5-8-2014

Analyzing Solutions to the Einstein Equations Using Differential Geometry

Jordan Rozum
Utah State University

Analyzing Solutions to the Einstein Equations using Differential Geometry

Final Project for PHYS 4900: Research in Physics

Jordan Rozum, USU, 05/08/2014

Part 1: The Simplest Example, Minkowski Space

In part one, I walk through some examples using the Minkowski metric.

To start working with DifferentialGeometry, we have to load in the required packages. We'll also set some preferences to make the output look nicer.

```
> with(DifferentialGeometry):
> with(Tensor):
> Preferences("TensorDisplay", 1):
> interface(typesetting = extended):
```

We will start with the most fundamental solution to the Einstein equations, the Minkowski Metric. Before we can define the metric, we have to set up
manifold on which it will exist. We'll do this using the DGsetup command. When defining a manifold, it takes a list of coordinates and a manifold name.
The verbose option will print make it print a list of the variables DifferentialGeometry has reserved for the manifold.

```
> DGsetup([t,x,y,z],M,verbose);
```

The following coordinates have been protected:

$$[t, x, y, z]$$

The following vector fields have been defined and protected:

$$[\partial_t, \partial_x, \partial_y, \partial_z]$$

The following differential 1-forms have been defined and protected:

$$[dt, dx, dy, dz]$$

frame name: M **(1.1)**

Now that our manifold is defined, we can create a tensor using these coordinates. Maple will not know that this tensor is our metric, so we will have to pass
this tensor into many of our commands as an argument later. The evalDG command instructs Maple to interpret the argument using DifferentialGeometry.
Here, &t is short for &tensor, which also works and is the tensor product.

```
M > g := evalDG( dx &t dx + dy &t dy + dz &t dz - dt &t dt ); #
    The Minkowski Metric
```

$$g := -dt \otimes dt + dx \otimes dx + dy \otimes dy + dz \otimes dz \quad \textbf{(1.2)}$$

The first thing we'll do with this metric is check that it satisfies the Einstein equations. This is a vacuum sollution, meaning that the energy-momentum tensor
is zero. Also, there is no Λ, or cosmological constant for this sollution. Therfore, $G + \Lambda g = 8\,\pi T$ reduces to $G = 0$. We can easily check that G, the Einsten
tensor, is zero using the EinsteinTensor command:

```
M > G := EinsteinTensor(g);
```

$$G := 0\,\partial_t \otimes \partial_t \tag{1.3}$$

There are other things we could calculate. For example, the Reimann curvature tensor and the Christoffel symbols:

```
M > R := CurvatureTensor(g);
```

$$R := 0\,\partial_t \otimes dt \otimes dt \otimes dt \tag{1.4}$$

```
M > Gamma := Christoffel(g,"FirstKind");
```

$$\Gamma := 0\,dt \otimes dt \otimes dt \tag{1.5}$$

These are all pretty boring, but we'll come back to these commands for more interesting cases later. For now, we'll move on to algebraic classifications.
Three important classifications of metrics are the Petrov type, Segre type, and the Petrov-Peblanski type.

The Petrov type classifies the metric according to the eigenvectors of the Weyl tensor (the trace-free part of the curvature tensor, which can be calculated
using WeylTensor(g)). We can find the Petrov type using PetrovType(g).

The Petrov-Peblanski type classifies the metric according to the eigenvectors of the Peblanski tensor, which has the same algebraic symmetries as the
Weyl tensor. We actually calculate the Petrov-Peblanski type using SegreType(g), which returns two things; the first is the Petrov-Peblanski type,
and the second is the Segre type.

The Segre type classifies the metric according to the eigenvectors of the Ricci tensor (the trace component of the curvature tensor, which can be calculated
using RicciTensor(g)).

```
M > PetrovType(g);
```

"O" **(1.6)**

```
M > SegreType(g);
```

"O", "[(1,111)]" **(1.7)**

These classifications are helpful because they don't depend on the coordinates in which the metric is expressed. Another coordinate invariant property of
metrics is the symmetry they exhibit, expressed as the isometry algebra. The isometry algebra is a Lie

algebra whose basis vectors are (some linear
combination of) the so-called Killing vectors. While the Killing vectors themselves are coordinate dependent, the algebra they form under commutation
is unique up to isomorphism. Later in this tutorial, we will take a look at ways to classify these algebras, but for now, we'll just calculate the isometry algebra.

```
M > KV := KillingVectors(g);
```

$$KV := \left[z\,\partial_t + t\,\partial_z,\ -\partial_t,\ -z\,\partial_x + x\,\partial_z,\ -z\,\partial_y + y\,\partial_z,\ \partial_z,\ y\,\partial_t + t\,\partial_y,\ -y\,\partial_x + x\,\partial_y,\ \partial_y,\ x\,\partial_t + t\,\partial_x,\ \partial_x\right] \qquad (1.8)$$

KV is a list of vectors (expressed as derivatives), X, such that the Lie derivative of the metric with respect to X is zero. We can easily test this:

```
M > LieDerivative(KV,g);
```

$$[0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt,\ 0\,dt \otimes dt] \qquad (1.9)$$

