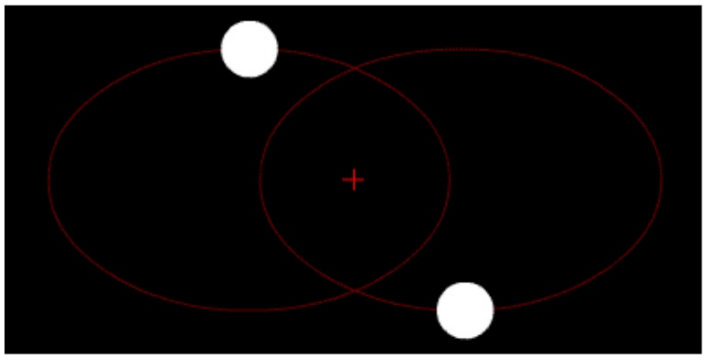
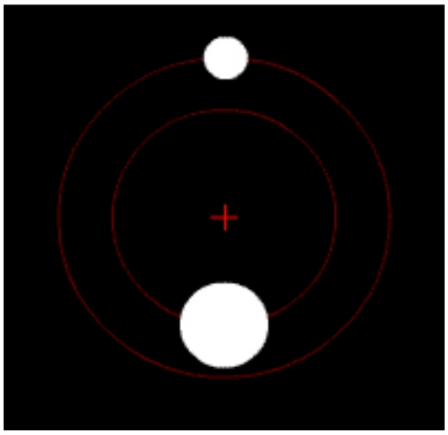
Моделирование гравитационного взаимодействия тел с сопоставимой массой

Задача 2-х тел

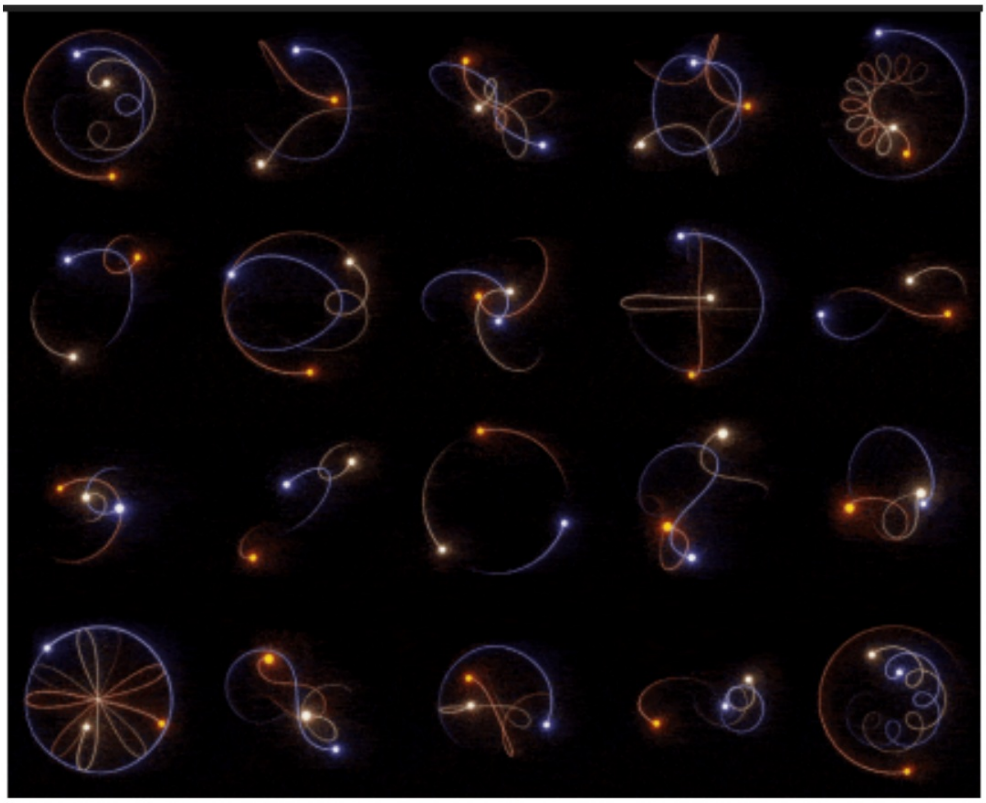


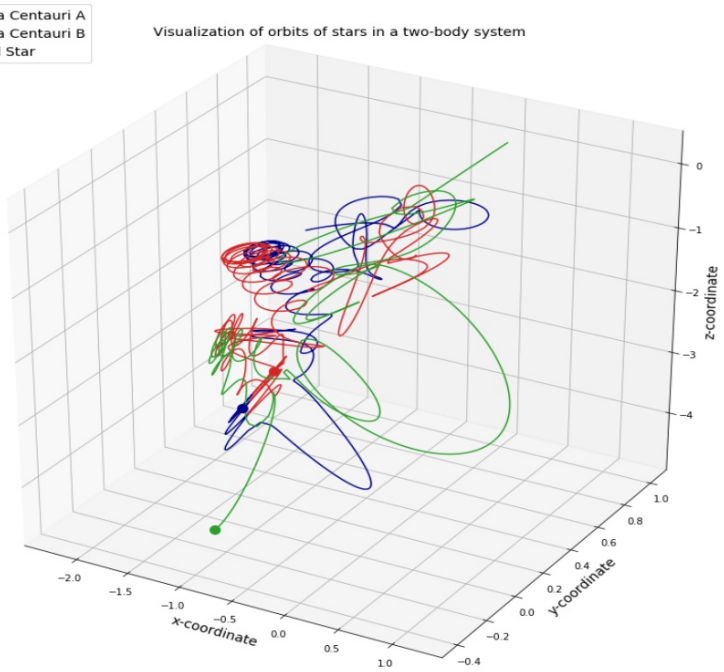
Два тела с одинаковой массой, движущиеся вокруг общего центра масс по эллиптическим



Два тела с небольшой разницей в массах движущиеся по круговым орбитам вокруг общего центра масс. Этот специфический тип орбиты подобен системе Плутон - Харон.

