```
[17]: from sympy import MatrixSymbol, Matrix, Array, pprint
from sympy import symbols, diff, exp, log, cos, sin, simplify, Rational
from sympy.core.symbol import Symbol
from sympy import pi

import numpy as np
import sympy as sp
from IPython.display import display, Latex
```

Tensor operations

```
[18]: def INDX(i, place, num_indx):
    """
    Acceses an index at 'place' for 'num_indx' order tensor
    T_(a0 ... âp ... an-1) = T[INDX(i, place=p, num_indx=n)] = T[:,...<-p-> ,␣
↪i, :,...<-(n-p-1)->]
    """
    indx = []
    assert place<num_indx
    for j in range(num_indx):
        if place==j: indx.append(i)
        else: indx.append(slice(None))
    return tuple(indx)
```

```
[19]: def contract(T, g=None, g_inv=None, num_indx=2, upper=1, indx=(0, 1)):
    """
    contracts indecies indx=(a_p, a_q) on tensor T with 'num_indx',
    'upper' of whom are upper. If upper=0, all indecies are assumed lower.
    With indx=(a_k, a_l), upper=n, num_indx=n+m, this gives
    T^(a_0...a_n-1)_(a_n...a_n+m-1) -> T^(a_0...a_k=a...a_n-1)_(a_n...a_k...
↪a_n+m-1),
    with the necesarry metric. If wrong metric is given, this wil throw error.
    """
    assert indx[0] < indx[1]  # we have to know if the index to the left␣
↪dissapears
    dim = np.shape(T)[0]
    a = (indx[0] < upper) + (indx[1] < upper) # number of upper indecies to be␣
↪contracted
    if a==2: g0 = g # two upper
    elif a==0: g0 = g_inv # two lower
    else: g0 = np.identity(dim, dtype=Rational)

    Tc = Rational(0) * np.ones((T.shape)[:-2], dtype=Rational)
    for i in range(dim):
        for j in range(dim):
            Tc += g0[i, j] * (T[INDX(i, indx[0], num_indx)])[INDX(j, indx[1] -␣
↪1, num_indx - 1)]
```

```python
    return Tc

def raise_indx(T, g_inv, indx, num_indx):
    """
    Raise index 'indx' of a tensor T with 'num_indx' indices.
    """
    dim = np.shape(T)[0]
    Tu = np.zeros_like(T)
    for i in range(dim):
        I = INDX(i, indx, num_indx)
        for j in range(dim):
            J = INDX(j, indx, num_indx)
            Tu[I] += g_inv[i, j] * T[J]
    return Tu

def lower_indx(T, g, indx, num_indx):
    return raise_indx(T, g, indx, num_indx)

def get_g_inv(g):
    return np.array(Matrix(g)**(-1))
```

Calculate Christoffel symbols and Riemann curvature tensor

[20]:
```python
def Christoffel(g, g_inv, var):
    """
    Work out the christoffel symbols, given a metric an its variables
    Γ^i_jk = C[i, j, k]
    """
    dim = len(var)
    C = np.zeros((dim, dim, dim), dtype=Symbol)
    for i in range(dim):
        for j in range(dim):
            for k in range(dim):
                for m in range(dim):
                    C[i, j, k] += Rational(1, 2) * (g_inv)[i, m] * (
                        diff(g[m, k], var[j])
                        + diff(g[m, j], var[k])
                        - diff(g[k, j], var[m])
                    )

    return C
```

[21]:
```python
def Riemann_tensor(C, var):
    """
    Riemann_tensor(Christoffel_symbols, (x_1, ...)) = R[i, j, k, l] = R^i_jkl
    Compute the Riemann tensor from the Christoffel symbols
```

```
    """
    dim = len(var)
    R = np.zeros([dim] * 4, dtype=Symbol)
    indx = [(i, j, k, l)
        for i in range(dim)
        for j in range(dim)
        for k in range(dim)
        for l in range(dim)
    ]

    for (a, b, r, s) in indx:
        R[a, b, r, s] += diff(C[a, b, s], var[r]) - diff(C[a, b, r], var[s])
        for k in range(dim):
            R[a, b, r, s] += C[a, k, r] * C[k, b, s] - C[a, k, s] * C[k, b, r]
    return R
```

Printing functions

[22]:
```
print_latex = False

def print_christoffel(C, var):
    """ A function for dsiplaying christoffels symbols """
    output = []
    for i in range(len(var)):
        txt = "$$"
        txt += "\\Gamma^" + sp.latex(var[i]) + "_{\\mu \\nu} ="
        txt += sp.latex(Matrix(C[i]))
        txt += "$$"
        print(txt) if print_latex else print()
        output.append(display(Latex(txt)))

    return output

def print_matrix(T):
    txt = "$$" + sp.latex(Matrix(T)) +"$$"
    print(txt) if print_latex else print()
    return display(Latex(txt))

def print_scalar(T):
    txt = "$$" + sp.latex(T) +"$$"
    print(txt) if print_latex else print()
    return display(Latex(txt))

def print_eq(eq):
    txt = "$$" + sp.latex(eq) +"=0" + "$$"
    print(txt) if print_latex else print()
    return display(Latex(txt))
```

