Question 2: Inverse 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)
```

```
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]]
```

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

```
returns
```


```
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]])
```


```
# Calculate F(-2)(p)
Y_weights = (1/assets)*(np.ones((assets,1)))
Y_returns = np.sort(((returns.T)@Y_weights).flatten())
F_2_inverse_y = []
P_y = []
for eta in Y_returns:
    events = Y_returns<=eta
```

```python
    prob = np.count_nonzero(events)/len(events)
    F_2_inverse_y_temp = prob*(1/np.count_nonzero(events))*(np.sum(Y_returns[events]))                  # Take care in case of non uniform probability
    F_2_inverse_y.append(F_2_inverse_y_temp)
    P_y.append(prob)



F_inverse_y_dict = dict(zip(P_y,F_2_inverse_y))


k=0
events_0  = Y_returns<=Y_returns[-1]
prob_0 = np.count_nonzero(events_0)/len(events_0)
event_cuts = []

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 event in event_cuts:
        prob = np.count_nonzero(event)/len(event)
        g_x_events = returns.T[event,:]@(weights)
        constraints.append(((1/(np.count_nonzero(event)))*cp.sum(g_x_events)) >= ((1/(prob))*F_inverse_y_dict[prob]))

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

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

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

    # Calcualte t
    delta_t = []
    for t in Z_k:
        B_events = Z_k <= t
        prob = np.count_nonzero(B_events)/len(B_events)
        F_k_inv = F_inverse_y_dict[prob]
        E_Z_k_cond = (1/(np.count_nonzero(B_events)))*np.sum(Z_k[B_events])
        delta_t_temp =  ((1/(prob))*F_k_inv) - E_Z_k_cond
        delta_t.append(delta_t_temp)

    delta_max = np.max(delta_t)
    t_max = Z_k[np.argmax(delta_t)]

    # print(delta_max)

    if delta_max <= 0:
        print("Problem",problem.value)
        print("weights",weights.value)
        print("Conditions satisfied")
        break
    else:
        events_b = Z_k<=t_max
        print(f"violated events ",np.argwhere(events_b).T)
        prob = np.count_nonzero(events_b)/len(events_b)
        event_cuts.append(events_b)


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

```
violated events  [[5]]


 Iteration no.  1
 violated events  [[6]]


 Iteration no.  2
 violated events  [[1 3 5 6 9]]


 Iteration no.  3
 violated events  [[3]]


 Iteration no.  4
 violated events  [[1 3 6]]
```

```
Iteration no.  5
violated events  [[3 6]]


Iteration no.  6
violated events  [[1 3 5 6]]


Iteration no.  7
violated events  [[1 2 3 4 5 6 9]]


Iteration no.  8
Problem 0.013845283868824157
weights [[5.43608832e-09]
 [3.66588426e-02]
 [9.45000135e-09]
 [7.21658932e-02]
 [3.09857154e-08]
 [1.89657760e-01]
 [3.42977925e-09]
 [1.63699706e-01]
 [2.84291324e-01]
 [2.53526425e-01]]
Conditions satisfied
```

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

```
Problem 0.013845283868824157
weights [[5.43608832e-09]
 [3.66588426e-02]
 [9.45000135e-09]
 [7.21658932e-02]
 [3.09857154e-08]
 [1.89657760e-01]
 [3.42977925e-09]
 [1.63699706e-01]
 [2.84291324e-01]
 [2.53526425e-01]]
```

Start coding or generate with AI.