Исследование модели системы 3-х тел





Альфа Центавра двойная звездная система

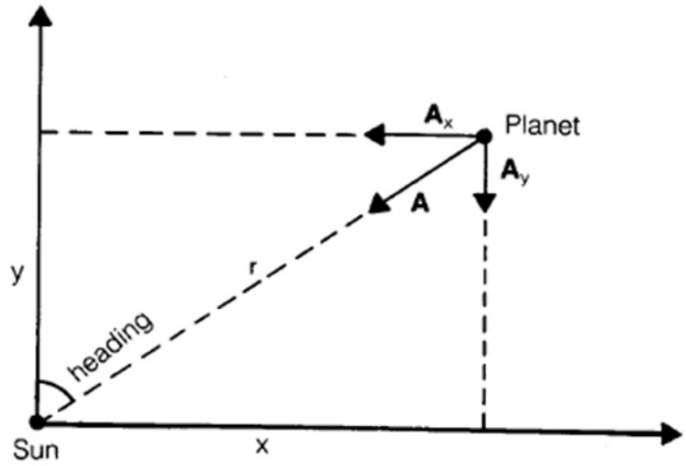
Уравнения движения

Мы видели, что между любыми двумя телами существует гравитационная сила. Эта сила определяется законом всемирного тяготения Ньютона:

$$F = \frac{GmM}{r^2}$$

где r - расстояние между центрами. Это закон движения Ньютона, который говорит нам, как эта сила влияет на движение тела. Она вызывает ускорение:

$$A = \frac{GM}{r^2}$$



Теперь ускорение - это вектор. Поскольку гравитационная сила притягивает к Солнцу, ускорение данной планеты направлено на Солнце. Если мы поместим Солнце в начало системы координат, мы увидим из рисунка, что компоненты x и y вектора ускорения имеют вид

$$A_x = - A \sin (\text{heading})$$

$$A_y = - A \cos (\text{heading})$$

Но мы также видим из рисунка, что

sin (heading) = x/r

cos (heading) = y/r

поэтому

$A_x = - A*x/r$

и

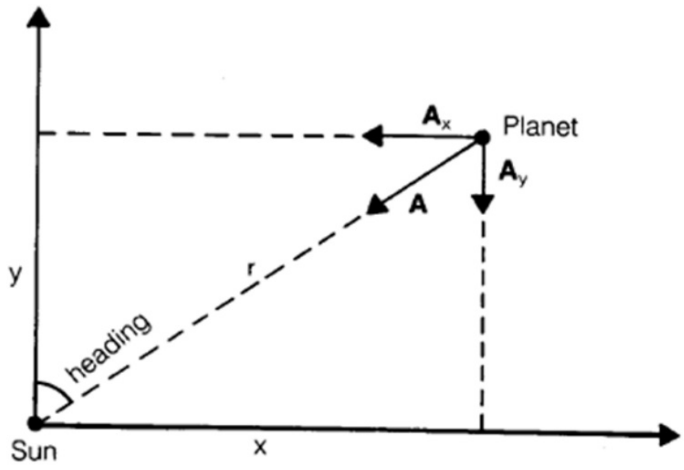
$A_y = - A*y/r$

или т.к. $A=GM/r^2$

$A_x = -GMx/r^3$

и

$A_y = - GMy/r^3$



```
comm = MPI.COMM_WORLD # оформление коммуникатора (команды исполнителей)
rank = comm.Get_rank() # номер компьютера, исполняющего код
size = comm.Get_size() # количество компьютеров в коммуникаторе
m1=2
m2=1
```

Компьютер (прцесс) с rank == 1 управляет движением тела №1

Компьютер (прцесс) с rank == 2 управляет движением тела №2

m1 - масса тела №1

m2 - масса тела №2

x - разность соотвествующих координат взаимодействующих тел

Для тела №1 $x = x_1 - x_2$ для Ax и $y = y_1 - y_2$ для Ay

Для тела №2 $x = x_2 - x_1$ для Ax и $y = y_2 - y_1$ для Ay

R - скалярная величина (модуль) расстояния между центрами тел

```
# Определение составляющих ускорения
def acc(x,R):
        if rank == 1: return -m2*x/(R**3)
        if rank == 2: return -m1*x/(R**3)
```

В рассматриваемой модели гравитационная постоянная равна 1. Масса и расстояния также представлены в условных единицах.

