Computational Physics - Homework 6

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
%matplotlib inline
from scipy.integrate import solve_ivp
import scipy
import numpy as np
from math import cos,sin,exp,sqrt
from math import pi
```

Problem 1:

In [2]:
```
# define spectral intensity as function of s and Iv
print("f(s,Iv) = jv - alpha_v * Iv")
print("with step size h = S_n+1 - S_n")

# after one step of eulers method
print("Iv_n+1 = Iv_n + h * f(S_n+1, Iv_n+1)")
print("Since it is possible to isolate Iv_n+1 on one sideof the equation for each iter
print("Each step can be solved for explicitly")
```

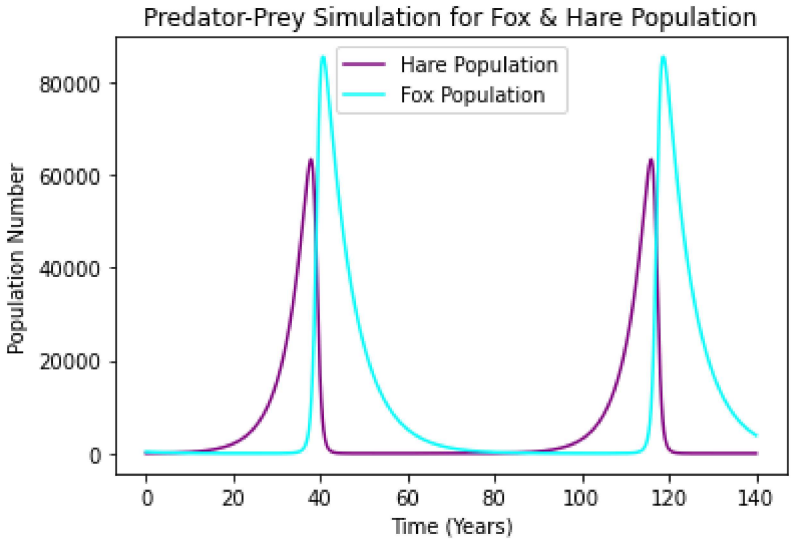
```
f(s,Iv) = jv - alpha_v * Iv
with step size h = S_n+1 - S_n
Iv_n+1 = Iv_n + h * f(S_n+1, Iv_n+1)
Since it is possible to isolate Iv_n+1 on one sideof the equation for each iteration
of euler's method
Each step can be solved for explicitly
```

Problem 2:

In [3]:
```
def population_sim(variables, t, params):
    x = variables[0] #hares
    y = variables[1] #foxes


    alpha = params[0]
    beta = params[1]
    delta = params[2]
    gamma = params[3]

    dxdt = alpha * x - beta * x * y
    dydt = - gamma * y + delta * x * y

    return([dxdt, dydt])

# initial population
y0 = [40,350] # [hares, foxes]
# y0 = [gamma/delta , alpha/beta] - steady state
t = np.linspace(0,140,num=1000) #time interval

