# Friction coefficien data normalizition modeling

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import matplotlib.ticker as ticker  
from sklearn.preprocessing import StandardScaler  
from sklearn.gaussian\_process import GaussianProcessRegressor  
from sklearn.gaussian\_process.kernels import RBF,ConstantKernel as c  
from sklearn.gaussian\_process.kernels import WhiteKernel, ConstantKernel  
from sklearn.model\_selection import ShuffleSplit  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from mpl\_toolkits.mplot3d import Axes3D  
import seaborn as sns  
from matplotlib.colors import LogNorm  
from matplotlib import cm  
from matplotlib.pyplot import MultipleLocator  
import xlwt  
import warnings

sns.set()  
warnings.filterwarnings('ignore')

plt.rcParams['font.sans-serif'] = 'SimSun'  
plt.rcParams['axes.unicode\_minus'] = False

font1 = {  
 'family': 'SimSun',  
 'weight': 'bold',  
 'size': 18  
}  
font2 = {  
 'family': 'Times New Roman',  
 'weight': 'normal',  
 'size': 15  
}  
font3 = {  
 'family': 'Times New Roman',  
 'weight': 'normal',  
 'size': 10  
}

## Load data

filepath = r'C:\Users\Administrator\Desktop\experimental\_data\30%-friction coefficient.xlsx'# dataset address  
df = pd.read\_excel(filepath, skiprows=0)  
df.head()

power

feed rate

friction

0

700

1.0

0.6200

1

800

1.0

0.6315

2

900

1.0

0.6163

3

1000

1.0

0.6004

4

1100

1.0

0.5800

# remove missing values  
df.dropna(axis=0,inplace=True)  
df.shape  
df

power

feed rate

friction

0

700

1.00

0.6200

1

800

1.00

0.6315

2

900

1.00

0.6163

3

1000

1.00

0.6004

4

1100

1.00

0.5800

5

1200

1.00

0.6048

6

700

0.85

0.6442

7

800

0.85

0.6277

8

900

0.85

0.5909

9

1000

0.85

0.6417

10

1100

0.85

0.5850

11

1200

0.85

0.6134

12

700

0.70

0.6788

13

800

0.70

0.6312

14

900

0.70

0.6201

15

1000

0.70

0.6305

16

1100

0.70

0.5628

17

1200

0.70

0.5590

18

700

0.55

0.6181

19

800

0.55

0.6083

20

900

0.55

0.6050

21

1000

0.55

0.5933

22

1100

0.55

0.5857

23

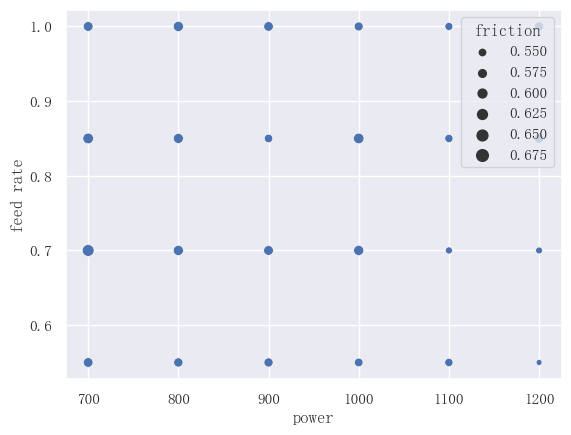
1200

0.55

0.5432

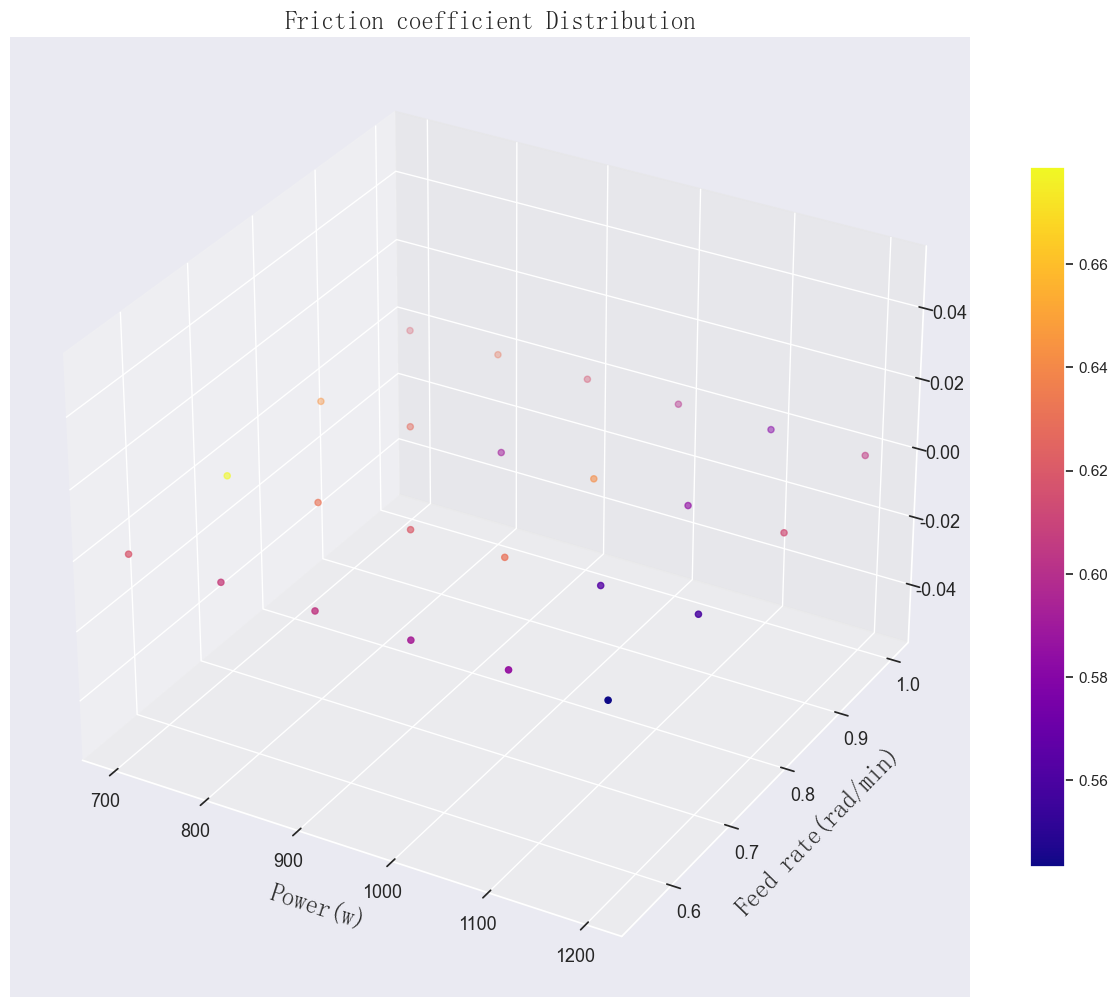
# examine friction coefficient distribution   
sns.scatterplot(x='power', y='feed rate', size='friction',\  
 data=df)