Компьютер (прцесс) с rank == 1 располагает в пространстве изображение (ball) тела №1 как точку красного цвета.
Компьютер (прцесс) с rank == 2 располагает в пространстве изображение (ball) тела №2 как точку зелёного цвета.

```
# размещение на орбите
def orbit(x, y, SPEED, direction):
        alpha= math.radians(direction)
# Стартовая позиция тела
        Vx=SPEED*math.cos(alpha)
        Vy=SPEED*math.sin(alpha)
        return Vx, Vy
```

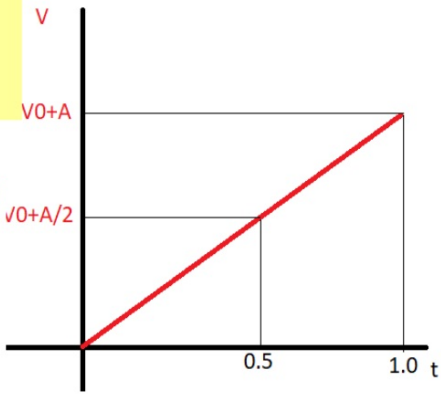
x, y - космические координаты тела в момент старта

SPEED - абсолютная величина скорости движения тела в первый момент моделироания

direction - направление движения спутника в момент старта (градусы относительно оси x)

Vx, Vy - составляющие скорости движения тела в момент старта

В рамках компьютерной модели считаем, что на каждом шаге моделирования ускорение является постоянным. Точность (адекватность) модели определяется величиной шага моделирования (h).



Лучшее решение для описания движения с **переменным ускорением** - использовать **приближение средней точки**.

Если скорость через 10 с равна 3,0 м/с, а скорость через 11 с - 3,3 м/с, то хорошее приближение скорости в течение 10 с - 3,15 м/с - средняя скорость.

Другой способ выразить этот результат - установить среднюю скорость равной 10 с, **плюс половину изменения скорости**. Помните, что изменение скорости - это ускорение.

```
# Перемещение по орбите
def move(x,y,Rx,Ry,R, Vx, Vy):
        h=0.05
# изменение скорости на величину ускорения
        Vx= Vx+acc(Rx,R)*h/2.
        Vy= Vy+acc(Ry,R)*h/2.
# определяем текущие координаты спутника в космосе
        x=x+Vx*h
        y=y+Vy*h
        return x,y,Vx,Vy
```

h - единица шага врмени моделирования. В данном случае 0.05 сек.

```
if rank==0:
  global t1
  from matplotlib.animation import FuncAnimation
  fig, axes = plt.subplots(figsize=(10,10))
  axes.set_xlim(-2., 2.)
  axes.set_ylim(-4., 0.)
  axes.set_aspect("equal")
  axes.grid(True,alpha=0.3)
  r=0.03
  N = 50 # число кадров анимации
  x1=[comm.recv(source=1)]
  y1=[comm.recv(source=1)]
  x2=[comm.recv(source=2)]
  y2=[comm.recv(source=2)]
  for t in range(N):
    comm.send(N-t, dest=1)
    comm.send(N-t, dest=2)
    comm.send(x1[t], dest=1)
    comm.send(y1[t], dest=1)
    comm.send(x2[t], dest=2)
    comm.send(y2[t], dest=2)
    x=x1[t]-x2[t]
    y=y1[t]-y2[t]
    R=math.sqrt(x**2+y**2)
    comm.send(x, dest=1)
    comm.send(-x, dest=2)
    comm.send(y, dest=1)
    comm.send(-y, dest=2)
    comm.send(R, dest=1)
    comm.send(R, dest=2)
    x1.append(comm.recv(source=1) )
    y1.append(comm.recv(source=1) )
    x2.append(comm.recv(source=2))
    y2.append(comm.recv(source=2))
    print(t)
    if t%10==0:
      print(x1[t],y1[t],x2[t],y2[t])
```

t_b_mpi014_25.py

Компьютер (прцесс) с rank == 0
- координирует работу компьютеров (процессов) №1 и №2
- накапливает информацию о положении в пространстве каждого из 2-х тел
- готовит файлы с кадрами анимации

```
t1=0
def update(frame):
  global t1
  print(t1)
  ball = matplotlib.patches.Circle((x1[t1],y1[t1]), radius=r, fill=True,color="r")
  axes.add_patch (ball)
  ball = matplotlib.patches.Circle((x2[t1],y2[t1]), radius=r, fill=True,color="g")
  axes.add_patch (ball)
  t1=t1+1
print("******")
ani = FuncAnimation(fig, update, frames=N-1)
ani.save('2_b.mp4', writer='ffmpeg', fps=30)
print("******")
```

```
from mpi4py import MPI
import math
#from random import random, randint, uniform
import matplotlib.pyplot as plt
import matplotlib.patches
import matplotlib.lines
import matplotlib.path
```

Получение стартовых положений тел в списки (массивы) координат.
Оформление цикла наполнения списков координат описывающих движение каждого тела.
Отправка каждому из процессов №1 и №2 информации об оставшихся кадрах и о состоянии тел перед очередным шагом моделирования.
Получение от процессов №1 и №2 информации о состоянии тел после очередного шага моделирования
Формирование и сохранение файла анимации

