FirstAssignmentDanieleTrisotto

November 1, 2025

1 Course of Multiagent Systems for smart machining: First Assignment

Student: Daniele Trisotto

Email: daniele.trisotto@studenti.unitn.it or daniele.trisotto@eciu.eu

University of Trento

1.1 Assignment:

Calculate the minimum and maximum Specific Cutting Pressure for each diameter, in both cases : Constant Cutting Speed and Constant Spindle Speed.

For that, you should identify the minimum and maximum power (Pc) used in each diameter during machining, calculate the cutting force (Fc) in each case and finally the specific pressure, also called, specific cutting force (Kc).

You can present the values in a table or in a colonne graphic, for example.

1.2 Brief Summary of Data Processing Steps

The raw data from both cutting and no-cut experiments were first imported, converted to numeric format, and cleaned by removing any missing values. The cutting power was obtained as the difference between the measured power during cutting and the corresponding no-cut condition ($P_{diff} = P_{cut} - P_{nocut}$). To ensure that only relevant machining points were considered, the dataset was filtered to include positions with programmed position x $\leq 47mm$, integer X values (representing actual machining positions), and Z positions between 0 and -40 mm (the actual cutting region). For each discrete diameter, outliers were removed in two stages: first, a DBSCAN clustering algorithm was applied on the time variable to isolate the main machining interval and discard isolated irrelevant points; second, a Median Absolute Deviation (MAD)–based threshold was used to remove extreme amplitude values (amplitude outliers). In this step, all samples whose power difference deviated more than k times the robust standard deviation (computed as $1.4826 \times MAD$) from the median were discarded ($k = 3$ for Vc = 150 m/min and $k = 2$ for N = 1200 rpm, because data are less variate). The cleaned dataset was then used to compute the specific cutting pressure ($K_c = \frac{P_c \cdot 60}{V_c \cdot a_p \cdot f}$), where ($P_c$) is the filtered power difference, (V_c) is the cutting speed (either constant or diameter-dependent), (a_p) the depth of cut, and (f) the feed per revolution. Finally, the minimum, maximum, and mean values of (K_c) were calculated for each diameter.

1.3 Importing Data

[21]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker


# File data CSV
filedataC150 = 'ANNACHPTVC150.csv'                          #cut
filedataNC150 = 'ANNACHPTVC150VIDE.csv'                     #nocut
filedataC1200 = 'ANNACHPT1200RPM.csv'                       #cut
filedataNC1200 = 'ANNACHPT1200RPMVIDE.csv'                  #nocut

# Reading csv
dfC150 = pd.read_csv(filedataC150, skiprows=10)
dfNC150 = pd.read_csv(filedataNC150, skiprows=10)
dfC1200 = pd.read_csv(filedataC1200, skiprows=10)
dfNC1200 = pd.read_csv(filedataNC1200, skiprows=10)

# # Columns names / Same for all data
dfC150.columns = [
    'time',
    'power',
    'prog_pos_z',
    'prog_pos_x',
    'meas_pos_z',
    'meas_pos_x',
    'spindle1' # only the first data does not have the spindle 2 data
]
dfNC150.columns = [
    'time',
    'power',
    'prog_pos_z',
    'prog_pos_x',
    'meas_pos_z',
    'meas_pos_x',
    'spindle1',
    'spindle2'
]
dfC1200.columns = [
    'time',
    'power',
    'prog_pos_z',
    'prog_pos_x',
    'meas_pos_z',
    'meas_pos_x',
```

```
    'spindle1',
    'spindle2'
]
dfNC1200.columns = [
    'time',
    'power',
    'prog_pos_z',
    'prog_pos_x',
    'meas_pos_z',
    'meas_pos_x',
    'spindle1',
    'spindle2'
]

cols_to_convert = dfC150.columns.difference(['time'])

