**CR6 Results Analysis A**

**Python Imports**

**import** numpy **as** np

**import** pandas **as** pd

**from** prettypandas **import** PrettyPandas

**import** patsy

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** statsmodels.api **as** sm

**import** statsmodels.formula.api

**from** pyomo.environ **import** \*

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.preprocessing **import** PolynomialFeatures

**from** IPython.display **import** display, Markdown, HTML

%matplotlib inline

PlotWidth = 6

**import** warnings

warnings.filterwarnings('ignore')

*# helper functions for this notebook*

*# use SVG for matplotlib-based figures*

%matplotlib inline

%config InlineBackend.figure\_format = 'svg'

**def** **coded\_to\_actual**(coded\_data, actual\_lows, actual\_highs):

"""Converts a pandas DataFrame from coded units to actuals."""

actual\_data = coded\_data.copy()

**for** col **in** actual\_data.columns:

**if** **not** (col **in** actual\_highs **and** col **in** actual\_lows):

**continue**

**try**:

*# convert continuous variables to their actual value*

actual\_data[col] \*= 0.5 \* (float(actual\_highs[col]) - float(actual\_lows[col]))

*# don't need to cast to float here, if either are not a float exception will have been thrown*

actual\_data[col] += 0.5 \* (actual\_highs[col] + actual\_lows[col])

**except** ValueError:

*# assume 2 level categorical*

actual\_data[col] = actual\_data[col].map({-1: actual\_lows[col], 1: actual\_highs[col]})

**return** actual\_data

**def** **get\_tick\_labels**(key, lows, highs, units):

"""Returns a list of low/high labels with units (e.g. [8mm, 10mm])"""

**return** [str(lows[key]) + units[key], str(highs[key]) + units[key]]

**def** **backward\_regression**(X, y,

threshold\_out,

verbose=True):

included=list(X.columns)

**while** **True**:

changed=**False**

model = sm.OLS(y, sm.add\_constant(pd.DataFrame(X[included]))).fit()

*# use all coefs except intercept*

pvalues = model.pvalues.iloc[1:]

worst\_pval = pvalues.max() *# null if pvalues is empty*

**if** worst\_pval > threshold\_out:

changed=**True**

worst\_feature = pvalues.idxmax()

included.remove(worst\_feature)

**if** verbose:

print('Drop {:30} with p-value {:.6}'.format(worst\_feature, worst\_pval))

**if** **not** changed:

**break**

**return** included

**def** **build\_model**(X,values,verbose=True):

X = [sub.replace('alh', 'model.X1') **for** sub **in** X]

X = [sub.replace('aps', 'model.X2') **for** sub **in** X]

X = [sub.replace('aid', 'model.X3') **for** sub **in** X]

X = [sub.replace('arw', 'model.X4') **for** sub **in** X]

X = [sub.replace('awt', 'model.X5') **for** sub **in** X]

X = [sub.replace(':', '\*') **for** sub **in** X]

model = str(values[0])

i=1

**for** v **in** X:

model += " + " + str(values[i]) + " \* " + v

i += 1

**if** verbose:

print(model)

**return** model

**Process CSV Files**

*# importing the pandas library*

**import** pandas **as** pd

*# reading the csv file using read\_csv*

*# storing the data frame in variable called df*

df\_cost = pd.read\_csv('https://raw.githubusercontent.com/wilsongis/3DP\_Experiments/main/Data/cr6\_cost\_raw.txt', sep='\t')

df\_time = pd.read\_csv('https://raw.githubusercontent.com/wilsongis/3DP\_Experiments/main/Data/cr6\_time\_raw.txt', sep='\t')

*# creating a list of column names by*

*# calling the .columns*

list\_of\_columns\_cost = list(df\_cost.columns)

list\_of\_columns\_time = list(df\_time.columns)

*# displaying the list of column names*

print('List of Cost column names : ',

list\_of\_columns\_cost)

print('List of Time column names : ',

list\_of\_columns\_time)

List of Cost column names : ['trial', 'lh', 'ps', 'id', 'rw', 'wt', 'alh', 'aps', 'aid', 'arw', 'awt', 'rep', 'cost']

