steps
double precision, parameter :: g=9.8                !free-fall accelleration in m/s**2
double precision, parameter :: x0=0.0,y0=0.0        !initial position
double precision :: v0=35.0,angle=68.0,theta   !initial speed [in m/s] and angle [in degrees]
double precision t,x,y,v0x,v0y,vy,pi
double precision x_ex,y_ex,vy_ex,t_ex
integer n

pi=4.0*atan(1.0)  !calculate pi
theta = angle * pi / 180.0 !convert theta to radian

v0x=v0*cos(theta) !initial velocity along x --- doesn't change since we neglect air resistanc
v0y=v0*sin(theta) !initial velocity along y

! find the time of flight
! reformulate will give us t = 2v_0sin(theta) /g analytically
t_final = (2 * v0y) / g

Nt=int(t_final/dt)  !total steps

```

Define all values first

Use Euler's form to find the motion

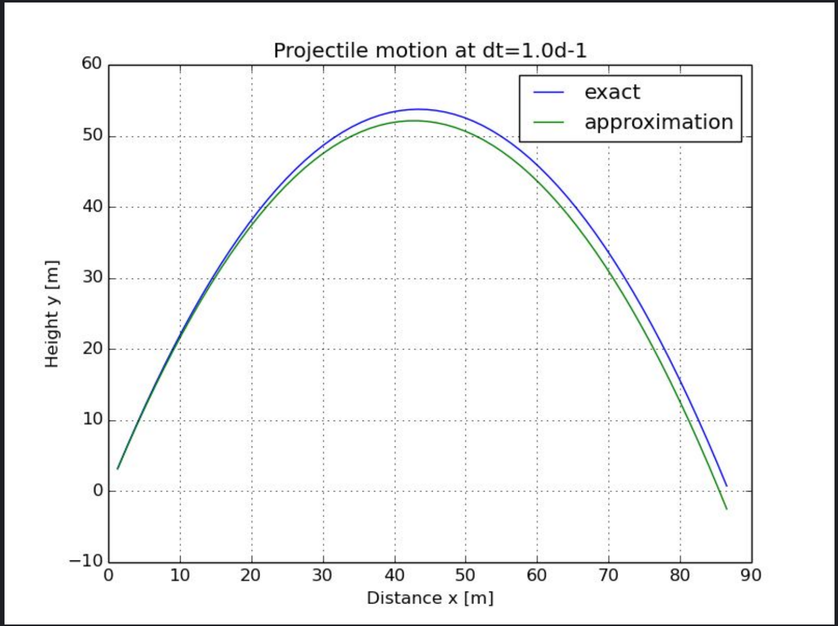
```fortran
!open files
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/x_t.dat',unit=10)
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/y_t.dat',unit=11)
open(file='/Users/phihung/Documents/PHY/first/homework/hw_01/vy_t.dat',unit=12)

n=0.0
! set initial conditions for exact calculation as well
x=x0
y=x0
vy=v0y

do while (x>=0.0 .and. y>=0.0) !propagate until the projectile hits the ground

  !numerical
  x=x+v0x*dt !update the x position
  vy=vy-g*dt !update velocity in y dir
  y=y+vy*dt !update the y position

  !exact
  n=n+1 !update the iteration
  t=dt*float(n) !find time passed
  x_ex=x0+v0x*t !find the exact x location
  vy_ex=v0y-g*t !update the exact y speed
  y_ex=y0+v0y*t-0.5*g*(t**2) !update the exact y location

  !write the files
  write(10,*) t,x,x_ex
  write(11,*) t,y,y_ex
  write(12,*) t,vy,vy_ex

enddo

close(unit=10)
close(unit=11)
close(unit=12)

