# Microhardness data normalizition modeling

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
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  
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%-microharness.xlsx'# dataset address  
df = pd.read\_excel(filepath, skiprows=0)  
df.head()

power

feed rate

microhardness

0

700

1.0

466

1

800

1.0

503

2

900

1.0

501

3

1000

1.0

505

4

1100

1.0

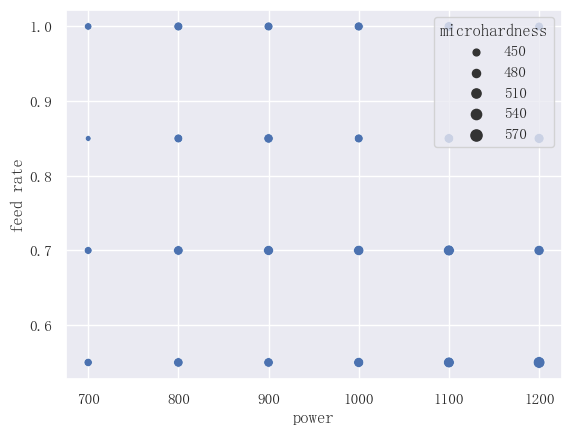
520

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

(24, 3)

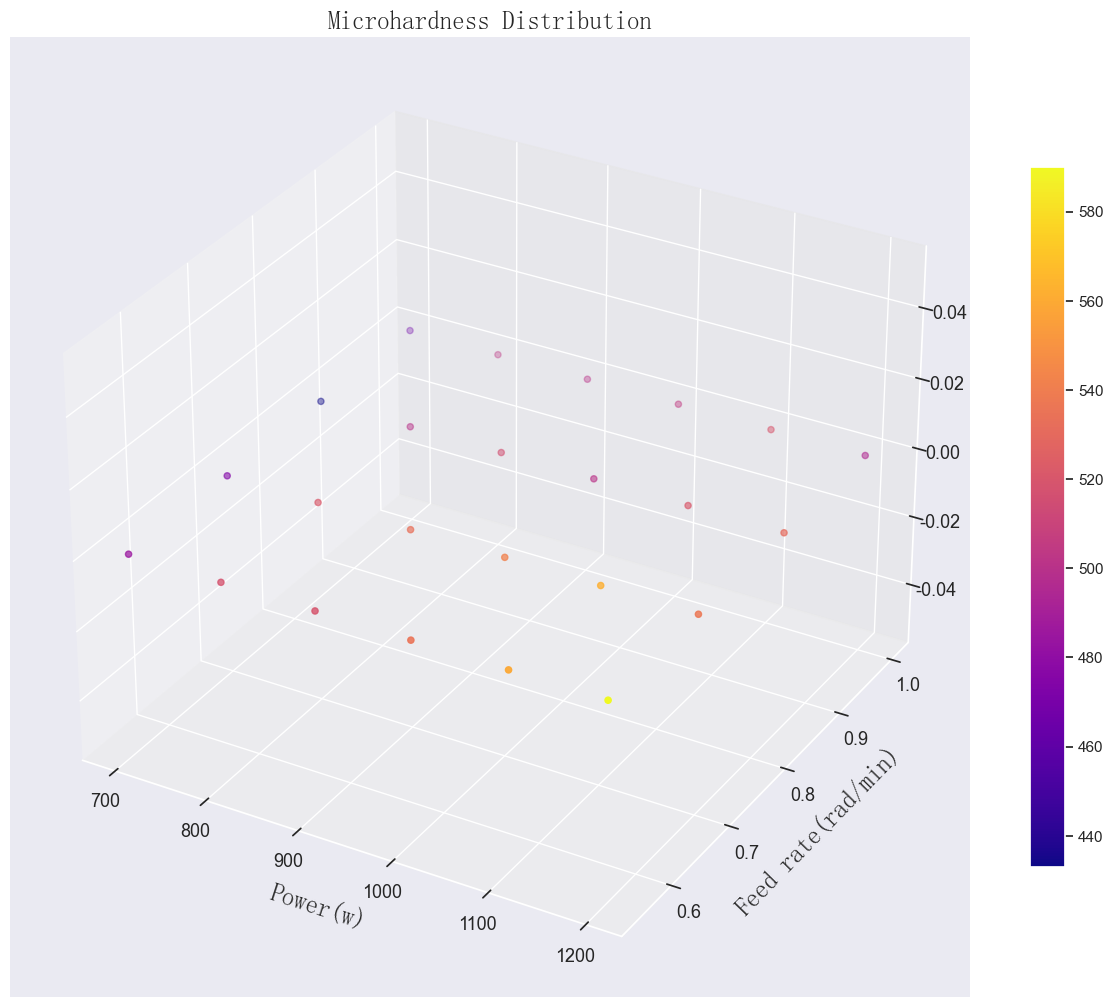
# examine microhardness distribution   
sns.scatterplot(x='power', y='feed rate', size='microhardness',\  
 data=df)