Алгоритм работы компьютера (процесса) №1

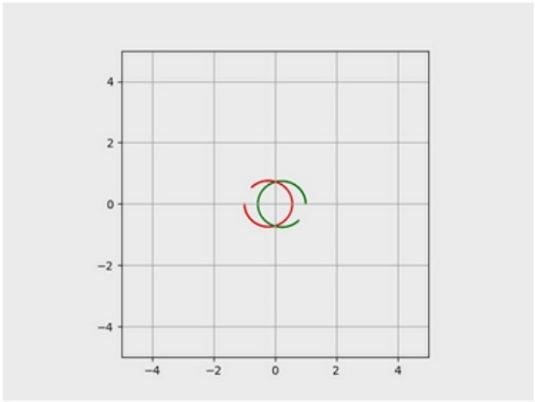
```
if rank==1:
# вектор данных x1, y1, v1, dir
  x, y = -1., 0
  comm.send(x, dest=0)
  comm.send(y, dest=0)
  Vx, Vy = orbit(x, y, .6, 270.)
  c=comm.recv(source=0)
  while(c>0):
    x = comm.recv(source=0)
    y = comm.recv(source=0)
    Rx = comm.recv(source=0)
    Ry = comm.recv(source=0)
    R = comm.recv(source=0)
    x, y, Vx, Vy = move(x, y, Rx, Ry, R, Vx, Vy)
    comm.send(x, dest=0)
    comm.send(y, dest=0)
    c=comm.recv(source=0)
  print("#1")
  comm.send(x, dest=0)
  comm.send(y, dest=0)
```

Формирование исходной информации о теле №1 и передача этой информации процессу №0
Цикл моделирования движения тела №1

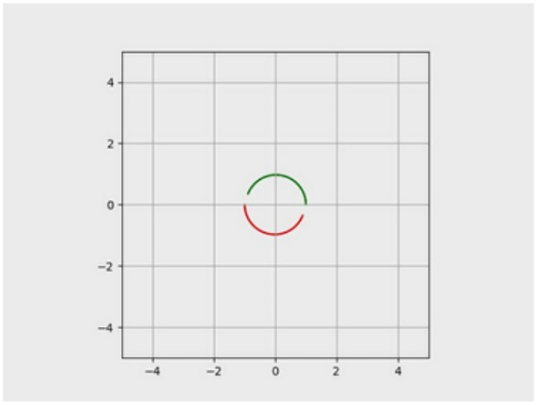
Моделирование движения 2-х тел равной массы (m1=m2=1; x1=-1, y1=0; x2=1, y2=0)

V1=0.3
Dir1=270

V2=0.3
Dir2=90



t_b03_03.avi



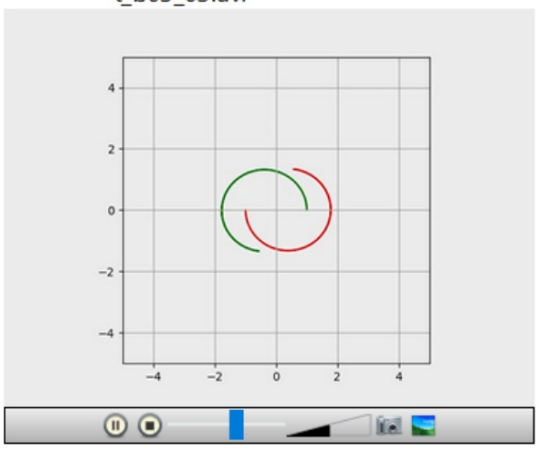
V1=0.35
Dir1=270

V2=0.35
Dir2=90

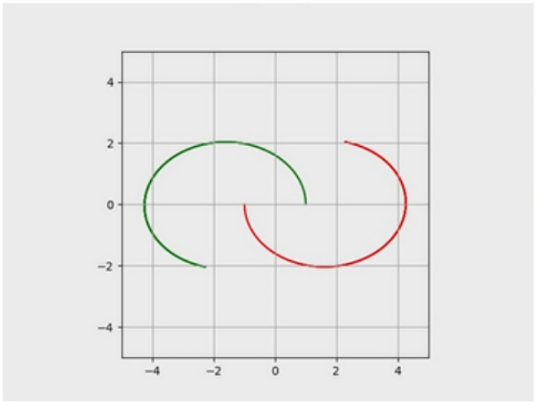
t_b03_035.avi

V1=0.4
Dir1=270

V2=0.4
Dir2=90



t_b03_04.avi



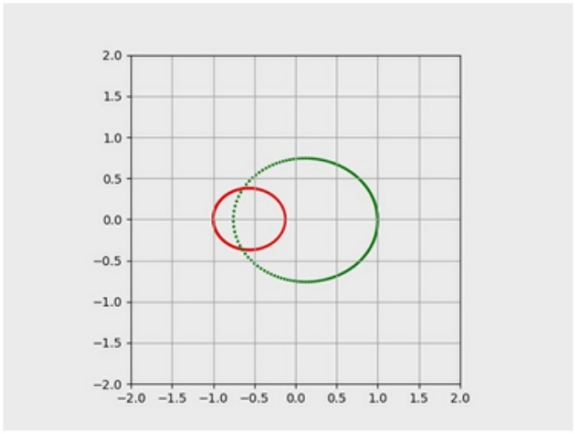
V1=0.45
Dir1=270

V2=0.45
Dir2=90

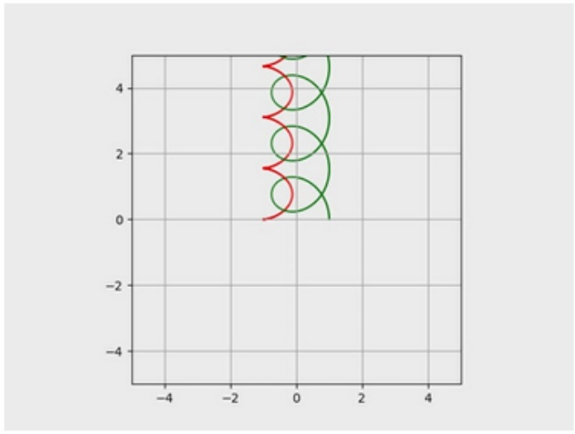
t_b03_045.avi

Гравитационное взаимодействие 2-х тел (m1=2, x1=-1, y1=0; m2=1, x2=1, y2=0)