List of Time column names : ['trial', 'lh', 'ps', 'id', 'rw', 'wt', 'alh', 'aps', 'aid', 'arw', 'awt', 'rep', 'time']

display((Markdown("### Statistics for Cost")))

df\_cost.cost.describe()

**Statistics for Cost**

**count** 80.000000

**mean** 0.530875

**std** 0.019500

**min** 0.480000

25% 0.510000

50% 0.540000

75% 0.540000

**max** 0.560000

**Name**: **cost**, **dtype**: **float64**

plt.figure(figsize=(PlotWidth, PlotWidth))

sns.boxplot(data=df\_cost['cost'])

plt.title('Box Plot of Cost')

plt.ylabel('Cost')

plt.show()

plt.figure(figsize=(PlotWidth, PlotWidth))

df\_cost['cost'].hist()

plt.title('cost')

plt.xlabel('cost)')

plt.show()

plt.figure(figsize=(PlotWidth\*2, PlotWidth))

plt.scatter(df\_cost['trial'], df\_cost['cost'])

plt.title('cost vs. trial')

plt.xlabel('trial')

plt.ylabel('cost')

plt.show()

display((Markdown("### Statistics for Time")))

df\_time.time.describe()

**Statistics for Time**

**count** 80.00000

**mean** 12681.40000

**std** 3360.13591

**min** 8480.00000

25% 9464.25000

50% 12989.00000

75% 15460.75000

**max** 18098.00000

**Name**: **time**, **dtype**: **float64**

plt.figure(figsize=(PlotWidth, PlotWidth))  
sns.boxplot(data=df\_time[‘time’])  
plt.title(‘Box Plot of Time’)  
plt.ylabel(‘Time’)  
plt.show()

plt.figure(figsize=(PlotWidth, PlotWidth))  
df\_time[‘time’].hist()  
plt.title(‘time’)  
plt.xlabel(‘time)’)  
plt.show()

plt.figure(figsize=(PlotWidth\*2, PlotWidth))  
plt.scatter(df\_time[‘trial’], df\_time[‘time’])  
plt.title(‘time vs. trial’)  
plt.xlabel(‘trial’)  
plt.ylabel(‘time’)  
plt.show()

**Cost Analysis**

f = 'cost ~ (alh+aps+aid+arw+awt)\*\*2'

y, X = patsy.dmatrices(f, df\_cost, return\_type='dataframe')

print(y[:5])

print(X[:5])

**cost**

0 0.51

1 0.51

2 0.54

3 0.51

4 0.54

**Intercept** **alh** **aps** **aid** **arw** **awt** **alh**:aps **alh**:aid **alh**:arw **alh**:awt \

0 1.0 0.16 50.0 0.25 0.4 0.8 8.0 0.040 0.064 0.128

1 1.0 0.28 50.0 0.25 0.4 1.2 14.0 0.070 0.112 0.336

2 1.0 0.16 60.0 0.25 0.4 1.2 9.6 0.040 0.064 0.192

3 1.0 0.28 60.0 0.25 0.4 0.8 16.8 0.070 0.112 0.224

4 1.0 0.16 50.0 0.15 0.4 1.2 8.0 0.024 0.064 0.192

**aps**:aid **aps**:arw **aps**:awt **aid**:arw **aid**:awt **arw**:awt

0 12.5 20.0 40.0 0.10 0.20 0.32

1 12.5 20.0 60.0 0.10 0.30 0.48

2 15.0 24.0 72.0 0.10 0.30 0.48

3 15.0 24.0 48.0 0.10 0.20 0.32

4 7.5 20.0 60.0 0.06 0.18 0.48

*## An intercept is not added by default, so we need to add that here*

X = sm.add\_constant(X)

results = sm.OLS(y, X).fit()

results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))

sns.regplot(x=results.predict(X), y=y)

plt.xlabel('Predicted cost')

plt.ylabel('Actual cost')

plt.title('Actual vs. Predicted cost')

plt.show()

