Computational Astrophysics

ASTR 660, Spring 2021
計算天文物理

Lecture 2

Instructor: Prof. Kuo-Chuan Pan
kuochuan.pan@gapp.nthu.edu.tw

Class website

https://kuochuanpan.github.io/courses/109ASTR660_CA/

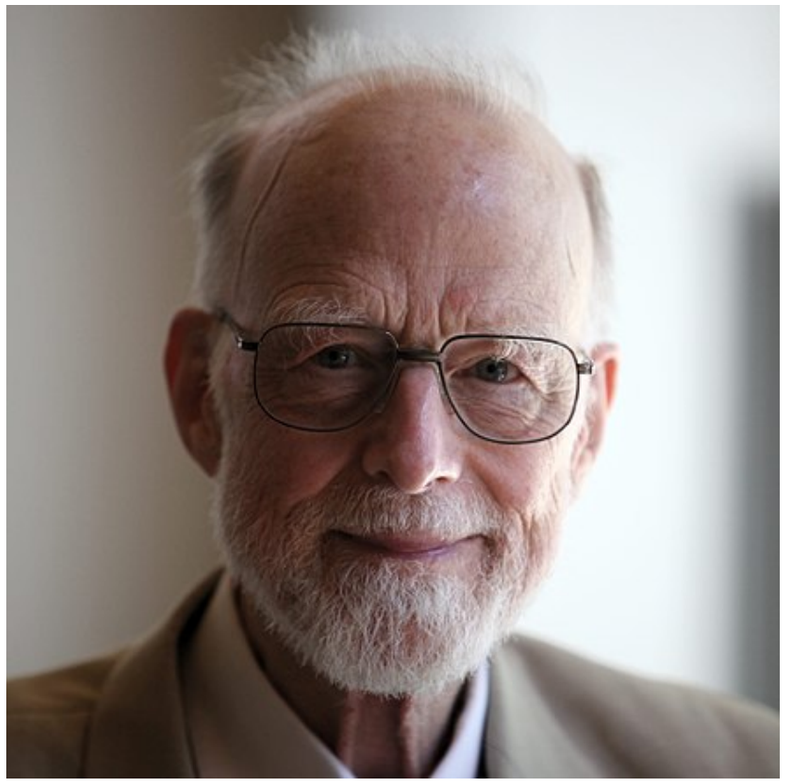
Plan for today

- Introduction to Fortran
- Fortran practices
- Fortran projects

0. Demo
1. PI calculation
2. Angry bird
3. Binary stars

Introduction to Fortran

Introduction to Fortran



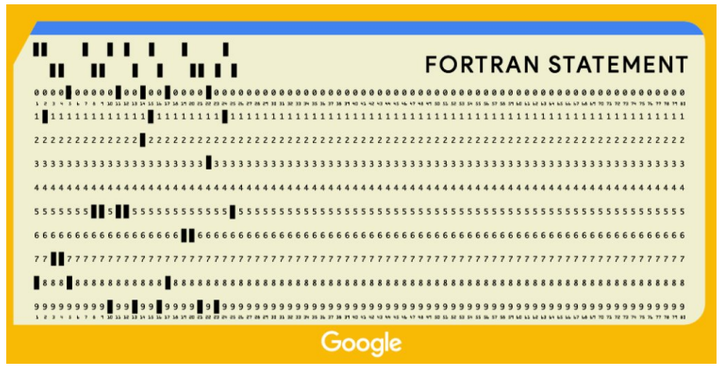
"I don't know what the language of the year 2000 will look like, but I know it will be called FORTRAN."

—Tony Hoare, winner of the 1980 Turing Award, in 1982.

Introduction to Fortran

- Fortran/FORTRAN (Formula Translation)
- Designed for numeric computation and scientific computing

- **FORTRAN** (**1954**, for the IBM 704, 32 statements)
- FORTRAN II (1958)
- FORTRAN III
- FORTRAN IV
- FORTRAN 66 (1966)
- **FORTRAN 77** (1977, structured programming)
- **Fortran 90** (modular programming; free-format)
- Fortran 95
- **Fortran 2003** (object-oriented programming)
- Fortran 2008 (concurrent programming)



C (1972)
C99 (199)
C11 (2011)
C++ (1998)
C++ (11/14/17)

Introduction to Fortran

- Fortran/FORTRAN (Formula Translation)
- Designed for numeric computation and scientific computing

- ~~**FORTRAN** (**1954**, for the IBM 704, 32 statements)~~
- ~~FORTRAN II (1958)~~
- ~~FORTRAN III~~
- ~~FORTRAN IV~~
- ~~FORTRAN 66 (1966)~~
- ~~**FORTRAN 77** (1977, structured programming)~~
- **Fortran 90** (modular programming; free-format)
- Fortran 95
- **Fortran 2003** (object-oriented programming)
- Fortran 2008 (concurrent programming)

C (1972)
C99 (199)
C11 (2011)
C++ (1998)
C++ (11/14/17)

Introduction to Fortran

- Why Fortran?
- Why physicists/astrophysicists still use Fortran?

Introduction to Fortran

- (modern) Fortran is still the dominant language for high performance computing of physical systems (i.e. galaxies, cosmology, hydrodynamics, magneto-hydrodynamics, molecular dynamics, climate,… etc.)

- The other language is C++

Introduction to Fortran

- It is fast! On most benchmarks, Fortran and C++ are the fastest! (~ 100 times faster than python)

So, the question becomes: Why not C++?

- Fortran has legacy code
- Fortran is easier to learn for physics students

No “;”

Why Fortran?

$$A_{ij} = \begin{bmatrix} a_{??} & a_{??} & a_{??} \\ a_{??} & a_{??} & a_{??} \\ a_{??} & a_{??} & a_{??} \end{bmatrix}$$

Why Fortran?

integer, dimension(3,3) : : a

int a[3][3]

$$A_{ij} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$A_{ij} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Fortran

C/C++

`C=A*B` gives an element-by-element multiplication of A and B

Need a **for** loop

It has features for scientific computing

- Array handling features:

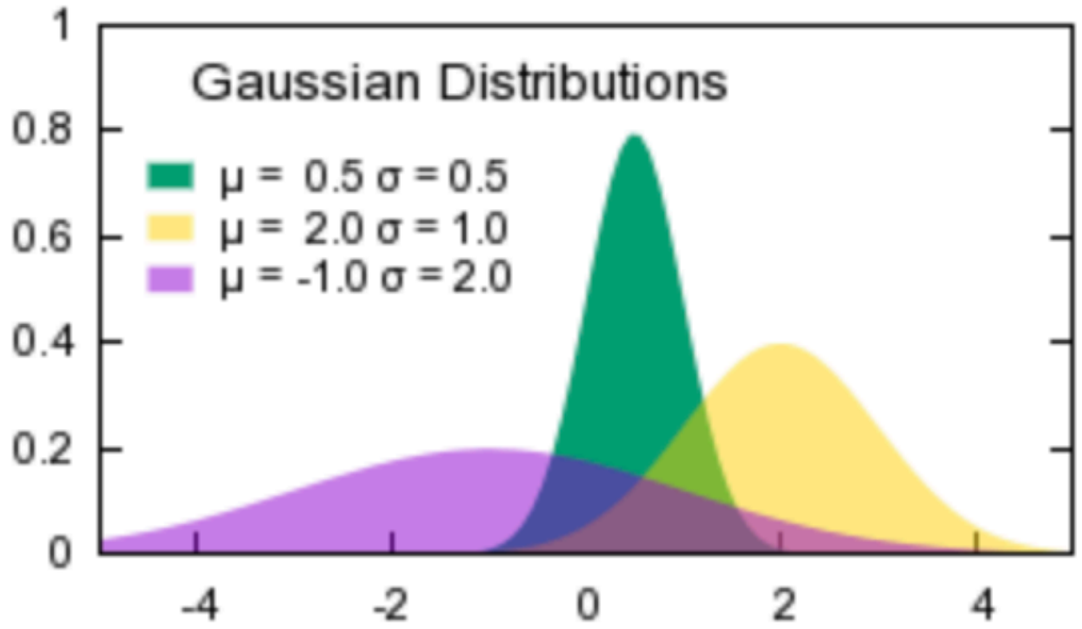
```
c = a * b

where (array .lt. 0.0) array = 0.0

double precision, dimension(-1:10) :: array

double precision, dimension(:,:), allocatable :: array_2d
allocate(array_2d(xdim, ydim))
```

And more …. (we will talk about later)

Demo and Exercise: Fortran

Install: gnuplot

Or use your favorite plotting software



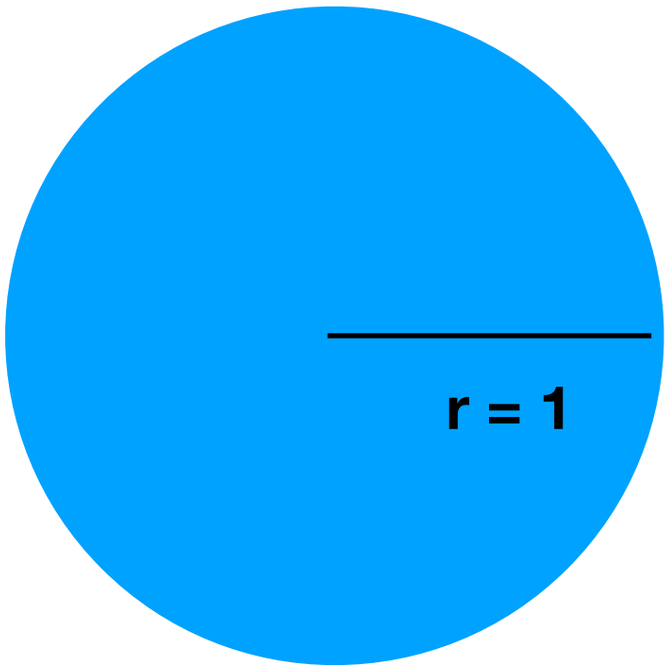
Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms.

