PHY483 Problem Set 2

Friday, October 18, 2024

Jace Alloway - 1006940802 - alloway1

Problem 1

(a) *An explanatory derivation of the Laplacian in spherical coordinates:*

$$\nabla_i \nabla^i \Psi(r,\theta,\varphi) = \left[\frac{\partial^2 r}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r} + \frac{1}{r}\left(\frac{\partial^2}{\partial \theta^2} + \frac{\cos\theta}{\sin\theta}\frac{\partial}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\varphi^2}\right)\right]\Psi(r,\theta,\varphi). \tag{1.1}$$

Introduction of curvilinear coordinates and basis vectors: Let us represent spatial dimensions with latin indices $(i, j, k, \dots)$. A set of basis vectors is any collection of basis vectors which span a coordinate space. You will recall this from working in linear algebra and in flat Minkowski spacetime coordinates, where the basis vectors are identical at every point (ie. $\hat{\mathbf{t}}, \hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$). In curvilinear coordinates, basis vectors are not the same at each point. This is simply because, in a coordinate change, the tangent space of all vectors at a point P on a manifold will be different at different locations, say, at a nearby point $P + \delta P$. This is initiated by the curvature of the surface, hence the basis vectors which govern our coordinate system will not always be identical depending on our location.

Under a curvilinear change of variables, we can write

$$x^i = f(x^{0'}, x^{1'}, \dots) \tag{1.2}$$

where the old coordinates are functions of the new coordinates. From (1.2), under this transformation, we see that the differentials of the coordinates x^i transform as

$$dx^i = \frac{\partial f}{\partial x^{j'}}\, dx^{j'} \tag{1.3}$$

A line element $d\mathbf{s}$ in one coordinate basis expanded as

$$d\mathbf{s} = \mathbf{e}_i dx^i \tag{1.4}$$

would be expanded in the transformed basis respectively as

$$d\mathbf{s} = \mathbf{e}_i{}' dx^{i'}. \tag{1.5}$$

Thus, for a vector quantity $d\mathbf{s}$ expanded in term of the old basis vectors $\{\mathbf{e}^i\}$, we write the new basis vectors as

$$\mathbf{e}_j{}' = \frac{\partial \mathbf{s}}{\partial x^{j'}} \tag{1.6}$$

provided that $\mathbf{e}_i = \dfrac{\partial \mathbf{s}}{\partial x^i}$ in the old basis. The magnitude of this line element is taken to be

$$ds^2 = \mathbf{e}_i dx^i \cdot \mathbf{e}_j dx^j = \mathbf{e}_i{}' dx^{i'} \cdot \mathbf{e}_j{}' dx^{j'} \tag{1.7}$$

where we have forced the ds^2 to be invariant regardless of coordinate basis. This is called a diffeomorphism in a curvilinear change of basis (ie. spacetime), and is a class C^∞ mapping and is invertible. In light of this, our metric tensor g_{ij} will not always be equivalent to the Minkowski metric η_{ij}. From (1.6) (or (1.7)), we obtain the expression for the metric tensor g_{ij},

$$\begin{aligned} ds^2 &= \mathbf{e}_i \cdot \mathbf{e}_j dx^i dx^j && (1.8)\\ &= g_{ij} dx^i dx^j && (1.9)\\ \implies g_{ij} &= \mathbf{e}_i \cdot \mathbf{e}_j && (1.10)\\ \implies ds^2 &= g_{ij} dx^{i'} dx^{j'}. && (1.11)\end{aligned}$$

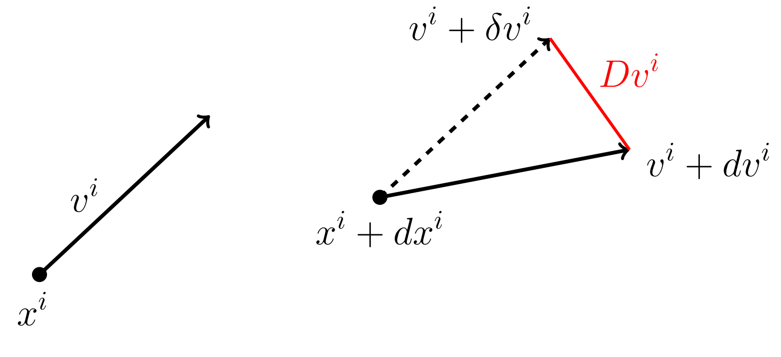
The contravariant metric g^{ij} is equivalently defined as the inverse of g_{ij} with the inverse basis vectors, so that

$$\begin{aligned} g^{ij} &= \mathbf{e}^i \cdot \mathbf{e}^j && (1.12)\\ \implies g_{ij} g^{ik} &= (\mathbf{e}_i \cdot \mathbf{e}_j)(\mathbf{e}^i \cdot \mathbf{e}^k) = \delta^k_j && (1.13)\end{aligned}$$

where we obtain (1.13) via linearity. Just like any other change of basis, one may introduce a diagonalization to g_{ij} so that there are new basis vectors defined in a so-called 'orthonormal basis'. To raise and lower indices between contravariant and covariant vectors or tensors, we must apply g_{ij} to transform according to the curvature, instead of η_{ij} in flat space. That is, $v^i = g^{ij} v_j$ and vice-versa. We note that these expansions are similar for contravariant and covariant vectors.

Derivatives in curvilinear coordinates: Since basis vectors change under a change of basis, one must also define an 'invariant derivative' (commonly called a 'covariant derivative') which is a derivative operation which will act on the new coordinates. Instead of acting in the vector tangent space at each coordinate, which will be different at each point, we aim to express a derivative function which acts 'in the manifold' of the curvature, hence that changes as position changes. We must first investigate how basis vectors change along a direction q in a curvilinear basis.

To assist in our derivation, suppose there is a vector v^i in the at point x^i in a curved basis. Now suppose that, at a neighbouring point $x^i + dx^i$, that same vector (transformed according to the manifold curvature) is $v^i + dv^i$. Performing a direct translation (a 'parallel transport') of v^i at x^i to $x^i + dx^i$ without changing the orientation of v^i, the resulting change in the translated vector is δv^i.



Thus, the difference between the two vectors located at the same point $x^i + dx^i$ is

$$Dv^i = dv^i - \delta v^i, \tag{1.14}$$

known as the absolute differential. Since the sum of two vectors must transform like a vector, with linear dependence, δv^i must transform as

$$dv^i = \frac{\partial v^i}{\partial x^j} dx^j = \partial_j v^i \, dx^j, \tag{1.15}$$

hence we can write the expansion

$$\delta v^i = \Gamma^i_{kl} v^k dx^l \tag{1.16}$$

where the Γ^i_{kl} are coefficients called 'connection coefficients' or 'Christoffel symbols' and are functions of the coordinates. We write this to ensure linear dependence of vector transformations in terms of the v^k components and the dx^l differentials over space. Hence we write the covariant (total) derivative of a vector quantity as

$$\frac{Dv^i}{Dx^l} \equiv D_l v^i = \partial_l v^i - \Gamma^i_{kl} v^k. \tag{1.17}$$

For the vector v^i written as it's contraction with the basis vectors, $\mathbf{v} = v^i \mathbf{e}_i$, we find that the basis vectors change (following from (1.17) and the product rule on $D_l(v^i \mathbf{e}_i)$, just a contraction with $\mathbf{e}_i$) as

$$\partial_l \mathbf{e}_k = \Gamma^i_{kl} \mathbf{e}_i \tag{1.18}$$

and similarly for a covariant vector,

$$\partial_l \mathbf{e}^k = -\Gamma^k_{jl} \mathbf{e}^j \tag{1.19}$$

where the last step also follows from orthogonality, that $\mathbf{e}^i \cdot \mathbf{e}_j = \delta^i_j$.

