```python
import numpy as np
from FixedVariables import gamma
from TimeIntegration import make_mesh
from FixedVariables import xmax, xmin
from InputVariables import var

G = gamma
Ms = 1.44061

# compute the pressure, density, velocity and the speed of sound from an 3-dimensional array of primitive variables.
def p_(w):
    return w[-1]

def rho_(w):
    return w[0]

def v_(w):
    return w[1]

def a_(w):
    return np.sqrt(G*p_(w)/rho_(w))

# computes the values of density, velocity and pressure in the five different regions
def RegionR():
    return [1, 0, 1]

def RegionL():
    return [8, 0, 8/G]

def Region1():
    R = RegionR()
    one = [0, 0, 0]
    one[0] = rho_(R)/(2/((G+1)*np.square(Ms)) + (G-1)/(G+1))
    one[1] = (2/(G+1)*(Ms - 1/Ms))
    one[2] = ((2*G*np.square(Ms))/(G+1)-(G-1)/(G+1))*p_(R)
    return one

def Region2():
    one = Region1()
    L = RegionL()
    two = [0, 0, 0]
    two[1] = v_(one)
    two[2] = p_(one)
    two[0] = np.power(p_(two)/p_(L), 1/G)*rho_(L)
    return two
```

```python

def RegionE(i, t):
    L = RegionL()
    E = [0, 0, 0]
    E[1] = (2/(G+1))*(a_(L) + i/t)
    a = a_(L) - (G-1)*v_(E)/2
    E[2] = p_(L)*np.power(a/a_(L), 2*G/(G-1))
    E[0] = (G*p_(E))/(np.square(a_(L) - (G-1)*v_(E)/2))
    return E

# computes for time t the analytic solution as a (nx, 3)-dimensional array
def analytic_solution(nx,t):
    x = make_mesh(xmin, xmax, nx)
    x.pop(0)
    x.pop(-1)
    a_sol = []
    R = RegionR()
    L = RegionL()
    one = Region1()
    two = Region2()
    for i in range(nx):
        if x[i] < -a_(L)*t:
            a_sol.append(L)
        elif x[i] >= -a_(L)*t and x[i] <= (v_(two)-a_(two))*t:
            a_sol.append(RegionE(x[i],t))
        elif x[i] > (v_(two)-a_(two))*t and x[i] < v_(two)*t:
            a_sol.append(two)
        elif x[i] >= v_(two)*t and x[i] <= Ms*t:
            a_sol.append(one)
        else:
            a_sol.append(R)
    return a_sol

# computes from the analytic solution the values for density, velocity and pressure
def analytic_solver(nx, t):
    A = analytic_solution(nx, t)
    if var == 'mass density':
        a = [A[i][0] for i in range(nx)]
    elif var == 'velocity':
        a = [A[i][1] for i in range(nx)]
    elif var == 'pressure':
        a = [A[i][2] for i in range(nx)]
    return a


```