### 星体运动模拟系统说明

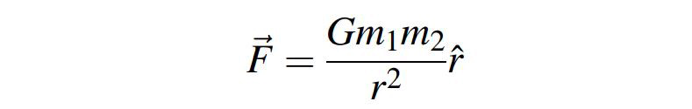
1. **组员：**闫谅，舒允豪
2. **系统介绍**

据2019年11月15日的一则科学报告显示，我们目前已经发现4000多个系外行星，不过，这些系外行星的运行情况不大一样，它们所环绕的主恒星情况不一样。耶拿大学天体物理研究所和大学天文台的一项新研究结果表明，目前已经发现有200多个系外行星所在的恒星系是多星系统。也就是说，如果站在这200多个系外行星表面，我们可以看到两个或者两个以上“太阳”。

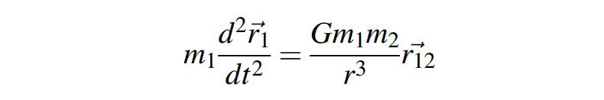
我们希望建立一个系统，通过确定初始条件，可以简单的模拟所有星体系统的运动，并求出该系统各个星体运动坐标的数值解。

1. **物理模型**

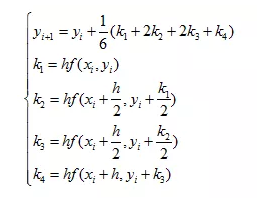
1.万有引力公式



2.牛顿第二定律



1. 四阶龙格库塔法



1. **代码实现**

1.初始条件设定函数

在本函数中输入星体数，总运行时间，星体质量、位置、初速度以及步长。

def parameter():  
 n = int(input('输入星体个数（建议大于3小于10），随机产生参数并运行，或输入0自行设置参数\n'))  
  
 step = 100000  
  
 if n != 0:  
 time = int(input('输入总运行时间（建议大于10000000）\n'))  
 data = [[] for i in range(n)]  
 for i in range(n):  
 data[i] = [random.randint(1, 10)\*1e30, random.randint(-10, 10)\*1e9,  
 random.randint(-10, 10)\*1e9, random.randint(-10, 10)\*1e9,  
 random.randint(-5, 5)\*1e3, random.randint(-5, 5)\*1e3,  
 random.randint(-5, 5)\*1e3]  
  
 else:  
 n = int(input('输入星体个数: '))  
 data = [[0]]\*n  
 for i in range(n):  
 print('请输入第%d个星体的质量、横坐标、纵坐标、水平方向速度、竖直方向速度，采用国际单位，空格间隔' % (i+1))  
 data[i] = list(map(int, list(input().split())))  
  
 time, step = map(int, input('输入总运行时间和时间步长，国际单位，空格间隔\n').split())  
  
 s = []  
 for i in range(n):  
 s.append(data[i][0])  
 for i in range(n):  
 if max(s) != min(s):  
 s[i] = (s[i] - min(s)) \* 30 / (max(s) - min(s)) + 20  
 else:  
 s[i] = 40  
  
 return data, n, time, step, s

1. 速度函数  
   def velocity\_processing(data, n):  
     
    velocity = [[0]\*3 for i in range(n)]  
     
    # 一阶导为速度  
    for i in range(n):  
    velocity[i][0] = data[i][4]  
    velocity[i][1] = data[i][5]  
    velocity[i][2] = data[i][6]  
     
    return velocity  
     
   3.加速度函数  
   def acceleration\_processing(data, n):  
     
    g\_constant = 6.67e-11  
     
    acceleration = [[0]\*3 for i in range(n)]  
    # 二阶导为加速度  
    for i in range(n):  
    for j in range(i+1, n):  
     
    r2 = (data[i][1] - data[j][1]) \*\* 2 + ((data[i][2] - data[j][2]) \*\* 2) + ((data[i][3] - data[j][3]) \*\* 2)  
    f = data[i][0] \* data[j][0] \* g\_constant / r2  
     
