UNIVERSITY OF OSLO

COMPUTATIONAL PHYSICS

Project 5

Authors:

Birgitte Madsen, 66

Magnus Isaksen, 14

Soumya Chalakkal, 51

Autumn 2015

UiO : University of Oslo

Department of Physics
University of Oslo
Sem Sælands vei 24
0371 Oslo, Norway
+47 22 85 64 28
http://www.mn.uio.no/fysikk/english/

Course:

Computational Physics

Project number:

5

Link to GitHub folder:

`https://?????`

Hand-in deadline:

Friday, December 11, 2015

Project Members:

Birgitte Madsen, 66
Magnus Isaksen, 14
Soumya Chalakkal, 51

Copies: 1
Page count: 11
Appendices: 0
Completed: ????, 2015

The content of the report is freely available, but publication (with source) may only be made with the agreement of the authors.

ABSTRACT

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

CHAPTER

2 METHOD

The source codes for the algorithms described in this chapter can be found in the Github folder `https://?????.` [1]

2.1 Newtonian two-body problem in three dimension

[2] $\mathbf{r}(t)$ is the three-dimensional space vector consisting of the coordinated $(x(t), y(t), z(t))$, whilst $\mathbf{v}(t)$ is the three-dimensional velocity vector with coordinates $(v_x(t), v_y(t), v_z(t))$, both of which are dependent on time.

In general, the differential equation that is considered is

$$\frac{dy}{dt} = f(t, y) \tag{2.1}$$

Which yields that

$$y(t) = \int f(t, y)dt \tag{2.2}$$

[3] For the two bodies in a three dimensional Newtonian gravitational field this corresponds to six coupled differential equations given by the vector equations

$$\frac{d\mathbf{r}}{dt} = \mathbf{v} \qquad \text{and} \qquad \frac{d\mathbf{v}}{dt} = -\frac{GM_1M_2}{r^3}\mathbf{r} \tag{2.3}$$

[4] in which M_1 and M_2 [5] are the masses of the two bodies, respectively, whilst r is the distance between the bodies. The equations in (2.3) are computed by the script given below in which $drdt$ corresponds to the derivative of the coordinates of the position, and $dvdt$ corresponds to the derivative of the velocity coordinates.

```
void Derivative(double r[3], double v[3], double (&drdt)[3], double (&dvdt)[3], double
    G, double mass){
```

[1] FiXme Note: fix these lines
[2] FiXme Note: make short intro, e.q. drawing
[3] FiXme Note: do we need to write y_{i+1} eq from p. 250 in lecture notes??
[4] FiXme Note: maybe we should divide by mass as on p. 248??
[5] FiXme Note: fix the this with M_1 and M_2

```
    drdt[0] = v[0];
    drdt[1] = v[1];
    drdt[2] = v[2];

    double distance_squared = r[0]*r[0] + r[1]*r[1] + r[2]*r[2];
    double newtonian_force = -G*mass/pow(distance_squared,1.5);
    dvdt[0] = newtonian_force*r[0];
    dvdt[1] = newtonian_force*r[1];
    dvdt[2] = newtonian_force*r[2];
}
```

2.1.1 Velocity-Verlet method

- Remember to write about accuracy of algorithm!! Consider the Taylor expansion of the vector function $\mathbf{x}(t_i \pm \delta t)$:

$$\mathbf{x}(t_i \pm \delta t) = \mathbf{x}(t_i) \pm \mathbf{v}(t_i)\delta t + \frac{\delta t^2}{2} \tag{2.4}$$

[6]

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \mathbf{v}(t)\delta t + \frac{1}{2}\mathbf{a}(t)\delta t^2 \tag{2.5}$$

$$\mathbf{v}(t+\delta t) = \mathbf{v}(t) + \frac{1}{2}(\mathbf{a}(t) + \mathbf{a}(t+\delta t))\delta t \tag{2.6}$$

The velocity is in the algorithm calculated [7] by first calculating

$$\mathbf{v}_{part1}(t+\delta t) = \mathbf{v}(t) + \frac{1}{2}\mathbf{a}(t)\delta t \tag{2.7}$$

and then use ?? [8] to determine $\mathbf{a}(t+\delta t)$, which is then used to compute the remaining term of Eq. (2.6) as

$$\mathbf{v}_{part2}(t+\delta t) = \frac{1}{2}\mathbf{a}(t+\delta t)\delta t \tag{2.8}$$

2.1.2 Fourth Order Runge-Kutta Method

- Remember to write about accuracy of algorithm!!