# # Conversion in numeric format
for df in [dfC150, dfNC150, dfC1200, dfNC1200]:
    for col in cols_to_convert:
        df[col] = pd.to_numeric(df[col], errors='coerce')




dfC150 = dfC150.dropna()
dfNC150 = dfNC150.dropna()
dfC1200 = dfC1200.dropna()
dfNC1200 = dfNC1200.dropna()
```

Programed Parameters:

[22]:
```
ap=1  #mm
f=0.2 #mm/rev
Vc150=150 #m/min
```

1.4 Specific Cutting Pressure Calculation

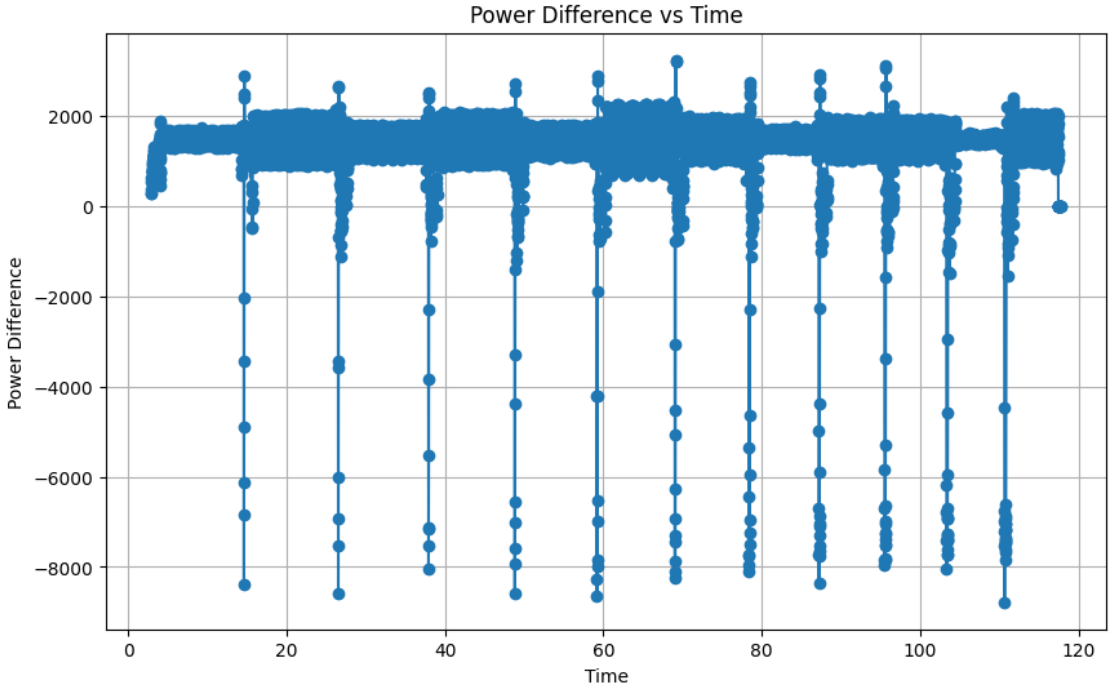
1.4.1 Calculating Cutting Power

[23]:
```
# find minimum difference in terms of length between two dataframes
n150 = min(len(dfC150), len(dfNC150))
n1200 = min(len(dfC1200), len(dfNC1200))

# Initializing a column of power difference
dfC150['power_diff']=0.0
dfC1200['power_diff']=0.0

# Calculating the power difference between cutting and non cutting
```

```
dfC150.loc[dfC150.index[:n150], 'power_diff'] = dfC150['power'].iloc[:n150].
 ↪values - dfNC150['power'].iloc[:n150].values
dfC1200.loc[dfC1200.index[:n1200], 'power_diff'] = dfC1200['power'].iloc[:
 ↪n1200].values - dfNC1200['power'].iloc[:n1200].values