To turn these vectors into a Lie algebra, we will need to load the LieAlgebras package. Then we can pass KV to the LieAlgebraData command, which will
calculate the structure constants and other internal data for the Lie algebra. Finally, we initialize the Lie algebra with the DGsetup command:

```
M > with(LieAlgebras):
M > LD := LieAlgebraData(KV,isometry_alg); # Calculate, among
    other things, the structure constant.
     # The second argument will be the algebra name.
```

$$LD := [e1, e2] = e5,\ [e1, e3] = -e9,\ [e1, e4] = -e6,\ [e1, e5] = e2,\ [e1, e6] = -e4,\ [e1, e7] = 0,\ [e1, e8] = 0,\ [e1, e9] = -e3,\ [e1, e10] = 0,\ [e2, e3] = 0,\ [e2, e4] = 0,\ [e2, e5] = 0,\ [e2, e6] = -e8,\ [e2, e7] = 0,\ [e2, e8] = 0,\ [e2, e9] = -e10,\ [e2, e10] = 0,\ [e3, e4] = -e7,\ [e3, e5] = e10,\ [e3, e6] = 0,\ [e3, e7] = e4,\ [e3, e8] = 0,\ [e3, e9] = -e1,\ [e3, e10] = -e5,\ [e4, e5] = e8,\ [e4, e6] = -e1,\ [e4, e7] = -e3,\ [e4, e8] = -e5,\ [e4, e9] = 0,\ [e4, e10] = 0,\ [e5, e6] = 0,\ [e5, e7] = 0,\ [e5, e8] = 0,\ [e5, e9] = 0,\ [e5, e10] = 0,\ [e6, e7] = -e9,\ [e6, e8] = e2,\ [e6, e9] = -e7,\ [e6, e10] = 0,\ [e7, e8] = e10,\ [e7, e9] = -e6,\ [e7, e10] = -e8,\ [e8, e9] = 0,\ [e8, e10] = 0,\ [e9, e10] = e2 \qquad (1.10)$$

```
M > DGsetup(LD);
```

Lie algebra: isometry_alg **(1.11)**

Below, we will take a look at the multiplication table for the algebra. For now, this is all we'll do with the isometry algebra. In a later tutorial, we'll look
at how we can classify the Lie algebra we've generated.

```
M > MultiplicationTable(isometry_alg,"LieTable");
```

isometry_alg	$e1$	$e2$	$e3$	$e4$	$e5$	$e6$	$e7$	$e8$	$e9$	$e10$
$e1$	0	$e5$	$-e9$	$-e6$	$e2$	$-e4$	0	0	$-e3$	0
$e2$	$-e5$	0	0	0	0	$-e8$	0	0	$-e10$	0
$e3$	$e9$	0	0	$-e7$	$e10$	0	$e4$	0	$-e1$	$-e5$
$e4$	$e6$	0	$e7$	0	$e8$	$-e1$	$-e3$	$-e5$	0	0
$e5$	$-e2$	0	$-e10$	$-e8$	0	0	0	0	0	0
$e6$	$e4$	$e8$	0	$e1$	0	0	$-e9$	$e2$	$-e7$	0
$e7$	0	0	$-e4$	$e3$	0	$e9$	0	$e10$	$-e6$	$-e8$
$e8$	0	0	0	$e5$	0	$-e2$	$-e10$	0	0	0
$e9$	$e3$	$e10$	$e1$	0	0	$e7$	$e6$	0	0	$e2$
$e10$	0	0	$e5$	0	0	0	$e8$	0	$-e2$	0

(1.12)

One final command we'll talk about is the SubspaceType command. It takes the Killing vectors and the metric and uses these to calculate an induced metric
on the tangent space at a point. It then computes the signature of that induced metric to determine whether the subspace type is Riemannian, Pseudo-Riemannian,
or Null.

```
M > SubspaceType( g, KV,[x=a,y=b,z=c,t=d] );
```

4, *Fail* **(1.13)**

The SubspaceType command failed because the point specified was too generic. This command computes the sign of a certain determinant to decide the subspace
type. We can get an idea of what is causing Maple to fail here by increasing the infolevel of the command:

```
M > infolevel[SubspaceType]:=2;
```

$infolevel_{SubspaceType} := 2$ **(1.14)**

```
M > SubspaceType( g, KV,[x=a,y=b,z=c,t=d] );
The orbit type is "Null" if the induced metric on the orbit is
degenerate
The orbit type is "Riemannian" if the induced metric on the
orbit is positive or negative definite
The orbit type is "PseudoRiemannian" if the induced metric on
the orbit is non-degenerate and indefinite
   The matrix h defining the induced metric is:
   Matrix(4, 4, [[-c^2+d^2,c,d*a,d*b],[c,-1,0,0],[d*a,0,a^2+c^2,
b*a],[d*b,0,b*a,b^2+c^2]])
The determinant of the matrix h is: -c^4*d^2
```

4, *Fail* **(1.15)**

Here, Maple was unable to determine the sign of $-c^4 \cdot d^2$. This is because c or d could be zero, or even complex. We can specify that these parameters are non-zero real numbers by using the assuming command. Note that use of less-than or greater-than symbols implicity declares that the variables are real.

```
M > SubspaceType( g, KV,[x=a,y=b,z=c,t=d] ) assuming c>0 or c<0,
    d>0 or d<0;
The orbit type is "Null" if the induced metric on the orbit is
degenerate
The orbit type is "Riemannian" if the induced metric on the
orbit is positive or negative definite
The orbit type is "PseudoRiemannian" if the induced metric on
the orbit is non-degenerate and indefinite
   The matrix h defining the induced metric is:
   Matrix(4, 4, [[-c^2+d^2,c,d*a,d*b],[c,-1,0,0],[d*a,0,a^2+c^2,
b*a],[d*b,0,b*a,b^2+c^2]])
The determinant of the matrix h is: -c^4*d^2
```

4, "PseudoRiemannian" **(1.16)**

Part 2: Classifying the Minkowski Space Isometry Algebra

In part two, I continue analyzing the Minkowski metric by looking at its isometry algebra in more detail.