<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['microhardness'],\  
 cmap=cm.plasma)  
plt.colorbar(ax\_obj, shrink=0.7)  
ax.set\_title('Microhardness 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='microhardness').T  
data.sort\_index(ascending=False, inplace=True)  
data

power

700

800

900

1000

1100

1200

feed rate

1.00

466

503

501

505

520

495

0.85

433

500

516

501

521

531

0.70

474

521

533

542

562

537

0.55

485

518

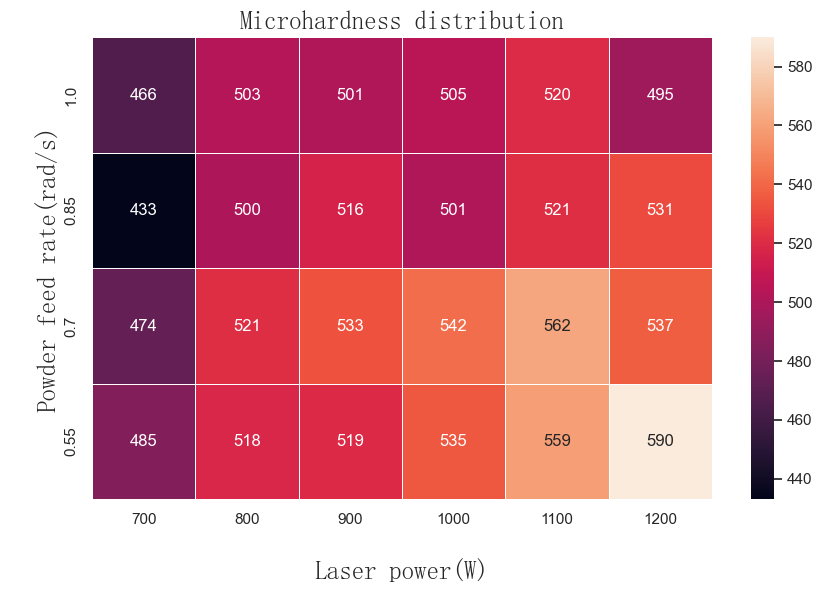
519

535

559

590

# plot the color map  
fig= plt.subplots(figsize=(10,6))  
ax\_obj1 = sns.heatmap(data, annot=True, fmt="d", linewidths=.6)  
plt.title('Microhardness 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[['microhardness']])  
scaler\_Y.mean\_  
scaler\_Y.scale\_  
new\_Y = scaler\_Y.transform(df[['microhardness']])

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

new\_df.head()

