

```
!pip install PySDDP==0.0.11 # Instala o Pacote PySDDP
```

```
Collecting PySDDP==0.0.10
  Downloading https://files.pythonhosted.org/packages/f7/8e/61679a672fdf0f39f8af1f83c
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (from
Collecting pyswarm
  Downloading https://files.pythonhosted.org/packages/79/1e/254c108b5e65c65d57a83a9a
Requirement already satisfied: typing in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: cvxopt in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from cy
Building wheels for collected packages: pyswarm
  Building wheel for pyswarm (setup.py) ... done
  Created wheel for pyswarm: filename=pyswarm-0.6-cp36-none-any.whl size=4481 sha256=
  Stored in directory: /root/.cache/pip/wheels/37/c5/f6/b33b9ac00040cb95c1f00af982a41
Successfully built pyswarm
Installing collected packages: pyswarm, PySDDP
Successfully installed PySDDP-0.0.10 pyswarm-0.6
```

## ▼ Plota Árvore de Decisão

```
from PySDDP import PowerSystem
from copy import deepcopy
from graphviz import Digraph
import numpy as np

def plot_tree(Caso, Cenario, Estagios, Aberturas, Nome_Arq):
    """Gera um pdf com estágios e aberturas.

    Keyword arguments:
    Cenario -- Especifique o cenario de afluencia a ser utilizado no primeiro estagio
    Estagios -- Especifique a profundidade da árvore, ou seja, quantas camadas terá a árvo
    Aberturas -- Especifique o número aberturas para cada nó
    Nome_Arq -- Especifique o Nome do arquivo de saída
    """

    if Aberturas > Caso.sistema["DGer"]["Nr_Cen"]:
        print ("Número de Cenários Menor que Número de Aberturas")
        print ("Método plot_tree interrompido!")
        return

    if Cenario > Caso.sistema["DGer"]["Nr_Cen"]:
        print ("Cenario escolhido superior ao Número de Cenários disponíveis")
        print ("Método plot_tree interrompido!")
```

```

    return

if Estagios > Caso.sistema["DGer"]["Nr_Est"]:
    print ("Número de Estágios Superior aos dados informados no problema.")
    print("Método plot_tree interrompido!")
    return

def cria_tree(lista, CasoEstudo, Aberturas, Estagios):

    estagio = lista[-1]["Estagio"] + 1
    anterior = lista[-1]["Ordem"]
    if estagio == Estagios:
        return
    for icen in range(Aberturas):
        elemento = { "Anterior": anterior,
                     "Estagio": estagio,
                     "Afluencia": CasoEstudo.sistema["UHE"][0]["Afl"][estagio][icen],
                     "Ordem": len(lista),
                     }
        lista.append(elemento)
        cria_tree(lista, CasoEstudo, Aberturas, Estagios)

estagio = 0
elemento = {
    "Anterior" : None,
    "Estagio" : 0,
    "Afluencia" : Caso.sistema["UHE"][0]["Afl"][estagio][0],
    "Ordem" : 0,
}

lista = []
lista.append(elemento)
cria_tree(lista, Caso, Aberturas, Estagios)

g = Digraph('G', filename=Nome_Arq)
for elemento in lista:
    if (elemento["Anterior"] != None):
        probabilidade = 100/(Aberturas**elemento["Estagio"])
        g.edge(str(elemento["Anterior"]), str(elemento["Ordem"]),
               label=str(round(probabilidade,2))+'%')

for elemento in lista:
    g.node(str(elemento["Ordem"]),label= " Afl: "+ str(elemento["Afluencia"]))

g.node("0", style="filled", fillcolor="green")

for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
    if (iest%2) == 0:
        Cor = 'red'
        Preenche = 'green:cyan'
    else:
        Cor = 'red'
        Preenche = 'lightblue:cyan'

    with g.subgraph(name='cluster'+str(iest+1)) as c:

```

```

c.attr(fillcolor=Preenche,
       label='Estágio '+str(iest+1),
       fontcolor=Cor,
       style='filled',
       gradientangle='270')
c.attr('node',
       shape='box',
       fillcolor='red:yellow',
       style='filled',
       gradientangle='90')
for elemento in lista:
    if elemento["Estagio"]==iest:
        c.node(str(elemento["Ordem"]))

g.attr("graph",pad="0.5", nodesep="1", ranksep="2")

g.view()

