●Spectral Analysis 3 libtfr by Python 3.6

#Spectral analysis of EEG during general anesthesia with sevoflurane

・・In this section, we will perform spectrum analysis by Python using EEG Analyzer and digital EEG data file eeg_bis.tsv acquired from a patient under general anesthesia of sevoflurane (during the transition phase from anesthesia to arousale). The influence of window functions, the application of the multitaper method, and the development into spectrograms will be shown.

EEG analysis during the transition from anesthesia to arousal

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.fftpack import fft
from scipy import signal as sig #Signal function import from SciPy package

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg')
```

・First, the EEG data from the digital EEG data file eeg_bis_10_20_2.tsv acquired from the BIS monitor through the EEG Analyzer are imported into the Pandas DataFrame df_tsv.

In [2]:
```
df_tsv = pd.read_table('../eeg_bis_20220629_sev_EME.tsv')
```

・Let's display the first five items of the imported data set.

In [3]:
```
df_tsv.head(5)
```

Out[3]:

	.Ch	Time	ch[0]	ch[1]	ch[2]	ch[3]	ch[4]	ch[5]	ch[6]	ch[7]	ch[8]	ch[9]	ch[10]	ch[11]	ch[12]	ch[13]	ch[14]	ch[15]
0	ch1:	12:15:17	6.70	3.80	1.75	-1.75	-1.95	1.10	5.15	10.30	15.70	19.65	20.60	17.55	12.90	12.75	18.00	23.80
1	ch1:	12:15:17	26.30	26.60	25.60	23.00	17.85	8.90	0.20	-0.95	4.10	7.10	7.00	6.20	7.05	10.10	13.05	13.15
2	ch1:	12:15:17	10.45	4.45	-2.90	-12.30	-24.95	-35.85	-38.35	-30.90	-17.70	-5.70	4.10	11.35	11.60	6.05	-4.35	-20.85
3	ch1:	12:15:17	-39.00	-52.55	-64.65	-76.25	-83.35	-82.55	-75.25	-64.40	-52.55	-42.90	-36.65	-32.25	-29.05	-24.50	-19.45	-20.40
4	ch1:	12:15:17	-27.60	-34.85	-38.25	-35.30	-26.20	-15.60	-7.65	-3.50	-2.55	-4.50	-7.75	-12.00	-20.40	-28.80	-30.85	-27.15

・Let's display the first five items of the imported data set.

In [4]:
```
df_tsv.tail(5)
```

Out[4]:

	.Ch	Time	ch[0]	ch[1]	ch[2]	ch[3]	ch[4]	ch[5]	ch[6]	ch[7]	ch[8]	ch[9]	ch[10]	ch[11]	ch[12]	ch[13]	ch[14]	ch[15]
4678	ch1:	12:25:18	5.65	-1.30	-5.40	-1.50	10.25	13.10	13.4	8.70	9.85	17.95	16.30	5.70	10.60	21.15	13.30	-5.60
4679	ch1:	12:25:18	6.65	20.25	17.00	10.60	12.55	10.20	20.5	18.85	12.50	22.15	33.80	22.90	14.50	12.85	21.85	22.60
4680	ch1:	12:25:18	17.05	14.30	15.10	16.15	29.15	29.00	31.9	36.95	32.90	28.40	36.15	34.20	36.05	48.30	41.95	39.25
4681	ch1:	12:25:18	53.75	55.75	49.75	58.85	65.15	69.65	70.7	72.40	82.45	87.25	85.80	93.45	95.95	91.35	89.50	94.80
4682	ch1:	12:25:18	105.20	113.00	104.95	99.35	108.80	111.40	109.3	112.30	111.55	108.90	111.80	113.35	107.90	110.50	114.30	111.75

・Create a new Pandas DataFrame df_eeg and set the column names as "Time" and "eeg".

In [5]:
```
cols = ['Time', 'eeg']
df_eeg = pd.DataFrame(data=None, index=[], columns=cols)
```

・Of the df_tsv (line 109611), copy the voltage µV data corresponding to the status under deep sevoflurane anesthesia (line 97,000-99,048) to df_eeg.