# putting alpha, beta, gamma, delta in a list
params = [0.2, 2e-5, 2.5e-5, 0.15]


y = odeint(population_sim, y0, t, args=(params,))
```

```
hare_pop = y[:,0]
plt.title("Predator-Prey Simulation for Fox & Hare Population")
plt.plot(t,y[:,0], color = "purple", label = "Hare Population")
plt.plot(t,y[:,1], color = "cyan", label = "Fox Population")
plt.xlabel("Time (Years)")
plt.ylabel("Population Number")
plt.legend()
```

Out[3]: <matplotlib.legend.Legend at 0x185ba086be0>



In [4]:
```
print("The time interval between peaks is about:", 80, "years")
print("The max hare population is:", max(hare_pop), "hares")
```

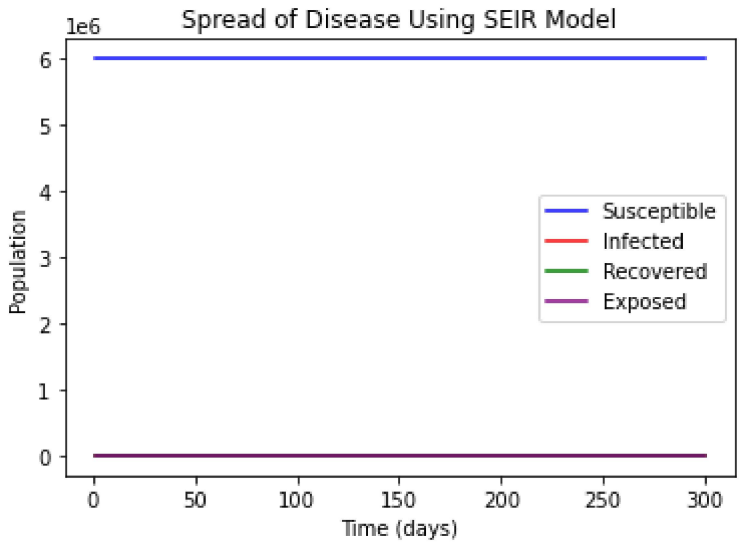
```
The time interval between peaks is about: 80 years
The max hare population is: 63343.67926344497 hares
```

Problem 3:

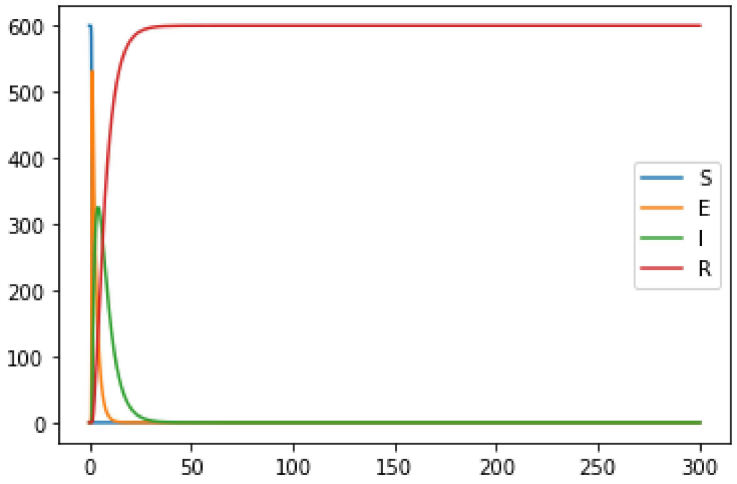
In [5]:
```
def SEIR_model(variables, t, params):
    S = variables[0]
    I = variables[1]
    R = variables[2]
    E = variables[3]

    beta = params[0]
    gamma = params[1]
    sigma = params[2]

    N = S + I + R + E

    # set up ODEs
    dSdt = (-beta * I * S)/N
    dEdt = (-beta * I * S)/N - sigma * E
    dIdt = (sigma * E) - (gamma * I)
    dRdt = (gamma * I)

    return ([dSdt, dEdt, dIdt, dRdt])

t = list(range(1,301))
params = [0.514, 0.20, 0.50] #beta, gamma, sigma per day
y0 = [5999998, 2, 0, 0] #susceptible, infected, removed, exposed
```

```
y = odeint(SEIR_model, y0, t, args = (params,))

plt.title("Spread of Disease Using SEIR Model")
plt.plot(t, y[:,0], color = "blue", label = "Susceptible")
plt.plot(t, y[:,1], color = "red", label = "Infected")
plt.plot(t, y[:,2], color = "green", label = "Recovered")
plt.plot(t, y[:,3], color = "purple", label = "Exposed")
plt.xlabel("Time (days)")
plt.legend()
plt.ylabel("Population")

# so this is wrong
```

Out[5]:
```
Text(0, 0.5, 'Population')
```



In [6]:
```
def seir_f(t, y, beta, sigma, gamma):
    s, e, i, r = y
    return np.array([-beta * i * s,
                     -sigma * e + beta * i * s,
                     -gamma * i + sigma * e,
                     gamma * i])

# try some parameter values
beta = 0.514
sigma = 0.50
gamma = 0.20

sol = solve_ivp(seir_f, [0, 300], [599.99, 2e-4, 0, 0],
                args=(beta, sigma, gamma))

fig = plt.figure(); ax = fig.gca()
curves = ax.plot(sol.t, sol.y.T)
ax.legend(curves, ['S', 'E', 'I', 'R']);