```

## ▼ PL Único (Equivalente Determinístico do Problema Estocástico)

```

import os
from cvxopt.modeling import variable, solvers, op, matrix
from PySDDP import PowerSystem
import time

def pl_unico_tree(Caso, Cenario, imprime = False):

    solvers.options['show_progress'] = True
    solvers.options['glpk'] = dict(msg_lev='GLP_MSG_OFF')

    #
    # Cria Variáveis de Decisão
    # vf, vt, vv, gt ==> [iest][abertura][0][usina]
    # deficit ==> [iest][abertura]
    #

    vf = []
    vt = []
    vv = []
    gt = []
    deficit = []

    for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
        vf.append([])
        vt.append([])
        vv.append([])
        gt.append([])
        deficit.append([])
        for iabertura in range(Caso.sistema["DGer"]["Nr_Cen"] ** iest):

```

```

vf[iest].append([])
vt[iest].append([])
vv[iest].append([])
gt[iest].append([])
deficit[iest].append([])
vf[iest][iabertura].append(variable(len(Caso.sistema["UHE"]),
                                     "Volume Final no Estagio " +
                                     str(iest) +
                                     " Abertura " +
                                     str(iabertura)))
vt[iest][iabertura].append(variable(len(Caso.sistema["UHE"]),
                                     "Volume Turbinado no Estagio " +
                                     str(iest) +
                                     " Abertura " +
                                     str(iabertura)))
vv[iest][iabertura].append(variable(len(Caso.sistema["UHE"]),
                                     "Volume Vertido no Estagio " +
                                     str(iest) +
                                     " Abertura " +
                                     str(iabertura)))
gt[iest][iabertura].append(variable(len(Caso.sistema["UTE"]),
                                     "Geração Térmica no Estagio " +
                                     str(iest) +
                                     " Abertura " +
                                     str(iabertura)))
deficit[iest][iabertura].append(variable(1,
                                          "Déficit no Estagio " +
                                          str(iest) +
                                          " Abertura " +
                                          str(iabertura)))

#
# Define Função Objetivo (fob)
#

fob = 0
for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
    for iabertura in range(Caso.sistema["DGer"]["Nr_Cen"] ** iest):
        constante = 1 / (Caso.sistema["DGer"]["Nr_Cen"] ** iest)
        for iusi, termica in enumerate(Caso.sistema["UTE"]):
            fob += float(constante) * termica["Custo"] * gt[iest][iabertura][0][iusi]
        fob += float(constante) * Caso.sistema["DGer"]["CDef"] * deficit[iest][iabertu
        for iusi, hidreletrica in enumerate(Caso.sistema["UHE"]):
            fob += float(constante) * 0.001 * vv[iest][iabertura][0][iusi]

# Constrói Conjunto de Restrições

restricoes = []

#
# Balanço Hídrico
# Para cada nó existe uma restrição de Balanço Hídrico por Usina
#

for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
    abertura_anterior = 0