In [6]:
```
#for row in range(len(df_tsv)):
for row in range(0, 4682):
    # Import only 1024 lines of data. Since 8 lines / second, 128 seconds = 2 minutes 8 seconds
    #Number of data = 16,384
    #By playing with the range here, you can change which brain waves you can take out.

    t_tmp = df_tsv.iat[row,1]
    for column in range(2, 18):
            eeg_tmp = df_tsv.iat[row, column]
            data_tmp = pd.Series( [t_tmp, eeg_tmp], index=df_eeg.columns )
            df_eeg = df_eeg.append(data_tmp, ignore_index=True)
```

・Let's confirm and display 1,024 rows x 16 = 16,383 µV data (for 128 seconds) captured in df_eeg.

In [7]:
```
df_eeg
```

Out[7]:

	Time	eeg
0	12:15:17	6.70
1	12:15:17	3.80
2	12:15:17	1.75
3	12:15:17	-1.75
4	12:15:17	-1.95
...	...	...
74907	12:25:18	93.45
74908	12:25:18	95.95
74909	12:25:18	91.35

74910	12:25:18	89.50
74911	12:25:18	94.80

74912 rows × 2 columns

・Save the data imported to df_eeg as a csv file.

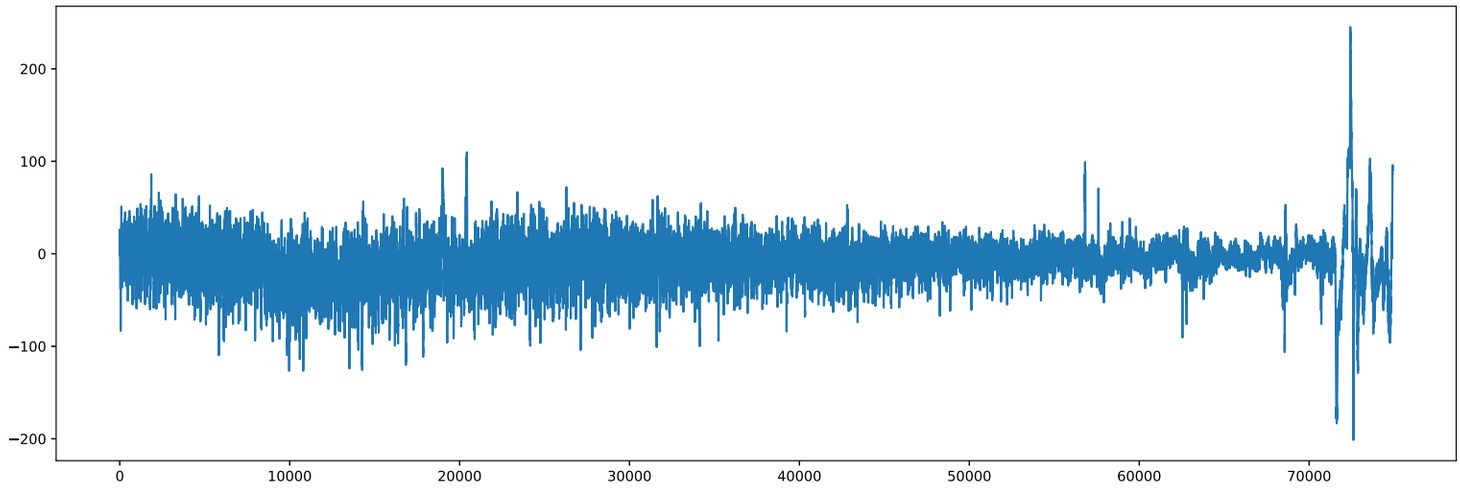
In [8]:
```
df_eeg.to_csv("data_eeg/eeg_linealized_04.csv")
```

・Transfer the df_eeg time and μV data to the new Python arrays Time and eeg, respectively.

In [9]:
```
#Place brain wave data in the data frame
time = df_eeg['Time'].values
eeg = df_eeg['eeg'].values
```

・Display the data captured in the array eeg using Matplotlib function plot().