<Axes: xlabel='power', ylabel='feed rate'>



png

sns.set\_style('darkgrid')  
fig = plt.figure(figsize=(12, 10))  
ax = Axes3D(fig)  
fig.add\_axes(ax)  
ax\_obj = ax.scatter(df['power'], df['feed rate'],c=df['friction'],\  
 cmap=cm.plasma)  
plt.colorbar(ax\_obj, shrink=0.7)  
ax.set\_title('Friction coefficient Distribution', fontdict=font1)  
ax.set\_xlabel('\nPower(w)', fontdict=font1)  
ax.set\_ylabel('\nFeed rate(rad/min)', fontdict=font1)  
plt.tick\_params(labelsize=13)  
plt.show()



png

data = df.pivot(index='power', columns='feed rate', values='friction').T  
data.sort\_index(ascending=False, inplace=True)  
data

power

700

800

900

1000

1100

1200

feed rate

1.00

0.6200

0.6315

0.6163

0.6004

0.5800

0.6048

0.85

0.6442

0.6277

0.5909

0.6417

0.5850

0.6134

0.70

0.6788

0.6312

0.6201

0.6305

0.5628

0.5590

0.55

0.6181

0.6083

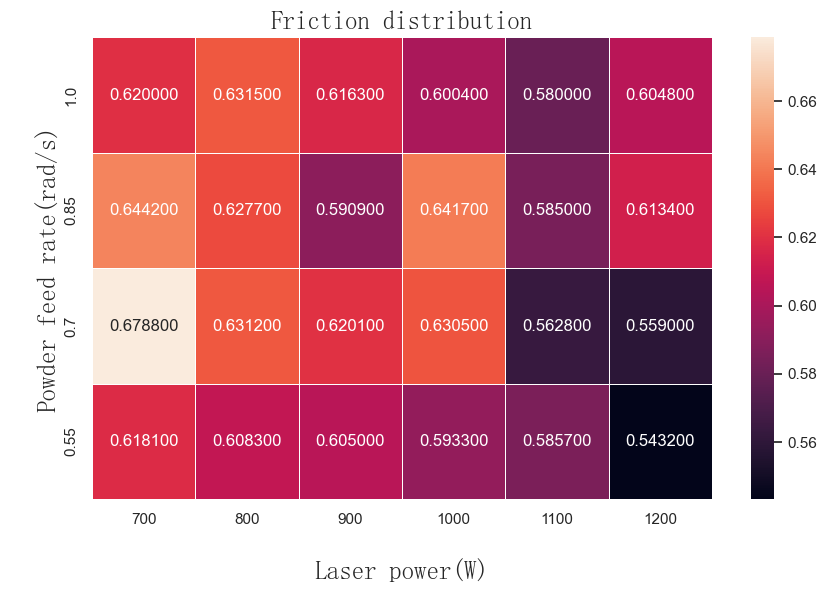
0.6050

0.5933

0.5857

0.5432

# plot the color map  
fig= plt.subplots(figsize=(10,6))  
ax\_obj1 = sns.heatmap(data, annot=True, fmt="f", linewidths=.6)  
plt.title('Friction distribution', fontdict=font1)  
plt.xlabel('\nLaser power(W)', fontdict=font1)  
plt.ylabel('\nPowder feed rate(rad/s)', fontdict=font1)  
a, b = data.shape  
plt.show()



png

## Data normalization, 0 mean, 1 variance

scaler\_X = StandardScaler().fit(df[['power', 'feed rate']])  
scaler\_X.mean\_  
new\_X = scaler\_X.transform(df[['power', 'feed rate']])

scaler\_Y = StandardScaler().fit(df[['friction']])  
scaler\_Y.mean\_  
scaler\_Y.scale\_  
new\_Y = scaler\_Y.transform(df[['friction']])

# standardized data table  
new\_df = pd.DataFrame()  
new\_df['power'] = new\_X[:, 0].ravel()  
new\_df['feed rate'] = new\_X[:, 1].ravel()  
new\_df['friction'] = new\_Y.ravel()

new\_df.head()