V1= **0.2**
Dir=270
V2= **0.4**
Dir=90



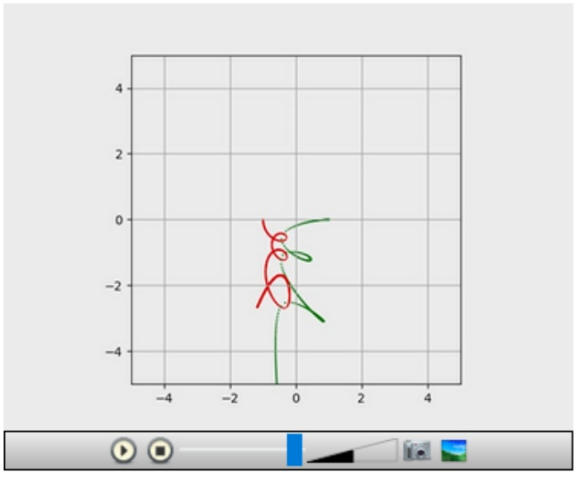
t_b04_02_4.avi



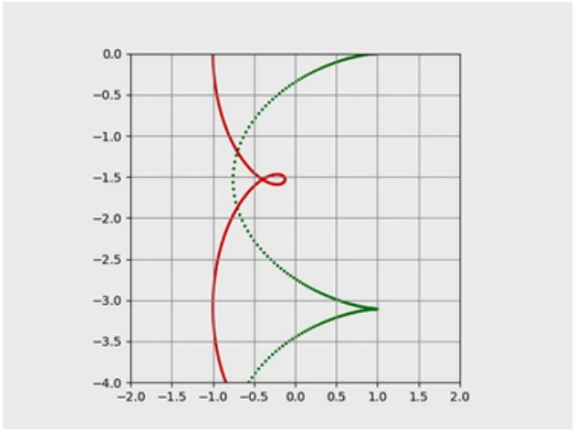
t_b04_0_6.avi

V1=0
Dir=270
V2= **0.6**
Dir=90

V1=0.3
Dir=270
V2=0
Dir=90



t_b04_03_0.avi



t_b04_06_0_D.avi

V1= **0.6**
Dir=270
V2=0
Dir=90

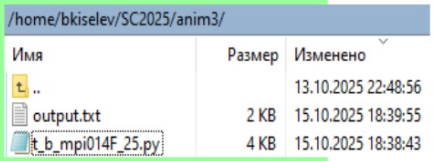
Построение анимации вне кластера

Основное отличие в части программы для rank=0

```
if rank==0:
   global t1
   from matplotlib.animation import FuncAnimation
   fig, axes = plt.subplots(figsize=(10,10))
   axes.set_xlim(-2., 2.)
   axes.set_ylim(-4., 0.)
   axes.set_aspect("equal")
   axes.grid(True,alpha=0.3)
   r=0.03
   Fout = open ("output.txt", "w")
   N = 50
   x1=comm.recv(source=1)
   y1=comm.recv(source=1)
   x2=comm.recv(source=2)
   y2=comm.recv(source=2)
   for t in range(N):
      comm.send(N-1-t, dest=1)
      comm.send(N-1-t, dest=2)
      comm.send(x1[t], dest=1)
      comm.send(y1[t], dest=1)
      comm.send(x2[t], dest=2)
      comm.send(y2[t], dest=2)
      x=x1[t]-x2[t]
      y=y1[t]-y2[t]
      R=math.sqrt(x**2+y**2)
      comm.send(x, dest=1)
      comm.send(-x, dest=2)
      comm.send(y, dest=1)
      comm.send(-y, dest=2)

      comm.send(R, dest=1)
      comm.send(R, dest=2)

      x1=comm.recv(source=1)
      y1=comm.recv(source=1)
      x2=comm.recv(source=2)
      y2=comm.recv(source=2)
      print(t)
      Fout.write ( "{:f} {:f} {:f} {:f}\n".format(x1,y1,x2,y2))
      if t%10==0:
         print(x1,y1,x2,y2)
   Fout.close()
```



t_b_mpi014F_25.py

output.txt (фрагмент)

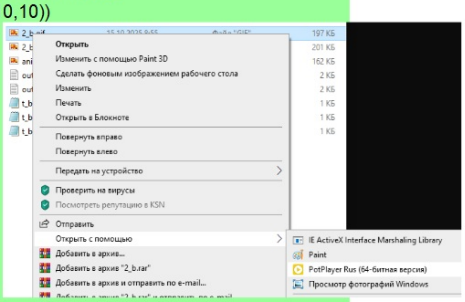
```
-1.000000 0.000000 1.000000 0.000000
-0.999687 -0.030000 0.999375 0.000000
-0.999062 -0.059995 0.998125 -0.000009
-0.998124 -0.089981 0.996248 -0.000038
-0.996873 -0.119953 0.993745 -0.000094
-0.995308 -0.149906 0.990615 -0.000188
-0.993428 -0.179835 0.986856 -0.000330
-0.991234 -0.209736 0.982467 -0.000529
-0.988724 -0.239603 0.977448 -0.000794
-0.985898 -0.269432 0.971795 -0.001137
-0.982754 -0.299217 0.965509 -0.001566
-0.979293 -0.328954 0.958586 -0.002092
-0.975512 -0.358637 0.951024 -0.002727
* * *
```