WSL users: you might need to install Xming, Cygwin X, or vcXsrv for x11

Linux: `sudo apt-get install gnuplot`

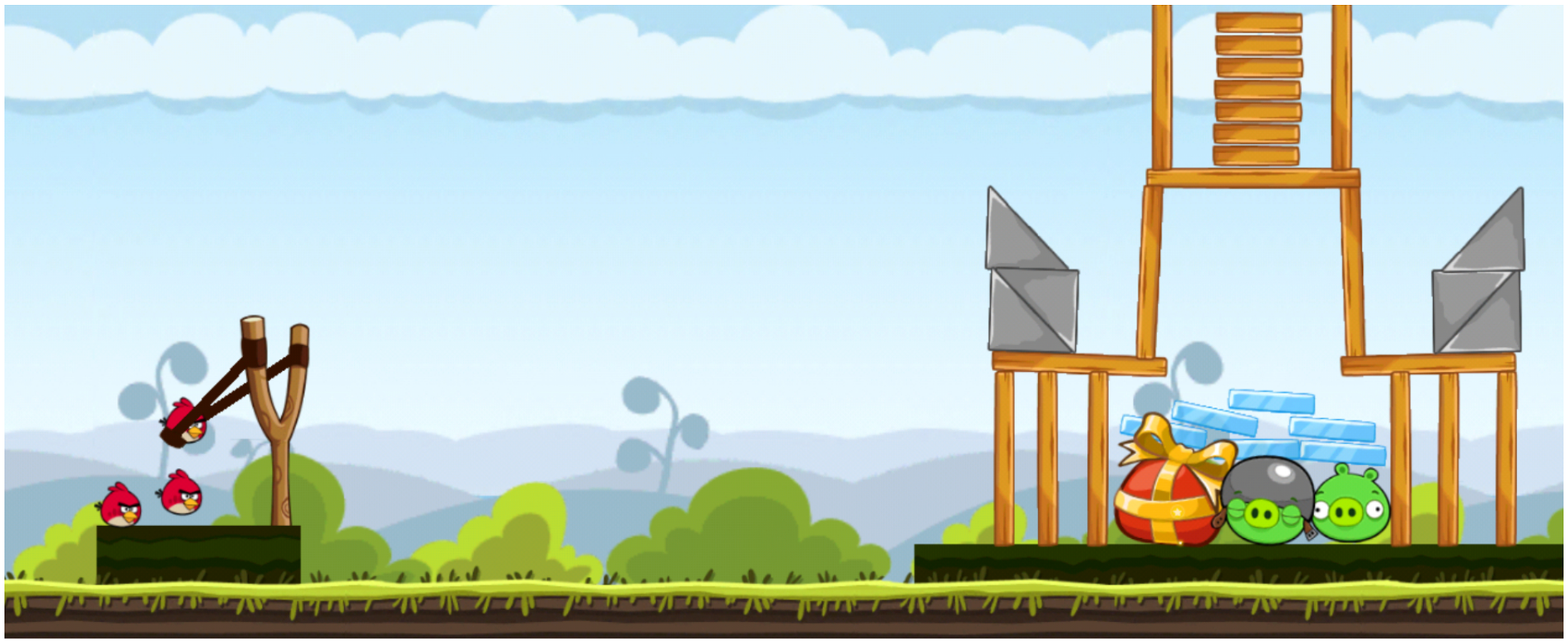
Mac: `brew install gnuplot`

Exercise 1: Pi



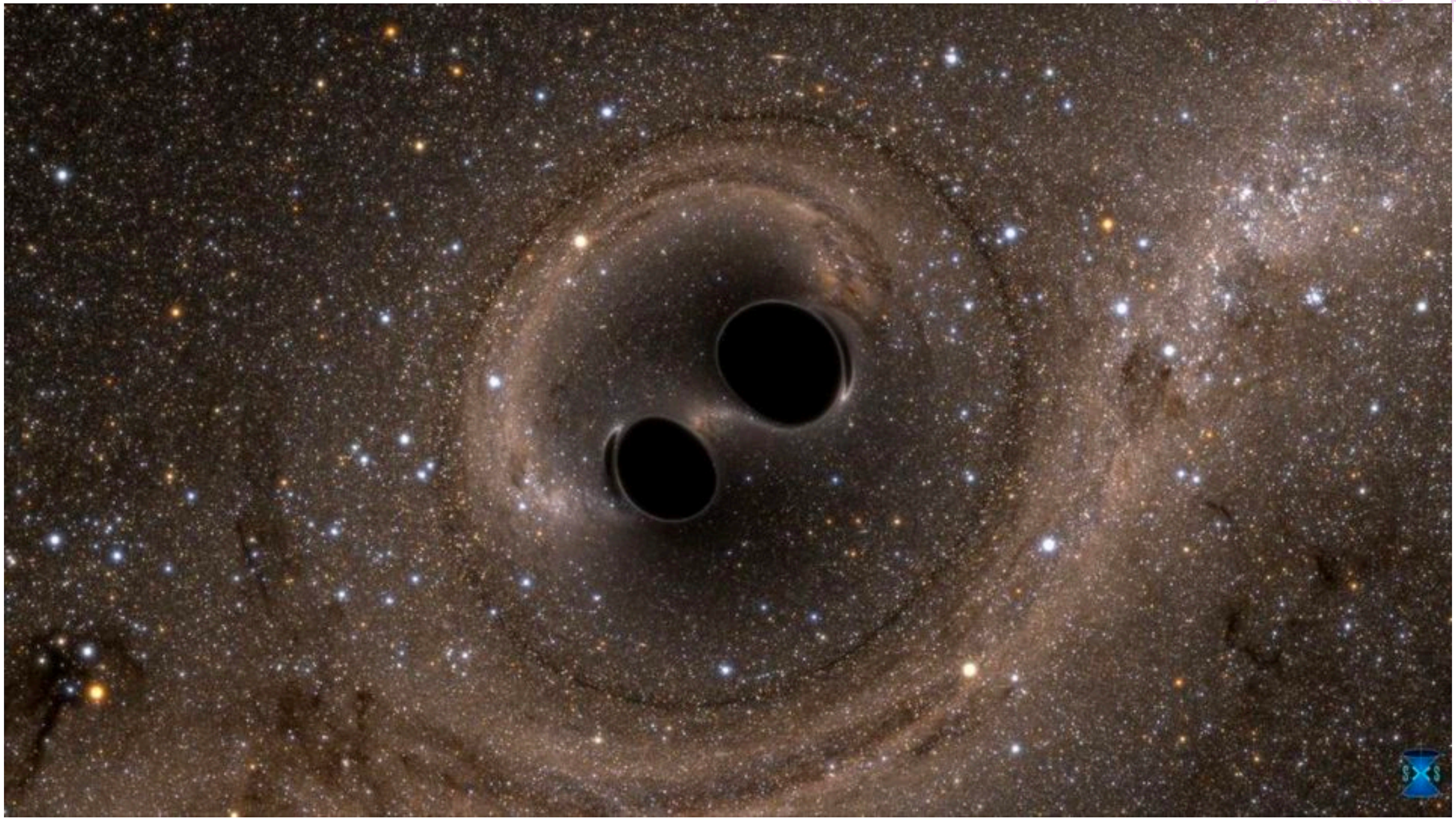
Calculate PI by evaluating the area of a circle with radius = 1

Exercise 2: Angry bird



Simulate the trajectory of an angry bird with given angle and initial velocity

Exercise 3: Binary Evolution



Demo 1: Hello world in Fortran

- Create a file named hello.f90

```
1 program hello_world
2     print *, "Hello World!"
3 end program hello_world
4
```

- Compile the code by **gfortran hello.f90**
- Execute the program by **./a.out**
- If you don't like "a.out", then try **gfortran hello.f90 -o hello**

Demo 1: Hello world

- Add some comments in your code

```
!-------------------------------
!
! The hello world program
!
! Kuo-Chuan Pan 2020.03.12
!
!-------------------------------
program hello_world
    ! this is a comment
    print *, "Hello World!"
end program hello_world