power

feed rate

friction

0

-1.46385

1.341641

0.406076

1

-0.87831

1.341641

0.795097

2

-0.29277

1.341641

0.280913

3

0.29277

1.341641

-0.256951

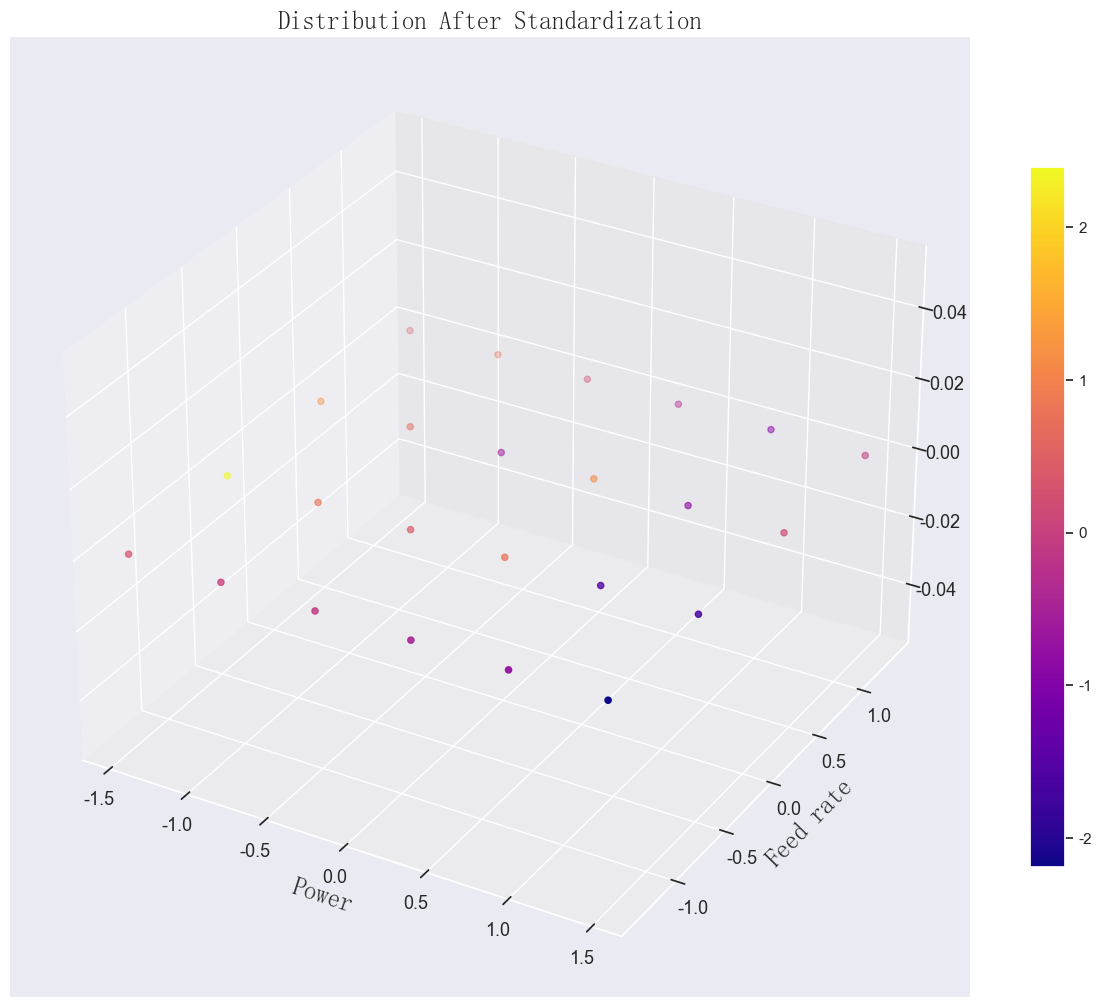
4

0.87831

1.341641

-0.947040

# check the distribution of the standardized data  
sns.set\_style('darkgrid')  
fig = plt.figure(figsize=(12, 10))  
ax = Axes3D(fig)  
fig.add\_axes(ax)  
ax\_obj = ax.scatter(new\_df['power'], new\_df['feed rate'],c=new\_df['friction'],\  
 cmap=cm.plasma)  
plt.colorbar(ax\_obj, shrink=0.7)  
ax.set\_title('Distribution After Standardization', fontdict=font1)  
ax.set\_xlabel('Power', fontdict=font1)  
ax.set\_ylabel('Feed rate', fontdict=font1)  
plt.tick\_params(labelsize=13)  
plt.show()



png

# Cross-validation model evaluation

## approximately 10-fold

new\_df.shape

(24, 3)

shuffle = ShuffleSplit(n\_splits=10, test\_size=10, random\_state=42)  
index = np.arange(24)  
for train\_index, test\_index in shuffle.split(index):  
 print(train\_index)  
 print(test\_index)

[ 2 12 15 3 4 22 17 20 23 7 10 14 19 6]  
[ 8 16 0 18 11 9 13 1 21 5]  
[ 8 3 6 22 2 13 17 14 15 9 16 23 21 11]  
[ 5 18 1 7 0 10 12 19 4 20]  
[19 3 0 2 7 12 11 6 14 1 20 8 17 13]  
[23 15 4 21 9 22 5 10 16 18]  
[23 8 22 6 18 7 14 15 13 3 9 1 11 17]  
[ 5 2 10 19 0 12 20 16 21 4]  
[15 18 2 10 23 16 7 11 21 8 6 0 12 14]  
[ 3 13 5 20 1 4 19 17 9 22]  
[17 10 5 21 20 2 4 15 0 1 11 7 8 6]  
[13 19 3 9 22 12 23 16 14 18]  
[11 3 0 21 18 2 12 9 19 14 8 6 13 22]  
[ 5 1 16 10 7 15 17 20 23 4]  
[23 19 10 13 3 16 11 5 12 9 1 21 18 6]  
[14 4 20 2 7 15 0 17 22 8]  
[17 9 20 0 4 13 21 7 5 6 2 8 12 19]  
[22 14 23 1 11 16 3 15 10 18]  
[ 0 20 6 5 11 17 23 18 19 21 4 16 2 1]  
[ 9 8 7 12 3 14 15 13 22 10]

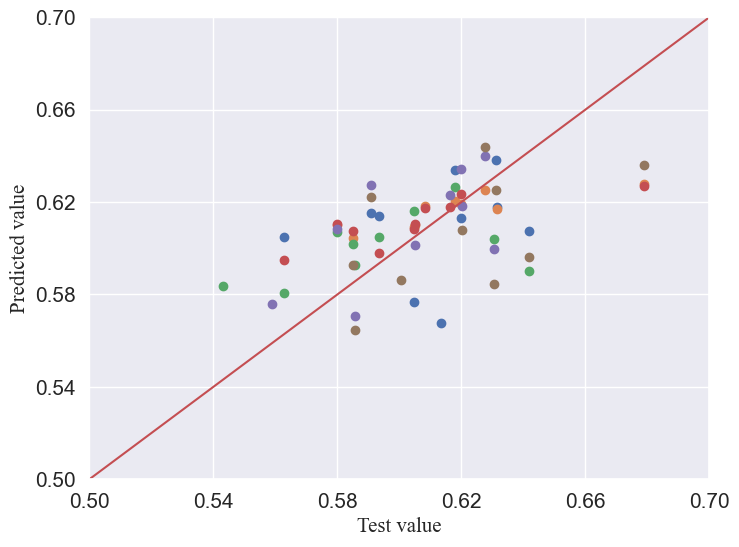
x2 = [0.5, 0.54, 0.58, 0.62, 0.66, 0.7]  
sns.set\_style('darkgrid')

def cv\_loop2(reg):  
 mses = []  
 maes = []  
 y\_test\_res = []  
 y\_predict\_res = []  
 maes\_relative = []  
 i = 1  
 indx = [1,2,3, 4, 8, 10]  
 plt.figure(figsize=(8, 6))  
 for train\_index, test\_index in shuffle.split(index):  
 # extract the training and test datasets  
 x\_train = new\_df[['power', 'feed rate']].iloc[train\_index]  
 y\_train = new\_df['friction'].iloc[train\_index]  
 x\_test = new\_df[['power', 'feed rate']].iloc[test\_index]  
 y\_test = new\_df['friction'].iloc[test\_index]  
 # model prediction and training  
 reg.fit(x\_train, y\_train)  
 print(reg.log\_marginal\_likelihood())  
 y\_predict = reg.predict(x\_test).reshape(-1,1)  
 y\_test\_arr = np.array(y\_test.tolist())  
 y\_test\_arr = y\_test\_arr.reshape(-1,1)  
 y\_test\_inverse = scaler\_Y.inverse\_transform(y\_test\_arr)  
 y\_predict\_inverse = scaler\_Y.inverse\_transform(y\_predict)  
 if i in indx:  
 plt.scatter(y\_test\_inverse, y\_predict\_inverse)  
 y\_test\_res.append(y\_test\_inverse)  
 y\_predict\_res.append(y\_predict\_inverse)  
 mse = mean\_squared\_error(y\_test\_inverse, y\_predict\_inverse, squared=False)  
 mses.append(mse)  
 mae = mean\_absolute\_error(y\_test\_inverse, y\_predict\_inverse)  
 maes.append(mae)  
 maes.append(mae)  
 maes\_relative.append(mae/np.mean(y\_test\_inverse))  
 i += 1  
 plt.plot(x2, x2, 'r-')  
 plt.xlim(0.5, 0.7)  
 plt.ylim(0.5, 0.7)  
 plt.xticks(x2)  
 plt.yticks(x2)  
 plt.tick\_params(labelsize=15)  
 plt.xlabel('Test value', fontdict=font2)  
 plt.ylabel('Predicted value', fontdict=font2)  
 plt.show()  
 return mses, maes, maes\_relative, y\_test\_res, y\_predict\_res

