B EXAMPLE CODE

```python
import numpy as np
import cxvpy as cp

# We assume the following is already defined:
# - `A`, list of matrices A_i
# - `reserves`, list of vectors R_i
# - `fees`, list of fees for each CFMM i
# - `num_tokens`, list of n_i
# - `t`, amount of asset 1 provided to the network

current_assets = np.array([t, 0, 0])

# Build variables
deltas = [cp.Variable(n_i, nonneg=True) for n_i in num_tokens]
lambdas = [cp.Variable(n_i, nonneg=True) for n_i in num_tokens]
psi = cp.sum(A_i @ (L - D) for A_i, D, L in zip(A, deltas, lambdas))

# Objective is to trade t of asset 1 for a maximum amount of asset 3
obj = cp.Maximize(psi[2])

# Reserves after trade
new_reserves = [R + gamma_i*D - L \
    for R, gamma_i, D, L in zip(reserves, fees, deltas, lambdas)]

# Trading function constraints
cons = [
    # Balancer pool with weights 3, 2, 1
    cp.geo_mean(new_reserves[0], p=np.array([3, 2, 1])) \
        >= cp.geo_mean(reserves[0], p=np.array([3, 2, 1])),

    # Uniswap v2 pools
    cp.geo_mean(new_reserves[1]) >= cp.geo_mean(reserves[1]),
    cp.geo_mean(new_reserves[2]) >= cp.geo_mean(reserves[2]),
    cp.geo_mean(new_reserves[3]) >= cp.geo_mean(reserves[3]),

    # Constant sum pool
    cp.sum(new_reserves[4]) >= cp.sum(reserves[4]),
    new_reserves[4] >= 0,

    # Allow all assets at hand to be traded
    psi + current_assets >= 0
]

# Set up and solve problem
prob = cp.Problem(obj, cons)
prob.solve()

print(f"amount of asset 3 received: {psi[2].value}")
```