OLS Regression Results

==============================================================================

Dep. Variable: cost R-squared: 0.971

Model: OLS Adj. R-squared: 0.964

Method: Least Squares F-statistic: 141.4

Date: Wed, 28 Jul 2021 Prob (F-statistic): 4.75e-43

**Time:** 19:27:10 Log-Likelihood: 343.19

No. Observations: 80 AIC: -654.4

Df Residuals: 64 BIC: -616.3

Df Model: 15

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

Intercept 0.3328 0.040 8.322 0.000 0.253 0.413

alh 0.0083 0.091 0.092 0.927 -0.173 0.189

aps -0.0019 0.001 -2.880 0.005 -0.003 -0.001

aid -0.1825 0.108 -1.691 0.096 -0.398 0.033

arw 0.3513 0.028 12.757 0.000 0.296 0.406

awt 0.2413 0.026 9.188 0.000 0.189 0.294

alh:aps 0.0029 0.001 2.111 0.039 0.000 0.006

alh:aid -0.6250 0.138 -4.523 0.000 -0.901 -0.349

alh:arw 0.1563 0.035 4.523 0.000 0.087 0.225

alh:awt -0.1563 0.035 -4.523 0.000 -0.225 -0.087

aps:aid 0.0105 0.002 6.332 0.000 0.007 0.014

aps:arw -0.0026 0.000 -6.332 0.000 -0.003 -0.002

aps:awt 0.0011 0.000 2.714 0.009 0.000 0.002

aid:arw 0.4125 0.041 9.950 0.000 0.330 0.495

aid:awt -0.4625 0.041 -11.156 0.000 -0.545 -0.380

arw:awt -0.2594 0.010 -25.025 0.000 -0.280 -0.239

==============================================================================

Omnibus: 114.302 Durbin-Watson: 2.141

Prob(Omnibus): 0.000 Jarque-Bera (JB): 3820.537

Skew: -4.440 Prob(JB): 0.00

Kurtosis: 35.669 Cond. No. 3.13e+04

==============================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.13e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

**Reduced Cost Model**

cost\_included = backward\_regression(X,y,.05)

cost\_included.pop(0)

print(cost\_included)

**Drop** alh **with** p-value 0.927017

**Drop** aid **with** p-value 0.0905131

['aps', 'arw', 'awt', 'alh:aps', 'alh:aid', 'alh:arw', 'alh:awt', 'aps:aid', 'aps:arw', 'aps:awt', 'aid:arw', 'aid:awt', 'arw:awt']

y = df\_cost['cost']

*#y = df\_cost['time']*

X = X[cost\_included]

*## An intercept is not added by default, so we need to add that here*

X = sm.add\_constant(X)

results = sm.OLS(y, X).fit()

results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))

sns.regplot(x=results.predict(X), y=y)

plt.xlabel('Predicted cost')

plt.ylabel('Actual cost')

plt.title('Actual vs. Predicted cost')

plt.show()