kernel\_1 = 1.0 \* RBF(length\_scale=1,\  
 length\_scale\_bounds=(1e-3, 1e3)) +\  
WhiteKernel(noise\_level=1e-4, noise\_level\_bounds=(1e-10, 1e+2))  
reg\_1 = GaussianProcessRegressor(kernel=kernel\_1,\  
 n\_restarts\_optimizer=20,\  
 alpha=0, random\_state=42)

mses2, maes2, maes\_relative, y\_test\_res, y\_predict\_res = cv\_loop2(reg\_1)  
mses2

-17.354545998419084  
-19.090632559738403  
-16.630513425402164  
-18.50601037931942  
-19.840376660686147  
-15.498906265103779  
-17.001476449070683  
-19.38757890264254  
-18.88130649000571  
-12.95110844810442



png

[0.027175344674240366,  
 0.020632481292177724,  
 0.02586067575025751,  
 0.02310308652681755,  
 0.017362005300649838,  
 0.034477063061697,  
 0.030962911885091753,  
 0.020059619753770744,  
 0.020884420508134095,  
 0.028598368044668185]

#output RMSE, MAE, RMD  
np.mean(mses2), np.mean(maes2), np.mean(maes\_relative)

(0.02491159767975048, 0.019782432686140206, 0.032679887703016684)

# Gaussian Process Regression

## noisy kernel function, Gaussian kernel with White noise kernel

kernel\_1 = 1.0 \* RBF(length\_scale=50.0,\  
 length\_scale\_bounds=(1, 1e2))+ WhiteKernel(noise\_level=1e-1, noise\_level\_bounds=(1e-3, 1e-1))  
reg\_1 = GaussianProcessRegressor(kernel=kernel\_1,\  
 n\_restarts\_optimizer=10,\  
 alpha=0.1, random\_state=42)  
reg\_1.fit(new\_df[['power', 'feed rate']], new\_df['friction'])

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

GaussianProcessRegressor

reg\_1.kernel

1\*\*2 \* RBF(length\_scale=50) + WhiteKernel(noise\_level=0.1)

## Export hyperparameters

reg\_1.get\_params(),reg\_1.log\_marginal\_likelihood()

({'alpha': 0.1,  
 'copy\_X\_train': True,  
 'kernel\_\_k1': 1\*\*2 \* RBF(length\_scale=50),  
 'kernel\_\_k2': WhiteKernel(noise\_level=0.1),  
 'kernel\_\_k1\_\_k1': 1\*\*2,  
 'kernel\_\_k1\_\_k2': RBF(length\_scale=50),  
 'kernel\_\_k1\_\_k1\_\_constant\_value': 1.0,  
 'kernel\_\_k1\_\_k1\_\_constant\_value\_bounds': (1e-05, 100000.0),  
 'kernel\_\_k1\_\_k2\_\_length\_scale': 50.0,  
 'kernel\_\_k1\_\_k2\_\_length\_scale\_bounds': (1, 100.0),  
 'kernel\_\_k2\_\_noise\_level': 0.1,  
 'kernel\_\_k2\_\_noise\_level\_bounds': (0.001, 0.1),  
 'kernel': 1\*\*2 \* RBF(length\_scale=50) + WhiteKernel(noise\_level=0.1),  
 'n\_restarts\_optimizer': 10,  
 'n\_targets': None,  
 'normalize\_y': False,  
 'optimizer': 'fmin\_l\_bfgs\_b',  
 'random\_state': 42},  
 -32.73118684086089)

# create a grid for plotting test dat  
p\_min, p\_max = new\_df['power'].min(),\  
new\_df['power'].max()  
s\_min, s\_max = new\_df['feed rate'].min(),\  
new\_df['feed rate'].max()  
print(p\_max, p\_min)  
print(s\_max, s\_min)  
p\_set1, s\_set1 = np.meshgrid(np.arange(p\_min, p\_max + 0.3, 0.01),\  
 np.arange(s\_min, s\_max + 0.3, 0.01))

1.4638501094227996 -1.4638501094227996  
1.3416407864998738 -1.3416407864998738

p\_set1.shape

(299, 323)

output1, std = reg\_1.predict(np.c\_[p\_set1.ravel(), s\_set1.ravel()]\  
 , return\_std=True)  
output1, std = output1.reshape(p\_set1.shape), std.reshape(p\_set1.shape)

# Upper and lower confidence intervals

up, down = output1 \* (1 + 1.96\*std), output1 \* (1 - 1.96\*std)

output1

array([[ 0.50236006, 0.49623104, 0.49005578, ..., -1.86383876,  
 -1.86298419, -1.86188353],  
 [ 0.51811274, 0.51195099, 0.50574162, ..., -1.86960029,  
 -1.86872765, -1.86760791],  
 [ 0.53392203, 0.52772763, 0.52148422, ..., -1.87517721,  
 -1.87428649, -1.87314765],  
 ...,  
 [ 0.47997118, 0.48203047, 0.4840355 , ..., -0.15367611,  
 -0.14645806, -0.13926091],  
 [ 0.47773166, 0.47983515, 0.48188468, ..., -0.15704468,  
 -0.14983232, -0.14264068],  
 [ 0.47551302, 0.47765977, 0.47975285, ..., -0.16038489,  
 -0.15317901, -0.14599368]])

# lower bound of the Ggaussian confidence interval  
ucb\_1 = output1 + 0.1\*std  
ucb\_2 = output1 + 1\*std  
ucb\_3 = output1 + 100\*std

output1\_data = pd.DataFrame(columns=['power', 'feed rate', 'friction'])  
output1\_data['power'] = p\_set1.ravel()  
output1\_data['feed rate'] = s\_set1.ravel()  
output1\_data['friction'] = output1.ravel()  
output1\_data.shape

(96577, 3)

# denormalization of predicted data  
scale\_X = scaler\_X.inverse\_transform(output1\_data[['power', 'feed rate']])  
scale\_Y = scaler\_Y.inverse\_transform(output1\_data[['friction']])