    axi = (f / data[i][0]) \* ((data[j][1] - data[i][1]) / math.sqrt(r2))  
    axj = (f / data[j][0]) \* ((data[i][1] - data[j][1]) / math.sqrt(r2))  
    ayi = (f / data[i][0]) \* ((data[j][2] - data[i][2]) / math.sqrt(r2))  
    ayj = (f / data[j][0]) \* ((data[i][2] - data[j][2]) / math.sqrt(r2))  
    azi = (f / data[i][0]) \* ((data[j][3] - data[i][3]) / math.sqrt(r2))  
    azj = (f / data[j][0]) \* ((data[i][3] - data[j][3]) / math.sqrt(r2))  
     
    acceleration[i][0] += axi  
    acceleration[i][1] += ayi  
    acceleration[i][2] += azi  
    acceleration[j][0] += axj  
    acceleration[j][1] += ayj  
    acceleration[j][2] += azj  
     
    return acceleration  
     
     
   4.矩阵转换

def tolist(data, k\_, l\_, n, step):  
  
 li = [[]]\*n  
 for i in range(n):  
 li[i] = [0] + list(k\_[i]) + list(l\_[i])  
  
 return list(np.array(data) + step / 2 \* np.array(li))  
  
5.四阶龙格库塔法求解  
def runge\_kutta(data, n, array, t\_array, time, step):  
  
 step2 = time\_optimizer(data, n, step)  
  
 # 使用四阶龙格库塔法求数值解  
 k\_1 = velocity\_processing(data, n)  
 l\_1 = acceleration\_processing(data, n)  
 m\_1 = tolist(data, k\_1, l\_1, n, step2)  
 k\_2 = velocity\_processing(m\_1, n)  
 l\_2 = acceleration\_processing(m\_1, n)  
 m\_2 = tolist(data, k\_2, l\_2, n, step2)  
 k\_3 = velocity\_processing(m\_2, n)  
 l\_3 = acceleration\_processing(m\_2, n)  
 m\_3 = tolist(data, k\_3, l\_3, n, step2)  
 k\_4 = velocity\_processing(m\_3, n)  
 l\_4 = acceleration\_processing(m\_3, n)  
  
 for i in range(n):  
 data[i][1] += (k\_1[i][0] + 2 \* k\_2[i][0] + 2 \* k\_3[i][0] + k\_4[i][0]) \* step2 / 6  
 data[i][2] += (k\_1[i][1] + 2 \* k\_2[i][1] + 2 \* k\_3[i][1] + k\_4[i][1]) \* step2 / 6  
 data[i][3] += (k\_1[i][2] + 2 \* k\_2[i][2] + 2 \* k\_3[i][2] + k\_4[i][2]) \* step2 / 6  
 data[i][4] += (l\_1[i][0] + 2 \* l\_2[i][0] + 2 \* l\_3[i][0] + l\_4[i][0]) \* step2 / 6  
 data[i][5] += (l\_1[i][1] + 2 \* l\_2[i][1] + 2 \* l\_3[i][1] + l\_4[i][1]) \* step2 / 6  
 data[i][6] += (l\_1[i][2] + 2 \* l\_2[i][2] + 2 \* l\_3[i][2] + l\_4[i][2]) \* step2 / 6  
 array[i][0].append(data[i][1])  
 array[i][1].append(data[i][2])  
 array[i][2].append(data[i][3])  
  
 t\_array.append(time+step2)  
  
 return data, step2, array, t\_array  
  
  
6.绘图函数

可以自由选择绘制一定时间内的运动轨迹或是绘制完整的运动轨迹。

def plotting(array, n, time, step, el, az, f, s):  
  
 sns.set(style='darkgrid')  
 plt.ion()  
 track\_length = 2000 # 控制绘制的轨迹长度  
  
 # time // 1000可调整，控制绘图的间隔，除数越大程序运行速度越快  
 if time[-1] // 1000 > f:  
  
 plt.clf()  
 fig = plt.gcf()  
 ax = fig.gca(projection='3d')  
 ax.view\_init(el, az)  
  