Программа для персонального компьютера пользователя

```
import math
import matplotlib.pyplot as plt
import matplotlib.patches
import matplotlib.lines
import matplotlib.path

global t1
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation
fig, axes = plt.subplots(figsize=(10,10))
axes.set_xlim(-2., 2.)
axes.set_ylim(-4., 0.)
axes.set_aspect("equal")
axes.grid(True,alpha=0.3)
r=0.03
Fin= open ("output.txt")
N = 50

t1=0
def update(frame):
    global t1
    print(t1)
    s=Fin.readline().split()
    x1,y1,x2,y2=map(float,s)
    ball = matplotlib.patches.Circle((x1,y1), radius=r, fill=True,color="r")
    axes.add_patch (ball)
    ball = matplotlib.patches.Circle((x2,y2), radius=r, fill=True,color="g")
    axes.add_patch (ball)
    t1=t1+1
print("******")
ani = FuncAnimation(fig=fig, func=update, frames=N-1, interval=30,
repeat=False)
#plt.show();
ani.save('2_b.gif', writer='pillow', fps=30)
print("******")
Fin.close()
```



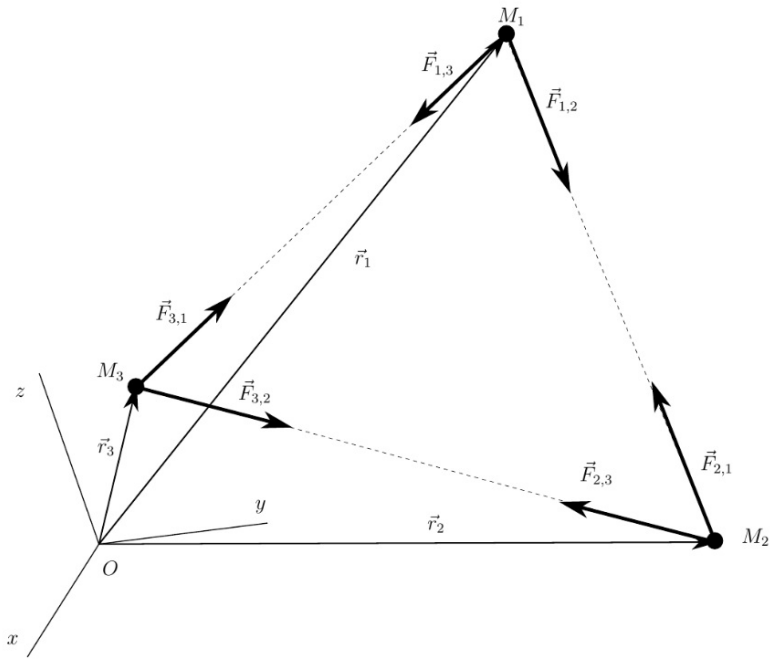
```
plt.show(); # режим show
#ani.save('2_b.gif', writer='pillow', fps=30)
```

t_b_014F.py

Анимация сохраняется в формате GIF.
Для просмотра в Windows пригоден PotPlayer Rus

Задание

Построить компьютерную модель движения системы из 3-х тел (m1=1, m2=2, m3=3).
Движение рассматривается в плоскости XY. Составляющая Vz =0 для всех тел.



Тела считаем материальными точками. Положение тел будем отсчитывать в произвольном базисе, с которым связана инерциальная система отсчета *Oxyz*. Положение каждого из тел задается радиус-вектором соответственно $\vec{r}_1$, $\vec{r}_2$ и $\vec{r}_3$. На каждое тело действует сила гравитационного притяжения со стороны двух других тел, причем в соответствии с третьей аксиомой динамики точки (3-й закон Ньютона)

$$\begin{aligned} m_1 \frac{d^2\vec{r}_1}{dt^2} &= \vec{F}_{1,2} + \vec{F}_{1,3} \\ m_2 \frac{d^2\vec{r}_2}{dt^2} &= \vec{F}_{2,1} + \vec{F}_{2,3} \\ m_3 \frac{d^2\vec{r}_3}{dt^2} &= \vec{F}_{3,1} + \vec{F}_{3,2} \end{aligned}$$

$$\vec{F}_{i,j} = -\vec{F}_{j,i}$$

$$\begin{aligned} m_1 \frac{d^2\vec{r}_1}{dt^2} &= \vec{F}_{1,2} + \vec{F}_{1,3} \\ m_2 \frac{d^2\vec{r}_2}{dt^2} &= -\vec{F}_{1,2} + \vec{F}_{2,3} \\ m_3 \frac{d^2\vec{r}_3}{dt^2} &= -\vec{F}_{1,3} - \vec{F}_{2,3} \end{aligned}$$