power

feed rate

microhardness

0

-1.46385

1.341641

-1.525692

1

-0.87831

1.341641

-0.380456

2

-0.29277

1.341641

-0.442360

3

0.29277

1.341641

-0.318551

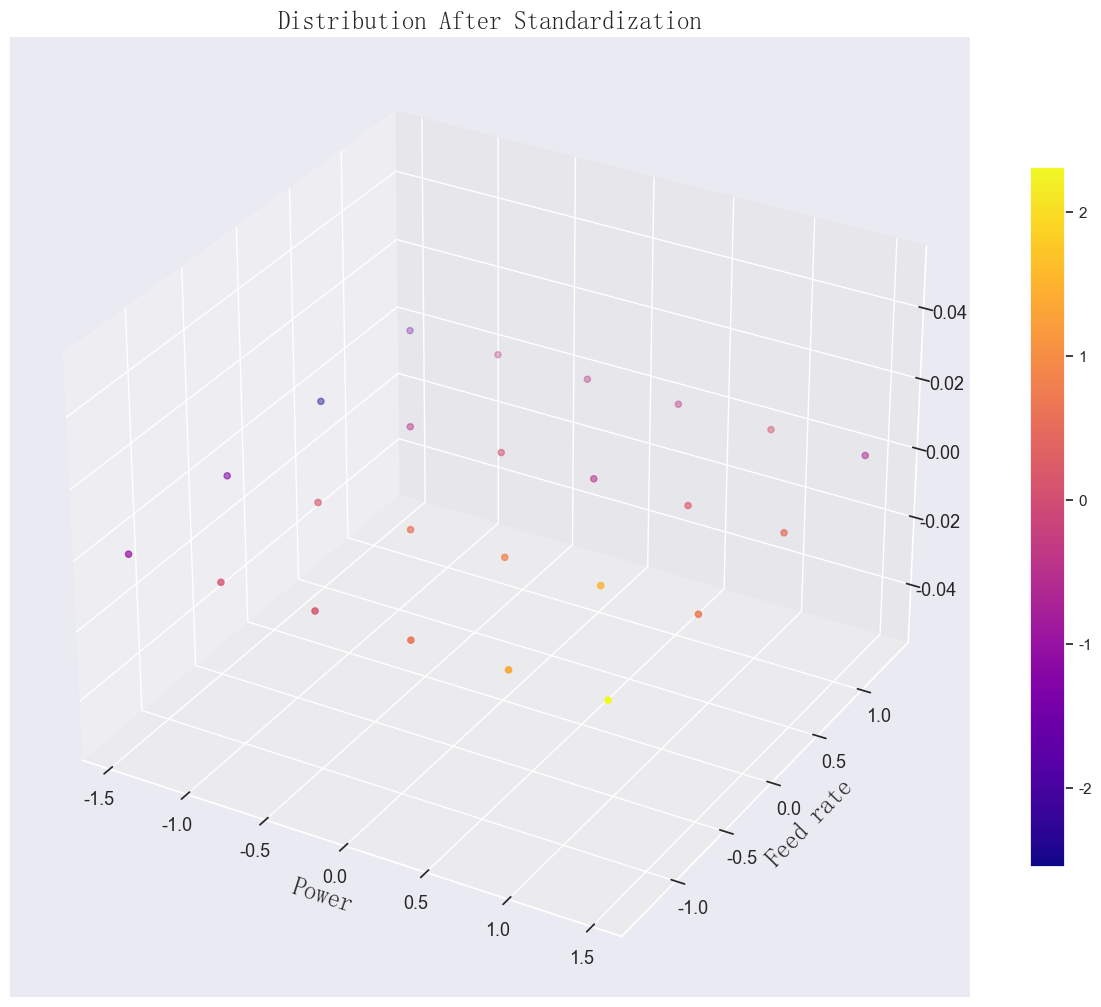
4

0.87831

1.341641

0.145734

# 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['microhardness'],\  
 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

shuffle = ShuffleSplit(n\_splits=10, test\_size=4, random\_state=42)  
index = np.arange(16)  
for train\_index, test\_index in shuffle.split(index):  
 print(train\_index)  
 print(test\_index)

[13 11 8 9 2 15 4 7 10 12 3 6]  
[ 0 1 5 14]  
[ 9 12 10 3 0 15 14 5 11 7 1 4]  
[ 6 2 8 13]  
[ 4 14 0 7 3 6 2 12 11 9 15 10]  
[ 1 13 8 5]  
[12 7 4 10 11 9 13 15 3 1 6 8]  
[ 0 5 2 14]  
[ 5 8 9 13 1 3 0 2 7 12 11 6]  
[14 10 4 15]  
[ 0 10 6 5 7 3 13 15 11 9 1 14]  
[ 2 8 4 12]  
[ 5 15 6 11 0 10 8 9 7 4 1 12]  
[14 13 2 3]  
[ 8 3 10 1 6 4 14 12 15 7 0 2]  
[11 9 13 5]  
[13 0 9 10 5 12 2 4 7 14 6 15]  
[ 3 1 11 8]  
[ 3 14 7 10 15 8 11 6 9 4 0 2]  
[12 13 1 5]

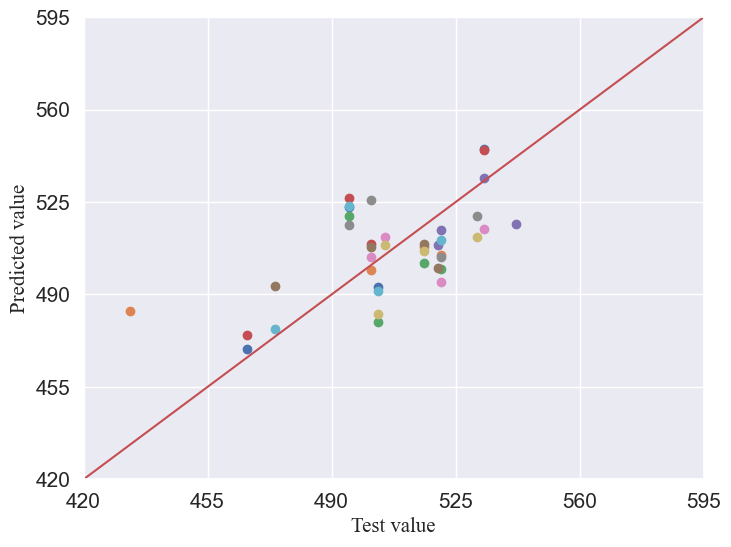
x2 = [420, 455, 490, 525, 560, 595]  
sns.set\_style('darkgrid')

def cv\_loop2(reg):  
 mses = []  
 maes = []  
 maes\_relative = []  
 i = 1  
 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['microhardness'].iloc[train\_index]  
 x\_test = new\_df[['power', 'feed rate']].iloc[test\_index]  
 y\_test = new\_df['microhardness'].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)  
 plt.scatter(y\_test\_inverse, 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\_relative.append(mae/np.mean(y\_test\_inverse))  
 i += 1  
 plt.plot(x2, x2, 'r-')  
 plt.xlim(420, 595)  
 plt.ylim(420, 595)  
 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