 # 绘制星体一定运行时间内的运动轨迹  
 for i in range(n):  
 if len(array[i][0]) < track\_length:  
 ax.plot(array[i][0], array[i][1], array[i][2])  
 ax.scatter(array[i][0][-1], array[i][1][-1], array[i][2][-1], s=s[i])  
 else:  
 index = 0  
 for j in range(len(time)-1, 0):  
 if time[n] < time[-1] - track\_length \* step:  
 index = n  
 break  
 ax.plot(array[i][0][index:-1], array[i][1][index:-1], array[i][2][index:-1])  
 ax.scatter(array[i][0][-1], array[i][1][-1], array[i][2][-1], s=s[i])  
  
 # 绘制星体的完整运动轨迹  
 # for i in range(n):  
 # ax.plot(array[i][0], array[i][1], array[i][2])  
 # ax.scatter(array[i][0][-1], array[i][1][-1], array[i][2][-1], s=s[i])  
  
 """ax.legend(["[" + str("%e" % array[i][0][-1]) + " , " + str("%e" % array[i][1][-1]) + " , "  
 + str("%e" % array[i][2][-1]) + "]" for i in range(n)], loc='upper right')"""  
 # plt.title('program has run for %.2f units of time' % time[-1])  
 plt.pause(0.01)  
 el, az = ax.elev, ax.azim  
 plt.savefig('%d.jpg'%f)  
 plt.ioff()  
 f += 1  
  
 return el, az, f  
  
7.碰撞检测函数函数

防止星体距离过近时产生的突变现象，在星体距离小时及时调整步长保证拟合的精度，并在星体碰撞时结束程序。  
def time\_optimizer(data, n, step):  
  
 g\_constant = 6.67e-11  
 constant = 0.00001 # 该参数可以调整，在尽可能小的情况下可以避免星体距离过近时因为步长过长导致误差较大的情况  
  
 for i in range(n):  
 for j in range(i+1, n):  
 cdist = math.sqrt((data[i][1] - data[j][1]) \*\* 2 + (data[i][2] - data[j][2]) \*\* 2  
 + (data[i][3] - data[j][3]) \*\* 2)  
 force = g\_constant \* data[i][0] \* data[j][0] / cdist \*\* 2  
 ai = force / data[i][0]  
 aj = force / data[j][0]  
 if math.sqrt(constant \* cdist / ai) <= constant \* step \  
 or math.sqrt(constant \* cdist / aj) <= constant \* step:  
 print("星体之间距离过近，程序终止")  
 exit()  
  
 # 调整步长  
 if math.sqrt(constant \* cdist / ai) <= step:  
 step = math.sqrt(constant \* cdist / ai)  
 if math.sqrt(constant \* cdist / aj) <= step:  
 step = math.sqrt(constant \* cdist / aj)  
  
 return step  
  
8.GIF生成函数  
def create\_gif(f):  
  
 images = []  
 for i in range(f):  
 im = imageio.imread('%d.jpg' % i)  
 images.append(im)  
 imageio.mimsave('结果.gif', images, 'GIF', duration=0.01)

# #######################主程序###############################  
  
# 导入星体的各项参数，程序总运行时间，时间步长  
star\_parameter, star\_number, total\_time, time\_step, size = parameter()  
  
t = 0  
position\_array = [[[] for j in range(3)] for i in range(star\_number)] # 记录星体的位置列表  
time\_array = [] # 记录时间  
flag2 = 0  
azim = -60 # 绘图视角  
elev = 30 # 绘图视角  
  
while t < total\_time:  
  
 # 利用龙格库塔法更新坐标  
 star\_parameter, t\_step, position\_array, time\_array = runge\_kutta(star\_parameter, star\_number,  
 position\_array, time\_array, t, time\_step)  
  
 # 绘图  
 elev, azim, flag2 = plotting(position\_array, star\_number, time\_array, time\_step, elev, azim, flag2, size)  
  
 t += t\_step  
  
# 生成动态图  
create\_gif(flag2)  
  
print('运行时间结束，程序终止')