#Filtering only machining positions less than 47 and exact machining X␣
 ↪positions and Z positions between 0 and -40
dfC150_filtered = dfC150[(dfC150['prog_pos_x'] <= 47)&(np.
 ↪isclose(dfC150['prog_pos_x'] % 1, 0.0))&(dfC150['prog_pos_z']<=0.
 ↪0)&(dfC150['prog_pos_z']>=-40.0)].copy()
dfC1200_filtered = dfC1200[(dfC1200['prog_pos_x'] <= 47)&(np.
 ↪isclose(dfC1200['prog_pos_x'] % 1, 0.0))&(dfC1200['prog_pos_z']<=0.
 ↪0)&(dfC1200['prog_pos_z']>=-40.0)].copy()
```

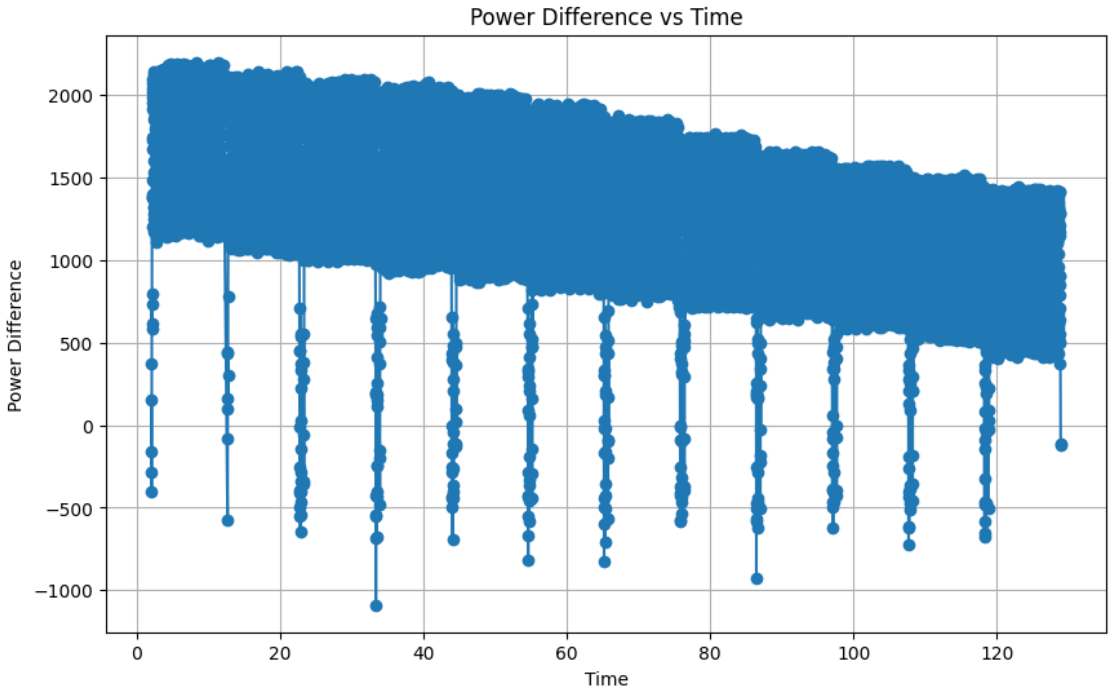
Cutting power for constant cutting speed

[24]:
```
plt.figure(figsize=(10, 6))
plt.plot(dfC150_filtered['time'], dfC150_filtered['power_diff'], marker='o',␣
 ↪linestyle='-')
plt.xlabel('Time')
