Symbolic and Numerical Analysis in General Relativity with Open Source Computer Algebra Systems

Tolga Birkandan *[1], Ceren Güzelgün[1], Elif Şirin[1], and Mustafa Can Uslu[2]

[1]Istanbul Technical University, Department of Physics, Istanbul, Turkey
[2]Istanbul Technical University, Institute of Informatics, Istanbul, Turkey

March 30, 2017

Abstract

We study three computer algebra systems, namely SageMath (with SageManfolds package), Maxima (with ctensor package) and Python language (with GraviPy module), which allow tensor manipulation for general relativity calculations. We present simple examples and give a benchmark of these systems. After the general analysis, we focus on the SageMath+SageManifolds system to analyze and visualize the solutions of the massless Klein–Gordon equation and geodesic motion with Hamilton–Jacobi formalism.

1 Introduction

Computer algebra systems are essential tools for theoretical physics for some decades. They are mainly important in general relativity (GR) where lengthy tensorial and differential geometry calculations are inevitable. Many of these programs have special internal or external packages for tensor manipulation and differential geometry calculations. For comprehensive reviews, see [1,2]. The review of early works can be found in [3]. These packages can also be extended for more specialized calculations [4].

Even though commercial programs dominate the area, codes written on commercial programs cannot be distributed easily since those programs may not be available for scientists with lack of resources for purchasing such software. Besides, many of the manipulations treated in calculations do not need a sophisticated computation engine.

Open source computer algebra systems such as SageMath (also known as "Sage") [5] and Maxima [6] provide a complete toolkit for general relativity and quantum field theory applications with their particular packages. Some freely available programming languages such as Python [7] also offer special tools for abovementioned manipulations.

We choose to study general systems rather than specialized tensor manipulation or general relativity packages (e.g. Cadabra [8], Redberry [9], etc.). The reason for this choice is that tensor manipulation is generally an auxiliary step in GR calculations. A general program which can deal with symbolic and numerical analysis of the results constitute a complete calculation toolkit. In most cases, computer algebra systems (or programming languages) are supported with particular packages for tensor manipulation. We also exclude specialized tools like GYOTO [10] in our analysis.

In this work, we will employ SageMath (with SageManifolds package [11]), Maxima (with ctensor package [12]), and Python language (with Sympy and GraviPy modules [13]) for some essential calculations in general relativity and present benchmark results for these systems.

---

*E-mail address: birkandant@itu.edu.tr

The ever-developing open source SageMath program has gathered many utilities such as Maxima, GAP, R, and the power of Python language with well-known Python modules like NumPy, SymPy and matplotlib. SageMath can be installed on personal computers and moreover it has a powerful cloud computing server on which the user can work on projects anywhere and share them with other users easily [5]. The recent package SageManifolds for tensor and differential geometry calculations is also installed on the cloud server. These properties make the Sage+SageManifolds system the best open source choice for general relativity and quantum field theory.

Analysis of the Klein–Gordon equation, mainly its radial part is the first step in most quantum gravity problems involving black holes [14]. We will study its solution using SageMath.

Computationally, it is easier to define and solve the first order differential equations rather than the second order equations. Hamilton–Jacobi formalism yields first order differential equations for the geodesic motion [15]. We will use this formalism to derive the equations and solve them as a numerical initial value problem with SageMath routines. Numerical results need to be presented in a graphical way to see their structure efficiently. SageMath system is equipped with comprehensive visualization tools and we will use these tools to present our results.

In the next section, we define the spacetimes to be used. In the third section we review three computation systems and give simple examples. We also give a benchmark of these systems as a subsection. In the fourth section we focus on the SageMath+SageManifolds system to analyze the massless Klein–Gordon equation and geodesic motion.

2   Schwarzschild and Kerr solutions

We will be working for Schwarzschild and Kerr spacetimes. In general, a line element is defined by

$$ds^2 = g_{\mu\nu}dx^\mu dx^\nu. \tag{1}$$

If one takes $G_4 = c = 1$ (where G_4 is Newton's gravitational constant in four dimensions and c is the speed of light in vacuum), the Schwarzschild metric can be written as [15]

$$g_{\mu\nu} = \begin{bmatrix} \left(1 - \frac{2M}{r}\right) & 0 & 0 & 0 \\ 0 & -\left(1 - \frac{2M}{r}\right)^{-1} & 0 & 0 \\ 0 & 0 & -r^2 & 0 \\ 0 & 0 & 0 & -r^2 sin^2\theta \end{bmatrix}. \tag{2}$$

We will take the order of coordinates as $\{t, r, \theta, \phi\}$ and M is the mass of the black hole. The coefficient of the radial part is singular at $r = 2M$ which describes the "event horizon".

The Kerr black hole has the metric [15]

$$g_{\mu\nu} = \begin{bmatrix} \left(1 - \frac{2M}{\rho^2}\right) & 0 & 0 & \frac{2aMrsin^2\theta}{\rho^2} \\ 0 & -\frac{\rho^2}{\Delta} & 0 & 0 \\ 0 & 0 & -\rho^2 & 0 \\ \frac{2aMrsin^2\theta}{\rho^2} & 0 & 0 & -sin^2\theta\left[(r^2 + a^2) + \frac{2a^2Mrsin^2\theta}{\rho^2}\right] \end{bmatrix}. \tag{3}$$

Here, M is the mass and $a = J/M$ where J is the angular momentum. Two functions are defined as

$$\rho^2 = r^2 + a^2 cos^2\theta, \tag{4}$$

$$\Delta = r^2 - 2Mr + a^2. \tag{5}$$

The roots of $\Delta = 0$ correspond to the locations of the Cauchy horizon (the smaller root) and the event horizon (the larger root) of the black hole.

3 Open source computer algebra systems for basic GR

We will only focus on the procedures of

- Defining a spacetime
- Calculating Christoffel symbols ($\Gamma^{\mu}_{\nu\rho}$)
- Calculating Ricci tensor ($R_{\mu\nu}$)
- Calculating Einstein tensor ($G_{\mu\nu}$)
- Displaying components

in our examples. The examples will be studied using three computation systems, namely

- SageMath+SageManifolds
- Maxima+ctensor
- Python+GraviPy

The codes are enhanced with comment lines which can be helpful for analyzing the procedures step by step. Details on the procedures and further examples can be found in related references.

