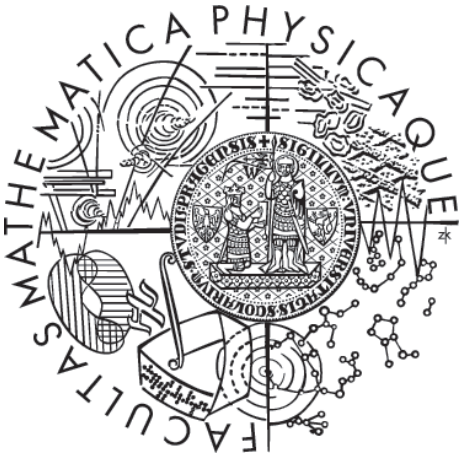
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Miroslav Tomášik

Rovnice geodetiky v prostorčasech s helikální symetrií
Geodesics in helically symmetric spacetimes

Department of Theoretical Physics

Supervisor of the bachelor thesis: Mgr. Martin Scholtz, Ph.D.
Study programme: Physics
Specialization: Mathematical modelling

Prague 2012

Acknowledgement

I would like to thank to my mother and father, who made possible for me to study what I like and are helping me as much as they can. Another great thank goes to my friend and supervisor Martin Scholtz, who has been supporting me from the beginning of my studies. Without his unlimited help, this thesis would never be finished.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources and with the help of my supervisor.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, date ..... signature of the author

Název práce: Geodetiky v helikálně symetrických prostoročasech

Autor: Miroslav Tomášik

Katedra: Ústav teoretické fyziky

Vedoucí bakalářské práce: Mgr. Martin Scholtz, Ph.D.
Ústav aplikované matematiky, ČVUT v Praze

Abstrakt: V této bakalářské práci zkoumáme rovnice geodetiky v helikálně symetrických prostoročasech v rámci linearizované Einsteinově gravitaci. Práce rozšiřuje připravovaný článek Bičák, Scholtz, Bohata[2]. Nejprve zavedeme standardní numerické metody pro řešení soustavy obyčejných diferenciálních rovnic, jež poté aplikujeme na newtonovské řešení popisující binární systém. Následně prezentujeme helikálně symetrické řešení linearizovaných Einsteinových rovnic a numerický kód řešící rovnice geodetiky na zadaném pozadí. Diskutujeme podmínky existence tohoto řešení a nakonec prezentujeme výsledky získané numerickou simulací. Uvádíme několik konkréntích příkladů geodetik, vybrané fázové portréty získané metodou Lyapunovových exponentů a znázorňujeme kauzální strukturu helikálně symetrického prostoročasu.

Klíčová slova: obecná relativita, helikální symetrie, numerická relativita

Title: Geodesics in helically symmetric spacetimes

Author: Miroslav Tomášik

Department: Department of Theoretical Physics

Supervisor: Mgr. Martin Scholtz, Ph.D.
Department of Applied Mathematics, CTU in Prague

Abstract:

In this bachelor thesis we investigate geodesics in helically symmetric spacetimes in the framework of linearized Einstein's gravity. Work is an extension of paper by Bičák, Scholtz and Bohata[2] which is under preparation. First we introduce standard numerical methods for solving systems of ordinary differential equations. Next we present helically symmetric solution of linearized Einstein's equations and numerical code solving the geodesic equation on given background. We discuss conditions of existence of helically symmetric solution and finally we present selected results obtained by numerical simulations. We give present few particular examples of geodesics, selected phase portraits obtained by the method of the Lyapunovov exponents and visualize the causal structure of helically symmetric spacetime.

Keywords: general relativity, helical symmetry, numerical relativity

Contents

Introduction

One of the most interesting and important open questions in Einstein's general theory of relativity is the existence of helically symmetric spacetimes, i.e. the existence of solutions of Einstein's equations possessing helical symmetry. Roughly speaking, helically symmetric solutions describe sources moving along closed trajectories in a periodic manner. In the context of the theory of gravity, the simplest example of helically symmetric system is the so-called binary system consisting of two black holes or neutron stars orbiting about their common centre of mass at constant angular velocity. Such systems are of particular interest as they can serve as the sources of gravitational waves which could be detected by terrestrial detectors.

While the helically symmetric solutions have been proven to exist in relativistic theories on the flat background (electrodynamics[8], scalar gravity[1]), in general relativity even the very notion of "helical symmetry" appears to be subtle and have not been defined satisfactorily yet. The main difficulty is due to fact that in general curved spacetime one cannot canonically identify points lying in different spacelike slices. Nevertheless, in linearized Einstein's theory, these obstacles are not present and one can define usual helical Killing vector with reference to flat background metric.

Two-body problem in linearized gravity

In work by Bičák and Scholtz [2] it was shown that helically symmetric solution describing two particles forming a binary system indeed exists and yields correct Newtonian limit. Moreover, asymptotic properties of this solution have been analyzed. In order to show that the equilibrium configuration of two-body system is feasible, the authors proceed in a following way.

First they compute gravitational field produced by one of the particles under consideration (this particle is called A-particle for brevity) which is represented by the trace-reversed metric tensor $\bar{h}^{\pm}_{\mu\nu}$ subject to linearized Einstein's equations. In usual "Lorenz gauge", Einstein's equations reduce to inhomogeneous wave equation with the energy-momentum tensor on the right hand side. Wave equation possesses retarded and advanced solutions, the former denoted by $-$, the latter by $+$, which can be obtained by usual technique of Green's functions.

The next step is to compute the Christoffel symbols ${}^{\pm}\Gamma^{\rho}_{\mu\nu}$ of affine connection and analyze the geodesic equation of the second particle forming the binary system, called B-particle. The geodesic equations have familiar form

$$\ddot{x}^{\rho} = -\,{}^{\pm}\Gamma^{\rho}_{\mu\nu}\,\dot{x}^{\mu}\,\dot{x}^{\nu}. \tag{1}$$

System of both particles will be in equilibrium (and thus exhibit a helical symmetry) if by adjusting the parameters of the system one can set $\ddot{x}^{\mu} = 0$. The problem is simplified heavily in the co-rotating frame in which the four-velocity $\dot{x}^{\mu}$ has one component only. Then the conditions of equlibrium reduce to

$${}^{\pm}\,\Gamma^{\rho}_{00}|_{B} = 0,$$

where the subscript B indicates that the Christoffel symbols are evaluated at the position of B-particle.

An immediate result is that the angular component of equilibrium condition is always non-vanishing for any choice of the parameters. This is a consequence of finite speed of the field propagation. However, taking retarded and advanced solution in a symmetric way,

$$\bar{h}_{\mu\nu} = \frac{1}{2}\left(\bar{h}^{+}_{\mu\nu} + \bar{h}^{-}_{\mu\nu}\right),$$

the angular force can be eliminated so that the angular component of equilibrium condition is satisfied identically. The analysis performed in [2] shows that then only the radial component of geodesic equation is non-trivial and the condition of equilibrium becomes

$$\Gamma^{r}_{00}|_{B} = 0.$$

This equation is solved numerically. In [2] it is shown that for appropriate choice (to be discussed later) of the parameters an equilibrium condition can be satisfied. Thus, the existence of helically symmetric solution in linearized Einstein's gravity have been established.

Helical symmetry in GR

As we explained in the previous section, helical symmetry represents a kind of periodic motion. Periodicity in general is genuinely non-local notion, for any observer must "wait" at least one period to observe periodic motion. However, in the flat spacetime (and in the linearized gravity as well), the helical symmetry is a *local* isometry generated by the Killing vector

$$\xi = \partial_t + \omega \partial_\phi, \tag{2}$$

where ω is common angular velocity of both orbiting particles and ϕ. In the Minkowski spacetime, both vector fields ∂_t and ∂_ϕ are the Killing vectors and therefore their linear combination with constant coefficients is also the Killing vector. A crucial point is that the orbits of ∂_ϕ are closed (spacelike) curves. This is the reason why the existence of Killing vector ξ implies the existence of *discrete* symmetry in time – periodicity.

Based on the last observation, it has been suggested by Bonnazola et al. [3] how to generalize definition of helical Killing vector to curved spacetimes. According to their definition the Killing vector ξ is called *helical* if it can be written (non-uniquely) in the form (2) where ∂_t is timelike vector and ∂_ϕ is spacelike vector with closed orbits. The difference between flat helical Killing vector and the definition [3] is that now ∂_t and ∂_ϕ are not assumed to be Killing vectors, only their combination (2).

Slightly more general definition has been given by Friedman et al. [4]. Their definition tries to express the non-local character of time-periodicity. Killing vector ξ with the flow χ on manifold M is called *helical* if there exists smallest $T > 0$ such that points P and $\chi_T(P)$ are timelike separated for each

$$P \in M - \mathcal{T},$$

where $\mathcal{T}$ is the history of a spacelike two-sphere $\mathcal{S}$,

$$\mathcal{T} = \{\chi_t(\mathcal{S}) | t \in \mathbb{R}\}.$$

The reason why we exclude the history of two-sphere $\mathcal{S}$ from the manifold is that if the initial point of the orbit of ξ is chosen under the black hole horizon (having S^2 topology), the orbit cannot escape from the black hole.

We can see that in order to define the helical symmetry one cannot deal merely with the local properties of the spacetime. Either we have to assume the orbits of ∂_ϕ to be closed, or we have to consider timelike translation by finite parametric distance T.

Goals of the thesis

Work [2] was intended as a step towards understanding helical symmetry in full general relativity. The goal of this bachelor thesis is to continue the analysis of helically symmetric spacetimes. One can hope that by detailed analysis of the linearized situation it is possible to get a hint how to construct helically symmetric solution in non-linear case of full Einstein's equations.

Global properties of the spacetime can be characterized by its causal structure, i.e. the structure of null cones, and by its geodesics. That is the goal of present bachelor thesis. In this work we solve geodesic equations using the connection obtained in [2]. It is worth to emphasize that in [2] the connection was used merely to obtain the condition of equilibrium. Technically, the Christoffel symbols were always evaluated at the position of A- or B- particle. Expressions for the Christoffel symbols simplify significantly in this case. In order to solve the geodesic equation, however, we have to consider full expressions for Christoffel symbols and solve the second order system of ordinary differential equations (1). There is, unfortunately, no hope to solve this system analytically and so we are reliant to numerical solution.

The goals of the thesis may be summarized as follows:

- implement the calculation of the Christoffel symbols;
- implement appropriate numerical method to solve the geodesic equation (1);
- find the light cones and geodesics.

1. Numerical methods

The problems we deal with in this thesis do not admit an analytical solution in a closed form and we have to employ numerical methods and implement them in appropriate computer language. In this chapter we introduce numerical methods used in the thesis and present their implementation in C++. In the next chapter we test them on the problem of motion in the Newtonian gravity. In this chapter we essentially follow the textbook by Garcia [5].

1.1 Dynamical system

Although the equations of motion are usually second order equations, they can be written as a system of first order equations which is often more convenient. For example, in the Hamiltonian formulation of classical mechanics, the system of N Lagrange equations of the second order can be split into the set of $2N$ first-order Hamilton equations.

In our notation, *dynamical system* is a system of N first-order ordinary differential equations

$$\dot{x}_a(s) = f_a(s, x(s)), \quad a = 1, 2, \dots N, \tag{1.1}$$

where x_i are unknown functions of parameter s and f_a are given functions. Values of x_i can be regarded as the coordinates in an abstract *phase space*, so that the solution

$$x_a = x_a(s)$$

can be visualised as a curve in the phase space. This curve is called the *phase trajectory* and functions f_a can be regarded as the components of the vector tangent to the phase trajectory.

If functions f_i do not depend on parameter s explicitly, dynamical system is called *autonomous*, otherwise it is *non-autonomous*. Notice, however, that any non-autonomous system can be written as an autonomous one if parameter s is treated as a new dependent variable subject to trivial equation $\dot{s} = 1$. It is clear that in physical applications, the parameter s will be related to time. However, in the theory of relativity spatial and temporal coordinates are treated equivalently. In addition, we have to distinguish between time t associated with some frame of reference and the proper time of particle under investigation. In order to incorporate the time we introduce function

$$x_0 = t.$$

In Newtonian theory there is no difference between the time and proper time, so the variable x_0 is governed by equation

$$\dot{x}_0 = 1.$$

It will not be the case in relativistic theory, however.

To conclude, dynamical system is a system of first-order ODE's in the form

$$\boxed{\begin{aligned} \dot{x}_a(s) &= f_a(s, x(s)), \quad \text{(equations of motion)} \\ x_a(0) &= x_a^{(0)}, \quad \text{(initial conditions)} \\ a &= 0, 1, \dots N. \end{aligned}} \qquad \text{Dynamical system} \tag{1.2}$$

1.2 Euler method

The simplest method to solve the system of ordinary differential equations is that of Euler. It is based on discretized version of relation for derivative of the function. In the case of continuous function $x_a = x_a(s)$ we define its derivative by

$$\dot{x_a}(s) = \lim_{h \to 0} \frac{x_a(s+h) - x_a(s)}{h}$$

if the limit exists. Since the most of computers have only a finite memory, it is necessary to replace continuous functions by their discretized version. Thus, instead of continuous parameter s we introduce the sequence $s^{(k)}$ defined by

$$s^{(k)} \;=\; k\,h, \tag{1.3}$$

where h is a value sufficiently small to imitate continuous character of the true parameter s, e.g. $h = 10^{-6}$. Similarly, continuous functions $x_a(s)$ and $f_a(s)$ will be replaced by the sequence of values

$$x_a^{(k)} \;=\; x(s^{(k)}), \quad f_a^{(k)} \;=\; f_a(s^{(k)}, x^{(k)}).$$

Now, suppose that $x_a(s)$ is analytic. Then the value $x_a^{(k+1)} = x_a(s^{(k+1)})$ can be expanded in the Taylor series as follows:

$$\begin{aligned} x_a^{(k+1)} &= x_a(s^{(k+1)}) \;=\; x_a(s^{(k)} + h) \\ &= x_a^{(k)} \;+\; h\,\dot{x}_a(s^{(k)}) \;+\; \mathcal{O}\left(h^2\right). \end{aligned} \tag{1.4}$$

On the other hand, using the definition of dynamical system (1.2), we have

$$\dot{x}_a(s^{(k)}) \;=\; f_a(s^{(k)}, x^{(k)}).$$

Thus, neglecting terms of order $\mathcal{O}\left(h^2\right)$ in (1.4), we arrive at simple reccurent formula

$$x_a^{(k+1)} = x_a^{(k)} \;+\; h\,f_a^{(k)}. \tag{1.5}$$

Using this formula one can find approximate solution to (1.2) provided that the initial conditions are given. This simplest integration scheme is called *the Euler method*.

$$\boxed{\begin{array}{rcll} x_a^{(0)} & = & x_a(0) & \text{initial conditions, starting point} \\ x_a^{(k+1)} & = & x_a^{(k)} \;+\; h\,f_a^{(k)} & \text{integration step} \end{array}} \qquad \text{Euler's method} \tag{1.6}$$

Euler's method is so called *first order* method which means that its local truncation error Δ is of the second order,

$$\Delta \;=\; \mathcal{O}\left(h^2\right).$$

This method usually works with reasonable accuracy only for very simple dynamical systems. There exist another first-order methods, e.g. the Euler-Cromer method. As explained above, Euler's method is based on discretized version of the definition of derivative,

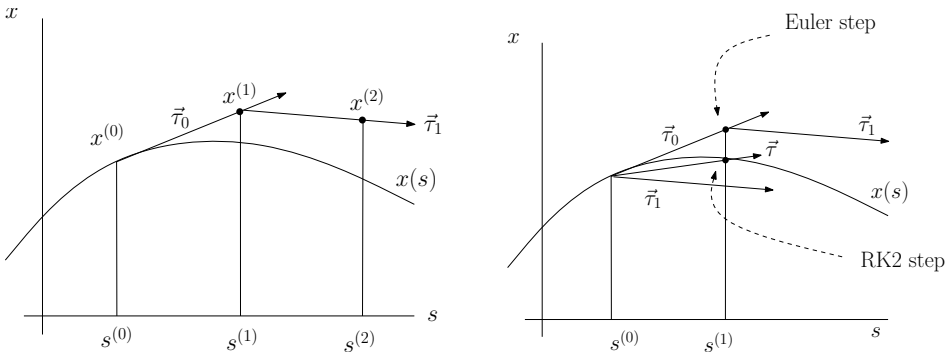
$$f'(x^{(k)}) \;=\; \frac{f(x^{(k+1)}) \;-\; f(x^{(k)})}{h}.$$

Because in this relation the derivative is computed from the future value of f, it is called the *forward derivative*. One can, however, use the *backward derivative*

$$f'(x^{(k)}) \;=\; \frac{f(x^{(k)}) \;-\; f(x^{(k-1)})}{h}.$$

Certain other variations of these relations (like the *centered derivative*) can be used to obtain methods which are formally still of the first order, but yield much better results. We omit discussion of these methods for several reasons.

Recall that we wish to solve the geodesic equation in the gravitational field of binary system. Nevertheless, the first step is to show that a binary system can exist at all! Thus, we will study the motion of one component of binary system (called B-particle) in the field of the other component (called A-particle). As will be explained in the following chapters, we will solve the geodesic equation of B-particle in particular coordinate system associated with the co-rotating frame. In these coordinates, binary system is in equlibrium (periodic motion exhibiting helical symmetry) if the coordinates of both particles are constant, i.e. first and second derivatives are zero for all times. As soon as we satisfy conditions of equilibrium and make these derivatives vanish, any numerical method leads to constant coordinates. For this reason, the Euler method works for circular orbits with the same accuracy as higher order method. But once we simulate non-circular trajectories (e.g. elliptic trajectories in the Newtonian case), none of the first order methods is accurate enough. In fact, we will need the fourth order Runge-Kutta method which is sufficient for all subsequent simulations, see below.



a) Euler method b) RK2 method

Figure 1.1: Comparison of the Euler method and the second-order Runge-Kutta method.

1.3 Runge-Kutta methods

Reason why the Euler method is necessarily inaccurate can be understood by naive geometrical intuition. Consider simple first-order differential equation of the form

$$\dot{x}(s) \;=\; f(s).$$

The solution $x = x(s)$ is schematically plotted in figure 1.3a. Let $(s^{(0)}, x^{(0)})$ be the initial point at which the simulation starts. The main idea of the Euler method is to construct the tangent $\vec{\tau}_1$ to the graph of the solution at the initial point. Then we move along this tangent by parametric distance h to point $(s^{(1)}, x^{(1)})$. This point is said to be the next point of an approximate solution. In figure 1.3, two points of an approximate solution are constructed. It is obvious that the trajectory obtained by the Euler method is necessarily diverging from the actual solution. Choosing smaller step h can increase the accuracy for few steps, but does not change the principle. This problem is particularly apparent when the trajectories representing the solution are closed curves, e.g. circles.

Thus, more sophisticated methods are needed. As we mentioned in the final part of previous section, there exist several first-order methods which are, however, significantly more accurate than the Euler method. For the reasons already explained, we do not discuss these methods in the thesis.

Runge-Kutta methods (RK methods) form an important family of numerical methods and allow one to compute the solution, in principle, to any desired order. We illustrate the superiority of RK methods over basic Euler method on the example of second-order Runge-Kuta method (RK2). Let us return to figure 1.3 and try another construction.

First, we proceed as in the Euler step: we start from the initial point $(s^{(0)}, x^{(0)})$ and construct the tangent $\vec{\tau}_0$. Then we move along $\vec{\tau}_0$ by parametric distance h to the point depicted as the Euler step in the figure 1.3b. At this point we construct the tangent $\vec{\tau}_1$. But now we transport this tangent to the initial point and construct the "average tangent"

$$\vec{\tau}_2 \;=\; \frac{1}{2}\,(\vec{\tau}_0 \;+\; \vec{\tau}_1)$$

and move *from the initial point* along $\vec{\tau}_2$ by parametric distance h so that we end at the point depicted as the RK2 step in the figure. We can see that, by construction, RK2 point is much closer to the actual solution than the solution obtained by the Euler method. It can be shown that the Runge-Kutta method just described is of the second order, which means that its local truncation error is

$$\Delta \;=\; \mathcal{O}\left(h^3\right).$$

Again, there exist several variants of this method which are still of the second order but can be more accurate in particular situations. In fact, variants where only halved Euler step is used are more usual. Our aim here was to illustrate how the original Euler method can be bettered by more cautious construction. For iteration scheme for RK2 method, see (1.7).

$$
\begin{aligned}
x_a^{(0)} &= x_a(0) \\
F_{a1} &= f_a\left(s^{(k)}, x^{(k)}\right) \\
F_{a2} &= f_a\left(s^{(k)} + h, x^{(k)} + h\,F_1\right) \\
x_a^{(k+1)} &= x_a^{(k)} + \frac{h}{2}\left(F_{a1} + F_{a2}\right)
\end{aligned}
\qquad \text{RK2 method} \tag{1.7}
$$

The most common method for integration of ordinary differential equations is the fourth-order Runge-Kutta (RK4) method with local truncation error

$$\Delta = \mathcal{O}\left(h^5\right).$$

In principle, it is possible to construct the Runge-Kutta method of an arbitrary order and thus achieve arbitrary precision. But, as one could suspect, computational effort increases rapidly with the order of method and the accuracy achieved is not worth the time needed to perform the calculations. It turns out that for the most of application, RK4 method is both sufficiently fast and accurate. Iteration scheme for RK4 method is shown in equation (1.8).

$$
\begin{aligned}
x_a^{(0)} &= x_a(0) \\
F_{a1} &= f_a\left(s^{(k)}, x^{(k)}\right) \\
F_{a2} &= f_a\left(s^{(k)} + \frac{h}{2}, x^{(k)} + \frac{h}{2}\,F_1\right) \\
F_{a3} &= f_a\left(s^{(k)} + \frac{h}{2}, x^{(k)} + \frac{h}{2}\,F_2\right) \\
F_{a4} &= f_a\left(s^{(k)} + h, x^{(k)} + h\,F_3\right) \\
x_a^{(k+1)} &= x_a^{(k)} + \frac{h}{6}\left(F_{a1} + 2\,F_{a2} + 2\,F_{a3} + F_{a4}\right)
\end{aligned}
\qquad \text{RK4 method} \tag{1.8}
$$

Although the Runge-Kutta methods can be further improved by implementing the adaptive step, for our puproses, RK4 method is sufficiently accurate as we will se in the next chapter. Inaccuracies will be treated by decreasing the integration step h. The only disadvantage of this approach rests in the loss of time but not the loss of precision.

1.4 Implementation

In this section we list the source code implementing numerical methods introduced above. These methods will be applied both to the Newtonian and the Einsteinian case and thus we wish to develop an algorithm independent of the system of equations to be solved. In order to achieve some degree of universality we split the problem (1.2) into two independent parts: evaluation of the right hand side of equations (1.2) and performing the integration step. While the first part is a matter of the system under consideration, the second part is a matter of method used to solve the system. In our algorithm we can therefore assume that some arbitrary functions f_a depending on the state variables are inserted as the arguments into the integration routine and proceed according to integration scheme (1.6) or (1.8). The head of each integration routine is of the form

```
void method_name (
   // reference to the function evaluating the right hand side
   void (*func)(
     double *f, // array of derivatives (to be evaluated)
```

```
      double *params, // additional parameters affecting calculations
      double *x  // array of current state variables
    ), // end of func declaration

    double *params,  // additional parameters passed to function func
    double *x,   // array of state variables (to be updated)
    int varsnum,  // number of variables present in x array
    double dt  // size of integration step
); // end of the head of method_name
```

Thus, each integration routine takes the pointer "func" referring to some function f_a which serves to evaluate the derivatives of unknown variables. According to the listing, this function may depend on some additional parameters stored in the pointer "params". Derivatives are evaluated at the point "x" and the resulting array of derivatives is written into array "f". Notice that values of additional parameters are not supposed to be used inside the integration routine and they are merely passed to "func". Full source code implementing the Euler method and RK4 method can be found in the listing 1.2. All routines concerning numerical solution of ordinary differential equations are part of library "ode"; for corresponding header file, see listing 1.1.

Listing 1.1: "ode.h" – header file for ODE library

```
1  #ifndef ODE_H_INCLUDED
2  #define ODE_H_INCLUDED
3
4  void euler( void (*func)(double *f, double * params, double * x),
5                   double *params, double *x, int varsnum, double dt );
6
7  void rk2( void (*func)(double *f, double * params, double * x),
8         double *params, double *x, int varsnum, double dt );
9
10 void rk4( void (*func)(double *f, double * params, double * x),
11        double *params, double *x, int varsnum, double dt );
12
13
14 #endif // ODE_H_INCLUDED
```

Listing 1.2: "ode.cpp" – library for ordinary differential equations

```
/* Euler method */
void euler( void (*func)(double *f, double * params, double * x),
                  double *params, double *x, int varsnum, double dt )
{
    double *f = new double[varsnum];
    func(f, params, x);
    for (int i = 0; i < varsnum; i++)
        x[i] += dt * f[i];
    delete[] f;
}


/* second order Runge-Kutta */
void rk2( void (*func)(double *f, double * params, double * x),
        double *params, double *x, int varsnum, double dt )
{
    double *f1 = new double[varsnum];
    double *f2 = new double[varsnum];
    double *x1 = new double[varsnum];
    func(f1, params, x);
    for (int i = 0; i < varsnum; i++)
        x1[i] = x[i] + dt * f1[i];
    func(f2, params, x1);
    for (int i = 0; i < varsnum; i++)
        x[i] += (f1[i] + f2[i]) * dt / 2;
    delete[] f1; delete[] f2; delete[] x1;
}


/* fourth order Runge-Kutta */
void rk4( void (*func)(double *f, double * params, double * x),
        double *params, double *x, int varsnum, double dt )
{
    double *f1 = new double[varsnum];
    double *f2 = new double[varsnum];
    double *f3 = new double[varsnum];
    double *f4 = new double[varsnum];
    double *x1 = new double[varsnum];
    double *x2 = new double[varsnum];
    double *x3 = new double[varsnum];
    func(f1, params, x);
    for (int i = 0; i < varsnum; i++)
        x1[i] = x[i] + dt * f1[i] / 2;
    func(f2, params, x1);
    for (int i = 0; i < varsnum; i++)
        x2[i] = x[i] + dt * f2[i] / 2;
    func(f3, params, x2);
    for (int i = 0; i < varsnum; i++)
        x3[i] = x[i] + dt * f3[i];
    func(f4, params, x3);
    for (int i = 0; i < varsnum; i++)
        x[i] += (f1[i] + 2*f2[i] + 2*f3[i] + f4[i]) * dt / 6;
    delete[] f1; delete[] f2; delete[] f3; delete[] f4;
    delete[] x1; delete[] x2; delete[] x3;
}
```

Slightly different case is the adaptive Runge-Kutta routine, for it must be able to change the magnitude of integration step "dt". One possibility is to pass the argument "dt" by reference, i.e. as a pointer, another possibility is to return value of the function. Here we have decided for the second possibility. Adaptive RK4 routine is presented in the listing 1.3. Notice that the routine cannot decrease the time step infinitely many times.

Listing 1.3: "Adaptive RK4 method

```
#define max_error 1E-8
#define min_error 1E-10
/* adaptive step Runge-Kutta */
double rka( void (*func)(double *f, double * params, double * x),
        double *params, double *x, int varsnum, double dt )
{
    double *x_single = new double[varsnum];
    double *x_double = new double[varsnum];
    double error;
    int steps = 0;

    do {
        for (int i=0; i<varsnum; i++)
            x_single[i] = x[i];
        for (int i=0; i<varsnum; i++)
            x_double[i] = x[i];
        // one long step
        rk4(func, params, x_single, varsnum, dt);
        // two half-steps
        rk4(func, params, x_double, varsnum, dt/2);
        rk4(func, params, x_double, varsnum, dt/2);
        // comparison
        error = 0;
        for (int i = 0; i<varsnum; i++)
          error += (x_single[i] - x_double[i])*(x_single[i] - x_double[i]);
        error = sqrt(error);
        if (error < min_error) dt = 10*dt;
        if (error > max_error) dt = dt*0.1;
        steps++;

    } while ((error > max_error)&&(steps<100));
    for (int i=0; i<varsnum; i++)
        x[i] = x_single[i];
    delete[] x_single;
    delete[] x_double;
    return dt;
}
```

2. Newtonian solution

As explained in the introduction, our goal is to find the geodesics of particles moving in the gravitational field produced by source with helical symmetry in linearized approximation of Einstein's equations. Since the geodesic equation of our interest is not solvable analytically, we adopt standard algorithms to solve it numerically. The purpose of this chapter is to introduce corresponding Newtonian equations of motion in appropriate coordinates in order to

- verify that our numerical algorithms give correct behaviour in the Newtonian case;
- compare relativistic solution to corresponding Newtonian solution.

2.1 Equations of motion

Derivation of Newtonian equations of motion is rather straightforward. We assume that the source of gravitational field is a point particle moving along the circular orbit of radius a at constant angular velocity ω. Its position vector as a function of time is given by

$$\boldsymbol{a}(t) = \begin{pmatrix} a\cos\omega t & a\sin\omega t & 0 \end{pmatrix}. \tag{2.1}$$

Gravitational field of the source is characterized by the field strentgth $\boldsymbol{K}$

$$\boldsymbol{K}(t,\boldsymbol{x}) = -G\,m\,\frac{\boldsymbol{x}-\boldsymbol{a}(t)}{|\boldsymbol{x}-\boldsymbol{a}(t)|^3} \tag{2.2}$$

where m is the mass of the source particle.

We wish to derive the equations of motion of the test particle in co-rotating frame. Co-rotating frame is a frame rotating about the $z-$axis at angular velocity ω, so that the source particle is at rest with respect to this frame. Coordinates associated with the co-rotating frame can be introduced as follows. Let (r,ϕ,z) be standard cylindrical coordinates defined by

$$x \;=\; r\cos\phi, \quad y \;=\; r\sin\phi, \quad z \;=\; z.$$

Cylindrical coordinates of the source parametrized by time read

$$r \;=\; a, \quad \phi \;=\; \omega\,t, \quad z \;=\; 0.$$

Thus, the only varying coordinate is the azimuth angle ϕ. Dependence of ϕ on time can be removed defining the coordinate

$$\hat{\phi} \;=\; \phi \;-\; \omega\,t,$$

for this coordinate is constant and equal to zero. Triple $(r,\hat{\phi},z)$ will be called co-rotating coordinates associated with the co-rotating frame.

Equations of motion will be derived in standard Lagrangian formalism. Kinetic energy of the test particle in co-rotating coordinates reads

$$T \;=\; \frac{1}{2}\left(\dot{r}^2 \;+\; r^2(\omega+\dot{\hat{\phi}})^2 \;+\; \dot{z}^2\right).$$

Notice that we did not include the mass of the particle because the motion in gravitational field does not depend on the mass of the test particle. Symbol m is reserved for the mass of the source. Components of the field strength with respect to co-rotating coordinates read

$$\begin{aligned} K_r &= -G\,m\,\frac{r \;-\; a\,\cos\hat{\phi}}{R^3}, \\ K_{\hat{\phi}} &= -G\,m\,\frac{a\,r\,\sin\hat{\phi}}{R^3}, \\ K_z &= -G\,m\,\frac{z}{R^3}, \end{aligned} \tag{2.3}$$

where

$$R = \sqrt{a^2 + r^2 + z^2 - 2ar\cos\hat{\phi}}. \tag{2.4}$$

Equations of motion follow from the Lagrange equations of the second kind,

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{q}_a} - \frac{\partial T}{\partial q_a} = K_a,$$

where $q_a = (r, \hat{\phi}, z)$ and $K_a = (K_r, K_{\hat{\phi}}, K_z)$. Resulting equations read

$$\begin{aligned}
\ddot{r} &= r(\omega + \dot{\hat{\phi}})^2 - m\,G\,\frac{r - a\cos\hat{\phi}}{R^3}, \\
\ddot{\hat{\phi}} &= -\,m\,G\,\frac{a\,\sin\hat{\phi}}{r\,R^3} - \frac{2\,\dot{r}}{r}\,(\omega + \dot{\hat{\phi}}), \\
\ddot{z} &= -\,m\,G\,\frac{z}{R^3}.
\end{aligned} \tag{2.5}$$

2.2 Choice of units

When solving differential equations on computer, it is often useful to adapt the units to problem under consideration in order to avoid using too large or too small values of the quantities involved. For example, when we study motion of Earth about the Sun, one astronomical unit (mean distance Earth–Sun) is more appropriate measure of distance than meter and one year is more appropriate time unit than second. In this section we transform equations of motion (2.5) in this manner.

However, our motivation here is to be able to compare the Newtonian solution to the solution obtained in the framework of linearized Einstein's gravity. Instead of angular velocity ω we introduce dimensionless parameter

$$\alpha = \frac{\omega\,a}{c} \tag{2.6}$$

so that we can expand relativistic solution in powers of α. Introducing the speed of light c in the Newtonian theory is somewhat artifical but it reflects what we will need later. Thus, we proceed completely as in the relativistic case.