# but also this is wrong
# but i think this is less wrong, so i am going with this
```



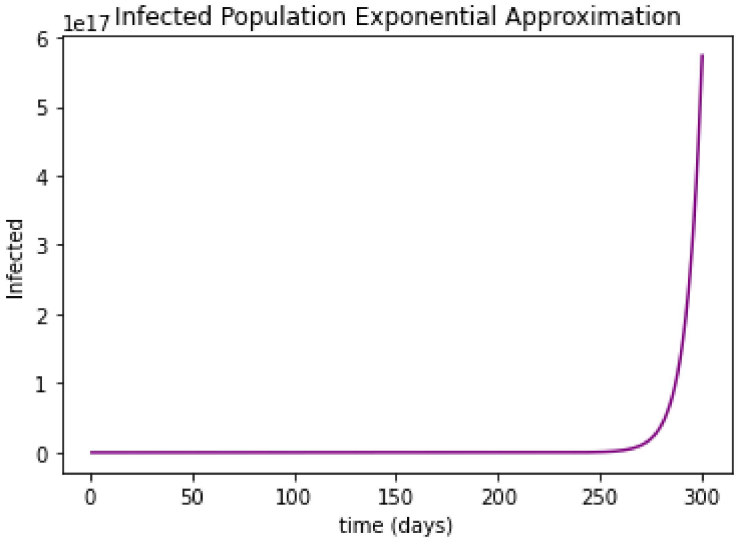
In [7]:
```
infected = list(sol.y.T[:,2])
print("The max number of infected people is:", max(infected))
index = infected.index(max(infected))
print("This occurs at:", t[index], "days")
print("It looks like its around:", 15, "days on the plot" )
# 133 days makes more sense maybe? but the plot says 15. the plot is probably wrong
```

```
The max number of infected people is: 325.5038907198857
This occurs at: 133 days
It looks like its around: 15 days on the plot
```

In [8]:
```
print("I1 = I0 * e^kt1 and I2 = I0 * e^kt2")
print("setting these equal to each other and dividing:")
print("I1/I2 = e^kt1/e^kt2")
print("flip and solving for k:")
print("I2/I1 = kt2 - kt1")
print("k = (I2/I1) * (t2 - t1)^-1")


t = list(range(1,301))
I0 = 2
k = 0.13399 # from using equation at two points
def I(k, t):
    i = I0 * exp(k * t)
    return i

values = []
for i in t:
    v = I(k,i)
    values.append(v)

plt.title("Infected Population Exponential Approximation")
plt.plot(t, values, color = 'purple')
plt.xlabel("time (days)")
plt.ylabel("Infected")
```

```
I1 = I0 * e^kt1 and I2 = I0 * e^kt2
setting these equal to each other and dividing:
I1/I2 = e^kt1/e^kt2
flip and solving for k:
I2/I1 = kt2 - kt1
k = (I2/I1) * (t2 - t1)^-1
```

Out[8]:
```
Text(0, 0.5, 'Infected')
```

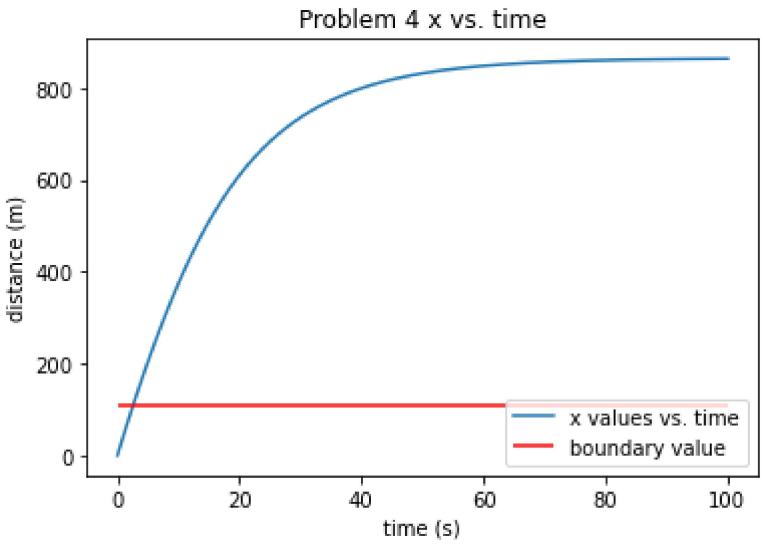


Problem 4:

In [9]:
```
def baseball_path(variables, t):
    # variable bois
    x = variables[0]
    x_prime = variables[1]
    y = variables[2]
    y_prime = variables[3]

    # ODEs
    dxdt = x_prime

    dx_primedt = -0.00258 * (0.29 + (0.22/(1 + exp((sqrt(x_prime**2 + y_prime**2)/5.2)

    dydt = y_prime

    dy_primedt = -0.00258 * (0.29 + (0.22/(1 + exp((sqrt(x_prime**2 + y_prime**2)/5.2)
    return [dxdt, dx_primedt, dydt, dy_primedt]