```

Demo 2-3: Variables

```fortran
program data_type
    implicit none
    integer*2 :: short
    integer*4 :: long
    integer*8 :: verylong
    integer*16 :: veryverylong

    real*4  :: real   = 1.0
    real*8  :: double = 1.0
    real*16 :: quad   = 1.0

    print *, "Integers:"

    print *, huge(short)
    print *, huge(long)
    print *, huge(verylong)
    print *, huge(veryverylong)

    print *, "Real numbers:"
    print *, real
    print *, double
    print *, quad

end program data_type
```

```fortran
program data_type
    implicit none
    integer :: int
    real    :: rel = 1.0

    print *, "Integers:"
    print *, huge(int)

    print *, "Real numbers:"
    print *, rel

end program data_type
```

- gfortran numbers.f90
- gfortran -fdefault-real-8 -fdefault-integer-8 numbers.f90

Demo 4: Variables and parameters


```
1  program data_type
2      implicit none
3      logical       :: is_person
4      character*40  :: name
5      complex       :: complex_var
6      real, parameter :: msun = 1.989e33 ! a constant
7
8      name        = "Kuo-Chuan Pan"
9      is_person   = .true.
10     complex_var = (3,5)
11
12     print *, name
13     print *, is_person
14     print *, complex_var
15     print *, msun
16
17 end program data_type
18
```

Demo 5: Implicit Typing


```
1 program variables
2     !
3     ! implicit typing [NOT recommend]
4     !
5     implicit double precision(A-H,O-Z)
6
7     a = 1.0
8     b = 2.5
9     c = a + b
10
11    print *, "c = a + b =", c
12
13    i = 1.0
14    j = 2.5
15    k = i + j
16
17    print *, "k = i + j =", k
18
19 end program variables
20
```

Demo 6: Formatted Output


```fortran
program output
    implicit none
    type Star
        character(len = 10) :: name
        integer(kind=4)     :: id
        double precision    :: age
        double precision    :: distance
        double precision    :: magnitude
        logical             :: is_double
    end type Star

    character*40                :: file_name
    character*4                 :: id_str
    integer, parameter          :: nstars = 100

    type(Star), dimension(nstars) :: stars
    integer :: n

    file_name = "stars.txt"
    ! create the file
    open(unit=1,file=trim(file_name))

    ! write the header
    write(1,11) "# ", "Name", "ID", "double", "Age", "Distance", "Magnitude"
    write(1,11) "# ", " ", " ", " ", "[yr]", "[pc]", "[]"

    do n = 1, nstars
        ! generate the data
        write(id_str, 10) n
        stars(n)%name      = "Star "//id_str
        stars(n)%id        = n
        stars(n)%age       = sqrt(real(n)**3)
        stars(n)%distance  = real(n)**2 - 1.
        stars(n)%magnitude = real(n)**2.5
        stars(n)%is_double = (mod(n,5) .eq. 3)

        ! write to the file
        write(1,12) stars(n)%name, stars(n)%id, stars(n)%is_double, stars(n)%age, stars(n)%distance, stars(n)%magnitude
    enddo
    close(1)

10  format(I4)
11  format(a2, a10, a5, a7, 3a24)
12  format(2x, a10, i5, l7, 3e24.14)

end program output
```

Formatted Output

Purpose		*Edit Descriptors*	
Reading/writing **INTEGER**s		**I***w*	**I***w.m*
Reading/writing **REAL**s	Decimal form	**F***w.d*	
	Exponential form	**E***w.d*	**E***w.d***E***e*
	Scientific form	**ES***w.d*	**ES***w.d***E***e*
	Engineering form	**EN***w.d*	**EN***w.d***E***e*
Reading/writing **LOGICAL**s		**L***w*	
Reading/writing **CHARACTER**s		**A**	**A***w*
Positioning	Horizontal	***n*X**	
	Tabbing	**T***c*	**TL***c* and **TR***c*
	Vertical	**/**	
Others	Grouping	***r*(....)**	
	Format Scanning Control	**:**	
	Sign Control	**S**, **SP** and **SS**	
	Blank Control	**BN** and **BZ**	

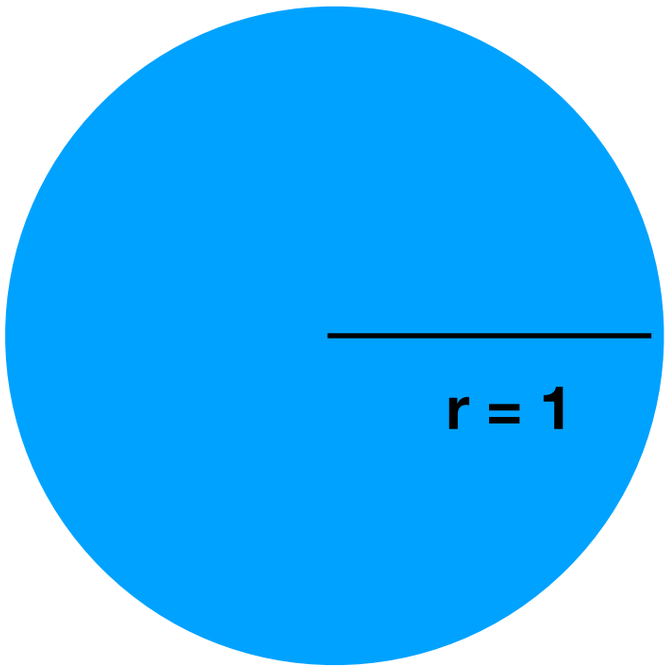
- ***w***: the number of positions to be used
- ***m***: the minimum number of positions to be used
- ***d***: the number of digits to the right of the decimal point
- ***e***: the number of digits in the exponent part

Demo 7: The “goto” statement

```
10 40  print *, "to"
11     goto 15
12
13 30  print *, "powerful,"
14     goto 22
15
16 22  print *, "but"
17     go to 20
18
19 20  print *, "very"
20
21     if (i .eq. 0) then
22         goto 30
23     else
24         goto 40
25     endif
```

Not Prof. Goto !!!

Exercise 1: Pi

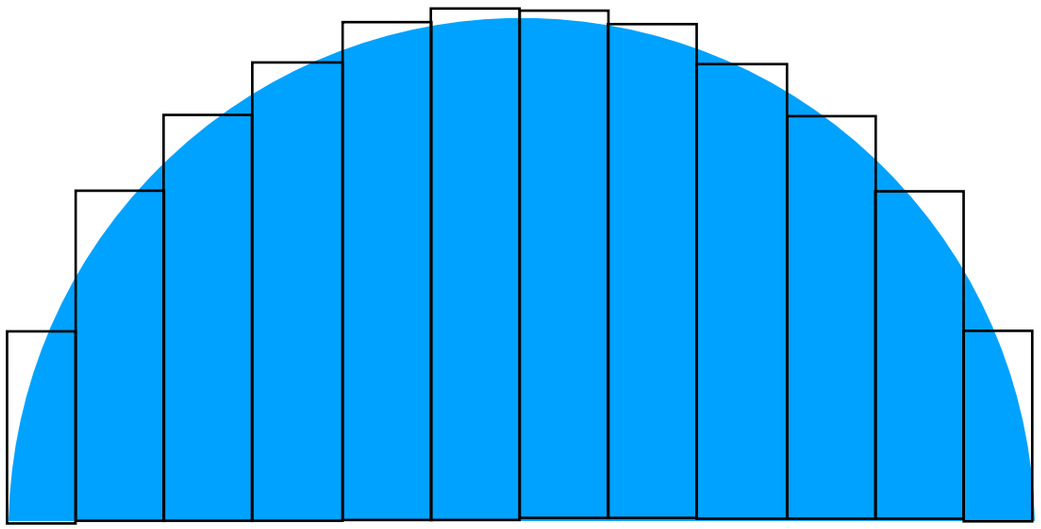


Calculate PI by evaluating the area of a circle with radius = 1

Exercise 1: Pi

The area of the circle can be approximated by the area of sum of these small rectangles x 2

The area of each rectangle is dA = dx * h



Step 1 See pi1.f90


```
program pi

    implicit none

    real    :: x, dx, h, dA, area
    integer :: i, N

    print *, "Type the number of partition (N) you want to make"
    read *, N

    !-- initialize values
    !   dx = (1 - (-1))/ N
    !
    dx   =
    area =

    !-- calculate each rectangle and sum the area
    do i = 1, N
       x =
       h =
       dA =
       area = area + dA
    enddo

    !-- print out the result
    print *, "PI = ", 2.*area


end program pi
```

Step 2, use function

See pi2.f90

```
real     :: my_func
```

```
h = my_func(x)
```

```
real function my_func(x)
    ! the function return the y values of a half circle with radius = 1
    real :: x
    my_func = sqrt(1.0 - x**2)
    return
end function
```