First we introduce new time coordinate $x_0 = ct$ so that both time and lengths are measured in meters. Natural length associated with binary system is the radius of orbit of the source particle a. We define dimensionless coordinates $(\tilde{t}, \tilde{r}, \hat{\phi}, \tilde{z})$ by

$$\tilde{t} = \frac{c\,t}{a}, \quad \tilde{r} = \frac{r}{a}, \quad \hat{\phi} = \hat{\phi}, \quad \tilde{z} = \frac{z}{a}. \tag{2.7}$$

Next we define dimensionless mass parameter by

$$\tilde{m} = \frac{m\,G}{a\,c^2}. \tag{2.8}$$

Function R defined by (2.4) can be written in the form

$$R = a\,\tilde{R} = a\sqrt{1 + \tilde{r}^2 + \tilde{z}^2 - 2\tilde{r}\cos\hat{\phi}}.$$

Let us discuss some situations of particular interest to see what values dimensionless quantities introduced above acquire. Consider binary system Sun–Earth. For simplicity we assume that the orbit of the Earth is circular and has radius of one astronomical unit, in SI system,

$$r_E = 1\,\mathrm{AU} = 149\ 598\ 000\,\mathrm{km} \doteq 1,5 \times 10^{11}\,\mathrm{m}.$$

Masses of the Sun and the Earth are

$$\begin{aligned}
m_\odot &= 1,998\ 92 \times 10^{30}\,\mathrm{kg} \doteq 2 \times 10^{30}\,\mathrm{kg}, \\
m_E &= 5,974\ 1 \times 10^{24}\,\mathrm{kg} \doteq 6 \times 10^{24}\,\mathrm{kg}.
\end{aligned} \tag{2.9}$$

Radius of the orbit of the Sun $r_\odot$ can be calculated from the fact that the center of mass must be fixed, i.e. the condition

$$m_\odot\, r_\odot \;=\; m_E\, r_E$$

must be satisfied, from which

$$r_\odot = \; 447\;105,623\,\mathrm{m} \doteq 5,5\times 10^5\,\mathrm{m}. \tag{2.10}$$

In the notation used in this thesis, for the Sun–Earth system we have

$$a \;=\; r_S, \quad b \;=\; r_E,$$

since we simulate the motion of Earth in the gravitational field of the Sun. Now we can rescale units according to equations (2.7) and (2.8). Dimensionless mass of the Sun is

$$\tilde{m} = \frac{m_\odot\, G}{r_\odot\, c^2} \;=\; 3,3\times 10^{-3}. \tag{2.11}$$

Radius r_E measured in multiples of $r_\odot$ is given by dimensionless parameter

$$\tilde{b} \;=\; 334\;600.$$

Angular velocity of the Earth motion is

$$\omega \;=\; \frac{2\,\pi}{1\,\mathrm{yr}} \;=\; 1,992\times 10^{-7}\,\mathrm{s}$$

and corresponding dimensionless parameter

$$\alpha = \frac{\omega\, c}{r_\odot} \;=\; 2,983\times 10^{-10}. \tag{2.12}$$

We assumed, for simplicity, that the orbit of the Earth is circular, which is of course not true. Our aim is not, however, to perform exact simulation of the Earth motion, just to illustrate, what values our dimensionless coordinates can take for binary systems comparable to the Sun-Earth system.

Since now we will work in these units and thus omit the tildas for simplicity. Equations of motion (2.5) then read

$$\begin{aligned}
\ddot{r} &= -\,m\,\frac{r-\cos\hat{\phi}}{R^3} \;+\; r\,(\alpha \,+\, \dot{\hat{\phi}})^2,\\
\ddot{\hat{\phi}} &= -\,\frac{m\,\sin\hat{\phi}}{r\,R^3} \;-\; \frac{2\,\dot{r}}{r}\,(\alpha \,+\, \dot{\hat{\phi}}),\\
\ddot{z} &= -\,\frac{m\,z}{R^3}.
\end{aligned} \tag{2.13}$$

In (2.13), all quantities are dimensionless, dot means the derivative with respect to dimensionless time, and R is given by

$$R \;=\; \sqrt{1+r^2+z^2-2r\cos\hat{\phi}}.$$

All lengths are measured in multiples of the radius a. We will often refer to α as angular velocity, although it is a dimensionless parameter $\alpha = \omega a/c$.

2.3 Newtonian equilibrium

By an appropriate choice of parameter m, b and α one can always achieve the binary system to be in equilibrium. By *equilibrium* we mean helically symmetric configuration when both components of binary system are moving along the circular trajectories at the same (and constant) angular velocity. Condition for equilibrium can be obtained from equations of motion (2.13).

Imagine we insert a test particle at position with coordinates

$$r = b,\ \hat{\phi} = \pi,\ z = 0$$

and let its initial velocity be

$$\dot{r} = 0, \quad \dot{\hat{\phi}} = 0, \quad \dot{z} = 0.$$

Notice that condition $\dot{\hat{\phi}} = 0$ means that the particle is at rest in the co-rotating frame, i.e. it is orbiting with angular velocity α. Equations of motion (2.13) simplify to

$$\ddot{r} = -m\,\frac{1}{(1+b)^2} + \alpha^2\, b, \quad \ddot{\hat{\phi}} = \ddot{z} = 0.$$

The system will be in equilibrium if all second derivatives vanish. We can see that in the Newtonian theory the second derivative of angular coordinate vanishes automatically, which does not hold in relativistic theory, cf. (3.40). Derivative $\ddot{r}$ can be set to zero imposing the condition

$$\alpha_N = \sqrt{\frac{m}{b(1+b)^2}} \tag{2.14}$$

where the subscript N stands for "Newtonian". Equation (2.14) is the *Newtonian condition of equilibrium*. Its most important feature is that for arbitrary values of b and m there always exists α for which the system is in equilibrium. This α is unique up to sign which merely corresponds to the opposite sense of the rotation.

For initial conditions given above and for the value of α_N given by (2.14) the trajectory of the orbit of the test particle will be circular and its angular velocity will be constant. This fact can serve as a test of numerical methods to be used.

2.4 Implementation

Finally, after introducing necessary numerical methods in chapter 1 and introducing the Newtonian equations of motion in this chapter, we are ready to apply numerical methods and solve the Newtonian equations of motion. In the section 1.4 we have introduced general schema to implement the Euler method and RK4 method and presented the source code 1.2, page 9. We emphasized the universality of integration routines in the sense that they can be applied to arbitrary system of first-order ordinary differential equations. Here we provide routine implementing the right hand side of these equations, i.e. functions f_a in (1.2).

Notice, first, that although the equations of motion (2.13) are second-order equations, it is trivial to convert them to first-order system. Indeed, defining new variables

$$v_r = \dot{r}, \quad v_\phi = \dot{\hat{\phi}}, \quad v_z = \dot{z},$$

system (2.13) splits to six equations of the first order:

$$\begin{aligned}
\dot{r} &= v_r, \quad & \dot{v}_r &= -m\,\frac{r-\cos\hat{\phi}}{R^3} + r\,(\alpha + v_\phi)^2,\\
\dot{\hat{\phi}} &= v_\phi, \quad & \dot{v}_\phi &= -\frac{m\,\sin\hat{\phi}}{r\,R^3} - \frac{2\,v_r}{r}\,(\alpha + v_\phi),\\
\dot{z} &= v_z, \quad & \dot{v}_z &= -\frac{m\,z}{R^3}.
\end{aligned} \tag{2.15}$$

With this definition, the state of the particle moving in gravitational field is described not only by its coordinates but also by its velocities. The state vector "x" appearing in the integration routines in listing 1.2 consists of eight variables

$$x = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (t, r, \hat{\phi}, z, v_t, v_r, v_\phi, v_z).$$

Formally we treat time t as an independent variable. Its conjugated velocity in the Newtonian case is, of course,

$$v_t = \dot{t} = 1$$

and corresponding second derivative must vanish. However, in the relativistic case one has

$$\mathrm{d}t = \gamma\,\mathrm{d}s$$

where γ is the Lorentz factor and $\mathrm{d}s$ is the proper time, since in relativity we parametrize the worldlines by proper time rather than by inertial time.

Newtonian equations of motion (2.15) for the test particle moving in the gravitational field of the source particle moving periodically along circular orbits can be implemented as shown in the listing 2.1, where full program for solving equations of motion for arbitrary initial conditions is listed.

Listing 2.1: Newtonian solution with brief comments

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include "ode.h" // library for diff. equations

#define PI 3.14159265358 // Bulgarian constant


using namespace std;


/*
params[0] = m, params[1] = alpha
x = {t, r, hphi, z, u0, u1, u2, u3}
*/
void newton (double *f, double *params, double * x) {
    double R = sqrt( 1 + x[1]*x[1] + x[3]*x[3] - 2*x[1]*cos(x[2]) );
    double kappa = params[0] / (R*R*R);
    // derivatives of coordinates = velocities
    f[0] = 1;
    f[1] = x[5]; f[2] = x[6]; f[3] = x[7];
    // derivatives of velocities = acceleration
    f[4] = 0;
    f[5] = - kappa * (x[1] - cos(x[2])) + x[1]
           * (params[1] + x[6]) * (params[1] + x[6]);
    f[6] = - kappa * sin(x[2]) / x[1] -(2*x[5]/x[1])*(params[1] + x[6]);
    f[7] = - kappa * x[3];
}

/*
evaluates whether maximal time or max. number of orbits
have been reached
*/
bool finished(double *x, double *fin_params, double *params) {
    bool time = x[0] > fin_params[0];
    bool orbits = (x[2] + params[1]*x[0] - PI)/(2*PI) > fin_params[1];
    if (time) cout << "maximal time exceeded";
    if (orbits) cout << "number of orbits reached";
    return (time||orbits);
}


/*
 writes the state "x" of the system into file f
*/
void write_coordinates(ofstream *f, double *x, double *params) {
    // corotating coords. (1,2,3,4)
    for (int i = 0; i < 4; i++)
        (*f) << x[i] << " ";
    // phi = hphi + alpha * t (5)
```