First, we'll catch up to where we left off in part one:

```
> # Load packages and set display preferences:
> with(DifferentialGeometry): with(Tensor): with(LieAlgebras):
> Preferences("TensorDisplay", 1):
> interface(typesetting = extended):

> # Set up a manifold:
> DGsetup([t,x,y,z],M,verbose);
```

The following coordinates have been protected:

$[t, x, y, z]$

The following vector fields have been defined and protected:

$[\partial_t, \partial_x, \partial_y, \partial_z]$

The following differential 1-forms have been defined and protected:

$[dt, dx, dy, dz]$

frame name: M **(2.1)**

```
M > # Define the Minkowski metric:
M > g := evalDG( dx &t dx + dy &t dy + dz &t dz - dt &t dt );
```

$$g := -dt \otimes dt + dx \otimes dx + dy \otimes dy + dz \otimes dz \tag{2.2}$$

```
M > # Get the Killing vectors:
M > KV := KillingVectors(g);
```

$$KV := \left[z\,\partial_t + t\,\partial_z,\ -\partial_t,\ -z\,\partial_x + x\,\partial_z,\ -z\,\partial_y + y\,\partial_z,\ \partial_z,\ y\,\partial_t + t\,\partial_y,\ -y\,\partial_x + x\,\partial_y,\ \partial_y,\ x\,\partial_t + t\,\partial_x,\ \partial_x\right] \tag{2.3}$$

```
M > # Get the Lie algebra data and call it isometry_alg:
M > LD := LieAlgebraData(KV, isometry_alg);
```

$$LD := [e1, e2] = e5,\ [e1, e3] = -e9,\ [e1, e4] = -e6,\ [e1, e5] = e2,\ [e1, e6] = -e4,\ [e1, e7] = 0,\ [e1, e8] = 0,\ [e1, e9] = -e3,\ [e1, e10] = 0,\ [e2, e3] = 0,\ [e2, e4] = 0,\ [e2, e5] = 0,\ [e2, e6] = -e8,\ [e2, e7] = 0,\ [e2, e8] = 0,\ [e2, e9] = -e10,\ [e2, e10] = 0,\ [e3, e4] = -e7,\ [e3, e5] = e10,\ [e3, e6] = 0,\ [e3, e7] = e4,\ [e3, e8] = 0,\ [e3, e9] = -e1,\ [e3, e10] = -e5,\ [e4, e5] = e8,\ [e4, e6] = -e1,\ [e4, e7] = -e3,\ [e4, e8] = -e5,\ [e4, e9] = 0,\ [e4, e10] = 0,\ [e5, e6] = 0,\ [e5, e7] = 0,\ [e5, e8] = 0,\ [e5, e9] = 0,\ [e5, e10] = 0,\ [e6, e7] = -e9,\ [e6, e8] = e2,\ [e6, e9] = -e7,\ [e6, e10] = 0,\ [e7, e8] = e10,\ [e7, e9] = -e6,\ [e7, e10] = -e8,\ [e8, e9] = 0,\ [e8, e10] = 0,\ [e9, e10] = e2 \tag{2.4}$$

```
M > # Initialize the Lie algebra:
M > DGsetup(LD);
```

$$\text{Lie algebra: isometry\_alg} \tag{2.5}$$

```
M > # This is optional, but it can help to visualize the Lie
    algebra with the MultiplicationTable command:
M > MultiplicationTable(isometry_alg, "LieTable");
```

isometry_alg	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
e1	0	e5	− e9	− e6	e2	− e4	0	0	− e3	0
e2	− e5	0	0	0	0	− e8	0	0	− e10	0
e3	e9	0	0	− e7	e10	0	e4	0	− e1	− e5
e4	e6	0	e7	0	e8	− e1	− e3	− e5	0	0
e5	− e2	0	− e10	− e8	0	0	0	0	0	0
e6	e4	e8	0	e1	0	0	− e9	e2	− e7	0
e7	0	0	− e4	e3	0	e9	0	e10	− e6	− e8
e8	0	0	0	e5	0	− e2	− e10	0	0	0
e9	e3	e10	e1	0	0	e7	e6	0	0	e2
e10	0	0	e5	0	0	0	e8	0	− e2	0

(2.6)

Now that we've caught up, we can start analyzing this Lie algebra. The first step (unless the algebra is trivial enough to identify on sight) is to perform a Levi decomposition.
This splits the Lie algebra into a solvable part (the radical, usually labeled r) and a semi-simple part

(usually labeled s):

```
M > LDec := LeviDecomposition(isometry_alg);
```

$$LDec := [[e2, e5, e8, e10], [e1, e3, e4, e6, e7, e9]] \quad (2.7)$$

We can check that the first part is solvable and the second is semi-simple by using the Query command:

```
isometry_alg > Query(LDec[1],"Solvable"); Query(LDec
               [2],"Semisimple");
```

$$true$$

$$true \quad (2.8)$$

We will examine the solvable part, or radical, first. The solvable Lie algebras are usually much more difficult to classify than the semi-simple algebras, but in this case, we're in luck.

```
isometry_alg > LDr := LieAlgebraData(LDec[1],r);
```

$$LDr := [e1, e2] = 0, [e1, e3] = 0, [e1, e4] = 0, [e2, e3] = 0, [e2, e4] = 0, [e3, e4] = 0 \quad (2.9)$$

The solvable part has all zero structure constants, so this is the four-dimensional Abelian Lie algebra. Notice that the vectors were relabeled when we called LieAlgebraData;
the vectors that are called e2, e5, e8, and e10 in the isometry algebra are now called e1, e2, e3, and e4 within the radical of the isometry algebra.