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 = cv\_loop2(reg\_1)

-13.412538494550702  
-11.97214495008163  
-13.810910481789517  
-13.767548265459304  
-14.60368346492887  
-14.92854556779503  
-14.814737028759598  
-13.370902193873171  
-14.715341346510659  
-13.464172582444105



png

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

(17.796925509306437, 15.133874990829066, 0.02990012502880124)

# Gaussian Process Regression

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

kernel\_1 = 1.0 \* RBF(length\_scale=1,\  
 length\_scale\_bounds=(1e-3, 1e2)) +\  
WhiteKernel(noise\_level=1e-3, noise\_level\_bounds=(1e-10, 1e1))  
reg\_1 = GaussianProcessRegressor(kernel=kernel\_1,\  
 n\_restarts\_optimizer=12,\  
 alpha=0, random\_state=42)  
reg\_1.fit(new\_df[['power', 'feed rate']], new\_df['microhardness'])

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

theta = reg\_1.kernel\_.k1.theta  
noise\_level = reg\_1.kernel\_.k2.noise\_level  
print('optimized theta: {}\n optimized noise\_level: {}'\  
 .format(theta, noise\_level))

optimized theta: [0.78458968 0.8398592 ]  
 optimized noise\_level: 0.239298005523145

## Export hyperparameters

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

({'alpha': 0,  
 'copy\_X\_train': True,  
 'kernel\_\_k1': 1\*\*2 \* RBF(length\_scale=1),  
 'kernel\_\_k2': WhiteKernel(noise\_level=0.001),  
 'kernel\_\_k1\_\_k1': 1\*\*2,  
 'kernel\_\_k1\_\_k2': RBF(length\_scale=1),  
 'kernel\_\_k1\_\_k1\_\_constant\_value': 1.0,  
 'kernel\_\_k1\_\_k1\_\_constant\_value\_bounds': (1e-05, 100000.0),  
 'kernel\_\_k1\_\_k2\_\_length\_scale': 1,  
 'kernel\_\_k1\_\_k2\_\_length\_scale\_bounds': (0.001, 100.0),  
 'kernel\_\_k2\_\_noise\_level': 0.001,  
 'kernel\_\_k2\_\_noise\_level\_bounds': (1e-10, 10.0),  
 'kernel': 1\*\*2 \* RBF(length\_scale=1) + WhiteKernel(noise\_level=0.001),  
 'n\_restarts\_optimizer': 12,  
 'n\_targets': None,  
 'normalize\_y': False,  
 'optimizer': 'fmin\_l\_bfgs\_b',  
 'random\_state': 42},  
 -24.90949851584758)

# 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([[-0.79011718, -0.77982157, -0.76949433, ..., 1.65955374,  
 1.65979042, 1.6599775 ],  
 [-0.79393538, -0.78362508, -0.77328318, ..., 1.65560917,  
 1.65583558, 1.65601244],  
 [-0.79776061, -0.7874358 , -0.7770794 , ..., 1.65161011,  
 1.65182623, 1.65199285],  
 ...,  
 [-1.30023403, -1.29307306, -1.28590472, ..., -0.42134032,  
 -0.42346986, -0.42560408],  
 [-1.29757646, -1.29044241, -1.2833011 , ..., -0.42577253,  
 -0.42790494, -0.4300419 ],  
 [-1.29488864, -1.28778153, -1.28066726, ..., -0.43015956,  
 -0.4322948 , -0.43443445]])

# 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', 'microhardness'])  
output1\_data['power'] = p\_set1.ravel()  
output1\_data['feed rate'] = s\_set1.ravel()  
output1\_data['microhardness'] = 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[['microhardness']])