```
52      double phi = x[2] + params[1] * x[0];
53      (*f) << phi << " ";
54      // Cartesian (6, 7)
55      (*f) << x[1] * cos(phi) << " " << x[1] * sin(phi);
56
57      (*f) << endl;
58      //
59      cout << "time = " << x[0] << ", r = " << x[1] << ",
60          orbits = " << (phi-PI)/(2*PI) << endl;
61  }
62
63
64  int main()
65  {
66      ofstream out1 ("newton.txt");
67      ofstream out3 ("source.txt");
68
69      double m = 3.3E-3; // mass of Earth
70      double b = 334600; // radius of Earth's orbit
71      double alpha = sqrt(m/b)/(1+b); //equilibrium angular velocity
72      double tmax= 2*PI/alpha; // estimated orbit
73      double ds = tmax * 1E-6;
74      tmax = tmax * 30; // max time = 10 orbits
75
76      // state vector, x = [t, r, phi, z, vt, vr, vphi, vz]
77      double xN[8] = {0, b, PI, 0, /**/ 1, 0, 1.05E-10, 0 };
78
79      double params[3] = {m, alpha}; // mass, angular velocity
80      double finish_params[2] = {tmax, 2};//max time, max number of orbits
81
82      double steps = 0;
83
84      //
85      cout << "alpha = " << params[1] << endl;
86      cout << "Start simulation pressing any key" <<endl;
87      cin.ignore(1);
88
89
90      while (!finished(xN, finish_params, params)) {
91          euler(newton, params, xN, 8, ds); //perform one integration step
92          steps++;
93          if (steps == 10000) {
94              // save current state to file
95              write_coordinates(&out1, xN, params);
96              // motion of the source
97              out3 << xN[0] << " " << cos(alpha*xN[0]) <<
98                  " " << sin(alpha*xN[0]) << " " << 0 << endl;
99              steps = 0;
100         }
101
102     }
103
104     out1.close();
105     out2.close();
106     out3.close();
107     return 0;
108 }
```

2.5 Selected numerical solutions

Now we are in position to test numerical methods introduced in chapter 1 and apply them to problem of motion in the Newtonian gravitational field. Let us emphasize that at this moment

only the test particle is subject to equations of motion. The source particle is, by definition, moving along circular orbit of radius 1 (in the units chosen in section 2.2) at angular velocity α. We do not investigate the reasons why the source is moving in prescribed way, we are interested just in the motion of the test particle in given gravitational field.

Our binary system is described by three parameters:

- mass of the source particle m,
- radius of the orbit of the test particle b,
- angular velocity α of the source particle.

The radius a of the orbit of the first particle has been eliminated by the choice of units and the motion of the test particle is not affected by its mass. Hence, there are no other parameters characterizing the system.

The program works as follows. First, the values of m and b are entered and relation (2.14) can be used to compute the equilibrium value of α, that is, the value of angular velocity for which the binary system can stay in equilibrium for arbitrarily long time. Then the length of one period of the test particle can be estimated as

$$T = \frac{2\pi}{\alpha}.$$

Initial position and initial velocity of the test particle are chosen to be

$$r(0) = b,\ \hat{\phi}(0) = \pi,\ z(0) = 0,\ \dot{r}(0) = \dot{\hat{\phi}}(0) = \dot{z}(0) = 0,$$

so that the test particle is at rest with respect to co-rotating frame. Then the simulation starts.

Program generates two files containing data in the plain text format. File "source.txt" contains informations about the motion of the source particle and contains only three columns: time, and the x-coordinate and $y-$coordinate of the source particle at given time. File "newton.txt" contains following 7 columns (numbers are taken from randomly chosen simulation):

t	r	$\hat{\phi}$	z	$\phi = \hat{\phi} + \alpha t$	x	y
1.37658	3	3.14159	0	3.20442	-2.99408	-0.188372
2.75315	3	3.14159	0	3.26726	-2.97634	-0.376
4.12973	3	3.14159	0	3.33009	-2.94686	-0.562144
...	...	...	...	...	...	...

Simulation ends when desired number of orbits of the test particle has been reached. If the trajectory of the test particle under given initial condition is not bounded, simulation ends after exceeding conveniently chosen maximal time. This maximal time is automatically computed as a multiple of estimated period T.

At the beginning of our analysis we will use the simplest Euler method. Usually, when modelling motion of test particles in the Newtonian gravitational field, one encounters serious difficulties following from the inaccuracy of this method. For example, trajectories of the test particles which should be closed according to theoretical prediction fail to be closed in the simulation. Employing the Runge-Kutta methods (or other more accurate methods) it is possible to recover closed orbits in numerical simulation. It is very important to identify errors caused by numerical algorithm, otherwise one can deduce from simulation physical phenomena which are in fact not present.

However, this problem is much less obvious in our case thanks to the coordinate system used. If we simulate planar motion in the Cartesian coordinates, both x and y are varying over wide range of values. In order to obtain closed orbit, the coordinates x and y must return to their initial values after one period. Thus, even a small inaccuracy in the numerical algorithm will cause the decay of the orbit. On the other hand, in the corotating frame the orbits are closed if relevant coordinates r and $\hat{\phi}$ are *constant*. They do not have to return to some initial values, they must be constant during one period all the time. If we satisfy the condition of equilibrium at the initial time, all derivatives will be zero and therefore the coordinates will remain constant regardless on the accuracy of the method being used. To conclude, the discrepancy between the most trivial Euler method and more sophisticated methods is much smaller than in usual applications.

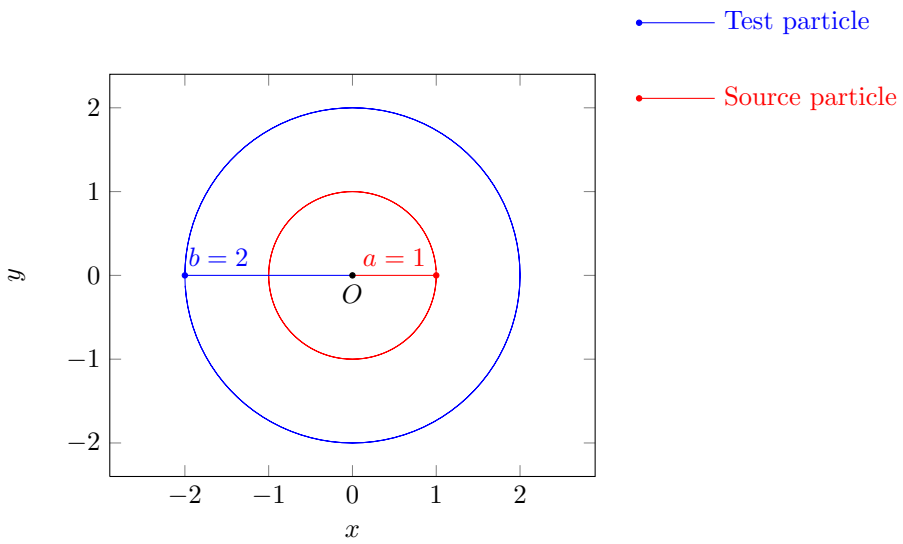


Figure 2.1: Values $m = 1$, $b = 2$ and equilibrium angular velocity $\alpha_N = 0,235\ 702$. Initial position of the test particle is $(x, y) = (-2, 0)$. The trajectory of the source particle starts at point $(x, y) = (1, 0)$ and it is a circle of radius $a = 1$. On the other hand, trajectory of the test particle is a result of simulation.

Nevertheless, we start our analysis with the Euler method and show that for extremely eccentric trajectories this method does not give correct answers and must be replaced.

Our first task is to test whether the trajectory of the test particle is indeed circular when the test particle is orbiting at equilibrium angular velocity α given by (2.14) and the initial velocity of the test particle is set to zero with respect to co-rotating frame. Under these initial conditions the test particle must stay at rest in co-rotating frame, which means that it must stay in uniform circular motion in the inertial frame.

In order to verify that our simulation yields the same result we subsequently choose several values of parameters m and b. Figure 2.1 shows the simulation for $m = 1$ and $b = 2$. For equilibrium value of $\alpha_N = 0,235\ 702$ we obtain perfectly circular orbit of radius $b = 2$. In this case, numerical simulation gives expected result. We can ask what happens if we choose slightly different value of α. The answer is not obvious, however, because the source particle is in its own circular motion regardless on the motion of the test particle. Thus, choosing $\alpha \neq \alpha_N$ does not produce usual solution of the two-body Kepler problem with eliptical orbits! Rather it yields chaotic motion of the test particle driven by periodic gravitational force. Later we demonstrate that this is indeed the case when the radius a is comparable to radius b, or, in our dimensionless units, when $b \approx 1$.

On the other hand, when $b \gg 1$, the motion of the source particle should not affect the motion of the test particle. For example, we have seen in the section 2.2 that for the system Earth-Sun the values of parameters are

$$b = 334\ 600, \quad m = 3,3 \times 10^{-3}.$$

Again, for the equilibrium value of $\alpha_N = 2,983 \times 10^{-10}$ we recover circular motion of "the Earth", see figure 2.2. Now, if we perturb the value of b slightly but leave α_N unchanged, circular trajectory of the test particle must be deformed into an ellipse.

In following examples we choose a value of b, compute equilibrium angular velocity α_N, but then choose the initial radius $r(0)$ to be

$$r(0) = k\,b$$

where k is a coefficient close to unity. We investigate the influence of k on the character of orbits. Results for several choices of k are plotted in figure 2.3. Recall that these solutions have been obtained by the Euler method.

From figure 2.3 we can see that elliptical character of the trajectories is more apparent for the choice $k < 1$. To see that the Euler method indeed breaks down for highly elliptical

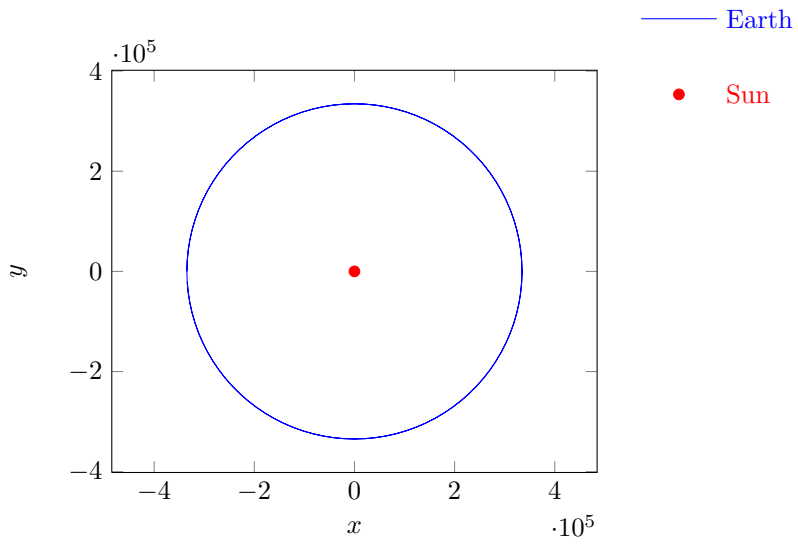


Figure 2.2: Values $m = 3,3 \times 10^{-10}$, $b = 334\,600$ (notice the scaling of the axes) and equilibrium angular velocity $\alpha_N = 2,983 \times 10^{-10}$ corresponding to circular motion of the Earth about the Sun. Trajectory of the Sun is represented by the single point at the origin. We can see that the motion remains circular after one period which justifies the numerical method used in this case.

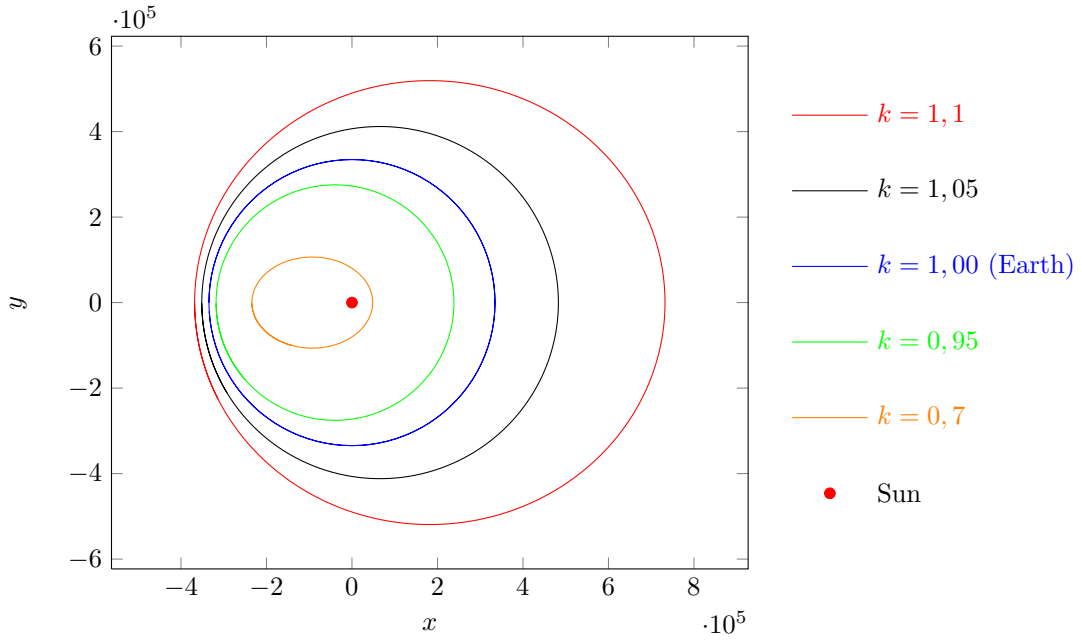


Figure 2.3: Again, we consider parameters for the Earth-Sun system, $m = 3,3 \times 10^{-10}$, $b = 334\,600$ and equilibrium angular velocity $\alpha_N = 2,983 \times 10^{-10}$, but the initial radius is chosen to be $r_0 = kb$ where the coefficient k acquires values close to one. Obtained trajectories are in accordance with theoretical predictions.

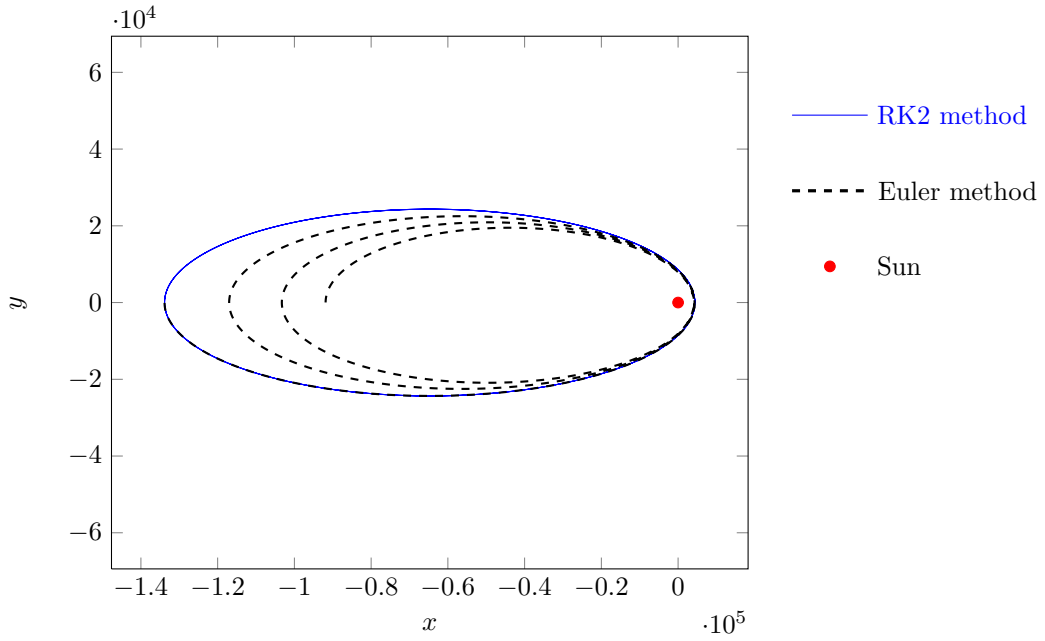


Figure 2.4: Parameters for the Earth-Sun system, $m = 3,3\times10^{-10}$, $b = 334\,600$ and equilibrium angular velocity $\alpha_N = 2,983\times10^{-10}$, but the initial radius is chosen to be $r_0 = kb$ where $k = 0.4$. First three "orbits" of the test particle are shown: Euler method breaks down. Simulation based on the Runge-Kutta 2nd order method yields correct result.

trajectories, let us choose $k = 0.4$. Corresponding solution obtained via Euler method is plotted in the figure 2.4. From the exact solution of the problem we know that the trajectory of the test particle must be closed ellipse but the Euler method produces decaying orbit. Had we not known the exact solution, we might think that the decay is due to some physical effect. Hence, it is absolutely necessary to test numerical methods on the problems with known solution, otherwise we cannot trust any result obtained by simulation. The same figure 2.4 shows, however, that more accurate Runge-Kutta method of second order gives correct result, closed ellipse with high eccentricity. Comparison of RK2 and RK4 method for even more eccentric trajectories is shown in figure 2.5.

Now we can return to the question what happens if the equilibrium configuration is slightly perturbed, but when the radius b is comparable to the radius a, or, in our units, if $b \approx 1$. In such a case, the equilibrium configuration is expected to be unstable because the motion of the source particle is not negligible compared to the motion of the test particle. Thus, a small perturbation should lead to diverging and chaotic trajectories. Figure 2.6 shows two trajectory with initially close points with radii $1,03b$ and $1,04b$. As can be seen in the figure, these trajectories slowly diverge from each other.

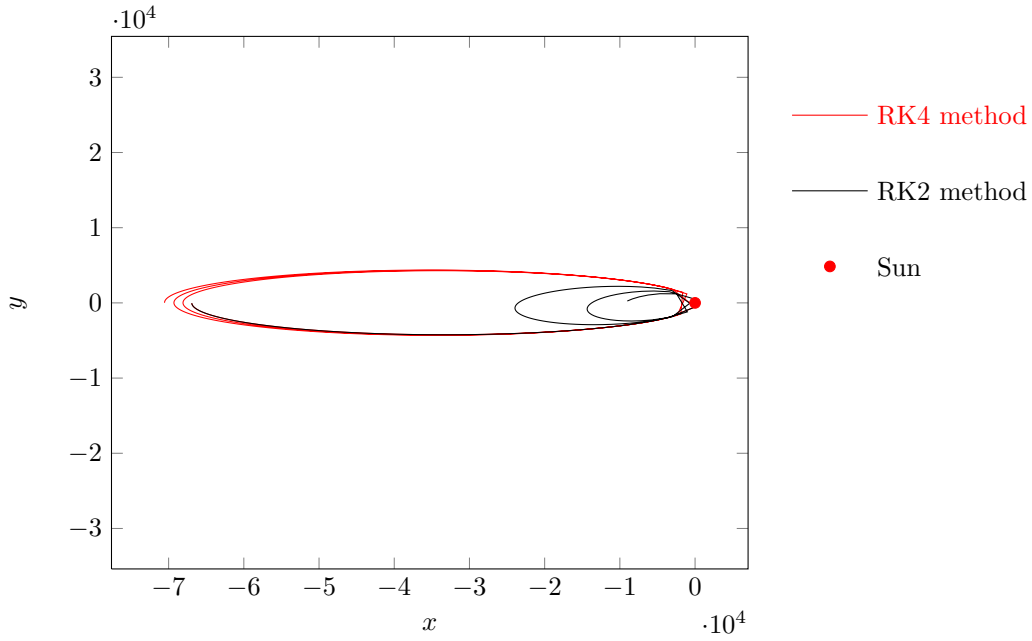


Figure 2.5: Parameters for the Earth-Sun system, $m = 3,3 \times 10^{-10}$, $b = 334\,600$ and equilibrium angular velocity $\alpha_N = 2,983 \times 10^{-10}$, but the initial radius is chosen to be $r_0 = kb$ where $k = 0.2$. RK2 method produces completely unsatisfactory solution, RK4 method gives much better result.

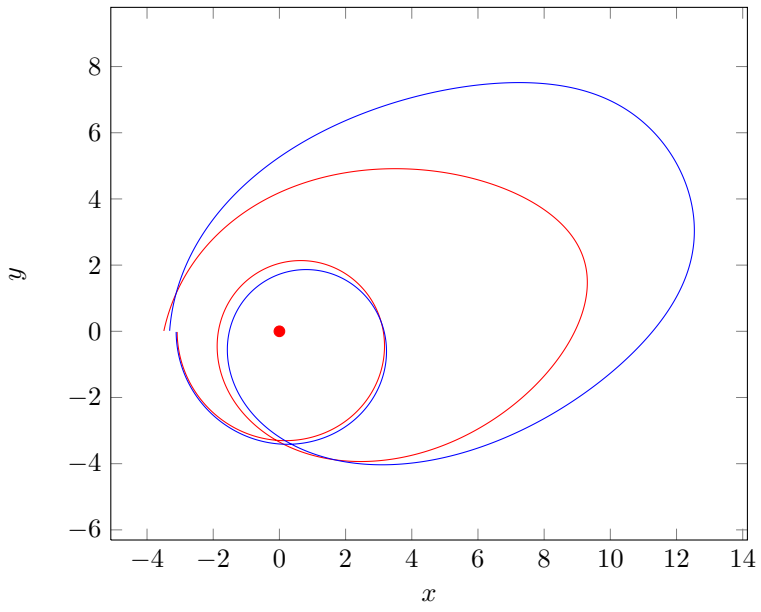


Figure 2.6: Orbits for two close initial conditions with radii, $1,03b$ and $1,04b$, where $b = 3$.

3. Helicaly symmetric solution

In this chapter we summarize the results obtained in [2] important for our problem. In the first section we review basic relations of linearized Einstein's theory. While this is a very standard topic, our treatment contains an important discussion on the consistency of linearized theory. Then we present the solution to linearized Einstein's equations representing the particle orbiting along the circular trajectory and discuss the conditions of equilibrium.

3.1 Linearized gravity

Einstein's general relativity is a theory of gravitation, space and time. Its most striking feature is the idea that the matter and its distribution determine the geometry of the spacetime and, conversely, the geometry of spacetime determines the motion of the matter it contains. This mutual influence is encapsulated in Einstein's equations. According to conventions used in this text, Einstein's equations read

$$R_{\mu\nu} - \frac{1}{2} R\, g_{\mu\nu} = -8\pi T_{\mu\nu}. \tag{3.1}$$

The Ricci tensor $R_{\mu\nu}$ and scalar curvature $R = R^{\mu}_{\mu}$ depend linearly on second derivatives of metric tensor $g_{\mu\nu}$ but non-linearly on its first derivatives. These non-linearities are responsible for mutual interaction between the geometry and the matter. Moreover, contracted Bianchi identities imply vanishing of the covariant divergence of the energy-momentum tensor,

$$\nabla^{\mu} T_{\mu\nu} = 0. \tag{3.2}$$

In many cases, this condition is strong enough to determine the equations of motion of the sources represented by $T_{\mu\nu}$.

If $T_{\mu\nu} = 0$ everywhere, i.e. in the absence of sources, metric tensor reduces to that of the Minkowski spacetime denoted by $\eta_{\mu\nu}$. We use the signature

$$\eta_{\mu\nu} = \mathrm{diag}\,(1, -1, -1, -1)\,. \tag{3.3}$$

Thus, if the sources are weak, metric tensor can be written in the form

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu} \tag{3.4}$$

where $h_{\mu\nu}$ is the perturbation much smaller than the background flat metric:

$$h_{\mu\nu} \ll \eta_{\mu\nu}.$$

In this regime, full non-linear Einstein's equations (3.1) can be replaced by their linearized version where all products of $h_{\mu\nu}$ with itself are neglected.

Several simplifications apply to linearized theory. First, all indices are raised and lowered by flat metric $\eta^{\mu\nu}$ and $\eta_{\mu\nu}$. The only exception is inverse metric $g^{\mu\nu}$ given by simple relation

$$g^{\mu\nu} = \eta^{\mu\nu} - h^{\mu\nu}, \tag{3.5}$$

where $h^{\mu\nu} = \eta^{\mu\alpha}\eta^{\nu\alpha}h_{\alpha\beta}$. Indeed, the matrix product of $g_{\mu\nu}$ and $g^{\mu\nu}$ yields

$$g^{\mu\alpha}\, g_{\alpha\nu} = \delta^{\mu}_{\nu} + h^{\mu}_{\ \nu} - h^{\mu}_{\ \nu} - h^{\mu\alpha}\, h_{\alpha\nu} = \delta^{\mu}_{\nu} + \mathcal{O}\left(h^2\right).$$

Therefore, $g^{\mu\nu}$ defined by (3.5) is the inverse of $g_{\mu\nu}$ in the linear approximation.

In order to derive linearized form of Einstein's equations one needs to find the Christoffel symbols of the affine connection. General expression for the Christoffel symbols acquire familiar form

$$\Gamma^{\rho}_{\mu\nu} = \frac{1}{2}\, g^{\rho\sigma}\left(\partial_{\mu} g_{\sigma\nu} + \partial_{\nu} g_{\mu\sigma} - \partial_{\sigma} g_{\mu\nu}\right). \tag{3.6}$$

Now, in the Minkowski coordinates $x^\mu = (t, x, y, z)$, the flat metric has constant components (3.3) and the expression for linearized Christoffel symbols derived from metric (2) reduces to

$$\Gamma^\rho_{\mu\nu} = \frac{1}{2}\,\eta^{\rho\sigma}\,(\partial_\mu h_{\sigma\nu} \;-\; \partial_\nu h_{\mu\sigma} \;-\; \partial_\sigma h_{\mu\nu})\,. \tag{3.7}$$

Notice, however, that this is not the case in general curvilinear coordinate system, for the components of the flat metric can be coordinate dependent. We emphasize this point because in this section we present linearized Einstein's equations in the Minkowski coordinates but the solution of these equations will be transformed into co-rotating curvilinear coordinates. What we are going to solve in this thesis is the geodesic equation in co-rotating coordinates in which the flat metric is in fact coordinate dependent. Hence, the connection will contain also the derivatives of $\eta_{\mu\nu}$ not present in (3.7). In such case one has to return to original expression (3.6), substitute linearized metric (3.4) and only then neglect terms quadratic in $h_{\mu\nu}$.

Derivation of linearized Einstein's equations can be found in any standard textbook on General Relativity, e.g. in Wald [9]. Following the standard procedure we introduce quantity $\bar{h}_{\mu\nu}$ by

$$\bar{h}_{\mu\nu} = h_{\mu\nu} \;-\; \frac{1}{2}\,\eta_{\mu\nu}\,h \tag{3.8}$$

where $h = \eta^{\mu\nu}h_{\mu\nu}$ is the trace of perturbation $h_{\mu\nu}$. Since the relation $\bar{h} = -h$ holds, we will refer to $\bar{h}_{\mu\nu}$ as the trace-reversed metric tensor for brevity. Linearized Einstein's equations then acquire the form

$$-\partial^\alpha \partial_{(\beta}\bar{h}_{\nu)\alpha} \;+\; \frac{1}{2}\,\eta_{\beta\nu}\,\partial^\rho\partial^\sigma\bar{h}_{\rho\sigma} \;+\; \frac{1}{2}\,\Box\bar{h}_{\beta\nu} = -\,8\,\pi\,T_{\beta\nu}. \tag{3.9}$$

Frightening expression (3.9) can be simplified by reducing the gauge freedom of general relativity. The point is that general relativity must be diffeomorphism invariant. In other words, arbitrary coordinate transformation does not affect the physical content of the theory. Any (passive) coordinate transformation can be regarded as an active transformation of the manifold into itself. Such a transformation is called diffeomorphism. Under diffeomorphism, the metric tensor is transformed as well. However, the transformed metric tensor represents the same physical situation and therefore the metric tensor is not unique. It is a close analogy to the situation in electrodynamics where the four-potential A_μ is defined only up to transformation

$$A_\mu \;\mapsto\; A_\mu \;+\; \partial_\mu\chi$$

where χ is arbitrary scalar function. It can be shown that there always exists χ such that the transformed potential will satisfy the so called *Lorenz gauge condition*

$$\partial_\mu A^\mu \;=\; 0.$$

Under this additional condition, the equations for A_μ acquire the form of wave equation

$$\Box A_\mu \;=\; j_\mu,$$

where j_μ is the four-current. This equation can be solved explicitly using the method of Green's functions:

$$A_\mu(x) \;=\; \int G(x-x')\,j_\mu(x')\,\mathrm{d}^4x,$$

where $G(x-x')$ is the Green function of the d'Alembert operator,

$$\Box G(x-x') \;=\; \delta(x-x').$$

Natural question arises whether solution A_μ obtained by this method satisfies the Lorenz gauge condition. The answer is straightforward:

$$\begin{aligned}\partial^\mu A_\mu(x) = \int \frac{\partial G(x-x')}{\partial x^\mu} j^\mu(x')\,\mathrm{d}^4x \;&=\; -\int \frac{\partial G(x-x')}{\partial x'^\mu} j^\mu(x')\,\mathrm{d}^4x \\ &=\; -\oint G(x-x')\,j_\mu(x')\,\mathrm{d}\sigma^\mu \;+\; \int G(x-x')\,\frac{\partial j^\mu(x')}{\partial x'^\mu}\,\mathrm{d}^4x,\end{aligned}$$

where we integrated by parts in the last step. The surface integral vanishes provided that j_μ is zero near infinity. Remaining term vanishes if the four-current satisfies the continuity equation

$$\partial_\mu j^\mu \;=\; 0$$

which must be always satisfied because of charge conservation. Thus, if the four-current satisfies the continuity equation, the solution obtained by Green's function method satisfies the Lorenz gauge condition.

Similarly, the metric tensor is unique up to the addition of the Lie derivative of arbitrary vector field ξ_μ,

$$g_{\mu\nu} \;\mapsto\; g_{\mu\nu} \,+\, \partial_\mu \xi_\nu \,+\, \partial_\nu \xi_\mu .$$

By an appropriate choice of ξ_μ one can always satisfy condition analogous to the Lorenz condition,

$$\partial^\mu \bar{h}_{\mu\nu} = 0. \tag{3.10}$$

Linearized Einstein's equations then simplify to

$$\Box \bar{h}_{\mu\nu} = -\,16\,\pi\,T_{\mu\nu}. \tag{3.11}$$

This is an inhomogeneous wave equation again and can be solved using the Green functions.

The question whether $\bar{h}_{\mu\nu}$ satisfies the Lorenz gauge is a more subtle one, however. In analogy with electromagnetic case, one can argue that the Lorenz gauge condition will be satisfied if the energy-momentum tensor satisfies

$$\partial^\mu T_{\mu\nu} = \; 0. \tag{3.12}$$

Obviously, this is linearized version of equation (3.2) expressing the conservation of energy-momentum, a direct consequence of Einstein's equations. The situation is very similar to electromagnetic case.

The problem arises in a following way. Suppose that energy-momentum tensor is given by

$$T_{\mu\nu} \;=\; \rho\, u_\mu\, u_\nu$$

which corresponds to a dust (perfect fluid with zero pressure). Taking the divergence we find

$$\partial^\mu T_{\mu\nu} \;=\; u_\nu\, \partial_\mu (\rho\, u^\mu) \;+\; \rho\, u^\mu\, \partial_\mu\, u_\nu .$$

The first term vanishes by the conservation of mass, so that equation (3.12) can be satisfied only if

$$u^\mu \partial_\mu u_\nu = \; 0. \tag{3.13}$$

This is the geodesic equation in the flat spacetime and we know that geodesics in the flat spacetimes are straight lines. It means that in linearized theory the particles can move along the straight lines only! This is inconsistent with our basic assumption that the sources are moving along circular trajectories. Thus, one cannot construct consistent linearized theory with sources in non-trivial motion. We will return to this point later.

Nevertheless, we continue in a standard way, bearing the problem of consistency in mind. Imposing the Lorenz gauge, linearized Einstein's equations acquire the form (3.11). In the following section we present their solution for the source with helical symmetry.

Trace-reversed metric tensor $\bar{h}_{\mu\nu}$ can be used to obtain the perturbation $h_{\mu\nu}$ by relation inverse to (3.8),

$$h_{\mu\nu} \;=\; \bar{h}_{\mu\nu} \,-\, \frac{1}{2}\,\bar{h}\,\eta_{\mu\nu}.$$

From this metric tensor, the Christoffel symbols can be computed using relations (3.6) or (3.7). Christoffel symbols is an analogy of the field strength in the Newtonian gravity. Free test particle in the gravitational field is moving along geodesics, i.e. curves parametrized by the proper time $x^\mu = x^\mu(s)$ which are solutions of geodesic equation

$$\frac{\mathrm{d}^2 x^\mu}{\mathrm{d}s^2} \;+\; \Gamma^\mu_{\alpha\beta}\, \frac{\mathrm{d}x^\alpha}{\mathrm{d}s}\, \frac{\mathrm{d}x^\beta}{\mathrm{d}s} = 0. \tag{3.14}$$

In this thesis we take the solution $\bar{h}_{\mu\nu}$ and Christoffel symbols derived in [2] and numerically solve the geodesic equation.

3.2 Solution of the wave equation

Einstein's equations (3.11) can be solved by the method of Green's functions. By definition, the Green function of the d'Alembert operator is a function $G(x-x')$ of two spacetime events x and x' such that

$$\Box G(x-x') = \delta(x-x'). \tag{3.15}$$

Explicit form of $G(x-x')$ can be obtained, for example, by the Fourier transform. It turns out that equation (3.15) has two physically interesting solutions describing two different situations. First type of the Green function is called *advanced* and it is denoted by subscript "+", while the second type is called *retarded* and denoted by subscript "–". Both Green functions (see, e.g. [6]) can be written by single relation

$$G_{\pm}(x-x') = \frac{1}{4\pi}\frac{1}{|\boldsymbol{x}-\boldsymbol{x}'|}\,\Theta\left[\mp(t-t')\right]\,\delta\left(t-t'\pm|\boldsymbol{x}-\boldsymbol{x}'|\right). \tag{3.16}$$

Physical difference between both functions is immediate from the presence of the Θ-function. Retarded function vanishes for $t > t'$ which means that only times $t' > t$ will contribute to the solution. This is due to fact that the source at time t can influence particles only at later times. Nevertheless, the wave equation also admits advanced solutions when the source influences particles in the past only.

Now, consider arbitrary event x in which the source of the field is present. Its line cone is defined by equation

$$(t-t')^2 \;-\; (\boldsymbol{x}-\boldsymbol{x}')^2 \;=\; 0,$$

where $x' = (t', \boldsymbol{x}')$ is any event lying on the light cone. Obviously, taking the square root of the last equation yields

$$t - t' \;=\; \pm|\boldsymbol{x}-\boldsymbol{x}'|.$$

Thus, the presence of $\delta-$function ensures that only events lying on the light cone of the event x contribute to the solution of wave equation and Θ-function selects the future null cone (retarded solution) and the past null cone (advanced solution), respectively.

Imagine two wordlines describing the motion of two particles, see figure 3.1. The first particle is a source of the field which acts on the test particle. Let $x = (t, \boldsymbol{x})$ be any event on the worldline of the test particle. The test particle at time t is not affected by the source at the same time, for the signal needs some time to propagate from the source. Thus, the test particle is influenced by the motion of the source at earlier, retarded time

$$t' \;=\; t \;-\; |\boldsymbol{x}-\boldsymbol{x}'|.$$

Coordinates of the source at retarded time are $x'_{-} = (t', \boldsymbol{x}')$ and this event lies on the intersection of the past null cone of the test particle and the worldline of the source. Similar considerations apply to advanced solution.

Knowing the Green functions, the solution of (3.11) can be constructed by

$$\bar{h}^{\pm}_{\mu\nu}(x) = -\,16\,\pi \int G_{\pm}(x-x')\,T_{\mu\nu}(x')\,\mathrm{d}^4x', \tag{3.17}$$

because the application of $\Box$ to $h^{\pm}_{\mu\nu}$ yields

$$\begin{aligned}
\Box\bar{h}^{\pm}_{\mu\nu}(x) &= -\,16\,\pi \int \Box G_{\pm}(x-x')\,T_{\mu\nu}(x')\,\mathrm{d}^4x' \\
&= -\,16\,\pi \int \delta(x-x')\,T_{\mu\nu}(x')\,\mathrm{d}^4x' \\
&= -\,16\,\pi\,T_{\mu\nu}(x).
\end{aligned}$$

We will refer to metric tensor (3.17) as to retarded (–) and advanced (+) solution of Einstein's equations, respectively.

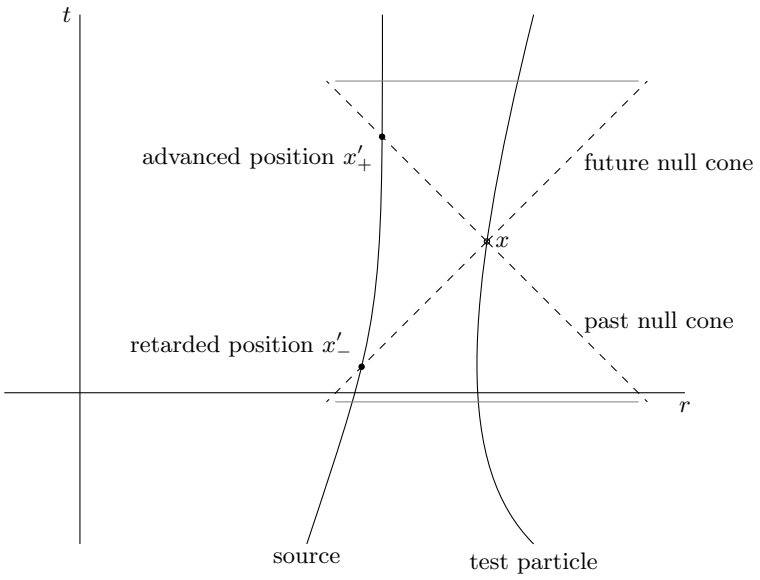


Figure 3.1: Wordlines of two particles.

Let us stress again that one can compute the coefficients of affine connection from (3.17) which represent the "strength" of gravitational field. We assume that any test particle moves along the geodesics given by (3.14). This assumption, however, is beyond the linear approximation we consider for the reasons explained in section 3.1, see equation (3.13).

Solution (3.17) can be simplified further taking (3.16) into account. Integral on the right hand side (3.17) is integral over four-dimensional spacetime. Since the time t' enters the Green function only through $\delta-$function, we can integrate over coordinate t' and obtain

$$\bar{h}^{\pm}_{\mu\nu}(x) = -4 \int \frac{T_{\mu\nu}(t \pm |\boldsymbol{x} - \boldsymbol{x}'|, \boldsymbol{x}')}{|\boldsymbol{x} - \boldsymbol{x}'|} \, \mathrm{d}\boldsymbol{x}' \tag{3.18}$$

where the integral is now taken over three-dimensional light cone

$$t' \;=\; t \pm |\boldsymbol{x} - \boldsymbol{x}'|.$$

3.3 Helically symmetric solution

In this section we present the solution of Einstein's equations given by (3.17) for particular choice of $T_{\mu\nu}$ describing the source particle in circular motion at constant angular velocity ω. Energy-momentum tensor of the point particle can be written in the form

$$T_{\mu\nu}(x) = \rho \, u_\mu \, u_\nu \tag{3.19}$$

where ρ is invariant rest-energy density and $u^\mu = (u^0, \boldsymbol{u})$ is the four-velocity of the particle. Let a^μ be the position four-vector of the source particle which, parametrized by inertial time, reads

$$a^\mu(t) = (t, a\,\cos(\omega t + \phi_0), a \sin(\omega t + \phi_0), 0) \;\equiv\; (t, \boldsymbol{a}(t)), \tag{3.20}$$

where $\boldsymbol{a}(t)$ is the position three-vector of the particle. If m is the mass of the particle, corresponding energy-density can be written in manifestly invariant form as

$$\rho(x) \;=\; m \int_{-\infty}^{\infty} \delta(x \,-\, a(\bar{s}))\, \mathrm{d}\bar{s}$$

where $\mathrm{d}\bar{s}$ is the proper time of the particle and

$$\delta(x - a(\bar{s})) \;=\; \delta(t - a^0(\bar{s}))\,\delta(\boldsymbol{x} \,-\, \boldsymbol{a}(\bar{s})).$$

Now we introduce standard time coordinate by

$$\mathrm{d}\bar{t} \,=\, \gamma\,\mathrm{d}s \,\equiv\, \frac{\mathrm{d}s}{\sqrt{1-v^2}}$$

(v is the magnitude of three-velocity of the particle) so that the expression for ρ integrates to

$$\rho(x) = \frac{m}{\gamma}\,\delta(\boldsymbol{x} \,-\, \boldsymbol{a}(t)). \tag{3.21}$$

Notice that $\gamma-$factor is constant for the particle in uniform circular motion. The four-velocity of the source particle with the position four-vector is defined by

$$u^\mu \;=\; \frac{\mathrm{d}a^\mu}{\mathrm{d}s} \;=\; \gamma\,\frac{\mathrm{d}a^\mu}{\mathrm{d}t}.$$

For brevity we denote differentiation with respect to the argument by dot. Energy-momentum tensor (3.19) can be then written in the form

$$T_{\mu\nu}(x) = m\,\gamma\,\delta(\boldsymbol{x} \,-\, \boldsymbol{a}(t))\,\dot{a}_\mu(t)\,\dot{a}_\nu(t). \tag{3.22}$$

This expression can be inserted into the general retarded/advanced solution given by (3.18):

$$\bar{h}^{\pm}_{\mu\nu}(x) \;=\; -\,4\,m\,\gamma\int \frac{\dot{a}_\mu(t \pm |\boldsymbol{x}-\boldsymbol{x}'|)\,\dot{a}_\nu(t \pm |\boldsymbol{x}-\boldsymbol{x}'|)}{|\boldsymbol{x}-\boldsymbol{x}'|}\delta(\boldsymbol{x}' - \boldsymbol{a}(t \pm |\boldsymbol{x}-\boldsymbol{x}'|)\,\mathrm{d}\boldsymbol{x}'.$$

Now we can perform the integral over $\delta-$function, taking nonlinear dependence of its argument on integration variable into account. After some rearrangements we arrive at solution

$$\bar{h}^{\pm}_{\mu\nu}(x) = -\,4\,m\,\gamma\,\frac{\dot{a}_\mu(t \pm |\boldsymbol{x}-\boldsymbol{x}'|)\,\dot{a}_\nu(t \pm |\boldsymbol{x}-\boldsymbol{x}'|)}{|\boldsymbol{x}-\boldsymbol{x}'| \pm (\boldsymbol{x}-\boldsymbol{x}')\cdot\dot{\boldsymbol{a}}(t \pm |\boldsymbol{x}-\boldsymbol{x}'|)} \tag{3.23}$$

where $\boldsymbol{x}'$ is now solution to equation

$$\boldsymbol{x}' \;=\; \boldsymbol{a}(t \pm |\boldsymbol{x}-\boldsymbol{x}'|).$$

If we define, for convenience, quantities

$$\begin{aligned}
&\boldsymbol{R}_\pm = \boldsymbol{x} \,-\, \boldsymbol{x}', \quad R_\pm \,=\, |\boldsymbol{R}_\pm| \quad \text{(advanced/retarded distance)}\\
&t_\pm = t \,\pm\, R_\pm \quad \text{(advanced/retarded time)}\\
&\rho_\pm = R_\pm \,\pm\, \boldsymbol{R}_\pm\cdot\dot{\boldsymbol{a}}(t_\pm),
\end{aligned} \tag{3.24}$$

the solution becomes

$$\bar{h}^{\pm}_{\mu\nu}(x) = -\,\frac{4\,m\,\gamma}{\rho_\pm}\,\dot{a}_\mu(t_\pm)\,\dot{a}_\nu(t_\pm) \tag{3.25}$$

where $\boldsymbol{R}_\pm$ is determined by implicit equation

$$\boldsymbol{R}_\pm \;=\; \boldsymbol{x} \,-\, \boldsymbol{a}(t_\pm).$$

Now we transform the solution into co-rotating frame. Since $\bar{h}^{\pm}_{\mu\nu}$ is covariant, second-rank tensor, under coordinate transformation $x^\mu \mapsto x'^\mu$ its component transform according to relation

$$\bar{h}^{\pm}_{\mu\nu} \;\mapsto\; J^\alpha_{\ \mu}\,J^\beta_{\ \nu}\,\bar{h}^{\pm}_{\mu\nu}$$

where $J^\alpha_{\ \mu}$ is the Jacobi matrix of coordinate transformation defined by

$$J^\alpha_{\ \mu} \;=\; \frac{\partial(x^0,x^1,x^2,x^2)}{\partial(x'^0,x'^1,x'^2,x'^3)}.$$

We subsequently apply several coordinate transformations. In the first step we introduce standard cylindrical coordinates by relations

$$t = t, \;\; x = r\,\cos\phi, \;\; y = r\,\sin\phi, \;\; z = z.$$

Jacobi matrix in this case reads

$$J^{\alpha}_{1\,\mu} \;=\; \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -r\,\sin\phi & 0 \\ 0 & \sin\phi & r\,\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Next we perform transformation to co-rotating frame introducing coordinate

$$\hat{\phi} \;=\; \phi \,-\, \omega t.$$

Corresponding Jacobi matrix has simple form

$$J^{\alpha}_{2\,\mu} \;=\; \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \omega & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Let us see how the quantities $\dot{a}_\mu(t_\pm)$ change under these transformations. By (3.20), we have in the Cartesian frame[1]

$$\dot{a}_\mu(t_\pm) \;=\; (1, \omega\, a\,\sin(\omega t_\pm + \phi_0), -\omega\, a\cos(\omega t_\pm + \phi_0), 0)\,.$$

Multiplication by $J^{\alpha}_{1\mu}$ gives components of $\dot{a}_\mu$ in cylindrical coordinates:

$$\dot{a}_\mu(t_\pm) \;=\; (1, \omega a \sin(\omega t_\pm + \phi_0 - \phi), -\omega a r \cos(\omega t_\pm + \phi_0 - \phi), 0)\,.$$

Thus, instead of advanced/retarded time it is convenient to introduce parameter

$$\theta_\pm \;=\; \omega t_\pm \,+\, \phi_0 \,-\, \phi \;=\; \omega\, t \;\pm\; \omega\, R_\pm \,+\, \phi_0 \,-\, \phi \;=\; \pm\omega\, R_\pm \,+\, \phi_0 \,-\, \hat{\phi}.$$

Next we multiply $\dot{a}_\mu$ by $J^{\alpha}_{2\mu}$ and find

$$\dot{a}_\mu(t_\pm) \;=\; \big(1 - \omega^2 a r\cos\theta_\pm,\; \omega a \sin\theta_\pm, \;\; -\omega a r\cos\theta_\pm,\; 0\big)\,.$$

Finally, we substitute this result into the solution (3.25) and express quantities $\rho_\pm, \theta_\pm$ and $\boldsymbol{R}_\pm$ in terms of co-rotating coordinates $t, r, \hat{\phi}$ and z and reintroduce standard SI units. Notice that all quantities are time-independent by the choice of co-rotating frame.

[1] In our signature, spatial components change the sign when raising or lowering indices

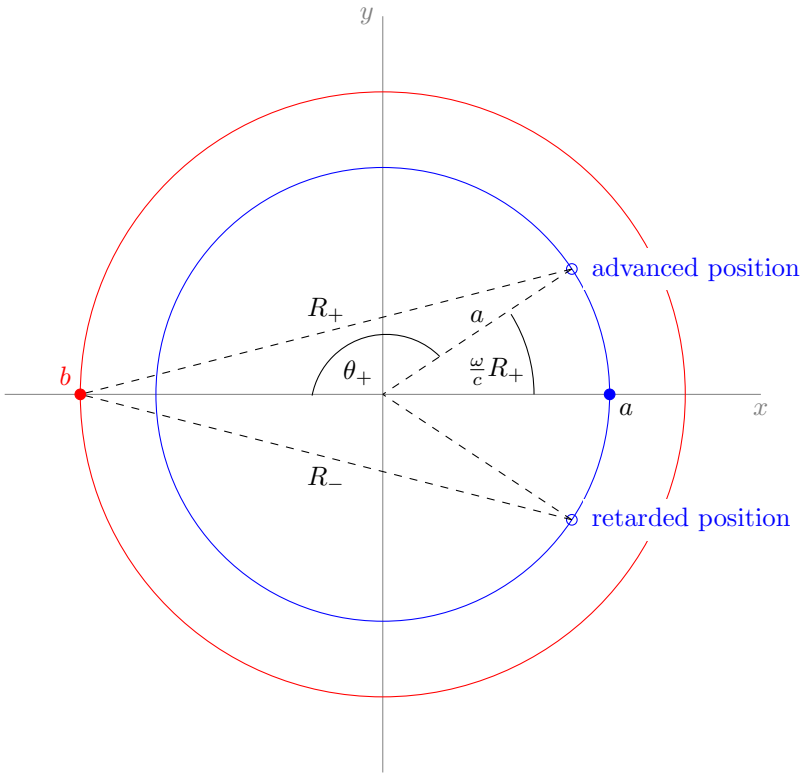


Figure 3.2: System of two particles at time $t = 0$. Red particle does not feel the force from black particle at the same time, but at retarded time or at advanced time.

Helically symmetric solution in co-rotating frame I

$$
\begin{aligned}
\bar{h}^{\pm}_{00} &= -\frac{4\,m\,G\,\gamma}{c^2\,\rho_\pm}\left(1 - \frac{\omega^2\,a\,r}{c^2}\cos\theta_\pm\right)^2 \\
\bar{h}^{\pm}_{01} &= -\frac{4\,m\,G\,\gamma}{c^3\,\rho_\pm}\left(1 - \frac{\omega^2\,a\,r}{c^2}\cos\theta_\pm\right)\omega\,a\,\sin\theta_\pm \\
\bar{h}^{\pm}_{02} &= \frac{4\,m\,G\,\gamma}{c^3\,\rho_\pm}\left(1 - \frac{\omega^2\,a\,r}{c^2}\cos\theta_\pm\right)\omega\,a\,r\,\cos\theta_\pm \\
\bar{h}^{\pm}_{11} &= -\frac{4\,m\,G\,\gamma}{c^4\,\rho_\pm}\,\omega^2\,a^2\,\sin^2\theta_\pm \\
\bar{h}^{\pm}_{12} &= \frac{4\,m\,G\,\gamma}{c^4\,\rho_\pm}\,\omega^2\,a^2\,r\,\sin\theta_\pm\,\cos\theta_\pm \\
\bar{h}^{\pm}_{22} &= -\frac{4\,m\,G\,\gamma}{c^4\,\rho_\pm}\,\omega^2\,a^2\,r^2\,\cos^2\theta_\pm
\end{aligned}
$$

where (3.26)

$$
\begin{aligned}
\theta_\pm &= \pm\frac{\omega}{c}\,R_\pm + \phi_0 - \hat{\phi} \\
R_\pm &= \sqrt{a^2 + r^2 + z^2 - 2\,a\,r\,\cos\theta_\pm} \\
\rho_\pm &= R_\pm \mp \frac{\omega\,a\,r}{c}\,\sin\theta_\pm
\end{aligned}
$$

It is useful to get some idea about geometrical meaning of quantities $\theta_\pm$ and $R_\pm$ which appeared in the solution. Since we are primarily interested in binary system, let us evaluate

these functions at the position of B-particle. Coordinates of B-particle in co-rotating frame are given by

$$r \,=\, a,\; \hat{\phi} \,=\, \phi_0 \,+\, \pi,\; z \,=\, 0.$$

With these values, functions $\theta_\pm$ and $R_\pm$ reduce to

$$\begin{aligned} \theta_\pm &= \pm\,\frac{\omega}{c}\,R_\pm \,-\, \pi, \\ R_\pm &= \sqrt{a^2 \,+\, b^2 \,-\, 2\,a\,b\,\cos\theta_\pm}. \end{aligned} \tag{3.27}$$

Configuration of binary system is plotted in figure 3.2. Since the gravitational field is propagating at finite speed c, B-particle is not affected by the motion of B-particle immediately, but at later or at earlier time, depending on the character of the solution taken into account. For advanced solution, B-particle is influenced by the field of A-particle produced at later time t_+ when the A-particle is located at *advanced position*. Quantity R_+ then represents *advanced distance* between both particles at time t_+. Gravitational field needs time

$$\Delta t \;=\; \frac{R_+}{c}$$

to propagate from the advanced position at time t_+ to the position of B-particle at time t. During this time, the radius-vector of A-particle sweeps out an angle

$$\vartheta_+ \,=\, \omega\,\Delta t \;=\; \frac{\omega}{c}\,R_+$$

called *deficit angle*. Definition (3.27) of θ_+ then shows that θ_+ is related to deficit angle by

$$\theta_+ \;=\; \vartheta_+ \,-\, \pi.$$

Another way of seeing this follows from the law of cosines. According to figure 3.2, B-particle, advanced position of A-particle and the origin of coordinate system form a triangle with sides a, b and R_+. They are related by the cosine law

$$a^2 \,+\, b^2 \,-\, 2\,a\,b\,\cos\Theta \;=\; R_+$$

where Θ is an angle between sides of lengths a and b. Comparing this relation to definition (3.27) of R_+ shows that angle Θ is in fact equal to θ_+. Similar considerations apply to retarded position. By the symmetry of configuration, it is evident that retarded and advanced distances are the same. This is not true, of course, when these functions are evaluated at points different from the position of B-particle.

An important test of validity of any relativistic solution is whether this solution has correct Newtonian limit. In order to answer this question we have to expand the solution into the series in variable ω determining velocities of A- and B- particle. However, when we say that the Newtonian limit corresponds to small velocities, it does not mean that the value of velocity is small, but rather that it is small compared to the speed of light c. Hence, we must expand the solution in dimensionless parameter

$$\alpha \;=\; \frac{\omega\,a}{c}$$

where ωa is the speed of A-particle. Radius b must be then expressed in multiples of a. Moreover, it is convenient to rescale all coordinates,

$$\tilde{t} \;=\; \frac{c\,t}{a}, \quad \tilde{r} \;=\; \frac{r}{a}, \quad \hat{\phi} \;=\; \hat{\phi}, \quad \tilde{z} \;=\; \frac{z}{a}. \tag{3.28}$$

and introduce dimensionless mass parameter

$$\tilde{m} = \frac{m\,G}{a\,c^2}. \tag{3.29}$$

The Jacobi matrix of this coordinate transformation is diagonal matrix

$$J^{\mu}_{3\alpha} \;=\; \mathrm{diag}(a, a, 1, a).$$

Similarly we define dimensionless analogons of $R_\pm$ and $\rho_\pm$ by

$$\begin{aligned}\tilde{R}_\pm &= \frac{1}{a}\,R_\pm \;=\; \sqrt{1 \,+\, \tilde{r}^2 \,+\, \tilde{z}^2 \,-\, 2\,\tilde{r}\,\cos\theta_\pm},\\ \tilde{\rho}_\pm &= \frac{1}{a}\,\rho_\pm \;=\; \tilde{R}_\pm \,\mp\, \alpha\,\tilde{r}\,\sin\theta_\pm.\end{aligned} \tag{3.30}$$

Recall that actual metric tensor is a sum of the flat metric and perturbation subject to linearized Einstein's equations. Flat metric and its inverse in co-rotating coordinates read

$$\eta_{\mu\nu} \;=\; \begin{pmatrix} 1-\frac{\omega^2 r^2}{c^2} & 0 & -\frac{\omega r^2}{c} & 0\\ 0 & -1 & 0 & 0\\ -\frac{\omega r^2}{c} & 0 & -r^2 & 0\\ 0 & 0 & 0 & -1\end{pmatrix},\;\; \eta^{\mu\nu} \;=\; \begin{pmatrix} 1 & 0 & -\frac{\omega}{c} & 0\\ 0 & -1 & 0 & 0\\ -\frac{\omega}{c} & 0 & -\frac{1}{r^2}+\frac{\omega^2}{c^2} & 0\\ 0 & 0 & 0 & -1\end{pmatrix}. \tag{3.31}$$

Transforming this metric to dimensionless coordinates we find

$$\eta_{\mu\nu} \;=\; \begin{pmatrix} a^2\left(1-\tilde{r}^2\alpha^2\right) & 0 & -a^2\tilde{r}^2\alpha & 0\\ 0 & -a^2 & 0 & 0\\ -a^2\tilde{r}^2\alpha & 0 & -a^2\tilde{r}^2 & 0\\ 0 & 0 & 0 & -a^2\end{pmatrix}.$$

We can see that we can completely eliminate constant a by trivial conformal rescaling

$$\eta_{\mu\nu} \;\mapsto\; a^{-2}\eta_{\mu\nu}.$$

Then we must rescale also the perturbation $\bar{h}^\pm_{\mu\nu}$:

$$\bar{h}^\pm_{\mu\nu} \;\mapsto\; a^{-2}\bar{h}^\pm_{\mu\nu}.$$

The last rescaling, fortunately, eliminates constant a even from the solution (3.26). Finally we can write down helically symmetric solution in appropriate coordinates and appropriate units. From now we work in these coordinates exclusively and hence omit the tildas decorating dimensionless coordinates.

HELICALLY SYMMETRIC SOLUTION IN CO-ROTATING FRAME II

(DIMENSIONLESS UNITS)

$$\begin{aligned}\bar{h}^\pm_{00} &= -\frac{4\,m\,\gamma}{\rho_\pm}\,\left(1 \,-\, \alpha^2\,r\,\cos\theta_\pm\right)^2\\ \bar{h}^\pm_{01} &= -\frac{4\,m\,\gamma}{\rho_\pm}\,\left(1 \,-\, \alpha^2\,r\,\cos\theta_\pm\right)\,\alpha\,\sin\theta_\pm\\ \bar{h}^\pm_{02} &= \;\;\frac{4\,m\,\gamma}{\rho_\pm}\,\left(1 \,-\, \alpha^2\,r\,\cos\theta_\pm\right)\,\alpha\,r\,\cos\theta_\pm\\ \bar{h}^\pm_{11} &= -\frac{4\,m\,\gamma}{\rho_\pm}\,\alpha^2\,\sin^2\theta_\pm\\ \bar{h}^\pm_{12} &= \;\;\frac{4\,m\,\gamma}{\rho_\pm}\,\alpha^2\,r\,\sin\theta_\pm\,\cos\theta_\pm\\ \bar{h}^\pm_{22} &= -\frac{4\,m\,\gamma}{\rho_\pm}\,\alpha^2\,r^2\,\cos^2\theta_\pm\end{aligned}$$

where (3.32)

$$\begin{aligned}\theta_\pm &= \pm\alpha\,R_\pm \,+\, \phi_0 \,-\, \hat{\phi}\\ R_\pm &= \sqrt{1 \,+\, r^2 \,+\, z^2 \,-\, 2\,r\,\cos\theta_\pm}\\ \rho_\pm &= R_\pm \,\mp\, \alpha\,r\,\sin\theta_\pm\end{aligned}$$

For completness, components of rescaled flat metric in dimensionless coordinates are

$$\eta_{\mu\nu} = \begin{pmatrix} 1-r^2\alpha^2 & 0 & -r^2\alpha & 0 \\ 0 & -1 & 0 & 0 \\ -r^2\alpha & 0 & -r^2 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad \eta^{\mu\nu} = \begin{pmatrix} 1 & 0 & -\alpha & 0 \\ 0 & -1 & 0 & 0 \\ -\alpha & 0 & \alpha^2 - \frac{1}{r^2} & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{3.33}$$

3.4 Equilibrium condition

In the previous section we have found the solution of Einstein's equations representing gravitational field produced by the particle orbiting along circular at constant angular velocity (A-particle). In the units chosen, the radius of A-particle's orbit is $a = 1$ and dimensionless parameter $\alpha = \omega a/c$ is used instead of angular velocity ω. As in the Newtonian case, we did not discuss *why* the A-particle is moving an a prescribed way. Now we insert another particle (called B-particle) into the gravitational field of A-particle and ask whether B-particle can move along circular trajectory of radius b (possibly different from radius a) at the same angular velocity α. Once we show that such motion of B-particle is feasible, we return to question whether A-particle can move in a prescribed way in the field of B-particle. Solving both equations of equilibrium one can deduce that equilibrium configuration exists and is consistent with Einstein's equations.

At this stage we therefore assume that the gravitational field of A-particle is given by (3.32). Metric tensor of the spacetime is

$$g^{\pm}_{\mu\nu} = \eta_{\mu\nu} + h^{\pm}_{\mu\nu} \tag{3.34}$$

where the flat metric components in co-rotating frame are given by (3.33) and perturbation $h^{\pm}_{\mu\nu}$ is related to trace-reversed metric tensor $\bar{h}^{\pm}_{\mu\nu}$ by

$$h^{\pm}_{\mu\nu} = \bar{h}^{\pm}_{\mu\nu} - \frac{1}{2}\,\eta_{\mu\nu}\,\bar{h}.$$

Components of $h^{\pm}_{\mu\nu}$ are not listed explictly, for their particular form is not important.

B-particle is assumed to follow geodesics of the spacetime with metric $g^{\pm}_{\mu\nu}$ although this assumption is beyond the linearized framework, see discussion in the section 3.1. In order to write down the components of geodesic equation (3.14) we have to calculate the Christoffel symbols ${}^{\pm}\Gamma^{\mu}_{\alpha\beta}$ derived from metric $g^{\pm}_{\mu\nu}$ via relation (3.6). Notice that now we are working in curvilinear coordinates and expression for the Christoffel symbols cannot be simplified to (3.7). However, relation (3.6) contains terms quadratic in $h^{\pm}_{\mu\nu}$ which must be neglected at the end of calculation.

General expressions for the Christoffel symbols are not included in the paper [2] on which this work is based but they are necessary for our numerical simulation. These expressions have been obtained using Mathematica software for the purposes of this thesis but we do not list their full form as well. Their combinations entering the geodesic equation, however, are present in the source code listing 4.3.

Since this section is concerned merely with the condition of equlibrium essential for all subsequent development, we present only relevant Christoffel symbols evaluated at the position of B-particle. If the B-particle is to orbit at angular velocity α together with the source A-particle, its initial position and initial four-velocity must be described by co-rotating coordinates

$$b^{\mu}(t) = (t, b, \phi_0 + \pi, 0), \quad \dot{b}^{\mu}(t) = (1, 0, 0, 0). \tag{3.35}$$

Notice that initial conditions can be imposed at arbitrary time t because the gravitational field is stationary in co-rotating frame. The question now is whether B-particle can stay at the initial position in co-rotating frame for all later times. That means that co-rotating coordinates of B-particle and its four-velocity must remain constant:

$$\ddot{b}^{\mu}(t) = 0 \quad \text{for all } \mu = 0, 1, 2, 3.$$

On the other hand, second derivatives of the position vector are determined by the geodesic equation

$$\ddot{b}^{\mu}(s) = -\,{}^{\pm}\Gamma^{\mu}_{\alpha\beta}\,\dot{b}^{\alpha}(s)\,\dot{b}^{\beta}(s)$$

where b^μ is parametrized by the proper time. For initial conditions (3.35), however, only components Γ^μ_{00} survive because the four-velocity has only zeroth component. The condition of equilibrium then reduces to

$$ {}^{\pm}\Gamma^{\mu}_{00}\big|_{\mathrm{B}} \;=\; 0 \quad \text{for all } \mu = 0,1,2,3. \tag{3.36} $$

Subscript B indicates that the Christoffel symbols are evaluated on the worldline of B-particle.

At the position of B-particle functions $\rho_\pm$, $\theta_\pm$ and $R_\pm$ given by (3.32) reduce to

$$ \begin{aligned} &\theta_\pm = \pm\,\alpha\, R_\pm \,-\, \pi, && \rho_\pm \;= R_\pm \,-\, \alpha\, b\, \sin\theta_\pm, \\ &R_\pm = \sqrt{1 \,+\, b^2 \,-\, 2\, b\, \cos\theta_\pm}. \end{aligned} \tag{3.37} $$

Recall that, according to figure 3.2, retarded and advanced distances $R_\pm$ are equal to each other. We can therefore denote

$$ R \;=\; R_+ \;=\; R_-. $$

It follows immediately that $\theta_\pm \;=\; \pm\,\alpha\, R \,-\, \pi$ and thus

$$ \sin\theta_- \;=\; -\,\sin\theta_+, \;\; \cos\theta_- \;=\; \cos\theta_+. $$

Hence we denote $\theta \;=\; \theta_+$ from which we have

$$ \sin\theta_\pm \;=\; \pm\,\sin\theta, \;\; \cos\theta_\pm \;=\; \cos\theta. $$

Similarly we introduce quantity

$$ \rho \;=\; \rho_+ \;=\; \rho_- \;=\; R \,\mp\, \alpha\, b\, \sin\theta. $$

With this notation, the Christoffel symbols evaluated at the position of B-particle read

$$ \begin{aligned} {}^{\pm}\Gamma^0_{00} = \pm\frac{\alpha b\gamma}{4\rho^3} m \big(&-4\alpha^2 \sin\theta + \alpha^6 b^3 \sin 4\theta + 2\alpha^4 b^3 \sin 2\theta + 6\alpha^5 b^2 \rho \cos 3\theta \\ &-2\alpha^4 b^2 \sin\theta - 6\alpha^4 b^2 \sin 3\theta - 4\alpha^2 b^2 \sin\theta - 2\alpha\rho \left(\alpha^4 b^2 - 2\alpha^2 \left(b^2+1\right) - 2\right) \cos\theta \\ &-8\alpha^4 b\rho^2 \sin 2\theta + 2\alpha^4 b \sin 2\theta - 16\alpha^3 b\rho \cos 2\theta + 10\alpha^2 b \sin 2\theta - 4\sin\theta \big), \end{aligned} \tag{3.38} $$

$$ \begin{aligned} {}^{\pm}\Gamma^1_{00} = -\alpha^2 b \,+\, \frac{\gamma m}{4\rho^3} \big(&-4\alpha^3 \rho \sin\theta - 4\alpha\rho \sin\theta - 2\alpha^6 b^4 \cos 3\theta + \alpha^6 b^3 \cos 4\theta + \alpha^6 b^3 \\ &+8\alpha^5 b^3 \rho \sin 2\theta + 10\alpha^4 b^3 + 4\alpha^2 b^3 - 2\alpha^5 b^2 \rho \sin\theta - 6\alpha^5 b^2 \rho \sin 3\theta \\ &-6\alpha^4 b^2 \cos 3\theta - 20\alpha^3 b^2 \rho \sin\theta + 2\alpha^2 b \cos 2\theta \left(\alpha^4 b^2 + \alpha^2 \left(7b^2 - 8\rho^2 + 1\right) + 5\right) \\ &-2\cos\theta \left(\alpha^6 b^4 + \alpha^4 b^2 \left(2b^2+9\right) + 2\alpha^2 \left(6b^2 - 4\rho^2 + 1\right) + 2\right) + 2\alpha^4 b \\ &+16\alpha^3 b\rho \sin 2\theta + 14\alpha^2 b + 4b \big), \end{aligned} \tag{3.39} $$

$$ \begin{aligned} {}^{\pm}\Gamma^2_{00} = \mp\,\frac{\gamma m}{4b\rho^3} \big(&-16\alpha^2 \rho^2 \sin\theta + 4\alpha^2 \sin\theta + \alpha^8 b^5 \sin 4\theta + 2\alpha^6 b^5 \sin 2\theta \\ &+6\alpha^7 b^4 \rho \cos 3\theta - 2\alpha^6 b^4 \sin\theta - 6\alpha^6 b^4 \sin 3\theta - 4\alpha^4 b^4 \sin\theta \\ &-8\alpha^6 b^3 \rho^2 \sin 2\theta + 2\alpha^6 b^3 \sin 2\theta - \alpha^6 b^3 \sin 4\theta + 8\alpha^4 b^3 \sin 2\theta \\ &-6\alpha^5 b^2 \rho \cos 3\theta - 2\alpha^4 b^2 \sin\theta + 6\alpha^4 b^2 \sin 3\theta - 16\alpha^3 b\rho \left(\alpha^2 b^2 - 1\right) \cos 2\theta \\ &-2\alpha\rho \left(2\alpha^2 + \alpha^6 b^4 - \alpha^4 b^2 \left(2b^2+3\right) + 2\right) \cos\theta + 16\alpha^4 b\rho^2 \sin 2\theta \\ &-2\alpha^4 b \sin 2\theta - 10\alpha^2 b \sin 2\theta + 4\sin\theta \big), \end{aligned} \tag{3.40} $$

$$ {}^{\pm}\Gamma^3_{00} = 0. \tag{3.41} $$

These expressions have been found using Mathematica software.

Can the condition of equlibrium, equation (3.36), be satisfied? In [2] it is shown numerically that while the symbol ${}^{\pm}\Gamma^1_{00}$ can be made to vanish but remaining symbols ${}^{\pm}\Gamma^0_{00}$ and ${}^{\pm}\Gamma^2_{00}$ can be not. This is physically reasonable result: orbiting B-particle must emit gravitational waves and loose the energy. The structure of the Christoffel symbols, however, indicate how to achieve the equilibrium. We can see that symbols ${}^{\pm}\Gamma^0_{00}$ and ${}^{\pm}\Gamma^2_{00}$ *change the sign* when switching between retarded and advanced solutions. Thus, if we choose the metric tensor to be

$$g_{\mu\nu} = \eta_{\mu\nu} \; + \; \frac{1}{2} \left(h^+_{\mu\nu} \; + \; h^-_{\mu\nu} \right) \tag{3.42}$$

rather than (3.34), these Christoffel symbols will vanish,

$$\begin{aligned} \Gamma^0_{00} = \; & \frac{1}{2} \left({}^+\Gamma^0_{00} \; + \; {}^-\Gamma^0_{00} \right) \; = \; 0, \\ \Gamma^2_{00} = \; & \frac{1}{2} \left({}^+\Gamma^2_{00} \; + \; {}^-\Gamma^2_{00} \right) \; = \; 0, \end{aligned}$$

while the other symbols remain unchanged

$$\begin{aligned} \Gamma^1_{00} = \; & \frac{1}{2} \left({}^+\Gamma^1_{00} \; + \; {}^-\Gamma^1_{00} \right) \; = \; {}^{\pm}\Gamma^1_{00}, \\ \Gamma^3_{00} = \; & \frac{1}{2} \left({}^+\Gamma^3_{00} \; + \; {}^-\Gamma^3_{00} \right) \; = \; 0. \end{aligned}$$

Then the only condition of equilibrium which must be satisfied is

$$\Gamma^1_{00}\big|_{\mathrm{B}} \; = \; 0.$$

In paper [2], this condition has been solved numerically using Mathematica. In this thesis we repeated corresponding calculations in C++ in more general context. Discussion of the solution is relegated to the next chapter.

4. Implementation

In this chapter we finally implement numerical solutions of the geodesic equation and the equation of equilibrium. We have seen in the previous chapter, equations (3.32), that the solution of Einstein's equations contains functions $\theta_\pm$ and $\rho_\pm$ which are given implicitly by the set of transcendent equations. Our first task is therefore to develop an algorithm for determining values of these functions at arbitrary spacetime point. Next we implement a routine for solving the equation of equlibrium and finally present the code for solving the geodesic equations. Routines developed in this chapter will be used in the next one to obtain new physical results.

4.1 Functions $\theta_\pm$, $R_\pm$ and $\rho_\pm$

Our first task is to solve transcendent equation

$$\theta_\pm = \pm\,\alpha\,\sqrt{1+r^2+z^2-2r\cos\theta_\pm}\;-\;\hat{\phi}. \tag{4.1}$$

Physically, angle $\theta_\pm$ represents advanced/retarded deficit angle. It is an angle swept by the radius of A-particle while the signals propagate from A-particle to B-particle.

To solve equation (4.1) we use standard *method of secants*. Let us define function

$$f_\pm(r,\hat{\phi},z,\alpha,\theta) \;=\; \theta_\pm \;\mp\; \sqrt{1+r^2+z^2-2r\cos\theta_\pm} \;+\; \hat{\phi}. \tag{4.2}$$

Obviously, the roots of function f for given $r,\hat{\phi},z$ and α are the solutions of (4.1). Since the cosine always acquires values from $\langle -1,1\rangle$, it is clear from (4.1) that the solution $\theta_\pm$ lies in the interval

$$\pm\,\alpha\,\sqrt{1+r^2+z^2-2r}-\hat{\phi} \;\le\; \theta_\pm \;\le\; \pm\,\alpha\,\sqrt{1+r^2+z^2+2r}-\hat{\phi}. \tag{4.3}$$

Thus, we can define temporary interval (x_1^0,x_2^0) by

$$\begin{aligned} x_1^0 &= \pm\,\alpha\,\sqrt{1+r^2+z^2-2r}-\hat{\phi},\\ x_2^0 &= \pm\,\alpha\,\sqrt{1+r^2+z^2+2r}-\hat{\phi}. \end{aligned} \tag{4.4}$$

Moreover we define

$$y_1^0 \;=\; f_\pm(r,\hat{\phi},z,\alpha,x_1^0), \qquad y_2^0 \;=\; f_\pm(r,\hat{\phi},z,\alpha,x_2^0). \tag{4.5}$$

Now we construct a line connecting points (x_1^0,y_1^0) and (x_2^0,y_2^0), see figure 4.1. This line intersects horizontal axis at the point

$$x^0 \;=\; x_1^0 \;-\; y_1^0\,\frac{x_2^0-x_1^0}{y_2^0-y_1^0}.$$

This is a first approximation to desired solution as it should be evident from figure 4.1. The value of f at that point will be denoted

$$y^0 \;=\; f_\pm(r,\hat{\phi},z,\alpha,x^0).$$

Now there are three possibilities. It can happen that y^0 is sufficiently close to zero and we can say that x^0 is the solution to desired accuracy. If it is not, we can constrinct the interval where the true solution lies:

- if $y^0\,y_1^0<0$, the solution lies in the interval (x_1^0,x^0);
- if $y^0\,y^2<0$, the solution lies in the interval (x^0,x_2^0).

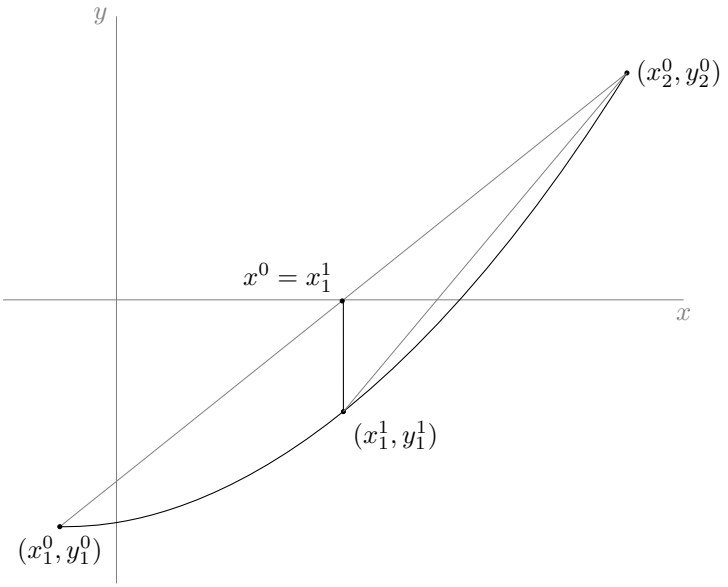


Figure 4.1: Illustration of the method of secants.

In figure 4.1, the second condition holds. We proceed iteratively and define

$$x_1^1 \;=\; x^0, \quad x_2^1 \;=\; x_2^0, \quad y_1^1 \;=\; f_{\pm}(r, \hat{\phi}, z, \alpha, x_1^1), \quad y_2^1 \;=\; y_2^0.$$

The line connecting points (x_1^1, y_1^y) and (x_2^1, y_2^1) intersects the $x-$axis at point $(x^1, 0)$ and the value x^1 is the next approximation to unknown root of function f, see figure 4.1.

To summarize, the initial interval (x_1^0, x_2^0) is defined by (4.4). Values y_1^0 and y_2^0 of function f are defined by (4.5). In every step of the algorithm, using the values x_1^i, x_2^i, y_1^i and y_2^i we construct an approximation of the root by

$$x^i = x_1^i \;-\; y_1^i \,\frac{x_2^i \,-\, x_1^i}{y_2^0 \,-\, y_1^0} \tag{4.6}$$

and evaluate the function f at this point:

$$y^i \;=\; f_{\pm}(r, \hat{\phi}, z, \alpha, x^i).$$

If the value is smaller then prescribed error ε, the algorithm stops and the approximated value of θ is

$$\theta_{\pm} \;=\; x^i.$$

Otherwise, the more narrow interval (x_1^{i+1}, x_2^{i+1}) containing the root is constructed by the rule

$$\begin{cases} x_1^{i+1} \;=\; x_1^i, \;\; x_2^{i+1} \;=\; x^i & \text{if } y_1^i \, y^i < 0, \\ x_1^{i+1} \;=\; x^i, \;\; x_2^{i+1} \;=\; x_2^i & \text{if } y_2^i \, y^i < 0. \end{cases} \tag{4.7}$$

This method is implemented in the source code listing 4.1.

Listing 4.1: Implementation of the method of secants