Step 3, use subroutine

See pi3.f90

```
  ! do integral
  call calculate_function_integral(N, area)
```

```
subroutine calculate_function_integral(N, A)
    implicit none
    integer, intent(in)  :: N
    real,    intent(out) :: A

    integer :: i
    real    :: my_func
    real    :: x, h, dx, dA

    !-- initialize values
    !   dx = (1 - (-1))/ N
    !

    dx  = 2.0 / N
    A   = 0.0      ! note that a and A are the same in fortran

    !-- calculate each rectangle and sum the area
    do i = 1, N
       x =
       h =
       dA =
       A  =
    enddo
    return
end subroutine calculate_function_integral
```

Step 4, convergence

See pi4.f90

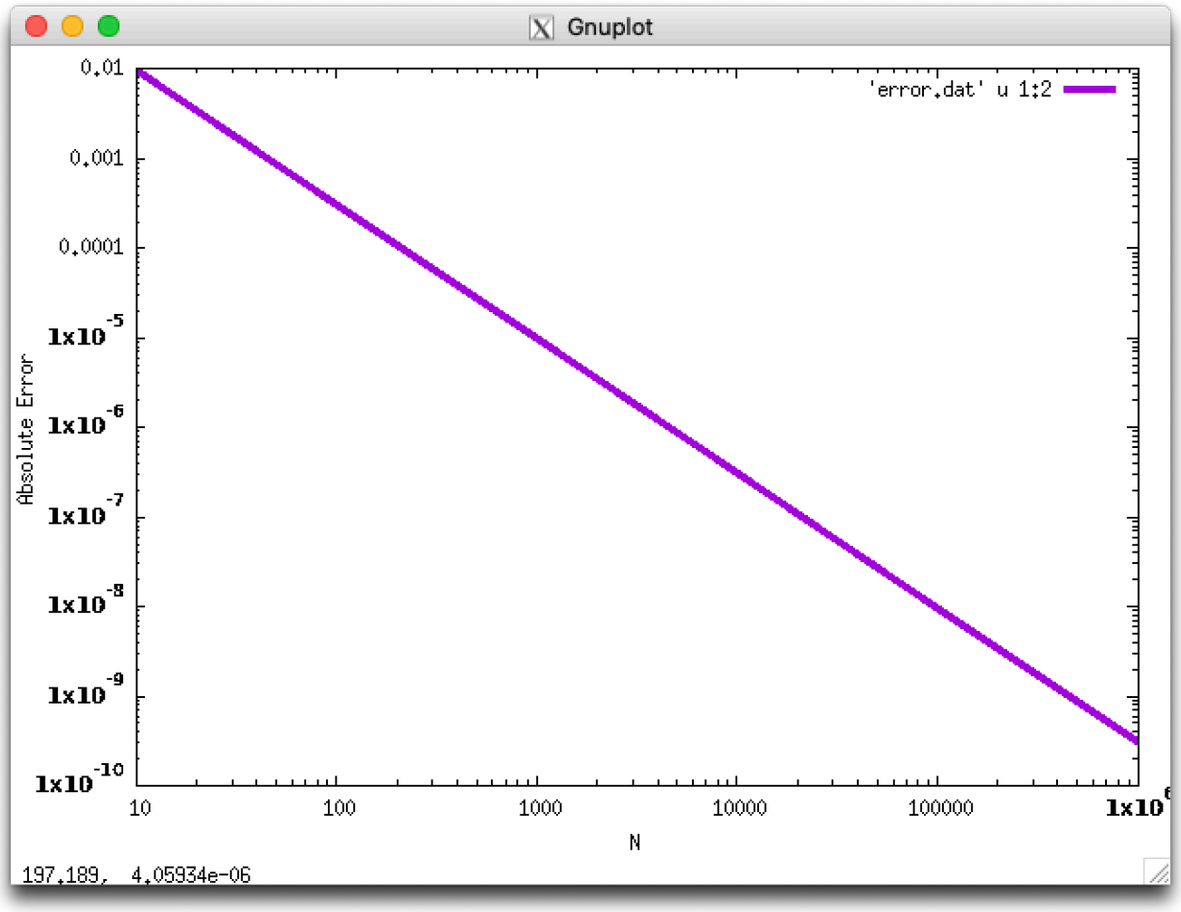
Check the error of your calculation with different N

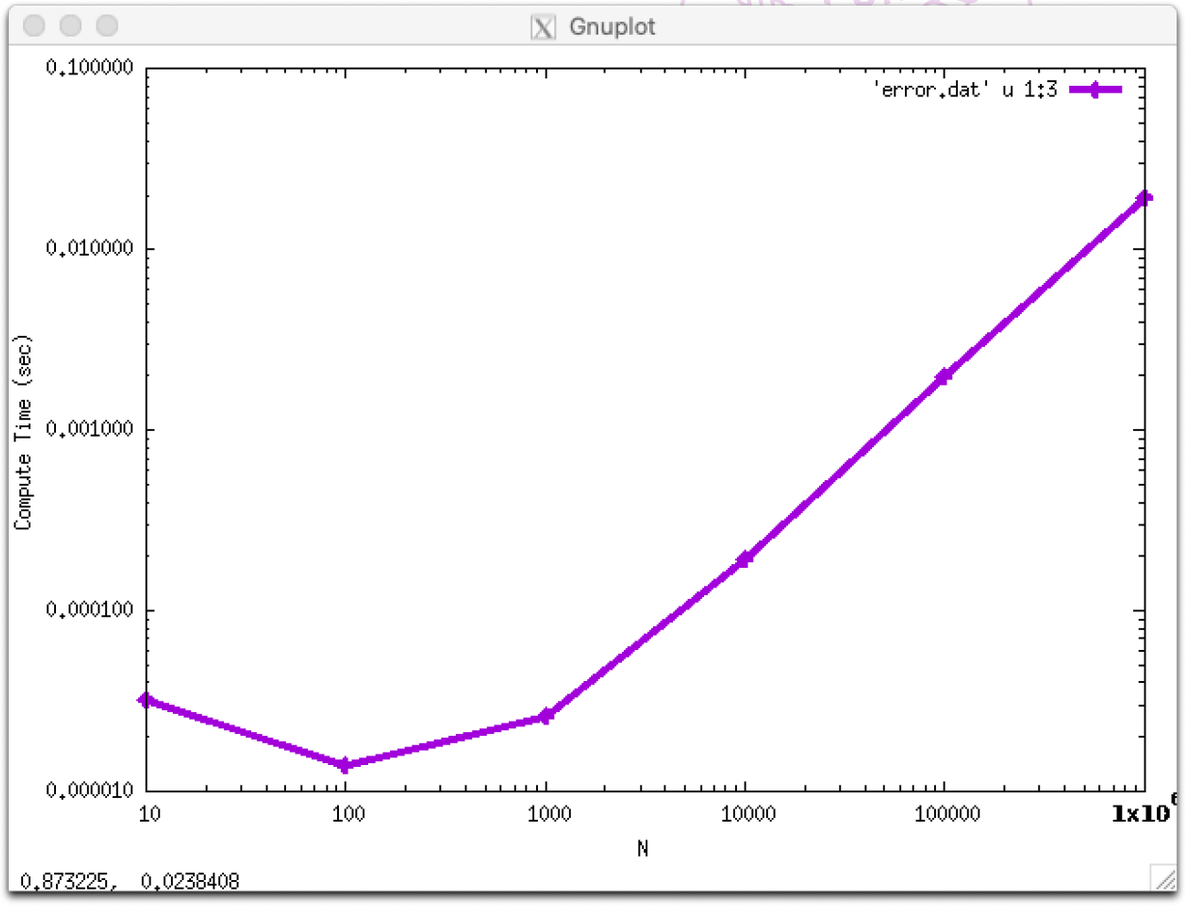
You could reuse the “calculate_function_integral” subroutine

```
integer, dimension(MAXN) ::  numbers

!--- setup numbers
numbers = (/10, 100, 1000, 10000, 100000, 1000000/)
```

```
do i=1, MAXN
    n = numbers(i)
    call calculate_function_integral(n, area)
    print *, "N = ",n, " PI = ", 2.*area
    error = abs(2.*area - pi)/pi

    call cpu_time(t2)
    write(11,200) numbers(i), error, (t2-t1)
    t1 = t2
enddo
```