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

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

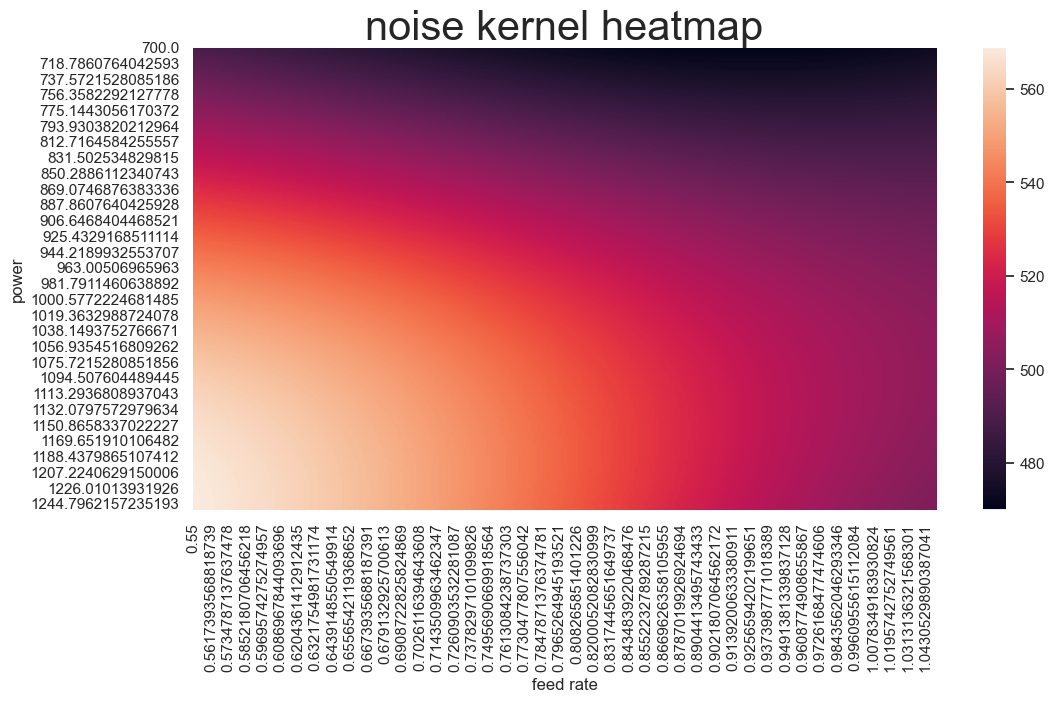
p\_inverse = scale\_X[:, 0].reshape(p\_set1.shape)  
s\_inverse = scale\_X[:, 1].reshape(s\_set1.shape)  
out\_inverse = scale\_Y.reshape(p\_set1.shape)

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

1.0497611929712034

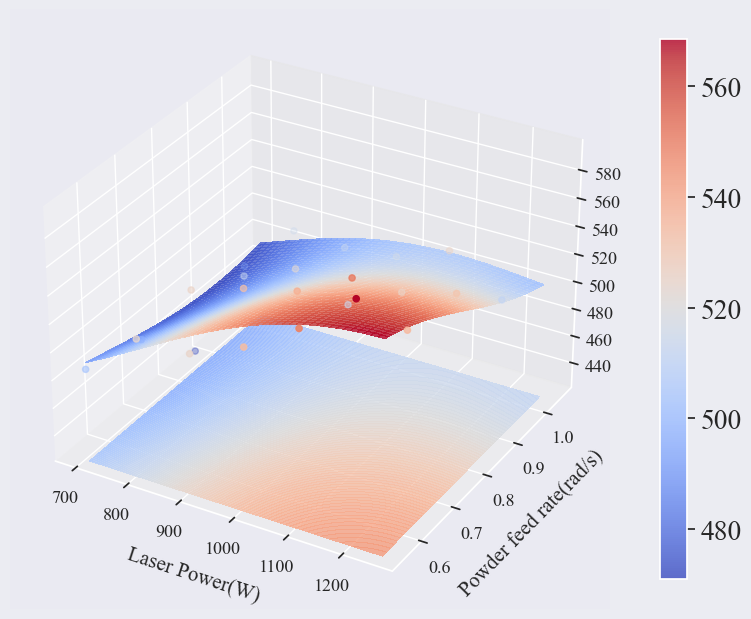
output\_pivot = new\_output.pivot(index='power', columns='feed rate', values='microhardness')  
f, ax = plt.subplots(figsize=(12,6))  
sns.heatmap(output\_pivot, ax=ax)  
ax.set\_title('noise kernel heatmap', fontsize=30)

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



png

# 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['microhardness'],\  
 c=df['microhardness'], cmap=cm.coolwarm)  
ax.contourf(p\_inverse, s\_inverse, out\_inverse, np.arange(420, 620, 1),\  
 zdir='output1', offset=418, cmap=cm.coolwarm, alpha=0.8)  
ax.set\_xlabel('\nLaser Power(W)', fontdict=font2)  
ax.set\_ylabel('\nPowder feed rate(rad/s)', fontdict=font2)  
plt.colorbar(surface, shrink=0.9)  
plt.tick\_params(labelsize=13)  
plt.show()