OLS Regression Results

==============================================================================

Dep. Variable: cost R-squared: 0.969

Model: OLS Adj. R-squared: 0.963

Method: Least Squares F-statistic: 160.7

Date: Wed, 28 Jul 2021 Prob (F-statistic): 1.22e-44

**Time:** 19:27:12 Log-Likelihood: 341.41

No. Observations: 80 AIC: -654.8

Df Residuals: 66 BIC: -621.5

Df Model: 13

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

const 0.3006 0.029 10.531 0.000 0.244 0.358

aps -0.0015 0.001 -2.669 0.010 -0.003 -0.000

arw 0.3541 0.028 12.824 0.000 0.299 0.409

awt 0.2460 0.026 9.423 0.000 0.194 0.298

alh:aps 0.0032 0.001 4.253 0.000 0.002 0.005

alh:aid -0.6815 0.128 -5.340 0.000 -0.936 -0.427

alh:arw 0.1581 0.034 4.671 0.000 0.091 0.226

alh:awt -0.1531 0.032 -4.764 0.000 -0.217 -0.089

aps:aid 0.0081 0.001 9.160 0.000 0.006 0.010

aps:arw -0.0026 0.000 -6.288 0.000 -0.003 -0.002

aps:awt 0.0011 0.000 2.695 0.009 0.000 0.002

aid:arw 0.3961 0.041 9.754 0.000 0.315 0.477

aid:awt -0.4898 0.039 -12.715 0.000 -0.567 -0.413

arw:awt -0.2594 0.010 -24.854 0.000 -0.280 -0.239

==============================================================================

Omnibus: 125.804 Durbin-Watson: 2.098

Prob(Omnibus): 0.000 Jarque-Bera (JB): 4980.386

Skew: -5.178 Prob(JB): 0.00

Kurtosis: 40.241 Cond. No. 2.66e+04

==============================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 2.66e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

cost\_eq = build\_model(cost\_included,results.params,**False**)

print("Cost = " + cost\_eq)

Cost = 0.30061301677651686 + -0.0014979526099359666 \* **model**.X2 + 0.3541048685686145 \* **model**.X4 + 0.24600811428103486 \* **model**.X5 + 0.003193676289093103 \* **model**.X1\***model**.X2 + -0.681545835829839 \* **model**.X1\***model**.X3 + 0.15813870197108404 \* **model**.X1\***model**.X4 + -0.15310216338152616 \* **model**.X1\***model**.X5 + 0.008101719131677326 \* **model**.X2\***model**.X3 + -0.0026250000000000544 \* **model**.X2\***model**.X4 + 0.0011250000000000322 \* **model**.X2\***model**.X5 + 0.3961480849887108 \* **model**.X3\***model**.X4 + -0.4897531916854896 \* **model**.X3\***model**.X5 + -0.2593750000000006 \* **model**.X4\***model**.X5

**Time Analysis**

f = 'time ~ (alh+aps+aid+arw+awt)\*\*2'

y, X = patsy.dmatrices(f, df\_time, return\_type='dataframe')

print(y[:5])

print(X[:5])

**time**

0 18098.0

1 8741.0

2 14493.0

3 10191.0

4 14914.0

**Intercept** **alh** **aps** **aid** **arw** **awt** **alh**:aps **alh**:aid **alh**:arw **alh**:awt \

0 1.0 0.16 50.0 0.25 0.4 0.8 8.0 0.040 0.064 0.128

1 1.0 0.28 50.0 0.25 0.4 1.2 14.0 0.070 0.112 0.336

2 1.0 0.16 60.0 0.25 0.4 1.2 9.6 0.040 0.064 0.192

3 1.0 0.28 60.0 0.25 0.4 0.8 16.8 0.070 0.112 0.224

4 1.0 0.16 50.0 0.15 0.4 1.2 8.0 0.024 0.064 0.192

**aps**:aid **aps**:arw **aps**:awt **aid**:arw **aid**:awt **arw**:awt

0 12.5 20.0 40.0 0.10 0.20 0.32

1 12.5 20.0 60.0 0.10 0.30 0.48

2 15.0 24.0 72.0 0.10 0.30 0.48

3 15.0 24.0 48.0 0.10 0.20 0.32

4 7.5 20.0 60.0 0.06 0.18 0.48

*## An intercept is not added by default, so we need to add that here*

X = sm.add\_constant(X)

results = sm.OLS(y, X).fit()

results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))

sns.regplot(x=results.predict(X), y=y)

plt.xlabel('Predicted Time')

plt.ylabel('Actual Time')

plt.title('Actual vs. Predicted Time')

plt.show()

OLS Regression Results

==============================================================================

Dep. Variable: time R-squared: 0.985

Model: OLS Adj. R-squared: 0.982

Method: Least Squares F-statistic: 287.3

Date: Wed, 28 Jul 2021 Prob (F-statistic): 1.17e-52

**Time:** 19:27:13 Log-Likelihood: -593.61

No. Observations: 80 AIC: 1219.

Df Residuals: 64 BIC: 1257.