Now we will move on to the semi-simple part of our isometry algebra.

```
isometry_alg > LDs := LieAlgebraData(LDec[2],s);
```

$$LDs := [e1, e2] = -e6, [e1, e3] = -e4, [e1, e4] = -e3, [e1, e5] = 0, [e1, e6] = -e2, [e2, e3] = -e5, [e2, e4] = 0, [e2, e5] = e3, [e2, e6] = -e1, [e3, e4] = -e1, [e3, e5] = -e2, [e3, e6] = 0, [e4, e5] = -e6, [e4, e6] = -e5, [e5, e6] = -e4 \quad (2.10)$$

Again, LieAlgebraData has relabeled the vectors. We can fix this if we want to keep track of which vectors are which by giving DGsetup a list of names for the vectors and
one forms. If we are content with the relabeling, we by no means have to supply these lists. However, if we supply one list, we must supply the other.

```
isometry_alg > DGsetup(LDs,['e1', 'e3', 'e4', 'e6',
               'e7', 'e9'],['o1', 'o3', 'o4', 'o6',
               'o7', 'o9']);
```

$$Lie\ algebra:\ s \quad (2.11)$$

```
isometry_alg > MultiplicationTable(s,"LieTable");
```

(2.12)

s	e1	e3	e4	e6	e7	e9
e1	0	− e9	− e6	− e4	0	− e3
e3	e9	0	− e7	0	e4	− e1
e4	e6	e7	0	− e1	− e3	0
e6	e4	0	e1	0	− e9	− e7
e7	0	− e4	e3	e9	0	− e6
e9	e3	e1	0	e7	e6	0

(2.12)

Now we can begin the classification algorithm. There are several steps, which are outlined below with some explanation. This is barely enough information to carry out the algorithm, much less understand it, but it should be enough to give some vague idea of what's going on.

1. **Find a Cartan subalgebra, h, of s**. This is a nilpotent self-generating subalgebra. In the case of semi-simple Lie algebras, all Cartan subalgebras are abelian, and some sense, they may be thought of as maximal abelian subalgebras.

2. **Find the root space decomposition (RSD) of s with respect to h**. This one requires a little more background. First, if we fix a vector in a Lie algebra, it can act on all other vectors linearly via the Lie bracket, and thus has a matrix representation. This matrix can be defined for each vector in a Lie algebra, and is called the adjoint of that vector. The adjoint matrices of the basis vectors of h are diagonalizable and have a mutual set of eigenvectors. Each eigenvector therefore has associated with it a list of eigenvalues (called a root) that corresponds to the list of basis vectors in h. Each root has one and only one eigenvector associated with it and vice-versa. The RSD splits a semi-simple Lie algebra into a Cartan subalgebra plus the eigenvalues associated with each root. The relationship between the roots completely characterizes the Lie algebra.

3. **Select the positive simple roots**. As mentioned above, the roots obey certain properties. One such property is that if a simultaneous eigenvector of the adjoint matrices of the Cartan subalgebra has the eigenvalue list [a1, a2, . . ., ar], then there is another simultaneous eigenvector with an eigenvalue list [-a1,-a2, . . ., -ar]. Therefore, there is a degree of overspecification in the RSD. If we pick half the roots to be positive (there cannot be an all-zero root for a semi-simple Lie algebra), then the other half are determined.
Additionally, it is possible that some of the positive roots are positive integer linear combinations of the others. Those that are not positive integer linear combinations of the others are called simple. The additive relationships between positive simple roots and the rest of the roots is a unique basis-independent characterization of any semi-simple Lie algebra.

4. **Put into standard form**. The additive relationships between positive simple roots and all other roots can be compactly represented in a Cartan matrix. There are only a few possible Cartan matrices, and they are given a letter (A, B, C, up to G) and a subscript for their dimension (e.g. B_2). This completely characterizes the semi-simple Lie algebra

as a complex Lie algebra (i.e., any complex change of basis vectors for the Lie algebra leaves the Cartan matrix unchanged). Since isometry algebras correspond to real variables, we would like to only allow real changes in basis, so there is one more step, once we know which complex Lie algebra we have.

5. **Compute the signature of the Killing form**. For semi-simple Lie algebras of low enough dimension, each real form of a complex Lie algebra has a unique Killing form signature. Luckily, the largest possible isometry algebra we can have is dimension 10, so the semi-simple part will always have low enough dimension for this trick to work. The down-side, however, is that as of now, Maple doesn't have this information programmed in, so you either have to look it up yourself, or simply know which algebras have which Killing form signatures.