```

```

contador = 0
for iabertura in range(Caso.sistema["DGer"]["Nr_Cen"] ** iest):
    if contador == Caso.sistema["DGer"]["Nr_Cen"]:
        contador = 0
        abertura_anterior += 1
    for iusi, uhe in enumerate(Caso.sistema["UHE"]):
        if iest == 0:
            restricoes.append(vf[iest][iabertura][0][iusi] ==
                              float(Caso.sistema["UHE"][iusi]["VI"]) +
                              float(Caso.sistema["UHE"][iusi]["Afl"][iest][Cenario
vt[iest][iabertura][0][iusi] -
vv[iest][iabertura][0][iusi]))
        else:
            restricoes.append(vf[iest][iabertura][0][iusi] ==
                              vf[iest - 1][abertura_anterior][0][iusi] +
                              float(Caso.sistema["UHE"][iusi]["Afl"][iest][contado
vt[iest][iabertura][0][iusi] -
vv[iest][iabertura][0][iusi]))

    contador += 1

#
# Atendimento à Demanda
# Para cada nó existe uma Restrição de Atendimento à Demanda
#

for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
    for iabertura in range(Caso.sistema["DGer"]["Nr_Cen"] ** iest):
        demanda = 0
        for iusi, uhe in enumerate(Caso.sistema["UHE"]):
            demanda += float(Caso.sistema["UHE"][iusi]["Prod"]) * vt[iest][iabertura][
        for iusi, ute in enumerate(Caso.sistema["UTE"]):
            demanda += gt[iest][iabertura][0][iusi]
        demanda += deficit[iest][iabertura][0]
        restricoes.append(demanda == Caso.sistema["DGer"]["Carga"][iest])

#
# Restricoes de Canalização
#

for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
    for iabertura in range(Caso.sistema["DGer"]["Nr_Cen"] ** iest):
        for iusi, uhe in enumerate(Caso.sistema["UHE"]):
            restricoes.append(vf[iest][iabertura][0][iusi] >= Caso.sistema["UHE"][iusi
            restricoes.append(vf[iest][iabertura][0][iusi] <= Caso.sistema["UHE"][iusi
            restricoes.append(vt[iest][iabertura][0][iusi] >= 0)
            restricoes.append(vt[iest][iabertura][0][iusi] <= Caso.sistema["UHE"][iusi
            restricoes.append(vv[iest][iabertura][0][iusi] >= 0)
        for iusi, ute in enumerate(Caso.sistema["UTE"]):
            restricoes.append(gt[iest][iabertura][0][iusi] >= 0)
            restricoes.append(gt[iest][iabertura][0][iusi] <= Caso.sistema["UTE"][iusi
            restricoes.append(deficit[iest][iabertura][0] >= 0)

#
# Computa o instante de tempo no qual o processo iterativo iniciou

```

```

#
t = time.time()

#
# Cria problema de otimização
#

problema = op(fob, restricoes)

#
# Chama solver GLPK e resolve o problema de otimização linear
#

problema.solve('dense', 'glpk')

#
# Calcula o tempo decorrido desde o início do algoritmo
#
print("Tempo decorrido no PL Único - Árvore Completa", time.time() - t)

#
# Armazena Resultado
#

ResNos = []
Contador = 0
Pula = len(Caso.sistema["UHE"])*(Caso.sistema["DGer"]["Nr_Cen"]**Caso.sistema["DGer"]
Pula = int(Pula)
for iest in range(Caso.sistema["DGer"]["Nr_Est"]):
    for iabertura in range(Caso.sistema["DGer"]["Nr_Cen"] ** iest):
        lista_uhe = []
        CustoTotal = 0.
        for i, iusi in enumerate(Caso.sistema["UHE"]):
            elemento = {
                "vf": vf[iest][iabertura][0][i].value()[0],
                "vt": vt[iest][iabertura][0][i].value()[0],
                "vv": vv[iest][iabertura][0][i].value()[0],
                "cma": restricoes[Contador].multiplier.value[0]
            }
            CustoTotal += 0.01 * vv[iest][iabertura][0][i].value()[0]
            lista_uhe.append(elemento)
            Contador += 1
        lista_ute = []
        for i, iusi in enumerate(Caso.sistema["UTE"]):
            elemento = {
                "gt": gt[iest][iabertura][0][i].value()[0]
            }
            CustoTotal += iusi["Custo"] * gt[iest][iabertura][0][i].value()[0]
            lista_ute.append(elemento)
        CustoTotal += Caso.sistema["DGer"]["CDef"] * deficit[iest][iabertura][0].value
        DGer = {
            "Deficit": deficit[iest][iabertura][0].value[0],
            "CMO": restricoes[Pula].multiplier.value[0],
            "CustoImediato": CustoTotal
        }

```

```

        Pula += 1
        No = { "Dger": DGer,
                "UHE": lista_uhe,
                "UTE": lista_ute,
                }
        NoCompleto = {
                        "Estagio": iest,
                        "Ordem": iabertura,
                        "Resultado": No
                    }
        ResNos.append(NoCompleto)

    return(ResNos, fob.value()[0])