Но мы также видим из рисунка, что

sin (heading) = x/r

cos (heading) = y/r

поэтому

$A_x = - A*x/r$

и

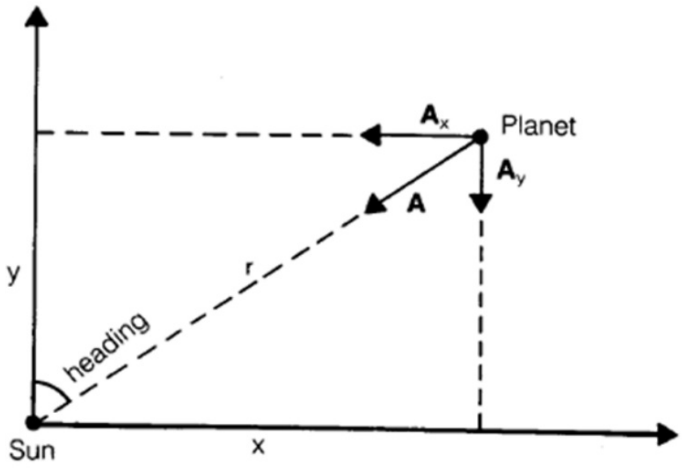
$A_y = - A*y/r$

или т.к. $A=GM/r^2$

$A_x = -GMx/r^3$

и

$A_y = - GMy/r^3$




```
def acc(x,R,m):
        return -m*x/(R**3)
```

Фунция используется для вычисления составляющих ускорения A_x и A_y при взаимодействии каждой пары тел

```
from mpi4py import MPI
import numpy as np
import math
import pylab
import matplotlib.patches
import matplotlib.lines
import matplotlib.path

# Подготовка анимации
comm = MPI.COMM_WORLD # оформление коммуникатора
(команды исполнителей)
rank = comm.Get_rank() # номер компьютера, исполняющего код
size = comm.Get_size() # количество компьютеров в
коммуникаторе
m1, m2, m3 = 1., 1., 0.
X=[[0., 0., 0],
   [0., 0., 0],
   [0., 0., 0]]
Y=[[0., 0., 0],
   [0., 0., 0],
   [0., 0., 0]]
```

Матрицы составляющих парного смещения тел относительно друг друга

```
   X[0][1]=-(x2-x1)
   X[1][0]=-X[0][1]
   X[0][2]=-(x3-x1)
   X[2][0]=-X[0][2]
   X[1][2]=-(x3-x2)
   X[2][1]=-X[1][2]
#
   Y[0][1]=-(y2-y1)
   Y[1][0]=-Y[0][1]
   Y[0][2]=-(y3-y1)
   Y[2][0]=-Y[0][2]
   Y[1][2]=-(y3-y2)
   Y[2][1]=-Y[1][2]
```

```
 # размещение на орбите
def orbit(x1,y1,SPEED,direction):

        x=x1
        y=y1
        alpha= math.radians(direction)
# Стартовая позиция тела
        Vx=SPEED*math.cos(alpha)
        Vy=SPEED*math.sin(alpha)
        return Vx, Vy #, ball
```

Процесс с rank=1 моделирует движение красного тела
Процесс с rank=2 моделирует движение зелёного тела
Процесс с rank=3 моделирует движение синего тела

```
def move(x,y,X,Y,Vx,Vy):
        h=0.005 # шаг моделирования, чем меньше, тем точнее модель
# изменение скорости на величину ускорения
        if rank==1:
            R=math.sqrt(X[0][1]*X[0][1]+Y[0][1]*Y[0][1])
            Acc1x=acc(X[0][1],R,m2)
            Acc1y=acc(Y[0][1],R,m2)
            R=math.sqrt(X[0][2]*X[0][2]+Y[0][2]*Y[0][2])
            Acc2x=acc(X[0][2],R,m3)
            Acc2y=acc(Y[0][2],R,m3)
        if rank==2:
            R=math.sqrt(X[1][0]*X[1][0]+Y[1][0]*Y[1][0])
            Acc1x=acc(X[1][0],R,m1)
            Acc1y=acc(Y[1][0],R,m1)
            R=math.sqrt(X[1][2]*X[1][2]+Y[1][2]*Y[1][2])
            Acc2x=acc(X[1][2],R,m3)
            Acc2y=acc(Y[1][2],R,m3)
        if rank==3:
            R=math.sqrt(X[2][0]*X[2][0]+Y[2][0]*Y[2][0])
            Acc1x=acc(X[2][0],R,m1)
            Acc1y=acc(Y[2][0],R,m1)
            R=math.sqrt(X[2][1]*X[2][1]+Y[2][1]*Y[2][1])
            Acc2x=acc(X[2][1],R,m2)
            Acc2y=acc(Y[2][1],R,m2)
        Vx= Vx+(Acc1x+Acc2x)*h/2.
        Vy= Vy+(Acc1y+Acc2y)*h/2.
# определяем текущие координаты тела в космосе
        x=x+Vx*h
        y=y+Vy*h
        return x,y,Vx,Vy
```

Перемещение по орбите
Функция может использоваться каждым из процессов моделирующих движение тел

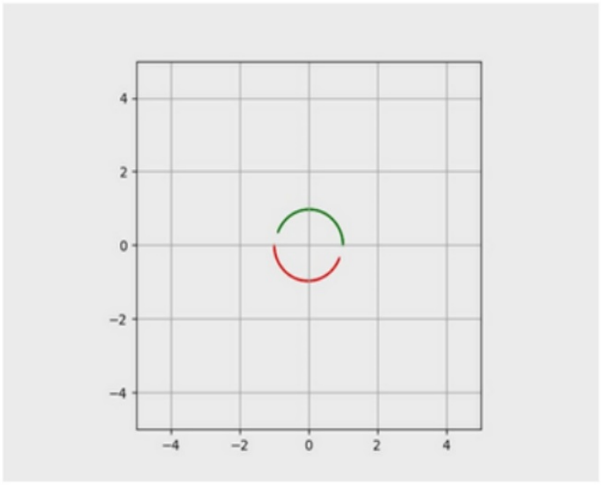
Процесс rank=0 координирует взаимодействие процессов, связанных с телами, собирает от них информацию и строит кадры анимации.