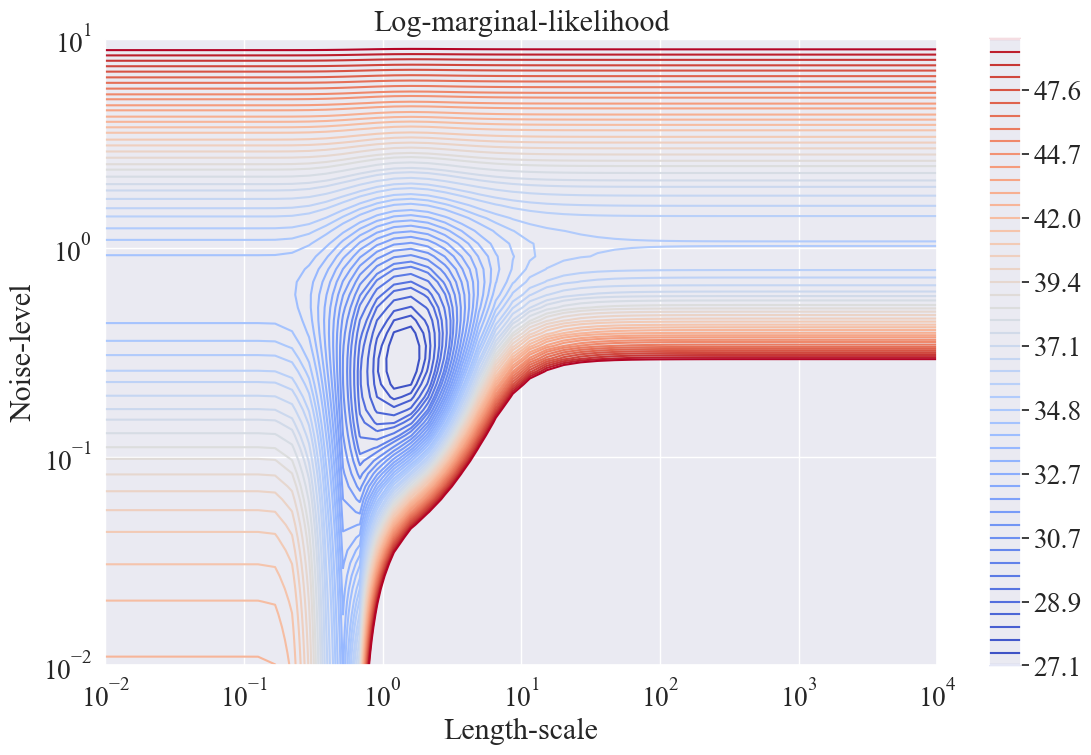


png

## 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  
27.114380349856493



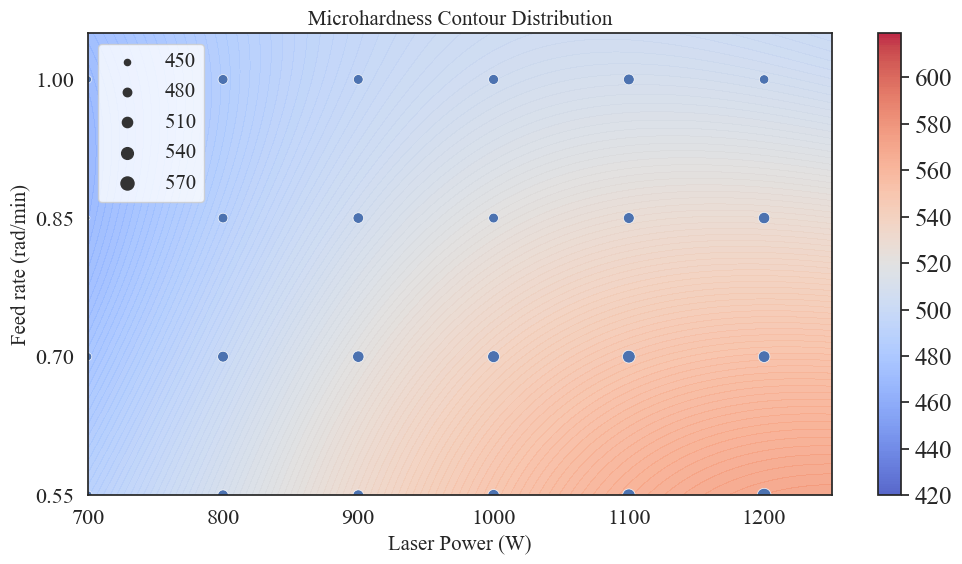
png

# 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(420, 620, 1),\  
 offset=0, cmap=cm.coolwarm,\  
 alpha=0.85)  
sns.scatterplot(x='power', y='feed rate', size='microhardness',\  
 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('Microhardness Contour Distribution', fontdict=font2)  
plt.xlabel('Laser Power (W)', fontdict=font2)  
plt.ylabel('Feed rate (rad/min)', fontdict=font2)  
plt.show()