3.1 SageMath+SageManifolds

SageMath is an open source computer algebra system which collects many powerful open source packages and modules with a Python-like language [5]. SageManifolds was first started as an independent tensor analysis and differential geometry package to be installed on SageMath [11]. Now, SageManifolds is an internal package for SageMath and does not require an additional installation.

We start our program by reseting the SageMath environment. Then we define the four-dimensional manifold, the black hole mass and the coordinates. The coordinates are defined along with their ranges. We define the Lorentzian metric by giving its components as a matrix. Finally, we calculate and display Christoffel symbols, Ricci tensor and Einstein tensor elements.

```
1  reset();
2  # Define 4-dim. the manifold "Man":
3  Man = Manifold(4, 'Man', r'\mathcal{M}');
4
5  # Define the parameter "M" (mass):
6  M = var('M');
7
8  # Define the coordinates {t=0, r=1, theta(=th)=2, phi=3} with ranges
9  # (BL = Boyer-Lidquist)
10 BL.<t,r,th,ph> = Man.chart(r't r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi');
11
12 # Define the metric "g" on manifold "Man":
13 g = Man.lorentzian_metric('g');
14
15 # Enter the Schwarzschild metric components:
16 g[0,0] = (1-(2*M)/r);
17 g[1,1] = -1/(1-(2*M)/r);
18 g[2,2] = -r^2;
19 g[3,3] = -(r*sin(th))^2;
20
21 # Display the metric
```

```
22  show('The Schwarzschild metric:');
23  show(g.display());
24  #####################
25  # Christoffel symbols
26  nab = g.connection()
27  # Display all components
28  show(nab.display())
29  # Display a single component
30  show(nab[1,1,1])
31  #####################
32  # Ricci tensor
33  Ric = g.ricci();
34  # Display all components
35  show(Ric.display())
36  # Display a single component
37  show(Ric[1,1])
38  #####################
39  # Einstein tensor
40  ET=g.ricci()-(1/2)*g*g.ricci_scalar()
41  # Display all components
42  ET.set_name('E')
43  show(ET.display())
44  # Display a single component
45  show(ET[1,2])
```

The lines between 16-19 defines the Schwarzschild metric. In order to define the Kerr metric, we need to change this part with

```
16  a = var('a');
17  rhosq=r^2+(a^2)*cos(th)^2;
18  Delta=r^2-2*M*r+a^2;
19
20  g[0,0]  = (1-(2*M*r)/rhosq);
21  g[1,1]  = -rhosq/Delta;
22  g[2,2]  = -rhosq;
23  g[3,3]  = -(sin(th)^2)*((r^2+a^2)+(2*(a^2)*M*r*sin(th)^2)/rhosq);
24  g[0,3]  = (2*a*M*r*sin(th)^2)/rhosq;
```

It should be noted that, we defined the functions ρ^2 and Δ, and the rotation parameter a. The Kerr black hole has non-diagonal, symmetric components unlike the Schwarzschild metric which is diagonal.

3.2   Maxima+ctensor

Maxima computer algebra system is the new and freely available version of Macsyma [6]. Macsyma is known as the first general, multipurpose computer algebra system which inspired many other systems for years. V. Toth reviews tensor packages in Maxima in his article [12]. Here, we will use the ctensor package which provides component tensor manipulation.

We first kill all environmental variables, then load the ctensor package. We set `ratwtlvl:false` for no truncation and `ratfac:true` to factorize the tensor components automatically. We then define the dimension of the spacetime and the coordinates. We enter the metric as a matrix. Before the calculation of Christoffel symbols, Ricci tensor and Einstein tensor elements, we find the inverse tensor with `invert`.

```
1  kill(all);
2  if get('ctensor,'version)=false then load(ctensor);
3  (ratwtlvl:false,ratfac:true);
```

```
4  ("Specify the dimension of the manifold and the coordinate labels.")$
5  (dim:4,ct_coords:[t,r,theta,phi]);
6  ("Enter the metric.")$
7  lg:matrix([(1-2*M/r),0,0,0],[0,-1/(1-2*M/r),0,0],[0,0,-r^2,0],[0,0,0,-(r^2)*
       sin(theta)^2]);
8  ug:invert(lg)$
9  ("Compute the Christoffel symbols and display all components")$
10 christof(mcs)$
11 ("Compute the Ricci tensor and display all components")$
12 uricci(true)
13 ("Compute the Einstein tensor and display all components")$
14 einstein(true);
15 ("Display single components")$
16 mcs[1,1,2];
17 ric[1,2];
18 ein[2,2];
```

The lines 6-7 should be changed as

```
6  rhosq:r^2+(a^2)*cos(theta)^2$
7  Delta:r^2-2*M*r+a^2$
8  ("Enter the general static spherically symmetric metric.")$
9  lg:matrix([(1-(2*M*r)/rhosq),0,0,(2*a*M*r*sin(theta)^2)/rhosq],[0,-rhosq/Delta
       ,0,0],[0,0,-rhosq,0],[(2*a*M*r*sin(theta)^2)/rhosq,0,0,-(sin(theta)^2)*((r
       ^2+a^2)+(2*(a^2)*M*r*sin(theta)^2)/rhosq)]);
```

for the Kerr metric.

3.3 Python+GraviPy

Python is a multipurpose, object-oriented programming language [7] which can easily be expanded with modules. The module GraviPy provides tensor calculation methods and it works on a freely available symbolic analysis module SymPy [13].

We start our program by importing the GraviPy module. Then we define the four–vector of coordinates and the black hole mass. We define the metric as a diagonal matrix. We then calculate and display Christoffel symbols, Ricci tensor and Einstein tensor elements.

```
1  from gravipy import *
2  ######################
3  # Coordinates (\chi is the four-vector of coordinates)
4  t, r, theta, phi, M = symbols('t, r, theta, phi, M')
5  x = Coordinates('\chi', [t, r, theta, phi])
6  ######################
7  # Metric tensor
8  Metric = diag((1-2*M/r), -1/(1-2*M/r), -r**2, -r**2*sin(theta)**2)
9  g = MetricTensor('g', x, Metric)
10 ######################
11 # Christoffel symbols
12 Ga = Christoffel('Ga', g)
13 # Display all components
14 print(Ga(All, All, All))
15 # Display a single component
16 print(Ga(1,2,1))
17 ######################
18 # Ricci tensor
19 Ri = Ricci('Ri', g)
20 # Display all components
```

```
21  print(Ri(All, All))
22  # Display a single component
23  print(Ri(1, 2))
24  #####################
25  # Einstein tensor
26  G = Einstein('G', Ri)
27  # Display all components
28  print(G(All, All))
29  # Display a single component
30  print(G(3, 3))
31  #####################
```