# initial stuff
t = np.linspace(0, 100, num = 301)
x0 = 0
y0 = 0.9
v = 51.5
# 45 to start, then 20 ( too low), 80, 55, 52, 51.5 (best)

# guess initial xprime, yprime
xprime = v * cos(pi/6)
yprime = v * sin(pi/6)

initial = [x0, xprime, y0, yprime]
y = scipy.integrate.odeint(baseball_path, initial, t)
baseball_x = y[:, 0]
baseball_y = y[:, 2]
plt.plot(t, baseball_x, label = "x values vs. time")

plt.hlines(106.9, t[0], t[-1], colors = "red", label = "boundary value")
plt.title("Problem 4 x vs. time")
plt.xlabel("time (s)")
```

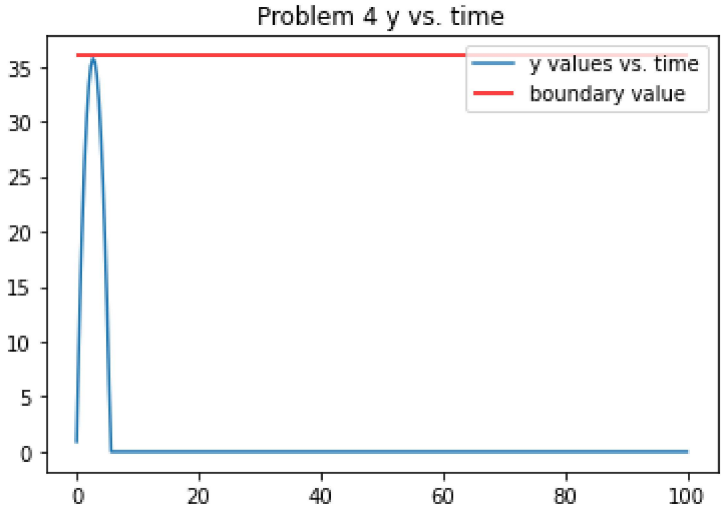
```
plt.ylabel("distance (m)")
plt.legend()
```

Out[9]:
```
<matplotlib.legend.Legend at 0x185bb948e50>
```



In [10]:
```
py_baseball_y = baseball_y.tolist()
tolerance = 0.0001
max_y = max(py_baseball_y)

index = 0
for y in py_baseball_y:
    if y < 0:
        py_baseball_y[index] = 0
    index = index + 1

plt.plot(t, py_baseball_y, label = "y values vs. time")
plt.hlines(36.0, t[0], t[-1], colors = "red", label = "boundary value")
plt.title("Problem 4 y vs. time")
plt.legend()
```

Out[10]:
```
<matplotlib.legend.Legend at 0x185ba88c970>
```

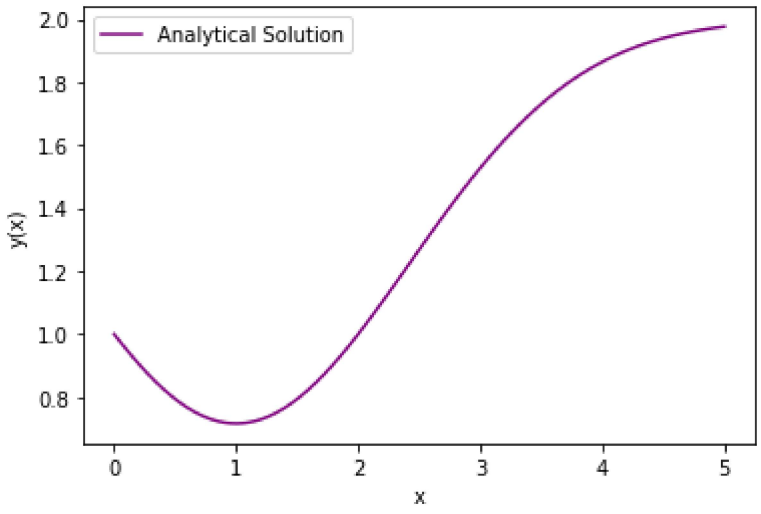


In [11]:
```
print("if the ball did not hit the stadium, it would have traveled ", max(baseball_x),
```