```
#include <math.h>
#include <iostream>
using namespace std;

double abs (double x)
{
    return (x<0)?(-x):x;
}

/* definition of function theta
r, phi, z corotating coordinates
alpha angular velocity from (0, 1)
theta test value of theta
sgn sign corresponding to character of the solution, retarded / advance
*/
double f( double r, double phi, double z,
          double alpha, double theta, int sgn)
{
    return
      theta - sgn*alpha*sqrt( 1+r*r+z*z-2*r*cos(theta) ) + phi;
}

/* solves equation for theta */
double solve(double r, double phi, double z,
             double alpha, int sgn)
{
    /* estimation of the solution corresponding to theta = 0, pi */
    double x1 = sgn*alpha*sqrt(1 + r*r + z*z - 2*r) - phi;
    double x2 = sgn*alpha*sqrt(1 + r*r + z*z + 2*r) - phi;
    /* values of f at x1, x2 */
    double y1 = f(r, phi, z, alpha, x1, sgn);
    double y2 = f(r, phi, z, alpha, x2, sgn);

    int steps = 0;
    /* approximating the solution */
    /* solution lies in (x1, x2)
       algorithm continues while |x1-x2| < epsilon */
    double x, y;
    y = y1;
    while ((abs(y) > 1E-10)&&(steps <= 100))
    {
        steps++;
        double k = (y2 - y1) / (x2 - x1);
        x = x1 - y1/k; // secant
        //x = (x1+x2)/2; //average
        y = f(r, phi, z, alpha, x, sgn);
        if (y1*y < 0) /* solution lies in (x1, x)*/
        {
            x2 = x; y2 = y;
        } else //solution lies in (x, x2)
        {
            x1 = x; y1 = y;
        }
    }
    if (steps >= 100)
    {
        cout << "Method failed to converge: ";
        cout << "precision achieved = " << y << endl;
    }
    return x;
}

```

```
63
64  // returns the triple (theta, R, rho)
65  double * rhothetaR(double r, double phi, double z, double alpha, int sgn)
66  {
67      double *result = new double[3];
68      result[0] = solve(r, phi, z, alpha, sgn); // theta
69      result[1] = sqrt( 1 + r*r + z*z - 2*r*cos(result[0])); // R;
70      result[2] = result[1] - sgn * alpha * r * sin(result[0]); //rho
71      return result;
72  }
73
74  double power (double x, int n)
75  {
76      double y = 1;
77      for(int i = 1; i <= abs(n); i++)
78          y *= x;
79      return (n>0)?y:1/y;
80  }
```

4.2 Half-binary-system code

Now we are able to compute values of functions $\theta_{\pm}, R_{\pm}$ and $\rho_{\pm}$ at arbitrary spacetime point. Consequently, we can compute values of the Christoffel symbols and thus solve the equations of motion numerically. We call this solution "half-binary-system" to emphasize that we still do not solve equations of motions of both particles: we assume that the motion of A-particle is circular at angular velocity α and investigate the motion of B-particle in the gravitational field of A-particle. Corresponding code is presneted in the listing 4.3 below.

Listing 4.2: "connection.h" header file

```
1   #ifndef CONNECTION_H_INCLUDED
2   #define CONNECTION_H_INCLUDED
3
4   double * rhothetaR(double r, double phi, double z, double alpha, int sgn);
5   double norm(double *x, double *params);
6   double normalize(double *x, double *params);
7   void einstein(double *f, double *params, double * x);
8   void einstein_retadv(double *f, double *params, double * x);
9   double equilib_mass1 (double *params, double *x, double b);
10
11  #endif // CONNECTION_H_INCLUDED
```

Listing 4.3: "connection.cpp" library

```
1   #include <math.h>
2   #include <iostream>
3   using namespace std;
4
5   double abs (double x)
6   {
7       return (x<0)?(-x):x;
8   }
9
10
11
12  /* definition of function theta
13
14  r, phi, z corotating coordinates
15  alpha angular velocity from (0, 1)
16  theta test value of theta
17  sgn sign corresponding to characetr of the solution, retarded / advance
```

```
18
19  */
20  double f( double r, double phi, double z, double alpha, double theta, int sgn)
21  {
22      return theta - sgn*alpha* sqrt( 1+r*r+z*z-2*r*cos(theta) ) + phi;
23  }
24
25
26  /* solves equation for theta */
27  double solve(double r, double phi, double z, double alpha, int sgn)
28  {
29      /* estimation of the solution corresponding to theta = 0, pi */
30      double x1 = sgn*alpha*sqrt(1 + r*r + z*z - 2*r) - phi;
31      double x2 = sgn*alpha*sqrt(1 + r*r + z*z + 2*r) - phi;
32      /* values of f at x1, x2 */
33      double y1 = f(r, phi, z, alpha, x1, sgn);
34      double y2 = f(r, phi, z, alpha, x2, sgn);
35
36      int steps = 0;
37      /* y1 * y2 < 0 ? */
38      if (y1*y2 > 0)
39      {
40          cout << "Relation y1*y2 <= 0 is not satisfied" << endl;
41          cout << "input values:" << endl;
42          cout << " r = " << r << ", phi = " << phi << ", z = "<< z << ",
43               alpha = " << alpha << ", sgn = " << sgn << endl;
44          cout << "estimated values:" << endl;
45          cout << " x1 = " << x1 << ", x2 = " << x2 << endl;
46          cout << " y1 = " << y1 << ", y2 = " << y2 << endl;
47          cin.ignore(1);
48          return 0; /* angle theta should never be zero */
49      }
50      /* approximating the solution */
51      /* solution lies in (x1, x2)
52         algorithm continues while |x1-x2| < epsilon */
53      double x, y;
54      y = y1;
55      while ((abs(y) > 1E-10)&&(steps <= 100))
56      {
57          steps++;
58          double k = (y2 - y1) / (x2 - x1);
59          x = x1 - y1/k; // secant
60          //x = (x1+x2)/2; //average
61          y = f(r, phi, z, alpha, x, sgn);
62          if (y1*y < 0) /* solution lies in (x1, x)*/
63          {
64              x2 = x;
65              y2 = y;
66          } else //solution lies in (x, x2)
67          {
68              x1 = x;
69              y1 = y;
70          }
71      }
72      if (steps >= 100)
73      {
74          cout << "Method failed to converge: ";
75          cout << "precision achieved = " << y << endl;
76      }
77      //cout << "steps = " << steps << endl;
78      //cout << "accuracy reached: y = " << y << endl;;
79      return x;
80  }
```

```
81
82
83  // returns the triple (theta, R, rho)
84  double * rhothetaR(double r, double phi, double z, double alpha, int sgn)
85  {
86      double *result = new double[3];
87      result[0] = solve(r, phi, z, alpha, sgn); // theta
88      result[1] = sqrt( 1 + r*r + z*z - 2*r*cos(result[0])); // R;
89      result[2] = result[1] - sgn * alpha * r * sin(result[0]); //rho
90      return result;
91  }
92
93  double power (double x, int n)
94  {
95      double y = 1;
96      for(int i = 1; i <= abs(n); i++)
97          y *= x;
98      return (n>0)?y:1/y;
99  }
100
101 double norm(double *x, double *params) {
102     double alpha = params[1];
103     return
104         x[4]*x[4] - x[5]*x[5] - x[1]*x[1]*x[6]*x[6] - x[7]*x[7]
105         - 2 * x[1] * x[1] * x[4] * x[6] * alpha
106         - x[1] * x[1] * x[4] * x[4] * alpha * alpha;
107 }
108
109 void normalize(double *x, double *params) {
110     double n = norm(x, params);
111     for (int i=0; i < 4; i++)
112         x[i+4] = x[i+4] / sqrt(n);
113 }
114
115
116 // params = {m, alpha, sign}
117 void einstein(double *f, double *params, double * x) {
118     double *rtR = rhothetaR(x[1], x[2], x[3], params[1], params[2]);
119     double COS = cos(rtR[0]);
120     double SIN = sin(rtR[0]);
121     double rho = rtR[2];
122     double m = params[0];
123     double alpha = params[1];
124     double gamma = 1/sqrt(1-params[1]*params[1]);
125     double r = x[1];
126     double z = x[3];
127     double u0 = x[4]; double u1 = x[5]; double u2 = x[6]; double u3 = x[7];
128     int sign = params[2];
129
130     //derivatives of coordinates
131     f[0] = x[4]; f[1] = x[5]; f[2] = x[6]; f[3] = x[7];
132
133     //second derivatives of coordinates
134     // d^2 t / ds^2
135     f[4]=gamma*m*power(rho,-3)*(2*x[4]*(COS*x[5]-r*x[5]+r*SIN*x[6]-x[7]*z)+
136 COS*x[4]*power(alpha,6)*power(r,3)*(2*SIN*(2*COS*SIN*x[5]+r*x[6]-2*r*x[6]*
137 power(COS,2))+rho*sign*x[4]*(1-2*power(COS,2)+4*power(SIN,2)))+
138 COS*SIN*power(alpha,7)*(1-2*power(COS,2))*power(r,4)*power(x[4],2)+
139 alpha*(2*SIN*x[5]*(rho*sign*x[4]+2*COS*x[5]-2*x[7]*z)+4*COS*x[6]*(x[5]
140 -SIN*x[6])*power(r,2)+SIN*power(r,3)*power(x[6],2)+r*(-2*COS*x[6]*(rho*sign
141 *x[4]+2*COS*x[5]-2*x[7]*z)+4*x[5]*x[6]*power(SIN,2)+SIN*(power(x[4],2)-
142 3*power(x[5],2)+power(x[7],2))))+power(alpha,5)*power(r,2)*(-2*r*x[6]*
143 (2*rho*sign*x[4]+r*SIN*x[6])*power(COS,3)-SIN*x[4]*(r*x[4]+4*rho*sign*x[5]
```