The lines 2-8 should be modified as

```
2   #####################
3   # Coordinates (\chi is the four-vector of coordinates)
4   t, r, theta, phi, M, a, rhosq, Delta = symbols('t, r, theta, phi, M, a, rhosq,
        Delta')
5   x = Coordinates('\chi', [t, r, theta, phi])
6   #####################
7   # Metric tensor
8   rhosq = r**2+(a**2)*cos(theta)**2
9   Delta = r**2-2*M*r+a**2
10  Metric = Matrix([[(1-(2*M*r)/rhosq),0,0,(2*a*M*r*sin(theta)**2)/rhosq],[0,-
        rhosq/Delta,0,0],[0,0,-rhosq,0],[(2*a*M*r*sin(theta)**2)/rhosq,0,0,-(sin(
        theta)**2)*((r**2+a**2)+(2*(a**2)*M*r*sin(theta)**2)/rhosq)]])
```

in order to define the Kerr metric. The diagonal matrix definition of the Schwarzschild case is changed with a general metric.

3.4 Benchmark for the open source computer algebra systems

We performed some calculations for the Schwarzschild and Kerr spacetimes on Python+GraviPy (Python 2.7.12), SageMath+SageManifolds (SageMath 7.5.1) and Maxima+ctensor (Maxima 5.37.2) systems.

The Schwarzschild metric has only diagonal elements, while Kerr solution has non–diagonal elements in its metric. We aimed to measure the effect of this difference in the computations. We used the metric given in equation 2 for the Schwarzschild case and for the Kerr metric the metric is given in equation 3. The SageMath+SageManifolds system allows parallel computing on CPU by adding the line `Parallelism().set(nproc=4)` for 4–cores. We also measured the effect of this option in calculations.

For our analysis, we used the code–block profiling method and measured the wall–clock time as it has more importance for the general user. We calculated the Christoffel symbols, Ricci tensor and Einstein tensor one by one on each system. Metric definition and displaying the results are not included in the time measurement. By averaging the timing results for 10 runs on each system, the Table 1 is generated.

In Python, we used the `datetime` module. This method shows the time value up to six decimals.

```
1   from gravipy import *
2   from datetime import datetime
3   ...
4   riccitime_=datetime.now()
5   Ri=Ricci('Ri',g)
6   _riccitime=datetime.now()
7   print "Elapsed time...", _riccitime-riccitime_
```

```
8   ...
```

In SageMath, we employed the inline `time` command. This command shows time with two decimals.

```
1   ...
2   time Ric=g.ricci()
3   ...
```

In Maxima, we used `elapsed_real_time()` command which also shows time with two decimals.

```
1   ...
2   t0:elapsed_real_time()$
3   uricci(false);
4   t1:elapsed_real_time()-t0;
5   ...
```

The test computer has an Intel i7 4500U CPU (4096 KB cache for each 4 cores), 8 GB (DDR3) RAM, NVIDIA GeForce 840M GPU and 1 TB HDD with 8GB swap.

Metric - Calculation	Python	SageMath (core=1)	SageMath (core=4)	Maxima
Schwarzschild - Christoffel	0.000056	1.57	1.28	0.10
Schwarzschild - Ricci	0.000040	3.73	3.62	0.00
Schwarzschild - Einstein	0.000036	5.33	5.08	0.00
Kerr - Christoffel	0.000073	23.67	9.00	0.04
Kerr - Ricci	0.000050	128.71	85.48	0.40
Kerr - Einstein	0.000044	152.39	107.42	6.34

Table 1: Benchmark results. (All numbers are in seconds).

As seen from the Table 1, some calculations for the Schwarzschild metric seem unsuitable for parallelization due to chunk sizes of the CPU cores. However, using multiple cores for Kerr calculations makes a significant effect on calculation time.

According to these results, Python and Maxima would be the best choices for tensor manipulation. However, SageMath being powered by SageManifolds, provides an easy-to-use and combined toolkit for the general user.

4   Further examples on scalar wave equation and geodesics with SageMath and SageManifolds

We will focus on the SageMath+SageManifolds system to perform a set of calculations for the Schwarzschild metric. We will first define the spacetime by giving its variables and components. Then we will perform calculations for two examples. In the first one, we calculate the Klein–Gordon equation for a massless scalar field, extract its radial part and solve this differential equation numerically. We will further compare our numerical result graphically with the asymptotic form of the analytical result. In the next example, we will perform a very simple geodesic analysis for this spacetime using the Hamilton–Jacobi formalism.

Generalization of the example codes to other metrics is straightforward. The codes for the calculation of Klein–Gordon and Hamilton–Jacobi equations can be used for general metrics without modification. For the detailed calculations, we applied some metric-related information in the code to see the manipulation, equation solving and plotting capabilities of the computation system.

The reader should follow the line numbers to execute the codes without problem. The code for the definition of the spacetime is between lines (1-24). The analysis of the Klein–Gordon equation starts in line 25 and ends in line 126. Then in the next section, the study of geodesics starts again with line 25 and ends in line 102. This means that, the reader should first execute the lines (1-24) for the metric definition before each physical example.

4.1 Definition of the spacetime in SageManifolds

This part was studied before in Section 3.1, while giving examples for simple calculations in GR. Nevertheless, we will place it here in order to present a complete code structure.

```
1  reset();
2
3  # Define 4-dim. the manifold "Man":
4  Man = Manifold(4, 'Man', r'\mathcal{M}');
5
6  # Define the parameter "M" (mass):
7  M = var('M');
8
9  # Define the coordinates {t=0, r=1, theta(=th)=2, phi=3} with ranges
10 # (BL = Boyer-Lidquist)
11 BL.<t,r,th,ph> = Man.chart(r't r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi');
12
13 # Define the metric "g" on manifold "Man":
14 g = Man.lorentzian_metric('g');
15
16 # Enter the Schwarzschild metric components:
17 g[0,0] = (1-(2*M)/r);
18 g[1,1] = -1/(1-(2*M)/r);
19 g[2,2] = -r^2;
20 g[3,3] = -(r*sin(th))^2;
21
22 # Display the metric
23 show('The Schwarzschild metric:');
24 show(g.display());
```

In the Schwarzschild case, the manifold has four dimensions and the only variable is the black hole mass M. We define four spacetime coordinates and enter the components of the diagonal metric. These definitions may need some modifications for other spacetimes.