```

```

def despacho_pddd_est(Caso, VI, AFL, pote_de_corte, iest, ordem, imprime):

    Num_UHE = len(Caso.sistema["UHE"])

    Num_UTe = len(Caso.sistema["UTE"])

    #
    # Cria Variáveis de Decisão
    #

    vf = variable(Num_UHE, "Volume Final na Usina")
    vt = variable(Num_UHE, "Volume Turbinado na Usina")
    vv = variable(Num_UHE, "Volume Vertido na Usina")
    gt = variable(Num_UTe, "Geração na Usina Térmica")
    deficit = variable(1, "Déficit de Energia no Sistema")
    alpha = variable(1, "Custo Futuro")

    # Construção da Função Objetivo

    fob = 0

    for i, iusi in enumerate(Caso.sistema["UTE"]):
        fob += iusi['Custo'] * gt[i]

    fob += Caso.sistema["DGer"]["CDef"] * deficit[0]

    for i, iusi in enumerate(Caso.sistema["UHE"]):
        fob += 0.01 * vv[i]

    fob += 1.0 * alpha[0]

    # Definição das Restrições

    restricoes = []

    # Balanço Hídrico

    for i, iusi in enumerate(Caso.sistema["UHE"]):
        restricoes.append(vf[i] == float(VI[i]) + float(AFL[i]) - vt[i] - vv[i])

```

```

# Atendimento à Demanda

AD = 0

for i, iusi in enumerate(Caso.sistema["UHE"]):
    AD += iusi["Prod"] * vt[i]

for i, usi in enumerate(Caso.sistema["UTE"]):
    AD += gt[i]

AD += deficit[0]

restricoes.append(AD == Caso.sistema["DGer"]["Carga"][iest - 1])

# Restricoes Canalização

for i, iusi in enumerate(Caso.sistema["UHE"]):
    restricoes.append(vf[i] >= iusi["Vmin"])
    restricoes.append(vf[i] <= iusi["Vmax"])
    restricoes.append(vt[i] >= 0)
    restricoes.append(vt[i] <= iusi["Engol"])
    restricoes.append(vv[i] >= 0)

for i, iusi in enumerate(Caso.sistema["UTE"]):
    restricoes.append(gt[i] >= 0)
    restricoes.append(gt[i] <= iusi["Capac"])

restricoes.append(deficit[0] >= 0)

restricoes.append(alpha[0] >= 0)

#
# Insere inequações correspondentes aos cortes
#

for icorte in pote_de_corte:
    if icorte['Estagio'] == iest and icorte["Ordem"] == ordem:
        equacao = 0
        for iusi in range(Num_UHE):
            equacao += float(icorte['Coefs'][iusi]) * vf[iusi]
        equacao += float(icorte['Termo_Indep'])
        restricoes.append(alpha[0] >= equacao)

#
# Cria problema de otimização
#

problema = op(fob, restricoes)

#
# Chama solver GLPK e resolve o problema de otimização linear
#

problema.solve('dense', 'glpk')

```



```

#
# Armazena resultados do problema em um dicionário de dados
#

Dger = {
    "Deficit": deficit[0].value()[0],
    "CMO": restricoes[Num_UHE].multiplier.value[0],
    "CustoTotal": fob.value()[0],
    "CustoFuturo": alpha[0].value()[0]
}

lista_uhe = []
for i, iusi in enumerate(Caso.sistema["UHE"]):
    resultado = {
        "vf": vf[i].value()[0],
        "vt": vt[i].value()[0],
        "vv": vv[i].value()[0],
        "cma": restricoes[i].multiplier.value[0]
    }
    lista_uhe.append(resultado)

lista_ute = []
for i, iusi in enumerate(Caso.sistema["UTE"]):
    resultado = {
        "gt": gt[i].value()[0]
    }
    lista_ute.append(resultado)

resultado = {
    "DGer": Dger,
    "UHE": lista_uhe,
    "UTE": lista_ute
}

#
# Imprime resultados em tela
#

if imprime:
    print("Custo Total:", fob.value())

    for i, usi in enumerate(Caso.sistema["UHE"]):
        print(vf.name, i, "é", vf[i].value(), "hm3")
        print(vt.name, i, "é", vt[i].value(), "hm3")
        print(vv.name, i, "é", vv[i].value(), "hm3")

    for i, usi in enumerate(Caso.sistema["UTE"]):
        print(gt.name, i, "é", gt[i].value(), "MWmed")

    print(deficit.name, "é", deficit[0].value(), "MWmed")

    print(alpha.name, "é", alpha[0].value(), "$")

    for i, iusi in enumerate(Caso.sistema["UHE"]):

