Technical Report/Documentation

Edison Murairi

December 2020

1 Introduction and Pre-requisites

This program has two parts - the main program writen in C, and a python script *rk.py* which is an alternative script to solve the equations of motion. The main part requires a C compiler, and the python script requires *Python 3*, and the module *Numpy*.

2 Compiling and Running

To generate all the executables, type the command:

make
[This command will generate all the executables.]

The main executable is called output. Therefore, the simplest way to run the program is by typing the command:

.\output

The program can also be ran with arguments. It can receive two arguments: the small mass m, and the big mass M. For example, to run the program with $m = 2$, and $M = 15$ (in some units), type the command:

.\output -m 2 -M 15

If no argument is given, the program will use the default values $m = 1$, and $M = 10$.

The package also comes with a Python script called *rk.py*, which solves the equations of motion for $\frac{M}{m} = 60$. If desired, this can be used, and modified to accommodate other mass ratios.

Note that currently, the equations of motion solver in the main program does not produce the correct result. Therefore, it is preferable to use *rk.py*.

3 Output Files

Upon completion, the main program will output three files:

- **gradient_results.csv**: This file contains the 2-D minus gradient of the effective potential in comma separated format. The columns are: x, y, $\frac{\partial U}{\partial x}$, and $\frac{\partial U}{\partial y}$
- **l4_orbits.csv**: This file contains the x, and y coordinates of the orbit near L_4 in comma separated format arranged in that order.
- **l5_orbits.csv**: This file contains the x, and y coordinates of the orbit near L_5 in comma separated format arranged in that order.

Upon completion, the python script *rk.py* will output two files:

- **l4.csv**: This file contains the x, and y coordinates of the orbit near L_4 in comma separated format arranged in that order.
- **l4.csv**: This file contains the x, and y coordinates of the orbit near L_5 in comma separated format arranged in that order.

4 Description of the Main Program

The main program contains 7 modules: **parameters.c, diff_equations.c, rungeKutta.c, helper_subroutines.c, potential.c, gradient.c, and stable_orbits.c**. The main file is called **main.c**. Next, we will describe each module.

4.1 parameters.c

This module defines the basic parameters used for further calculations. These parameters are $k = \frac{m}{M}$, $\lambda = \frac{k}{1+k}$, $r_m = (1 - \lambda)R$, and $rM = \lambda R$.

4.2 diff_equations.c

This module defines the differential equations of motion derived from the Hamiltonian in [1]. It also contains the definition its helper function S, and s given in [1].

4.3 rungeKutta.c

This module implements the 4th order Runge Kutta method to solve the differential equations of motion defined in **diff_equations.c**. The algorithm is as follows:

Given the initial coordinates r_0, ϕ_0, p_0, and l_0 at time t_0, and the number of iterations n, to calculate the coordinate at time t, do as follows:

```
compute $h = \frac{t-t_0}{n}$

define i = $t_0$

while i < t:

k1 = h * f(r, t)
k2 = h * f(r + 0.5 * k1, t + 0.5*h)
k3 = h * f(r + 0.5 * k2, t + 0.5*h)
k4 = h * f(r + k3, t + h)
v += (k1 + 2*k2 + 2*k3 + k4)/6
i + h
```

where f is a vector containing the differential equations $[\dot{r}, \dot{\phi}, \dot{p}, \dot{l}]$, and v contains the soltuion in that order.

We see that this function will then perform $\mathcal{O}(n)$ computations. Therefore, its complexity is $\mathcal{O}(n)$.

4.4 helper_subroutines.c

This module contains the subroutines necessary for the flow of data. Currently, these functions are for writing, and can be expanded as needed. The functions are:

- write_results: This function takes an integer N as the number of items, three arrays, and a string for the name of the output file. It writes the data into the output file as comma separated value. Each array represents a column.
- write_gradient: This function takes an integer N, four arrays, and a string as the output file. It write the content of the four arrays into the file as comma separated value. Each array will represent a column in the output file.

The pseudo codes for each is as follows:

```
for item in array 1, ... k (k arrays):

write item(array1), item(array2), ... item(array k) on outputfile
```

Therefore, they are of complexity $\mathcal{O}(N)$ where N is the number of item.

We have two such functions because the write_gradient is specific while the other is more general. We used the write_results to write the orbits on files.

4.5 potential.c

This module defines the function to evaluate the dimensionless potential given in [1].

4.6 gradient.c

This module uses the definition of the potential in **potential.c** to compute its gradient. The pseudo codes are given below:

```
Define a rectangular grid of size $m$ by $n$
Define a small constant $h$
For each point of the grid, compute
```

$$\frac{\partial U}{\partial x} \approx \frac{U(x+h,y)-U(x,y)}{h} \tag{1}$$

$$\frac{\partial U}{\partial y} \approx \frac{U(x,y+h)-U(x,y)}{h} \tag{2}$$

```
write the results using the module write_gradient.
```

Because each point on the grid is processed only twice, the complexity is $\mathcal{O}(4mn) = \mathcal{O}(mn)$.