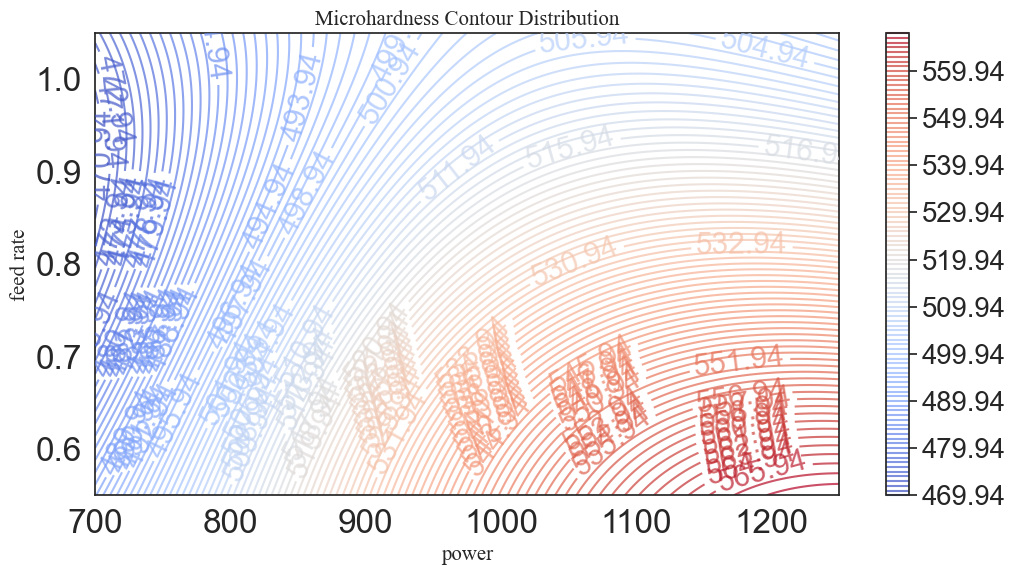


png

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

(469.9414283576257, 568.9217848129492)

# 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(), 1),  
 off\_set=0, cmap=cm.coolwarm, alpha=0.7)  
plt.colorbar()  
plt.title('Microhardness 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', 'microhardness'])  
output1\_data['power'] = p\_inverse.ravel()  
output1\_data['feed rate'] = s\_inverse.ravel()  
output1\_data['microhardness'] = 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

microhardness

ucb-0.01

ucb-0.1

ucb-1

ucb-10

0

700.000000

0.55

489.764768

489.770561

489.822696

490.344048

495.557565

1

701.707825

0.55

490.097396

490.103178

490.155222

490.675655

495.879985

2

703.415650

0.55

490.431045

490.436818

490.488771

491.008301

496.203606

3

705.123475

0.55

490.765701

490.771464

490.823328

491.341972

496.528408

4

706.831301

0.55

491.101347

491.107100

491.158878

491.676650

496.854377

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', 'microhardness', '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=False)  
 filter\_data = sort\_data[['power', 'feed rate', 'microhardness', u]].iloc[0:k]  
 index = [u]  
 filter\_data.index = index  
 filter\_data.columns = ucb\_res.columns  
 ucb\_res = pd.concat([ucb\_res, filter\_data])  
 print(ucb\_res)

power feed rate microhardness ucb  
ucb-0.01 1249.919691 0.55 568.921785 568.927959  
 power feed rate microhardness ucb  
ucb-0.01 1249.919691 0.55 568.921785 568.927959  
ucb-0.1 1249.919691 0.55 568.921785 568.983526  
 power feed rate microhardness ucb  
ucb-0.01 1249.919691 0.55 568.921785 568.927959  
ucb-0.1 1249.919691 0.55 568.921785 568.983526  
ucb-1 1249.919691 0.55 568.921785 569.539194  
 power feed rate microhardness ucb  
ucb-0.01 1249.919691 0.55 568.921785 568.927959  
ucb-0.1 1249.919691 0.55 568.921785 568.983526  
ucb-1 1249.919691 0.55 568.921785 569.539194  
ucb-10 1249.919691 0.55 568.921785 575.095876