Let's apply these steps to our semi-simple algebra, s:

```
isometry_alg > h := CartanSubalgebra(s); # Step 1
```

$$h := [e1, e7] \qquad (2.13)$$

```
s > RSD := RootSpaceDecomposition(h); # Step 2
```

$$RSD := \mathrm{table}([[-1, -I] = e3 - I\, e4 - I\, e6 + e9, [-1, I] = e3 + I\, e4 + I\, e6 + e9, [1, -I] = e3 - I\, e4 + I\, e6 - e9, [1, I] = e3 + I\, e4 - I\, e6 - e9]) \qquad (2.14)$$

```
s > PR := PositiveRoots(RSD); # Step 3
```

$$PR := \left[\left[\begin{array}{c} 1 \\ -I \end{array}\right], \left[\begin{array}{c} 1 \\ I \end{array}\right]\right] \qquad (2.15)$$

```
s > SR := SimpleRoots(PR); # Step 3 continued
```

$$SR := \left[\left[\begin{array}{c} 1 \\ -I \end{array}\right], \left[\begin{array}{c} 1 \\ I \end{array}\right]\right] \qquad (2.16)$$

```
s > C:=CartanMatrix(SR, RSD); # Step 4; This matrix corresponds
    to B2 or A1 + A1
```

$$C := \left[\begin{array}{cc} 2 & 0 \\ 0 & 2 \end{array}\right] \qquad (2.17)$$

```
s > KF := KillingForm(s); # Step 5
```

$$KF := 4\, o1 \otimes o1 - 4\, o3 \otimes o3 - 4\, o4 \otimes o4 + 4\, o6 \otimes o6 - 4\, o7 \otimes o7 + 4\, o9 \otimes o9 \qquad (2.18)$$

```
s > QuadraticFormSignature(KF,output="dimensions"); # Step 5
    continued
```

$$[3, 3, 0] \qquad (2.19)$$

From the quadratic form siganture and the Cartan matrix, we can determine (either from a table or from memory) that the semi-simple part of our Lie algebra is in fact SO(3,1).
Therefore, our isometry algebra is the semi-direct sum of the four-dimensional abelian Lie algebra with SO(3,1).

Part 3: Examples using MetricSearch

In part three, I go over how to use MetricSearch to retrieve cataloged metrics and analyze a couple of example metrics.

We will start by loading in the packages we'll need and setting preferences. The new packages in this tutorial are the Library package, which contains a catalog
of metrics, and the Tools package, which will help us classify solvable Lie algebras.

```
> with(DifferentialGeometry):
> with(Tensor):
> with(LieAlgebras):
> with(Library): with(Tools):
> Preferences("TensorDisplay", 1):
> interface(typesetting = extended):
```

We will use MetricSearch command to get a list of metrics with certain properties. The below command will return a list of Einstein Maxwell metrics with
isometry dimension 4. A list of search options can be found in the help file. Alternatively, MetricSearch can be called without arguments to launch a graphical
version.

```
> Results := MetricSearch(["PrimaryDescription" =
  "EinsteinMaxwell", "IsometryDimension" = 4]);
Results := [["HawkingEllis", 1, [5, 26, 1]], ["Stephani", 1, [12, 21, 1]], ["Stephani", 1,      (3.1)
    [13, 46, 1]], ["Stephani", 1, [13, 48, 1]], ["Stephani", 1, [15, 21, 1]], ["Stephani", 1,
    [15, 21, 2]], ["Stephani", 1, [15, 27, 2]], ["Stephani", 1, [15, 27, 3]], ["Stephani", 1,
    [15, 27, 5]], ["Stephani", 1, [15, 27, 6]], ["Stephani", 1, [28, 44, 1]], ["Stephani", 1,
    [28, 44, 2]], ["Stephani", 1, [28, 44, 3]], ["Stephani", 1, [28, 44, 4]], ["Stephani", 1,
    [28, 44, 5]], ["Stephani", 1, [28, 44, 6]]]
```

To initialize one of these results, we use the Retrieve command. We can get a list of what parameters and features are recorded by calling Retrieve with the
search result as the only argument.

```
> Retrieve(op(Results[2]));
```

$$\text{table}\left(\left[\text{"PlebanskiPetrovType"} = \text{"D"}, \text{"SegreType"} = \text{"[(1,1)(11)]"}, \text{"OrthonormalTetrad"} = \left[[1,0,0,0], \left[0, \frac{\_x}{\_a}, 0, 0\right], \left[0, 0, \frac{\_x}{\_a}, 0\right], \left[\frac{2\,\_y}{\_x}, 0, 0, \frac{1}{\_x}\right]\right], \text{"Bugs"} \right.\qquad (3.2)$$

= ["Electromagnetic field in 12.22 is incorrect"], "IsometryDimension" = 4, "Authors"

= ["McLenaghan, Tariq (1975)", "Tupper (1976)"], "SecondaryDescription"

= ["Homogeneous", "SimpleTransitive"], "BasePoints" = [[_t = 0, _x = 1, _y = 0, _φ

= 0]], "Metric" = $\left[[-1, 0, 0, 2\_y], \left[0, \frac{\_a^2}{\_x^2}, 0, 0\right], \left[0, 0, \frac{\_a^2}{\_x^2}, 0\right], [2\_y, 0, 0, \_x^2 - 4\_y^2] \right]$, "SimplifyEval" = [[ ]], "Parameters" = [_a, _k, _κ0], "PetrovType" = "I",

"ElectromagneticField" = $\left[\left[0, \frac{2\cos(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}\,\_x}, 0, 0\right], \left[-\frac{2\cos(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}\,\_x}, 0, 0, \frac{4\,\_y\cos(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}\,\_x}\right], \left[0, 0, 0, \frac{2\sin(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}}\right], \left[0, -\frac{4\,\_y\cos(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}\,\_x}, -\frac{2\sin(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}}, 0\right]\right]$, "PrimaryDescription" = "EinsteinMaxwell", "Reference"