# variance denormalization  
scale\_std = scaler\_Y.inverse\_transform(std)

new\_output = pd.DataFrame(columns=['power', 'feed rate', 'friction'])  
new\_output['power'] = scale\_X[:, 0].ravel()  
new\_output['feed rate'] = scale\_X[:, 1].ravel()  
new\_output['friction'] = scale\_Y.ravel()

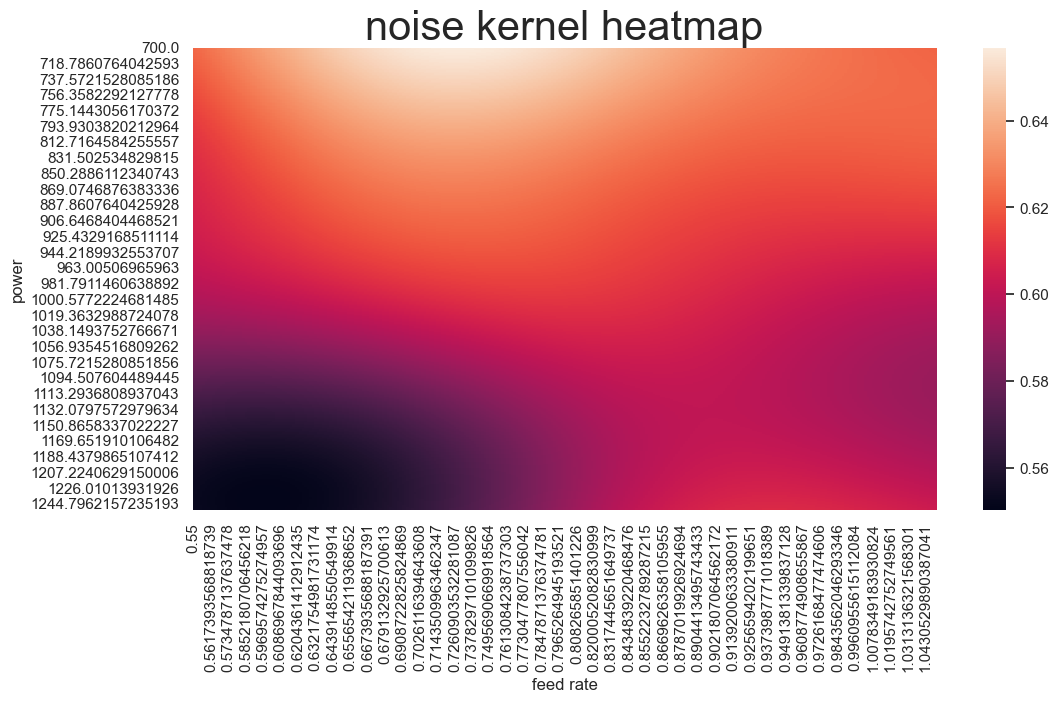
p\_inverse = scale\_X[:, 0].reshape(p\_set1.shape)  
s\_inverse = scale\_X[:, 1].reshape(s\_set1.shape)  
out\_inverse = scale\_Y.reshape(p\_set1.shape)

p\_inverse.max()  
s\_inverse.max()

1.0497611929712034

output\_pivot = new\_output.pivot(index='power', columns='feed rate', values='friction')  
f, ax = plt.subplots(figsize=(12,6))  
sns.heatmap(output\_pivot, ax=ax)  
ax.set\_title('noise kernel heatmap', fontsize=30)

Text(0.5, 1.0, 'noise kernel heatmap')

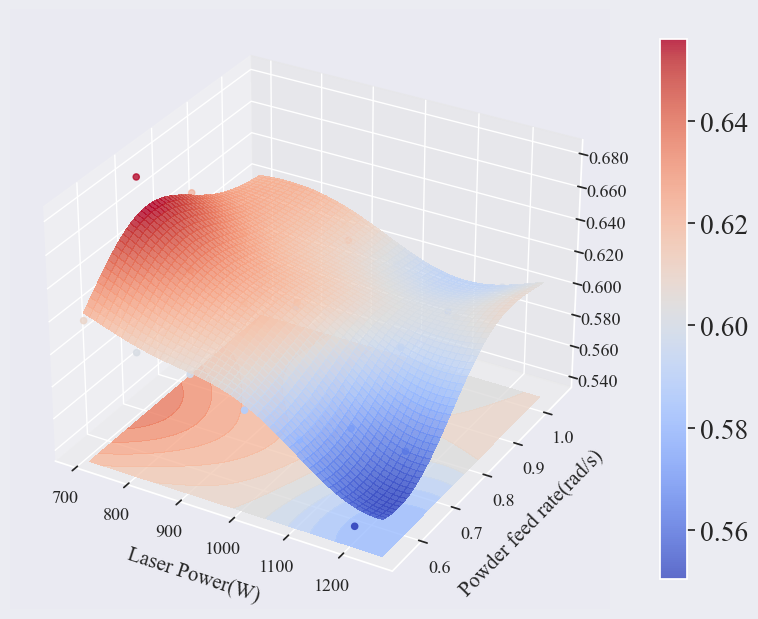


png

df.iloc[:,2].max(),df.iloc[:,2].min()

(0.6788, 0.5432)

# plot surface, contour, and combined scatter plot  
sns.set(  
 style='darkgrid',  
 font='Times New Roman',  
 font\_scale=1.8  
)  
fig = plt.figure(figsize=(10, 6), facecolor='#ebecf2')  
ax = Axes3D(fig)  
fig.add\_axes(ax)  
surface = ax.plot\_surface(p\_inverse, s\_inverse, out\_inverse,\  
 cmap=cm.coolwarm, linewidth=0,\  
 alpha=0.8, antialiased=False)  
ax.scatter(df['power'], df['feed rate'], df['friction'],\  
 c=df['friction'], cmap=cm.coolwarm)  
ax.contourf(p\_inverse, s\_inverse, out\_inverse, np.arange(0.5, 0.7, 0.01),\  
 zdir='output1', offset=0.53, cmap=cm.coolwarm, alpha=0.8)  
ax.set\_xlabel('\nLaser Power(W)', fontdict=font2)  
ax.set\_ylabel('\nPowder feed rate(rad/s)', fontdict=font2)  
ax.zaxis.set\_major\_formatter(ticker.FormatStrFormatter('%.3f'))  
plt.colorbar(surface, shrink=0.9)  
plt.tick\_params(labelsize=13)  
plt.show()