Exercise 1: Pi





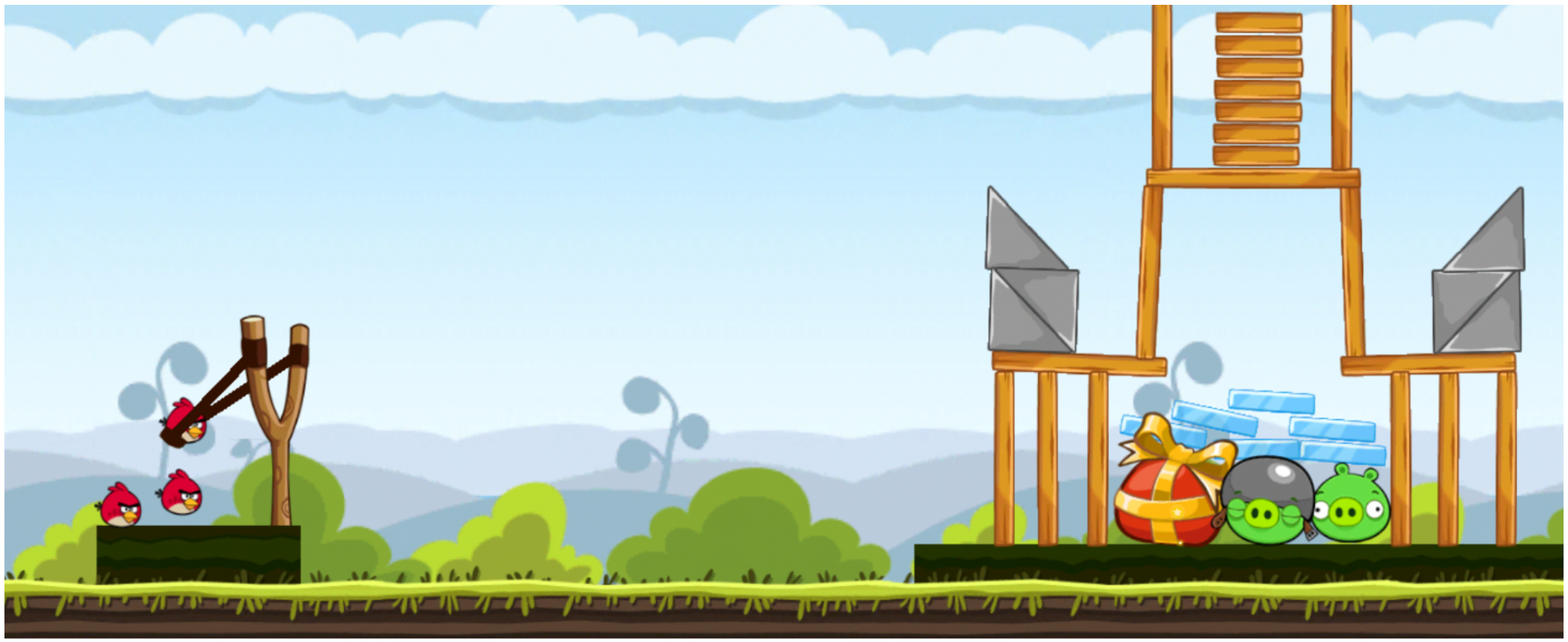
gnuplot
set logscale
plot 'error.dat' u 1:2 w l

Debugging

General tips:

- Reduce the problem …
- Convert the problem into an automated test …
- Don’t assume things work
- Be clear in mind about correct behavior (the physical meaning)
- Fix one problem at a time
- Ask your friend to have a look ….

Exercise 2: Angry bird

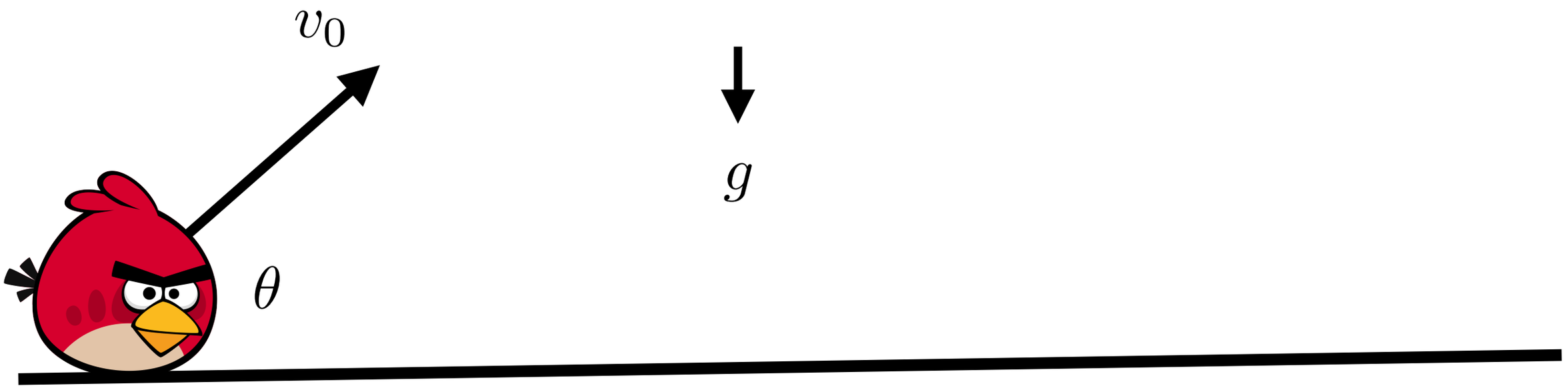


Simulate the trajectory of an angry bird with given angle and initial velocity

Exercise 2: Angry bird

- We know the analytical solution

$$x = (v_0 \cos\theta)t$$
$$y = (v_0 \sin\theta)t - \frac{1}{2}gt^2$$

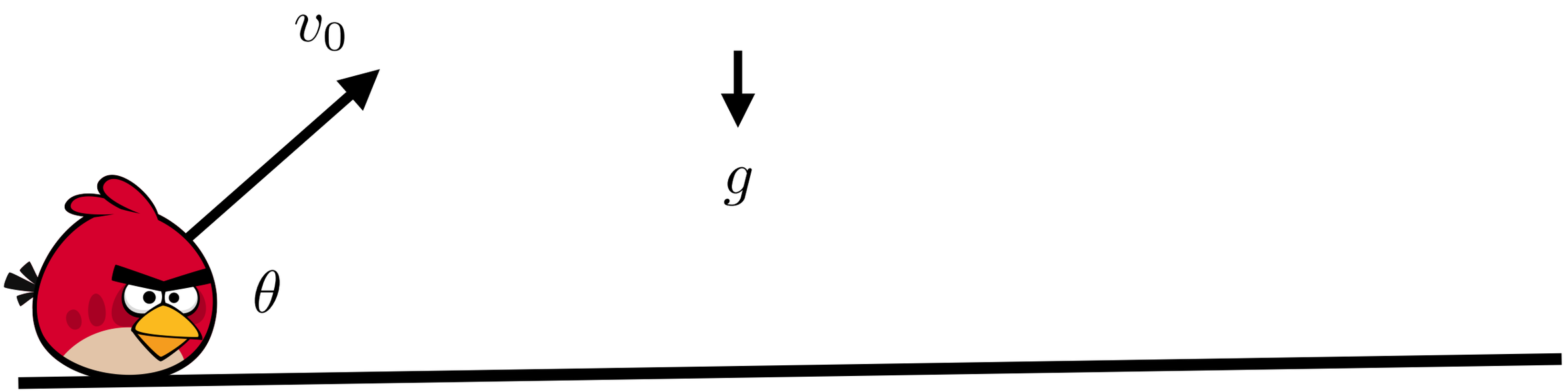


Exercise 2: Angry bird

- We could also start from Newton's laws

$$\frac{dv_x}{dt} = a_x = 0 \qquad \frac{dx}{dt} = v_x$$

$$\frac{dv_y}{dt} = a_y = -g \qquad \frac{dy}{dt} = v_y$$

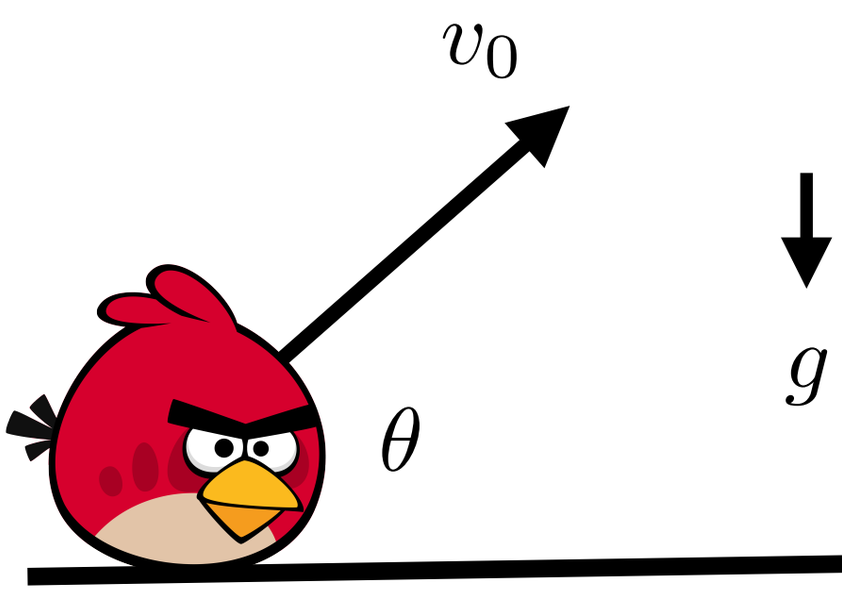


Exercise 2: Angry bird

- Approximate $$\frac{dx}{dt} = \frac{\Delta x}{\Delta t} = v_x$$

- Use a first-order finite difference scheme (Euler method)