plt.ylabel('Power Difference')
plt.title('Power Difference vs Time')
plt.grid(True)
plt.show()
```



Cutting Power for constant spindle speed

[25]:
```
plt.figure(figsize=(10, 6))
plt.plot(dfC1200_filtered['time'], dfC1200_filtered['power_diff'], marker='o',␣
↪linestyle='-')
plt.xlabel('Time')
plt.ylabel('Power Difference')
plt.title('Power Difference vs Time')
plt.grid(True)
plt.show()
```



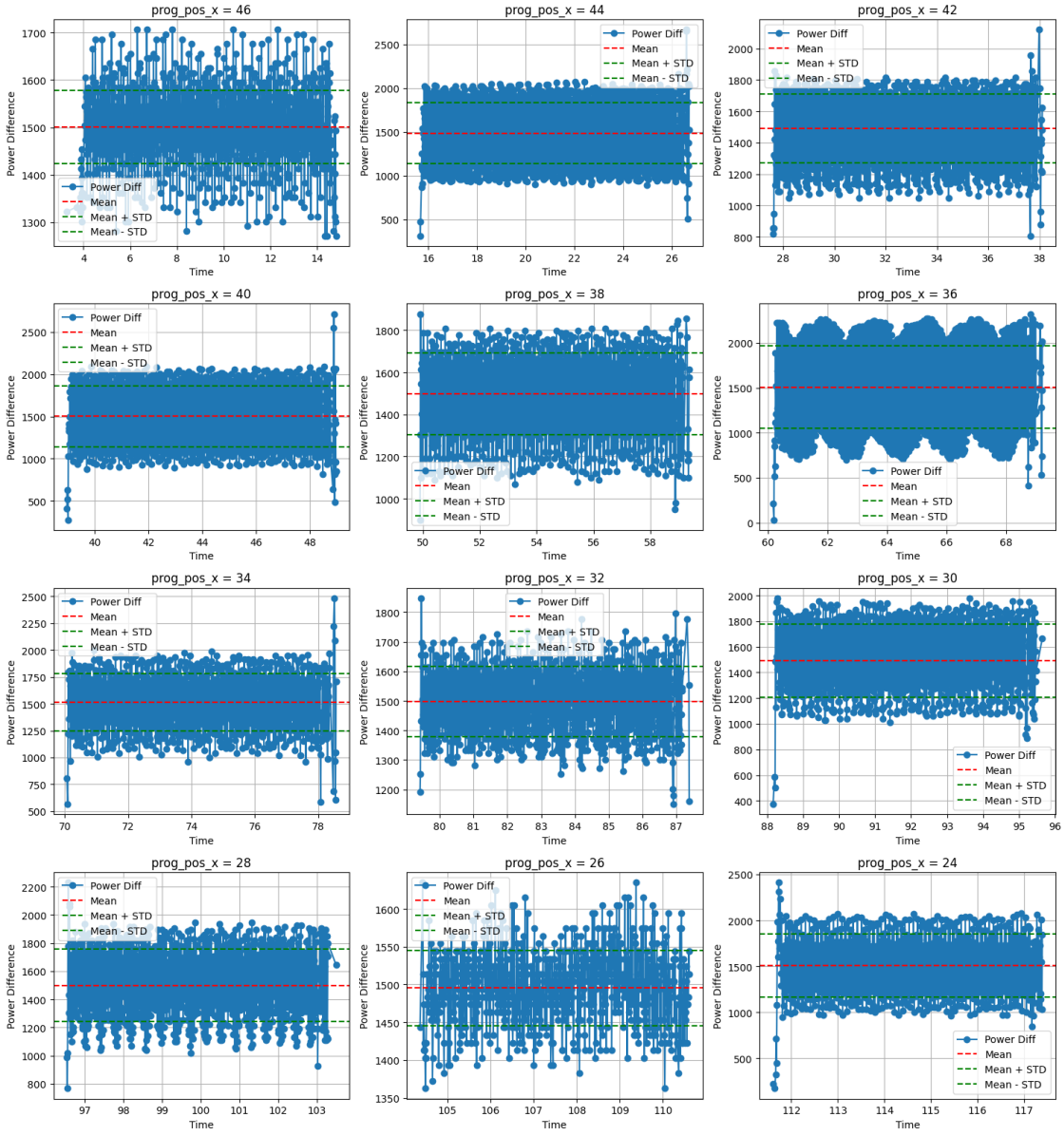
From the previous graphs we can see that there are some outliers, in the next section outliers are removed

1.4.2 Outliers Removing from Vc=150

[26]:
```
from sklearn.cluster import DBSCAN
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
import pandas as pd

