Assignment 3 of Computational Astrophysics in NTHU

Wei-Hsiang Yu 游惟翔

March 24, 2021

1 Written Assignments

Q1 : Linear Algebra

$$A = \begin{bmatrix} 2 & 0 & -1 \\ 5 & 1 & 0 \\ 0 & 1 & 3 \end{bmatrix}$$

determinant of the 3×3 matrix:

$$detA = \begin{bmatrix} 2 & 0 & -1 \\ 5 & 1 & 0 \\ 0 & 1 & 3 \end{bmatrix} = 2 \begin{bmatrix} \times & \times & \times \\ \times & 1 & 0 \\ \times & 1 & 3 \end{bmatrix} - 0 \begin{bmatrix} \times & \times & \times \\ 5 & \times & 0 \\ 0 & \times & 3 \end{bmatrix} - 1 \begin{bmatrix} \times & \times & \times \\ 5 & 1 & \times \\ 0 & 1 & \times \end{bmatrix}$$

$$= 2 \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} - \begin{bmatrix} 5 & 1 \\ 0 & 1 \end{bmatrix} = 6 - 5 = 1$$

inverse of the 3×3 matrix:

$$A^{-1} = \frac{1}{detA} A_{adj} \tag{1}$$

For the adjugate matrix,

$$Matrix = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad adj(Matrix) = \begin{bmatrix} + \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} & - \begin{bmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{bmatrix} & + \begin{bmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{bmatrix} \\ - \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} & + \begin{bmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{bmatrix} & - \begin{bmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{bmatrix} \\ + \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} & - \begin{bmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{bmatrix} & + \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} \end{bmatrix} \tag{2}$$

So we put matrix A into Eq.1 and apply Eq.2 , and finally get the result A^{-1}

$$A^{-1} = \begin{bmatrix} 3 & -1 & 1 \\ -15 & 6 & -5 \\ 5 & -2 & 2 \end{bmatrix}$$

Q2 : Special Relativity

For Lorentz transformation,

$$x\ direction\ LT_x = \begin{bmatrix} \gamma & -\beta_x\gamma & 0 & 0 \\ -\beta_x\gamma & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad y\ direction\ LT_y = \begin{bmatrix} \gamma & 0 & -\beta_y\gamma & 0 \\ 0 & 1 & 0 & 0 \\ -\beta_y\gamma & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Where $\beta_x = \frac{v_x}{c}$,$\beta_y = \frac{v_y}{c}$

Because the feature of matrix calculate, the matrix next to the last element in the equation will be the first one affect to the element. So, LT_xLT_yA will first perform y-direction boost on matrix A.

$$v_x \to v_y\ LT_yLT_x = \begin{vmatrix} \gamma^2 & -\beta_x\gamma^2 & -\beta_x\gamma & 0 \\ -\beta_x\gamma & \gamma & 0 & 0 \\ -\beta_y\gamma^2 & -\beta_x\beta_y\gamma^2 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad v_y \to v_x\ LT_xLT_y = \begin{vmatrix} \gamma^2 & -\beta_x\gamma & -\beta_y\gamma^2 & 0 \\ -\beta_x\gamma^2 & \gamma & -\beta_x\beta_y\gamma^2 & 0 \\ -\beta_y\gamma^2 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

2 Programming Assignments

Q1 : Python Exercise

1a.Redo fortran with Python

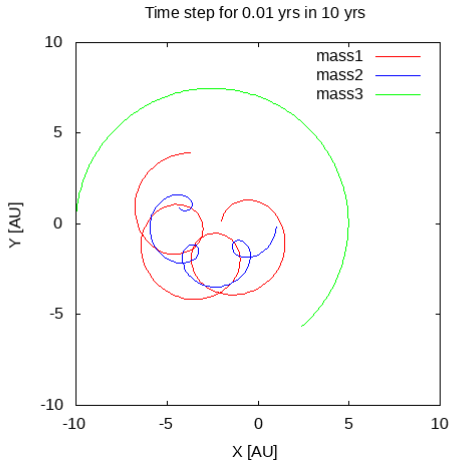

Figure 1: Result of 3-body system in Python.

(a) fortran vs Python in update pos/vel

```
!

do i =1, N
    ! update pos/vel/acc/time
    ! 先更新 位置pos, 在更新 速度vel
    stars(i)%xt = stars(i)%x +stars(i)%vx *dt
    stars(i)%x = 0.5*(stars(i)%x +(stars(i)%xt +stars(i)%vxt *dt))
    stars(i)%yt = stars(i)%y +stars(i)%vy *dt
    stars(i)%y = 0.5*(stars(i)%y +(stars(i)%yt +stars(i)%vyt *dt))
    stars(i)%vxt = stars(i)%vx +stars(i)%ax *dt
    stars(i)%vx = 0.5*(stars(i)%vx +(stars(i)%vxt +stars(i)%ax *dt))
    stars(i)%vyt = stars(i)%vy +stars(i)%ay *dt
    stars(i)%vy = 0.5*(stars(i)%vy +(stars(i)%vyt +stars(i)%ay *dt))
enddo


do j=1, N
```

```
    """
    for i in np.arange(0, N):
        yin = np.array([starnp[i][2],
                        starnp[i][3],
                        starnp[i][4],
                        starnp[i][5]])

        yout = rk2(dt, yin, derive, starnp, i)

        starnp[i][2]=yout[0]
        starnp[i][3]=yout[1]
        starnp[i][4]=yout[2]
        starnp[i][5]=yout[3]


    for j in np.arange(0, N):
```