$$x^{t^{n+1}} = x^{t^n} + v_x^{t^n} \times \Delta t$$



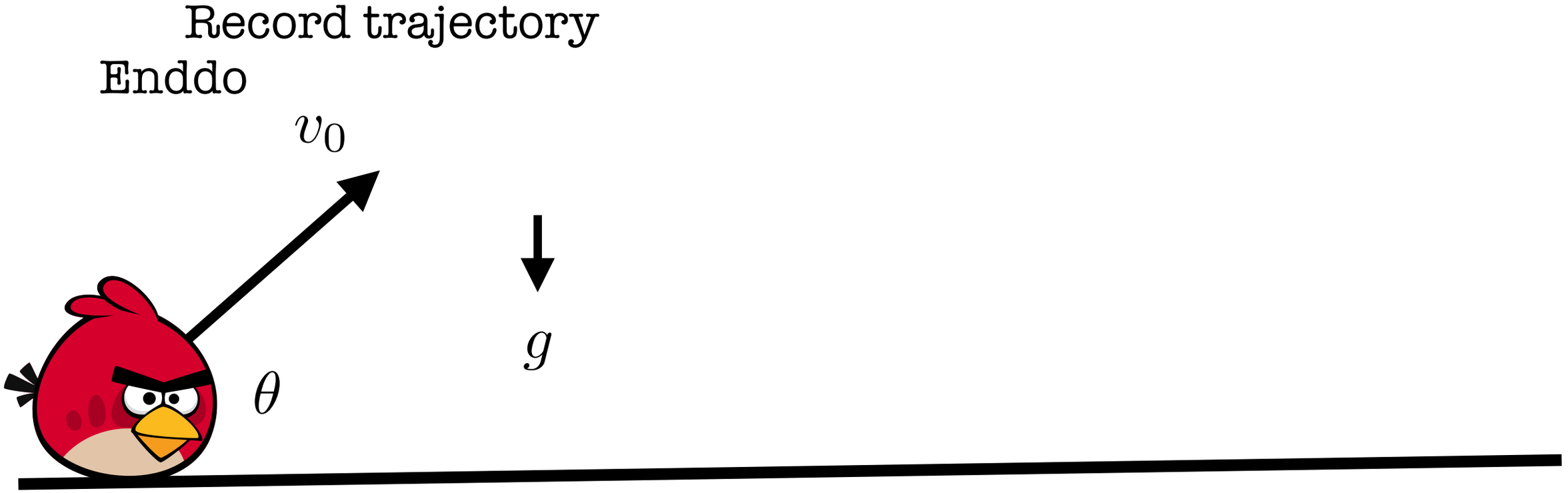
$$y^{t^{n+1}} = y^{t^n} + v_y^{t^n} \times \Delta t$$

$$v_x^{t^{n+1}} = v_x^{t^n} + a_x^{t^n} \times \Delta t$$

$$v_y^{t^{n+1}} = v_y^{t^n} + a_y^{t^n} \times \Delta t$$

Exercise 2: Angry bird


```
Initialize time, x0,y0, vx0, vy0, ax0, ay0
Do while (y >= 0 )
      update x and y
      update vx and vy
      update ax and ay (if necessary)
      update time
      Record trajectory
Enddo
```



Exercise 2: Module

Program
use constants
use physics
Implicit none
Initialize
Do while (y >= 0)
    update()
Enddo
End program

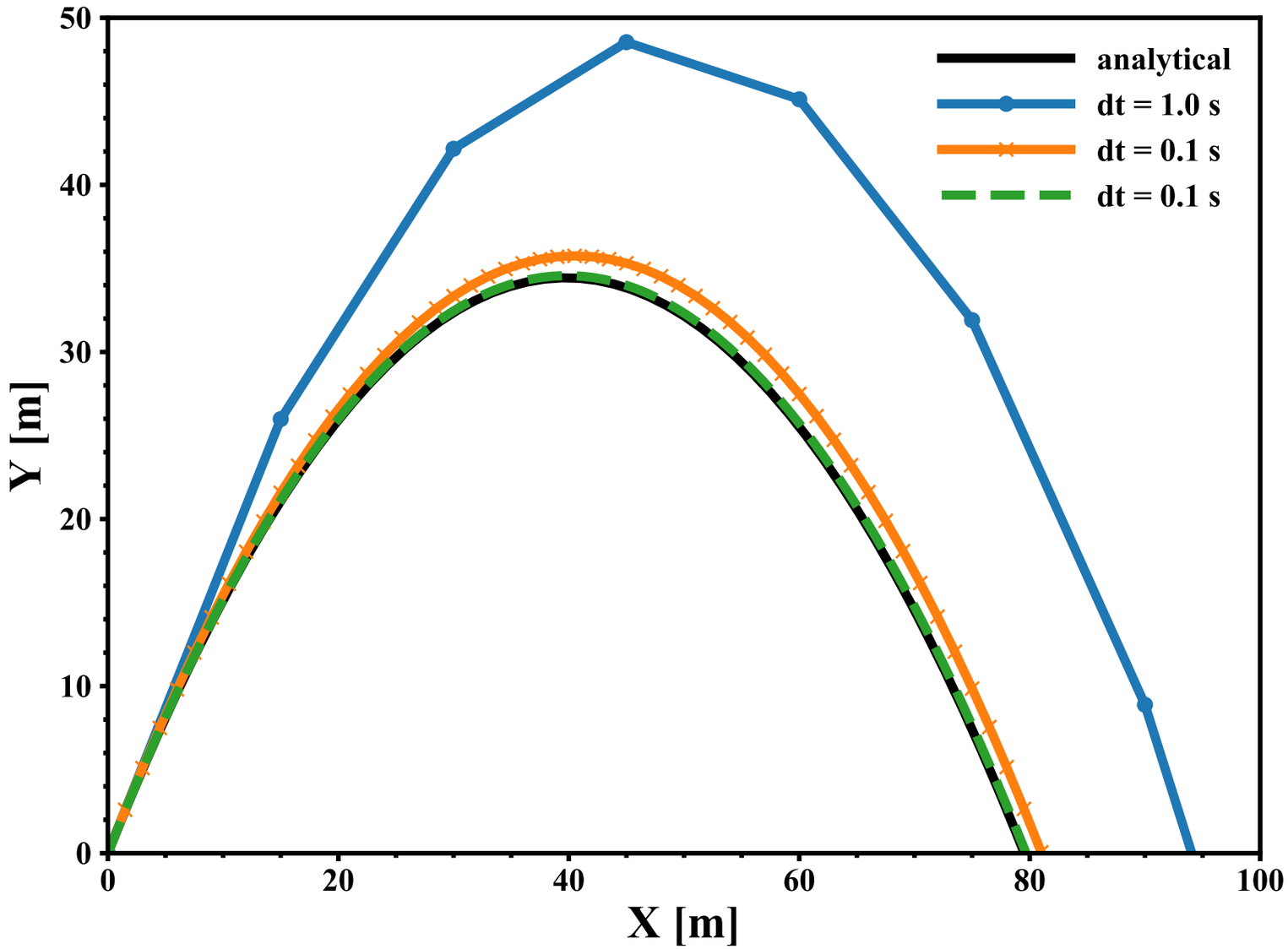
```f90
module constants

    implicit none

    real, parameter :: c = 2.99792458e8 ! m/s
    real, parameter :: g = 9.8          ! m/s/s
    real, parameter :: pi = 4.0*atan(1.0)

    contains

        subroutine show_constants()
            implicit none
            print *, "g  =", g
            print *, "pi =", pi

        end subroutine show_constants

end module constants
```

```f90
module physics
    implicit none
    contains
        subroutine update(dt,x0,y0,vx0,vy0,x,y,vx,vy)
            use constants, only : g
            implicit none
            real, intent(in)  :: dt,x0, y0, vx0, vy0
            real, intent(out) :: x,y,vx,vy
            x = x0 + vx0*dt
            y = y0 + vy0*dt
            vx = vx0 + 0.0*dt
            vy = vy0 - g*dt
            return
        end subroutine update

end module physics
```