Df Model: 15

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

Intercept 3.886e+04 4869.801 7.980 0.000 2.91e+04 4.86e+04

alh -9.373e+04 1.1e+04 -8.494 0.000 -1.16e+05 -7.17e+04

aps -101.9583 81.061 -1.258 0.213 -263.896 59.980

aid 6178.6667 1.31e+04 0.470 0.640 -2.01e+04 3.24e+04

arw 2096.5000 3353.230 0.625 0.534 -4602.347 8795.347

awt -8507.9167 3197.582 -2.661 0.010 -1.49e+04 -2120.012

alh:aps 302.8333 168.294 1.799 0.077 -33.372 639.039

alh:aid -4.994e+04 1.68e+04 -2.968 0.004 -8.36e+04 -1.63e+04

alh:arw 1.009e+04 4207.345 2.399 0.019 1688.613 1.85e+04

alh:awt 2.909e+04 4207.345 6.913 0.000 2.07e+04 3.75e+04

aps:aid 301.9000 201.953 1.495 0.140 -101.547 705.347

aps:arw -137.1250 50.488 -2.716 0.008 -237.987 -36.263

aps:awt -18.6750 50.488 -0.370 0.713 -119.537 82.187

aid:arw 5830.0000 5048.814 1.155 0.252 -4256.165 1.59e+04

aid:awt -1.288e+04 5048.814 -2.552 0.013 -2.3e+04 -2798.835

arw:awt 1626.2500 1262.203 1.288 0.202 -895.291 4147.791

==============================================================================

Omnibus: 117.964 Durbin-Watson: 1.973

Prob(Omnibus): 0.000 Jarque-Bera (JB): 3766.256

Skew: 4.734 Prob(JB): 0.00

Kurtosis: 35.252 Cond. No. 3.13e+04

==============================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.13e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

**Time Reduced Model**

time\_included = backward\_regression(X,y,.05)

time\_included.pop(0)

print(time\_included)

**Drop** aps:awt **with** p-**value** 0.712687

**Drop** aid **with** p-**value** 0.63764

**Drop** arw **with** p-**value** 0.549529

**Drop** aid:arw **with** p-**value** 0.12172

**Drop** alh:aps **with** p-**value** 0.0744513

**Drop** arw:awt **with** p-**value** 0.0725949

['alh', 'aps', 'awt', 'alh:aid', 'alh:arw', 'alh:awt', 'aps:aid', 'aps:arw', 'aid:awt']

y = df\_time['time']

X = X[time\_included]

*## An intercept is not added by default, so we need to add that here*

X = sm.add\_constant(X)

results = sm.OLS(y, X).fit()

results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))

sns.regplot(x=results.predict(X), y=y)

plt.xlabel('Predicted Time')

plt.ylabel('Actual Time')

plt.title('Actual vs. Predicted Time')

plt.show()

OLS Regression Results

==============================================================================

Dep. Variable: time R-squared: 0.983

Model: OLS Adj. R-squared: 0.981

Method: Least Squares F-statistic: 453.8

Date: Wed, 28 Jul 2021 Prob (F-statistic): 2.19e-58

**Time:** 19:27:14 Log-Likelihood: -599.26

No. Observations: 80 AIC: 1219.

Df Residuals: 70 BIC: 1242.

Df Model: 9

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

const 3.942e+04 1154.479 34.142 0.000 3.71e+04 4.17e+04

alh -7.914e+04 6029.887 -13.124 0.000 -9.12e+04 -6.71e+04

aps -128.1914 25.154 -5.096 0.000 -178.360 -78.023

awt -8860.8498 1369.537 -6.470 0.000 -1.16e+04 -6129.395

alh:aid -4.626e+04 1.65e+04 -2.798 0.007 -7.92e+04 -1.33e+04

alh:arw 1.23e+04 4097.512 3.002 0.004 4127.272 2.05e+04

alh:awt 2.909e+04 4317.386 6.737 0.000 2.05e+04 3.77e+04

aps:aid 434.5856 102.753 4.229 0.000 229.652 639.519

aps:arw -57.7179 16.919 -3.411 0.001 -91.462 -23.974

aid:awt -1.138e+04 4760.148 -2.390 0.020 -2.09e+04 -1883.394

==============================================================================

Omnibus: 103.336 Durbin-Watson: 1.908

Prob(Omnibus): 0.000 Jarque-Bera (JB): 2137.913

Skew: 3.999 Prob(JB): 0.00

Kurtosis: 27.030 Cond. No. 2.14e+04

==============================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 2.14e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

time\_eq = build\_model(time\_included,results.params,**False**)

print("Time = " + time\_eq)