In [10]:
```
#Time-series graph display of EEG data
fig, ax = plt.subplots(figsize=(18,6))
plt.plot(eeg)
plt.savefig('data_eeg/spectral_analysis_3_fig_1.svg')
```



Data preprocessing

・Let's calculate the mean and standard deviation of eeg.

In [11]:
```
m=np.mean(eeg)
sd=np.std(eeg)
m, sd
```

Out[11]:
```
(-9.047095925886374, 25.190010135621716)
```

・Count the number of null (NA) data.

In [12]:
```
df_eeg['eeg'].isnull().sum()
```

Out[12]:
```
0
```

・Outlier removal: Change the data that exceeds 3 times the standard deviation from the mean value to None.

In [13]:
```
for i in range(len(eeg)):
    if  df_eeg['eeg'][i] > m+3*sd:
      df_eeg['eeg'][i] = None
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [14]:
```
for i in range(len(eeg)):
    if  df_eeg['eeg'][i] < m-3*sd:
      df_eeg['eeg'][i] = None
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

・After removing the outliers, count the number of null (NA) data again.

In [15]:
```
df_eeg['eeg'].isnull().sum()
```

Out[15]:
```
1154
```

・Make a copy of df_eeg and perfrom spline interpolation.

In [16]:
```
df_eeg_copy = df_eeg.copy()
df_eeg_copy.interpolate(inplace=True)
```

· Count the number of null (NA) data in the copy of df_eeg.

In [17]:
```
df_eeg_copy['eeg'].isnull().sum()
```

Out[17]: 0

· Save the data with spline completion to a csv file.

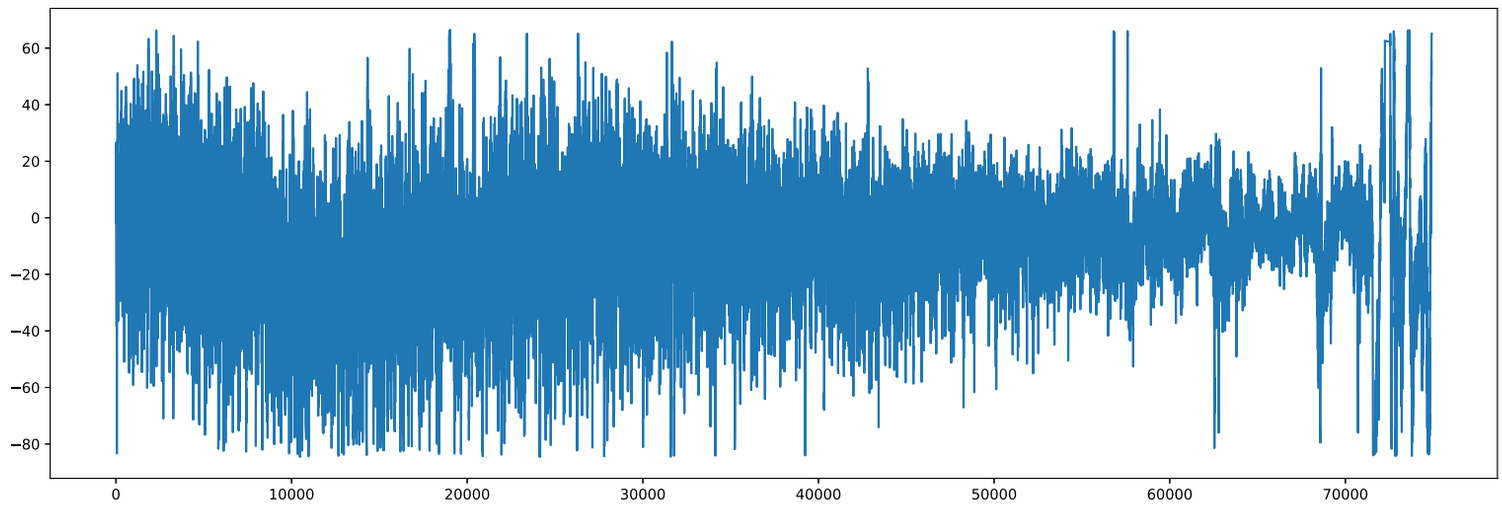
In [18]:
```
df_eeg.to_csv("data_eeg/eeg_linealized_04_ip.csv")
```

· Copy the eeg data to the array eeg_ip.