Metric $g_{\mu\nu}$ for spherically symmetric spacetime

[23]:
```
t, r, th, ph = symbols("t, r, \\theta, \\phi")
x1 = r * cos(ph) * sin(th)
x2 = r * sin(ph) * sin(th)
x3 = r * cos(th)

one = Rational(1)
eta = sp.diag(one, -one, -one, -one)
var = (t, r, th, ph)
J = Matrix([t, x1, x2, x3]).jacobian(var)
g = np.array(simplify(J.T *eta* J))

a = sp.Function("\\alpha", real=True)(r)
b = sp.Function("\\beta", real=True)(r)
g[0, 0] *= exp(2 * a)
g[1, 1] *= exp(2 * b)
g_inv = get_g_inv(g)

print_matrix(g)
print_matrix(g_inv)
```

$$\begin{bmatrix} e^{2\alpha(r)} & 0 & 0 & 0 \\ 0 & -e^{2\beta(r)} & 0 & 0 \\ 0 & 0 & -r^2 & 0 \\ 0 & 0 & 0 & -r^2 \sin^2(\theta) \end{bmatrix}$$

$$\begin{bmatrix} e^{-2\alpha(r)} & 0 & 0 & 0 \\ 0 & -e^{-2\beta(r)} & 0 & 0 \\ 0 & 0 & -\frac{1}{r^2} & 0 \\ 0 & 0 & 0 & -\frac{1}{r^2 \sin^2(\theta)} \end{bmatrix}$$

[24]:
```
C = Christoffel(g, g_inv, var)
c = print_christoffel(C, var)
```

$$\Gamma^t_{\mu\nu} = \begin{bmatrix} 0 & \frac{d}{dr}\alpha(r) & 0 & 0 \\ \frac{d}{dr}\alpha(r) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Gamma^{r}_{\mu\nu} = \begin{bmatrix} e^{2\alpha(r)}e^{-2\beta(r)}\frac{d}{dr}\alpha(r) & 0 & 0 & 0 \\ 0 & \frac{d}{dr}\beta(r) & 0 & 0 \\ 0 & 0 & -re^{-2\beta(r)} & 0 \\ 0 & 0 & 0 & -re^{-2\beta(r)}\sin^{2}(\theta) \end{bmatrix}$$

$$\Gamma^{\theta}_{\mu\nu} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{r} & 0 \\ 0 & \frac{1}{r} & 0 & 0 \\ 0 & 0 & 0 & -\sin(\theta)\cos(\theta) \end{bmatrix}$$

$$\Gamma^{\phi}_{\mu\nu} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{r} \\ 0 & 0 & 0 & \frac{\cos(\theta)}{\sin(\theta)} \\ 0 & \frac{1}{r} & \frac{\cos(\theta)}{\sin(\theta)} & 0 \end{bmatrix}$$

[25]:

```
Rie = Riemann_tensor(C, var)
Ricci = contract(Rie, num_indx=4, upper=1, indx=(0, 2))

for i in range(4):
    print_scalar(Ricci[i, i].factor())
```

$$\frac{\left(r\left(\frac{d}{dr}\alpha(r)\right)^{2} - r\frac{d}{dr}\alpha(r)\frac{d}{dr}\beta(r) + r\frac{d^{2}}{dr^{2}}\alpha(r) + 2\frac{d}{dr}\alpha(r)\right)e^{2\alpha(r)}e^{-2\beta(r)}}{r}$$

$$-\frac{r\left(\frac{d}{dr}\alpha(r)\right)^{2} - r\frac{d}{dr}\alpha(r)\frac{d}{dr}\beta(r) + r\frac{d^{2}}{dr^{2}}\alpha(r) - 2\frac{d}{dr}\beta(r)}{r}$$

$$-\left(r\frac{d}{dr}\alpha(r) - r\frac{d}{dr}\beta(r) - e^{2\beta(r)} + 1\right)e^{-2\beta(r)}$$

$$-\left(r\frac{d}{dr}\alpha(r) - r\frac{d}{dr}\beta(r) - e^{2\beta(r)} + 1\right)e^{-2\beta(r)}\sin^{2}(\theta)$$

[26]:
```
R = contract(Ricci, g_inv=g_inv, upper=0).simplify()
print_scalar(R)
```

$$\frac{2\left(r^{2}\left(\frac{d}{dr}\alpha(r)\right)^{2}-r^{2}\frac{d}{dr}\alpha(r)\frac{d}{dr}\beta(r)+r^{2}\frac{d^{2}}{dr^{2}}\alpha(r)+2r\frac{d}{dr}\alpha(r)-2r\frac{d}{dr}\beta(r)-e^{2\beta(r)}+1\right)e^{-2\beta(r)}}{r^{2}}$$