write(*,*) 'exact time of flight', t_final
write(*,*) 'calculated time of flight',t
end
```

Create plots

I created plots at different dt to check the convergence value

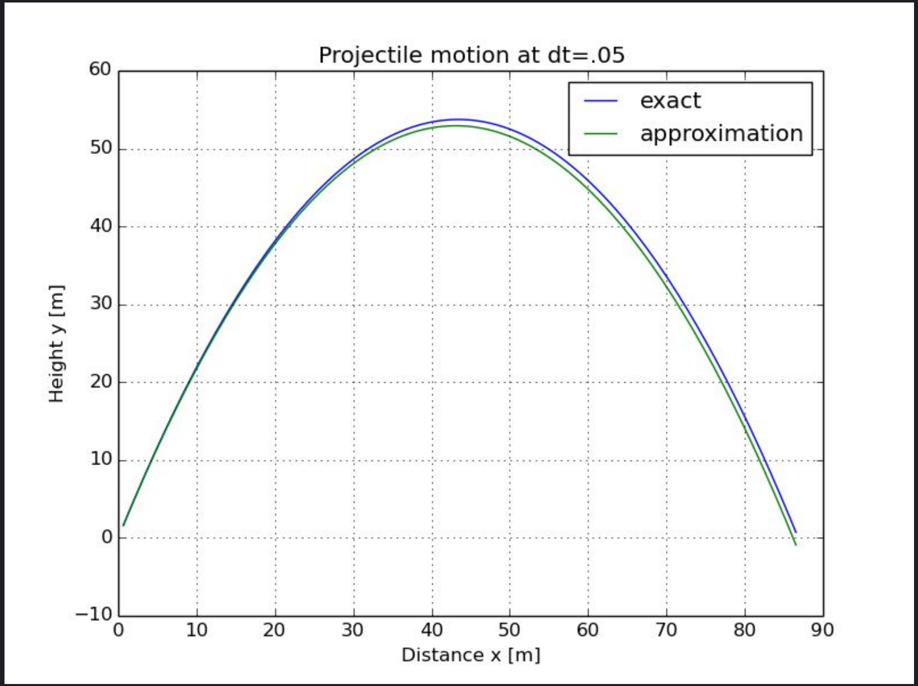
```python
import matplotlib.pyplot as plt
import numpy as np

#load data
x_data = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/x_t.dat')
y_data = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/y_t.dat')
v_data = np.loadtxt('/Users/phihung/Documents/PHY/first/homework/hw_01/vy_t.dat')

#plot graph
plt.plot(x_data[:,2],y_data[:,2], label='exact')
plt.plot(x_data[:,1],y_data[:,1],label='approximation')
plt.title(r'Projectile motion at dt=1.0d-1')

plt.xlabel('Distance x [m]')
plt.ylabel('Height y [m]')

plt.grid(True)
plt.legend()
plt.show()
```

dt=1.0d-1




```
exact time of flight    6.6227417778082049
calculated time of flight    6.6000000000000005
```

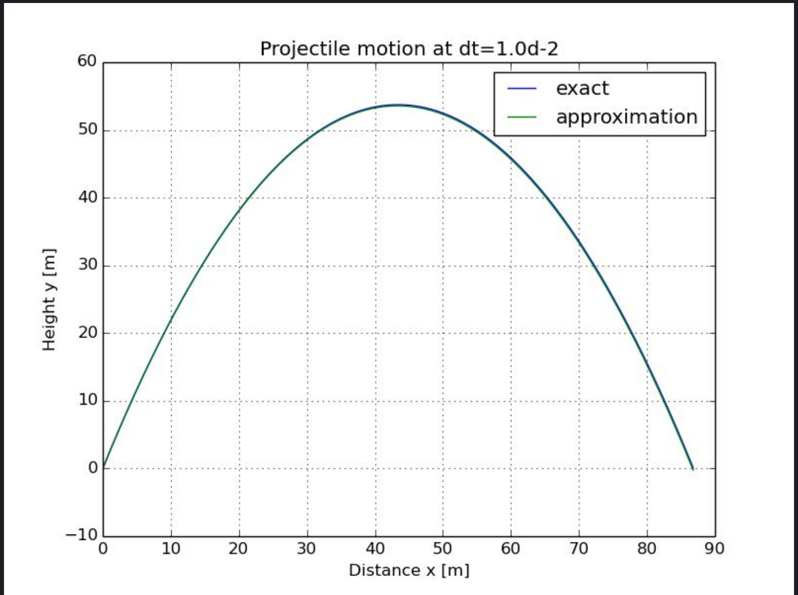
dt=.05




```
exact time of flight   6.6227417778082049
calculated time of flight  6.6000000983476639
```

dt=1.0d-2

```
exact time of flight    6.6227417778082049
calculated time of flight  6.6200000000000001
```



Show an obvious convergence

Conclusion:

dt=1.0d-2 seems to be a good time step for converging