## Question 2 : Large Scale Stochastic Dominance

```
import cvxpy as cp
import numpy as np
import pandas as pd
```

```
# df = pd.read_excel(r'C:\Users\aayus\Documents\GitHub\StochOpt\stochastic-dominance\returns_data.xlsx')
# returns = df.iloc[:,1:].to_numpy()[1:]
# print(returns)
```

```
returns = np.array([[ 0.004, -0.025,  0.009,  0.012,  0.047, -0.019,  0.006, -0.037,
         0.025,  0.021,  0.017,  0.019],
       [ 0.014,  0.   , -0.039,  0.016, -0.006,  0.07 , -0.021, -0.022,
         0.019,  0.025,  0.054,  0.04 ],
       [ 0.001,  0.006,  0.005,  0.019,  0.016,  0.057, -0.052,  0.027,
         0.039,  0.   ,  0.011,  0.002],
       [-0.012, -0.021,  0.062,  0.036, -0.002, -0.038,  0.015, -0.003,
         0.024,  0.012,  0.048, -0.007],
       [-0.043,  0.005,  0.023,  0.   ,  0.023,  0.04 ,  0.034,  0.029,
        -0.013, -0.04 ,  0.011,  0.003],
       [ 0.015, -0.027, -0.01 , -0.027,  0.002,  0.038,  0.056, -0.004,
         0.08 ,  0.001,  0.013,  0.026],
       [-0.001,  0.011,  0.056, -0.024,  0.019, -0.048, -0.015,  0.019,
         0.062,  0.023,  0.002, -0.017],
       [ 0.039,  0.03 ,  0.003, -0.004,  0.016, -0.021,  0.003,  0.018,
        -0.026, -0.022,  0.026,  0.073],
       [ 0.017,  0.02 , -0.024, -0.004,  0.019,  0.039, -0.03 ,  0.025,
         0.021,  0.054, -0.011,  0.056],
       [ 0.108, -0.003,  0.061,  0.008,  0.024, -0.037, -0.013,  0.053,
        -0.009, -0.021,  0.026, -0.009]])
```

```
mean_returns= np.resize(returns.mean(axis=1),(10,1))
print("mean",mean_returns)
```

```
mean [[0.00658333]
 [0.0125    ]
 [0.01091667]
 [0.0095    ]
 [0.006     ]
 [0.01358333]
 [0.00725   ]
 [0.01125   ]
 [0.01516667]
 [0.01566667]]
```

```
np.mean(mean_returns)
```

```
0.010841666666666666
```

```
assets = 10
senarios = 12
```

```
# Compute of E(max(0,n-y))
Y_weights = (1/assets)*(np.ones((assets,1)))
Y_returns = np.sort(((returns.T)@Y_weights).flatten())
V = []
for eta in Y_returns:
    v_j = np.sum((eta-Y_returns)[Y_returns< eta])/(len(Y_returns))
    V.append(v_j)
```

```
weights  = cp.Variable(shape=(assets,1),name="weights")
S = cp.Variable(shape=(senarios,senarios),name="slack")
X_returns = returns.T@weights

constraints = []
for j,eta in enumerate(Y_returns):
    for i,x in enumerate(X_returns):
        constraints.append(x+S[i,j]>=eta)

for j,v_j in enumerate(V):
    constraints.append((1/(len(Y_returns)))*cp.sum(S[:,j])<=v_j)

constraints.extend([cp.sum(weights)==1,S>=0,weights>=0])

objective = cp.Maximize((mean_returns.T@weights))
problem = cp.Problem(objective, constraints)
```

```
problem.solve()
```

```
0.013845284660021651
```

```
X_returns = np.sort((returns.T@weights.value).flatten())
print("Optimized Returns",X_returns)
print("Equally weighted Returns",Y_returns)
```

```
Optimized Returns [-0.0017     -0.0004      0.00319999  0.0053      0.00839842  0.01028123
  0.0141204   0.01562975  0.01703332  0.02170916  0.03148095  0.04109018]
Equally weighted Returns [-0.0017 -0.0004  0.0032  0.0053  0.0081  0.0105  0.0142  0.0146  0.0158
  0.0186  0.0197  0.0222]
```

```
print("Optimal Weights",weights.value)
```

```
Optimal Weights [[8.05783707e-10]
 [3.66586340e-02]
 [1.79852513e-09]
 [7.21659992e-02]
 [5.90082030e-09]
 [1.89657805e-01]
 [3.18553277e-10]
 [1.63699545e-01]
 [2.84291684e-01]
 [2.53526323e-01]]
```

Start coding or generate with AI.