Contents

1 Abstract

2 Introduction

3 Physical theory

3.1 Gravitation

In this project, a solar system will be studied. By solar system, I mean a system where only gravitational forces effect the bodies, and where there is a large mass fixed in origo[1]. Newtons gravitational law states that the gravitational force on a body with mass m from another body with mass M and relative position $\vec{r}$ is given by

$$\vec{F}_{\mathrm{G}} = -\frac{GmM}{\|\vec{r}\|^2}\vec{r}$$

where G is the gravitational constant, $6.67 \cdot 10^{-11}\ \mathrm{N\,m^2/s^2}$. The direction of the force is given by the fact that gravity is an attractive force. If one of the objects is the sun, the mass is denoted by $M_\odot$. With the sun placed in origo, r is simply the norm of the position vector of the planet with mass m.

If there are n planets in the solar system, in addition to the sun, the sum of the forces on planet

[1] This is a reasonable approximation, as the mass of the sun is much larger than the masses of the planets.

i with mass m_i is

$$\sum \vec{F}_i = \sum_{\substack{k=0 \\ k\neq i}}^{n} \frac{G m_i m_k}{\|\vec{r}_k - \vec{r}_i\|^3}(\vec{r}_k - \vec{r}_i)$$

with $m_0 = M_\odot$ and $\vec{r}_0 = \vec{0}$. From Newton's second law, we know that $\sum \vec{F}_i = m_i \vec{a}_i$, so the acceleration of planet i is given by

$$\vec{a}_i = \sum_{\substack{k=0 \\ k\neq i}}^{n} \frac{G m_k}{\|\vec{r}_k - \vec{r}_i\|^3}(\vec{r}_k - \vec{r}_i) \tag{1}$$

As the sun has been fixed to origo, $\vec{a}_0$ is set to $\vec{0}$.

3.2 Choice of units

In the solar system, seconds and meters are unpractical, as planets are millions of kilometers apart and take years to do one lap around the sun. As such, it is common to use so-called astronomical units, where 1 ua is the mean distance between the sun and the earth, and time is measured in years. To express the gravitational constant in these units, we can use that if the earth were moving in a circle around the sun, the acceleration in Newton's second law would be given by the sentripetal acceleration:

$$\frac{mv^2}{r} = \frac{GmM_\odot}{r^2} \implies G = \frac{r}{M_\odot} v^2 = \frac{1 \text{ ua}}{M_\odot} \cdot \left(\frac{2\pi \cdot 1 \text{ ua}}{1 \text{ yr}}\right)^2 = \frac{4\pi^2}{M_\odot} \cdot 1 \text{ ua}^3/\text{yr}^2$$

With this change of units, equation (1) can be written as

$$\vec{a}_i = \sum_{\substack{k=0 \\ k\neq i}}^{n} 4\pi^2 \frac{m_k}{M_\odot} \frac{\vec{r}_k - \vec{r}_i}{\|\vec{r}_k - \vec{r}_i\|^3} \cdot 1 \text{ ua}^3/\text{yr}^2 \tag{2}$$

4 Mathematical theory

Equation (2), together with some initial conditions, determines the motion of the bodies in the solar system. When written out in components, the equation gives a coupled set of differential equations. This set of equations is difficult, if at all possible, to solve analytically, so numerical work is required. As per usual, the time is discretised as $t_i = t_0 + ih$, where h is the time step, $h = (t_n - t_0)/n$. Acceleration, velocity and position are discretised correspondingly.

4.1 Forward Euler

The Forward Euler method, also called the Explicit Euler method, and hereafter called simply the Euler method, uses a first order Taylor polynomial to approximate a solution to the

diffrential equation. With $x'(t) = v(t)$ and $v'(t) = a(t)$, we have that

$$\vec{r}_i(t+h) \approx \vec{r}_i(t) + h\vec{v}_i(t)$$
$$\vec{v}_i(t+h) \approx \vec{v}_i(t) + h\vec{a}_i(t)$$

Discretised version:

$$\vec{r}_{i,j+1} \approx \vec{r}_{i,j} + h\vec{v}_{i,j}$$
$$\vec{v}_{i,j+1} \approx \vec{v}_{i,j} + h\vec{a}_{i,j}$$

To clarify the indices: $\vec{a}_{i,j}$ is the acceleration of planet i at time step j. This is calculated from equation (2) on the preceding page.