Часть программы для процессов, моделирующих движение тел

```
if 1 <= rank <= 3:
# вектор данных x1, y1, v1, dir
  if rank==1: x, y, V, dir= -1, 0, 0.35, 90.
  if rank==2: x, y, V, dir=  1, 0, 0.35, 270.
  if rank==3: x, y, V, dir=  0, 0, 0, 0.
  comm.send(x, dest=0)
  comm.send(y, dest=0)
  X = comm.recv(source=0) #comm.send(X,dest=1)
  Y = comm.recv(source=0) #comm.send(Y,dest=1)
  Vx, Vy = orbit(x, y, V, dir)
  c= comm.recv(source=0) #comm.send(N,dest=1)
  print(c)
  while (c>0):
    x, y, Vx, Vy = move(x, y, X, Y, Vx, Vy)
    c=comm.recv(source=0) #comm.send(N-1-t, dest=1)
    comm.send(x, dest=0)
    comm.send(y, dest=0)
    comm.send(ball, dest=0)
    X = comm.recv(source=0) #comm.send(X,dest=1)
    Y = comm.recv(source=0) #comm.send(Y,dest=1)
  comm.send(x, dest=0)
  comm.send(y, dest=0)
```

Проверка правильности работы mpi программы для 3-х тел сравнением результатов, полученных программой для 2-х тел

Моделирование движения 2-х тел равной массы (m1=m2=1; x1=-1, y1=0; x2=1, y2=0)



V1=0.35
Dir1=270

V2=0.35
Dir2=90

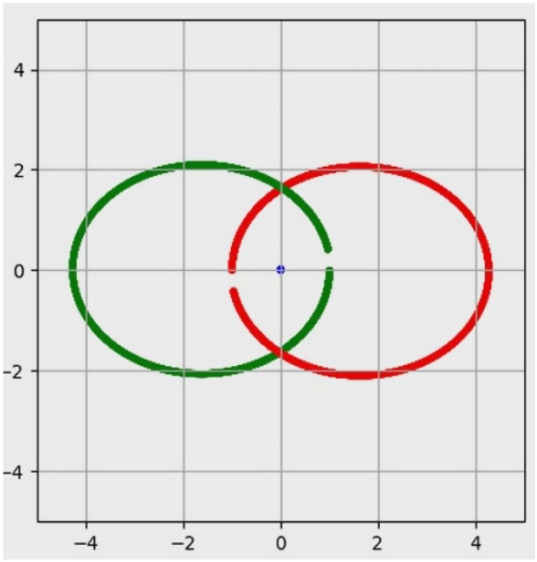
При правильной работе mpi программы для 3-х тел (m3=0, x3=0, y3=0) результат, должен быть аналогичен полученному программой для 2-х тел

```
# Расчет ускорений
def ACC(x,y,X,Y):
        if rank==1:
            R=math.sqrt(X[0][1]*X[0][1]+Y[0][1]*Y[0][1])
            Acc1x=acc(X[0][1],R,m2)
            Acc1y=acc(Y[0][1],R,m2)
            R=math.sqrt(X[0][2]*X[0][2]+Y[0][2]*Y[0][2])
            Acc2x=acc(X[0][2],R,m3)
            Acc2y=acc(Y[0][2],R,m3)
        if rank==2:
            R=math.sqrt(X[1][0]*X[1][0]+Y[1][0]*Y[1][0])
            Acc1x=acc(X[1][0],R,m1)
            Acc1y=acc(Y[1][0],R,m1)
            R=math.sqrt(X[1][2]*X[1][2]+Y[1][2]*Y[1][2])
            Acc2x=acc(X[1][2],R,m3)
            Acc2y=acc(Y[1][2],R,m3)
        if rank==3:
            R=math.sqrt(X[2][0]*X[2][0]+Y[2][0]*Y[2][0])
            Acc1x=acc(X[2][0],R,m1)
            Acc1y=acc(Y[2][0],R,m1)
            R=math.sqrt(X[2][1]*X[2][1]+Y[2][1]*Y[2][1])
            Acc2x=acc(X[2][1],R,m2)
            Acc2y=acc(Y[2][1],R,m2)
        return Acc1x, Acc1y, Acc2x, Acc2y
```

На каждом шаге моделирования сначала, исходя из текущего расположения тел, рассчитываются величины ускорений для каждого тела.

```
def move(x, y, Vx,Vy, Acc1x, Acc1y, Acc2x, Acc2y, h):
        Vx= Vx+(Acc1x+Acc2x)*h/2.
        Vy= Vy+(Acc1y+Acc2y)*h/2.
# определяем текущие координаты спутника в космосе
        x=x+Vx*h
        y=y+Vy*h
        return x,y,Vx,Vy
```

Пример устойчивой модели для 3-х тел при m1=m2



Пример неустойчивой системы