= "Stephani", "NewtonConstant" = _κ0, "IsotropyType" = "F15",

"ElectromagneticPotential" = $\left[0, -\frac{2\cos(2\ln(\_k\,\_x))\,(2\,\_\phi\,\_y - \_t)}{\sqrt{\_\kappa0}\,\_x}, -\frac{2\,\_\phi\sin(2\ln(\_k\,\_x))}{\sqrt{\_\kappa0}}, 0\right]$, "Coordinates" = [_t, _x, _y, _φ], "KillingVectors" = [[0,

_x, _y, −_φ], [2 _φ, 0, 1, 0], [0, 0, 0, 1], [1, 0, 0, 0]], "SideConditionsAssuming" = [[0

$< \_a, 0 < \_\kappa 0]]$, "CosmologicalConstant" = 0, "EnergyMomentumTensor"

$$= \left[\left[\frac{2(\_x^2+4\_y^2)}{\_x^2\_\kappa 0\_a^2}, 0, 0, \frac{4\_y}{\_x^2\_\kappa 0\_a^2}\right], \left[0, -\frac{2\_x^2}{\_a^4\_\kappa 0}, 0, 0\right], \left[0, 0, \frac{2\_x^2}{\_a^4\_\kappa 0}, 0\right],\right.$$

$$\left.\left[\frac{4\_y}{\_x^2\_\kappa 0\_a^2}, 0, 0, \frac{2}{\_x^2\_\kappa 0\_a^2}\right]\right], \text{"OrbitType"} = \text{"PseudoRiemannian"}, \text{"NullTetrad"}$$

$$= \left[\left[\frac{\sqrt{2}(2\_y+\_x)}{2\_x}, 0, 0, \frac{\sqrt{2}}{2\_x}\right], \left[\frac{\sqrt{2}(-2\_y+\_x)}{2\_x}, 0, 0, -\frac{\sqrt{2}}{2\_x}\right], \left[0, \frac{\sqrt{2}\_x}{2\_a},\right.\right.$$

$$\left.\left.\frac{\frac{I}{2}\sqrt{2}\_x}{\_a}, 0\right], \left[0, \frac{\sqrt{2}\_x}{2\_a}, \frac{-\frac{I}{2}\sqrt{2}\_x}{\_a}, 0\right]\right], \text{"SideConditionsSimplify"} = [[\ ]],$$

"Domains" = $[[0 < \_x]]$, "OrbitDimension" = 4, "Comments"
= ["_k parametrizes the most general electromagnetic invariant with respect to the last 3 Killing vectors"]])

To actually use this solution, we can call Retrieve with additional options. To start, we will grab the metric, cosomological constant, energy momentum tensor so
we can check that the solution is valid. We will need to name a manifold for the metric to exist on.

```
N1 > g := op(Retrieve(op(Results[2]),manifoldname=N1,output=
      ["Metric"]));
```

$$g := -dt \otimes dt + 2\,y\,dt \otimes d\phi + \frac{\_a^2}{x^2}\,dx \otimes dx + \frac{\_a^2}{x^2}\,dy \otimes dy + 2\,y\,d\phi \otimes dt + \left(x^2 - 4\,y^2\right) d\phi \otimes d\phi \quad (3.3)$$

```
N1 > T := op(Retrieve(op(Results[2]),manifoldname=N1,output=
      ["EnergyMomentumTensor"]));
```

$$T := \frac{2(x^2+4y^2)}{x^2\_\kappa 0\_a^2}\,\partial_t \otimes \partial_t + \frac{4y}{x^2\_\kappa 0\_a^2}\,\partial_t \otimes \partial_\phi - \frac{2x^2}{\_a^4\_\kappa 0}\,\partial_x \otimes \partial_x + \frac{2x^2}{\_a^4\_\kappa 0}\,\partial_y \otimes \partial_y + \frac{4y}{x^2\_\kappa 0\_a^2}\,\partial_\phi \otimes \partial_t + \frac{2}{x^2\_\kappa 0\_a^2}\,\partial_\phi \otimes \partial_\phi \quad (3.4)$$

```
N1 > Lambda := op(Retrieve(op(Results[2]),manifoldname=N1,
      output=["CosmologicalConstant"]));
```

$$\Lambda := 0 \quad (3.5)$$

```
N1 > G := EinsteinTensor(g);
```

$$G := \frac{2(x^2+4y^2)}{x^2\_a^2}\,\partial_t \otimes \partial_t + \frac{4y}{x^2\_a^2}\,\partial_t \otimes \partial_\phi - \frac{2x^2}{\_a^4}\,\partial_x \otimes \partial_x + \frac{2x^2}{\_a^4}\,\partial_y \otimes \partial_y \quad (3.6)$$

$$+ \frac{4 y}{x^2 \_a^2} \partial_\phi \otimes \partial_t + \frac{2}{x^2 \_a^2} \partial_\phi \otimes \partial_\phi$$

We can see that G is proportional to T. In particular, if $\_\kappa 0 = \frac{1}{8\pi}$, the Einstein equations, $G + \Lambda g = 8\pi T$, are satisfied. (Different values for $\_\kappa 0$ correspond to different choices for units).