In connection to the metric in (1.12), one may express the connection coefficients in terms of the basis vectors or metric via the covariant derivative:

$$\begin{aligned} \partial_l g_{ij} &= (\partial_l \mathbf{e}_i) \cdot \mathbf{e}_j + \mathbf{e}_i \cdot (\partial_l \mathbf{e}_j) && (1.20) \\ &= \Gamma^k_{li} \mathbf{e}_k \cdot \mathbf{e}_j + \mathbf{e}_i \cdot \Gamma^k_{lj} \mathbf{e}_k && (1.21) \end{aligned}$$

from which one can derive that

$$\Gamma^i_{jk} = \frac{1}{2} g^{il} (\partial_j g_{kl} + \partial_k g_{jl} - \partial_l g_{jk}) \tag{1.22}$$

In short, the covariant derivative involves how the vector itself changes, plus how the surrounding curvature changes according to the change in the basis vectors (this is analogous to the material derivative in fluid mechanics).

From Cartesian to spherical coordinates: Let us consider the change of basis from cartesian coordinates to spherical coordinates:

$$\begin{aligned} x &= r \sin\theta \cos\varphi \\ y &= r \sin\theta \sin\varphi \\ z &= r \cos\theta && (1.23) \end{aligned}$$

where the old coordinates have been expanded as functions of the new coordinates. By taking differentials as in (1.3), we can derive the expression for the transformed unit vectors in the new basis:

$$\begin{aligned} dx &= \sin\theta \cos\varphi \, dr + r \cos\theta \cos\varphi \, d\theta - r \sin\theta \sin\varphi \, d\varphi \\ dy &= \sin\theta \sin\varphi \, dr + r \cos\theta \sin\varphi \, d\theta + r \sin\theta \cos\varphi \, d\varphi \\ dz &= \cos\theta \, dr - r \sin\theta \, d\theta && (1.24) \end{aligned}$$

hence, by inverting the transformation matrix to write $(dr, d\theta, d\varphi)$ in terms of (dx, dy, dz),

$$\begin{aligned}
dr &= \sin\theta\cos\varphi\,dx + \sin\theta\sin\varphi\,dy + \cos\theta\,dz \\
d\theta &= \frac{1}{r}\cos\theta\cos\varphi\,dx + \frac{1}{r}\cos\theta\sin\varphi\,dy - \frac{1}{r}\sin\theta\,dz \\
d\varphi &= -\frac{1}{r\sin\theta}\sin\varphi\,dx + \frac{1}{r\sin\theta}\cos\varphi\,dy \qquad (1.25)
\end{aligned}$$

We then find the representation of the basis vectors from one set of coordinates to another according to equation (1.6),

$$\begin{aligned}
\hat{\mathbf{r}} &= \sin\theta\cos\varphi\,\hat{\mathbf{x}} + \sin\theta\sin\varphi\,\hat{\mathbf{y}} + \cos\theta\,\hat{\mathbf{z}} \\
\hat{\boldsymbol{\theta}} &= \frac{1}{r}\cos\theta\cos\varphi\,\hat{\mathbf{x}} + \frac{1}{r}\cos\theta\sin\varphi\,\hat{\mathbf{y}} - \frac{1}{r}\sin\theta\,\hat{\mathbf{z}} \\
\hat{\boldsymbol{\varphi}} &= -\frac{1}{r\sin\theta}\sin\varphi\,\hat{\mathbf{x}} + \frac{1}{r\sin\theta}\cos\varphi\,\hat{\mathbf{y}}. \qquad (1.26)
\end{aligned}$$

The line element ds^2 can then be expanded from (1.7) and from collecting like terms in (1.25), which is

$$\begin{aligned}
ds^2 &= dx^2 + dy^2 + dz^2 && (1.27) \\
&= (\hat{\mathbf{x}}\cdot\hat{\mathbf{x}})\,dr^2 + (\hat{\mathbf{y}}\cdot\hat{\mathbf{y}})\,d\theta^2 + (\hat{\mathbf{z}}\cdot\hat{\mathbf{z}})\,d\varphi^2 && (1.28) \\
&= (\sin^2\theta\cos^2\varphi + \sin^2\theta\sin^2\varphi + \cos^2\theta)\,dr^2 \\
&\qquad + r^2(\cos^2\theta\cos^2\varphi + \cos^2\theta\sin^2\varphi + \sin^2\theta)\,d\theta^2 \\
&\qquad + r^2\sin^2\theta(\sin^2\varphi + \cos^2\varphi)\,d\varphi^2 && (1.29) \\
&= dr^2 + r^2\,d\theta^2 + r^2\sin^2\theta\,d\varphi^2 && (1.30)
\end{aligned}$$

hence the metric is already diagonalized, since we have required the unit vectors in the new basis to be orthogonal (that is, $\hat{\mathbf{r}}\cdot\hat{\boldsymbol{\theta}} = \hat{\mathbf{r}}\cdot\hat{\boldsymbol{\varphi}} = \hat{\boldsymbol{\varphi}}\cdot\hat{\boldsymbol{\theta}} = 0$) and the covariant form of the metric tensor, from (1.11), is

$$g_{ij} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & r^2\sin^2\theta \end{pmatrix}. \qquad (1.31)$$

This metric then tells us how vectors scale under the coordinate transformation. The inverse is equivalently

$$g^{ij} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & \frac{1}{r^2\sin^2\theta} \end{pmatrix}. \qquad (1.32)$$

Since each of the new basis vectors are defined as functions of the new coordinates, they transform in proportion to the connection coefficients in (1.19). These can be derived from taking derivatives in each variable from (1.26) and re-writing the derivated vector in terms of the same basis vectors:

$$\begin{aligned}
\frac{\partial\hat{\mathbf{r}}}{\partial r} &= -\frac{\partial}{\partial r}[\sin\theta\cos\varphi\,\hat{\mathbf{x}} + \sin\theta\sin\varphi\,\hat{\mathbf{y}} + \cos\theta\,\hat{\mathbf{z}}] \\
&= 0 \qquad (1.33) \\
\frac{\partial\hat{\mathbf{r}}}{\partial\theta} &= -(\cos\theta\cos\varphi\,\hat{\mathbf{x}} + \cos\theta\sin\varphi\,\hat{\mathbf{y}} - \sin\theta\,\hat{\mathbf{z}})
\end{aligned}$$

$$
\begin{aligned}
&= -r\,\hat{\boldsymbol{\theta}} && (1.34)\\
\frac{\partial \hat{\mathbf{r}}}{\partial \varphi} &= -(-\sin\theta\sin\varphi\,\hat{\mathbf{x}} + \sin\theta\cos\varphi\,\hat{\mathbf{y}})\\
&= -r\sin^2\theta\,\hat{\boldsymbol{\varphi}} && (1.35)
\end{aligned}
$$

thus the connection coefficient in r is diagonal, since each derivative in each basis vector yielded a quantity proportional to said basis vector:

$$
\Gamma^r_{ij} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -r & 0 \\ 0 & 0 & -r\sin^2\theta \end{pmatrix}. \tag{1.36}
$$