```
*power(SIN,2))+2*SIN*power(COS,2)*(4*rho*sign*x[4]*x[5]+2*r*SIN*x[5]*x[6]+
3*r*power(x[4],2))+COS*(2*r*rho*sign*x[4]*x[6]+8*r*rho*sign*x[4]*x[6]*
power(SIN,2)-2*power(SIN,3)*power(x[5],2)+SIN*(-((1+power(r,2)-
4*power(rho,2))*power(x[4],2))+power(x[5],2)+power(r,2)*power(x[6],2)+
power(x[7],2))))-power(alpha,3)*(2*rho*SIN*x[5]*(-(sign*x[4])+2*COS*rho*x[5])
+COS*SIN*power(r,4)*power(x[6],2)+power(r,3)*(2*COS*(rho*sign*x[4]+
2*COS*x[5])*x[6]+SIN*(-power(x[4],2)+power(x[6],2)-6*power(COS,2)*
power(x[6],2)))+r*(2*COS*rho*(sign*x[4]-2*COS*rho*x[5])*x[6]+4*x[5]*x[6]*
power(rho,2)*power(SIN,2)-2*power(SIN,3)*power(x[5],2)+SIN*(-4*COS*x[5]*x[7]*z
+4*rho*sign*x[4]*(4*COS*x[5]-x[7]*z)-power(x[4],2)+power(x[5],2)+
4*power(COS,2)*power(x[5],2)+power(x[7],2)))+power(r,2)*(4*rho*sign*SIN*x[4]*
(-x[5]+2*SIN*x[6])+(-8*rho*sign*x[4]*x[6]+4*x[6]*x[7]*z)*power(COS,2)
-4*x[5]*x[6]*power(COS,3)+COS*SIN*(8*SIN*x[5]*x[6]+5*power(x[4],2)-
3*power(x[5],2)-4*power(rho,2)*power(x[6],2)+power(x[7],2))))+
r*power(alpha,4)*(-2*SIN*(x[4]*x[6]*power(r,2)+2*SIN*x[4]*x[5]*power(rho,2)
+2*r*rho*sign*SIN*(SIN*x[5]*x[6]+power(x[4],2)))+
power(COS,2)*(-4*x[4]*(x[5]-3*SIN*x[6])*power(r,2)+2*x[4]*x[5]*(-1+
2*power(rho,2))+4*r*(-(x[4]*x[7]*z)+rho*sign*(2*SIN*x[5]*x[6]+
power(x[4],2))))-2*r*power(COS,3)*(-2*x[4]*x[5]+r*rho*sign*power(x[6],2))
+COS*(8*r*SIN*x[4]*x[6]*power(rho,2)-2*x[4]*(-(x[7]*z)+SIN*x[6]*power(r,3)
+r*(-x[5]+SIN*x[6]+4*x[5]*power(SIN,2)))+rho*sign*(-((1+power(r,2))*
power(x[4],2))+power(x[5],2)-6*power(SIN,2)*power(x[5],2)+power(r,2)*
power(x[6],2)+4*power(r,2)*power(SIN,2)*power(x[6],2)+power(x[7],2))))+
power(alpha,2)*(2*(-(x[4]*x[7]*z)+2*rho*sign*SIN*x[6]*(x[5]-SIN*x[6])*power(r,2)
+SIN*x[4]*x[6]*power(r,3)+r*(-(x[4]*x[5])+SIN*x[4]*x[6]+2*rho*sign*SIN
*x[6]*x[7]*z+2*x[4]*x[5]*power(SIN,2))+2*rho*sign*power(SIN,2)*power(x[5],2))+
power(COS,2)*(-6*r*x[4]*x[5]-4*rho*sign*power(x[5],2)+4*rho*sign*power(r,2)*
power(x[6],2))-COS*(-2*x[5]*(x[4]+2*rho*sign*x[7]*z)+x[4]*(-6*x[5]
+10*SIN*x[6])*power(r,2)+rho*sign*power(r,3)*power(x[6],2)+
r*(-6*x[4]*x[7]*z+rho*sign*(16*SIN*x[5]*x[6]+power(x[4],2)-
3*power(x[5],2)+power(x[7],2))))));

//d^2r/dt^2
f[5]=power(rho,-3)*(2*gamma*m*(alpha*u0+u2)*power(alpha,2)*power(COS,3)*
power(r,2)*(3*alpha*u0+u2+u2*power(alpha,2)*power(r,2)+u0*power(alpha,3)
*power(r,2))-
2*gamma*m*power(alpha,4)*power(COS,4)*power(r,3)*power(alpha*u0+u2,2)+
r*power(rho,3)*power(alpha*u0+u2,2)-COS*gamma*m*(4*r*SIN*u2*u3*z*power(alpha,2)
+2*alpha*u0*(2*alpha*rho*sign*u3*z-3*u2*power(r,2)+4*rho*sign*SIN*u2*
power(alpha,3)
*power(r,3)+u2*power(alpha,4)*power(r,4)+power(alpha,2)*(4*SIN*u1*power(rho,2)
+r*(r*u2+4*SIN*u3*z-u2*power(r,3)-4*r*u2*power(SIN,2))))-
2*alpha*u1*(-4*rho*sign*SIN*u3*z*power(alpha,2)+alpha*SIN*u2*power(r,2)*
(1+power(alpha,2)*(-1+2*power(SIN,2)))+r*(alpha*u3*z*(-1+power(alpha,2)*
(1-2*power(SIN,2)))+rho*sign*u2*(1+power(alpha,2)*(-1+6*power(SIN,2)))))+
(-1+8*r*rho*sign*SIN*power(alpha,3)+4*rho*sign*SIN*power(alpha,5)*power(r,3)
-power(alpha,4)*power(r,4)+power(alpha,6)*power(r,4)+
power(alpha,2)*(-1-6*power(r,2)+4*power(rho,2)))*power(u0,2)+
(1+4*r*rho*sign*SIN*power(alpha,3)+power(alpha,4)*power(r,2)*(-1+
2*power(SIN,2))
+power(alpha,2)*(-1+power(r,2)+2*power(SIN,2)))*power(u1,2)-power(r,2)*
power(u2,2)+power(alpha,2)*power(r,2)*power(u2,2)+4*rho*sign*SIN*
power(alpha,3)*power(r,3)*power(u2,2)-
power(alpha,2)*power(r,4)*power(u2,2)+power(alpha,4)*power(r,4)*power(u2,2)-
4*power(alpha,2)*power(r,2)*power(SIN,2)*power(u2,2)-
power(u3,2)+power(alpha,2)*power(u3,2)-power(alpha,2)*power(r,2)*power(u3,2)
+power(alpha,4)*power(r,2)*power(u3,2))+alpha*gamma*m*power(COS,2)*
(-4*r*u0*u2+2*u0*power(alpha,4)*power(r,2)*(3*rho*sign*SIN*u0+2*SIN*u3*z+
r*(u2-2*u2*power(SIN,2)))+
power(alpha,5)*power(r,3)*power(u0,2)+2*power(alpha,2)*
(-5*u0*u2*power(r,3)+2*r*u0*u2*power(rho,2)+rho*sign*(2*r*u2*u3*z+
2*SIN*power(u1,2)
-SIN*power(r,2)*power(u2,2)))-alpha*r*(5*power(u0,2)-power(u1,2)+
```

```
3*power(r,2)*power(u2,2)
+power(u3,2))+r*power(alpha,3)*(4*r*SIN*u2*u3*z+4*rho*sign*u0*(r*SIN*u2+u3*z)+
(-1-7*power(r,2)+4*power(rho,2))*power(u0,2)+(-1+2*power(SIN,2))*power(u1,2)
+power(r,2)*power(u2,2)-4*power(r,2)*power(SIN,2)*power(u2,2)+power(u3,2)))+
gamma*m*(2*u1*u3*z+2*u1*u3*z*power(alpha,2)*(-1+2*power(SIN,2))-
r*(4*alpha*u0*(u2+rho*sign*u3*z*power(alpha,3)+u2*power(alpha,2)*power(rho,2))
*power(SIN,2)+
u3*(u3-u3*power(alpha,2)+4*rho*sign*u2*z*power(alpha,3)*power(SIN,2))
+2*SIN*u1*u2*
(1+power(alpha,2)*(-1+2*power(SIN,2)))+(1+power(alpha,2)+
4*power(alpha,4)*power(rho,2)*power(SIN,2))*power(u0,2)+(-1+power(alpha,2)*
(1-2*power(SIN,2)))*power(u1,2))+
alpha*rho*sign*SIN*power(r,2)*(6*alpha*u0*u2+2*u0*u2*power(alpha,3)*
(-1+2*power(SIN,2))
-power(alpha,4)*power(u0,2)+power(u2,2)+
power(alpha,2)*(5*power(u0,2)+(-1+4*power(SIN,2))*power(u2,2)))+
(-1+power(alpha,2))*power(r,3)*power(alpha*u0+u2,2)+
rho*sign*SIN*power(alpha,3)*(power(u0,2)+(1-2*power(SIN,2))*power(u1,2)
-power(u3,2))+
alpha*SIN*(4*u0*u3*z+rho*sign*(power(u0,2)-power(u1,2)+power(u3,2)))));


    // d^2 phi / dt^2
   f[6] =
power(r,-1)*power(rho,-3)*(-2*u1*u2*power(rho,3)+COS*gamma*m*u0*power(alpha,7)*
power(r,4)*(-2*SIN*(2*COS*SIN*u1+r*u2-2*r*u2*power(COS,2))+rho*sign*u0*(-1+2*
power(COS,2)-4*power(SIN,2)))+COS*gamma*m*SIN*power(alpha,8)*(-1+2*power(COS,2))*
power(r,5)*power(u0,2)+gamma*m*power(alpha,6)*power(r,3)*(SIN*u0*(r*u0+4*rho*sign*
u1*power(SIN,2))-2*SIN*power(COS,2)*(4*rho*sign*u0*u1+2*r*SIN*u1*u2+3*r*power(u0,2))+
power(COS,3)*(4*r*rho*sign*u0*u2-2*SIN*power(u0,2)+2*SIN*power(r,2)*power(u2,2))+
COS*(-2*r*rho*sign*u0*u2-8*r*rho*sign*u0*u2*power(SIN,2)+2*power(SIN,3)*power(u1,2)+
SIN*((2+power(r,2)-4*power(rho,2))*power(u0,2)-power(u1,2)-power(r,2)*power(u2,2)-
power(u3,2))))+gamma*m*power(alpha,5)*power(r,2)*(4*u0*u1*power(COS,4)+
2*SIN*(u0*u2*power(r,2)+2*SIN*u0*u1*power(rho,2)+2*r*rho*sign*SIN*(SIN*u1*u2+
power(u0,2)))+4*power(COS,2)*(u0*(u1-3*SIN*u2)*power(r,2)+u0*u1*(-power(rho,2)+
power(SIN,2))+r*(u0*u3*z-rho*sign*(2*SIN*u1*u2+power(u0,2))))-2*power(COS,3)*
(2*u0*(2*r*u1+u3*z)+rho*sign*(power(u0,2)-power(r,2)*power(u2,2)))+
COS*(2*r*SIN*u0*(4*SIN*u1+u2+u2*power(r,2))-8*r*SIN*u0*u2*power(rho,2)+
rho*sign*((2+power(r,2)+4*power(SIN,2))*power(u0,2)+(-1+6*power(SIN,2))*power(u1,2)
-power(r,2)*power(u2,2)-4*power(r,2)*power(SIN,2)*power(u2,2)-power(u3,2))))+
gamma*m*(2*r*u2*(-(COS*u1)+r*u1+u3*z)+SIN*(power(u0,2)+power(u1,2)-power(r,2)*
power(u2,2)+power(u3,2)))+gamma*m*power(alpha,2)*(-2*r*u2*(r*u1+u3*z-(5*r*u1+2*u3*z)*
power(COS,2)+2*u1*power(COS,3)+COS*(-(r*rho*sign*u0)-u1+3*r*u3*z+3*u1*power(r,2)))
-4*u1*u2*power(r,2)*power(SIN,2)+2*power(SIN,3)*
power(u1,2)-SIN*(4*(COS-r)*u1*u3*z+4*rho*sign*u0*(2*r*u1+u3*z)+(-1+5*COS*r+4*
power(rho,2))*power(u0,2)+(1+9*COS*r-4*power(COS,2)-3*power(r,2))*power(u1,2)
-power(r,2)*power(u2,2)+2*power(COS,2)*power(r,2)*power(u2,2)-
5*COS*power(r,3)*power(u2,2)+power(r,4)*power(u2,2)+power(u3,2)+COS*r*power(u3,2)+
power(r,2)*power(u3,2)))+alpha*(4*gamma*m*u0*u1*power(COS,2)-2*(u0*u1*power(rho,3)+
gamma*m*(r*(rho*sign*SIN*u1*u2-2*u0*u3*z)+u0*(-2*u1+SIN*u2)*power(r,2)-2*u0*u1*
power(SIN,2)))-COS*gamma*m*(4*u0*(2*r*u1+u3*z)+rho*sign*(power(u0,2)+power(u1,2)-
power(r,2)*power(u2,2)+power(u3,2))))+gamma*m*r*power(alpha,4)*(4*r*u1*u2*
power(COS,4)+power(COS,2)*(4*SIN*u1*(-(rho*sign*u0)+u3*z)+
(-8*rho*sign*u0*u2+4*u2*u3*z)*power(r,2)+2*u2*(2*u1-3*SIN*u2)*power(r,3)+
r*(-2*u1*u2*(1+2*power(rho,2))+SIN*(6*power(u0,2)+8*power(u1,2))))-
2*power(COS,3)*(2*r*u2*(2*r*u1+u3*z)+SIN*(2*power(u1,2)-power(r,2)*power(u2,2)))+
SIN*(-4*rho*sign*u0*(u1-2*SIN*u2)*power(r,2)-4*rho*sign*u0*u1*power(SIN,2)+
power(r,3)*(-power(u0,2)+power(u2,2))+r*(-4*rho*sign*u0*u3*z+4*SIN*u1*u2*power(rho,2)
-2*power(u0,2)+power(u1,2)-2*power(SIN,2)*power(u1,2)+power(u3,2)))+
COS*(2*r*u2*(r*u1+u3*z+rho*sign*u0*(1+power(r,2)))+8*u1*u2*power(r,2)*
power(SIN,2)-2*power(SIN,3)*power(u1,2)+SIN*(-4*r*u1*u3*z+8*rho*sign*u0*
(3*r*u1+u3*z)+(-1+4*power(r,2)+8*power(rho,2))*power(u0,2)+
(1-3*power(r,2)+4*power(rho,2))*power(u1,2)-power(r,2)*power(u2,2)
```

```
+power(r,4)*power(u2,2)-4*power(r,2)*power(rho,2)*power(u2,2)+
power(u3,2)+power(r,2)*power(u3,2))))+gamma*m*power(alpha,3)*
(-2*SIN*(2*rho*SIN*u1*(rho*u0+sign*u3*z)+(2*SIN*u0*u1+u2*(u0+2*rho*sign*u3*z))*
power(r,2)+2*rho*sign*u2*(u1-SIN*u2)*power(r,3)+
u0*u2*power(r,4)+r*rho*sign*(-(u1*u2)+2*SIN*(power(u0,2)+2*power(u1,2))))+
2*power(COS,3)*(-4*r*u0*u1-2*rho*sign*power(u1,2)+rho*sign*power(r,2)*power(u2,2))+
4*power(COS,2)*(rho*u1*(rho*u0+sign*u3*z)+4*u0*u1*power(r,2)+r*(2*u0*u3*z+
rho*sign*(-3*SIN*u1*u2+power(u0,2)+2*power(u1,2)))-
rho*sign*power(r,3)*power(u2,2))+COS*(8*r*SIN*u0*u2*power(rho,2)-2*r*u0*
(4*r*u3*z+(4*u1-5*SIN*u2)*power(r,2)+4*u1*power(SIN,2))+
rho*sign*(8*r*SIN*u2*u3*z+4*r*u1*(6*r*SIN*u2-u3*z)-power(u0,2)+
(1-3*power(r,2)+2*power(SIN,2))*power(u1,2)-
power(r,2)*power(u2,2)+power(r,4)*power(u2,2)-4*power(r,2)*power(SIN,2)*
power(u2,2)+power(u3,2)+power(r,2)*power(u3,2)))));

  // d^2 z / dt^2
  f[7]=
gamma*m*power(rho,-3)*(-2*alpha*rho*sign*SIN*u1*u3-4*alpha*SIN*u0*u1*z+
2*rho*sign*SIN*u1*u3*power(alpha,3)z*power(u0,2)-
z*power(alpha,2)*power(u0,2)+2*r*(u1*(u3-u3*power(alpha,2)+2*rho*sign*
(alpha*u0+u2)*z*power(alpha,3)*power(SIN,2))+
SIN*(u2*(u3*(-1+power(alpha,2))+2*rho*sign*u0*z*power(alpha,2))+2*rho*
sign*z*power(alpha,3)*power(u0,2)))-z*power(u1,2)+
z*power(alpha,2)*power(u1,2)-2*z*power(alpha,2)*power(SIN,2)*power(u1,2)-
2*r*power(alpha,2)*power(COS,2)*(u1*(2*alpha*(rho*sign+alpha*r*SIN)*
(alpha*u0+u2)*z+u3*(-1+power(alpha,2)))+
r*z*(4*alpha*u0*u2+3*power(alpha,2)*power(u0,2)+power(u2,2)))+z*
(-1+power(alpha,2))*power(r,2)*power(alpha*u0+u2,2)+
2*z*power(alpha,4)*power(COS,3)*power(r,3)*power(alpha*u0+u2,2)+
z*power(u3,2)-z*power(alpha,2)*power(u3,2)+
COS*(2*u1*(2*(rho*sign*u0+r*SIN*(2*alpha*u0+u2))*z*power(alpha,2)+
u3*(-1+power(alpha,2))*(1+power(alpha,2)*power(r,2)))+
z*power(alpha,2)*(4*alpha*rho*sign*SIN+r*(1+power(alpha,2)*
(-1+2*power(SIN,2))))*power(u1,2)+
alpha*r*(4*u0*u2*z+2*u0*u2*z*power(alpha,2)*power(r,2)-2*u0*u2*z*
power(alpha,4)*power(r,2)-z*power(alpha,5)*power(r,2)*power(u0,2)-
2*rho*sign*(u2*(u3*(-1+power(alpha,2))+4*r*SIN*u0*z*power(alpha,3))
+2*r*SIN*z*power(alpha,4)*power(u0,2)+
2*r*SIN*z*power(alpha,2)*power(u2,2))+alpha*(2*r*SIN*u2*u3+5*z*power(u0,2)
+z*power(r,2)*power(u2,2)-z*power(u3,2))+power(alpha,3)*(-2*r*SIN*u2*u3+
z*power(r,2)*(power(u0,2)-power(u2,2))+z*(power(u0,2)+power(u3,2))))));

}


void einstein_retadv(double *f, double *params, double * x) {
    double *f1 = new double[8];
    double *f2 = new double[8];
    double *_params = new double[3];
    _params[0] = params[0]; _params[1] = params[1];
    _params[2] = -1; // compute retarded solutions
    einstein(f1, _params, x);
    _params[2] = 1; //compute advances solutions
    einstein(f2, _params, x);
    // add both solutions
    for (int i = 0; i < 8; i++)
        f[i] = 0.5 * (f1[i]+f2[i]);
    delete[] f1; delete[] f2;
    delete[] _params;
}

double equilib_mass1 (double *params, double *x, double b)
{
```

```
    double *rtR = rhothetaR(x[1], x[2], x[3], params[1], params[2]);
    double COS = cos(rtR[0]);
    double SIN = sin(rtR[0]);
    double rho = rtR[2];
    double alpha = params[1];
    double gamma = 1/sqrt(1-params[1]*params[1]);
    return
-(b*power(alpha,2)*power(gamma,-1)*power(rho,3)*power(-b+COS+
alpha*rho*SIN-b*power(alpha,2)+COS*power(alpha,2)+rho*SIN*power(alpha,3)-
8*b*COS*rho*SIN*power(alpha,3)+6*COS*power(alpha,2)*power(b,2)+
5*rho*SIN*power(alpha,3)*power(b,2)-
rho*SIN*power(alpha,5)*power(b,2)-power(alpha,2)*power(b,3)+power(alpha,4)*
power(b,3)-4*COS*rho*SIN*power(alpha,5)*power(b,3)+
COS*power(alpha,4)*power(b,4)-COS*power(alpha,6)*power(b,4)-5*b*power(alpha,2)*
power(COS,2)-b*power(alpha,4)*power(COS,2)+
6*rho*SIN*power(alpha,5)*power(b,2)*power(COS,2)-7*power(alpha,4)*
power(b,3)*power(COS,2)+power(alpha,6)*power(b,3)*power(COS,2)+
6*power(alpha,4)*power(b,2)*power(COS,3)+2*power(alpha,6)*power(b,4)*power(COS,3)
-2*power(alpha,6)*power(b,3)*power(COS,4)-
4*COS*power(alpha,2)*power(rho,2)+4*b*power(alpha,4)*
power(COS,2)*power(rho,2)-4*b*power(alpha,4)*power(rho,2)*power(SIN,2),-1));
}
```

4.3 Condition of equlibrium

Let us return to question whether the equilibrium configuration can be achieved by an appropriate choice of the parameters. Recal from section 3.4 that the equilibrium can be achieved only when both retarded and advanced solutions are taken into account. Then only one condition must be satisfied, namely

$$\Gamma^1_{00}\big|_B \;=\; 0$$

where the Christoffel symbol is given by (3.39). This condition can be solved with respect to mass explicitly:

$$\begin{aligned} m = &-\alpha^2 b\gamma\rho^3 \big[\alpha^3\rho\sin\theta - 4\alpha^2\rho^2\cos\theta + \alpha^2\cos\theta + \alpha\rho\sin\theta + 2\alpha^6 b^4\cos^3\theta \\ &-\alpha^6 b^4\cos\theta + \alpha^4 b^4\cos\theta - 2\alpha^6 b^3\cos^4\theta + \alpha^6 b^3\cos^2\theta - 4\alpha^5 b^3\rho\sin\theta\cos\theta \\ &-7\alpha^4 b^3\cos^2\theta + \alpha^4 b^3 - \alpha^2 b^3 - \alpha^5 b^2\rho\sin\theta + 6\alpha^5 b^2\rho\sin\theta\cos^2\theta + 6\alpha^4 b^2\cos^3\theta \\ &+5\alpha^3 b^2\rho\sin\theta + 6\alpha^2 b^2\cos\theta - 4\alpha^4 b\rho^2\sin^2\theta + 4\alpha^4 b\rho^2\cos^2\theta - \alpha^4 b\cos^2\theta \\ &-8\alpha^3 b\rho\sin\theta\cos\theta - 5\alpha^2 b\cos^2\theta - \alpha^2 b - b + \cos\theta\big]^{-1} \end{aligned} \tag{4.8}$$

So, we can simulate the motion of B-particle for given values of α and b and for equilibrium value of m given by (4.8). For example, we choose

$$\alpha \;=\; 0,1, \quad b \;=\; 3.$$

Equilibrium value of mass is then

$$m \;=\; 0,440\ 410\ 226\ 193.$$

The result of the simulation is plotted in figure 4.2. In this figure we compare different kind of solutions of Einstein's equations. Purely retarded solution (which is the usual one) yields expected result: particle is loosing the energy by radiation and its orbit decays, the radius of its orbit is decreasing. Thus, our model describes the inspiral process correctly, at least in the framework of linearized theory. The advanced solution describes particle absorbing the radiation coming from infinity, so its energy is increasing and the radius of its orbit, consequently, too. Finally, combining the retarded and advanced solution according to (3.42) gives the equilibrium configuration.

We have, thus, verified that in linearized gravity helically symmetric solutions exist and it is possible to achieve an equilibrium configuration of binary system. Moreover, we have justified the proof of existence present in [2] by numerical simulation.

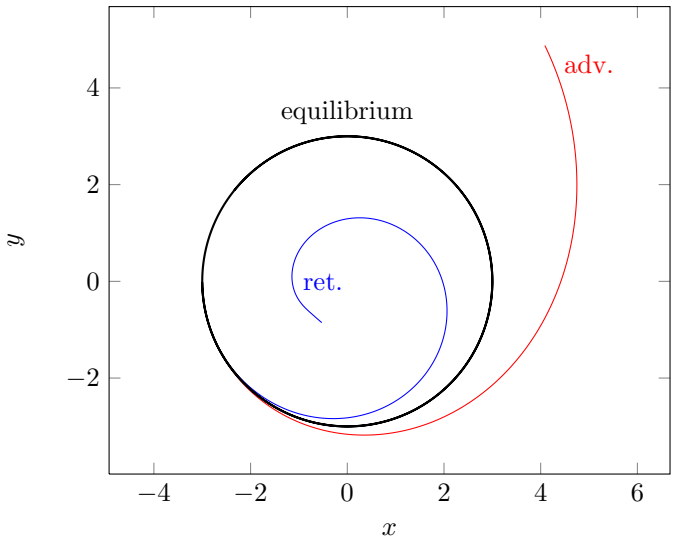


Figure 4.2: Comparison of solutions of different kinds. While equilibrium is impossible for purely retarded or purely advanced solution, their combination together with appropriate choice of the parameter yield equilibrium configuration. In this example, $\alpha = 0.1$, $b = 3$ and $m \doteq 0,44$.

4.4 Full binary system

Once more we emphasize that so far we did not discuss the equations of motion of the A-particle, we merely prescribed its motion. What we have found is the condition (4.8) which, if satisfied, ensures that the B-particle is orbiting along the circular trajectory of radius b at the same angular velocity α. Now we reverse the problem and ask whether also A-particle can move in a prescribed way in the field of B-particle. Fortunately, by the symmetry of the problem, one can immediately write down the solution for B-particle and repeat the calculations made in chapter 3. It turns out that the equilibrium mass of the B-particle is given by

$$\begin{aligned} m' = \alpha^2\rho^3\gamma' \big[&\alpha^2 + 2\alpha^6 b^4\cos^4\theta - \alpha^6 b^4\cos^2\theta + \alpha^4 b^4\cos^2\theta - 2\alpha^6 b^3\cos^3\theta + \alpha^6 b^3\cos\theta \\ &+\alpha^5 b^3\rho\sin\theta - 6\alpha^5 b^3\rho\sin\theta\cos^2\theta - 6\alpha^4 b^3\cos^3\theta - \alpha^3 b^3\rho\sin\theta - \alpha^2 b^3\cos\theta \\ &+4\alpha^5 b^2\rho\sin\theta\cos\theta + 4\alpha^4 b^2\rho^2\sin^2\theta - 4\alpha^4 b^2\rho^2\cos^2\theta + 7\alpha^4 b^2\cos^2\theta - \alpha^4 b^2 \\ &+8\alpha^3 b^2\rho\sin\theta\cos\theta + 5\alpha^2 b^2\cos^2\theta + \alpha^2 b^2 - \alpha^4 b\cos\theta - 5\alpha^3 b\rho\sin\theta + 4\alpha^2 b\rho^2\cos\theta \\ &-6\alpha^2 b\cos\theta - \alpha b\rho\sin\theta - b\cos\theta + 1\big]^{-1} \end{aligned} \tag{4.9}$$

which should be compared to (4.8).

Binary system will be in equilibrium if *both* conditions (4.8) and (4.9) are satisfied. Apparently, there are two parameters of binary system: angular velocity α and the radius of the orbit of B-particle b (recall that in our units the radius of A-particle is set to 1). Remaining parameters, the masses of both particles, can be then computed from equilibrium conditions. However, this is not the case, because the radius b and mass m' are not independent! They are constrained by the fact that the centre of mass of binary system must be located at the origin of the coordinates, i.e. by condition

$$\gamma\, m\, a = \gamma'\, m'\, b \quad i.e. \quad \frac{m'}{m} = b\sqrt{\frac{1-\alpha^2}{1-b^2\,\alpha^2}}. \tag{4.10}$$

Therefore, the mass of B-particle is automatically determined by the value of its radius. Then the only remaining free parameter is the angular velocity α and it must be chosen in such a way as to satisfy *both* conditions (4.8) and (4.9). It is obvious that for arbitrary b and m there will be no α such that all conditions are satisfied. In other words, the only parameter of binary system is the radius b from which an equilibrium value of α, m and m' can be computed. The

only exception is the case $b = 1$ when conditions (4.9) and (4.8) are identical so that the mass m can be chosen arbitrarily.

Implementation of the field of B-particle is listed in 4.4.

Listing 4.4: Implementation of the field of B-particle

```
1
2  // params = {alpha, b, m1, m2}
3  void einsteinprime(double *f, double *params, double *x, int sign) {
4      double b = params[1];
5      double *rtR = rhothetaRprime(x[1], x[2], x[3], params[0], b, sign);
6      double COS = cos(rtR[0]);
7      double SIN = sin(rtR[0]);
8      double rho = rtR[2];
9      double m = params[3];
10     double alpha = params[0];
11     double gamma = 1/sqrt(1-b*b*alpha*alpha);
12     double r = x[1];
13     double z = x[3];
14     double u0 = x[4]; double u1 = x[5]; double u2 = x[6]; double u3 = x[7];
15     delete rtR;
16     f[0] = x[4]; f[1] = x[5]; f[2] = x[6]; f[3] = x[7];
17
18     f[4] = gamma*m*power(rho,-3)*(2*u0*(b*COS*u1 - r*u1 + b*r*SIN*u2 - u3*z) +
19      COS*u0*power(alpha,6)*power(b,3)*power(r,3)*(2*b*SIN*(2*COS*SIN*u1 + r*u2
20      - 2*r*u2*power(COS,2)) +
21      rho*sign*u0*(1 - 2*power(COS,2) + 4*power(SIN,2))) +
22      COS*SIN*power(alpha,7)*power(b,4)*(1 - 2*power(COS,2))*power(r,4)*
23      power(u0,2) + power(alpha,5)*power(b,2)*power(r,2)*
24      (b*u0*(-4*rho*sign*SIN*u1*(-2*power(COS,2) + power(SIN,2)) +
25      r*(2*COS*rho*sign*u2*(1 - 2*power(COS,2)) +
26      SIN*u0*(-1 + 6*power(COS,2)) + 8*COS*rho*sign*u2*power(SIN,2)))
27      - COS*SIN*(power(r,2) - 4*power(rho,2))*power(u0,2) +
28      COS*SIN*power(b,2)*(4*COS*r*SIN*u1*u2 - power(u0,2) + power(u1,2) -
29      2*power(SIN,2)*power(u1,2) + power(r,2)*power(u2,2) -
30      2*power(COS,2)*power(r,2)*power(u2,2) + power(u3,2))) +
31      alpha*b*(2*SIN*u1*(rho*sign*u0 + 2*b*COS*u1 - 2*u3*z) +
32      4*COS*u2*(u1 - b*SIN*u2)*power(r,2) + SIN*power(r,3)*power(u2,2) +
33      r*(-2*COS*u2*(rho*sign*u0 + 2*b*COS*u1 - 2*u3*z) +
34      4*b*u1*u2*power(SIN,2) + SIN*(power(u0,2) - 3*power(u1,2) + power(u3,2)))) +
35      b*power(alpha,2)*(2*b*power(COS,2)*
36      (-3*r*u0*u1 - 2*rho*sign*power(u1,2) +
37      2*rho*sign*power(r,2)*power(u2,2)) +
38      2*(r*SIN*u0*u2*power(b,2) +
39      r*SIN*u2*(2*r*rho*sign*u1 + 2*rho*sign*u3*z + u0*power(r,2)) -
40      b*(u0*u3*z + r*u0*u1*(1 - 2*power(SIN,2)) -
41      2*rho*sign*power(SIN,2)*power(u1,2) +
42      2*rho*sign*power(r,2)*power(SIN,2)*power(u2,2))) -
43      COS*(-2*u1*(2*rho*sign*u3*z + u0*power(b,2)) +
44      u0*(-6*u1 + 10*b*SIN*u2)*power(r,2) +
45      rho*sign*power(r,3)*power(u2,2) +
46      r*(-6*u0*u3*z + rho*sign*(16*b*SIN*u1*u2 + power(u0,2)
47      - 3*power(u1,2) + power(u3,2))))) + b*power(alpha,3)*
48      (r*u0*(4*r*rho*sign*SIN*u1 + 4*rho*sign*SIN*u3*z +
49      (SIN*u0 - 2*COS*rho*sign*u2)*power(r,2)) +
50      power(b,2)*(2*rho*sign*SIN*u0*u1 +
51      4*COS*u1*u2*power(r,2)*(power(COS,2) - 2*power(SIN,2)) +
52      SIN*(-1 + 6*power(COS,2))*power(r,3)*power(u2,2) +
53      r*(-2*COS*rho*sign*u0*u2 + 2*power(SIN,3)*power(u1,2) +
54      SIN*(power(u0,2) - (1 + 4*power(COS,2))*power(u1,2) -
55      power(u3,2)))) - b*(4*r*u2*power(COS,2)*
56      (-2*r*rho*sign*u0 + r*u3*z + u1*power(r,2) - u1*power(rho,2)) +
57      4*r*rho*(2*r*sign*u0 + rho*u1)*u2*power(SIN,2) +
58      COS*SIN*(4*r*u1*(4*rho*sign*u0 - u3*z) +
```