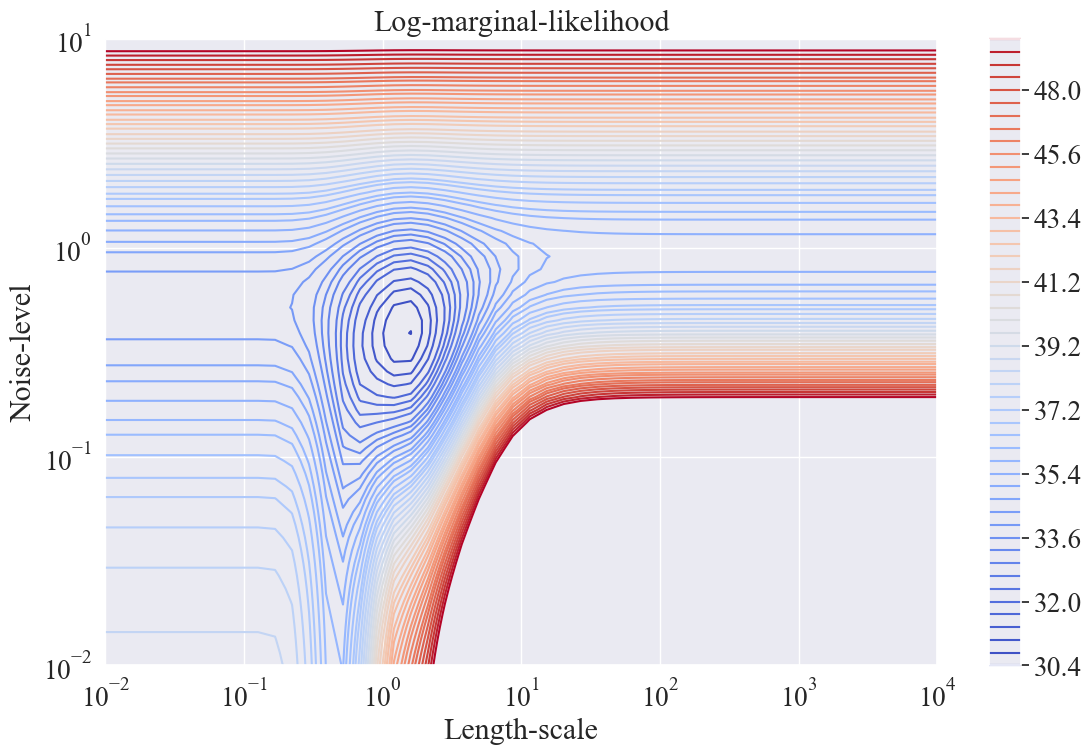


png

## Marginal likelihood visualization

plt.figure(figsize=(12, 8))  
theta0 = np.logspace(-2, 4, 50)  
theta1 = np.logspace(-2, 1, 50)  
Theta0, Theta1 = np.meshgrid(theta0, theta1)  
LML = [reg\_1.log\_marginal\_likelihood(np.log([0.36, t0, t1]))  
 for t0, t1 in zip(Theta0.ravel(), Theta1.ravel())]  
LML = np.reshape(LML, newshape=Theta0.shape)  
vmin, vmax = (-LML).min(), 50  
print(vmax)  
print(vmin)  
level = np.around(np.logspace(np.log10(vmin), np.log10(vmax), num=50), decimals=1)  
plt.contour(Theta0, Theta1, -LML, level, cmap=cm.coolwarm)  
plt.colorbar()  
plt.xscale("log")  
plt.yscale("log")  
plt.xlabel("Length-scale")  
plt.ylabel("Noise-level")  
plt.title("Log-marginal-likelihood")  
plt.tight\_layout()  
plt.show()

50  
30.39445349060818



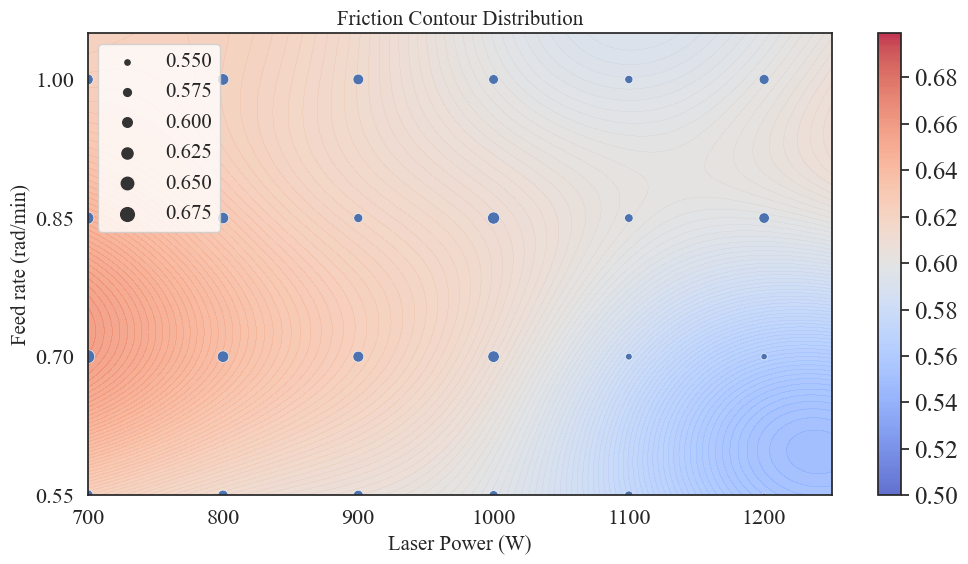
png

# surface visualization  
x = p\_inverse.ravel()  
y = s\_inverse.ravel()  
z = out\_inverse.ravel()  
point\_obj = zip(x, y, z)  
data = []  
for point in point\_obj:  
 data.append(point)  
len(data)

96577

sns.set\_style('white')  
font1 = {  
 'family': 'Times New Roman',  
 'weight': 'bold',  
 'size': 30  
}  
font2 = {  
 'family': 'Times New Roman',  
 'weight': 'normal',  
 'size': 15  
}  
font3 = {  
 'family': 'Times New Roman',  
 'weight': 'normal',  
 'size': 30  
}

# plot contour map  
fig, ax = plt.subplots(figsize=(12,6))  
plt.contourf(p\_inverse, s\_inverse,\  
 out\_inverse, np.arange(0.5, 0.7, 0.001),\  
 offset=0, cmap=cm.coolwarm,\  
 alpha=0.8)  
sns.scatterplot(x='power', y='feed rate', size='friction',\  
 data=df, sizes=(10, 100))  
plt.legend(loc='upper left', prop=font2)  
plt.tick\_params(labelsize=24)  
labels = ax.get\_xticklabels() + ax.get\_yticklabels()  
[label.set\_fontname('Times New Roman') for label in labels]  
cb = plt.colorbar()  
cb\_labels = cb.ax.yaxis.get\_ticklabels()  
for cb\_label in cb\_labels:  
 cb\_label.set\_family('Times New Roman')  
 cb\_label.set\_fontsize(18)  
x\_tick = [700 + i\*100 for i in range(6)]  
y\_tick = [0.55 + j\*0.15 for j in range(6)]  
plt.xticks(x\_tick, fontsize=16)  
plt.yticks(y\_tick, fontsize=16)  
plt.xlim(700, 1250)  
plt.ylim(0.55, 1.05)  
plt.title('Friction Contour Distribution', fontdict=font2)  
plt.xlabel('Laser Power (W)', fontdict=font2)  
plt.ylabel('Feed rate (rad/min)', fontdict=font2)  
plt.show()