From Taylor's formula, the error for a first order Taylor polynomial goes as $O(h^2)$. This is the error made in each step — the error is cumulated, so the total error will be proportional h.

4.2 Velocity-Verlet

The Velocity-Verlet method, hereafter called the Verlet method, is based on a second order Taylor polynomial.

$$\vec{r}_i(t+h) \approx \vec{r}_i(t) + h\vec{v}_i(t) + \tfrac{1}{2}h^2\vec{a}_i(t)$$
$$\vec{v}_i(t+h) \approx \vec{v}_i(t) + h\vec{a}_i(t) + \tfrac{1}{2}h^2\vec{a}_i'(t)$$

There is no explicit expression for $\vec{a}'(t)$, however it can be approximated using the good old formula

$$\vec{a}'(t) \approx \frac{\vec{a}(t+h) - \vec{a}(t)}{h}$$

Since the acceleration is independent of the velocity, the newly updated position, $\vec{a}(t+h)$, can be calculated using $\vec{r}(t+h)$. Inserting this into the expression for $\vec{v}(t+h)$, we get

$$\begin{aligned}\vec{v}_i(t+h) &\approx \vec{v}_i(t) + h\vec{a}_i(t) + \tfrac{1}{2}h(\vec{a}_i(t+h) - \vec{a}_i(t)) \\ &= \vec{v}_i(t) + \tfrac{1}{2}h(\vec{a}_i(t) + \vec{a}_i(t+h))\end{aligned}$$

The discretised version then becomes

$$\vec{r}_{i,j+1} \approx \vec{r}_{i,j} + h\vec{v}_{i,j} + \tfrac{1}{2}h^2\vec{a}_{i,j}$$
$$\vec{v}_{i,j+1} \approx \vec{v}_{i,j} + \tfrac{1}{2}h\left(\vec{a}_{i,j} + \vec{a}_{i,j+1}\right)$$

The error of a second order Taylor polynomial is given as $O(h^3)$. The approximation for $\vec{a}'(t)$ has an error proportional to h, but this error is multiplied with h^2 when inserted into the expression for $\vec{v}_i(t+h)$. As such, the error for each step is proportional to h^3. The error is again cumulated for each step, so the total error will be proportional to h^2. This is one order better than the Euler method.

Code snippet 1: cpp_ui.cpp

```cpp
#include <cstdio>
#include <vector>
#include <string>
#include <string.h>
#include "vec.h"
#include "vec3.h"
#include "planet.h"
#include "solarsystem.h"

int main(int argc, char* argv[]){
    if(argc < 22 || (argc - 6) % 8 != 0){
        printf("%d command line arguments provided. 5+8n required for n
            bodies, 21 minimum.\n", argc-1);
        return 1;
    }
    char* filename = argv[1];
    char* method = argv[2];
    double t0 = atof(argv[3]);
    double tn = atof(argv[4]);
    int n = atoi(argv[5]);
    char* name;
    int i; double m;
    vector<Planet> planets;
    double x,y,z,vx,vy,vz;
    for(i=6; i < argc; i+=8){
        name = argv[i];
        m = atof(argv[i+1]);
        x = atof(argv[i+2]);
        y = atof(argv[i+3]);
        z = atof(argv[i+4]);
        vx = atof(argv[i+5]);
        vy = atof(argv[i+6]);
        vz = atof(argv[i+7]);
        Planet p(name,m,x,y,z,vx,vy,vz);
        planets.push_back(p);
    }
    SolarSystem solarsystem(filename,planets,t0,tn,n);
    if(strcmp(method,"euler")==0){
        printf("Solving with Forward Euler.\n");
        solarsystem.solve_euler();
    }
}
```

Code snippet 2: planet.h

```
#ifndef PLANET_H
#define PLANET_H
#include "vec3.h"
#include "vec.h"
#include <cmath>
#include <cstdio>
using namespace std;

class Planet{
    public:
        Planet(char* name_in, double mass_relative_to_sun, double x_in,
           double y_in, double z_in, double vx_in, double vy_in, double
           vz_in){
            name = name_in;
            mass_relation_4pi2  = 4 * M_PI * M_PI * mass_relative_to_sun;
            x = x_in;
            y = y_in;
            z = z_in;
            vx = vx_in;
            vy = vy_in;
            vz = vz_in;
            printf("Planet %s created with x=%f, y=%f, z=%f, vx=%f, vy=%f,
               vz=%f, m/M=%f\n",name, x,y,z,vx,vy,vz,mass_relative_to_sun);
        }
        void reset_acceleration(){
            ax = 0; ay = 0; az = 0;
        }
        void euler_update_position();
        void euler_update_velocity();
        void calculate_acceleration(Planet other);
        void set_dt(double dt_in){dt = dt_in;}
        double get_mass_relation_4pi2(){return mass_relation_4pi2;}
        double x, y, z, vx, vy, vz, ax, ay, az;
    private:
        char* name;
        double mass_relation_4pi2, dr_norm, dt;
};

#endif
```