```
      4*power(rho,2)*power(u1,2) + power(r,4)*power(u2,2) +
      power(r,2)*(5*power(u0,2) - 3*power(u1,2) -
      4*power(rho,2)*power(u2,2) + power(u3,2))))) +
      b*r*power(alpha,4)*(-2*COS*u0*(COS*u1 + r*SIN*u2)*power(b,3) -
      2*b*u0*(COS*r*SIN*u2*(power(r,2) - 4*power(rho,2)) +
      2*power(COS,2)*(-(r*rho*sign*u0) + r*u3*z + u1*power(r,2) -
      u1*power(rho,2)) + 2*rho*(r*sign*u0 + rho*u1)*power(SIN,2)) -
      COS*rho*sign*power(r,2)*power(u0,2) +
      power(b,2)*(4*r*SIN*(3*r*u0 + 2*rho*sign*u1)*u2*power(COS,2) -
      2*r*SIN*u2*(r*u0 + 2*rho*sign*u1*power(SIN,2)) -
      2*r*power(COS,3)*(-2*u0*u1 + r*rho*sign*power(u2,2)) +
      COS*(2*u0*(r*u1 + u3*z - 4*r*u1*power(SIN,2)) +
      rho*sign*(-power(u0,2) + power(u1,2) -
      6*power(SIN,2)*power(u1,2) + power(r,2)*power(u2,2) +
      4*power(r,2)*power(SIN,2)*power(u2,2) + power(u3,2))))));

     f[5] = power(rho,-3)*(-(alpha*gamma*m*power(b,2)*
        (r*u0*power(alpha,3)*((8*COS*rho*sign*SIN*u2 +
         u0*(-1 + 7*power(COS,2)))*power(r,2) -
         4*rho*(rho*u0 + sign*u3*z)*(power(COS,2) - power(SIN,2))) +
         4*r*u0*u2*(power(COS,2) + power(SIN,2)) +
         4*COS*rho*sign*SIN*power(alpha,4)*power(r,3)*power(u0,2) +
         alpha*(-2*COS*SIN*u1*u2*power(r,2) + 2*u1*u3*z*(1 - 2*power(SIN,2)) +
         r*(u3*(4*COS*SIN*u2*z + u3*(-1 + power(COS,2))) +
         (1 + 5*power(COS,2))*power(u0,2) -
         (-1 + power(COS,2) + 2*power(SIN,2))*power(u1,2)) +
         (-1 + 3*power(COS,2))*power(r,3)*power(u2,2)) +
         2*power(alpha,2)*(r*u2*power(COS,2)*
         (-2*rho*(rho*u0 + sign*u3*z) + 5*u0*power(r,2)) +
         r*u2*(-(u0*power(r,2)) + 2*rho*(rho*u0 + sign*u3*z)*power(SIN,2)) +
         2*COS*SIN*(2*rho*u1*(rho*u0 + sign*u3*z) +
         r*(2*u0*u3*z + rho*sign*(2*power(u0,2) + power(u1,2))) +
         rho*sign*power(r,3)*power(u2,2))))) +
         r*power(rho,3)*power(alpha*u0 + u2,2) -
         gamma*m*(-2*u1*u3*z + power(r,3)*power(alpha*u0 + u2,2) +
         r*(power(u0,2) - power(u1,2) + power(u3,2))) +
         COS*gamma*m*r*power(alpha,4)*power(b,4)*
         (2*r*SIN*u1*u2*(-1 + 2*power(SIN,2)) -
         2*power(COS,3)*power(r,2)*power(alpha*u0 + u2,2) +
         COS*(2*alpha*u0*u2*power(r,2)*(1 - 2*power(SIN,2)) +
         (-1 + power(alpha,2)*power(r,2))*power(u0,2) +
         (-1 + 2*power(SIN,2))*power(u1,2) + power(r,2)*power(u2,2) -
         4*power(r,2)*power(SIN,2)*power(u2,2) + power(u3,2))) +
         gamma*m*power(alpha,2)*power(b,3)*
         (2*(alpha*u0 + u2)*power(COS,3)*power(r,2)*
         (3*alpha*u0 + u2 + u2*power(alpha,2)*power(r,2) +
          u0*power(alpha,3)*power(r,2))
          + 2*alpha*SIN*power(COS,2)*
         (2*alpha*u2*(rho*sign*u0 + u3*z)*power(r,2) +
         u0*(3*rho*sign*u0 + 2*u3*z)*power(alpha,2)*power(r,2) +
         rho*sign*(2*power(u1,2) - power(r,2)*power(u2,2))) +
         SIN*(2*r*u1*u2*(1 - 2*power(SIN,2)) +
         2*rho*sign*u0*u2*power(alpha,2)*power(r,2)*(-1 + 2*power(SIN,2)) -
         rho*sign*power(alpha,3)*power(r,2)*power(u0,2) +
         alpha*rho*sign*(power(u0,2) + power(u1,2) -
         2*power(SIN,2)*power(u1,2) -
         power(r,2)*power(u2,2) + 4*power(r,2)*power(SIN,2)*power(u2,2) -
         power(u3,2))) - COS*(2*alpha*u0*u2*power(r,2)*
         (1 + power(alpha,2)*power(r,2) - 4*power(SIN,2)) +
         2*alpha*r*u1*(rho*sign*u2*(1 - 6*power(SIN,2)) +
         alpha*u3*z*(-1 + 2*power(SIN,2))) +
         (-1 + power(alpha,4)*power(r,4))*power(u0,2) +
         (1 + power(alpha,2)*power(r,2))*(-1 + 2*power(SIN,2))*power(u1,2) +
```

```
        power(r,2)*power(u2,2) + power(alpha,2)*power(r,4)*power(u2,2) -
        4*power(r,2)*power(SIN,2)*power(u2,2) + power(u3,2) +
        power(alpha,2)*power(r,2)*power(u3,2))) +
        b*gamma*m*(COS*(2*alpha*r*u1*(rho*sign*u2 - alpha*u3*z) +
        2*alpha*u0*(-2*alpha*rho*sign*u3*z + 3*u2*power(r,2) +
        u2*power(alpha,2)*power(r,4)) +
        (1 + power(alpha,4)*power(r,4) +
        power(alpha,2)*(6*power(r,2) - 4*power(rho,2)))*power(u0,2) -
        (1 + power(alpha,2)*power(r,2))*power(u1,2) +
        (1 + power(alpha,2)*power(r,2))*(power(r,2)*power(u2,2) + power(u3,2))) +
        SIN*(-2*r*u1*u2 + 6*rho*sign*u0*u2*power(alpha,2)*power(r,2) +
        5*rho*sign*power(alpha,3)*power(r,2)*power(u0,2) +
        alpha*(4*u0*u3*z + rho*sign*
        (power(u0,2) - power(u1,2) + power(r,2)*power(u2,2) + power(u3,2))))));

    f[6] = power(r,-1)*power(rho,-3)*(2*u2*(gamma*m*r*(r*u1 + u3*z)
     - u1*power(rho,3)) +
     COS*gamma*m*u0*power(alpha,7)*power(b,3)*power(r,4)*
     (-2*b*SIN*(2*COS*SIN*u1 + r*u2 - 2*r*u2*power(COS,2)) +
     rho*sign*u0*(-1 + 2*power(COS,2) - 4*power(SIN,2))) +
     COS*gamma*m*SIN*power(alpha,8)*power(b,4)*(-1 + 2*power(COS,2))*power(r,5)*
     power(u0,2) + b*gamma*m*(-2*COS*r*u1*u2 +
     SIN*(power(u0,2) + power(u1,2) - power(r,2)*power(u2,2) + power(u3,2))) +
     gamma*m*power(alpha,6)*power(b,2)*power(r,3)*
     (b*u0*(4*rho*sign*SIN*u1*(-2*power(COS,2) + power(SIN,2)) +
     r*(2*COS*rho*sign*u2*(-1 + 2*power(COS,2)) + SIN*(u0 - 6*u0*power(COS,2)) -
     8*COS*rho*sign*u2*power(SIN,2))) +
     COS*SIN*(power(r,2) - 4*power(rho,2))*power(u0,2) -
     COS*SIN*power(b,2)*(4*COS*r*SIN*u1*u2 + 2*(-1 + power(COS,2))*power(u0,2) +
     power(u1,2) - 2*power(SIN,2)*power(u1,2) + power(r,2)*power(u2,2) -
     2*power(COS,2)*power(r,2)*power(u2,2) + power(u3,2))) +
     b*gamma*m*power(alpha,2)*(2*rho*sign*u0*
     (-4*r*SIN*u1 - 2*SIN*u3*z + COS*u2*power(r,2)) -
     4*SIN*power(rho,2)*power(u0,2) +
     power(b,2)*(2*COS*r*u1*u2*(1 - 2*power(COS,2)) + 2*power(SIN,3)*power(u1,2) +
     SIN*(power(u0,2) + (-1 + 4*power(COS,2))*power(u1,2) +
     power(r,2)*power(u2,2) - 2*power(COS,2)*power(r,2)*power(u2,2) -
     power(u3,2))) - r*(-4*SIN*u1*u3*z + 6*COS*u1*u2*power(r,2) +
     SIN*power(r,3)*power(u2,2) +
     r*(6*COS*u2*u3*z + SIN*(-3*power(u1,2) + power(u3,2)))) +
     b*(2*r*u2*(5*r*u1 + 2*u3*z)*power(COS,2) -
     2*r*u2*(u3*z + r*(u1 + 2*u1*power(SIN,2))) +
     COS*SIN*(-4*u1*u3*z + 5*power(r,3)*power(u2,2) -
     r*(5*power(u0,2) + 9*power(u1,2) + power(u3,2))))) +
     alpha*(2*u0*(2*gamma*m*r*(r*u1 + u3*z) - u1*power(rho,3)) +
     4*gamma*m*u0*u1*power(b,2)*(power(COS,2) + power(SIN,2)) -
     b*gamma*m*(2*r*SIN*(r*u0 + rho*sign*u1)*u2 +
     COS*(4*u0*(2*r*u1 + u3*z) +
     rho*sign*(power(u0,2) + power(u1,2) - power(r,2)*power(u2,2) +
     power(u3,2))))) + b*gamma*m*r*power(alpha,4)*
     (-(r*u0*(4*r*rho*sign*SIN*u1 + 4*rho*sign*SIN*u3*z +
     (SIN*u0 - 2*COS*rho*sign*u2)*power(r,2))) +
     COS*power(b,3)*(2*COS*r*u1*u2*(-1 + 2*power(COS,2)) -
     2*power(SIN,3)*power(u1,2) +
     SIN*(-power(u0,2) + power(u1,2) - 4*power(COS,2)*power(u1,2) -
     power(r,2)*power(u2,2) + 2*power(COS,2)*power(r,2)*power(u2,2) +
     power(u3,2))) + power(b,2)*
     (2*COS*u1*u2*power(r,2)*(1 - 4*power(COS,2) + 4*power(SIN,2)) -
     4*SIN*u1*((rho*sign*u0 - u3*z)*power(COS,2) + rho*sign*u0*power(SIN,2)) +
     SIN*(1 - 6*power(COS,2))*power(r,3)*power(u2,2) +
     r*(2*COS*u2*(rho*sign*u0 + u3*z*(1 - 2*power(COS,2))) -
     2*power(SIN,3)*power(u1,2) +
     SIN*((-2 + 6*power(COS,2))*power(u0,2) + power(u1,2) +
```

```
185       8*power(COS,2)*power(u1,2) + power(u3,2)))) +
186       b*(4*r*u2*power(COS,2)*(-2*r*rho*sign*u0 + r*u3*z + u1*power(r,2) -
187       u1*power(rho,2)) + 4*r*rho*(2*r*sign*u0 + rho*u1)*u2*power(SIN,2) +
188       COS*SIN*(4*r*u1*(6*rho*sign*u0 - u3*z) +
189       4*rho*(2*sign*u0*u3*z + 2*rho*power(u0,2) + rho*power(u1,2)) +
190       power(r,4)*power(u2,2) +
191       power(r,2)*(4*power(u0,2) - 3*power(u1,2) - 4*power(rho,2)*power(u2,2) +
192       power(u3,2))))) + b*gamma*m*power(alpha,5)*power(r,2)*
193       (2*COS*u0*power(b,3)*(r*SIN*u2 + 2*u1*power(COS,3) + 2*COS*u1*power(SIN,2)) +
194       2*b*u0*(COS*r*SIN*u2*(power(r,2) - 4*power(rho,2)) +
195       2*power(COS,2)*(-(r*rho*sign*u0) + r*u3*z + u1*power(r,2) -
196       u1*power(rho,2)) + 2*rho*(r*sign*u0 + rho*u1)*power(SIN,2)) +
197       COS*rho*sign*power(r,2)*power(u0,2) -
198       power(b,2)*(4*r*SIN*(3*r*u0 + 2*rho*sign*u1)*u2*power(COS,2) -
199       2*r*SIN*u2*(r*u0 + 2*rho*sign*u1*power(SIN,2)) +
200       2*power(COS,3)*(2*u0*(2*r*u1 + u3*z) +
201       rho*sign*(power(u0,2) - power(r,2)*power(u2,2))) +
202       COS*(-8*r*u0*u1*power(SIN,2) +
203       rho*sign*(-2*(1 + 2*power(SIN,2))*power(u0,2) + power(u1,2) -
204       6*power(SIN,2)*power(u1,2) + power(r,2)*power(u2,2) +
205       4*power(r,2)*power(SIN,2)*power(u2,2) + power(u3,2))))) +
206       b*gamma*m*power(alpha,3)*(b*(2*COS*r*SIN*u2*
207       (12*r*rho*sign*u1 + 4*rho*(rho*u0 + sign*u3*z) + 5*u0*power(r,2)) +
208       4*power(COS,2)*(rho*u1*(rho*u0 + sign*u3*z) + 4*u0*u1*power(r,2) +
209       r*(2*u0*u3*z + rho*sign*(power(u0,2) + 2*power(u1,2))) -
210       rho*sign*power(r,3)*power(u2,2)) +
211       4*power(SIN,2)*(-(rho*u1*(rho*u0 + sign*u3*z)) - u0*u1*power(r,2) -
212       r*rho*sign*(power(u0,2) + 2*power(u1,2)) + rho*sign*power(r,3)*power(u2,2)
213       )) + power(b,2)*(2*r*SIN*(-(r*u0) + rho*sign*u1)*u2 -
214       12*r*rho*sign*SIN*u1*u2*power(COS,2) +
215       2*power(COS,3)*(-4*r*u0*u1 - 2*rho*sign*power(u1,2) +
216       rho*sign*power(r,2)*power(u2,2)) +
217       COS*(-8*r*u0*u1*power(SIN,2) +
218       rho*sign*(-power(u0,2) + power(u1,2) + 2*power(SIN,2)*power(u1,2) -
219       power(r,2)*power(u2,2) - 4*power(r,2)*power(SIN,2)*power(u2,2) +
220       power(u3,2)))) + r*
221       (-2*r*SIN*u2*(2*r*rho*sign*u1 + 2*rho*sign*u3*z + u0*power(r,2)) +
222       COS*(-4*rho*sign*u1*u3*z - 8*u0*u1*power(r,2) +
223       rho*sign*power(r,3)*power(u2,2) +
224       r*(-8*u0*u3*z + rho*sign*(-3*power(u1,2) + power(u3,2))))));
225
226      f[7] = gamma*m*power(rho,-3)*(2*r*u1*u3 -
227       2*COS*r*(COS*u1 + r*SIN*u2)*u3*power(alpha,4)*
228       power(b,4) - z*power(r,2)*power(alpha*u0 + u2,2) +
229       power(alpha,2)*power(b,3)*(2*SIN*(alpha*rho*sign*u1 + r*u2)*u3 -
230       4*SIN*u1*(alpha*u0 + u2)*z*power(alpha,2)*power(COS,2)*power(r,2) +
231       2*z*power(alpha,2)*power(COS,3)*power(r,3)*power(alpha*u0 + u2,2) +
232       COS*(2*u1*(u3 + u3*power(alpha,2)*power(r,2)) +
233       r*z*power(alpha,2)*(-1 + 2*power(SIN,2))*power(u1,2) -
234       alpha*r*(2*rho*sign*u2*u3 +
235       alpha*z*(2*alpha*u0*u2*power(r,2) +
236       (-1 + power(alpha,2)*power(r,2))*power(u0,2) +
237       power(r,2)*power(u2,2) - power(u3,2))))) -
238       z*(power(u0,2) + power(u1,2) - power(u3,2)) +
239       power(alpha,2)*power(b,2)*(2*r*u1*
240       (u3*(-1 + power(COS,2)) + 2*z*
241       (COS*SIN*u2 + rho*sign*u0*power(alpha,2)*(-power(COS,2) + power(SIN,2)) +
242       alpha*(2*COS*SIN*u0 - rho*sign*u2*power(COS,2) + rho*sign*u2*power(SIN,2))
243       )) + power(r,2)*(-2*z*power(COS,2)*
244       (4*alpha*u0*u2 + 3*power(alpha,2)*power(u0,2) + power(u2,2)) -
245       2*COS*SIN*(-(u2*u3) + 4*rho*sign*u0*u2*z*power(alpha,2) +
246       2*rho*sign*z*power(alpha,3)*power(u0,2) + 2*alpha*rho*sign*z*power(u2,2))\
247       +z*power(alpha*u0 + u2,2)) -
```

```
     z*(power(u0,2) + (-1 - 4*alpha*COS*rho*sign*SIN + 2*power(SIN,2))*power(u1,2) +
     power(u3,2))) + b*(-2*SIN*
     (r*u2*u3 + alpha*u1*(rho*sign*u3 + 2*u0*z) -
     2*r*rho*sign*u0*u2*z*power(alpha,2) -
     2*r*rho*sign*z*power(alpha,3)*power(u0,2)) +
     COS*(-2*u1*(u3 - 2*rho*sign*u0*z*power(alpha,2) +
     u3*power(alpha,2)*power(r,2)) + r*z*power(alpha,2)*power(u1,2) +
     alpha*r*(2*rho*sign*u2*u3 +
     z*(4*u0*u2 + 2*u0*u2*power(alpha,2)*power(r,2) + 5*alpha*power(u0,2) +
     power(alpha,3)*power(r,2)*power(u0,2) + alpha*power(r,2)*power(u2,2) -
     alpha*power(u3,2))))));
}


// params = {alpha, b, m1, m2}
void einstein_retadv(double *f, double *params, double * x) {
    // A - particle, retarded
    double *f1 = new double[8];
    einstein(f1, params, x, -1);
    // A - particle, advanced
    double *f2 = new double[8];
    einstein(f2, params, x, +1);
    // B - particle, retarded
    double *f3 = new double[8];
    einsteinprime(f3, params, x, -1);
    // B - particle, advanced
    double *f4 = new double[8];
    einsteinprime(f4, params, x, +1);
    // superposition
    for (int i = 0; i < 8; i++)
        f[i] = 0.5 * (f1[i] + f2[i] + f3[i] + f4[i]);
    // flat metric is included twice
    double u0 = x[4]; double u1 = x[5];
    double u2 = x[6];
    double r = x[1]; double alpha = params[0];
    f[5] -= r*power(alpha*u0+u2, 2);
    f[6] += 2*u1*(alpha*u0+u2)/r;
    delete[] f1; delete[] f2;
    delete[] f3; delete[] f4;
}
```

4.5 Lyapunov exponents

Binary system of A-particle and B-particle is not asymptotically flat because of the presence of both incoming and outgoing radiation. It is shown in [2] that the radiation does not satisfy usual peeling properties, namely, that the leading term in radiative component Ψ_4 of the Weyl tensor behaves as

$$\Psi_4^{\pm} \sim \frac{\sin\theta_{\pm}}{r},$$

where $\sin\theta_{\pm}$ is a rapidly oscillating function as $\theta_{\pm} \to \infty$ for $r \to \infty$. Thus, our binary system is not an isolated system in a strict sense (usual peeling requires $\sin\theta_{\pm}$ to be $r-$independent).

Nevertheless, for small masses and small angular velocities, the solution has correct Newtonian limit. The motion of test particles far from binary system is expected to coincide with the motion of the particles in the Newtonian field of point mass $m + m'$. However, in the section 2.5, figure 2.6, we have seen that the motion of the test particle can be rather complicated in the vicinity of the binary system and, in fact, it can be chaotic. Hence, it is interesting to investigate the behaviour and chaotic nature of geodesics near the binary system.

As a measure of chaotic properties we have chosen the so-called *Lyapunov exponents*. The idea is to compare two geodesics starting with almost identical but slightly different initial

conditions. System of differential equations is stable if the distance of geodesics does not grow too rapidly. Let us formulate this more precisely.

Let us consider a *dynamical system*, i.e. the system of first-order ordinary differential equations

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{4.11}$$

where $\boldsymbol{x} \in M$ is the vector of unknown variables and $\boldsymbol{f}(\boldsymbol{x})$ is the vector of first derivatives. The phase space is denoted by M. Let Φ_t be a flow of vector field $\boldsymbol{f}$ which means that Φ_t is a mapping

$$\Phi_t : M \mapsto M$$

such that for any point $\boldsymbol{x}_0 \in M$ is mapped to a point $\boldsymbol{x}(t) = \Phi_t \boldsymbol{x}_0$ such that $\boldsymbol{x}(t)$ is a solution of (4.11) with initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$:

$$\boldsymbol{x}(0) \;=\; \boldsymbol{x}_0, \quad \dot{\boldsymbol{x}}(t) \;=\; \boldsymbol{f}(\boldsymbol{x}(t)). \tag{4.12}$$

Now, let us choose two initial points $\boldsymbol{x}_0$ and $\boldsymbol{x}_0'$ which differ by a displacement vector $\boldsymbol{u}_0$:

$$\boldsymbol{x}_0' \;=\; \boldsymbol{x}_0 \,+\, \boldsymbol{u}_0.$$

Two solutions starting from these points are represented by flows

$$\Phi_t \boldsymbol{x}_0 \quad \text{and} \quad \Phi_t \boldsymbol{x}_0'.$$

We define a displacement vector $\boldsymbol{u}_t$ at time $t > 0$ by

$$\boldsymbol{u}_t \;=\; \Phi_t \boldsymbol{x}_0' \,-\, \Phi_t \boldsymbol{x}_0.$$

The *Lyapunov exponent* is defined by

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \log \frac{\|\boldsymbol{u}_t\|}{\|\boldsymbol{u}_0|} \tag{4.13}$$

where $\|\cdot\|$ is the norm. Clearly, if the distance $\|\boldsymbol{u}_t\|$ is linear in t, the Lyapunov exponent vanishes and the system is *stable*. If, on the other hand, $\lambda > 0$, two solutions are diverging in an exponential rate,

$$\|\boldsymbol{u}_t\| \;\sim\; e^{\lambda t}$$

and the system is *unstable*. Finally, for $\lambda < 0$ two solutions asymptotically converge and the system is *asymptotically stable*. It can be shown[7] that, numerically, the Lyapunov exponent can be found as a solution of the system

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ (D\boldsymbol{f}) \cdot \boldsymbol{\phi} \end{bmatrix} \tag{4.14}$$

where $D\boldsymbol{f}$ is a matrix of first derivatives of $\boldsymbol{f}$.

Here we apply a more naive but more straightforward method to find the Lyapunov exponent. Our phase space M is coordinatized by eight variables

$$x_a \;=\; (x^\mu, u^\mu) \;=\; (t, r, \hat{\phi}, z, u^t, u^r, u^{\hat{\phi}}, u^z).$$

We fix six of these coordinates and leave remaining two as free parameters p_1 and p_2. Next we choose the range of parameters p_1 and p_2:

$$p_1 \in (p_1^{\min}, p_1^{\max}), \quad p_2 \in (p_2^{\min}, p_2^{\max}).$$

Next we compute the geodesic starting from each point of $(p_1, p_2)-$plane and obtain the solution $x_a^1(s)$ where the proper time s ranges from 0 to some maximal value S. Then we slightly perturb

one of fixed coordinates and compute the geodesic $x_a^2(s)$ for same p_1 and p_2. The Lyapunov exponent is then calculated by formula

$$\lambda(p_1, p_2) = \frac{1}{S} \log \frac{\|x_a^1(S) - x_a^2(S)\|}{\|x_a^1(0) - x_a^2(0)\|} \tag{4.15}$$

where $\|\cdot\|$ is usual L_2 norm

$$\|x_a\| = \sqrt{\sum_{a=0}^{7} (x_a)^2}. \tag{4.16}$$

In this way we calculate the Lyapunov exponent for all p_1 and p_2 and plot λ as function of these variables. Main program including the calculation of the Lyapunov exponents is shown in the listing 4.5.

Listing 4.5: Main program and Lyapunov exponents

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include "ode.h"
#include "connection.h"

#define PI 3.14159265358

#define debugging true

using namespace std;


// params = {alpha, b, m1, m2}
// x = {t, r, hphi, z, u0, u1, u2, u3}
void newton (double *f, double *params, double * x) {
    double R = sqrt( 1 + x[1]*x[1] + x[3]*x[3] - 2*x[1]*cos(x[2]) );
    double b = params[1];
    double Rprime = sqrt(b*b+x[1]*x[1]+x[3]*x[3] + 2*b*x[1]*cos(x[2]));
    double kappa = params[2] / (R*R*R);
    double kappaprime = params[3] / (Rprime*Rprime*Rprime);
    // derivatives of coordinates = velocities
    f[0] = 1;
    f[1] = x[5]; f[2] = x[6]; f[3] = x[7];
    // derivatives of velocities = acceleration
    f[4] = 0;
        // field from A-particle
        f[5] = - kappa * (x[1] - cos(x[2])) +
            x[1] * (params[0] + x[6]) * (params[0] + x[6]);
        f[6] = - kappa * sin(x[2]) / x[1] - (2*x[5]/x[1])*(params[0] + x[6]);
        f[7] = - kappa * x[3];
        //field from B-particle
        f[5] += - kappaprime*(x[1] + b*cos(x[2]))
            + x[1]*(params[0] + x[6])*(params[0]+x[6]);
        f[6] += kappaprime*b*sin(x[2])/x[1] - (2*x[5]/x[1])*(params[0] + x[6]);
        f[7] += - kappaprime*x[3];
}


bool finished(double *x, double *fin_params, double *params) {
    bool time = x[0] > fin_params[0];
    bool orbits = (x[2] + params[1]*x[0] - PI)/(2*PI) > fin_params[1];
    if (time) cout << "maximal time exceeded";
    if (orbits) cout << "number of orbits reached";
    return (time||orbits);
```

```
47  }
48
49
50
51  // params = {alpha, b, m1, m2}
52  void write_coordinates(ofstream *f, double *x, double *params) {
53      // corot (1..4)
54      for (int i = 0; i < 4; i++)
55          (*f) << x[i] << " ";
56      // phi = hphi + alpha * t (5)
57      double phi = x[2] + params[0] * x[0];
58      (*f) << phi << " ";
59      // Cartesian (6, 7)
60      (*f) << x[1] * cos(phi) << " " << x[1] * sin(phi);
61
62      (*f) << endl;
63      //
64  }
65
66
67  double distance (double *x1, double *x2, double *params) {
68      double alpha = params[0];
69      double phi = x1[2] + alpha * x1[0];
70      double r = x1[1];
71      double T1 = x1[0];
72      double X1 = r * cos( phi );
73      double Y1 = r * sin( phi );
74      double Z1 = x1[3];
75      phi = x2[2] + alpha * x2[0];
76      r = x2[2];
77      double T2 = x2[0];
78      double X2 = r * cos( phi );
79      double Y2 = r * sin( phi );
80      double Z2 = x2[3];
81      return (T1-T2)*(T1-T2) - (X1-X2)*(X1-X2)
82        - (Y1-Y2)*(Y1-Y2) - (Z1-Z2)*(Z1-Z2);
83  }
84
85  double distance2(double *x1, double *x2) {
86      double d = 0;
87      for (int i = 0; i < 8; i++)
88          d += (x1[i]-x2[i])*(x1[i]-x2[i]);
89      return sqrt(d);
90  }
91
92  double *cartTocorot (double *xCart, double *params) {
93      double *x = new double[8];
94
95      x[0] = xCart[0];
96      x[1] = sqrt( xCart[1]*xCart[1] + xCart[2]*xCart[2]);
97      // corotating phi = arctg(y/x) - alpha*t
98      x[2] = atan(xCart[2]/xCart[1]) - params[0] * xCart[0];
99      x[3] = xCart[3];
100     // four-velocity
101     x[4] = xCart[4];
102     // u^r = (x ur + y uphi)/r
103     x[5] = ( xCart[1]*xCart[5] + xCart[2]*xCart[6] )/x[1];
104     // u^phi = (x u^phi - y u^x)/r^2 - alpha u^t
105     x[6] = ( xCart[1]*xCart[6] - xCart[2]*xCart[5])/
106           (x[1]*x[1]) - params[0] * xCart[4];
107     x[7] = xCart[7];
108     return x;
109 }
```

```
110
111
112 // params = {alpha, b, m1, m2}
113 void simulation(double *x1, double *x2, double *params, double dt, double tmax)
114 {
115     ofstream geof1 ("geodesic1.txt");
116     ofstream geof2 ("geodesic2.txt");
117     double ds1 = dt;
118     double ds2 = dt;
119     bool crash = false;
120     int steps=0;
121
122     while ((x1[0] < tmax)&&(!crash))
123     {
124         rk4(einstein_retadv, params, x1, 8, ds1);
125         rk4(einstein_retadv, params, x2, 8, ds2);
126
127         ds1 = (x1[1]<1.2*params[1])?0.01*dt:dt;
128         ds2 = (x2[1]<1.2*params[1])?0.01*dt:dt;
129
130         normalize(x1, params, false);
131         normalize(x2, params, false);
132         steps++;
133         if ((steps%1000)==0) cout << steps << " ";
134         crash = (steps>20000) || (abs(x1[1])<1E-3) || (abs(x2[1])<1E-3);
135         if (debugging) {
136             write_coordinates(&geof1, x1, params);
137             write_coordinates(&geof2, x2, params);
138         }
139     }
140     if (crash)
141         cout << "Max. time not reached" << endl;
142         cout << "steps = " << steps << ", r1 = "
143              << x1[1] << ", r2 = " << x2[1] << endl;
144     geof1.close();
145     geof2.close();
146 }
147
148
149 // params = {alpha, b, m1, m2}
150 int main()
151 {
152     //
153
154     /* cycle though the plane in phase space / parametric space */
155
156     ofstream lyapunf ("lyapunov.txt");
157
158     double b = 2;
159     double alpha = (b==1)?0.01:equilib_alpha(b);
160
161     double m1 = equilib_mass1(b, alpha);
162     double m2 = equilib_mass2(b, alpha);
163     double params[4] = {alpha, b, m1, m2};
164     cout << "alpha = " << alpha << ", b = " << b << ",
165          m1 = " << m1 << ", m2 = " << m2 << endl;
166     cout << "centre of mass = " << mass_center(b, alpha)
167          << " (should be zero) " << endl;
168     double torbit = 2*PI/alpha;
169     double ds = torbit/100;
170     double tmax = 50*torbit;
171
172
```

```
    double p1_min = -0.02;
    double p1_max = 0.02;
    double p2_min = 0.01;
    double p2_max = 0.05;
    double dp1 = 0.001;
    double dp2 = 0.001;
    float N = (p1_max-p1_min) * (p2_max-p2_min) / (dp1*dp2);
    int Step = 0;

    cout << "number of gedoesics = " << N << endl;
    cin.ignore(1);

    for (double p1 = p1_min; p1 <= p1_max; p1 += dp1){
        for (double p2 = p2_min; p2 <= p2_max; p2 += dp2)
        {
            double x1Cart[8] = {0, 150, 0, 0, /**/ 1, p1, p2, 0};
            double x2Cart[8] = {0, 150.1, 0, 0, /**/ 1, p1, p2, 0};
            double *x1 = cartTocorot(x1Cart, params);
            double *x2 = cartTocorot(x2Cart, params);
            cout << norm(x1, params)<<endl;
            normalize(x1, params, false);
            normalize(x2, params, false);


            cout << "Starting simulation (" << ++Step << "/" << N << ") at" << endl;


            for (int i=0; i < 8; i++)
                cout << "x[" << i << "] = " << x1[i] << ", ";
            cout << endl;

            double u0 = distance2(x1, x2);

            cout << "u0 = " << u0 << endl;

            simulation(x1, x2, params, ds, tmax);


            double u = distance2(x1, x2);
            cout << "u = " << u << endl;
            double l = (1/x1[0]) * log( u / u0 );
            cout << "lyap = " << l << endl;
            lyapunf << p1 << " " << p2 << " " << l << endl;
            if (debugging)
                cin.ignore(1);
            delete[] x1; delete[] x2;
        }
        lyapunf << endl;
    }

    lyapunf << "finish";
    lyapunf.close();

    return 0;
}
```

Now we can turn to the main goal of the thesis – investigation of geodesics in helically symmetric spacetimes.

5. Results

In this chapter we finally present results obtained by the program developed in previous chapters. Since there are many possible combinations of initial conditions and many possible choices of the parameters of the system, we present only selected examples demonstrating the nature of geodesics in the spacetime under consideration. However, a deeper analysis is highly desirable. In particular, we did not solve the question whether geodesics corresponding to periodic motion exist.

5.1 Equilibrium condition

In chapter 4 we have have derived conditions ensuring that the binary system be in equilibrium. We have shown that there are two distinct situations. If the radii of orbits of both particles coincide, which in our units corresponds to the case $b = 1$, angular velocity α can be chosen arbitrarily and the equilibrium value of mass can be computed from (4.8). However, if the radii are different, $b \neq 1$, there are two additional constraints for the parameters of the system. Mass of A-particle is still given by (4.8), while the mass of B-particle is given by (4.9) and both masses must satisfy the "center-of-mass condition" (4.10). Thus, once the value of b is chosen, values of angular velocity and masses follow from conditions just enumerated. These conditions have been solved numerically and visualized in figures 5.1.