Time = 39415.87499999855 + -79138.51046393832 \* **model**.X1 + -128.19137670175314 \* **model**.X2 + -8860.849838411767 \* **model**.X5 + -46255.95567867106 \* **model**.X1\***model**.X3 + 12299.502666120085 \* **model**.X1\***model**.X4 + 29085.416666666522 \* **model**.X1\***model**.X5 + 434.58559556786804 \* **model**.X2\***model**.X3 + -57.717904019688284 \* **model**.X2\***model**.X4 + -11377.20914127423 \* **model**.X3\***model**.X5

**Equations**

display(Markdown("Cost = "))

print(cost\_eq)

print("------")

display(Markdown("Time = "))

print(time\_eq)

Cost =

0.30061301677651686 + **-0**.0014979526099359666 \* **model**.X2 + 0.3541048685686145 \* **model**.X4 + 0.24600811428103486 \* **model**.X5 + 0.003193676289093103 \* **model**.X1\***model**.X2 + **-0**.681545835829839 \* **model**.X1\***model**.X3 + 0.15813870197108404 \* **model**.X1\***model**.X4 + **-0**.15310216338152616 \* **model**.X1\***model**.X5 + 0.008101719131677326 \* **model**.X2\***model**.X3 + **-0**.0026250000000000544 \* **model**.X2\***model**.X4 + 0.0011250000000000322 \* **model**.X2\***model**.X5 + 0.3961480849887108 \* **model**.X3\***model**.X4 + **-0**.4897531916854896 \* **model**.X3\***model**.X5 + **-0**.2593750000000006 \* **model**.X4\***model**.X5

**------**

Time =

39415.87499999855 + **-79138**.51046393832 \* **model**.X1 + **-128**.19137670175314 \* **model**.X2 + **-8860**.849838411767 \* **model**.X5 + **-46255**.95567867106 \* **model**.X1\***model**.X3 + 12299.502666120085 \* **model**.X1\***model**.X4 + 29085.416666666522 \* **model**.X1\***model**.X5 + 434.58559556786804 \* **model**.X2\***model**.X3 + **-57**.717904019688284 \* **model**.X2\***model**.X4 + **-11377**.20914127423 \* **model**.X3\***model**.X5

**Optimization**

model = ConcreteModel()

model.X1 = Var(within=NonNegativeReals)

model.X2 = Var(within=NonNegativeReals)

model.X3 = Var(within=NonNegativeReals)

model.X4 = Var(within=NonNegativeReals)

model.X5 = Var(within=NonNegativeReals)

model.C1 = Constraint(expr = model.X1 <= .28)

model.C2 = Constraint(expr = model.X2 <= 60)

model.C3 = Constraint(expr = model.X3 <= .25)

model.C4 = Constraint(expr = model.X4 <= .8)

model.C5 = Constraint(expr = model.X5 <= 1.2)

model.C6 = Constraint(expr = model.X1 >= .16)

model.C7 = Constraint(expr = model.X2 >= 50)

model.C8 = Constraint(expr = model.X3 >= .15)

model.C9 = Constraint(expr = model.X4 >= .4)

model.C10 = Constraint(expr = model.X5 >= .8)

model.f1 = Var()

model.f2 = Var()

model.C\_f1 = Constraint(expr = model.f1 == (0.30061301677651686 + -0.0014979526099359666 \* model.X2 + 0.3541048685686145 \* model.X4 + 0.24600811428103486 \* model.X5 + 0.003193676289093103 \* model.X1\*model.X2 + -0.681545835829839 \* model.X1\*model.X3 + 0.15813870197108404 \* model.X1\*model.X4 + -0.15310216338152616 \* model.X1\*model.X5 + 0.008101719131677326 \* model.X2\*model.X3 + -0.0026250000000000544 \* model.X2\*model.X4 + 0.0011250000000000322 \* model.X2\*model.X5 + 0.3961480849887108 \* model.X3\*model.X4 + -0.4897531916854896 \* model.X3\*model.X5 + -0.2593750000000006 \* model.X4\*model.X5))

