```python
from UpwindSolver import upwind_solver
from AnalyticSolution import analytic_solver
from EOC import cv_and_conquer
from InputVariables import cvAnalysis, nx, iniCond, var, nu_max
from FixedVariables import xmin, xmax
from TimeIntegration import make_mesh
from timeit import default_timer as timer
import matplotlib.pyplot as plt

start = timer()

if cvAnalysis == 'false':
    solution, time = upwind_solver(nx)
    plot = 'solution'
elif cvAnalysis == 'true':
    convergence = cv_and_conquer(nu_max)
    plot = 'convergence'
elif cvAnalysis == 'dissipation':
    num_sol = upwind_solver(nx)[-2][-1]
    a_sol = analytic_solver(nx, 0.2)
    plot = 'dissipation'

end = timer()

# defines the elapsed time between the start of the computation and the end of the computation (the computational
# effort to plot the results is not counted
def chrono(start, end):
    elapsed = end-start
    return elapsed

print("time elapsed = ",chrono(start,end))

# plots the solution for every time snap
if plot == 'solution':
    X = make_mesh(xmin, xmax, nx)
    X.pop(0)
    X.pop(-1)
    for i in range(len(solution)):
        if iniCond == 'acoustic' and var == 'mass density':
            plt.axis([0, 1, 0.1399, 0.1415])
        if iniCond == 'acoustic' and var == 'velocity':
            plt.axis([0, 1, -0.005, 0.005])
        if iniCond == 'acoustic' and var == 'pressure':
            plt.axis([0, 1, 0.0999, 0.1011])
        elif iniCond == 'fixed':
            plt.axis([-0.5, 0.5, 0, 9])
```

```
        plt.xlabel('x')
        plt.ylabel(var)
        plt.title(" t = %1.3f" %time[i])
        plt.plot(X, solution[i], marker='.')
        plt.show()

if plot == 'convergence':
    NU = [i for i in range(3,nu_max+1)]
    plt.axis([3,nu_max, -0.3, 0.7])
    plt.title("Empirical Order of Convergence")
    plt.xlabel("nu")
    plt.ylabel("EOC_nu")
    plt.plot(NU, convergence)
    plt.show()

if plot == 'dissipation':
    X = make_mesh(xmin, xmax, nx)
    X.pop(0)
    X.pop(-1)
    plt.plot(X, num_sol, marker='.')
    plt.plot(X, a_sol, marker='.')
    plt.axis([-0.5, 0.5, 0, 9])
    plt.title('t = 0.2')
    plt.xlabel('x')
    plt.ylabel(var)
    plt.legend(['numerical solution', 'analytical solution'])
    plt.show()

```