```
if the ball did not hit the stadium, it would have traveled  863.8293947806221 meters
```

Problem 5:

In [12]:
```
def f(x, y):
    return 0.5 * (x - 1) * (2 - y)

def analytical_solution(x):
    return 2 - np.exp(x/2 - (x/2)**2)

def ode_func(x, y):
    return 0.5 * (x - 1) * (2 - y)

def solve_ode_rk4(a, b, y0, tol):
    result = solve_ivp(ode_func, [a, b], [y0], method='RK45', rtol=tol, atol=1e-12)
    return result

def solve_ode_rkf45(a, b, y0, tol):
    result = solve_ivp(ode_func, [a, b], [y0], method='RK45', rtol=tol, atol=1e-12)
    return result

# initial stuff
x0 = 0
y0 = 1
x_values = np.linspace(x0, 5, 100)
y_analytical = [analytical_solution(x) for x in x_values]

# plot the ODE and analytical solution
plt.plot(x_values, y_analytical, label='Analytical Solution', color='purple')
plt.xlabel('x')
plt.ylabel('y(x)')
plt.legend()
plt.show()
```



In [13]:
```
# initial conditions
a = 0
b = 5
y0 = 1

# tolerances to check for accuracy
tols = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10]

function_calls_rk4 = []
relative_errors_rk4 = []
```

```
function_calls_rkf45 = []
relative_errors_rkf45 = []

for tol in tols:
    # solving the ODE using RK4
    result_rk4 = solve_ode_rk4(a, b, y0, tol)
    function_calls_rk4.append(result_rk4.nfev)
    relative_errors_rk4.append(max(result_rk4.y[0]) - analytical_solution(b))

    # solving the ODE using RKF45
    result_rkf45 = solve_ode_rkf45(a, b, y0, tol)
    function_calls_rkf45.append(result_rkf45.nfev)
    relative_errors_rkf45.append(max(result_rkf45.y[0]) - analytical_solution(b))

# minimum function calls for a relative error of 1e-6
min_calls_rk4 = function_calls_rk4[relative_errors_rk4.index(min(relative_errors_rk4))
min_calls_rkf45 = function_calls_rkf45[relative_errors_rkf45.index(min(relative_errors

print(f"Minimum function calls (RK4) for a relative error of 1e-6: {min_calls_rk4}")
print(f"Minimum function calls (RKF45) for a relative error of 1e-6: {min_calls_rkf45}
```