4.7 stable_orbits.c

This module uses the **rungeKutta.c** module to calculate the stable orbits near the Lagrange points L_4 and L_5. The pseudo codes are given below:

```
Define x4,y4 as the coordinates of $L_4$
Define x5, y5: as the coordinates of $L_5$
Define a time interval $t_0$ to $t_{max}$, and the number of time points in the intverval as $N$
For each time t in the interval:
run rungeKutta with initial conditions the coordinates of $L_4$ and $L_5$ with 0 velocities.
Write the results using the modulewrite_result.c
```

Therefore, we see that this algorithm runs runge kutta N times. Since the complexity of runge kutta is $\mathcal{O}(n)$ where n is the number of iterations, the complexity of this module is $\mathcal{O}(Nn)$.

Note: The python module *rk.py* has the same complexity because the only thing it does is implementing **rungeKutta.c** in python, and calculating the stable orbits this same way.

4.8 main.c

This module takes the free variables such as masses, and call the modules to compute the gradient, and to compute the orbits around L_4 and L_5.

The figure below summarizes how the main program works. The python script *rk.py* is an implementation of the left branch of the three in the figure.

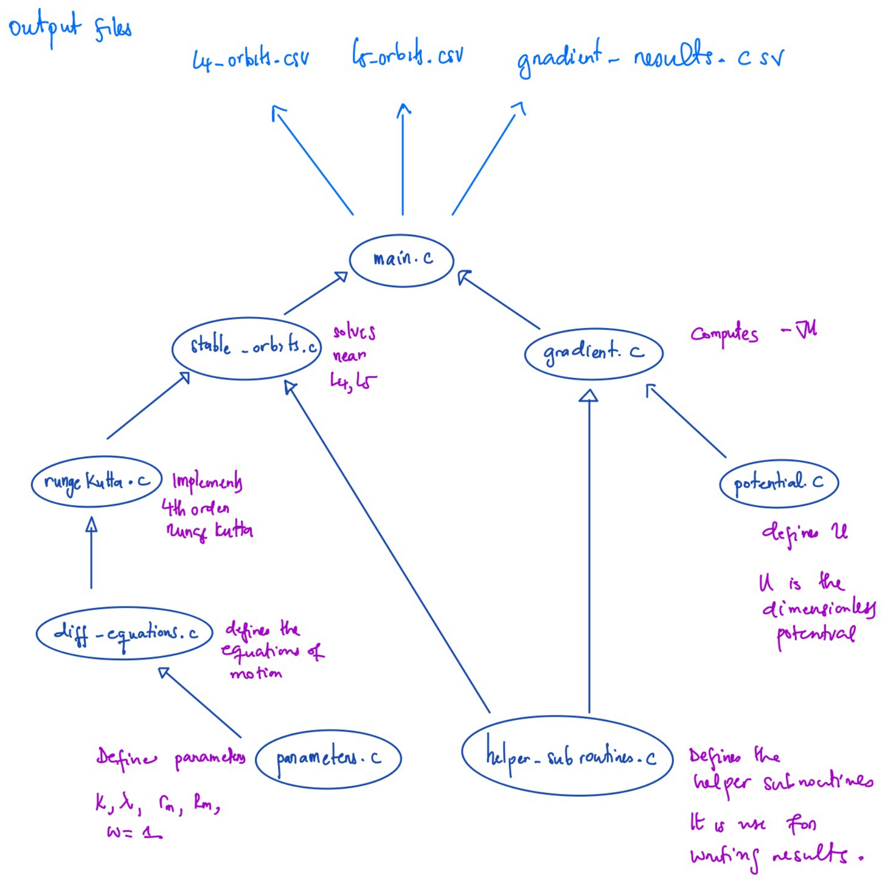
output files
L4-orbits.csv
L5-orbits.csv
gradient-results.csv
main.c
stable-orbits.c
solves near L4, L5
gradient.c
Computes $-\nabla\mathcal{U}$
runge Kutta.c
Implements 4th order runge kutta
potential.c
defines $\mathcal{U}$
$\mathcal{U}$ is the dimensionless potentral
diff-equations.c
defines the equations of motion
Define parameters
parameters.c
$k, \lambda, r_m, R_m,$ $\omega = 1$
helper-subroutines.c
Defines the helper subroutines
It is use for writing results.

Figure 1: Schematic Representation of the Functioning of the Main Program. The python part *rk.py* corresponds to the left branch of the three.

References

[1] Harald Grieshammer. Liberation motion in the celestial circular restricted three-body problem. *Fall 2020 Computational Physics 1 Project Description*, 2020.