Code snippet 3: planet.cpp

```
#include "planet.h"

void Planet::euler_update_position(){
    x += dt*vx;
    y += dt*vy;
    z += dt*vz;
    /*
    printf("%s, v: %f %f %f\n",name, vx, vy, vz);
    printf("%s, dt: %f\n",name, dt);
    printf("%s, v*dt: %f %f %f\n",name, dt*vx, dt*vy, dt*vz);
    */
}
void Planet::euler_update_velocity(){
    vx += dt*ax;
    vy += dt*ay;
    vz += dt*az;
    /*
    printf("%s, a: %f %f %f\n",name, ax, ay, az);
    printf("%s, dt: %f\n",name, dt);
    printf("%s, a*dt: %f %f %f\n",name, dt*ax, dt*ay, dt*az);
    */
}
void Planet::calculate_acceleration(Planet other){
    double dx, dy, dz;
    dx = other.x - x;
    dy = other.y - y;
    dz = other.z - z;
    double dr_norm = sqrt(dx*dx + dy*dy + dz*dz);
    double faktor = other.get_mass_relation_4pi2()/(dr_norm*dr_norm*dr_norm)
        ;
    ax += dx*faktor;
    ay += dy*faktor;
    az += dz*faktor;
}
```

Code snippet 4: solarsystem.h

```
#ifndef SOLARSYSTEM_H
#define SOLARSYSTEM_H
#include "planet.h"
#include <vector>
#include "vec3.h"
#include <string>
#include <cstdio>
using namespace std;

class SolarSystem{
    public:
        SolarSystem(char* filename, vector<Planet> planets_in, double t0_in,
            double tn_in, int n_in){
            planets = planets_in;
            number_of_planets = planets.size();
            t0 = t0_in;
            tn = tn_in;
            n = n_in;
            dt = (tn-t0)/(n-1);
            int i;
            for(i=0; i<number_of_planets; i++){
                planets[i].set_dt(dt);
            }
            printf("dt: %f\n",dt);
            file = fopen(filename,"w");
        }
        void finish(){fclose(file);}
        void solve_euler();
    private:
        vector<Planet> planets;
        int number_of_planets, n;
        double t0, tn, dt;
        FILE* file;
        void reset_accelerations();
        void calculate_accelerations();
        void euler_update_positions_velocities();
        void write_data(double t);
};

#endif
```

Code snippet 5: solarsystem.cpp

```
#include "solarsystem.h"

void SolarSystem::solve_euler(){
    int i;
    double t = t0;
    reset_accelerations();
    calculate_accelerations();
    write_data(t);
    for(i=0; i<n; i++){
        euler_update_positions_velocities();
        calculate_accelerations();
        write_data(t);
        reset_accelerations();
        t += dt;
    }
    finish();
}

void SolarSystem::reset_accelerations(){
    int i;
    for(i=0; i<number_of_planets; i++){
        planets[i].reset_acceleration();
    }
}

void SolarSystem::calculate_accelerations(){
    int i, j;
    for(i=0; i<number_of_planets-1; i++){
        for(j=i+1; j<number_of_planets; j++){
            planets[i].calculate_acceleration(planets[j]);
            planets[j].calculate_acceleration(planets[i]);
        }
    }
}

void SolarSystem::euler_update_positions_velocities(){
    int i;
    for(i=0; i<number_of_planets; i++){
        planets[i].euler_update_position();
        planets[i].euler_update_velocity();
    }
}

void SolarSystem::write_data(double t){
    fprintf(file,"%f",t);
    int i;
    for(i=0; i<number_of_planets; i++){
        fprintf(file," %f %f %f",planets[i].x,planets[i].y,planets[i].z);
        fprintf(file," %f %f %f",planets[i].vx,planets[i].vy,planets[i].vz);
        fprintf(file," %f %f %f",planets[i].ax,planets[i].ay,planets[i].az);
    }
    fprintf(file,"\n");
}
```