Exercise 2: Reuse Module

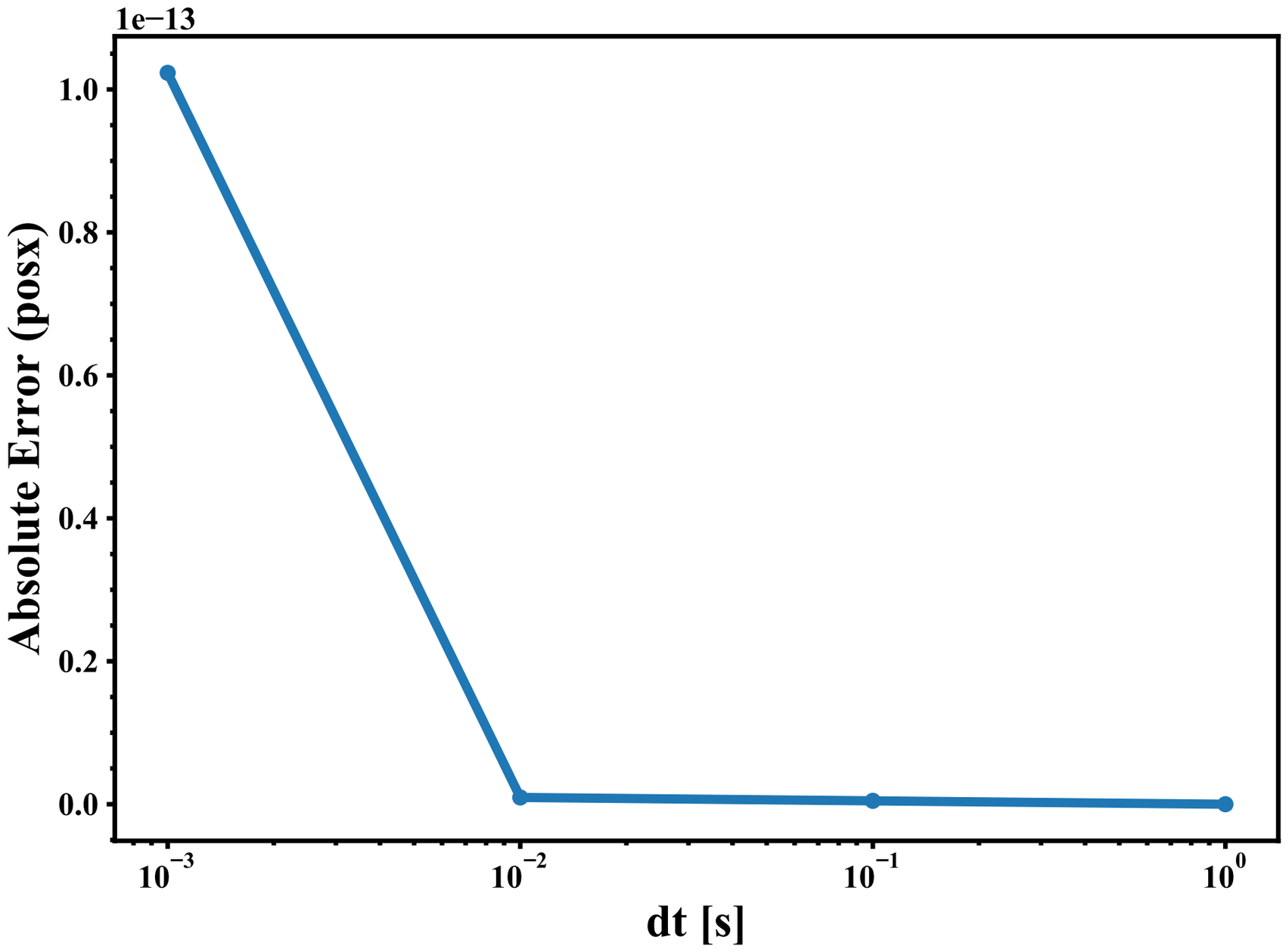
- Copy your constants and physics modules to constants.f90 and physics.f90
- Edit Angry3.f90
- Use Makefile "`make`"