4.2 Klein–Gordon equation in curved spacetime

Klein–Gordon equation for a massless scalar field can be written as [16]

$$\frac{1}{\sqrt{-g}}\partial_\mu(\sqrt{-g}g^{\mu\nu}\partial_\nu\Phi) = 0, \tag{6}$$

where Φ is the scalar field and it can be decomposed with the Ansatz

$$\Phi(t,r,\theta,\phi) = e^{-i\omega t}e^{ik\phi}\,R(r)\,S(\theta), \tag{7}$$

for the Schwarzschild metric.

We will first define the variables ω, k and the result KG. The inverse metric and $\sqrt{-g}$ are needed in calculation, thus are calculated. Then the full scalar function Φ, radial and angular parts of the solution Ansatz are defined. After giving the Ansatz, we start calculating the Klein-Gordon equation in two for loops. The outer loop is over μ components and the inner one is over ν components.

```
25  # Defining variables:
26  var('omega,k,KG');
27
28  # Inverse metric:
29  ginv = g.inverse();
30
31  # The square root of the absolute value
32  # of the metric determinant:
33  sqrtabsdetg=g.sqrt_abs_det().expr();
34
35  # The scalar function Phi(t,r,th,phi):
36  # The dependence on all coordinates
37  # is provided by "(*BL)"
38  Phi=function('Phi')(*BL)
39
40  # The scalar field Ansatz is given here.
41  # R : Radial part,
42  # S : Angular part
43  R=function('R')(r);
44  S=function('S')(th);
45  Phi=exp(-I*omega*t)*exp(I*k*ph)*R*S;
46
47  # Calculating the Klein-Gordon equation:
48  KG=0;
49  for mu in range(len(BL[:])):
50    for nu in range(len(BL[:])):
51      KG=KG+diff((ginv[mu,nu].expr()*sqrtabsdetg*diff(Phi,BL[nu])),BL[mu]);
52
53  # Displaying the Klein-Gordon equation "KG"
54  show('The full Klein-Gordon equation (variable name is KG):')
55  show(KG)
```

This part of the code is general. The reader should only change the solution Ansatz accordingly and the code can find the Klein-Gordon equation for any spacetime with any number of dimensions. The result is stored in variable KG.

In our example, the Klein–Gordon equation is found as

$$
\begin{aligned}
\text{KG} =& \frac{\omega^2 r^3 R\left(r\right) S\left(\theta\right) e^{\left(i\,k\phi - i\,\omega t\right)} \sin\left(\theta\right)}{2\,M - r} + \left(2\,M - r\right) r S\left(\theta\right) e^{\left(i\,k\phi - i\,\omega t\right)} \sin\left(\theta\right) \frac{\partial^2}{\left(\partial r\right)^2} R\left(r\right) \\
&+ \left(2\,M - r\right) S\left(\theta\right) e^{\left(i\,k\phi - i\,\omega t\right)} \sin\left(\theta\right) \frac{\partial}{\partial r} R\left(r\right) - r S\left(\theta\right) e^{\left(i\,k\phi - i\,\omega t\right)} \sin\left(\theta\right) \frac{\partial}{\partial r} R\left(r\right) \\
&+ \frac{k^2 R\left(r\right) S\left(\theta\right) e^{\left(i\,k\phi - i\,\omega t\right)}}{\sin\left(\theta\right)} - R\left(r\right) \cos\left(\theta\right) e^{\left(i\,k\phi - i\,\omega t\right)} \frac{\partial}{\partial \theta} S\left(\theta\right) \\
&- R\left(r\right) e^{\left(i\,k\phi - i\,\omega t\right)} \sin\left(\theta\right) \frac{\partial^2}{\left(\partial \theta\right)^2} S\left(\theta\right).
\end{aligned}
\tag{8}
$$

We can find the radial and angular parts of the Klein-Gordon equation after analyzing its structure. This part is semi-automatic and the user should supply some information.

We first divide the equation by the common factors and extract its operands in a vector. Then we define the separation constant λ_{aux}. Using a loop over all operands, we decide which one belongs to the radial part and which one belongs to the angular part of the equation. We take the radial derivative of the components and if the result is zero, the component is angular, if it is not zero, we add it to the radial part. We use collection and simplification commands to see the results in a good shape.

```
56  # We can analyze the Klein-Gordon equation
57  # to see how it is separated into
```

```
58  # radial and angular parts.
59
60  # Common factors will be divided
61  # This part should be given by the user
62  divideKGby=exp(-I*omega*t)*exp(I*k*ph)*sin(th)*R*S;
63  finalKG=expand(KG/divideKGby);
64
65  # Extract the operands in the expression:
66  fkgops=finalKG.operands()
67
68  # Find radial and angular parts:
69  # lambda_aux is the separation constant
70  var('lambda_aux');
71  KGradialpart=lambda_aux;
72  KGangularpart=-lambda_aux;
73  for term in fkgops:
74      if diff(term,r)==0:
75          KGangularpart=KGangularpart+term;
76      else:
77          KGradialpart=KGradialpart+term;
78
79  KGradialpart=expand(KGradialpart*R).simplify_full().collect(R).collect(diff(R,
        r)).collect(diff(R,r,r));
80  KGangularpart=expand(KGangularpart*S).simplify_full().collect(S).collect(diff(
        S,th)).collect(diff(S,th,th));
81
82  show('The radial part (variable name is KGradialpart):')
83  show(KGradialpart);
84  show('The angular part (variable name is KGangularpart):')
85  show(KGangularpart);
```

The radial part is then found as

$$\texttt{KGradialpart} = \frac{\left(\omega^2 r^3 + 2\,M\lambda_{aux} - \lambda_{aux}r\right)R\left(r\right)}{2\,M-r} + \frac{2\left(2\,M^2 - 3\,Mr + r^2\right)\frac{\partial}{\partial r}R\left(r\right)}{2\,M-r}$$
$$+ \frac{\left(4\,M^2 r - 4\,Mr^2 + r^3\right)\frac{\partial^2}{(\partial r)^2}R\left(r\right)}{2\,M-r}, \tag{9}$$

and the angular part is

$$\texttt{KGangularpart} = -\frac{\cos\left(\theta\right)\frac{\partial}{\partial\theta}S\left(\theta\right)}{\sin\left(\theta\right)} - \frac{\left(\lambda_{aux}\sin\left(\theta\right)^2 - k^2\right)S\left(\theta\right)}{\sin\left(\theta\right)^2} - \frac{\partial^2}{(\partial\theta)^2}S\left(\theta\right). \tag{10}$$

Further simplifications are obvious but SageMath's `desolve` command can not give symbolic solutions for these equations. However, numerical solutions can be studied using related methods. We will use `desolve_system_rk4` as an example and solve the radial equation numerically. This method uses a fourth–order Runge–Kutta scheme and in fact, the command `desolve_system_rk4` is used as a wrapper for the Maxima command `rk` [5].

