~\CSI Code\Ullage Volume Optimization Code\nos_blowdown_simulation.py

```python
# ---------------------------------------------------------------
# By Aadam Awad for the Columbia Space Initiative Rocketry Team
# ---------------------------------------------------------------

# The goal of this program is to perform numerical simulations to approximate the behavior of liquid nitrous oxide through a choked injecotr for a nitrous oxide hybrid rocket
# motor. It uses the REFPROP library, interfacing in Python via CoolProp, and displays the data in Matplotlib. This program should produce an approximate graph
# of pressure vs. time given several fixed constraints: injector CdA, nitrous oxide mass, and outside temperature as initial conditions.

# This is the second version of this model, designed to approximate tank blowdown while incorporating compressibility of both the vapor and nitrous oxide.
# Additionally, this model is modifying the mass discharge parameters to be based on an injector CdA rather than a fixed mass flow rate. This leaves
# more free variables to be tweaked before finalizing engine design, but will produce more accurate results and produce an accurate mass flow rate vs. time graph.

# Assumptions to simplify the model:
# - Heat transfer through the plumbing to the nitrous oxide is second order during blowdown
# - The nitrous oxide remains saturated throughout the burn (rate of boil-off is greater than rate of discharge). Empirical tests have shown this is valid
#   for initial ullage volume ratios exceeding 10-20%.
# - The bulk liquid nitrous is the same temperature throughout the burn (there is no temperature gradient between different regions of the liquid nitrous)
# - All choking occurs at the injector (modeled by Burnells) and the rest of the plumbing is sufficiently sized such that no choking occurs

import json, CoolProp.CoolProp as CP
from CoolProp.CoolProp import PropsSI
import matplotlib.pyplot as plt
import equilibrium_finder as eqf
import numpy as np
import sys
import math
from decimal import Decimal, ROUND_DOWN

sys.setrecursionlimit(50000)

# REFPROP path setting
CP.set_config_string(CP.ALTERNATIVE_REFPROP_PATH, 'c:\Program Files (x86)\REFPROP')

#--------------------------------------------------------------------------------
# Initial Parameters:

# Initial liquid nitrous oxide mass in kg
targetLiquidMass = 14

# Orifice count and diameter
oCount = 16
oArea = 0.0000064

# Total orifice area in m^2
a0 = oCount * oArea

```

```python
# Injector Cd (dimensionless)
Cd = 0.75

# Expected tank temperature at launch in K
temperature = 300

# Total tank volume in m^3
tankV = 0.026622735858012
tankVGal = Decimal(tankV*264.172)

# IMPORTANT: Time step that controls model fidelity, dV for initial conditions
timeStep = 0.001
dV = 0.0001

# Arrays for the final output
denseArr = []
tempArr = []
pressArr = []
massFlowArr = []
timeArr = []

#--------------------------------------------------------------------------------
# *Defining Initial States of the Bulk Fluid and Vapor*
# Assuming saturation conditions, the temperature and total tank volume can be used to find initial conditions bulk liquid volume, density,
# and compressibility, and bulk vapor volume, density, and compressibility.

# This is performed using a recursive function that iterates through values until it finds a saturated vapor - liquid mixture at the initial temperature and mass

initVaporCompress = PropsSI ("Z", "T", temperature, "Q", 1, "REFPROP::Nitrous oxide")

def initParams (liquidV, vaporV): # Call this function starting with liquidVol = 0, vaporVol = tankV
    newLiquidV = liquidV + dV
    newVaporV = vaporV - dV

    newLiquidMass = newLiquidV * PropsSI ("D", "T", temperature, "Q", 0, "REFPROP::Nitrous oxide")
    newVaporMass = newVaporV * PropsSI ("D", "T", temperature, "Q", 1, "REFPROP::Nitrous oxide") / initVaporCompress

    if newLiquidMass >= targetLiquidMass:
        return newLiquidV, newVaporV, newLiquidMass, newVaporMass, (newLiquidMass + newVaporMass) # Initial liquid volume, vapor volume, and total mass reading for the load cell

    return initParams (newLiquidV, newVaporV)

#--------------------------------------------------------------------------------
# *Blowdown Iterative Model*
# Architecture:
# - Starts with initial parameters given by initParams (0, tankV)
# (1) Remove some volume of liquid according to Burnell's equation for mass flow
# (2) Recalculate liquid and vapor volumes (with compressibility) after the new added volume decreased pressures
# (3) Calculate new bulk liquid temperature and vapor pressure from boil-off
# - Repeat steps (1) - (3) for each volume step dV

```

```python
liquidVol, vaporVol, liquidMass, vaporMass, totalMass = initParams (0, tankV)

def burnellEmpCoeff (press): # Linear approximation for the empirical coefficient C for Burnell's equation.