In [19]:
```
eeg_ip = df_eeg_copy['eeg'].values
```

· Plot and display eeg data.

In [20]:
```
#Time-series graph display of EEG data that complements and corrects missing values
fig, ax = plt.subplots(figsize=(18,6))
plt.plot(eeg_ip)
plt.savefig('data_eeg/spectral_analysis_3_fig_2.svg')
```

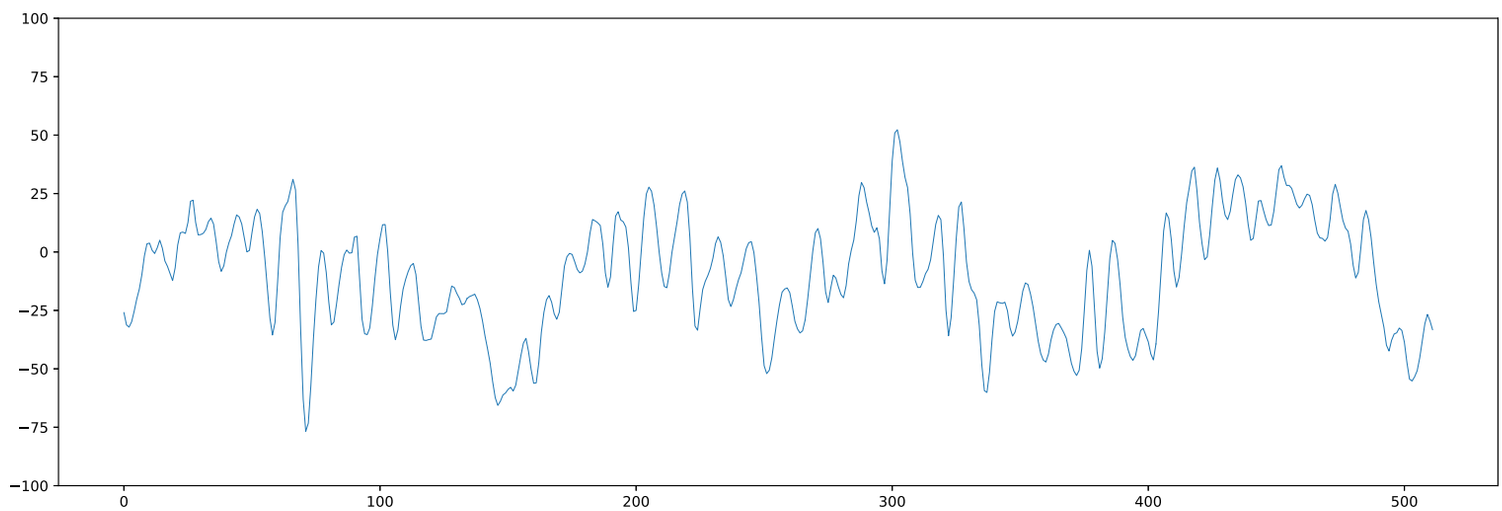


· From the eeg data, 512 cases (4 seconds, eeg_512) and 8,192 cases (64 seconds, eeg_8192) are extracted.