Numerical solvers can generally deal with first order equations. Our second order equation will yield two first order differential equations which will be solved simultaneously. We will define the first derivative of the radial function as an auxiliary function

$$\frac{dR}{dr} = R_{aux}, \tag{11}$$

and place it in the equation 9 to have

$$\frac{dR_{aux}}{dr} = -\frac{4\,M^2 R_{\text{aux}} + 2\,r^2 R_{\text{aux}} + 2\left(\lambda_{aux}R - 3\,rR_{\text{aux}}\right))M + \left(\omega^2 r^3 - \lambda_{aux}r\right)R}{4\,M^2 r - 4\,Mr^2 + r^3}. \tag{12}$$

The following code finds this set of equations as `radial1` and `radial2`. The solver can deal with the right hand sides of $\frac{df(x)}{dx} = g(f(x), x)$ type equations. Thus we isolate the derivatives by solving the differential equations as algebraic equations. `radial1` and `radial2` are the right hand sides of our equations to be solved.

```
86  # The auxiliary function
87  R_aux=function('R_aux')(r)
88
89  # Two auxiliary equations
90  radial1aux=diff(R,r)-R_aux;
91  radial2aux=KGradialpart.subs(diff(R,r)==R_aux).subs(diff(R,r,r)==diff(R_aux,r)
        );
92
93  # Right hand sides of the derivatives
94  radial1=(solve(radial1aux==0,diff(R,r)))[0].right();
95  radial2=(solve(radial2aux==0,diff(R_aux,r)))[0].right();
96
97  # Getting the outputs in input format
98  # (Outputs will be copied)
99  print(radial1)
100 print(radial2)
```

In their present forms, R and R_{aux} are defined as functions. However, in order to use `desolve_system_rk4` we need to define unknowns as variables. To do this, we need to copy the outputs for `radial1` and `radial2` and change them accordingly. We display the outputs by `print` command to see them in the input form which enables us to copy them easily.

After a formal analysis of the radial part, one can see that the radial solution can be given in terms of confluent Heun functions [17]. General and confluent forms of the Heun function are encountered in many applications in physics, especially as solutions of the wave equations [18–23].

The general Heun function is adapted by Oleg V. Motygin for GNU Octave [24]. However, no freely available packages or modules can deal with confluent Heun functions. Therefore we cannot compare our result with the full analytical solution. The asympotic form of the confluent Heun function is given in reference [17] as

$$R_\ell = \frac{C_\ell}{r} \sin\left[\omega r + 2M\omega\ln(r) - \frac{\ell\pi}{2} + \arg\left(\Gamma(\ell + 1 - 2iM\omega)\right)\right], \tag{13}$$

where $\lambda_{aux} = \ell(\ell + 1)$.

We first import the differential equation solver from `sage.calculus.desolvers`. We then define the variables (including the unknown functions) and the equations and copy the equations from the outputs of the code above. We give some numerical values to the parameters arbitrarily and solve the system for some arbitrary set of initial conditions. In our example we have $R(r = 0.3) = 1$ and $\frac{dR}{dr} = R_{aux}(r = 0.3) = 0.5$. The solution is stored in `radsol`.

`radsol` has the structure $[r, R, R_{aux}]$. `points=[[i,j] for i,j,k in radsol]` command creates `[i,j]` (or $[r, R]$) points for plotting. `radialsolution` stores the plot of the numerical result.

In the next part, we define the asymptotic form of the analytical solution and plot it for the same parameter set (we take $C_\ell = 250$ to match the amplitude). We display both plots together to show the agreement.

```
101 # Import the solver
102 from sage.calculus.desolvers import desolve_system_rk4
103
104 # Define unknowns as variables
105 var('R,R_aux,r');
106
```

```
107   # Define equations
108   radial1=R_aux;
109   radial2=-(4*M^2*R_aux + 2*r^2*R_aux + 2*(lambda_aux*R - 3*r*R_aux)*M + (omega
          ^2*r^3 - lambda_aux*r)*R)/(4*M^2*r - 4*M*r^2 + r^3);
110
111   # Substitute numerical values for parameters
112   radial2=radial2.subs(M=0.1,omega=0.2,k=2.0,lambda_aux=2.0)
113
114   # Solve the system and plot the solution
115   radsol=desolve_system_rk4([radial1,radial2],[R,R_aux],ics=[0.3,1,0.5],ivar=r,
          end_points=100,step=0.01);
116   points=[[i,j] for i,j,k in radsol];
117   radialsolution=list_plot(points,axes_labels=['$r$','$R$'],legend_label='
          Numerical solution');
118
119   # Asymptotic form of the analytic solution
120   var('Cl,L');
121   Rasym=Cl*(1/r)*sin(omega*r+2*M*omega*log(r)-(L*pi/2)+arg(gamma(L+1-2*I*M*omega
          )))
122   Rasymnum=Rasym.subs(Cl=250,M=0.1,omega=0.2,L=1.0)
123   asympplot=plot(Rasymnum,(r, 20, 100),linestyle='',marker='x',color='red',
          legend_label='Asymptotic solution')
124
125   # Display both plots
126   show(asympplot+radialsolution)
```