```

```

        print("O valor da água na usina", i, "é: ", restricoes[i].multiplier.value)

        print("O Custo Marginal de Operação é: ", restricoes[Num_UHE].multiplier.value)

        print("----- x ----- ")

#
# Retorna da função exportando os resultados
#

return resultado

```

## ▼ PDDD (Equivalente Determinístico do Problema Estocástico)

```

from PySDDP import PowerSystem
import numpy as np
from copy import deepcopy

def pddd_tree(Caso, Cenario, imprime = False):

    def recursiva(Caso, VI, AFL, Estagio, Ordem, PoteDeCorte, ResNos, imprime = False):
        Ordem_Deste_No = Ordem
        #
        # Somente no Estágio Inicial Existe um Nó Somente
        #
        if Estagio == 0:
            Fator = float(1)
        #
        # Em todos os outros nós existirão Tantas bifurcações quanto o número de cenários
        #
        else:
            Fator = float(Caso.sistema["DGer"]["Nr_Cen"])
        #
        # Despacha Nó - Forward
        #
        resultado_fwd = despacho_pddd_est(Caso, VI, AFL, PoteDeCorte, Estagio+1, Ordem_Deste_No)
        CI = (resultado_fwd["DGer"]["CustoTotal"] - resultado_fwd["DGer"]["CustoFuturo"])
        CT = resultado_fwd["DGer"]["CustoTotal"]
        Guarda = {
            "Estagio": Estagio,
            "Ordem": Ordem_Deste_No,
            "Resultado": resultado_fwd
        }
        ResNos.append(Guarda)
        #
        # Caso o nó resolvido seja de último estágio não há necessidade de bifurcar mais
        #
        if Estagio == Caso.sistema["DGer"]["Nr_Est"]-1:
            term_indep = resultado_fwd["DGer"]["CustoTotal"]
            coefs = []

```

```

for i, iusi in enumerate(resultado_fwd["UHE"]):
    coefs.append(-iusi["cma"])
    term_indep -= VI[i] * coefs[i]
Corte = {
    "Estagio": Estagio,
    "Ordem": Ordem,
    "Termo_Indep": term_indep/Fator,
    "Coefs": [ coef/Fator for coef in coefs ]
}
return (CT, CI/Fator, Corte, Ordem, ResNos)
#
# Caso o nó resolvido não seja de último estágio deve-se bifurcar o número de cenári
#
VF = []
for iusi in resultado_fwd["UHE"]:
    VF.append(iusi["vf"])
#
# Cria Corte Médio (Inicializa com Zero)
#
Corte = {
    "Estagio": Estagio+1,
    "Ordem": Ordem_Deste_No,
    "Termo_Indep": 0.,
    "Coefs": [0.] * len(Caso.sistema["UHE"])
}
for icen in range(Caso.sistema["DGer"]["Nr_Cen"]):
    AFL_CEN = []
    for iusi in Caso.sistema["UHE"]:
        AFL_CEN.append(iusi["Afl"][Estagio+1][icen])
    Ordem += 1
    [CTMedio, CIMedio, CorteMedio, Ordem, ResNos] = recursiva(Caso, VF, AFL_CEN, Est
    Corte["Termo_Indep"] += CorteMedio["Termo_Indep"]
    for i in range(len(CorteMedio["Coefs"])):
        Corte["Coefs"][i] += CorteMedio["Coefs"][i]
    CI += CIMedio
    PoteDeCorte.append(Corte)
    resultado_bkd = despacho_pddd_est(Caso, VI, AFL, PoteDeCorte, Estagio+1, Ordem_Deste
    term_indep = resultado_bkd["DGer"]["CustoTotal"]
    coefs = []
    for i, iusi in enumerate(resultado_bkd["UHE"]):
        coefs.append(-iusi["cma"])
        term_indep -= VI[i] * coefs[i]
    Corte = {
        "Estagio": Estagio,
        "Ordem": Ordem,
        "Termo_Indep": term_indep/Fator,
        "Coefs": [ coef/Fator for coef in coefs ]
    }
}

return (CT, CI/Fator, Corte, Ordem, ResNos)

tol = 0.01
iteracao = 1
Estagio = 0
VI = []