Many of the properties we might calculate for this metric are already in the Library package. One important exception is the isometry algebra classification. We will now procede to perform that classification. First, we will retrieve the Killing vectors for our metric (we could also compute them directly from the metric):

```
N1 > KV := op(Retrieve(op(Results[2]),manifoldname=N1,output=
     ["KillingVectors"]));
```

$$KV := \left[x \partial_x + y \partial_y - \phi \partial_\phi, 2 \phi \partial_t + \partial_y, \partial_\phi, \partial_t\right] \quad (3.7)$$

Now, we'll compute the structure constants and other data needed to initialize the Lie algebra:

```
N1 > LD1 := LieAlgebraData( KV, alg1 );
```

$$LD1 := [e1, e2] = -e2, [e1, e3] = e3, [e1, e4] = 0, [e2, e3] = -2\, e4, [e2, e4] = 0, [e3, e4] = 0 \quad (3.8)$$

Now, to actually initialize the algebra, we call DGsetup.

```
N1 > DGsetup( LD1 );
```

Lie algebra: alg1 **(3.9)**

It is usually best to start the classification of a Lie algebra by computing the Levi decomposition. However, if the algebra is solvable or semi-simple, the Levi decomposition is reduntant. Computationally, it is relatively inexpensive to check to see if the algebra is solvable or semi-simple, and for large or complicated algebras, the Levi decomposition can be time consuming. It just so happens that we have picked a solvable Lie algebra.

```
N1 > Query(alg1,"Solvable");
```

true **(3.10)**

Solvable Lie algebras are more difficult to classify than semi-simple ones. In this case, however, the structure constants are telling; e4 commutes with all other vectors. While classification algorithms exist for solvable Lie algebras of low dimension, such as this one, they are not implemented in Maple, and I have not written code to do this. Instead, I have written code to narrow down the list of candidate Lie algebras. A complete list of real solvable Lie algebras up to dimension six is included in the Library package. These are in the "Winternitz" resource. As an example, we can retrieve the the fourth three-dimensional solvable (real) Lie algebra:

```
N1 > LD_example := Retrieve("Winternitz",1,[3,4]);
```

$$LD\_example := [e1, e2] = 0, [e1, e3] = e1, [e2, e3] = -e2 \quad (3.11)$$

Below is a method to comb through the algebras listed in the Winternitz source to return a list of Lie algebra candidates. I will demonstrate its use
on the isometry algebra, alg1, that we created earlier. (This code relies on the Tools package to determine the dimension of the Lie algebra.)

```
my_LowD_SLA_explore := proc(alg)
     # This is a tool to help narrow down what solvable Lie Algebra alg
     local SE,i,j,myalg,is_candidate,Candidates,DS,mDS,numAlg,Dim;

     Candidates:=[]:

     Dim := DGinfo(alg,"FrameBaseDimension"):

     # Winternitz doesn't bother with Dim < 3, but I'll pretend for cons
     if Query(alg,"Abelian") then
          return [["Winternitz",1,[Dim,0]]]:
     fi:

     # If it were abelian, then we would've caught it already (only 2 2D
     if Dim = 2 then
          return [["Winternitz",1,[2,1]]]:
     fi:

     # This determined how many algebras in Winternitz we'll have to che
     if Dim = 3 then
          numAlg := 9:
     elif Dim = 4 then
          numAlg := 12:
     elif Dim = 5 then
          numAlg := 39:
     elif Dim = 6 then
          numAlg := 22:
     fi:

     # Isomorphic real Lie algebras have the same
     # - derived series (and therefore derived algebra)
     # - center
```

```
N1 > Candidates := my_LowD_SLA_explore(alg1);
```

$$Candidates := [["Winternitz", 1, [4, 8]]] \qquad (3.12)$$

In this case, we are lucky; only one candidate is returned. We now know that our Lie algebra, alg1, is the eighth Lie algebra in the Winternitz source.
To double-check this, we can initialize it and compare multiplication tables. To avoid confusion, we'll label our retrieved algebra vectors using *f*, rather
than the default *e*.

```
alg1 > LD2 := Retrieve(op(Candidates[1]),alg2);
```

$$LD2 := [e1, e2] = 0, [e1, e3] = 0, [e1, e4] = 0, [e2, e3] = e1, [e2, e4] = e2, [e3, e4] = -e3 \qquad (3.13)$$

```
alg1 > DGsetup(LD2,[f],[o]);
```

$$\text{Lie algebra: } alg2 \qquad (3.14)$$

```
alg1 > MultiplicationTable(alg1,"LieTable");
       MultiplicationTable(alg2,"LieTable");
```

alg1	*e1*	*e2*	*e3*	*e4*
e1	0	− *e2*	*e3*	0
e2	*e2*	0	− 2 *e4*	0
e3	− *e3*	2 *e4*	0	0
e4	0	0	0	0

alg2	*f1*	*f2*	*f3*	*f4*
f1	0	0	0	0
f2	0	0	*f1*	*f2*
f3	0	− *f1*	0	− *f3*
f4	0	− *f2*	*f3*	0

(3.15)

These don't exactly match, but we should be able to perform a simple change of basis to fix that. We'll slightly parameterize the change of basis and use the fact
that one of the basis vectors is in the center of the algebra.

```
alg2 > LD1a := LieAlgebraData([e4, a*e1, b*e2, c*e3],alg1a);
```

$$LD1a := [e1, e2] = 0, [e1, e3] = 0, [e1, e4] = 0, [e2, e3] = -a\, e3, [e2, e4] = a\, e4, [e3, e4] = -2\, b\, c\, e1 \qquad (3.16)$$

```
alg1 > LD2;
```

$$[e1, e2] = 0, [e1, e3] = 0, [e1, e4] = 0, [e2, e3] = e1, [e2, e4] = e2, [e3, e4] = -e3 \qquad (3.17)$$