The plot 1 of the solution gives an idea on the behavior of the radial function. Numerical solution and the asymptotic form of the analytical solution are plotted together to show that they agree for large r.
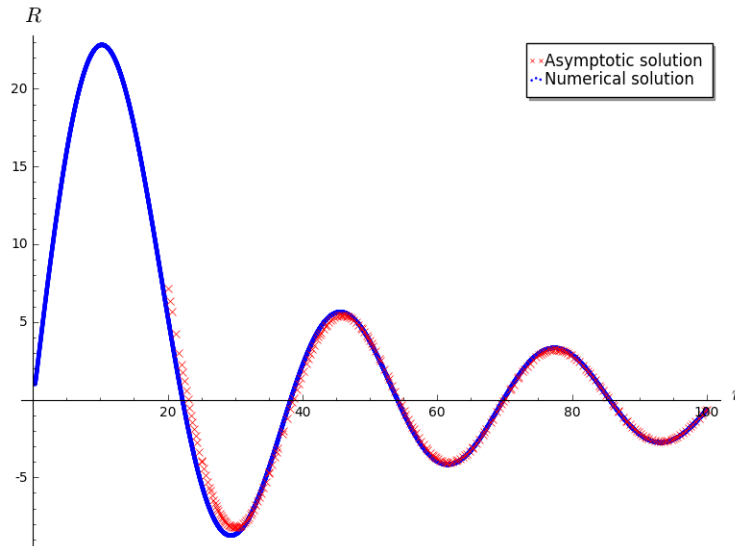

Figure 1: The radial part of the Klein–Gordon solution.

4.3   Visualizing geodesics

The geodesic motion in a spacetime can be described by the Hamilton–Jacobi equation [15].

$$\frac{\partial S}{\partial \eta} - \frac{1}{2}g^{\mu\nu}\frac{\partial S}{\partial x^\mu}\frac{\partial S}{\partial x^\nu} = 0, \tag{14}$$

where S denotes the Hamilton's principal function, $g^{\mu\nu}$ is the inverse metric and η is an affine parameter. The orbital equations then found using

$$\frac{\partial x^\mu}{\partial \eta} + g^{\mu\nu}\frac{\partial S}{\partial x^\nu} = 0. \tag{15}$$

This formalism gives first order differential equations which are needed in numerical schemes as mentioned in the previous section.

The Hamilton's principal function is decomposed as

$$S(\eta, t, r, \theta, \phi) = \frac{m^2}{2}\eta - Et + L\phi + F(r) + G(\theta), \tag{16}$$

for the Schwarzschild spacetime. For the equatorial geodesics we take $\theta = \pi/2$ and thus $G(\theta) = 0$. The metric components also change according to this constraint.

Before beginning our analysis, it should be emphasized that decomposition of the Hamilton's principle function S (and the scalar field Φ in the previous section) is not evident and needs a detailed analysis in general spacetimes, which is beyond the scope of our work. Moreover, analysis of geodesics is a very detailed study and our primitive example here aims only to visualize some geodesics.

The user should execute the lines (1-24) for the definition of the spacetime before running the codes below.

We start by defining our variables and functions. We also calculate the inverse metric as needed in the equations. After giving the Hamilton's principle function Ansatz, we calculate the Hamilton–Jacobi equation in two loops and set it to the variable `HJfull`. We call it `HJfull` as we have not put any conditions ($\theta = \pi/2$, etc.) on the equation yet.

The right hand sides (as needed by differential equation solvers) of the geodesic (orbital) equations are then calculated (using nested loops) and put in the vector `geodeqnrhs`. We display the equations both in the LaTeX format and as a vector.

```
25  # Define variables, functions and calculate inverse metric
26  var('eta,m,E,L,S,HJfull');
27  F=function('F')(r);
28  G=function('G')(th);
29  ginv = g.inverse();
30
31  # Define the principal function Ansatz
32  S=((eta*m^2)/2)-E*t+L*ph+F+G
33
34  # Calculate the Hamilton-Jacobi equation
35  HJfull=0;
36  for i in range(len(BL[:])):
37    for j in range(len(BL[:])):
38      HJfull=HJfull+ginv[i,j].expr()*diff(S,BL[i])*diff(S,BL[j]);
39  HJfull=(diff(S,eta)-(1/2)*HJfull);
40  show('The Full Hamilton-Jacobi equation (variable name is HJfull):')
41  show(HJfull);
42
43  show('The geodesic equations in LaTeX form (variable name is geodeqnrhs):')
44  geodeqnrhs=zero_vector(SR, len(BL[:]))
45
46  for mu in range(len(BL[:])):
47    for nu in range(len(BL[:])):
48      geodeqnrhs[mu]=geodeqnrhs[mu]-(ginv[mu,nu].expr())*diff(S,BL[nu]);
49    writeresult='D[0](%s)($\eta$) = $%s$' %(BL[mu],latex(geodeqnrhs[mu]));
50    show(writeresult);
51
```

```
52   show('Right hand sides of the geodesic equations as a vector')
53   show(geodeqnrhs)
```

The code above is general and it can work for any spacetime if the Hamilton's principal function is given accordingly. The codes below depend on the Schwarzschild metric.

Conventionally, we take $\theta = \pi/2$ to find the equatorial geodesics. We call the Hamilton–Jacobi equation with this constraint as HJst and the right hand sides of the orbital equations as geodeqnrhsst. We substitute $\theta = \pi/2$ and $G(\theta) = 0$ in the equations found in the general code.

```
54   # Take theta = pi/2:
55   var('HJst')
56
57   # Hamilton-Jacobi equation for theta=pi/2 (variable name is HJst)
58   HJst=(HJfull.subs(diff(G)==0)).subs(th=pi/2)
59   show(HJst)
60
61   # Right hand sides of the geodesic equations for theta=pi/2
62   # (variable name is geodeqnrhsst)
63   geodeqnrhsst=(geodeqnrhs.subs(diff(G)==0)).subs(th=pi/2)
64   show(geodeqnrhsst)
```

We find

$$\texttt{HJst} = \frac{m^2}{2} + \frac{E^2 r}{2(2M-r)} + \frac{L^2}{2r^2} - \frac{2M-r}{2r}\left(\frac{dF(r)}{dr}\right)^2, \tag{17}$$

and the nonzero components of geodeqnrhsst are

$$\frac{dt}{d\eta} = -\frac{Er}{2M-r}, \tag{18}$$

$$\frac{dr}{d\eta} = -\frac{2M-r}{r}\left(\frac{dF(r)}{dr}\right), \tag{19}$$

$$\frac{d\phi}{d\eta} = \frac{L}{r^2}, \tag{20}$$

We are now ready to solve the orbital equations. Here, we will use another solver, desolve_odeint by importing from sage.calculus.desolvers. This solver uses scipy.integrate module of Python. We will plot the geodesic curve, and a black disc with a radius equal to the event horizon radius will indicate the black hole. Thus, we import Circle from sage.plot.circle. We define the variables and give them arbitrary numerical values. We isolate $\frac{dF(r)}{dr}$ as derofradfun from HJst and place it in $\frac{dr}{d\eta}$ equation.