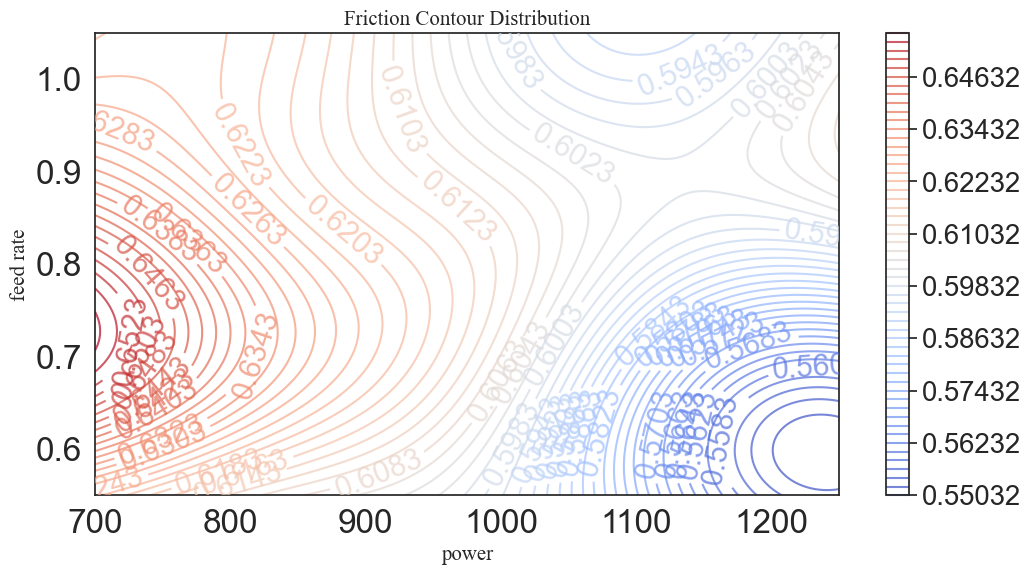


png

out\_inverse.min(),out\_inverse.max()

(0.5503183663548279, 0.6568872244464126)

# plot contour map  
plt.figure(figsize=(12,6))  
contour = plt.contour(p\_inverse, s\_inverse, out\_inverse,\  
 np.arange(out\_inverse.min(), out\_inverse.max(), 0.002),  
 off\_set=0, cmap=cm.coolwarm, alpha=0.7)  
plt.colorbar()  
plt.title('Friction Contour Distribution', fontdict=font2)  
plt.xlabel('power', fontdict=font2)  
plt.ylabel('feed rate', fontdict=font2)  
plt.clabel(contour)  
plt.tick\_params(labelsize=24)  
plt.show()



png

# UCB point selection

# upper bound of the confidence interval  
#β values of 0.1, 1, 10, 100  
ucb\_1 = out\_inverse + 0.01\*std  
ucb\_2 = out\_inverse + 0.1\*std  
ucb\_3 = out\_inverse + 1\*std  
ucb\_4 = out\_inverse + 10\*std

# calculate UCB data  
output1\_data = pd.DataFrame(columns=['power', 'feed rate', 'friction'])  
output1\_data['power'] = p\_inverse.ravel()  
output1\_data['feed rate'] = s\_inverse.ravel()  
output1\_data['friction'] = out\_inverse.ravel()  
output1\_data['ucb-0.01'] = ucb\_1.ravel()  
output1\_data['ucb-0.1'] = ucb\_2.ravel()  
output1\_data['ucb-1'] = ucb\_3.ravel()  
output1\_data['ucb-10'] = ucb\_4.ravel()  
output1\_data.head()

power

feed rate

friction

ucb-0.01

ucb-0.1

ucb-1

ucb-10

0

700.000000

0.55

0.622846

0.627526

0.669639

1.090770

5.302083

1

701.707825

0.55

0.622665

0.627325

0.669259

1.088609

5.282103

2

703.415650

0.55

0.622483

0.627123

0.668885

1.086509

5.262751

3

705.123475

0.55

0.622299

0.626920

0.668516

1.084471

5.244023

4

706.831301

0.55

0.622114

0.626717

0.668152

1.082494

5.225913

a = (p\_inverse.min() + p\_inverse.max())/2  
b = (s\_inverse.min() + s\_inverse.max())/2

x1, y1, x2, y2 = [], [], [], []  
for i in range(11):  
 x1.append(a)  
 y1.append(600 + 112\*i)  
 x2.append(100 + 26\*i)  
 y2.append(b)

a,b

(974.9598455532495, 0.7998805964856017)

# divide regions based on UCB for point selection  
temp1 = output1\_data[(output1\_data['power'] <= a) &\  
 (output1\_data['feed rate'] <= b)]  
temp2 = output1\_data[(output1\_data['power'] > a) &\  
 (output1\_data['feed rate'] <= b)]  
temp3 = output1\_data[(output1\_data['power'] <= a) &\  
 (output1\_data['feed rate'] > b)]  
temp4 = output1\_data[(output1\_data['power'] > a) &\  
 (output1\_data['feed rate'] > b)]

ucb\_res = pd.DataFrame(columns=['power', 'feed rate', 'friction', 'ucb'])  
temp = [output1\_data]  
ucb = ['ucb-0.01', 'ucb-0.1', 'ucb-1', 'ucb-10']  
k = 1  
for t in temp:  
 for u in ucb:  
 sort\_data = t.sort\_values(by=u, ascending=True)  
 filter\_data = sort\_data[['power', 'feed rate', 'friction', u]].iloc[0:k]  
 index = [u]  
 filter\_data.index = index  
 filter\_data.columns = ucb\_res.columns  
 ucb\_res = pd.concat([ucb\_res, filter\_data])