```
Minimum function calls (RK4) for a relative error of 1e-6: 20
Minimum function calls (RKF45) for a relative error of 1e-6: 20
```

Problem 6:

In [14]:

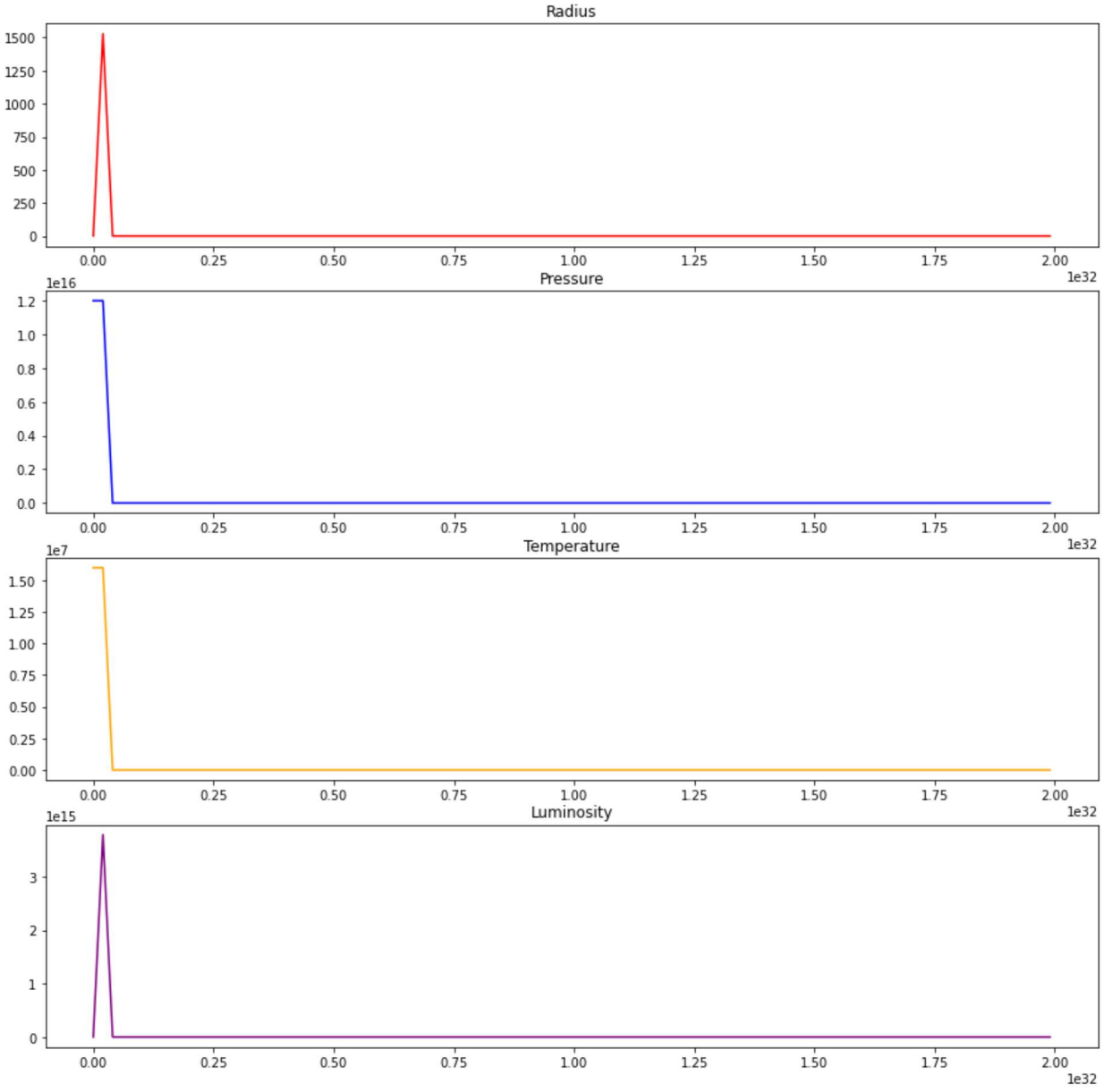
```
#constants
c = 3e8
kB = 1.380649e-23
G = 6.6743e-11
def hydro_equillibrium(variables, M):
    # i guess eps is the 4th initial condition?
    r = variables[0]
    P = variables[1]
    T = variables[2]
    L = variables[3]
    M = 0

    T6 = T / 1e6

    a = 7.565e-16
    p = (9.91e-28) * ((P - a * T**4/3) / (kB * T))
    k = 0.035 + 6.44e18 * (p / T**3.5)
    eps = 0.136 * p * T6**-2/3 * exp(-33.80 * T6**-1/3) + 4.69e11 * p**2 * T6**-3 * ex

    # all derivatives wrt to dM
    dr = 1 / (4 * pi * r**2 * p)
    dP = (-G*M) / (4 * pi * r**4)
    dT = (-3 * k * L) / (64 * pi**2 * a * c * T**3 * r**4)
    dL = eps

    return [dr, dP, dT, dL]

# initial stuff
M = np.linspace(1, 100 * 1.99e30, 100) # mass interval

r0 = 1
P0 = 1.2e16
T0 = 16e6
L0 = 0
```

```
variables = [r0, P0, T0, L0]

y = scipy.integrate.odeint(hydro_equillibrium, variables, M)
```

```
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\odepack.py:247: ODEintWarn
ing: Excess work done on this call (perhaps wrong Dfun type). Run with full_output =
1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
```

In [15]:
```
fig, ax = plt.subplots(nrows=4, ncols=1, sharex=False, figsize = (15,15))
ax[0].plot(M, y[:,0], label = 'radius', color = 'red')
ax[0].set_title("Radius")
ax[1].plot(M, y[:,1], label = 'pressure', color = 'blue')
ax[1].set_title("Pressure")
ax[2].plot(M, y[:,2], label = 'temperature', color = 'orange')
ax[2].set_title("Temperature")
ax[3].plot(M, y[:,3], label = 'luminosity', color = 'purple')
ax[3].set_title("Luminosity")
```

Out[15]:
```
Text(0.5, 1.0, 'Luminosity')
```



In [16]:
```
print("The core pressure is:", 1.2e16, "Pa")
print("The core temperature is:", 1.0e7, "K")
```

```
The core pressure is: 1.2e+16 Pa
The core temperature is: 10000000.0 K
```

In [17]:
```
# sorry i can not really get an answer for the last two parts with the absolute mess
# that i created  ):
```