We would like to solve time-dependent $\frac{dr}{dt}$ and $\frac{d\phi}{dt}$ equations, instead of the equations with affine-parameter (η) dependence. Thus, we divide $\frac{dr}{d\eta}$ and $\frac{d\phi}{d\eta}$ by $\frac{dt}{d\eta}$, and set them as our equations: geodeqn1 and geodeqn2.

We solve our equations for arbitrarily set initial conditions and put the results in the variable sol. The 0th column of sol carries the r values and the 1st column carries the ϕ values. Using these, we generate $\{x, y\}$ points, where $x = r\cos(\phi)$ and $y = r\sin(\phi)$. The line plot of these points forms the geodesic curve. We plot this curve and the black disc (a circle with parameters fill=True,rgbcolor='black') together to display the behavior.

```
65   # Importing the solver
66   from sage.calculus.desolvers import desolve_odeint
67   #Importing circle for visualizing the black hole
68   from sage.plot.circle import Circle
69
70   var('m_aux,L_aux,E_aux,M_aux,r_initial,ph_initial,eta_end,step_size');
```

```
71
72  ##########################
73  # Variables
74  m_aux=0; # Either 1 (timelike) or 0 (null).
75  M_aux=10;
76  L_aux=23;
77  E_aux=0.4429;
78  step_size=1;
79  eta_end=10000;
80  r_initial=10*M_aux;
81  ph_initial=0.3;
82  ##########################
83
84  # Derivative of F (the radial function)
85  # (variable name is derofradfun)
86  # We will use the first root
87  derofradfun=solve(HJst,diff(F,r));
88
89  # Define equations to solve
90  # dr/dt = geodeqn1
91  # dphi/dt = geodeqn2
92  geodeqn1=((geodeqnrhsst[1]/geodeqnrhsst[0]).subs(diff(F,r)==derofradfun[1].rhs
        ())).subs(E=E_aux,L=L_aux,m=m_aux,M=M_aux);
93  geodeqn2=(geodeqnrhsst[3]/geodeqnrhsst[0]).subs(E=E_aux,L=L_aux,m=m_aux,M=
        M_aux);
94
95  # Solve the equations
96  sol=desolve_odeint([geodeqn1,geodeqn2],[r_initial,ph_initial],srange(0,eta_end
        ,step_size),[r,ph])
97  p=line(zip(sol[:,0]*cos(sol[:,1]),sol[:,0]*sin(sol[:,1])))
98
99  # Plot the black hole as a circle
100 # Show the geodesics and the circle on the same plot
101 C=circle((0,0),2*M_aux,fill=True,rgbcolor='black');
102 show(C+p)
```

The figure 2 shows an example of a null geodesic ($m = m_{aux} = 0$) for an arbitrary set of parameters. Figure 3 shows the stable circular orbit for a timelike particle ($m = m_{aux} = 1$) and Figure 4 shows another example of a timelike geodesic.
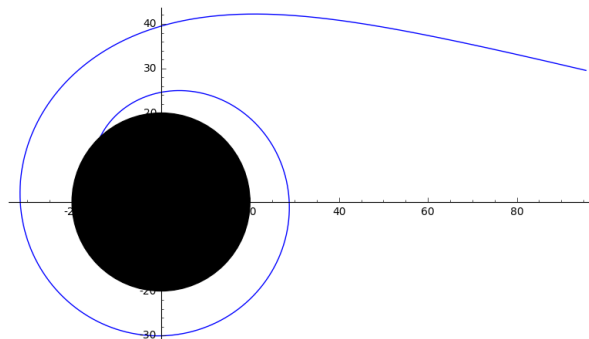

Figure 2: Example of a null geodesic for the Schwarzschild black hole ($\theta = \pi/2$).
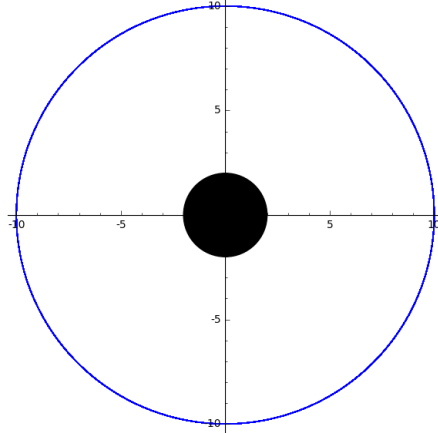
15

Figure 3: Stable circular orbit for the Schwarzschild black hole for a timelike particle ($\theta = \pi/2$).
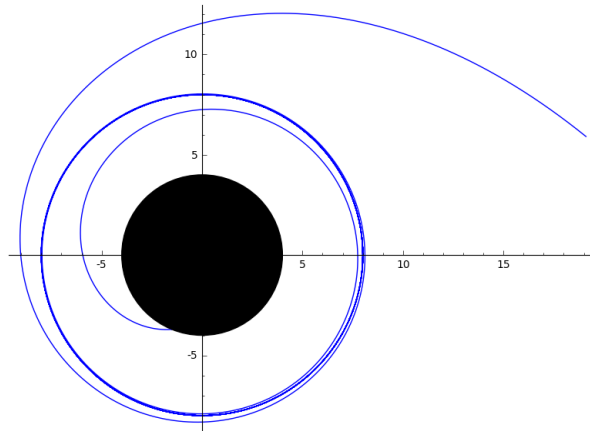

Figure 4: Example of a timelike geodesic for the Schwarzschild black hole ($\theta = \pi/2$).

5   Conclusion

We studied three computer algebra systems which allow tensor manipulation for GR calculations, and showed how one can define a spacetime, calculate Christoffel symbols, Ricci tensor and Einstein tensor for this spacetime using the specialized commands of these systems. We used SageMath with its tensor manipulation and differential geometry module SageManifolds, Maxima with its tensor component manipulation package ctensor and the Python language with the GraviPy module which runs on the symbolic analysis module SymPy. A benchmark for these systems is also provided.

In the next part, our main focus was the SageMath+SageManifolds system to perform two examples: Massless Klein–Gordon equation calculations and geodesic analysis for the Schwarzschild geometry. First, general codes that can run for general spacetimes are given, and then the analysis is specialized for the Schwarzschild case.