In a similar fashion, we can derive the connection coefficients in θ and φ as well:

$$
\begin{aligned}
\frac{\partial \hat{\boldsymbol{\theta}}}{\partial r} &= \frac{1}{r}\left[\frac{1}{r}\cos\theta\cos\varphi\,\hat{\mathbf{x}} + \frac{1}{r}\cos\theta\sin\varphi\,\hat{\mathbf{y}} - \frac{1}{r}\sin\theta\,\hat{\mathbf{z}}\right]\\
&= \frac{1}{r}\hat{\boldsymbol{\theta}} && (1.37)\\
\frac{\partial \hat{\boldsymbol{\theta}}}{\partial \theta} &= \frac{1}{r}\sin\theta\cos\varphi\,\hat{\mathbf{x}} - \frac{1}{r}\sin\theta\sin\varphi\,\hat{\mathbf{y}} - \frac{1}{r}\cos\theta\,\hat{\mathbf{z}}\\
&= \frac{1}{r}\hat{\mathbf{r}} && (1.38)\\
\frac{\partial \hat{\boldsymbol{\theta}}}{\partial \varphi} &= \frac{1}{r}\cos\theta\sin\varphi\,\hat{\mathbf{x}} + \frac{1}{r}\cos\theta\cos\varphi\,\hat{\mathbf{y}}\\
&= -\sin\theta\cos\theta\,\hat{\boldsymbol{\varphi}} && (1.39)\\
\frac{\partial \hat{\boldsymbol{\varphi}}}{\partial r} &= \frac{1}{r}\left[-\frac{1}{r\sin\theta}\sin\varphi\,\hat{\mathbf{x}} + \frac{1}{r\sin\theta}\cos\varphi\,\hat{\mathbf{y}}\right]\\
&= \frac{1}{r}\,\hat{\boldsymbol{\varphi}} && (1.40)\\
\frac{\partial \hat{\boldsymbol{\varphi}}}{\partial \theta} &= -\frac{1}{r\sin\theta}\cot\theta\sin\varphi\,\hat{\mathbf{x}} - \frac{1}{r\sin\theta}\cos\theta\cos\varphi\,\hat{\mathbf{y}}\\
&= \cot\theta\,\hat{\boldsymbol{\varphi}} && (1.41)\\
\frac{\partial \hat{\boldsymbol{\varphi}}}{\partial \varphi} &= \frac{1}{r\sin\theta}\cos\varphi\,\hat{\mathbf{x}} + \frac{1}{r\sin\theta}\sin\varphi\,\hat{\mathbf{y}}\\
&= \frac{1}{r\sin\theta}(\sin^2\theta+\cos^2\theta)\cos\varphi\,\hat{\mathbf{x}} + \frac{1}{r\sin\theta}(\sin^2\theta+\cos^2\theta)\sin\varphi\,\hat{\mathbf{y}} - \left(\frac{1}{r}\cos\theta - \frac{1}{r}\cos\theta\right)\hat{\mathbf{z}}\\
&= \frac{1}{r}(\sin\theta\cos\varphi\,\hat{\mathbf{x}} + \sin\theta\sin\varphi\,\hat{\mathbf{y}} + \cos\theta\,\hat{\mathbf{z}}) + \cot\theta\left(\frac{1}{r}\cos\theta\cos\varphi\,\hat{\mathbf{x}} + \frac{1}{r}\cos\theta\sin\varphi\,\hat{\mathbf{y}} - \frac{1}{r}\sin\theta\,\hat{\mathbf{z}}\right)\\
&= \frac{1}{r}\hat{\mathbf{r}} + \cot\theta\,\hat{\boldsymbol{\theta}}. && (1.42)
\end{aligned}
$$

Thus we find the other two connection coefficients to be

$$
\Gamma^\theta_{ij} = \begin{pmatrix} 0 & \frac{1}{r} & 0 \\ \frac{1}{r} & 0 & 0 \\ 0 & 0 & -\sin\theta\cos\theta \end{pmatrix} \tag{1.43}
$$

$$\Gamma^{\varphi}_{ij} = \begin{pmatrix} 0 & 0 & \frac{1}{r} \\ 0 & 0 & \cot\theta \\ \frac{1}{r} & \cot\theta & 0 \end{pmatrix} \tag{1.44}$$

Suppose we now have a scalar field $\Psi(r,\theta,\varphi)$, and we wish to take the Laplacian of this quantity. Since scalars need not transform like vectors, we just have

$$\begin{aligned} D_i D^i \Psi(r,\theta,\varphi) &= D_i(\partial^i \Psi(r,\theta,\varphi)) \\ &= (\partial_i \partial^i + \Gamma^j_{ji}\partial^i)\Psi(r,\theta,\varphi) \end{aligned} \tag{1.45}$$

Hence, taking the derivative of a scalar quantity in spherical coordinates yields

$$\begin{aligned} D_i D^i \Psi(r,\theta,\varphi) &= (\partial_r\partial^r + \partial_\theta\partial^\theta + \partial_\varphi\partial^\varphi)\Psi(r,\theta,\varphi) + (\Gamma^r_{rr}\partial^r + \Gamma^r_{r\theta}\partial^\theta + \Gamma^r_{r\varphi}\partial^\varphi \\ &\quad + \Gamma^\theta_{\theta r}\partial^r + \Gamma^\theta_{\theta\theta}\partial^\theta + \Gamma^\theta_{\theta\varphi}\partial^\varphi + \Gamma^\varphi_{\varphi r}\partial^r + \Gamma^\varphi_{\varphi\theta}\partial^\theta + \Gamma^\varphi_{\varphi\varphi}\partial^\varphi)\Psi(r,\theta,\varphi) && (1.46) \\ &= (\partial_r\partial^r + \partial_\theta\partial^\theta + \partial_\varphi\partial^\varphi)\Psi(r,\theta,\varphi) \\ &\qquad + \left(\frac{1}{r}\partial^r + \frac{1}{r}\partial^r + \cot\theta\partial^\theta\right)\Psi(r,\theta,\varphi) && (1.47) \\ &= \left(\frac{\partial^2}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2}{\partial\theta^2} + \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\varphi^2}\right)\Psi(r,\theta,\varphi) + \left(\frac{2}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\cot\theta\frac{\partial}{\partial\theta}\right)\Psi(r,\theta,\varphi) && (1.47) \\ &= \left[\frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\left(\frac{\partial^2}{\partial\theta^2} + \cot\theta\frac{\partial}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\varphi^2}\right)\right]\Psi(r,\theta,\varphi) && (1.48) \end{aligned}$$

where the last two lines followd from index lowering on the ∂^i's, which is done using the inverse metric in (1.32), $\partial_j = g_{ij}\partial^i$. Hence this is how the Laplacian is obtained using the change of basis in curvilinear coordinates and a covariant derivative.

(b) Using the connection coefficients previously found for spherical coordinates, we can now write the equations of motion for a particle in these coordinates. Starting from the equations of motion obtained to minimizing the action,

$$\frac{d^2x^i}{d\lambda^2} + \Gamma^i_{jk}\frac{dx^j}{d\lambda}\frac{dx^k}{d\lambda} = 0, \tag{1.49}$$

and utilizing the non-zero entries

$$\begin{aligned} \Gamma^r_{\theta\theta} &= -r & \Gamma^r_{\varphi\varphi} &= -r\sin^2\theta \\ \Gamma^\theta_{\theta r} &= \Gamma^\theta_{r\theta} = \frac{1}{r} & \Gamma^\theta_{\varphi\varphi} &= -\sin\theta\cos\theta \\ \Gamma^\varphi_{\varphi r} &= \Gamma^\varphi_{r\varphi} = \frac{1}{r} & \Gamma^\varphi_{\varphi\theta} &= \Gamma^\varphi_{\theta\varphi} = \cot\theta \end{aligned} \tag{1.50}$$

we have

$$\frac{d^2r}{d\lambda^2} - r\left(\frac{d\theta}{d\lambda}\right)^2 - r\sin^2\theta\left(\frac{d\varphi}{d\lambda}\right)^2 = 0 \tag{1.51}$$

$$\frac{d^2\theta}{d\lambda^2} + \frac{2}{r}\frac{dr}{d\lambda}\frac{d\theta}{d\lambda} - \sin\theta\cos\theta\left(\frac{\partial\varphi}{\partial\lambda}\right)^2 = 0 \tag{1.52}$$

$$\frac{d^2\varphi}{d\lambda^2} + \frac{2}{r}\frac{dr}{d\lambda}\frac{d\varphi}{d\lambda} + 2\cot\theta\frac{d\theta}{d\lambda}\frac{d\varphi}{d\lambda} = 0. \tag{1.53}$$