[27]:
```
G = Ricci - Rational(1, 2) * R * g
for i in range(4):
    G[i, i] = G[i, i].simplify().factor()
    print_scalar(G[i, i])
```

$$\frac{\left(2r\frac{d}{dr}\beta(r)+e^{2\beta(r)}-1\right)e^{2\alpha(r)}e^{-2\beta(r)}}{r^{2}}$$

$$\frac{2r\frac{d}{dr}\alpha(r)-e^{2\beta(r)}+1}{r^{2}}$$

$$r\left(r\left(\frac{d}{dr}\alpha(r)\right)^{2}-r\frac{d}{dr}\alpha(r)\frac{d}{dr}\beta(r)+r\frac{d^{2}}{dr^{2}}\alpha(r)+\frac{d}{dr}\alpha(r)-\frac{d}{dr}\beta(r)\right)e^{-2\beta(r)}$$

$$r\left(r\left(\frac{d}{dr}\alpha(r)\right)^{2}-r\frac{d}{dr}\alpha(r)\frac{d}{dr}\beta(r)+r\frac{d^{2}}{dr^{2}}\alpha(r)+\frac{d}{dr}\alpha(r)-\frac{d}{dr}\beta(r)\right)e^{-2\beta(r)}\sin^{2}(\theta)$$

0.0.1 Stress-energy tensor $T_{\mu\nu}$ for perfect fluid

[28]:
```
p = sp.Function("p")(r)
rho = sp.Function("\\rho")(r)

UU = np.zeros((4, 4), dtype=sp.Rational)
UU[0, 0] = exp(2 * a)

T = (p + rho) * UU - p * g
for i in range(4):
    T[i, i] = T[i, i].simplify()
print_matrix(T)
```

$$\begin{bmatrix} \rho(r)e^{2\alpha(r)} & 0 & 0 & 0 \\ 0 & p(r)e^{2\beta(r)} & 0 & 0 \\ 0 & 0 & r^2p(r) & 0 \\ 0 & 0 & 0 & r^2p(r)\sin^2(\theta) \end{bmatrix}$$

0.0.2 Einstin equation

$R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} = 8\pi G T_{\mu\nu}$

[29]:
```
G_newton = sp.Symbol("G")

eq = []
for i in range(len(G)):
    eq.append((G[i, i] - 8 * pi * G_newton * T[i, i]).simplify())

# Some manual simplification
Rtt = sp.Symbol("R_{\\theta \\theta}")
eq[0] = eq[0] * r**2 / exp(2 * a)/exp(-2*b ) * (-1 )
eq[1] = eq[1] * r**2 * (-1)
eq[2] = eq[2] / r / exp(-2*b)
eq[3] = eq[3].subs(eq[2], Rtt)
for i in range(len(G)):
    print_eq(eq[i].simplify())
```

$$8\pi Gr^2\rho(r)e^{2\beta(r)} - 2r\frac{d}{dr}\beta(r) - e^{2\beta(r)} + 1 = 0$$

$$8\pi Gr^2p(r)e^{2\beta(r)} - 2r\frac{d}{dr}\alpha(r) + e^{2\beta(r)} - 1 = 0$$

$$-8\pi Grp(r)e^{2\beta(r)} + r\left(\frac{d}{dr}\alpha(r)\right)^2 - r\frac{d}{dr}\alpha(r)\frac{d}{dr}\beta(r) + r\frac{d^2}{dr^2}\alpha(r) + \frac{d}{dr}\alpha(r) - \frac{d}{dr}\beta(r) = 0$$

$$R_{\theta\theta}re^{-2\beta(r)}\sin^2(\theta) = 0$$

Define $e^{2\beta} = [1 - 2Gm(r)/r]^{-1}$

[30]:
```
m = sp.Function("m", Real=True)(r)
f = (1 - 2 * G_newton * m / r)**(-1)
eq1 = (eq[0] * exp(- 2 *a)).simplify().subs(b, Rational(1, 2) * log(f)).
  ↪simplify().expand().simplify()
s = sp.solve(eq1, m.diff(r))
eq1 = m.diff(r) - s[0]
```

Use $\nabla_\mu T^{\mu r} = 0 \implies (p+\rho)\partial_r \alpha = -\partial_r p$.

[31]:
```
eq2 = (eq[1] * r**2).subs(exp(2 * b), f).simplify()
s = sp.solve(eq2, a.diff(r))
eq2 = a.diff(r) - s[0]
eq2 = ((a.diff(r) - s[0]).subs(a.diff(r), - p.diff(r) / (p + rho))*(p + rho)).
  ↪simplify()
s = sp.solve(eq2, p.diff())
eq2 = p.diff(r) - s[0].factor()
```

The TOV-equation and equation for $m(r)$, both expressions are equal to 0.

[32]:
```
print_eq(eq1)
print_eq(eq2)
```

$$-4\pi r^2 \rho(r) + \frac{d}{dr} m(r) = 0$$

$$\frac{G\left(4\pi r^3 p(r) + m(r)\right)\left(\rho(r) + p(r)\right)}{r\left(-2Gm(r) + r\right)} + \frac{d}{dr} p(r) = 0$$