In [21]:
```
eeg_512 = eeg_ip[5000:5512]
time_512 = time[5000:5512]
eeg_8192 = eeg_ip[5000:13192]
time_8192 = time[5000:13192]
```

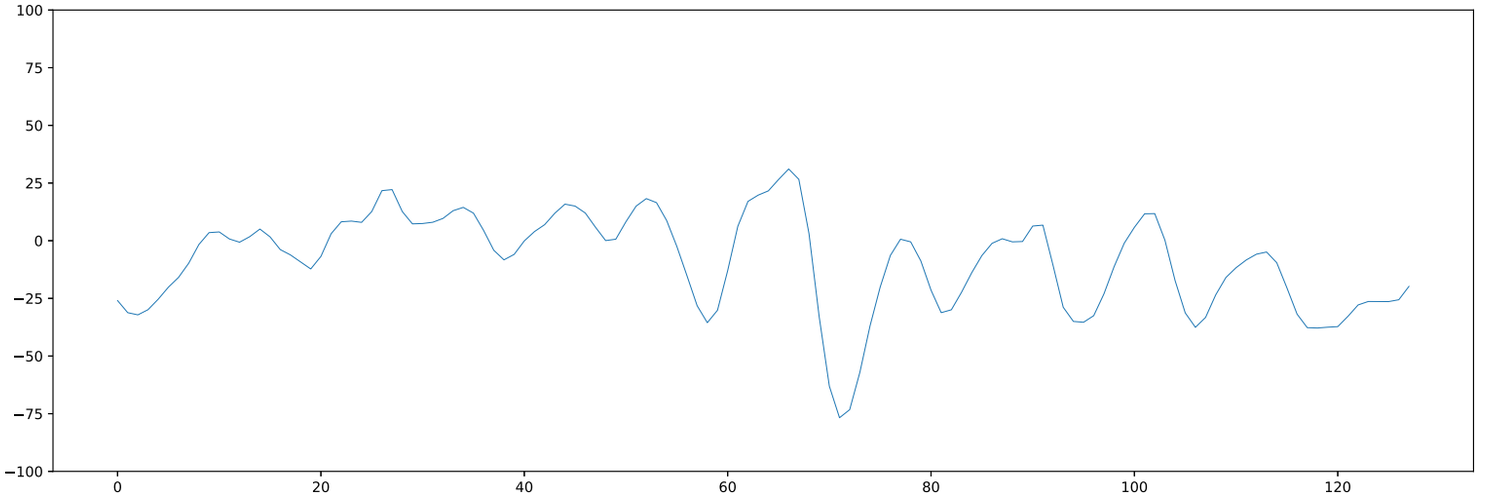
· Let's display of eeg_512

In [22]:
```
#Time-series graph display of EEG data that complements and corrects missing values
fig, ax = plt.subplots(figsize=(18,6))
plt.ylim(-100, 100)
plt.plot(eeg_512, linewidth =0.5)
plt.savefig('data_eeg/spectral_analysis_3_fig_3.svg')
```



· Display 128 samples for 1 second of eeg_512

In [23]:
```
#Time-series graph display of EEG data that complements and corrects missing values
fig, ax = plt.subplots(figsize=(18,6))
plt.ylim(-100, 100)
plt.plot(eeg_512[0:128], linewidth =0.5)
plt.savefig('data_eeg/spectral_analysis_3_fig_3_1.svg')
```



In [24]:

```
eeg_512
```

Out[24]:

```
array([-2.595e+01, -3.120e+01, -3.215e+01, -2.990e+01, -2.535e+01,
       -2.015e+01, -1.590e+01, -9.750e+00, -1.650e+00,  3.500e+00,
        3.800e+00,  7.500e-01, -6.500e-01,  1.750e+00,  5.050e+00,
        1.650e+00, -3.850e+00, -6.150e+00, -9.200e+00, -1.225e+01,
       -6.800e+00,  3.000e+00,  8.200e+00,  8.500e+00,  7.950e+00,
        1.265e+01,  2.170e+01,  2.215e+01,  1.265e+01,  7.300e+00,
        7.450e+00,  8.000e+00,  9.650e+00,  1.300e+01,  1.445e+01,
        1.195e+01,  4.450e+00, -4.150e+00, -8.300e+00, -5.950e+00,
       -1.000e-01,  3.950e+00,  6.950e+00,  1.190e+01,  1.585e+01,
        1.500e+01,  1.200e+01,  5.850e+00,  5.000e-02,  6.500e-01,
        8.250e+00,  1.500e+01,  1.825e+01,  1.650e+01,  8.750e+00,
       -2.450e+00, -1.520e+01, -2.820e+01, -3.555e+01, -3.015e+01,
       -1.300e+01,  6.250e+00,  1.705e+01,  1.975e+01,  2.160e+01,
        2.650e+01,  3.110e+01,  2.660e+01,  3.050e+00, -3.295e+01,
       -6.305e+01, -7.675e+01, -7.325e+01, -5.710e+01, -3.695e+01,
       -2.010e+01, -6.400e+00,  6.500e-01, -5.000e-01, -8.800e+00,
       -2.145e+01, -3.120e+01, -2.995e+01, -2.235e+01, -1.390e+01,
       -6.450e+00, -1.150e+00,  8.500e-01, -5.000e-01, -3.500e-01,
        6.350e+00,  6.750e+00, -1.070e+01, -2.885e+01, -3.500e+01,
       -3.535e+01, -3.250e+01, -2.310e+01, -1.130e+01, -1.100e+00,
        5.800e+00,  1.165e+01,  1.175e+01,  2.500e-01, -1.725e+01,
       -3.130e+01, -3.755e+01, -3.325e+01, -2.345e+01, -1.595e+01,
       -1.170e+01, -8.350e+00, -5.850e+00, -4.900e+00, -9.550e+00,
       -2.045e+01, -3.185e+01, -3.770e+01, -3.785e+01, -3.750e+01,