Noting that, in the azimuthal equation (1.53), each term contains a derivative in φ. This implies that some quantity must be conserved along the geodesic path of φ for the net equation to be zero, by Noether's theorem. To investigate this, let's first note that

$$\frac{d}{d\lambda}[\log(r^2\sin^2\theta)] = \frac{d}{d\lambda}[2\log r + 2\log\sin\theta] \tag{1.54}$$

$$= \frac{2}{r}\frac{dr}{d\lambda} + 2\cot\theta\frac{d\theta}{d\lambda} \tag{1.55}$$

by the chain rule. Hence (1.53) can be written as

$$0 = \frac{d}{d\lambda}\frac{d\varphi}{d\lambda} + \frac{d}{d\lambda}[\log(r^2\sin^2\theta)]\cdot\frac{d\varphi}{d\lambda} \tag{1.56}$$

$$= \frac{d}{d\lambda}\frac{d\varphi}{d\lambda} + \frac{1}{r^2\sin^2\theta}\frac{d}{d\lambda}[r^2\sin^2\theta]\cdot\frac{d\varphi}{d\lambda}. \tag{1.57}$$

Distributing the $r^2\sin^2\theta$, we notice that

$$0 = r^2\sin^2\theta\cdot\frac{d}{d\lambda}\left[\frac{d\varphi}{d\lambda}\right] + \frac{d}{d\lambda}[r^2\sin^2\theta]\cdot\frac{d\varphi}{d\lambda} \tag{1.58}$$

$$= \frac{d}{d\lambda}\left[(r^2\sin^2\theta)\frac{d\varphi}{d\lambda}\right] \tag{1.59}$$

hence the quantity

$$r^2\sin^2\theta\frac{d\varphi}{d\lambda} = k \tag{1.60}$$

is conserved along the geodesic (k is a constant). This quantity corresponds to the angular momentum in φ around the curvature of the sphere.

One may now show the solution for a straight line to the equations by implementing (1.60), by means of concatenating (1.51) and (1.52):

$$\frac{1}{r\sin^2\theta}(\ddot{r} - r\dot{\theta}^2) = \frac{1}{\sin\theta\cos\theta}\left(\ddot{\theta} + \frac{2}{r}\dot{r}\dot{\theta}\right) \tag{1.61}$$

$$\implies r\cot\theta(\ddot{r} - r\dot{\theta}^2) = r^2\ddot{\theta} + 2r\dot{r}\dot{\theta} \tag{1.62}$$

$$\implies r\cot\theta(\ddot{r} - r\dot{\theta}^2) = \frac{d}{d\lambda}[r^2\dot{\theta}] \tag{1.63}$$

A straight line in this basis is given for constant $r = R$ and the relation between great circles on the sphere, which is $\pm s\sin\varphi = \cot\theta$. Thus

$$-R^2\cot\theta\dot{\theta}^2 = R^2\ddot{\theta} \tag{1.64}$$

$$\implies -\int d\theta\cot\theta = \int\frac{d\dot{\theta}}{\dot{\theta}} \tag{1.65}$$

$$\implies \log\dot{\theta} + C'' = -\log\sin\theta \tag{1.66}$$

$$\implies \dot{\theta} = \frac{C'}{\sin\theta} \tag{1.67}$$

$$\implies \cos\theta = C\lambda \tag{1.68}$$

$$\implies \sin\theta = \sqrt{1-(C\lambda)^2} \tag{1.69}$$

hence $\sin\varphi = \mp\dfrac{C\lambda}{s\sqrt{1-(C\lambda)^2}}$ solves the equations in parametrizations of λ for constant parameters C and s. Note that this expression is related to (1.60) when calculating angular momentum.

Problem 2

(**a**) Consider the relativistic action of a charged particle

$$S = \int d\lambda \left[-\frac{1}{2} e^{-1}(\lambda) g_{\mu\nu}(x) \dot{x}^\mu \dot{x}^\nu - \frac{1}{2} e(\lambda) m^2 - q A_\mu(x) \dot{x}^\mu \right]. \tag{2.1}$$

Note that, under a reparametrization of the wordline $\lambda \to \lambda'$, the action is invariant. To show this, first note that $e(\lambda') \to \frac{d\lambda}{d\lambda'} e(\lambda)$. In each derivative, we have that $\frac{dx^\mu}{d\lambda'} \to \frac{dx^\mu}{d\lambda}\frac{d\lambda}{d\lambda'}$ by the chain rule, hence

$$\begin{aligned}
S &= \int d\lambda' \left[-\frac{1}{2} e^{-1}(\lambda') g_{\mu\nu}(x) \dot{x}^\mu \dot{x}^\nu - \frac{1}{2} e(\lambda') m^2 - q A_\mu(x) \dot{x}^\mu \right] && (2.2)\\
&= \int d\lambda' \left[-\frac{1}{2} (-e^{-2}(\lambda)) \frac{d\lambda'}{d\lambda} e(\lambda) g_{\mu\nu}(x) \frac{dx^\mu}{d\lambda}\frac{d\lambda}{d\lambda'}\frac{dx^\nu}{d\lambda}\frac{d\lambda}{d\lambda'} - \frac{1}{2}\frac{d\lambda}{d\lambda'} e(\lambda) m^2 - q A_\mu(x) \frac{dx^\mu}{d\lambda}\frac{d\lambda}{d\lambda'} \right] && (2.3)\\
&= \int d\lambda' \frac{d\lambda}{d\lambda'} \left[-\frac{1}{2} e^{-1}(\lambda) g_{\mu\nu}(x) \dot{x}^\mu \dot{x}^\nu - \frac{1}{2} e(\lambda) m^2 - q A_\mu(x) \dot{x}^\mu \right] && (2.4)\\
&= \int d\lambda \left[-\frac{1}{2} e^{-1}(\lambda) g_{\mu\nu}(x) \dot{x}^\mu \dot{x}^\nu - \frac{1}{2} e(\lambda) m^2 - q A_\mu(x) \dot{x}^\mu \right] && (2.5)
\end{aligned}$$

which is identical to S prior to the reparametrization.

(**b**) If we were to apply a Gauge transformation to the A_μ fields, we introduce a scalar quantity f, which is a function of the coordinates such that

$$A_\mu \to A_\mu - \frac{\partial f}{\partial x^\mu}. \tag{2.6}$$

Only considering the last term in (2.1), the action transforms as

$$\begin{aligned}
S_f &= -q \int d\lambda \, (A_\mu - \partial_\mu f) \dot{x}^\mu && (2.7)\\
&= -q \int d\lambda \left[A_\mu \dot{x}^\mu - \frac{\partial f}{\partial \lambda}\frac{\partial \lambda}{\partial x^\mu}\frac{\partial x^\mu}{\partial \lambda} \right] && (2.8)\\
&= -q \int d\lambda \left[A_\mu \dot{x}^\mu - \frac{\partial f}{\partial \lambda} \right] && (2.9)
\end{aligned}$$

We note that the last derivative of f in λ vanishes upon variation of the action, which will not affect the equations of motion. For the einbein terms 1 and 2, we note that varying the action will not affect the equations of motion regardless of choice of $e(\lambda)$ since we vary with respect to x^μ. The einbein will only affect constraints when varying with respect to $e(\lambda)$. One may further perform a parametrization transformation of λ, say a shift or scale, which would only affect the reparametrizations of the final equations.