# Diameter List
prog_values = list(range(46, 23, -2))

# Subplot settings
```

```
n_cols = 3
n_rows = (len(prog_values) + n_cols - 1) // n_cols
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows*4))
axes = axes.flatten()

# new dataframe for saving clean data
df_150 = pd.DataFrame()

for i, prog in enumerate(prog_values):
    df_prog = dfC150_filtered[dfC150_filtered['prog_pos_x'] == prog]
    if df_prog.empty:
        continue

    # DBSCAN for time outliers
    time_values = df_prog['time'].values.reshape(-1, 1)
    db = DBSCAN(eps=2.0, min_samples=5)
    labels = db.fit_predict(time_values)

    label_counts = Counter(labels)
    if -1 in label_counts:
        del label_counts[-1]
    if not label_counts:
        continue

    main_cluster = label_counts.most_common(1)[0][0]
    df_cluster = df_prog[labels == main_cluster]

    # robust standard deviation for amplitude outliers filtering
    median_val = df_cluster['power_diff'].median()
    mad_val = np.median(np.abs(df_cluster['power_diff'] - median_val))
    robust_std = 1.4826 * mad_val
    df_cluster_clean = df_cluster[np.abs(df_cluster['power_diff'] - median_val)␣
 ↪<= 3 * robust_std]

    # clean data saving
    df_150 = pd.concat([df_150, df_cluster_clean], ignore_index=True)

    # mean and standard deviation on clean data for plotting
    mean_val = df_cluster_clean['power_diff'].mean()
    std_val = df_cluster_clean['power_diff'].std()

    # plot
    ax = axes[i]
    ax.plot(df_cluster_clean['time'], df_cluster_clean['power_diff'],␣
 ↪marker='o', linestyle='-', label='Power Diff')
    ax.axhline(mean_val, color='red', linestyle='--', label='Mean')
```

```
    ax.axhline(mean_val + std_val, color='green', linestyle='--', label='Mean +␣
 ↪STD')
    ax.axhline(mean_val - std_val, color='green', linestyle='--', label='Mean -␣
 ↪STD')

    ax.set_title(f'prog_pos_x = {prog}')
    ax.set_xlabel('Time')
    ax.set_ylabel('Power Difference')
    ax.grid(True)
    ax.legend()