```

```

AFL = []
PoteDeCorte = []
for iusi in Caso.sistema["UHE"]:
    VI.append(iusi["VI"])
    AFL.append(iusi["Afl"][Estagio][Cenario])

#
# Computa o instante de tempo no qual o processo iterativo iniciou
#
t = time.time()

zinf_lista = []
zsup_lista = []
while True:
    Estagio = 0
    Ordem = 0
    ResNos = []
    [ ZINF, ZSUP, Corte, Ordem, ResNos] = recursiva(Caso, VI, AFL, Estagio, Ordem, PoteD
    zinf_lista.append(ZINF)
    zsup_lista.append(ZSUP)

    if np.abs(ZSUP - ZINF) < tol:
        break
    else:
        iteracao += 1

#
# Calcula o tempo decorrido desde o início do algoritmo
#
print("Tempo decorrido na PDDD - Árvore Completa", time.time() - t)

return(ResNos, ZINF, zinf_lista, zsup_lista)

```

```
import plotly.graph_objects as go
```

```
CasoEstudo = PowerSystem.Classroom()
```

```

CasoEstudo.sistema["UHE"][0]["Afl"] = [[25, 20, 23],
                                         [30, 22, 27],
                                         [29, 18, 24],
                                         [27, 20, 23],
                                         [22, 14, 18],
                                         [18, 13, 15],
                                         [18, 12, 15],
                                         [15, 9, 12],
                                         [12, 8, 10],
                                         [14, 12, 13],
                                         [19, 17, 16],
                                         [22, 18, 20]]

```

```

CasoEstudo.sistema["DGer"]["Nr_Est"] = 7
CasoEstudo.sistema["DGer"]["Nr_Cen"] = 2
CasoEstudo.sistema["DGer"]["Carga"] = [ 50., 60, 45,

```

```
50., 60, 45,  
50., 60, 45,  
50., 60, 45 ]
```

```
plot_tree(CasoEstudo, 0,3,3,"/Users/andremercato/Downloads/arvore")
```

```
[ res_plu, fob_plu ] = pl_unico_tree(CasoEstudo, 0,imprime=False)  
print(round(fob_plu,2))
```

```
[ res_pddd, fob_pddd, zinf, zsup ] = pddd_tree(CasoEstudo, 0, imprime=False)  
print(round(fob_pddd,2))  
print(zinf)  
print(zsup)
```

```
print(res_pddd[0])  
print(res_plu[0])
```

```
fig = None
```

```
fig = go.Figure()
```

```
for iest in range(7):  
    Nome = "Estagio " + str(iest+1)  
    VetorX = []  
    VetorY = []  
    for no in res_pddd:  
        if no["Estagio"] == iest:  
            VetorX.append(Nome)  
            VetorY.append(no["Resultado"]["UHE"][0]["vf"])  
    #fig.add_trace(go.Violin(x=VetorX,  
    #                        y=VetorY,  
    #                        name=Nome,  
    #                        box_visible=True,  
    #                        meanline_visible=True))  
  
    fig.add_trace(go.Box(x=VetorX, y=VetorY, name=Nome, boxmean=True))  
fig.show()
```

```
Número de Cenários Menor que Número de Aberturas
Método plot_tree interrompido!
Tempo decorrido no PL Único - Árvore Completa 0.3612532615661621
5434.45
Tempo decorrido na PDDD - Árvore Completa 2.876964569091797
5434.45
[-7.105427357601002e-17, 4337.5, 4337.499999999998, 5109.374999999999, 5432.597656
[21330.000000000007, 10554.062500000005, 6511.054997519842, 5567.492919921882, 545
{'Estagio': 0, 'Ordem': 0, 'Resultado': {'Dger': {'Deficit': 0.0, 'CMO': -314.4531
{'Estagio': 0, 'Ordem': 0, 'Resultado': {'Dger': {'Deficit': 0.0, 'CMO': -314.4531
```