(**c**) We can derive the equations of motion for the action (2.1) by varying $x^\mu \to x^\mu + \delta x^\mu$ and requiring $\delta S = 0$. We first note that, under these conditions,

$$\begin{aligned}
\delta g_{\mu\nu} &= \frac{\partial g_{\mu\nu}}{\partial x^\sigma} \delta x^\sigma && (2.10)\\
\delta \dot{x}^\mu &= \delta \frac{dx^\mu}{d\tau} = \frac{d\delta x^\mu}{d\tau} && (2.11)
\end{aligned}$$

$$\delta A_\mu = \frac{\partial A_\mu}{\partial x^\sigma}\delta x^\sigma \tag{2.12}$$

Choosing the temporal gauge $e(\tau) = \dfrac{1}{m}$ and applying the variational principle, we find

$$\begin{aligned}
\delta S = 0 &= \int d\tau \left[-\frac{m}{2}\delta g_{\mu\nu}\dot{x}^\mu \dot{x}^\nu - \frac{m}{2}g_{\mu\nu}\delta\dot{x}^\mu\dot{x}^\nu - \frac{m}{2}g_{\mu\nu}\dot{x}^\mu\delta\dot{x}^\nu\right] \\
&\quad + \int d\tau\, \delta\left[-\frac{m}{2}\right] + \int d\tau \left[-q\delta A_\mu \dot{x}^\mu - qA_\mu\delta\dot{x}^\mu\right] && (2.13)\\
&= -\frac{m}{2}\int d\tau \left[\partial_\sigma g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu\delta x^\sigma + g_{\mu\nu}\dot{x}^\nu\frac{\partial\delta x^\mu}{\partial\tau} + g_{\mu\nu}\dot{x}^\mu\frac{\partial\delta x^\nu}{\partial\tau}\right] \\
&\quad + (0) - q\int d\tau\, \left[\partial_\sigma A_\mu\dot{x}^\mu\delta x^\sigma + A_\mu\frac{\partial\delta x^\mu}{\partial\tau}\right] && (2.14)\\
&= \int d\tau \left[\frac{m}{2}\partial_\sigma g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu + q\partial_\sigma A_\mu\dot{x}^\mu\right]\delta x^\sigma \\
&\quad + \int d\tau \left[\frac{m}{2}g_{\mu\nu}\dot{x}^\nu + qA_\mu\right]\frac{\partial\delta x^\mu}{\partial\tau} + \int d\tau\left[\frac{m}{2}g_{\mu\nu}\dot{x}^\mu\frac{\partial\delta x^\nu}{\partial\tau}\right] && (2.15)
\end{aligned}$$

where we have multiplied by (-1) in (2.15) to eliminate the minus sign. Integrating the two integrals in the lower part of (2.15) by parts, and noting that we require δx^μ to vanish along the boundary, we find that

$$\begin{aligned}
0 &= \int d\tau \left[\frac{m}{2}\partial_\sigma g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu + q\partial_\sigma A_\mu\dot{x}^\mu\right]\delta x^\sigma \\
&\quad - \int d\tau\,\frac{d}{d\tau}\left[\frac{m}{2}g_{\mu\nu}\dot{x}^\nu + qA_\mu\right]\delta x^\mu - \int d\tau\,\frac{d}{d\tau}\left[\frac{m}{2}g_{\mu\nu}\dot{x}^\mu\right]\delta x^\nu && (2.16)\\
&= \int d\tau \left[\frac{m}{2}\partial_\sigma g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu + q\partial_\sigma A_\mu\dot{x}^\mu\right]\delta x^\sigma \\
&\quad - \int d\tau\, \left[\frac{m}{2}\partial_\sigma g_{\mu\nu}\dot{x}^\sigma\dot{x}^\nu + \frac{m}{2}g_{\mu\nu}\ddot{x}^\nu + q\partial_\sigma A_\mu\dot{x}^\sigma\right]\delta x^\mu \\
&\quad - \int d\tau\, \left[\frac{m}{2}\partial_\sigma g_{\mu\nu}\dot{x}^\sigma\dot{x}^\mu + \frac{m}{2}g_{\mu\nu}\ddot{x}^\mu\right]\delta x^\nu && (2.17)
\end{aligned}$$

We now require that everything within the brackets must be zero since δx^μ is arbitrary and is only zero along the boundary of the variation. Hence, re-indexing each variational element to α (instead of σ, μ, ν, etc) along with its contracted constituents, we have

$$\begin{aligned}
0 &= \frac{m}{2}\partial_\alpha g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu + q\partial_\alpha A_\mu\dot{x}^\mu - \frac{m}{2}\partial_\sigma g_{\alpha\nu}\dot{x}^\sigma\dot{x}^\nu - \frac{m}{2}g_{\alpha\nu}\ddot{x}^\nu - q\partial_\sigma A_\alpha\dot{x}^\sigma \\
&\quad - \frac{m}{2}\partial_\sigma g_{\mu\alpha}\dot{x}^\sigma\dot{x}^\mu - \frac{m}{2}g_{\mu\alpha}\ddot{x}^\mu && (2.18)\\
&= \frac{m}{2}\left[\partial_\alpha g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu - \partial_\sigma g_{\alpha\nu}\dot{x}^\sigma\dot{x}^\nu - \partial_\sigma g_{\mu\alpha}\dot{x}^\sigma\dot{x}^\mu\right] - \frac{m}{2}\left[g_{\alpha\nu}\ddot{x}^\nu + g_{\mu\alpha}\ddot{x}^\mu\right] \\
&\quad + q\left[\partial_\alpha A_\mu\dot{x}^\mu - \partial_\sigma A_\alpha\dot{x}^\sigma\right] && (2.19)
\end{aligned}$$

We may now use the identity $g_{\mu\nu}g^{\nu\kappa} = \delta^\kappa_\mu$, and this is imposed by multiplying both side of the equation by $g^{\kappa\alpha}$:

$$0 = \frac{m}{2}g^{\kappa\alpha}\left[\partial_\alpha g_{\mu\nu}\dot{x}^\mu\dot{x}^\nu - \partial_\sigma g_{\alpha\nu}\dot{x}^\sigma\dot{x}^\nu - \partial_\sigma g_{\mu\alpha}\dot{x}^\sigma\dot{x}^\mu\right] - \frac{m}{2}\left[g^{\kappa\alpha}g_{\alpha\nu}\ddot{x}^\nu + g^{\kappa\alpha}g_{\mu\alpha}\ddot{x}^\mu\right]$$

$$+ q\left[g^{\kappa\alpha}\partial_\alpha A_\mu \dot{x}^\mu - g^{\kappa\alpha}\partial_\sigma A_\alpha \dot{x}^\sigma\right] \tag{2.19}$$

Noting that the first term is the definition of the connection coefficient (Christoffel symbol) contracted with $\dot{x}^\mu \dot{x}^\nu$,

$$\Gamma^\kappa_{\mu\nu}\dot{x}^\mu\dot{x}^\nu = \frac{1}{2}g^{\kappa\alpha}[\partial_\sigma g_{\alpha\nu}\dot{x}^\sigma\dot{x}^\nu + \partial_\sigma g_{\nu\alpha}\dot{x}^\sigma\dot{x}^\nu - \partial_\alpha g_{\sigma\nu}\dot{x}^\sigma\dot{x}^\nu] \tag{2.20}$$

we factor out the minus sign and write

$$0 = -m\Gamma^\kappa_{\mu\nu}\dot{x}^\mu\dot{x}^\nu - \frac{m}{2}\left[g^{\kappa\alpha}g_{\alpha\nu}\ddot{x}^\nu + g^{\kappa\alpha}g_{\mu\alpha}\ddot{x}^\mu\right] + q\left[g^{\kappa\alpha}\partial_\alpha A_\mu\dot{x}^\mu - g^{\kappa\alpha}\partial_\sigma A_\alpha\dot{x}^\sigma\right]. \tag{2.21}$$

We further note that the second term in (2.21) becomes

$$\begin{aligned}
\frac{m}{2}\left[g^{\kappa\alpha}g_{\alpha\nu}\ddot{x}^\nu + g^{\kappa\alpha}g_{\mu\alpha}\ddot{x}^\mu\right] &= -\frac{m}{2}\left[\delta^\kappa_\nu\ddot{x}^\nu + \delta^\kappa_\alpha\ddot{x}^\mu\right] &(2.22)\\
&= \frac{m}{2}\left[2\ddot{x}^\kappa\right] &(2.23)\\
&= m\ddot{x}^\kappa &(2.24)
\end{aligned}$$

and the third term is the definition of the field tensor with a raised index coming from $g^{\kappa\alpha}$ with a contraction in $\dot{x}^\mu$,

$$F^\kappa_\mu\dot{x}^\mu = (\partial^\kappa A_\mu - \partial_\mu A^\kappa)\dot{x}^\mu \tag{2.25}$$