for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

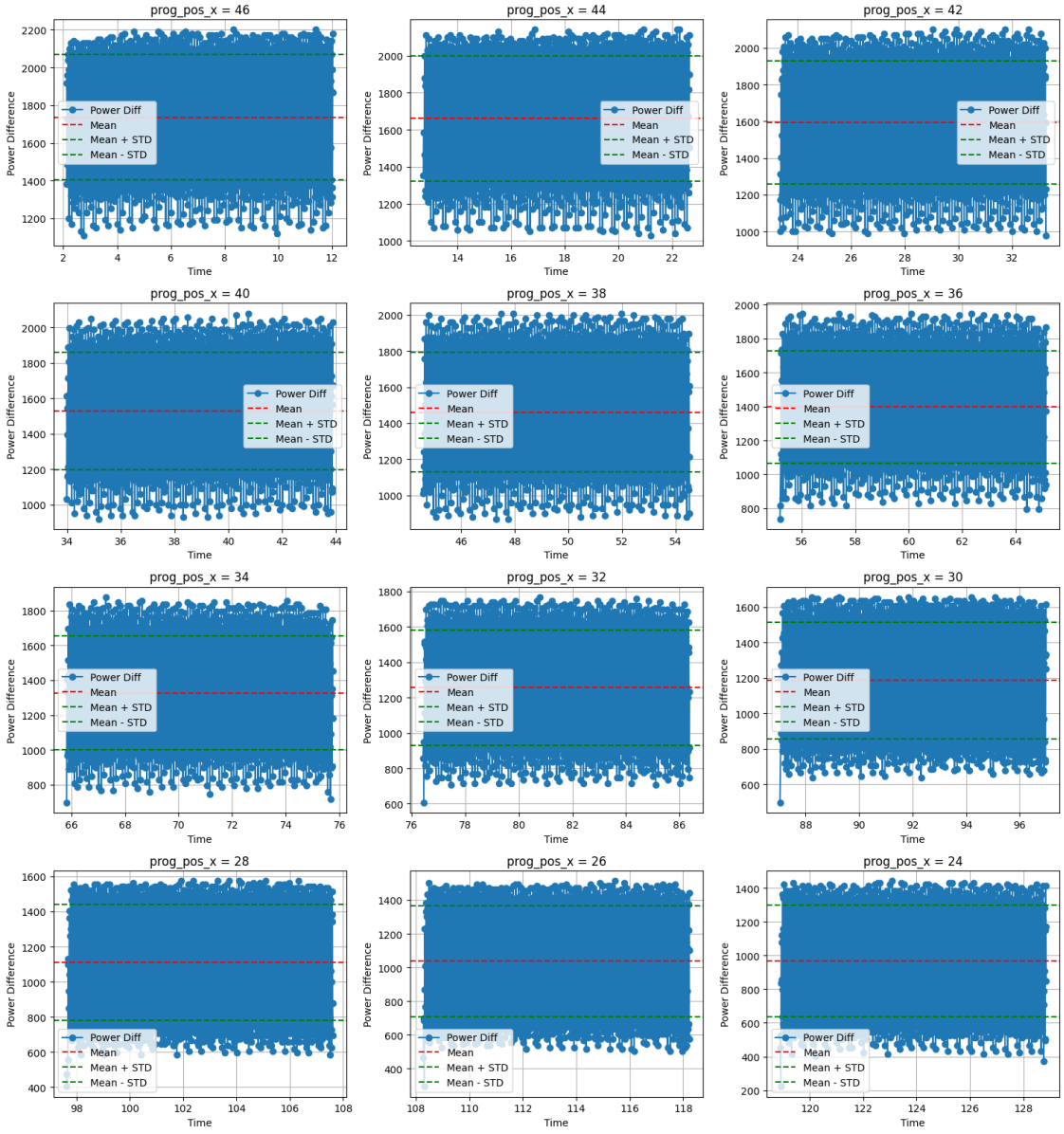


1.4.3 Outliers Removing from N=1200

[27]:
```
from sklearn.cluster import DBSCAN
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
import pandas as pd

# Diameter List
prog_values = list(range(46, 23, -2))
```

```
# Subplot settings
n_cols = 3
n_rows = (len(prog_values) + n_cols - 1) // n_cols
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows*4))
axes = axes.flatten()

# new dataframe for saving clean data
df_1200 = pd.DataFrame()

for i, prog in enumerate(prog_values):
    df_prog = dfC1200_filtered[dfC1200_filtered['prog_pos_x'] == prog]
    if df_prog.empty:
        continue

    # DBSCAN for time outliers
    time_values = df_prog['time'].values.reshape(-1, 1)
    db = DBSCAN(eps=2.0, min_samples=5)
    labels = db.fit_predict(time_values)

    label_counts = Counter(labels)
    if -1 in label_counts:
        del label_counts[-1]
    if not label_counts:
        continue

    main_cluster = label_counts.most_common(1)[0][0]
    df_cluster = df_prog[labels == main_cluster]

    # robust standard deviation for amplitude outliers filtering
    median_val = df_cluster['power_diff'].median()
    mad_val = np.median(np.abs(df_cluster['power_diff'] - median_val))
    robust_std = 1.4826 * mad_val
    df_cluster_clean = df_cluster[np.abs(df_cluster['power_diff'] - median_val)␣
 ↪<= 1.5 * robust_std]

    # clean data saving
    df_1200 = pd.concat([df_1200, df_cluster_clean], ignore_index=True)

    # mean and standard deviation on clean data for plotting
    mean_val = df_cluster_clean['power_diff'].mean()
    std_val = df_cluster_clean['power_diff'].std()

    # plot
    ax = axes[i]
    ax.plot(df_cluster_clean['time'], df_cluster_clean['power_diff'],␣
 ↪marker='o', linestyle='-', label='Power Diff')
```

```
    ax.axhline(mean_val, color='red', linestyle='--', label='Mean')
    ax.axhline(mean_val + std_val, color='green', linestyle='--', label='Mean +␣
