Question 2 Deepest Event Cut Method

```
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)
```

```
[[ 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]]
```

```
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)
```

```
dict_eta_V = dict(zip(Y_returns,V))
```

```
k=0
Eta = {Y_returns[-1]:Y_returns<=Y_returns[-1]}
# Eta = {}
while True:

    weights = cp.Variable(shape=(assets,1),name="weights")

    objective = cp.Maximize((mean_returns.T@weights))  # Objective function for first stage problem


    constraints = []
    for et in Eta:
        events = Eta[et]
        g_x_events = returns.T[events,:]@(weights)

        constraints.append(((1/(len(events)))*cp.sum(et -g_x_events )) <= dict_eta_V[et])

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

    # Solve Problem
    problem = cp.Problem(objective, constraints)
    problem.solve()

    Z_x =returns.T@(weights.value).flatten()

    # Calculate deltas
    delta_j = []
    for eta in Y_returns:
        delta_j_temp =  np.sum((eta-Z_x)[Z_x< eta])/(len(Z_x))-dict_eta_V[eta]
        delta_j.append(delta_j_temp)


    # Find out max eta
    delta_max = np.max(delta_j)
    eta_max = Y_returns[np.argmax(delta_j)]

    if delta_max <= 0:
        print("Problem",problem.value)
        print("weights",weights.value)
        print("Conditions satisfied")
        break
    else:
        print(f"iteration {k} events",np.argwhere(Z_x<eta_max).T)
        Eta[eta_max] = Z_x<eta_max

    k= k+1
    print("Iteration no. ",k)
```

```
iteration 0 events [[ 1  3  5  6  8  9 11]]
Iteration no.  1
iteration 1 events [[ 1  3  5  6  8 10]]
Iteration no.  2
iteration 2 events [[ 2  3  6 10]]
Iteration no.  3
iteration 3 events [[3 6]]
Iteration no.  4
iteration 4 events [[1 3 5 6 9]]
Iteration no.  5
iteration 5 events [[1 3]]
Iteration no.  6
iteration 6 events [[1 3 5 6]]
Iteration no.  7
iteration 7 events [[1 3 6]]
Iteration no.  8
iteration 8 events [[ 1  2  3  5  6  9 10]]
Iteration no.  9
iteration 9 events [[1 3 9]]
Iteration no.  10
iteration 10 events [[1 3]]
Iteration no.  11
iteration 11 events [[ 2  3  6 10]]
Iteration no.  12
iteration 12 events [[6]]
Iteration no.  13
iteration 13 events [[1 3 6]]
Iteration no.  14
iteration 14 events [[3]]
Iteration no.  15
iteration 15 events [[1 3 5 6]]
Iteration no.  16
iteration 16 events [[3 6]]
Iteration no.  17
iteration 17 events [[1 2 3 4 5 6 9]]
Iteration no.  18
Problem 0.013845284079889785
weights [[2.67754418e-09]
 [3.66589482e-02]
 [4.67283939e-09]
 [7.21657964e-02]
 [1.65301415e-08]
 [1.89657753e-01]
 [1.60744160e-09]
 [1.63699784e-01]
 [2.84291176e-01]
 [2.53526517e-01]]
Conditions satisfied
```

```
print("Problem",problem.value)
print("weights",weights.value)
```

```
Problem 0.013845284079889785
weights [[2.67754418e-09]
 [3.66589482e-02]
 [4.67283939e-09]
 [7.21657964e-02]
 [1.65301415e-08]
```

```
       [1.89657753e-01]
       [1.60744160e-09]
       [1.63699784e-01]
       [2.84291176e-01]
       [2.53526517e-01]]
```

Start coding or generate with AI.