Therefore, putting together (2.21), (2.24) and (2.25) together, we obtain the equations of motion for a massive particle subject to a gravitational field and an eletric field (or any other external field),

$$\begin{aligned}
0 &= -m\Gamma^\kappa_{\mu\nu}\dot{x}^\mu\dot{x}^\nu - m\ddot{x}^\kappa + qF^\kappa_\mu\dot{x}^\mu &(2.26)\\
\implies \Gamma^\kappa_{\mu\nu}\dot{x}^\mu\dot{x}^\nu + \ddot{x}^\kappa &= \frac{q}{m}F^\kappa_\mu\dot{x}^\mu. &(2.27)
\end{aligned}$$

We lastly note that the $\dot{x}^\mu$'s are functions of τ and the F^κ_μ, $\Gamma^\kappa_{\mu\nu}$ functions of the x^μ's, so we may also write the equations of motion with explicit dependence

$$\Gamma^\kappa_{\mu\nu}(x)\dot{x}^\mu(\tau)\dot{x}^\nu(\tau) + \ddot{x}^\kappa(\tau) = \frac{q}{m}F^\kappa_\mu(x)\dot{x}^\mu(\tau). \tag{2.28}$$

Problem 3 and Problem 4

3-(a) All code, inputs and outputs are given at the end. For this problem, I wrote my code step by step as per the problem, and later defined a class to calculate each of the steps as functions. I had a difficult time using the `sympy` shapes and matrices, not understanding how to index or how to overwrite matrix elements. Instead, I used `numpy` arrays, setting `dtype = object` in each instance to take `sympy` variables in. For some reason, this seemed to work, so I chose to continue the problem by using for-loops over the indices to fill the array elements. I will not include my initial testing code, but will just include my function class which I wrote below instead for the sake of space in this problem set. I wrote a class of functions so that I can just import the class and call any function I want in the future to calculate any tensor I want.

The downstairs metric $g_{\mu\nu}$ was specified for user input. The upstairs metric $g^{\mu\nu}$ was calculated from $g_{\mu\nu}$ using `sympy.Matrix().inv`, which is the inverse matrix. The delta function is easy obtained by just contracting the two together. From the invariant interval

$$ds^2 = L^2[dt^2 - \cosh^2 t(d\theta^2 + \sin^2\theta d\varphi^2)] \tag{3.1}$$

the `gdndn` array was given in matrix form as

$$g_{\mu\nu} = \begin{pmatrix} L^2 & 0 & 0 \\ 0 & -L^2\cosh^2 t & 0 \\ 0 & 0 & -L^2\cos^2 t\sin^2\theta \end{pmatrix} \tag{3.2}$$

3-(b) For this problem in calculating the connection coefficients, I introduce a standard procedure (I slaved away to try to figure this out) for calculating the tensors (and tensors to come).

1. Initialize empty arrays. If the range of indices was N (ie, N-dimensional), create an empty `numpy` array to store the future values. This was done with `numpy.empty((N, N, N), dtype = object)` for a rank 3 tensor such as $\Gamma^{\alpha}_{\mu\nu}$ (it doesn't matter whether the index is up or down in defining the empty array).

2. Create an empty contraction array prior to index summation. To calculate contractions, they have to be done via sum over a specific axis. I did this by defining an empty array of dimensionality $(1 \times N)$ to calculate each of the index elements for a contracton prior to summation.

3. Create a contraction array for summation. To perform contractions, I found it difficult to sum over specific axes for an arbitrary rank tensor. To do this instead, I defined a $(1 \times N)$-dimensional array filled with ones. I could then take the dot product for any contraction with this array with the (now filled) array with each of the contraction elements. We then automatically sum the indices I want contracted via `contract.dot(dotfill)`.

4. Loop over the indices. For a rank-i tensor with j distinct index contractions, we loop over $i + j$ indices. Each term may be calculated individually, and this can be done via addition, multiplication, or by taking derivatives. For this, I used $\partial_\mu g_{\alpha\beta} \equiv$ `sympy.diff(gdndn[alpha][beta], xup[mu])`. Contractions would be completed last in the loops, outside of the previous contraction loop.

5. Simplify and return the tensor matrix. This would return the whole tensor and components including zeros.

An instance of this process is shown in the code below under the `chrisUDD` function. I will now jsut briefly calculate three of the elements by hand:

$$\begin{aligned}
\Gamma^t_{\theta\theta} &= \frac{1}{2}g^{tt}[\partial_\theta g_{tt} + \partial_\theta g_{t\theta} - \partial_t g_{\theta\theta}] \qquad (3.3)\\
&= \frac{1}{2}\frac{1}{L^2}[(0) + (0) - \partial_t(-L^2\cosh^2 t)] \qquad (3.4)\\
&= \frac{1}{2}\frac{1}{L^2}L^2 \cdot 2\cosh t\sinh t \qquad (3.5)\\
&= \cosh t \sinh t \qquad (3.6)\\
\Gamma^\theta_{t\theta} &= \frac{1}{2}g^{\theta\theta}[\partial_t g_{\theta\theta} + \partial_\theta g_{t\theta} - \partial_\theta g_{\theta t}] \qquad (3.7)\\
&= \frac{1}{2}\frac{1}{(-L^2\cosh^2 t)}[\partial_t(-L^2\cosh^2 t) + (0) + (0)] \qquad (3.8)\\
&= \frac{1}{2}\frac{1}{(-L^2\cosh^2 t)} \cdot 2(-L^2\cosh t\sinh t) \qquad (3.9)\\
&= \tanh t \qquad (3.10)\\
\Gamma^\theta_{\varphi\varphi} &= \frac{1}{2}g^{\theta\theta}[\partial_\varphi g_{\theta\theta} + \partial_\varphi g_{\varphi\theta} - \partial_\theta g_{\varphi\varphi}] \qquad (3.11)\\
&= \frac{1}{2}\frac{1}{(-L^2\cosh^2 t)}[(0) + (0) - \partial_\theta(-L^2\cosh^2 t\sin^2\theta)] \qquad (3.12)\\
&= -\frac{1}{2}\frac{1}{(L^2\cosh^2 t)}L^2\cosh^2 t \cdot 2\sin\theta\cos\theta \qquad (3.13)\\
&= -\sin\theta\cos\theta. \qquad (3.14)
\end{aligned}$$

4-(a) The Riemann up-down-down-down tensor was calculated by implementing the same process outlined in 3-(b). Two non-zero components are $R^t_{\theta t\theta} = \cosh^2 t$ and $R^\theta_{tt\theta} = 1$.

$$\begin{aligned}
R^t_{\theta t\theta} &= \partial_t\Gamma^t_{\theta\theta} - \partial_\theta\cancelto{0}{\Gamma^t_{\theta t}} + \Gamma^\lambda_{\theta\theta}\cancelto{0}{\Gamma^t_{\lambda t}} - \Gamma^\lambda_{\theta t}\Gamma^t_{\lambda\theta} \qquad (4.1)\\
&= \partial_t[\cosh t\sinh t] - \tanh(t)\cosh t\sinh t \qquad (4.2)\\
&= \sinh^2 t + \cosh^2 t - \sinh^2 t \qquad (4.3)\\
&= \cosh^2 t \qquad (4.4)\\
&= \cosh(2t) - \frac{1}{2}\tanh(t)\cosh t\sinh 2t \qquad (4.5)
\end{aligned}$$

where I note that python outputs (4.5) however this simplifies trignonmetrically to (4.4). Similarly,

$$\begin{aligned}
R^\theta_{tt\theta} &= \partial_t\Gamma^\theta_{t\theta} - \partial_\theta\cancelto{0}{\Gamma^\theta_{tt}} + \Gamma^\lambda_{t\theta}\Gamma^\theta_{\lambda t} - \cancelto{0}{\Gamma^\lambda_{tt}}\Gamma^\theta_{\lambda\theta} \qquad (4.6)\\
&= \partial_t[\tanh(t)] + \Gamma^\theta_{t\theta}\Gamma^\theta_{\theta t} \qquad (4.7)\\
&= \operatorname{sech}^2 t + \tanh^2 t \qquad (4.8)\\
&= 1 \qquad (4.9)
\end{aligned}$$

4-(b) The Ricci tensor was found by the same procedure as described in 3-(a). In matrix representation, it is

$$R_{\mu\nu} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -2\cosh^2 t & 0 \\ 0 & 0 & -2\sin^2(\theta)\cosh^2 t \end{pmatrix} \tag{4.10}$$

By contracting (4.10) by raising one index (this introduces now a L^{-2}), the Ricci scalar was found to be

$$R = \frac{6}{L^2} \tag{4.11}$$

by means of the contraction described in 3-(a).