In our first example, we found the massless Klein–Gordon equation as a partial differential equation. We then separated it to the radial and angular parts and analyzed the radial part in detail. We confirmed our numerical result graphically using the asymptotic form of the analytical result.

In the second example, the orbit of a null or timelike particle around a Schwarzschild black hole was constructed from the geodesic equations. There are multiple methods of doing that, and among them,

the chosen one here is the Hamilton–Jacobi formalism as it yields first order differential equations that can be treated easier than the second order equations on the computer. The orbits are visualized using the plotting tools of SageMath.

A computation system should be chosen according to the needs of the problem. For example, if a research topic depends on the symbolic manipulations of special functions of mathematical physics, commercial packages are inevitable for most of the cases in the present situation. However, many problems do not need such specialized calculations and numerical analysis is sufficient to see the result. Therefore freely available packages offer complete computation systems for most of the scientific problems.

Acknowledgement

References

[1] C. Heinicke and F. W. Hehl, "Computer algebra in gravity," in *Computer Algebra Handbook*, J. Grabmeier, E. Kaltofen, V. Weispfennig (ed.s), Springer, Berlin (2003) [arXiv: gr-qc/0105094].

[2] A. V. Korolkova, D. S. Kulyabov and L. A. Sevastyanov, "Tensor computations in computer algebra systems," A. V. Korol'kova, D. S. Kulyabov, and L. A. Sevast'yanov. Tensor computations in computer algebra systems. Programming and Computer Software, 39(3):135–142, 2013 doi:10.1134/S0361768813030031 [arXiv:1402.6635 [cs.SC]].

[3] I. Cohen, I. Frick and J. E. Åman, "Algebraic Computing in General Relativity," Fundam. Theor. Phys. **9**, 139 (1984). doi:10.1007/978-94-009-6469-3_10

[4] T. Birkandan, "A Newman-Penrose Calculator for Instanton Metrics," Int. J. Mod. Phys. C **19**, 1277 (2008) doi:10.1142/S0129183108012807 [arXiv:0711.0613 [gr-qc]].

[5] *SageMath, the Sage Mathematics Software System (Version 7.5.1)*, The Sage Developers (2017) http://www.sagemath.org.

[6] *Maxima, a Computer Algebra System (Version 5.34.1)* (2014) http://maxima.sourceforge.net/.

[7] Python Software Foundation, *Python Language Reference, version 2.7* (2017) http://www.python.org.

[8] K. Peeters, "Introducing Cadabra: A Symbolic computer algebra system for field theory problems," hep-th/0701238 [HEP-TH].

[9] S. Poslavsky and D. Bolotin, "Redberry: a computer algebra system designed for tensor manipulation," J. Phys. Conf. Ser. **608**, no. 1, 012060 (2015). doi:10.1088/1742-6596/608/1/012060

[10] F. H. Vincent, T. Paumard, E. Gourgoulhon and G. Perrin, "GYOTO: a new general relativistic ray-tracing code," Class. Quant. Grav. **28**, 225011 (2011) doi:10.1088/0264-9381/28/22/225011 [arXiv:1109.4769 [gr-qc]] http://gyoto.obspm.fr/index.html.

[11] E. Gourgoulhon, M. Bejger and M. Mancini, "Tensor calculus with open-source software: the SageManifolds project," J. Phys. Conf. Ser. **600**, no. 1, 012002 (2015) doi:10.1088/1742-6596/600/1/012002 [arXiv:1412.4765 [gr-qc]]. http://sagemanifolds.obspm.fr/

[12] V. Toth, "Tensor manipulation in GPL Maxima," arXiv:cs/0503073v2 [cs.SC].

[13] *GraviPy, Tensor Calculus Package for General Relativity (Version 0.1.0)* (2014) `https://pypi.python.org/pypi/GraviPy`.

[14] T. Birkandan and M. Cvetič, "Conformal Invariance and Near-extreme Rotating AdS Black Holes," Phys. Rev. D **84**, 044018 (2011) doi:10.1103/PhysRevD.84.044018 [arXiv:1106.4329 [hep-th]].

[15] S. Chandrasekhar, *The Mathematical Theory of Black Holes,* Oxford University Press, New York (1992).

[16] N. D. Birrell, P. C. W. Davies, *Quantum Fields in Curved Space,* Cambridge University Press, Cambridge (1986).

[17] H. S. Vieira and V. B. Bezerra, "Confluent Heun functions and the physics of black holes: Resonant frequencies, Hawking radiation and scattering of scalar waves," Annals Phys. **373**, 28 (2016) doi:10.1016/j.aop.2016.06.016 [arXiv:1603.02233 [gr-qc]].

[18] A. Ronveaux (ed.), *Heun's Differential Equation*, Oxford University Press, Oxford (1995).

[19] S. Y. Slavyanov and W. Lay, *Special Functions, A Unified Theory Based on Singularities*, Oxford University Press, Oxford (2000).

[20] M. Hortaçsu, "Heun Functions and their uses in Physics" in Proceedings of the 13th Regional Conference on Mathematical Physics, Antalya, Turkey, October 27-31, 2010, U. Camci and I. Semiz (eds.), p. 23, World Scientific, Singapore (2013) [arXiv:1101.0471 [math-ph]].

[21] T. Birkandan and M. Hortaçsu, "Examples of Heun and Mathieu functions as solutions of wave equations in curved spaces," J. Phys. A **40**, 1105 (2007) [J. Phys. A **40**, 11203 (2007)] doi:10.1088/1751-8113/40/5/016, 10.1088/1751-8121/40/36/C01 [gr-qc/0607108].

[22] T. Birkandan and M. Hortaçsu, "Quantum Field Theory Applications of Heun Type Functions," Rept. Math. Phys. **79**, 81 (2017) doi:10.1016/S0034-4877(17)30022-8 [arXiv:1605.07848 [hep-th]].

[23] P. Fiziev and D. Staicova, "Application of the confluent Heun functions for finding the quasinormal modes of nonrotating black holes," Phys. Rev. D **84**, 127502 (2011) doi:10.1103/PhysRevD.84.127502 [arXiv:1109.1532 [gr-qc]].

[24] O. V. Motygin, "On numerical evaluation of the Heun functions," Proc. Days on Diffraction p. 222 (2015) doi: 10.1109/DD.2015.7354864 [arXiv:1506.03848 [math.NA]] `https://github.com/motygin/Heun_functions/`.