(b) fortran vs Python in update acc

```
do j=1, N
    fx=0
    fy=0
    fxt=0
    fyt=0
    do k=1 ,N
        if (j<k) then
            x = stars(j)%x - stars(k)%x
            y = stars(j)%y - stars(k)%y
        else if (j>k) then
            x = -(stars(j)%x - stars(k)%x)
            y = -(stars(j)%y - stars(k)%y)
        endif

        rsq = x*x + y*y
        angle = atan(y,x)
        force = G *stars(j)%mass *stars(k)%mass /rsq

        if (j<k) then
            fx=fx-force*cos(angle)
            fy=fy-force*sin(angle)
        else if (j>k) then
            fx=fx+force*cos(angle)
            fy=fy+force*sin(angle)
        endif

    enddo
    stars(j)%ax = fx/stars(j)%mass
    stars(j)%ay = fy/stars(j)%mass
```

```
for j in np.arange(0, N):
    fx=0
    fy=0

    for k in np.arange(0, N):
        if j==k:
            continue
        if j<k:
            x =starnp[j][2]-starnp[k][2]
            y =starnp[j][3]-starnp[k][3]
        elif j>k:
            x =-(starnp[j][2]-starnp[k][2])
            y =-(starnp[j][3]-starnp[k][3])

        rsq = x**2 + y**2
        angle = math.atan2(y,x)
        force = G *starnp[j][1] *starnp[k][1] /rs

        if j<k:
            fx=fx-force*math.cos(angle)
            fy=fy-force*math.sin(angle)
        elif j>k:
            fx=fx+force*math.cos(angle)
            fy=fy+force*math.sin(angle)

    starnp[j][6] = fx/starnp[j][1]
    starnp[j][7] = fy/starnp[j][1]


return starnp
```

Figure 2: fortran vs Python

In this part, I use the same structure as the last lecture's assignment to code my python program. I try to use the generalized version, which can make binary system extend to simulate in n body system. In this program, time will mainly take on the second part(Fig.1(b).) in my update function in *physics* file, because in this part the program need to calculate the distance between each two stars to measure the total force to act on every single star, the loop will take much time.

1b.Compare the performance

In conclusion, the performance of python used in *numpy* package is more efficient than fortran.
I use *timeit* package in python to calculate the time program run 5 times, the result is approximately 9 10[sec](Fig.2(a).). In fortran, I use the command *system_clock()* to do measurement.(Note: *system_clock()* measure the time take by whole program, if only want to measure CPU time, can use the command *cpu_time()*) The time fortran takes in 1 time is about 4.8[sec](Fig.2(b).) in average.

(a) Time for python take 5 times

```
(base) kevinyu@惟翔の電腦:/mnt/d/Google可
w3$
python time.py
Using Python 5 times...
Done!
Done!
Done!
Done!
Done!
Using Python 5 times will take you[sec]:
9.847532800000408
```

(b) Time for fortran take 1 times

```
(base) kevinyu@惟翔の電腦:/mnt/d/Google可
y/pro3/3b$ ./binaryinary
 DONE!
 Using fortran 1 times will take you[sec]
   4.9660000000000002
(base) kevinyu@惟翔の電腦:/mnt/d/Google可
y/pro3/3b$ ./binaryinary
 DONE!
 Using fortran 1 times will take you[sec]
   4.6760000000000002
```

Figure 3: Time spend by python vs fortran

Q2 : Gravitational Potential of 3-Body System

In this part, I use the command called: *contourf*, which is the instruction of package *matplotlib.pyplot*.

About x & y axis value, I Divide these axis by AU to make the axis more readable. The gravitational potential is also divided by AU, so the power of it come to $e13$. The following are levels I chose to plot the equipotential contours.

$$
\begin{aligned}
level1.&\quad 3.000 \times 10^{12} && (white - gray\ blue)\\
level2.&\quad 6.870 \times 10^{12} && (gray\ blue - light\ blue)\\
level3.&\quad 1.000 \times 10^{13} && (light\ blue - middle\ blue)\\
level4.&\quad 1.821 \times 10^{13} && (middle\ blue - dark\ blue)\\
level5.&\quad 3.000 \times 10^{13} && (dark\ blue - white)
\end{aligned}
$$

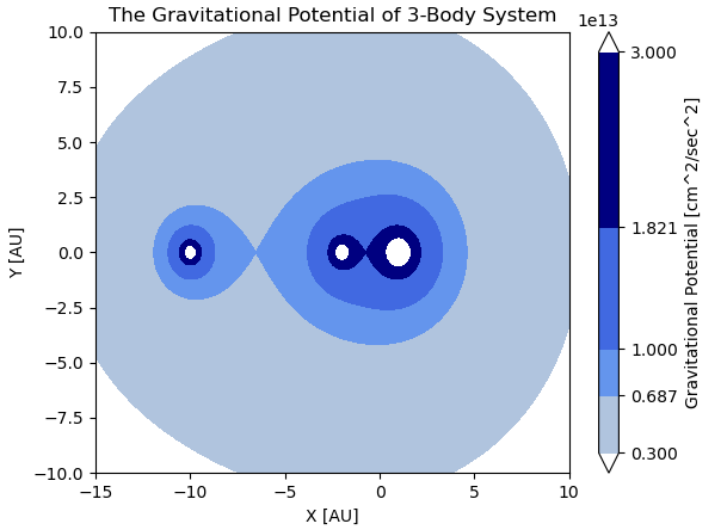

Figure 4: Gravitational Potential of 3-Body System in log scale when time=0.

Special remark **level2 & level3**, they all can imply there are points where have balanced gravitational force [1] in this 3-body system.(Lagrangian point 2 which lies on the line between two objects.).

In Fig.4., we can see two Lagrangian points: One is implied by line2, L_2 is on the line between left star(x=-10[AU]) & binary star in the right side. The other one is implied by line4, L_2 is inside the binary system.

[1] Lagrangian point:https://en.wikipedia.org/wiki/Lagrange_point