model.C\_f2 = Constraint(expr = model.f2 == (39415.87499999855 + -79138.51046393832 \* model.X1 + -128.19137670175314 \* model.X2 + -8860.849838411767 \* model.X5 + -46255.95567867106 \* model.X1\*model.X3 + 12299.502666120085 \* model.X1\*model.X4 + 29085.416666666522 \* model.X1\*model.X5 + 434.58559556786804 \* model.X2\*model.X3 + -57.717904019688284 \* model.X2\*model.X4 + -11377.20914127423 \* model.X3\*model.X5))

model.O\_f1 = Objective(expr = model.f1, sense=minimize)

model.O\_f2 = Objective(expr = model.f2, sense=minimize)

*# max f1 separately*

*# install glpk solver: sudo apt-get install glpk-utils*

model.O\_f2.deactivate()

solver = SolverFactory('ipopt') *#'cplex', 'ipopt'*

solver.solve(model)

print('( X1 , X2, X3, X4, X5 ) = ( ' + str(value(model.X1)) + ' , ' + str(value(model.X2)) + ' , ' + str(value(model.X3)) + ' , ' + str(value(model.X4)) + ' , ' + str(value(model.X5)) + ' )')

print('f1 = ' + str(value(model.f1)))

print('f2 = ' + str(value(model.f2)))

f2\_min = value(model.f2)

*# max f2 separately*

model.O\_f2.activate()

model.O\_f1.deactivate()

solver = SolverFactory('ipopt') *#'cplex', 'ipopt'*

solver.solve(model)

print('( X1 , X2, X3, X4, X5 ) = ( ' + str(value(model.X1)) + ' , ' + str(value(model.X2)) + ' , ' + str(value(model.X3)) + ' , ' + str(value(model.X4)) + ' , ' + str(value(model.X5)) + ' )')

print('f1 = ' + str(value(model.f1)))

print('f2 = ' + str(value(model.f2)))

f2\_max = value(model.f2)

*# apply augmented $\epsilon$-Constraint*

*# max f1 + delta\*s*

*# constraint f2 - s = e*

model.O\_f1.activate()

model.O\_f2.deactivate()

model.del\_component(model.O\_f1)

model.del\_component(model.O\_f2)

model.e = Param(initialize=0, mutable=**True**)

model.delta = Param(initialize=0.00001)

model.slack = Var(within=NonNegativeReals)

model.O\_f1 = Objective(expr = model.f1 + model.delta \* model.slack, sense=minimize)

model.C\_e = Constraint(expr = model.f2 - model.slack == model.e)

n = 100

step = int((f2\_max - f2\_min) / n)

steps = list(range(int(f2\_min),int(f2\_max),step)) + [f2\_max]

x1\_l, x2\_l, x3\_l, x4\_l, x5\_l = [], [], [], [], []

f1\_l, f2\_l = [], []

**for** i **in** steps:

model.e = i

solver.solve(model)

x1\_l.append(value(model.X1))

x2\_l.append(value(model.X2))

x3\_l.append(value(model.X3))

x4\_l.append(value(model.X4))

x5\_l.append(value(model.X5))

f1\_l.append(value(model.f1))

f2\_l.append(value(model.f2))

*# print(i, value(model.X1), value(model.X2), value(model.f1), value(model.slack), value(model.f2))*

( X1 , X2, X3, X4, X5 ) = ( 0.27999999699979045 , 50.00000021485858 , 0.24999996237920893 , 0.39999999572941514 , 0.8000000173812946 )

f1 = 0.49701939510293697

f2 = 10416.124260969951

( X1 , X2, X3, X4, X5 ) = ( 0.28000000999993935 , 60.000000599972694 , 0.2500000099939384 , 0.8000000096771722 , 1.2000000119993277 )

f1 = 0.53898523398684

f2 = 8557.622612553769

```python

```