4-(c) The field equation

$$R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R + \Lambda g_{\mu\nu} = 0 \tag{4.12}$$

was programmed into python. The first test was to see if the trial would yield zero (which it did), while the second was to obtain an explicit expression for Λ in Kronecker-delta format. The latter was accomplished by writing

$$0 = g^{\alpha\mu}(R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R + \Lambda g_{\mu\nu}) \tag{4.13}$$

$$= R^{\alpha}_{\nu} - \frac{1}{2}\delta^{\alpha}_{\nu}R + \Lambda\delta^{\alpha}_{\nu} \tag{4.14}$$

$$\implies \Lambda\delta^{\alpha}_{\nu} = \frac{1}{2}\delta^{\alpha}_{\nu}R - R^{\alpha}_{\nu} \tag{4.15}$$

The output was found to be $\Lambda = \frac{1}{L^2}$ exactly. I have included the code here, because I didn't include it as a method in my class below. The reason I didn't want to write my code to only print outputs of non-zero quantities was because I know I would find visualizing the tensor / indexing harder. If I print my outputs this way, I can easily locate components of the tensor instead of having to go back and use indices to locate specific entries. The matrix representations also look quite nice.

```
    #specify array for general output of the (reduced) field equations
output = np.empty((3,3), dtype = object)
    #specify array to determine the constant \Lambda
LambdaConstant = np.empty((3,3), dtype = object)
    #specify a temporary array to contract over the nu index
contr = np.empty(3, dtype = object)
    #iterate over indices
for mu in indexlist:
    for la in indexlist:
        for nu in indexlist:    #determine the (mu, nu) elements of the
                                output array by programming the field equation
            output[mu][nu] = Riccdndn[mu][nu] - 0.5 * gdndn[mu][nu] * RiccScal
                                                         + (1/L**2)*gdndn[mu][nu]
                                #contract over the nu index ONlY
                                (do not add other terms like the RiccScal term)
            contr[nu] = - Riccdndn[mu][nu]*gupup[nu][la]
                                    #do dot product to get contraction;
                                     then add the 0.5R term on the back
```

```
                                      to get the proper output
        LambdaConstant[mu][la] = sp.simplify(contr.dot(vdotfill))
                                + 0.5*RiccScal*delta[mu][la]

    #print values
print('field equation test (get zeros) & find Lambda')
print(output)
print(AlphaConstant)
print('------') #space
```

CODE:

```
import numpy as np
import sympy as sp


class calcTensor:
    '''
    ***Requires numpy as 'np' and sympy as 'sp' installation (import) to load.***

    **Class:** Functions to compute tensor identities.

    **Functions:**
        * coordinateGenerator
        * metricGenerator
        * deltaFunction
        * chrisUDD
        * riemannUDDD
        * ricciDD
        * ricciScalar
        * retrieveAll
        * classInfo
    '''
    def __init__(self):
        """
        Initialize the class.
        """
        pass

    def coordinateGenerator(
            vars: list[str]
            ) -> list[str]:
        """
        **Returns:** dtype = numpy.array. List of coordinate
         variables in N-vector contravariant tensor notation.

        *vars:* List of symbolic strings generated by sympy.symbols.
        """
        return np.array(vars, dtype=object)


    def metricGenerator(
            vars: list[str]
            ) -> tuple[list[str]]:
        """
        **Returns:** dtype = numpy.array. Metric tensor lower
         'g_dndn' and upper 'g_upup' for symbolic calculation
          (tuple). Dimensionality out is equivalent to dimensionality
```

```
      in.

    *vars:* Input array for 'g_dndn' metric of dimensionality
     NxN of symbolic strings generated by sympy.symbols.
    """
    dims = np.shape(vars)
    if dims[0] != dims[1]:
        raise ValueError('Input array-list *vars* must be of
         dimensionality NxN.')

    gdndn = np.array(vars, dtype=object)

    gupup = sp.Matrix(gdndn).inv()
    gupup = np.array(gupup, dtype=object)
    return gdndn, gupup


def deltaFunction(
        dimension: int
        ) -> list[int]:
    """
    **Returns:** dtype = numpy.array. Kronecker Delta function
     of dimensionality N. Iterable through by tensor indices.

    *dimension:* Input integer to specify dimension output array.
    """
    if type(dimension) != int:
        raise ValueError('Input variable *dimension* must be an integer.')

    N = dimension
    indexlist = np.arange(N)
    arr = np.zeros((N, N))
    for n in indexlist:
        arr[n][n] = 1

    return arr


def chrisUDD(
        g_upup: list[str],
        g_dndn: list[str],
        xup: list[str]
        ) -> list[str]:
    """
    **Returns:** dtype = numpy.array. NxNxN dimensional array of
     symbolic Christoffel symbols of tensor type 'up-down-down'
     generated from upstairs and downstairs metric tensors.
    *g_upup:* dtype = numpy.array. NxN upstairs metric tensor of
     symbolic sympy values.

    *g_dndn:* dtype = numpy.array. NxN downstairs metric tensor of
     symbolic sympy values.

    *xup:* dtype = numpy.array. Nx1 vector list of symbolic variables
     generated from sympy.symbols.
    """
    N = len(xup)
    indexlist = np.arange(N)

    vdotfill = np.ones(N)
    Chrisupdndn = np.empty((N,N,N), dtype=object)
    contract = np.empty((N), dtype=object)

    for mu in indexlist:
        for nu in indexlist:
            for la in indexlist:
                for si in indexlist:
                    contract[si] = 0.5*(g_upup[mu][si])
                                  *(sp.diff(g_dndn[si][la], xup[nu])
                                    + sp.diff(g_dndn[nu][si], xup[la])
                                    - sp.diff(g_dndn[nu][la], xup[si]) )
                Chrisupdndn[mu][nu][la] = sp.simplify(contract.dot(vdotfill))

    return Chrisupdndn


def riemannUDDD(
        christoffel: list[str],
        xup: list[str]
        ) -> list[str]:
    """
    **Returns:** dtype = numpy.array. NxNxNxN dimensional array of
     symbolic Riemann tensor of type 'up-down-down-down' generated
     from Christoffel symbols.

    *christoffel:* dtype = numpy.array. NxNxN Christoffel symbol tensor
     of symbolic sympy values.

    *xup:* dtype = numpy.array. 1xN vector list of symbolic variables
     generated from sympy.symbols.
    """
    N = len(xup)
    indexlist = np.arange(N)

    vdotfill = np.ones(N)
    Riemupdndndn = np.empty((N, N, N, N), dtype=object)
    contract = np.empty((N), dtype=object)

    for rho in indexlist:
        for si in indexlist:
            for mu in indexlist:
                for nu in indexlist:
                    for la in indexlist:
                        contract[la] = (christoffel[la][si][nu]
                                                *christoffel[rho][la][mu])
                                    - (christoffel[la][si][mu]
                                                *christoffel[rho][la][nu])
                    Riemupdndndn[rho][si][mu][nu] = contract.dot(vdotfill)
                                + sp.diff(christoffel[rho][si][nu], xup[mu])
                                - sp.diff(christoffel[rho][si][mu], xup[nu])

    return sp.simplify(Riemupdndndn)


def ricciDD(
        riemann: list[str]
        ) -> list[str]:
    """
    **Returns:** dtype = numpy.array. NxN dimensional array of symbolic Ricci
     tensor of type 'down-down' generated from Riemann tensor.

    *riemann:* dtype = numpy.array. NxNxNxN Riemann tensor of
     symbolic sympy values.
    """
    N = np.shape(riemann)[0]
    indexlist = np.arange(N)

    vdotfill = np.ones(N)
    Riccdndn = np.empty((N, N), dtype = object)
    contract = np.empty(N, dtype = object)

    for mu in indexlist:
        for nu in indexlist:
            for la in indexlist:
                contract[la] = riemann[la][mu][nu][la]
            Riccdndn[mu][nu] = sp.simplify(contract.dot(vdotfill))

    return Riccdndn

def ricciScalar(
        ricciTensor: list[str],
        g_upup: list[str]
        ) -> str:
```

```
    """
    **Returns:** dtype = string. Symbolic scalar value of type sympy
     variable. Contraction along two axes of Ricci tensor type 'down-down'.

    *ricciTensor:* dtype = numpy array. NxN Ricci curvature tensor
     of symbolic sympy values.

    *g_upup:* dtype = numpy array. NxN metric tensor of symbolic
     sympy values of type 'up-up' indexing.
    """
    N = np.shape(g_upup)[0]
    indexlist = np.arange(N)

    vdotfill = np.ones(N)
    contract2 = np.empty(N, dtype=object)
    contract1 = np.empty(N, dtype=object)
    for mu in indexlist:
        for nu in indexlist:        #determine the  nu-th term
            contract1[nu] = ricciTensor[mu][nu] * g_upup[mu][nu]
                                    #contract over nu
        contract2[mu] = sp.simplify(contract1.dot(vdotfill))
    RicciScalar = sp.simplify(contract2.dot(vdotfill))  #now contract over mu

    return RicciScalar


def retrieveAll(
        metric_vars: list[str],
        coordinates: list[str],
        return_type: str
        ) -> None:
    """
    **Returns:** Any chosen value determine from previous functions.

    *metric_vars:* numpy.array. Input array for 'g_dndn' metric
     of dimensionality NxN of symbolic strings generated
     by sympy.symbols.

    *coordinates:* numpy.array. List of symbolic strings
     generated by sympy.symbols.

    *return_type:* string. 'CC' for Christoffel symbols up-down-down.
     'RMT' for Riemann tensor up-down-down-down. 'RIT' for Ricci tensor
      down-down. 'RS' for Ricci Scalar.
    """

    xup = calcTensor.coordinateGenerator(coordinates)
    gdndn, gupup = calcTensor.metricGenerator(metric_vars)
```

```
christoffel = calcTensor.chrisUDD(gupup, gdndn, xup)

riemanntensor = calcTensor.riemannUDDD(christoffel, xup)

riccitensor = calcTensor.ricciDD(riemanntensor)
ricciscalar = calcTensor.ricciScalar(riccitensor, gupup)


for retry in range(5):
    if return_type == 'MT':
        print('g_dndn: {}'.format(gdndn))
        print('g_upup: {}'.format(gupup))
        break
    if return_type == 'CC':
        print(christoffel)
        break
    elif return_type == 'RMT':
        print(riemanntensor)
        break
    elif return_type == 'RIT':
        print(riccitensor)
        break
    elif return_type == 'RIS':
        print(ricciscalar)
        break
    elif return_type == 'ALL':
        print('----------')
        print('xup: {}'.format(xup))
        print('g_dndn: {}'.format(gdndn))
        print('g_upup: {}'.format(gupup))
        print('----------')
        print('christoffel: {}'.format(christoffel))
        print('----------')
        print('riemann tensor: {}'.format(riemanntensor))
        print('----------')
        print('ricci tensor: {}'.format(riccitensor))
        print('----------')
        print('ricci scalar: {}'.format(ricciscalar))
        print('----------')
        break
    else:
        return_type = input('Please enter a valid return_type
                    string (MT, CC, RMT, RIT, RIS, ALL):')
else:
    print('You keep making invalid choices. Now exiting.')
    def classInfo() -> any:
        """
        Retrieve class 'calcTensor' documentation.
        """
        print(help(calcTensor))
```