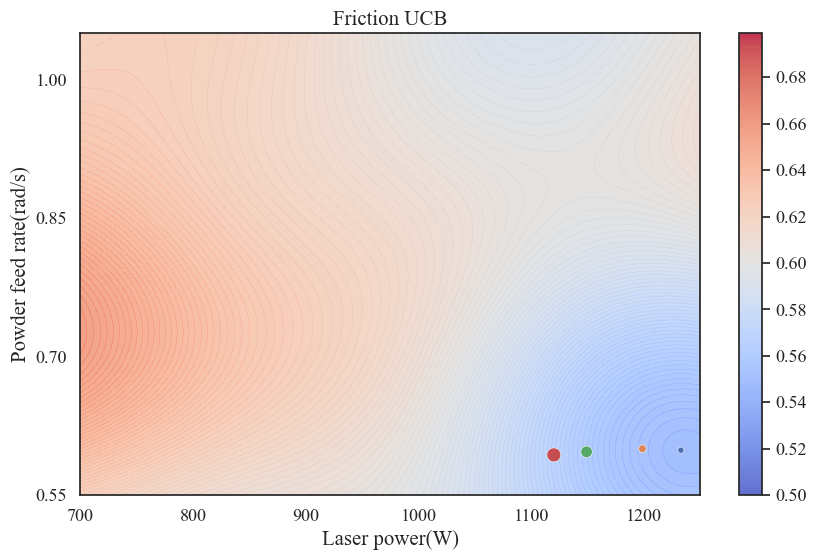
ucb\_data = ucb\_res.drop\_duplicates(subset=['power', 'feed rate'], keep='first')  
ucb\_data['ucbβ'] = ucb\_data.index  
ucb\_data = ucb\_data[['ucbβ', 'power', 'feed rate', 'friction', 'ucb']]  
ucb\_data.shape

(4, 5)

# write UCB selected data  
save\_path =r'C:\Users\Administrator\Desktop\3.xls'  
size = ucb\_data.shape  
work\_book = xlwt.Workbook(encoding='utf-8')  
work\_sheet = work\_book.add\_sheet('\_ucb')  
columns = list(ucb\_data.columns)  
for i in range(size[1]):  
 work\_sheet.write(0, i, columns[i])  
for i in range(size[0]):  
 for j in range(size[1]):  
 work\_sheet.write(i+1, j, ucb\_data.iloc[i, j])  
work\_book.save(save\_path)  
print('Done!')

Done!

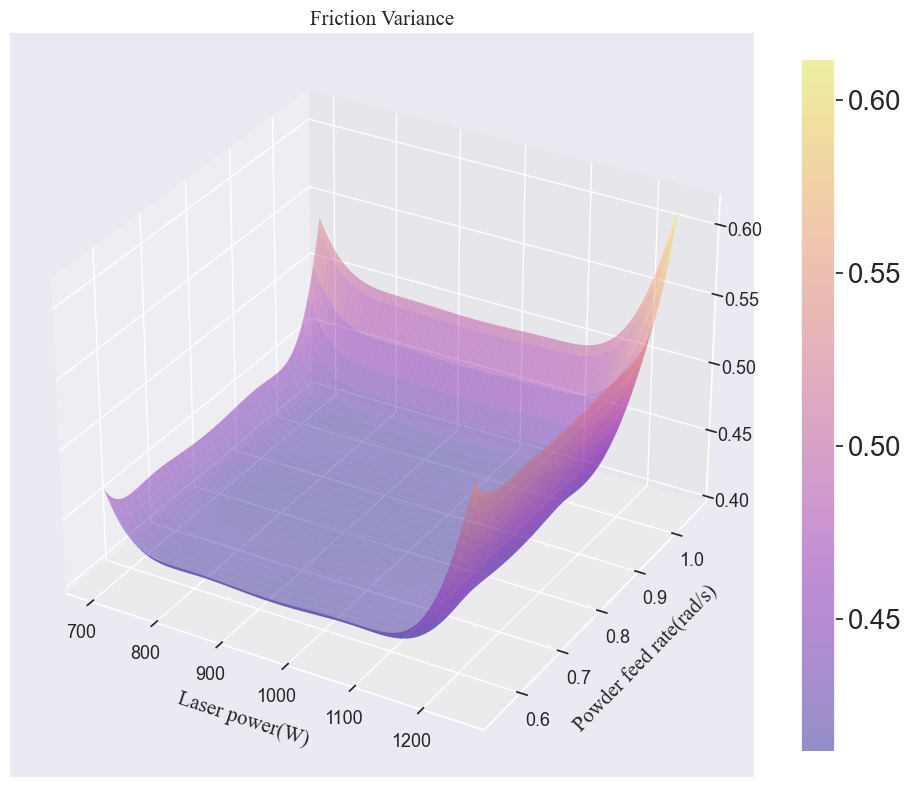
# visualize and compare UCB selected points  
# plot contour map   
sns.set\_style('white')  
fig, ax = plt.subplots(figsize=(10,6))  
plt.contourf(p\_inverse, s\_inverse,\  
 out\_inverse, np.arange(0.5, 0.7, 0.001),\  
 offset=0, cmap=cm.coolwarm,\  
 alpha=0.8)  
p1 = sns.scatterplot(x='power', y='feed rate', size='friction',\  
 hue='ucbβ',\  
 hue\_order=['ucb-0.01', 'ucb-0.1', 'ucb-1', 'ucb-10'],\  
 sizes=(20, 100), data=ucb\_data, legend=False)  
plt.tick\_params(labelsize=13)  
labels = ax.get\_xticklabels() + ax.get\_yticklabels()  
[label.set\_fontname('Times New Roman') for label in labels]  
cb = plt.colorbar()  
cb\_labels = cb.ax.yaxis.get\_ticklabels()  
for cb\_label in cb\_labels:  
 cb\_label.set\_family('Times New Roman')  
 cb\_label.set\_fontsize(13)  
x\_tick = [600 + i\*100 for i in range(7)]  
y\_tick = [0.55 + j\*0.15 for j in range(4)]  
plt.xticks(x\_tick, fontsize=13)  
plt.yticks(y\_tick, fontsize=13)  
plt.xlim(700, 1250)  
plt.ylim(0.55, 1.05)  
plt.plot(x1, y1)  
plt.plot(x2, y2, 'r')  
plt.title('Friction UCB', fontdict=font2)  
plt.xlabel('Laser power(W)', fontdict=font2)  
plt.ylabel('Powder feed rate(rad/s)', fontdict=font2)  
plt.show()



png

# Variance visualization

sns.set\_style('darkgrid')  
fig1 = plt.figure(figsize=(12,10))  
ax1 = fig1.add\_subplot(111, projection='3d')  
surf = ax1.plot\_surface(p\_inverse, s\_inverse, std, rstride=8,\  
 cstride=3, linewidth=0, alpha=0.4,\  
 antialiased=True, cmap=cm.plasma)  
ax1.set\_title('Friction Variance', fontdict=font2)  
ax1.set\_xlabel('\nLaser power(W)', fontdict=font2)  
ax1.set\_ylabel('\nPowder feed rate(rad/s)', fontdict=font2)  
cb1 = plt.colorbar(surf, shrink=0.9)  
cb1.set\_ticks([0.45, 0.5, 0.55, 0.6])  
cb1.update\_ticks()  
ax1.set\_zticks([0.4, 0.45, 0.5, 0.55, 0.6])  
plt.tick\_params(labelsize=13)  
plt.show()



png

std.min()

0.41176257547792056