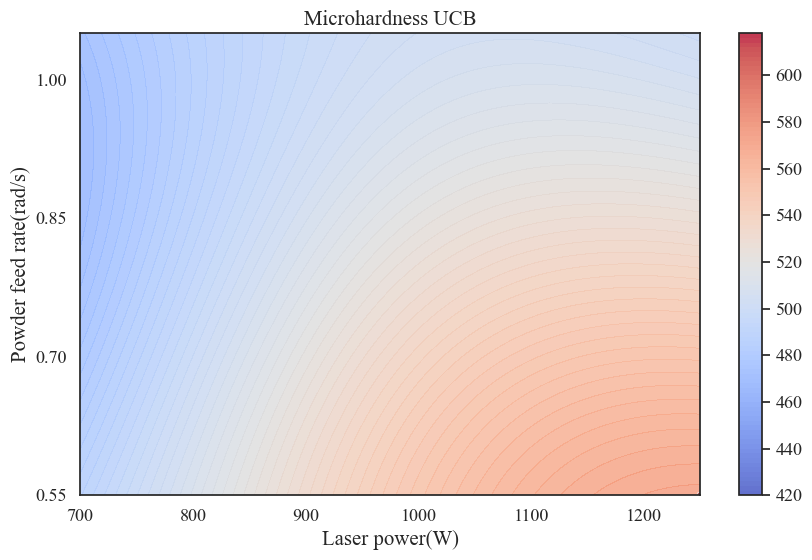
ucb\_data = ucb\_res  
ucb\_data['ucbβ'] = ucb\_data.index  
ucb\_data = ucb\_data[['ucbβ', 'power', 'feed rate', 'microhardness', 'ucb']]  
ucb\_data.shape

(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!

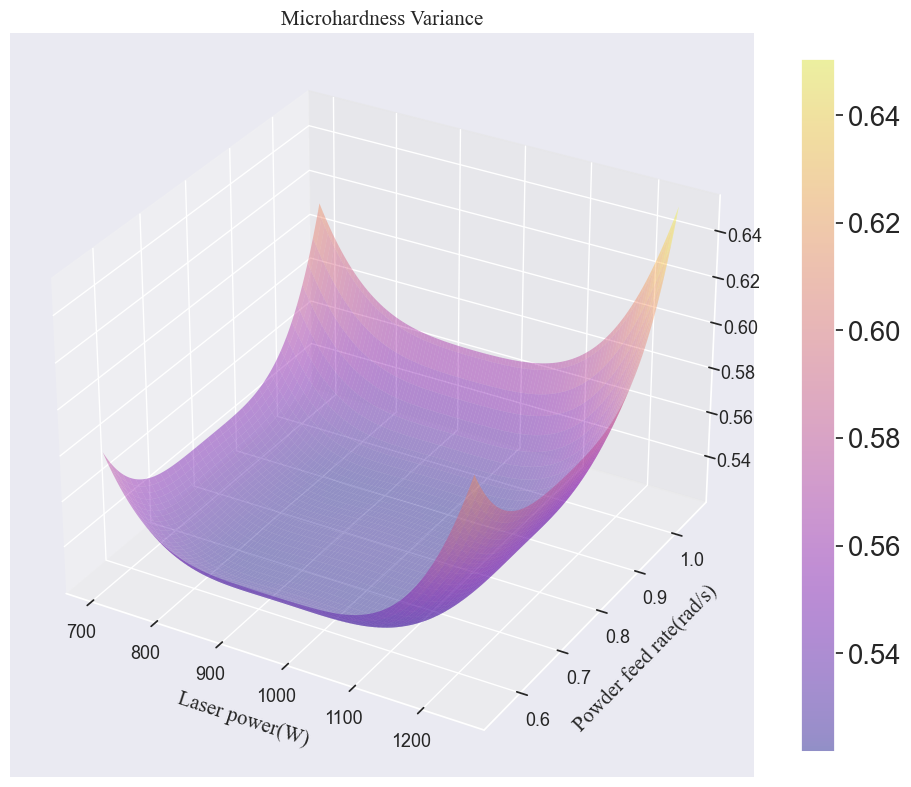
# visualize and compare UCB selected points  
# plot contour map   
sns.set\_style('white')  
fig, ax = plt.subplots(figsize=(10,6))  
plt.contourf(p\_inverse, s\_inverse,\  
 out\_inverse, np.arange(420, 620, 2),\  
 offset=0, cmap=cm.coolwarm,\  
 alpha=0.8)  
p1 = sns.scatterplot(x='power', y='feed rate', size='microhardness',\  
 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('Microhardness 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('Microhardness Variance', fontdict=font2)  
ax1.set\_xlabel('\nLaser power(W)', fontdict=font2)  
ax1.set\_ylabel('\nPowder feed rate(rad/s)', fontdict=font2)  
plt.colorbar(surf, shrink=0.9)  
plt.tick\_params(labelsize=13)  
plt.show()



png

std.min()

0.5217224756540765