Exercise 2: Angry bird

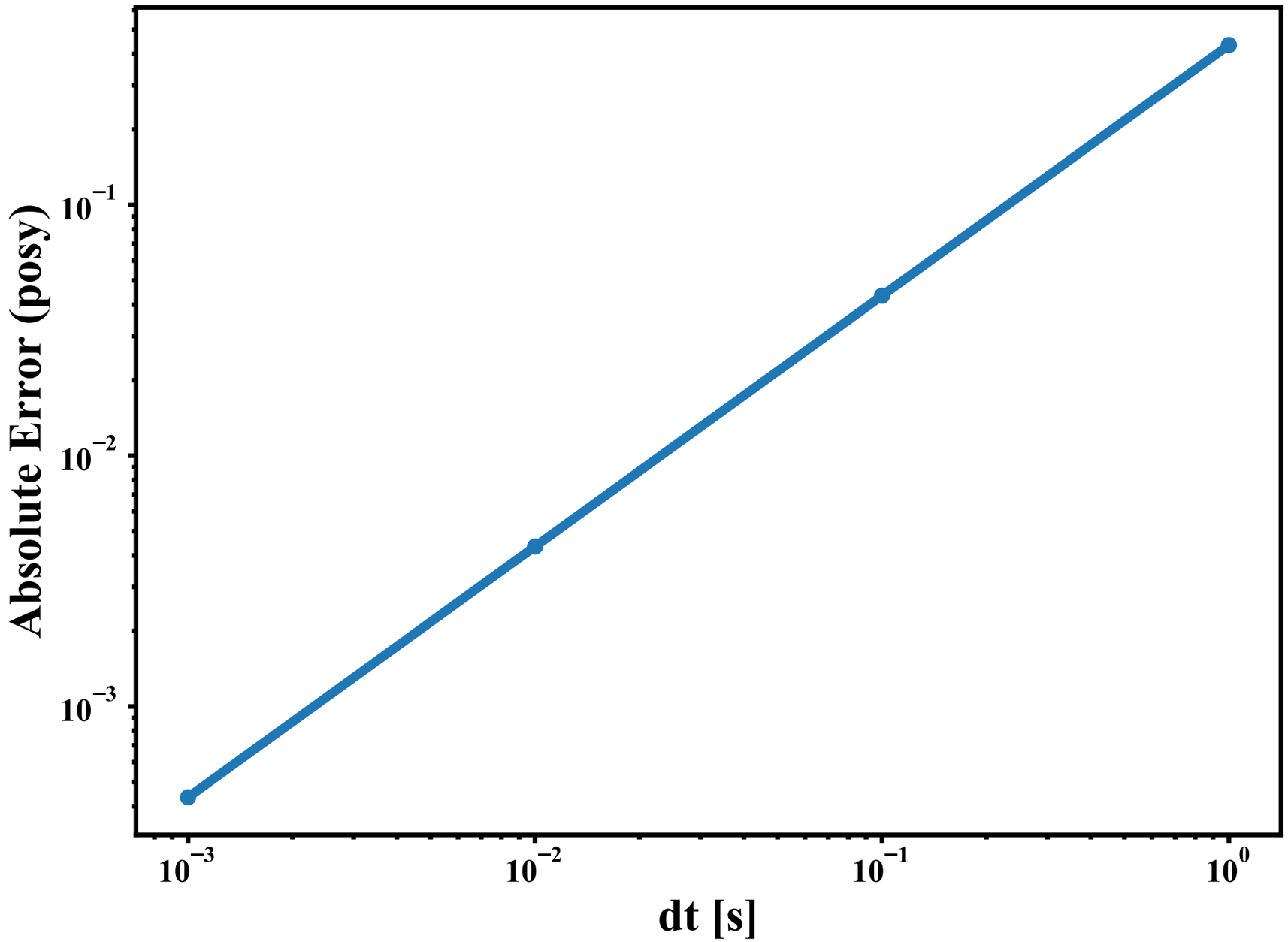


Note: don't use first order scheme in real-world application

Exercise 2: Angry bird



Exercise 2: Angry bird



Exercise 2: Air resistance

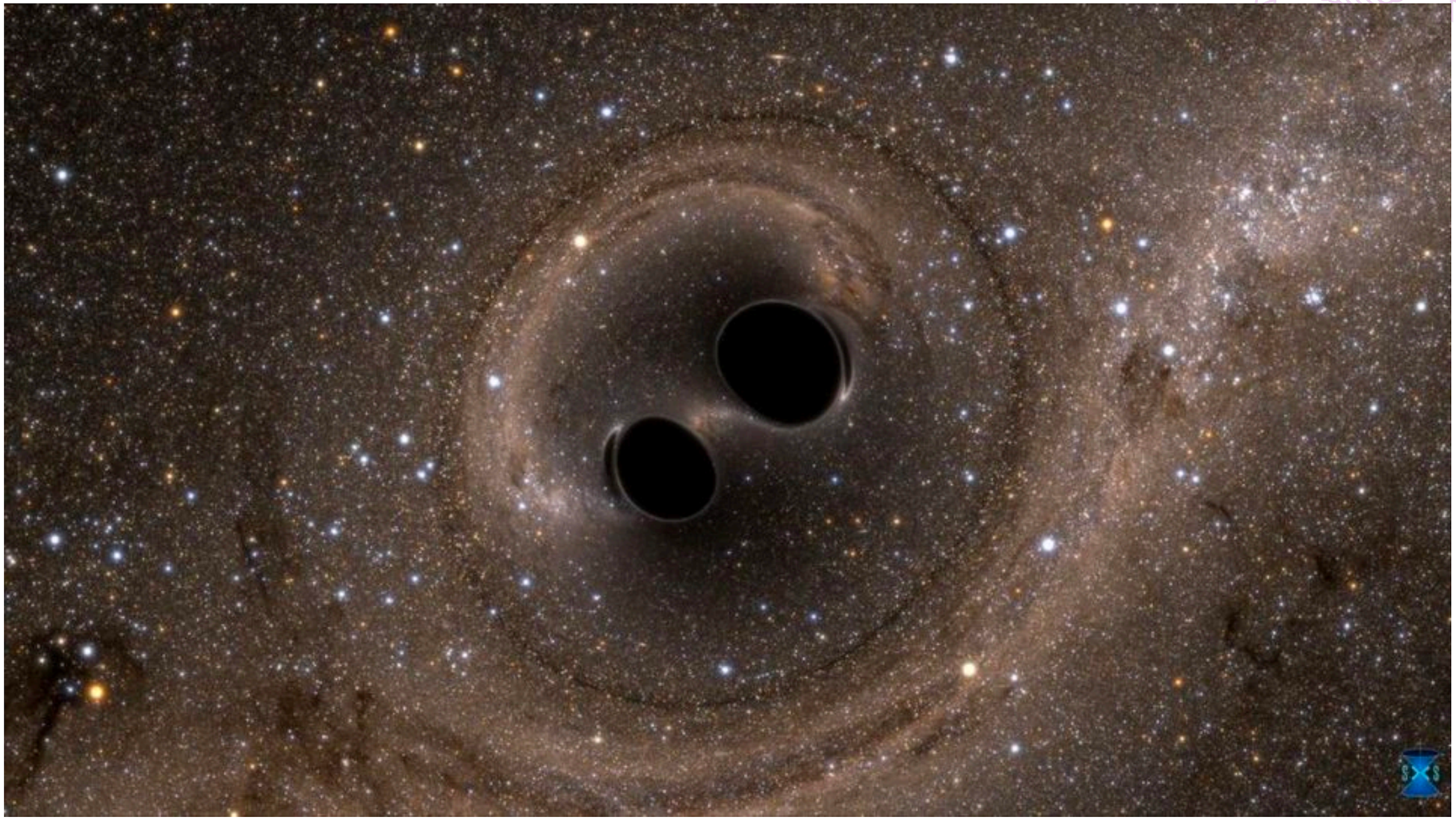
- Let's a drag force from air resistance

$$F = -K\eta v$$

Where K is the drag coefficient and η is the coefficient of viscosity

- Implement the drag force in your angry bird program and test different values of the drag coefficient and viscosity.

Exercise 3: Binary Black holes



Exercise 3: Binary Stars

- Let's write a program to simulate binary evolutions
- Assume 2D Newtonian gravity
- Assume 0 initial eccentricity
- Initial conditions:

$$G = 6.67428 \times 10^{-8} \mathrm{dyne\ cm^2 g^{-2}}$$
$$M_\odot = 1.989 \times 10^{33} \mathrm{g}$$
$$\mathrm{AU} = 1.49598 \times 10^{13} cm$$
$$M_1 = 1.0 M_\odot$$
$$M_2 = 2.0 M_\odot$$
$$a = 3 \mathrm{AU}$$

Exercise 3: Binary Stars

- Initial conditions:

$$G = 6.67428 \times 10^{-8} \mathrm{dyne\ cm^2 g^{-2}}$$

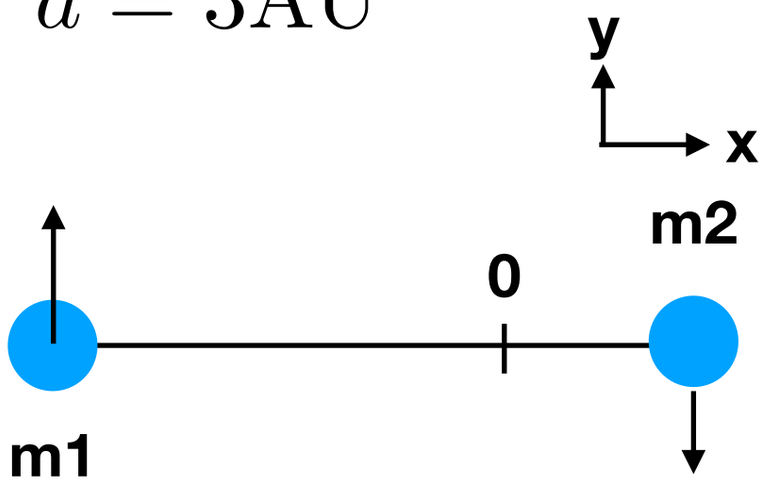
$$M_\odot = 1.989 \times 10^{33} \mathrm{g}$$

$$\mathrm{AU} = 1.49598 \times 10^{13} cm$$

$$M_1 = 1.0 M_\odot$$

$$M_2 = 2.0 M_\odot$$

$$a = 3\mathrm{AU}$$

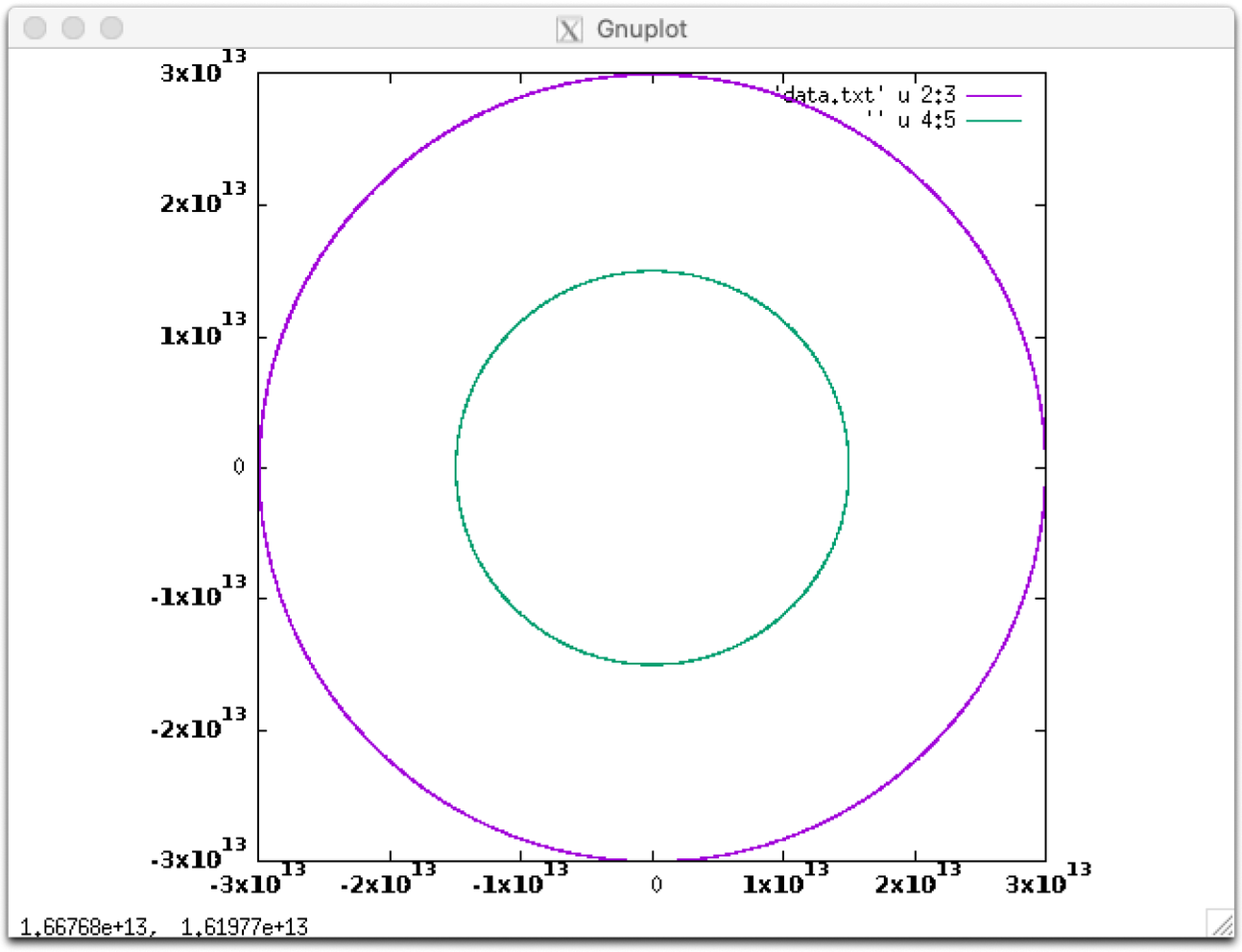


- Use Cartesian coordinates
- Set C.M at the origin
- Set initial y values = 0

$$P^2 = \frac{4\pi^2}{G(M_1 + M_2)} a^3$$

Exercise 3: Binary Stars

- Set tmax = 10 yr
- Try time step = 0.00001, 0.001, and 0.1 yr



Problem Set 2

https://kuochuanpan.github.io/courses/109ASTR660_CA/

Next lecture

- Introduction to Fortran (2)