In figure 5.1 we plot dependence of α on parameter b with the point $b = 1$ excluded (because the solution is not unique at that point, see below). An interesting feature of figure 5.1 is that the solution in fact does not exist for all values of b and all solutions lie in the interval

$$b \in (0,443; \quad 2,257). \tag{5.1}$$

For b smaller than $b_{\min} = 0,443$ and for b larger than $b_{\max} = 2,257$ there is no such value of α for which the system of both particles can stay in equilibrium. It is a clear difference from the Newtonian case.

Recall that in our units the radius of A-particle's orbit a is equal to one and radius b of B-particle is given in multiples of a. If we reverse the rôles of A- and B-particle, so that a will be given in multiples of b, we will obtain an interval $(b_{\max}^{-1}, b_{\min}^{-1})$. However, physical results cannot depend on our choice which particle is A-particle and which one is B-particle. Thus, numerical values of $a_{\min}$ and $a_{\max}$ must be the same as values $b_{\min}$ and $b_{\max}$. This is indeed a case, because we have

$$b_{\min} \; = \; \frac{1}{b_{\max}}.$$

This is an important test of consistency of the theory and of the numerics. In figure 5.2 we plot dependence of equilibrium masses on the value of b.

Finally, in figure 5.3 we discuss the case $b = 1$ when the orbits of both particles are the same. Then we can choose an arbitrary value of α and calculate the equilibrium value of mass m. Surpising feature of this dependence is that for each value of m there are in fact *two values* of α such that the system is in equilibrium.

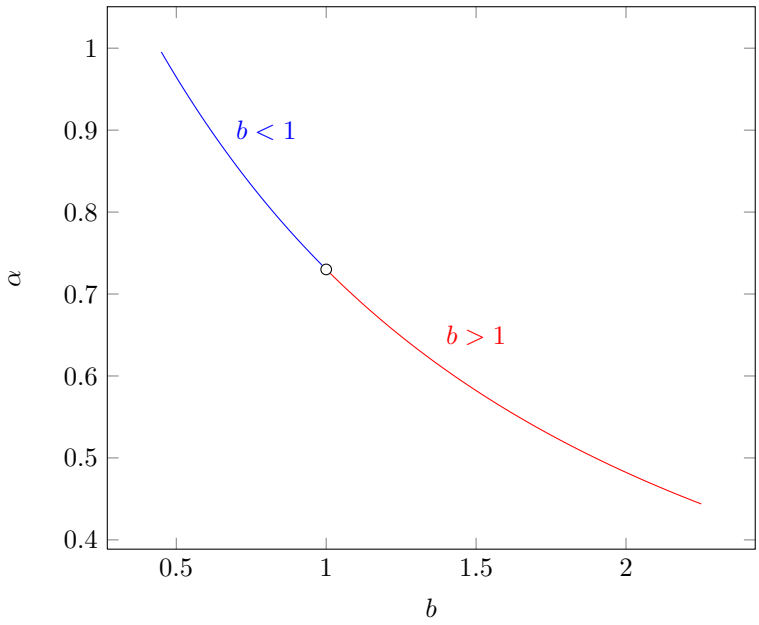


Figure 5.1: Equilibrium value of angular velocity α as a function of parameter b with point $b = 1$ excluded.

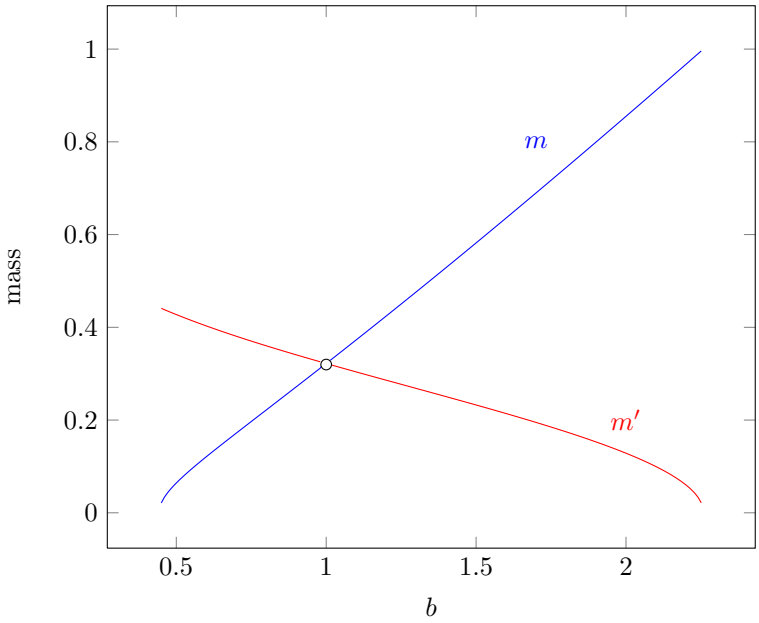


Figure 5.2: Equilibrium values of masses of A-particle m and B-particle m' as functions of parameter b with point $b = 1$ excluded.

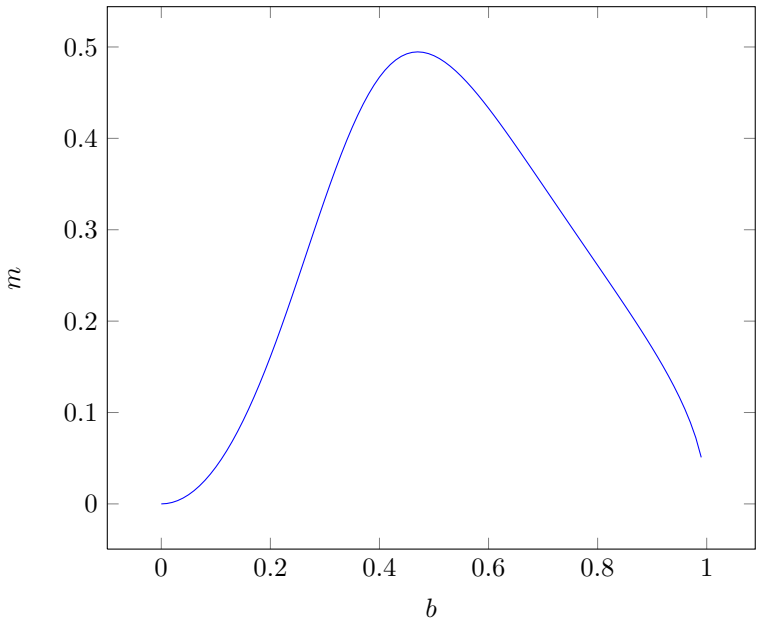


Figure 5.3: Equilibrium values of mass of both particles as a function of α for $b = 1$.

5.2 Selected geodesics

For convenience, we impose initial conditions in the Cartesian coordinates and use function cartToCorot, see listing 4.5, to transform to corotating coordinates in which the calculations are performed.

As a first example we consider particle with initial conditions

$$x^\mu(0) \;=\; (0, x, 0, 0), \quad u^\mu(0) \;=\; \gamma(1, 0, v_y, 0),$$

so that initially the particle lies on the $x-$axis at distance x from the origin and has only vertical $y-$component of three-velocity. The Lorentz factor γ is computed from x and v_y so as to satisfy $u^\mu u_\mu = 1$. Four cases corresponding to different choices of x and v_y are plotted in figures 5.4 and 5.5. In figures a)–d) we can see that for velocities $v_y > 10^{-8}$ the trajectories always escape to infinity. Trajectories for $v_y = 10^{-8}$ exhibit a transient phase of apparently chaotic motion, but finally escape to infinity anyway. This behaviour have been observed for many other choices of x and v_y. In figure 5.6 we plotted the trajectory of particle with initial v_x component of the velocity. For $v_x = v_y = 10^{-8}$ the trajectory again exhibits a transient phase and escapes afterwards.

Several other examples are plotted in figures 5.7–5.9. Initial conditions for these cases are written down under each figure or series of figures. Interesting feature of figure 5.7 is that the distance between trajectories decreases as these trajectories approach binary system. Figure 5.8b suggests that test particles can move along closed circular trajectories. However, such solutions are, as expected, unstable: figure 5.8b shows two geodesics with almost same initial conditions but one geodesic is attracted to binary system while the second one follows apparently closed trajectory.

5.3 Phase portraits

More systematic approach to investigation of geodesics is using a kind of phase portrait known from the theory of dynamical systems and Hamiltonian dynamics. The idea is to visualise the character of a family of geodesics differing by values of some parameters. An appropriate measure of the behaviour of geodesics is the Lyapunov exponent introduced in section 4.5. Hence, in the following we always fix all but two parameters determining the geodesic and vary remaining two parameters called p_1 and p_2. In such a way we can assign a number $\lambda(p_1, p_2)$ to each point of (p_1, p_2)-plane.

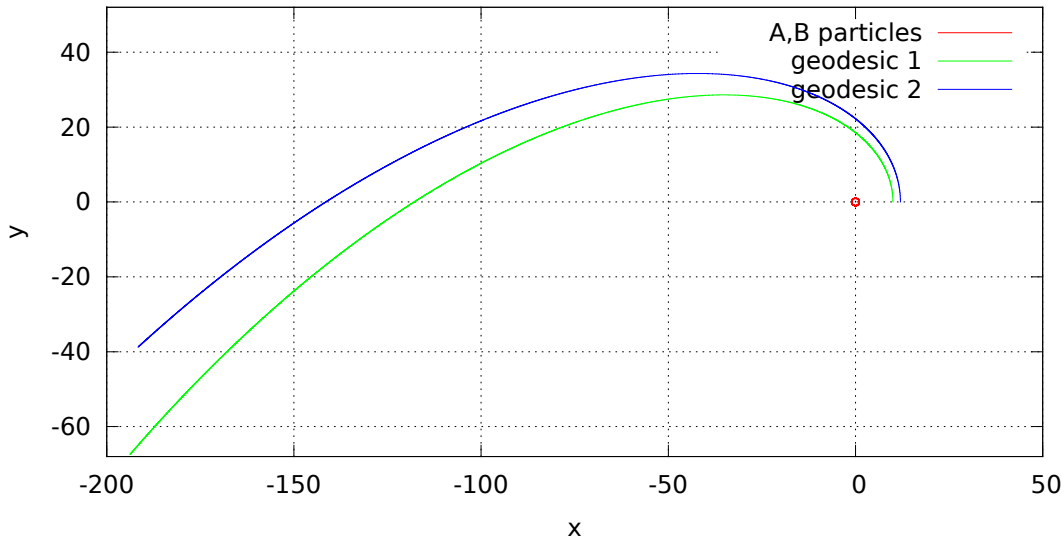
In figure 5.10 we use initial conditions for two close geodesics in the form

$$x_1^\mu(0) \;=\; (0, x_0, 0, 0), \quad u_1^\mu(0) = \gamma(1, 0, p_1, p_2)$$

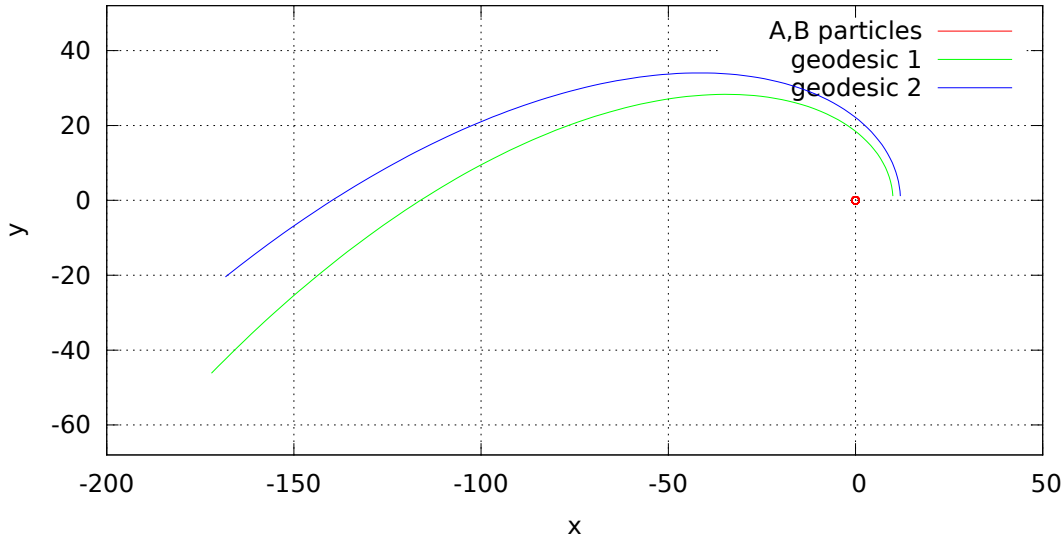
for the first geodesic and

$$x_2^\mu = (0, x_0', 0, 0), \quad u_2^\mu(0) = \gamma(1, 0, p_1, p_2)$$

for the second one. In figure 5.10a) we set $x_0 = 10$ and $x_0' = 10.1$. Then we vary p_1 and p_2, so for each (p_1, p_2) there is one geodesic starting from point $x_1^\mu(0)$ at velocity u_1^μ and a nearby geodesic starting from $x_2^\mu(0)$ at the same velocity $u_\mu^2(0)$ where velocities $u_1^\mu(0) = u_2^\mu(0)$ depend on p_1 and p_2. We solve geodesic equation for both geodesics, i.e. we let the solution to evolve from initial conditions. After sufficiently long time we compare actual distance of both solutions with the initial distance and compute the Lyapunov exponent according to (4.15). Repeating this procedure for each point of $(p_1, p_2)-$plane we obtain function $\lambda(p_1, p_2)$ which is then visualised as a density plot, i.e. each point of $(p_1, p_2)-$plane is coloured according to value of λ. For this purpose we used GNUplot software.

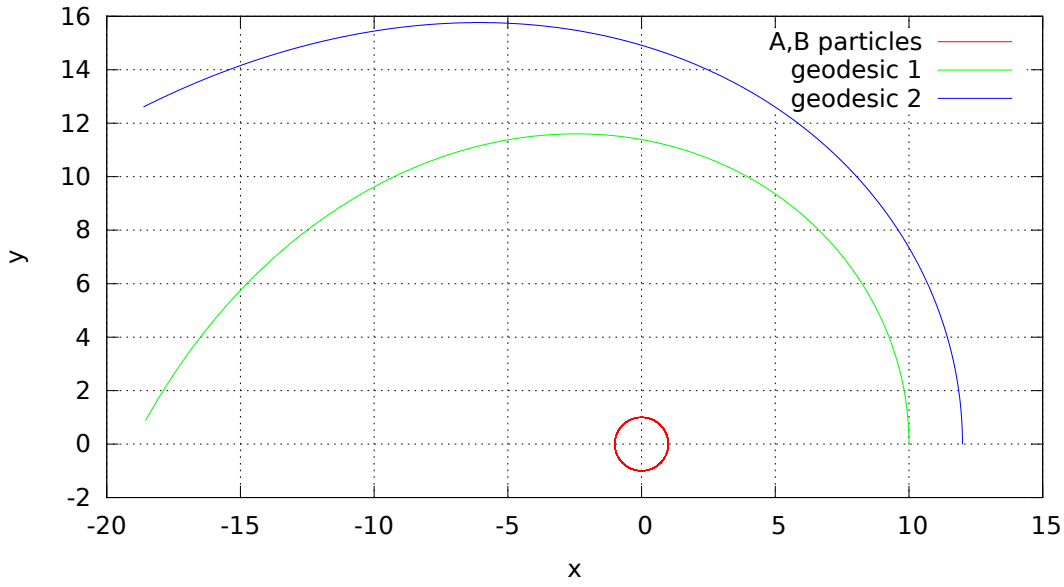


a) $v_y = 10^{-5}$

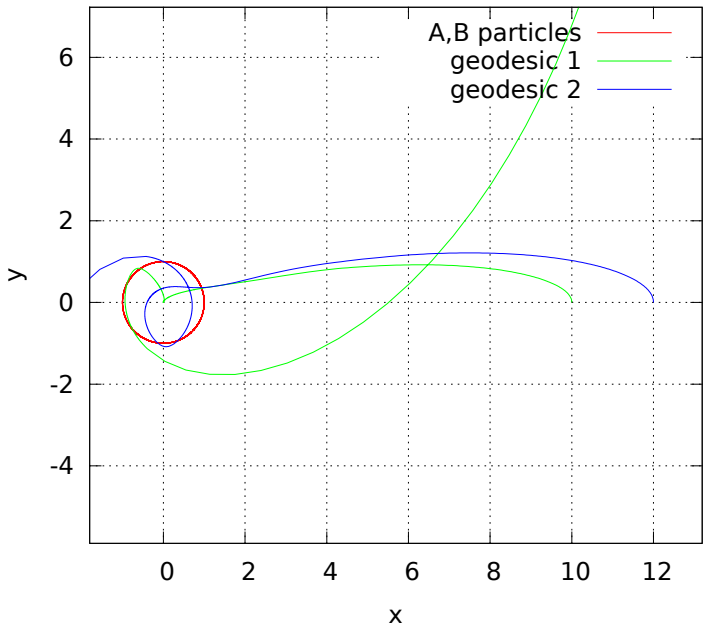


b) $v_y = 10^{-6}$

Figure 5.4: Comparison of two geodesics starting from separate points at the same velocity. Angular velocity $\alpha = 10^{-7}$, initial conditions in Cartesian coordinates: $x_1^\mu(0) = (0, 10, 0, 0)$, $x_2^\mu(0) = (0, 12, 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0, v_y, 0)$



c) $v_y = 10^{-7}$



d) $v_y = 10^{-8}$

Figure 5.5: Comparison of two geodesics starting from separate points at the same velocity. Angular velocity $\alpha = 10^{-7}$, initial conditions in Cartesian coordinates: $x_1^\mu(0) = (0, 10, 0, 0)$, $x_2^\mu(0) = (0, 12, 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0, v_y, 0)$

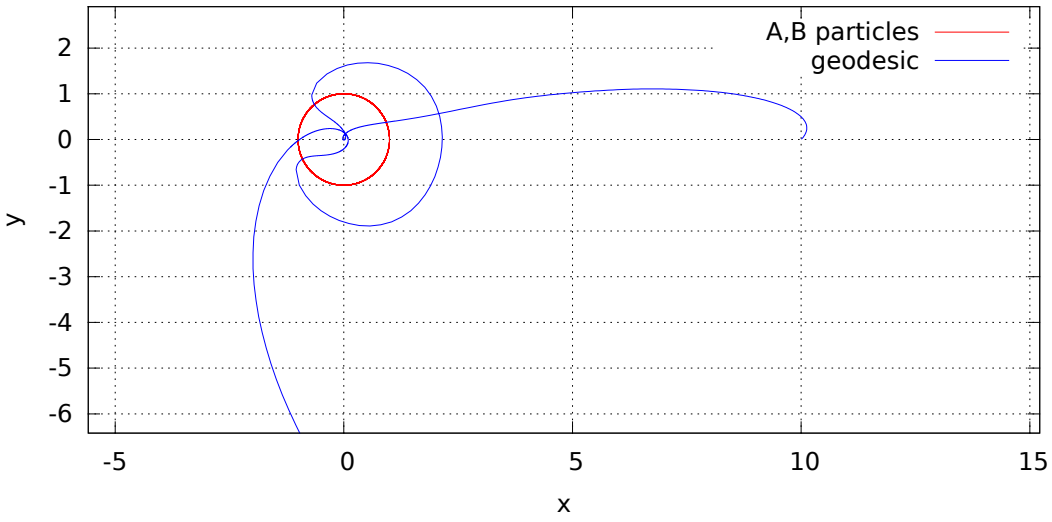
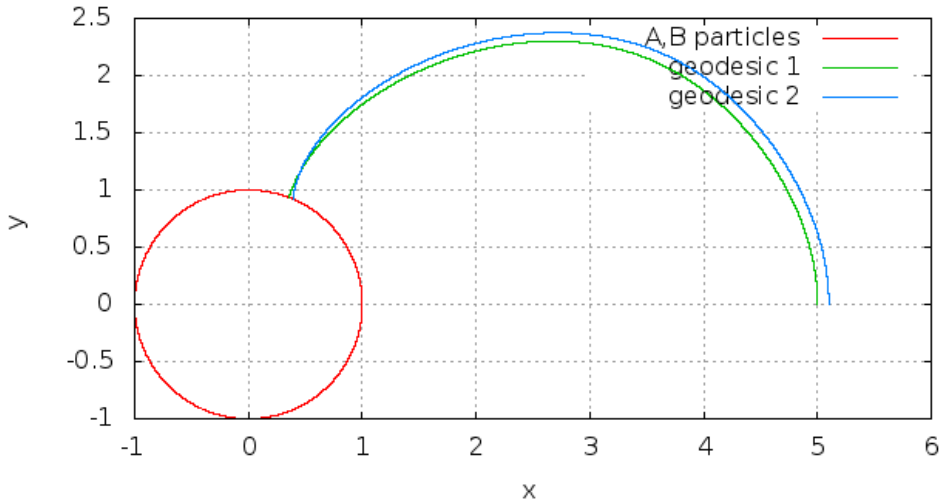
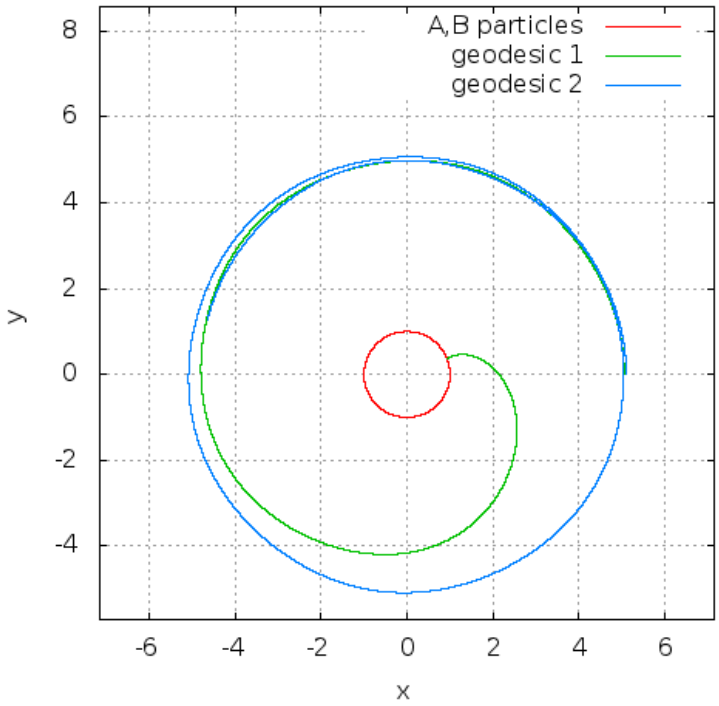


Figure 5.6: Angular velocity $\alpha = 10^{-7}$, initial conditions $x^\mu(0) = (0, 10, 0, 0)$, $u^\mu(0) = \gamma(1, 10^{-8}, 10^{-8}, 0)$.

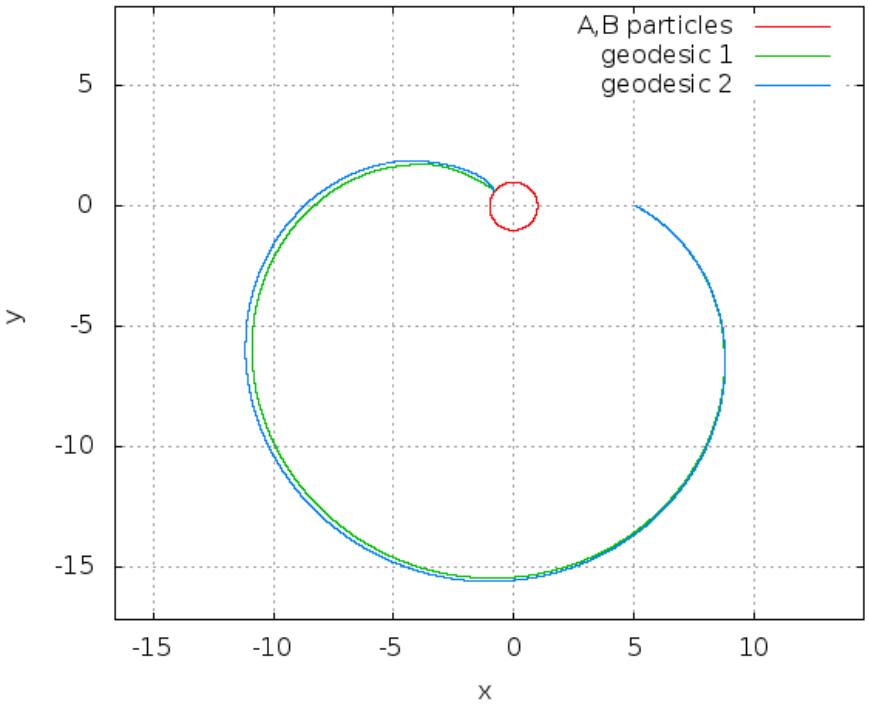


a) $x_1^\mu(0) = (0, 5, 0, 0)$, $x_2^\mu(0) = (0, 5.1, 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0, 0.16, 0)$

Figure 5.7: Comparison of two geodesics starting from separate points at the same velocity. Angular velocity $\alpha = 0, 22$, initial conditions are given in the Cartesian coordinates.



b) $x_1^\mu(0) = (0, 5.05, 0, 0)$, $x_2^\mu(0) = (0, 5.1, 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0, 0.3, 0)$



c) $x_1^\mu(0) = (0, 5.09, 0, 0)$, $x_2^\mu(0) = (0, 5.1, 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0.2, -0.1, 0)$

Figure 5.8: Comparison of two geodesics starting from separate points at the same velocity. Angular velocity $\alpha = 0, 22$, initial conditions are given in the Cartesian coordinates.

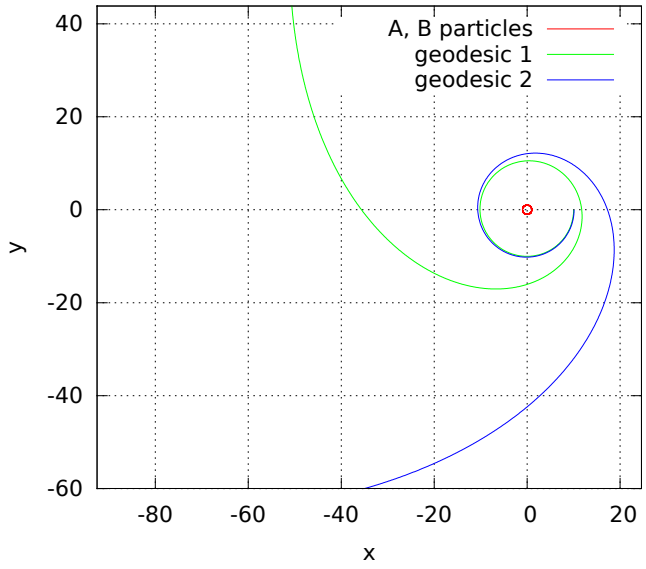
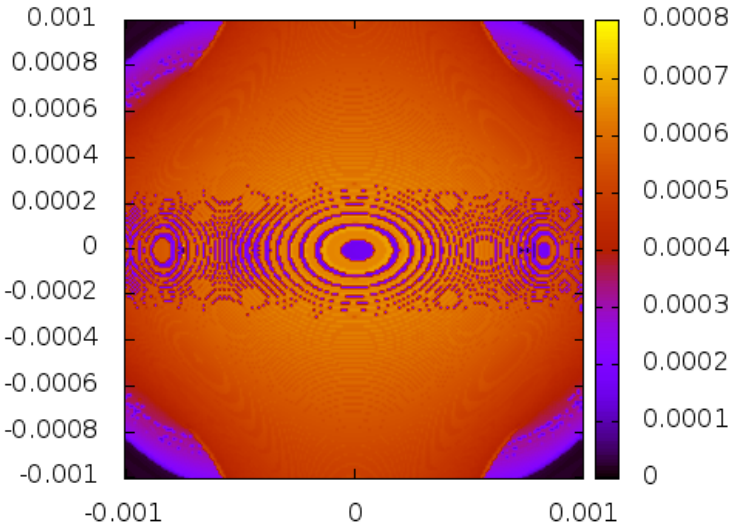
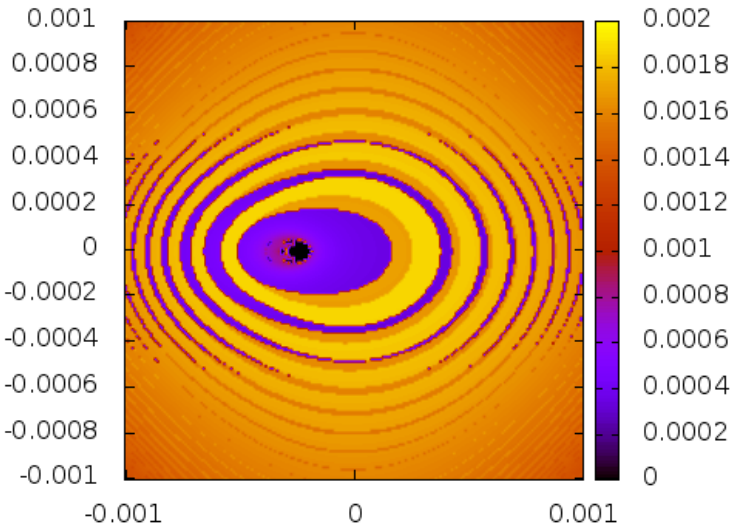


Figure 5.9: Comparison of two geodesics starting from separate points at the same velocity. Angular velocity $\alpha = 0,22$, initial conditions in Cartesian coordinates: $x_1^\mu(0) = (0, 10, 0, 0)$, $x_2^\mu(0) = (0, 10.1, 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0, -0.009, 0)$

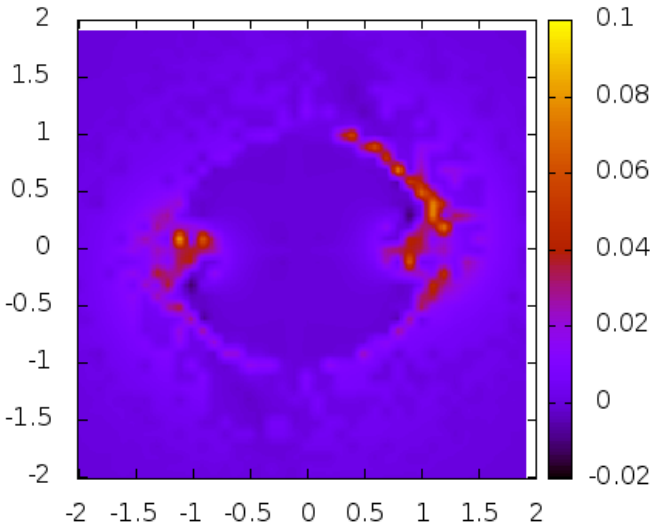


a) $x_0 = 10$, $x_0' = 10.1$

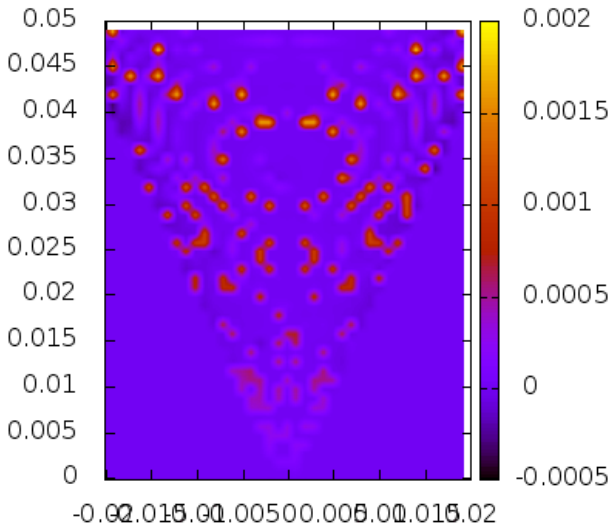


b) $x_0 = 5$, $x_0' = 5.1$

Figure 5.10: Initial conditions $x_1^\mu(0) = (0, x_0, 0, 0)$, $x_2^\mu = (0, x_0', 0, 0)$, $u_1^\mu(0) = u_2^\mu(0) = \gamma(1, 0, p_1, p_2)$, where $p_1 \in (-10^{-3}, 10^{-3})$, $p_2 \in (-10^{-5}, 10^{-5})$.