 ↪STD')
    ax.axhline(mean_val - std_val, color='green', linestyle='--', label='Mean -␣
 ↪STD')

    ax.set_title(f'prog_pos_x = {prog}')
    ax.set_xlabel('Time')
    ax.set_ylabel('Power Difference')
    ax.grid(True)
    ax.legend()

for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



1.4.4 KC Calculation for Vc=150

[34]:
```
# adding a column for Kc and calculating it
df_150['Kc']=df_150['power_diff']*60/(Vc150*ap*f)

# calculating Kc max and min
df_Kc_summary150 = (
    df_150
    .groupby('prog_pos_x')['Kc']
    .agg(['min', 'max', 'mean'])
    .reset_index()
```

```
    .rename(columns={'min': 'Kc_min', 'max': 'Kc_max', 'mean' :'Kc_mean'})
)

print(df_Kc_summary150.iloc[::-1].round(3).to_markdown())
```

	prog_pos_x	Kc_min	Kc_max	Kc_mean
11	46	2543.4	3411.39	3002.74
10	44	625.758	5329.03	2965.81
9	42	1614.86	4239	2978.97
8	40	545.015	5429.96	3002.78
7	38	1796.53	3754.55	2993.47
6	36	60.557	4642.72	3012.65
5	34	1130.4	4965.69	3023.92
4	32	2301.17	3693.99	2995.96
3	30	746.872	3956.4	2985.67
2	28	1534.12	4461.05	2999.5
1	26	2725.07	3270.09	2990.55
0	24	343.157	4824.39	3013.26

1.4.5 Kc Calculation for N=1200

[ ]:
```
N = 1200  # rpm

# Vc calculation
df_1200['Vc_dia'] = np.pi * df_1200['prog_pos_x'] * N / 1000  # m/min

# Kc calculation
df_1200['Kc'] = df_1200['power_diff'] * 60 / (df_1200['Vc_dia'] * ap * f)

df_Kc_summary1200 = (
    df_1200
    .groupby('prog_pos_x')['Kc']
    .agg(['min', 'max', 'mean'])
    .reset_index()
    .rename(columns={'min': 'Kc_min', 'max': 'Kc_max', 'mean':'Kc_mean'})
)

print(df_Kc_summary1200.iloc[::-1].round(3).to_markdown())
```

	prog_pos_x	Kc_min	Kc_max	Kc_mean
0	24	736.234	4785.52	3205.95
1	26	895.838	4633.64	3175.26
2	28	1147.38	4474.77	3156.17
3	30	1311.84	4390.63	3141.39
4	32	1505.93	4392.31	3123.15
5	34	1629.95	4393.78	3107.79

6	36	1628.64	4305.85	3088.7
7	38	1817.69	4206.05	3058.7
8	40	1827.2	4136.3	3041.03
9	42	1854.93	3977.58	3019.11
10	44	1861.88	3869.79	3002.6
11	46	1920.61	3806.3	3003.87