    return -0.000000015267*press + 0.2279

def injectorMassFlow (temp): # Calculates the volumetric flow rate per unit time dV/dt across the injector
     press = PropsSI ("P", "T", temp, "Q", 0, "REFPROP::Nitrous oxide")
     dens = PropsSI ("D", "T", temp, "Q", 0, "REFPROP::Nitrous oxide")
     mDot = Cd*a0*np.sqrt(2*dens*(press - press*(1 - burnellEmpCoeff(press)))) # Burnell's equation
     return mDot

def vaporizationH (temp): # Enthalpy of vaporization for nitrous oxide
    return PropsSI("H", "T", temp, "Q", 1, "REFPROP::Nitrous oxide") - PropsSI("H", "T", temp, "Q", 0, "REFPROP::Nitrous oxide")

def specificHeat (temp): # Isobaric specific heat capacity of nitrous oxide
    return PropsSI("C", "T", temp, "Q", 1, "REFPROP::Nitrous oxide")

def main (): # Performs the blowdown calculations
    curLiqVol = liquidVol
    curVapVol = vaporVol
    curTemp = temperature
    curPress = PropsSI ("P", "T", temperature, "Q", 0, "REFPROP::Nitrous oxide")

    inEquilibrium = False

    tracker = 0

    while curLiqVol >= 0.001:

        dens = PropsSI ("D", "T", curTemp, "Q", 0, "REFPROP::Nitrous oxide")
        mDot = injectorMassFlow(curTemp)
        deltaV = (mDot / dens) * timeStep

        if (inEquilibrium == True): # The system is in equilibrium, so we can remove some volume.
            tracker += 1
            timeArr.append (tracker * timeStep)

            curPress = curPress * (curVapVol / (curVapVol + deltaV))
            pressArr.append (curPress * 0.000145038)

            curLiqVol -= deltaV
            curVapVol += deltaV * PropsSI ("Z", "T", curTemp, "Q", 0, "REFPROP::Nitrous oxide")

            massFlowArr.append (mDot)

            inEquilibrium = False

        else:
            massSlice = deltaV / 0.01
            curVapVol += massSlice * PropsSI ("Z", "T", curTemp, "Q", 0, "REFPROP::Nitrous oxide") / PropsSI ("D", "T", curTemp, "Q", 1, "REFPROP::Nitrous oxide")
```

```
            curTemp = curTemp - (massSlice/(curLiqVol*dens)) * (vaporizationH(curTemp)/specificHeat(curTemp))

            curVapMass = curVapVol * PropsSI ("D", "P", curPress, "Q", 1, "REFPROP::Nitrous oxide")
            curPress = curPress * (1 + massSlice / curVapMass)

            tempPress = PropsSI ("P", "T", curTemp, "Q", 0, "REFPROP::Nitrous oxide")

            if (curPress >= tempPress):
                inEquilibrium = True

main()

#--------------------------------------------------------------------------------
# *Graphical interface*

plt.figure(1)

maxValPress = max(pressArr)

plt.ylim(0, maxValPress + 100)
plt.plot(timeArr, pressArr)
plt.xlabel('Time (s)')
plt.ylabel('Pressure (PSI)')
plt.title('Tank Volume: ' + str(tankVGal.quantize(Decimal('.01'))) + ' gal. Initial Nitrous Temp (K): ' + str(temperature))
plt.grid(linewidth = '0.5')

plt.figure(2)

maxValMDot = max(massFlowArr)

plt.ylim(0, maxValMDot + 1)
plt.plot(timeArr, massFlowArr)
plt.xlabel('Time (s)')
plt.ylabel('Mass Flow Rate (kg/s)')
plt.title('Tank Volume: ' + str(tankVGal.quantize(Decimal('.01'))) + ' gal. Initial Nitrous Temp (K): ' + str(temperature))
plt.grid(linewidth = '0.5')

plt.show()
```