       -3.725e+01, -3.280e+01, -2.785e+01, -2.635e+01, -2.635e+01,
       -2.640e+01, -2.560e+01, -1.980e+01, -1.460e+01, -1.520e+01,
       -1.775e+01, -1.980e+01, -2.260e+01, -2.215e+01, -1.990e+01,
       -1.905e+01, -1.860e+01, -1.805e+01, -2.040e+01, -2.410e+01,
       -2.940e+01, -3.595e+01, -4.150e+01, -4.755e+01, -5.560e+01,
       -6.240e+01, -6.565e+01, -6.385e+01, -6.120e+01, -6.035e+01,
       -5.880e+01, -5.790e+01, -5.950e+01, -5.700e+01, -5.080e+01,
       -4.440e+01, -3.895e+01, -3.695e+01, -4.265e+01, -5.020e+01,
       -5.620e+01, -5.600e+01, -4.735e+01, -3.430e+01, -2.580e+01,
       -2.045e+01, -1.865e+01, -2.145e+01, -2.645e+01, -2.880e+01,
       -2.580e+01, -1.590e+01, -6.100e+00, -2.000e+00, -6.000e-01,
       -1.050e+00, -4.100e+00, -7.450e+00, -8.950e+00, -8.250e+00,
       -5.200e+00,  0.000e+00,  8.050e+00,  1.390e+01,  1.330e+01,
        1.250e+01,  1.135e+01,  3.400e+00, -9.000e+00, -1.505e+01,
       -1.065e+01,  3.200e+00,  1.540e+01,  1.720e+01,  1.370e+01,
        1.290e+01,  1.065e+01,  1.750e+00, -1.335e+01, -2.545e+01,
       -2.500e+01, -1.405e+01,  1.000e-01,  1.470e+01,  2.485e+01,
        2.775e+01,  2.600e+01,  2.010e+01,  1.070e+01, -5.000e-02,
       -9.000e+00, -1.475e+01, -1.530e+01, -9.050e+00, -2.000e-01,
        6.450e+00,  1.290e+01,  2.040e+01,  2.485e+01,  2.610e+01,
        2.135e+01,  6.200e+00, -1.570e+01, -3.170e+01, -3.350e+01,
       -2.435e+01, -1.610e+01, -1.255e+01, -1.010e+01, -7.100e+00,
       -2.650e+00,  3.600e+00,  6.450e+00,  4.200e+00, -1.550e+00,
       -1.065e+01, -2.035e+01, -2.335e+01, -2.040e+01, -1.580e+01,
       -1.185e+01, -8.850e+00, -3.650e+00,  1.400e+00,  4.000e+00,
        4.400e+00, -2.500e-01, -1.000e+01, -2.155e+01, -3.635e+01,
       -4.870e+01, -5.205e+01, -5.060e+01, -4.530e+01, -3.705e+01,
       -2.935e+01, -2.250e+01, -1.720e+01, -1.585e+01, -1.540e+01,
       -1.740e+01, -2.325e+01, -2.965e+01, -3.290e+01, -3.465e+01,
       -3.375e+01, -2.915e+01, -2.010e+01, -9.850e+00,  4.500e-01,
        8.250e+00,  1.005e+01,  5.650e+00, -4.500e+00, -1.710e+01,
       -2.165e+01, -1.545e+01, -9.950e+00, -1.115e+01, -1.495e+01,
       -1.820e+01, -1.960e+01, -1.445e+01, -5.050e+00,  6.500e-01,
        5.100e+00,  1.340e+01,  2.410e+01,  2.975e+01,  2.755e+01,
        2.140e+01,  1.665e+01,  1.115e+01,  8.400e+00,  1.035e+01,