With $a = b = 1$ and $c = \frac{-1}{2}$, we get very close; the vectors are just labeled in a slightly different order.

```
alg1 > LD1a := LieAlgebraData([e4, e2, (-1/2)*e3, e1],alg1a);
```

$LD1a := [e1, e2] = 0, [e1, e3] = 0, [e1, e4] = 0, [e2, e3] = e1, [e2, e4] = e2, [e3, e4] = -e3$ **(3.18)**

Now the multiplication tables look the same so the result is verified:

```
alg1 > DGsetup(LD1a);
```

Lie algebra: alg1a **(3.19)**

```
alg1 > MultiplicationTable(alg1a,"LieTable");
       MultiplicationTable(alg2,"LieTable");
```

alg1a	$e1$	$e2$	$e3$	$e4$
$e1$	0	0	0	0
$e2$	0	0	$e1$	$e2$
$e3$	0	$-e1$	0	$-e3$
$e4$	0	$-e2$	$e3$	0

alg2	$f1$	$f2$	$f3$	$f4$
$f1$	0	0	0	0
$f2$	0	0	$f1$	$f2$
$f3$	0	$-f1$	0	$-f3$
$f4$	0	$-f2$	$f3$	0

(3.20)

Part 4: Example Codes

In part four, I provide the code, with minimal introduction, that I used to automate the procedures discussed in the previous tutorials.
For explanations of all the pieces, see the previous three tutorials.

We will start by loading in the packages we'll need and setting preferences as always.

```
> with(DifferentialGeometry):
> with(Tensor):
> with(LieAlgebras):
> with(Library): with(Tools):
> Preferences("TensorDisplay", 1):
> interface(typesetting = extended):
```

The below code is used for narrowing down candidates for solvable Lie algebras of low dimension and was used in Tutorial 3.

```
my_LowD_SLA_explore := proc(alg)
     # This is a tool to help narrow down what solvable Lie Algebra alg
     local SE,i,j,myalg,is_candidate,Candidates,DS,mDS,numAlg,Dim;

     Candidates:=[]:

     Dim := DGinfo(alg,"FrameBaseDimension"):

     # Winternitz doesn't bother with Dim < 3, but I'll pretend for cons
     if Query(alg,"Abelian") then
          return [["Winternitz",1,[Dim,0]]]:
     fi:

     # If it were abelian, then we would've caught it already (only 2 2D
     if Dim = 2 then
          return [["Winternitz",1,[2,1]]]:
     fi:

     # This determined how many algebras in Winternitz we'll have to che
     if Dim = 3 then
          numAlg := 9:
     elif Dim = 4 then
          numAlg := 12:
     elif Dim = 5 then
          numAlg := 39:
     elif Dim = 6 then
          numAlg := 22:
     fi:

     # Isomorphic real Lie algebras have the same
     # - derived series (and therefore derived algebra)
     # - center
```

This is an automated version of the method used to classify semi-simple Lie algebras.

```
SSClass := proc(alg)
     # Returns the Cartan Matrix for a semi-simple Lie algebra
     local CSA, RSD, PR, SR, CM:
     CSA := CartanSubalgebra(alg):
     RSD := RootSpaceDecomposition(CSA):
     PR  := PositiveRoots(RSD):
     SR  := SimpleRoots(PR):  CM  := CartanMatrix(SR,RSD):
     return CM;
end proc:
```

The code below takes a Lie algebra and performs a Levi-decomposition, then uses the two codes above. This is
essentially the procedure followed in Tutorial 3.

```
LieClassHelper := proc(alg)
     # This will help classify Lie Algebras
     local LDec, alg_ss, alg_sol, LD_ss, LD_sol, CM, Candidates:

     LDec := LeviDecomposition(alg):

     if nops(LDec[1]) > 0 then
          LD_sol := LieAlgebraData(LDec[1],alg_sol):
          DGsetup(LD_sol):
          Candidates := my_LowD_SLA_explore(alg_sol):
     else
          # Winternitz doesn't bother with Dim < 3, but I'll pretend for
          Candidates := [["Winternitz",1,[0,0]]]:
     fi:

     if nops(LDec[2]) > 2 then
          LD_ss := LieAlgebraData(LDec[2],alg_ss):
          DGsetup(LD_ss):
          CM := SSClass(alg_ss):
     else
          CM := Matrix([]):
     fi:

     return [Candidates, CM]:

end proc:
```

The code below calculates several metric properties of interest, as are outlined in Tutorial 1.

```
BlackBoxMetric:= proc( g, Lambda )
     # Takes a metric, g, and a cosmological constant, Lambda.
     # Returns a list containing:
     # - Curvature Tensor
     # - Einstein Tensor,
     # - Energy Momentum Tensor (assuming that the Einstein equations ar
     # - Petrov Type
     # - Segre Type
     # - Killing Vectors
     # - Subspace Type (note: this often fails without additional user i
     local R, G, PT, K, K_i, ST, T, g_inv, Segre, ans:

     R:=CurvatureTensor( g ):
     print( "R", R );

     G:=EinsteinTensor( g ):
     print( "G", G );

     if not( Lambda = 0 ) then
          g_inv:=InverseMetric( g ):
     else
          g_inv:=g: # if Lambda is 0, then the inverse metric
                    # doesn't matter (for now!), so no reason to find it
     fi:

     # T is the energy-momentum tensor given by g and Lambda
     T:=simplify( evalDG( ( G + Lambda * g_inv ) / ( 8 * Pi ) ) ):
     print( "T", T );

```