SAMPLE CLASS CALL AND INPUT (SEPERATE PYTHON FILE IN SAME DIRECTORY):

```
import sympy as sp
from functions import calcTensor as t

r, theta, phi = sp.symbols('r theta phi')

t.retrieveAll( [[1, 0, 0],
            [0, r**2,0],
            [0, 0, r**2*sp.sin(theta)**2]],
             [r, theta, phi], 'ALL')
```

DIRECT OUTPUTS:

```
kronecker delta
[[1 0 0]
 [0 1 0]
 [0 0 1]]
------
christoffel symbol
[[[0 0 0]
  [0 0.5*sinh(2*t) 0]
  [0 0 0.5*sin(theta)**2*sinh(2*t)]]

 [[0 1.0*tanh(t) 0]
  [1.0*tanh(t) 0 0]
  [0 0 -0.5*sin(2*theta)]]

 [[0 0 1.0*tanh(t)]
  [0 0 1.0/tan(theta)]
  [1.0*tanh(t) 1.0/tan(theta) 0]]]
------
riemann tensor
[[[[0 0 0]
   [0 0 0]
   [0 0 0]]

  [[0 -0.5*sinh(2*t)*tanh(t) + 1.0*cosh(2*t) 0]
   [0.5*sinh(2*t)*tanh(t) - 1.0*cosh(2*t) 0 0]
   [0 0 0]]
  [[0 0 -0.5*sin(theta)**2*sinh(2*t)*tanh(t) + 1.0*sin(theta)**2*cosh(2*t)]
   [0 0 1.0*sin(theta)*cos(theta)*sinh(2*t) - 0.5*sin(2*theta)*sinh(2*t)]
   [0.5*sin(theta)**2*sinh(2*t)*tanh(t) - 1.0*sin(theta)**2*cosh(2*t)
    -1.0*sin(theta)*cos(theta)*sinh(2*t) + 0.5*sin(2*theta)*sinh(2*t) 0]]


 [[[0 1.00000000000000 0]
   [-1.00000000000000 0 0]
   [0 0 0]]

  [[0 0 0]
   [0 0 0]
   [0 0 0]]

  [[0 0 0]
   [0 0 0.5*sin(2*theta)/tan(theta) - 0.25*cos(2*theta)*sinh(2*t)*tanh(t)
                 - 1.0*cos(2*theta) + 0.25*sinh(2*t)*tanh(t)]
   [0 -0.5*sin(2*theta)/tan(theta) + 0.25*cos(2*theta)*sinh(2*t)*tanh(t)
                 + 1.0*cos(2*theta) - 0.25*sinh(2*t)*tanh(t) 0]]]


 [[[0 0 1.00000000000000]
   [0 0 0]
   [-1.00000000000000 0 0]]

  [[0 0 0]
   [0 0 1.0*(-tan(theta)**2 - 1)/tan(theta)**2 - 0.5*sinh(2*t)*tanh(t)
                 + 1.0/tan(theta)**2]
   [0 -1.0*(-tan(theta)**2 - 1)/tan(theta)**2 + 0.5*sinh(2*t)*tanh(t)
                 - 1.0/tan(theta)**2 0]]

  [[0 0 0]
   [0 0 0]
   [0 0 0]]]]
------
ricci tensor
[[2.00000000000000 0 0]
[0 -2.0*cosh(t)**2 0]
[0 0 -2.0*sin(theta)**2*cosh(t)**2]]
------
ricci scalar
6.0/L**2
------
field equation test (get zeros) & find Lambda
[[0 0 0]
[0 0 0]
[0 0 0]]
```

```
[[1.0/L**2 0 0]
[0 1.0/L**2 0]
[0 0 1.0/L**2]]
```