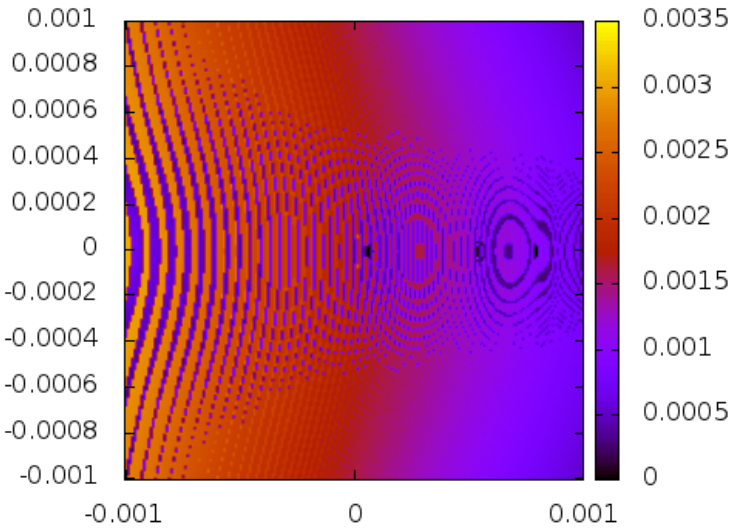


a) Initial conditions $x_1^\mu(0) = (0, p_1, p_2, 0)$, $x_2^\mu = (0, p_1 + 0.1, p_2, 0)$, $u_1^\mu(0) = u_2^\mu(0) = (1, p_1/1000, p_2/1000, 0)$, where $p_1 \in (-2, 2)$, $p_2 \in (-2, 2)$

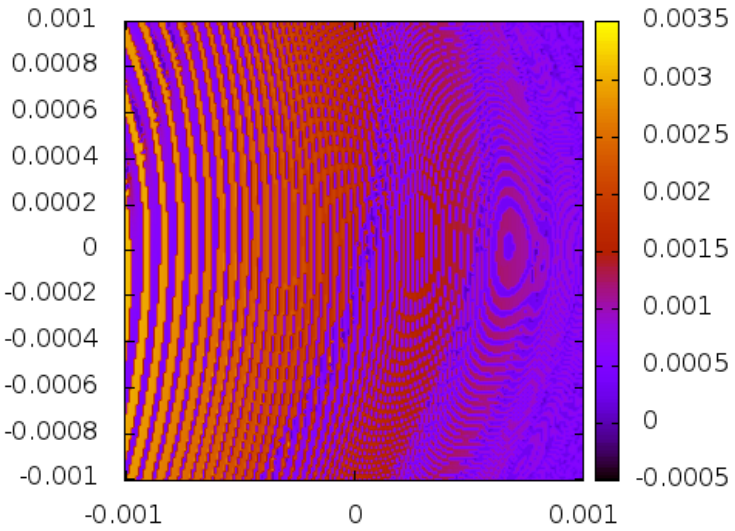


b) Initial conditions $x_1^\mu(0) = (0, 0, -100, 0)$, $x_2^\mu = (0, 0, -99, 0)$, $u_1^\mu(0) = u_2^\mu(0) = (1, p_1, p2, 0)$, where $p_1 \in (-0.02, 0.02)$, $p_2 \in (0, 0.05)$

Figure 5.11: Another two examples.

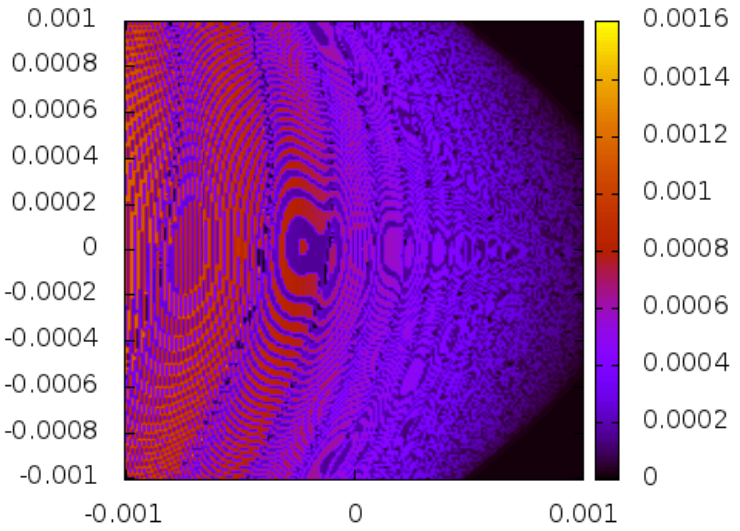


a) Initial conditions $x_1^\mu(0) = (0, 5, 0, 0)$, $x_2^\mu = (0, 5.1, 0, 0)$,
$u_1^\mu(0) = u_2^\mu(0) = (1, p_1, p_2, 0)$, where $p_1 \in (-0.01, 0.01)$, $p_2 \in (-0.01, 0.01)$

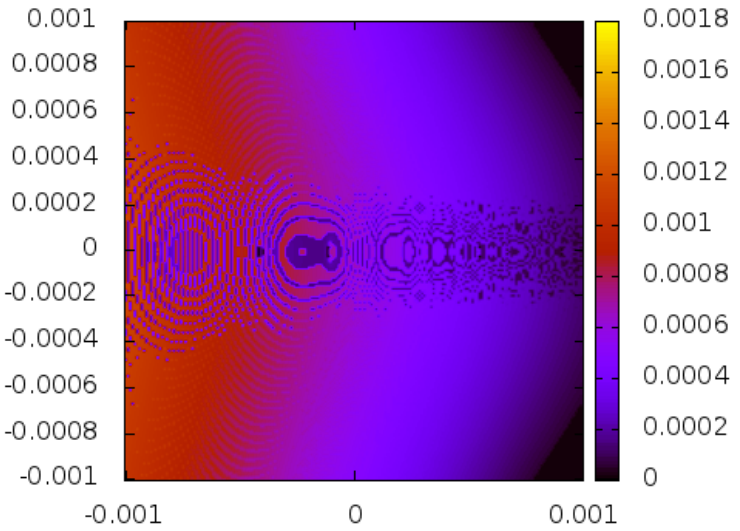


b) Initial conditions $x_1^\mu(0) = (0, 5, 0, 0)$, $x_2^\mu = (0, 5.1, 0, 0)$,
$u_1^\mu(0) = u_2^\mu(0) = (1, p_1, 0, p_2)$, where $p_1 \in (-0.01, 0.01)$, $p_2 \in (-0.01, 0.01)$

Figure 5.12: Particle released form $x-$axis. **a)** Phase portrait in (v_x, v_y) plane. **b)** Phase portrait in (v_x, v_z) plane.



a) Initial conditions $x_1^\mu(0) = (0, 10, 0, 0)$, $x_2^\mu = (0, 10.1, 0, 0)$,
$u_1^\mu(0) = u_2^\mu(0) = (1, p_1, p_2, 0)$, where $p_1 \in (-0.01, 0.01)$, $p_2 \in (-0.01, 0.01)$



b) Initial conditions $x_1^\mu(0) = (0, 10, 0, 0)$, $x_2^\mu = (0, 10.1, 0, 0)$,
$u_1^\mu(0) = u_2^\mu(0) = (1, p_1, 0, p_2)$, where $p_1 \in (-0.01, 0.01)$, $p_2 \in (-0.01, 0.01)$

Figure 5.13: Particle released form $x-$axis. **a)** Phase portrait in (v_x, v_y) plane. **b)** Phase portrait in (v_x, v_z) plane.

5.4 Causal structure

In this last section we present some results concerning the *causal structure* of the spacetime under consideration. By causal structure we mean the properties of light cones emerging from different spacetime points. In the following we always prescribe initial conditions in the form (we are still giving initial conditions in the Cartesian coordinates)

$$x^\mu(0) \;=\; (0, x_0, y_0, 0), \quad u^\mu(0) \;=\; (1, u^1, u^2, 0). \tag{5.2}$$

Initial four-velocity is chosen to be null, i.e.

$$1 \;-\; (u^1)^2 \;-\; (u^2)^2 \;=\; 0.$$

Null cone at given spacetime point is a two-dimensional projective space. By the choice $u^3 = 0$ we reduce this space to one-dimensional space so that null geodesics can be parametrized by single variable ϕ via

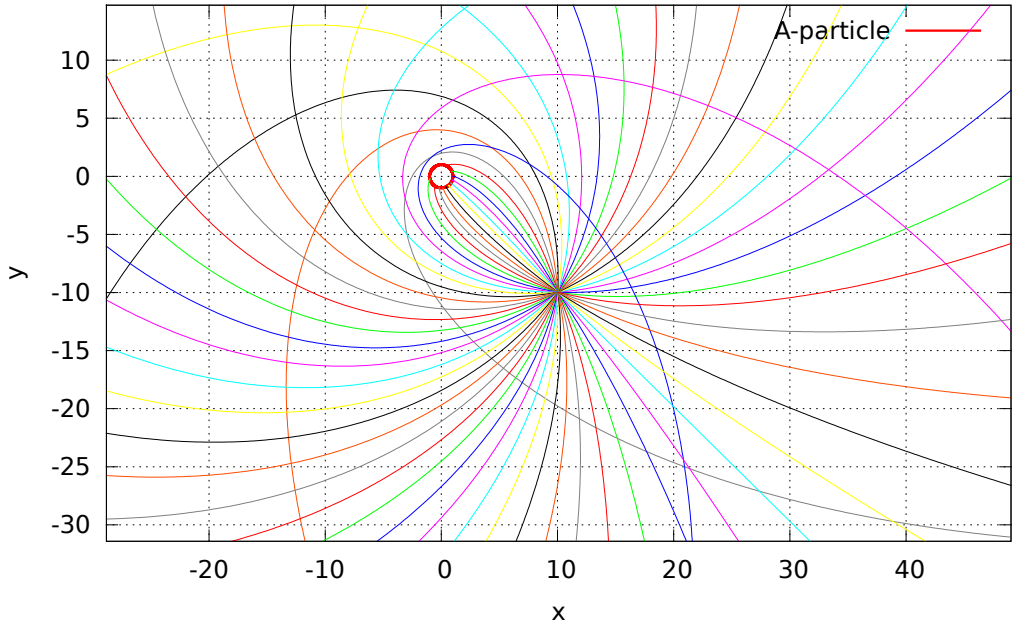
$$u^\mu(0) \;=\; (1, \cos\phi, \sin\phi, 0), \quad \phi \in (0, 2\pi).$$

In figures 5.14–5.17 we choose diferent initial points (x_0, y_0) and plot both

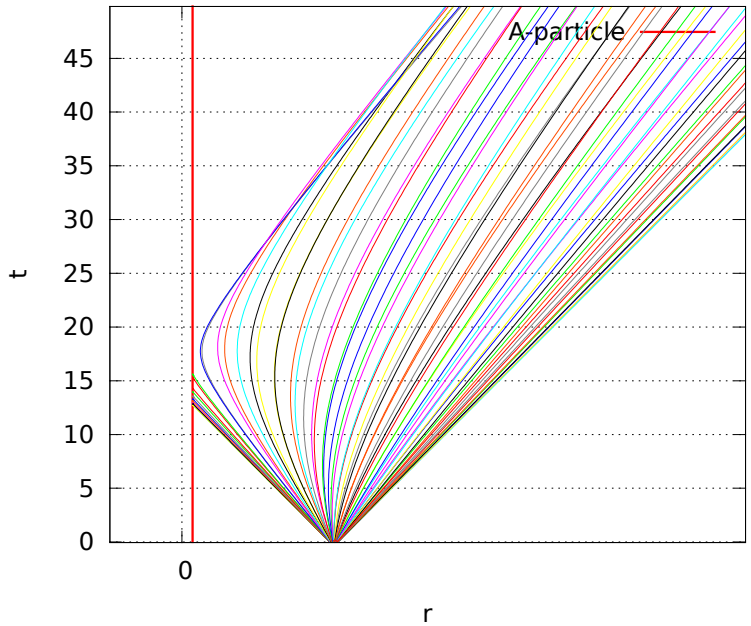
- spatial trajectories traced by null geodesics starting from initial point,
- spacetime diagrams visualizing null geodesics – world lines of light rays forming the light cone.

Notice that all curves plotted are indeed null geodesics although the figure could suggest that interior of the cone is filled with timelike geodesics. Different curves, however, correspond to different values of angular coordinate ϕ and form two-dimensional surface. For $\phi = 0$ we obtain a null geodesic lying in the plane (t, r); this is the "most-right" geodesic in the diagram. For $\phi \in (0, \pi)$, geodesics lie "behind" the $(t, r)-$plane, for $\phi = \pi$ we get the "most-left" geodesic. For $\phi \in (\pi, 2\pi)$, the geodesics lie in front of the $(t, r)-$plane and finally for $\phi = 2\pi$ we obtain the most-right geodesic again.

Since all light cones are very similar, we briefly discuss only the first one, figure 5.14b. Obvious feature of this diagram is that there is a region which cannot be reached from the initial point by a causal curve (null or timelike). For ϕ close to zero we have almost straight null geodesics as in the Minkowski spacetime. However, for increasing ϕ they become more bent as to escape to infinity. Geodesics in the left part of diagram are trapped by the binary system. Simulation had to be stopped at these points because of singular nature of point particles. Remaining null cones plotted in figures 5.16–5.17 exhibit similar behaviour.

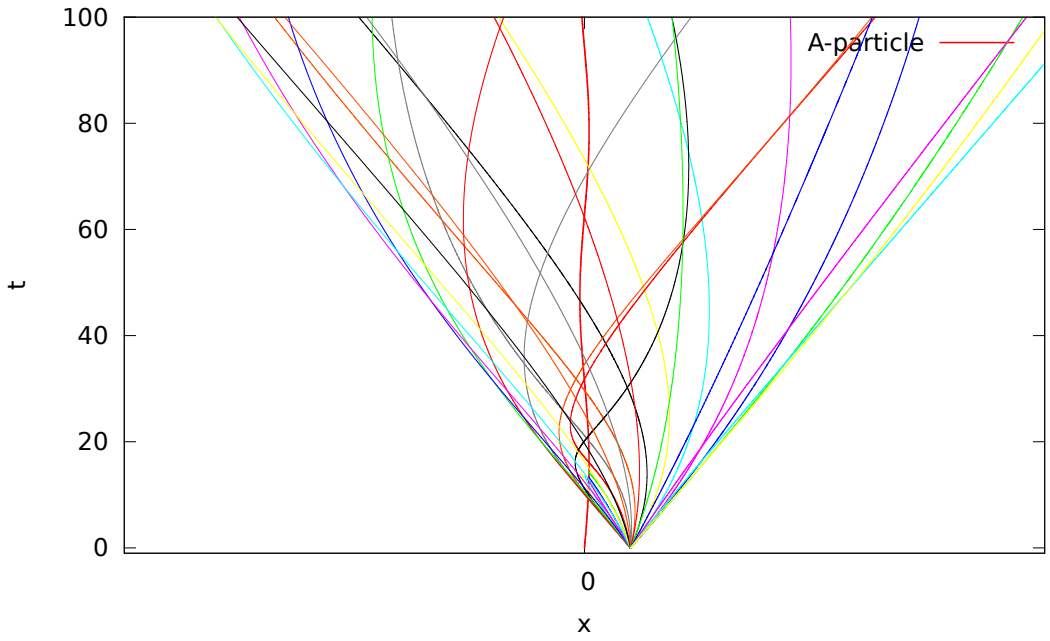


a) Trajectories of light rays forming null cone. Red circle represents trajectory of A-particle.

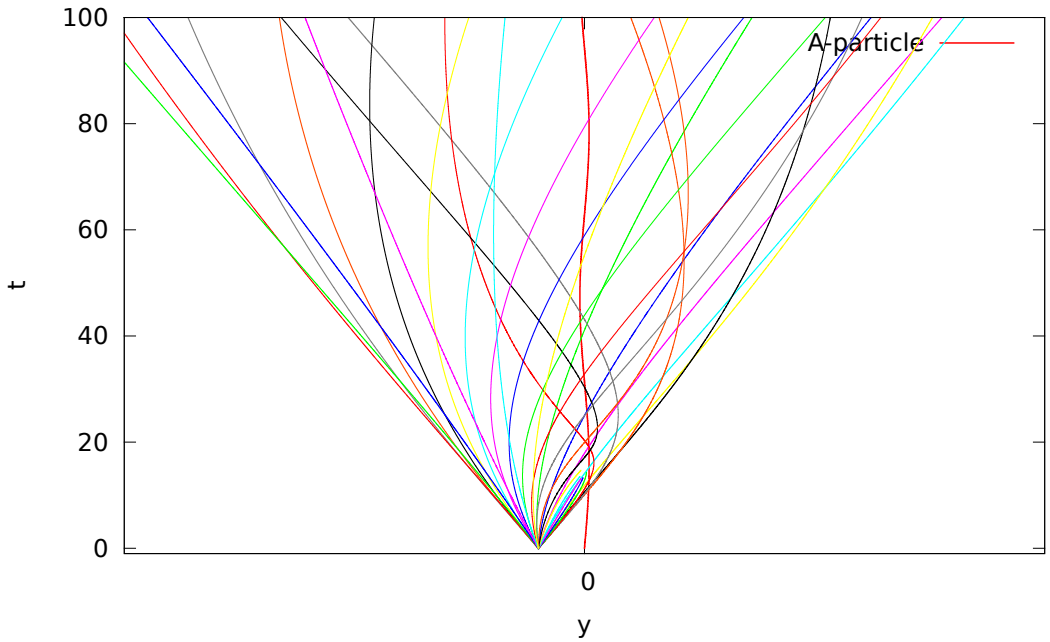


b) Null cone at point $(0, 10, -10, 0)$ in the plane (t, r). Red thick line represents worldline of A-particle.

Figure 5.14: All geodesics start at point $(0, 10, -10, 0)$ for $\alpha = 0.007$. Initial conditions have been chosen in the form $x^\mu(0) = (0, 10, -10, 0)$, $u^\mu(0) = (1, \cos\phi, \sin\phi, 0)$ so that for each $\phi \in (0, 2\pi)$ the four-velocity is a null vector, $u^\mu u_\mu = 0$.

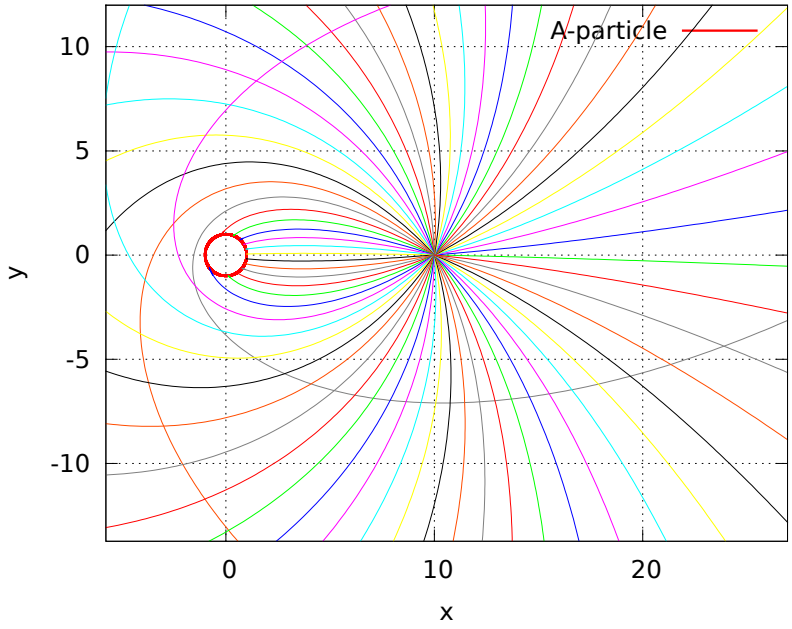


a) Null cone in plane (t, x)

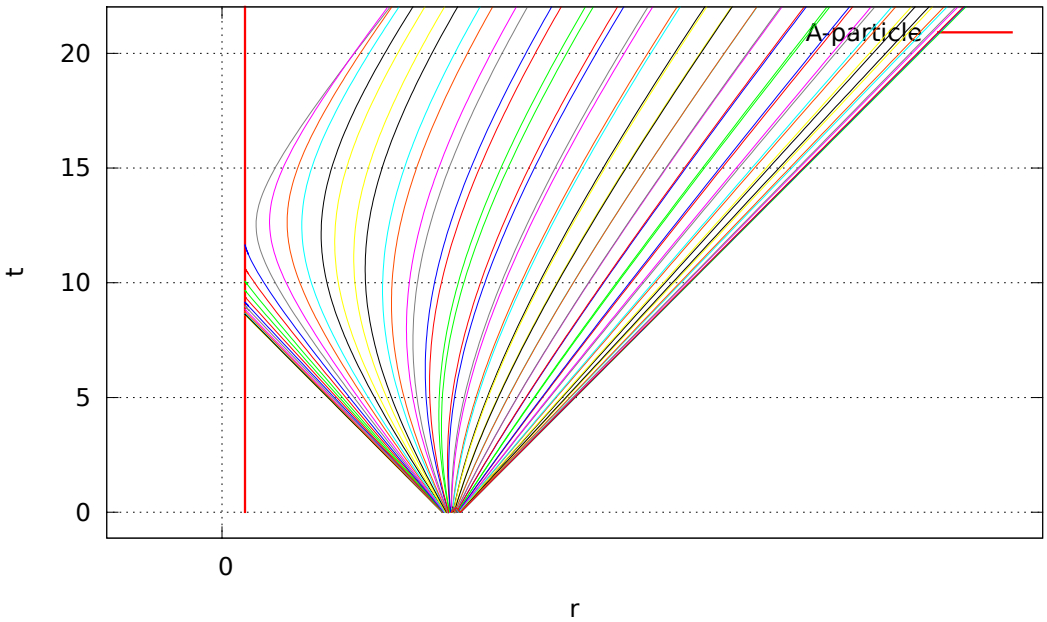


b) Null cone in plane (t, y)

Figure 5.15: All geodesics start at point $(0, 10, -10, 0)$ for $\alpha = 0.007$. Initial conditions have been chosen in the form $x^\mu(0) = (0, 10, -10, 0)$, $u^\mu(0) = (1, \cos\phi, \sin\phi, 0)$ so that for each $\phi \in (0, 2\pi)$ the four-velocity is a null vector, $u^\mu u_\mu = 0$.

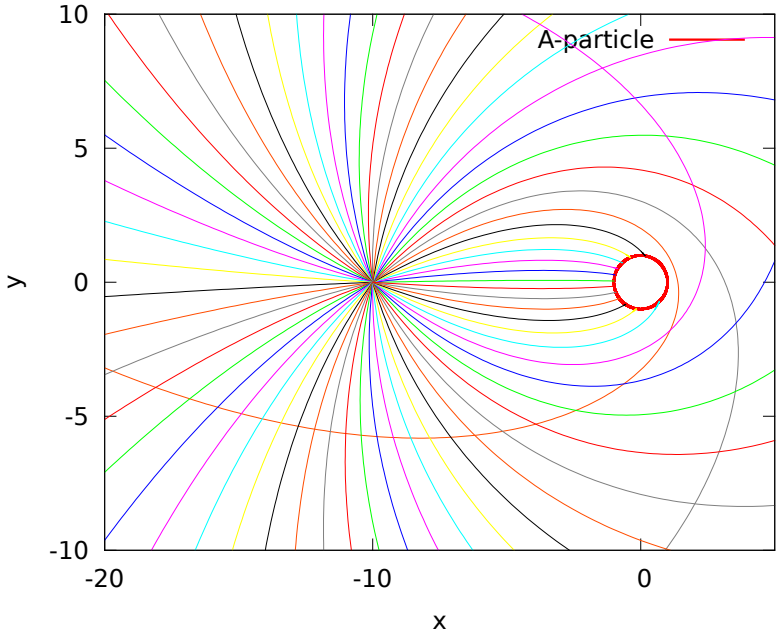


a) Trajectories of light rays forming null cone. Red circle represents trajectory of A-particle.

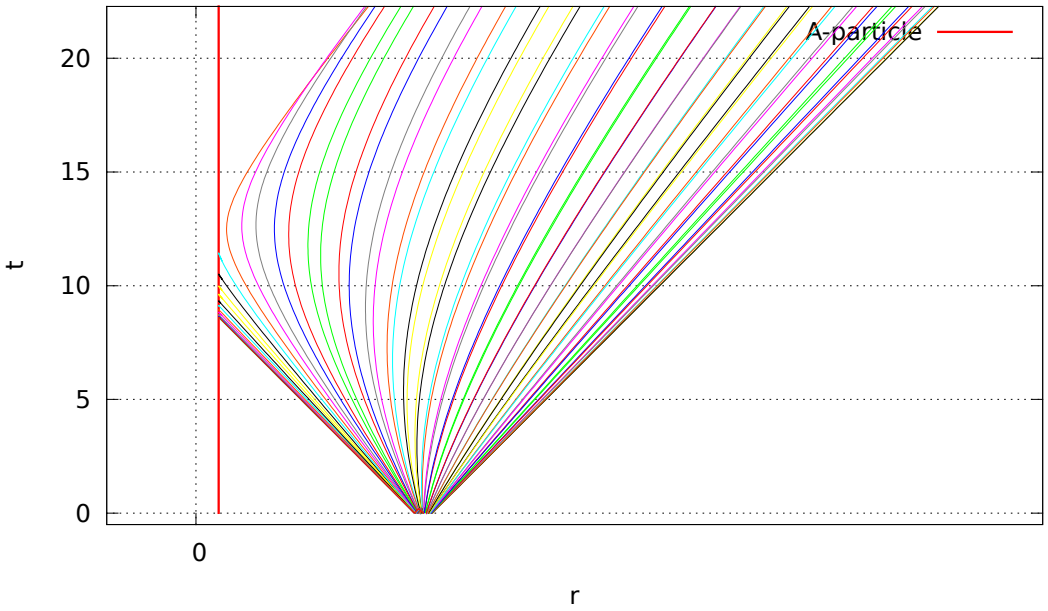


b) Null cone at point $(0, 10, 0, 0)$. Red thick line represents worldline of A-particle.

Figure 5.16: All geodesics start at point $(0, 10, 0, 0)$ for $\alpha = 0.001$. Initial conditions have been chosen in the form $x^\mu(0) = (0, 10, 0, 0)$, $u^\mu(0) = (1, \cos\phi, \sin\phi, 0)$ so that for each $\phi \in (0, 2\pi)$ the four-velocity is a null vector, $u^\mu u_\mu = 0$.



a) Trajectories of light rays forming null cone. Red circle represents trajectory of A-particle.



b) Null cone at point $(0, -10, 0, 0)$. Red thick line represents worldline of A-particle.

Figure 5.17: All geodesics start at point $(0, 10, 0, 0)$ for $\alpha = 0.001$. Initial conditions have been chosen in the form $x^\mu(0) = (0, -110, 0, 0)$, $u^\mu(0) = (1, \cos\phi, \sin\phi, 0)$ so that for each $\phi \in (0, 2\pi)$ the four-velocity is a null vector, $u^\mu u_\mu = 0$.

6. Epilogue

This thesis is devoted to the study of helical symmetry in linearized Einstein's gravity. Our goal was to investigate the geodesic equation of the test particle on the background of binary system consisting of two particles (called A and B particles) in uniform circular motion at constant angular velocity. This work is an extension of paper [2] which is still under preparation. In this paper, conditions of equilibrium for binary system are found and asymptotic properties (peeling) of gravitational field produced by this system are analyzed. In addition, as a by-product, the Christoffel symbols representing the connection derived from the metric have been found; however, these calculations are not present in [2]. Since the supervisor of the author of the thesis is a co-author of paper [2], the motivation for this thesis was to use these Christoffel symbols and investigate the nature of geodesics in helically symmetric spacetime in the framework of bachelor thesis. Hence, in this thesis, we use many results obtained in [2] and repeat some calculations which will appear therein.

The thesis is divided in five chapters; let us recapitulate their content. In chapter 1 we introduced the notion of dynamical system as a useful tool to analyze systems of ordinary differential equations(ODE's). We explained how a geodesic equations can be brought into a set of first-order differential equations. Then we described numerical methods available for solving ODE's: Euler method, Runge-Kutta methods of second and fourth order and the adaptive-Runge-Kutta method. Finally we presented computer codes implementing these methods in C++ language.

In chapter 2 we concentrated on the Newtonian solution representing binary system. We have derived equations of motion, introduced appropriate units and discussed the conditions of the Newtonian equilibrium. Next we used the Newtonian theory to test and justify numerical methods and their implementation introduced in chapter 1. We have compared the accuracy of these methods and found their limitations by studying the orbits of increasing eccentricity. We have concluded that Runge-Kutta method of fourth order (or its adaptive version) are sufficiently accurate for our purposes. We have also discussed possible chaotic nature of Newtonian trajectories in the vicinity of binary system.

The next chapter 3 has been written with the help of the supervisor. Here we essentially repeat the calculation [2] in order to clarify the character of the problem and the idea of its solution. Physical difference between advanced and retarded solutions is emphasized and it is concluded that only symmetric combination of both can yield a helically symmetric configuration. Roughly speaking, in the Newtonian case, force by which A-particle and B-particle interact with each other, is always radial, i.e. orthogonal to the trajectory. In the Einsteinian case, however, this "force" has also an angular part caused by retardation effects. Taking a symmetric combination of retarded and advanced solution automatically eliminates angular part of the force which is a necessary condition of equilibrium. Sufficient condition is then obtained by the requirement that the second derivatives of all coordinates (given by the geodesic equation) vanish. Solution obtained in chapter 3 involves some non-elementary functions which are given implicitly by the set of transcendent equations and we illustrate their geometrical meaning.

In chapter 4 we employ standard method of secants to solve transcendent equations mentioned above and present the computer code implementing this method. Then we present complete code for solving the geodesic equation in helically symmetric spacetime. Full form of the Christoffel symbols is included in the code. Since the Christoffel symbols are many-pages long expressions, they have been derived by the supervisor in Mathematica software and then converted to the syntax of C++. At the end of chapter 4 we briefly discuss a method of phase portraits for visualization of some aspects of geodesics. We introduce the notion of the Lyapunov exponent and explain its implementation in our source code.

In the last chapter 5 we present some results obtained by our computer programme. Unfortunately, several interesting questions are left open, e.g. the existence of closed orbits for test particles. Nevertheless, we have obtained a number of figures illustrating the nature of geodesics bu plotting their trajectories and phase portraits. Then we briefly addressed the question of the causal structure of helically symmetric spacetime.

The most important results of the thesis, beside gathering an experience with general relativity and numerical methods, is the conclusion that binary systems cannot stay in equilibrium for arbitrary values of their parameters. In our work we use units in which the radius of A-

particle is $a = 1$ and b is expressed in multiples of a; angular velocity of of binary system is $\alpha \in (0, 1)$, masses of A and B particle are denoted by m and m', respectively. We have found that once the value of $b \neq 1$ is chosen, there is a unique value of α such that the binary system is in equilibrium. Equilibrium masses are then determined by b and α. Surprisingly enough, even parameter b cannot be chosen arbitrarily, for the conditions of equilibrium possess non-trivial solutions[1] only for

$$b \in (0,443;\ 2,257).$$

If $b = 1$, the situation is slightly different, for now two conditions of equilibrium become identical and we can freely specify α. This is a surprising difference to the Newtonian case, where an equilibrium value of α can be obtained for arbitrary combination of b, m and m'. In chapter 5 we plotted graphs showing all equilibrium choices of α, b, m and m'.

Discussion above finalizes our analysis of helically symmetric spacetimes representing binary systems in linearized Einstein's gravity. We thank the opponent of this thesis (who is probably its last reader) for his patience. We believe that this work has revealed some non-trivial properties of helically symmetric spacetimes. However, despite our enormous effort, several open questions remain...

[1] Trivial solutions are those with $\alpha = 0$. In such a case the only equilibrium values of masses are $m = m' = 0$; a flat spacetime.

Bibliography

[1] Beig R, Heinzle J M, Schmidt B G, Helically Symmetric N-Particle Solutions in Scalar Gravity, *Phys. Rev. Lett* **(98)**, 2007.

[2] Bičák J, Scholtz M, Bohata M, Helically symmetric 2-body problem in linearized gravity, *in preparation*

[3] Bonnazola S, Gourgoulhon E, Marck J A, Relativistic formalism to compute quasi-equilibrium configurations of non-synchronized neutron star binaries, *Phys. Rev. D* **(56)**, 1997.

[4] Friedman J L, Uryū K, Shibata M, Thermodynamics of binary black holes and neutron stars, *Phys. Rev. D* **(65)**, 2002.

[5] Garcia A L, Numerical methods for physics, Upper Saddle River, Prentice Hall, 2000.

[6] Jackson, J D 1998, Classical Electrodynamics, 3rd Edition, ISBN 0-471-30932-X. Wiley-VCH, July 1998.

[7] Sandri M, Numerical calculation of Lyapunov exponents, *The Mathematica Journal*, 1996

[8] Schild A, Electromagnetic two-body problem. Phys. Rev. **(131)**, 1963.

[9] Wald R M, General relativity. University of Chicago Press, Chicago, 1984.