The Runge-Kutta method is based on Taylor expansions, with the next function value after a times step $\delta t = t_i - t_{i+1}$ being computed from four more or less improved slopes of the function in the points t_i, $t_i + \delta t/2$ and t_{i+1}.

[6] FiXme Note: fix this!!
[7] FiXme Note: ad to gange
[8] FiXme Note: fix this!

The first step of the RK4 method is to compute the slope k_1 of the function in t_i by

$$k_1 = \delta t f(t_i, y_i)$$

Then the slope k_1 at the midpoint is computed from k_1 as

$$k_2 = \delta t f(t_i + \delta t/2, y_i + k_1/2)$$

The slope at the midpoint is then improved from k_2 by

$$k_3 = \delta t f(t_i + \delta t/2, y_i + k_2/2)$$

from which the slope k_4 at the next step y_{i+1} is predicted to be

$$k_4 = \delta t f(t_i + \delta t, y_i + k_3)$$

From the computed slopes k_1, k_2, k_3 and k_4, the function value at $t_i + \delta t$ is computed as

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.9}$$

When implementing this for the two-body problem in three dimensions, it boils down to a continuous call of two functions, namely the function *Derivative* given in Sec. 2.1 and the function *updating_dummies* given below.

```
void updating_dummies(double dt, double drdt[3], double dvdt[3], double (&r_dummy)[3],
    double (&v_dummy)[3], double number, double (&kr)[3], double (&kv)[3], double
    r[3], double v[3])
{
   for (int i = 0; i<3; i++){
      kr[i] = dt * drdt[i];
      kv[i] = dt * dvdt[i];
      r_dummy[i] = r[i] + kr[i]/number;
      v_dummy[i] = v[i] + kv[i]/number;
   }
}
```

The function *updating_dummies* computes the values of k_1, k_2, k_3 and k_4 for all three space coordinates and velocity coordinates from the derivatives *drdt* and *dvdt* computed by the *Derivative* function. To compute the next step given by Eq. (2.9), the following succession of function calls are made until the time reaches the final time t_{final} after $(t_{final} - t_{inital})/\delta t$ time steps.

```
while(time<=t_final){
   Derivative(r,v,drdt,dvdt,G,mass);
   updating_dummies(dt,drdt,dvdt,r_dummy,v_dummy,2,k1r,k1v,r,v);
   Derivative(r_dummy,v_dummy,drdt,dvdt,G,mass);
   updating_dummies(dt,drdt,dvdt,r_dummy,v_dummy,2,k2r,k2v,r,v);
   Derivative(r_dummy,v_dummy,drdt,dvdt,G,mass);
   updating_dummies(dt,drdt,dvdt,r_dummy,v_dummy,1,k3r,k3v,r,v);
   Derivative(r_dummy,v_dummy,drdt,dvdt,G,mass);
   for (int i = 0; i<n; i++){
      k4r[i] = dt*drdt[i];
```

```
    k4v[i] = dt*dvdt[i];
}
for (int i=0; i<n;i++){
    r[i] = r[i] +(1.0/6.0)*(k1r[i]+2*k2r[i]+2*k3r[i]+k4r[i]);
    v[i] = v[i] +(1.0/6.0)*(k1v[i]+2*k2v[i]+2*k3v[i]+k4v[i]);
}
time += dt;
}
```

CHAPTER

3 RESULTS AND DISCUSSION

The results from running the codes described in Chap. 2 for computing the blah blah blah ?? can be found in the GitHub folder `https:/??`, together with the MatLab scripts for the plots presented in this chapter. [1]

[1] FiXme Note: fix these lines

CHAPTER

4

CONCLUSION

BIBLIOGRAPHY