        5.600e+00, -8.100e+00, -1.365e+01, -3.500e+00,  1.730e+01,
        3.890e+01,  5.090e+01,  5.230e+01,  4.720e+01,  3.885e+01,
        3.195e+01,  2.760e+01,  1.635e+01, -6.500e-01, -1.215e+01,
       -1.520e+01, -1.510e+01, -1.260e+01, -9.300e+00, -7.400e+00,
       -3.350e+00,  4.450e+00,  1.180e+01,  1.565e+01,  1.385e+01,
       -1.750e+00, -2.490e+01, -3.585e+01, -2.810e+01, -1.200e+01,
        5.700e+00,  1.920e+01,  2.140e+01,  1.025e+01, -4.250e+00,
       -1.265e+01, -1.615e+01, -1.760e+01, -2.050e+01, -3.145e+01,
       -4.835e+01, -5.935e+01, -6.010e+01, -5.145e+01, -3.725e+01,
       -2.545e+01, -2.135e+01, -2.175e+01, -2.200e+01, -2.150e+01,
       -2.510e+01, -3.240e+01, -3.600e+01, -3.425e+01, -2.975e+01,
       -2.330e+01, -1.670e+01, -1.325e+01, -1.390e+01, -1.795e+01,
       -2.355e+01, -3.075e+01, -3.810e+01, -4.345e+01, -4.625e+01,
       -4.710e+01, -4.370e+01, -3.765e+01, -3.340e+01, -3.105e+01,
       -3.055e+01, -3.245e+01, -3.450e+01, -3.695e+01, -4.250e+01,
       -4.770e+01, -5.105e+01, -5.285e+01, -5.065e+01, -4.140e+01,
       -2.525e+01, -7.850e+00,  7.500e-01, -6.050e+00, -2.455e+01,
       -4.255e+01, -4.970e+01, -4.570e+01, -3.425e+01, -1.850e+01,
       -2.600e+00,  4.950e+00,  3.650e+00, -3.100e+00, -1.410e+01,
       -2.780e+01, -3.660e+01, -4.140e+01, -4.480e+01, -4.635e+01,
       -4.440e+01, -3.915e+01, -3.360e+01, -3.270e+01, -3.555e+01,
       -3.845e+01, -4.365e+01, -4.620e+01, -3.935e+01, -2.545e+01,
       -7.850e+00,  9.000e+00,  1.670e+01,  1.435e+01,  5.050e+00,
       -8.150e+00, -1.500e+01, -1.100e+01, -9.000e-01,  1.090e+01,
        2.115e+01,  2.750e+01,  3.465e+01,  3.630e+01,  2.625e+01,
```

```
 1.270e+01,  3.500e+00, -3.250e+00, -2.100e+00,  7.600e+00,
 1.980e+01,  3.080e+01,  3.595e+01,  3.075e+01,  2.185e+01,
 1.580e+01,  1.385e+01,  1.735e+01,  2.460e+01,  3.095e+01,
 3.300e+01,  3.170e+01,  2.795e+01,  2.095e+01,  1.175e+01,
 5.000e+00,  5.800e+00,  1.395e+01,  2.180e+01,  2.200e+01,
 1.775e+01,  1.385e+01,  1.130e+01,  1.155e+01,  1.725e+01,
 2.625e+01,  3.520e+01,  3.690e+01,  3.195e+01,  2.845e+01,
 2.850e+01,  2.720e+01,  2.390e+01,  2.040e+01,  1.880e+01,
 2.000e+01,  2.270e+01,  2.475e+01,  2.420e+01,  2.025e+01,
 1.365e+01,  8.050e+00,  6.050e+00,  5.800e+00,  4.650e+00,
 6.150e+00,  1.405e+01,  2.480e+01,  2.895e+01,  2.525e+01,
 1.915e+01,  1.345e+01,  1.025e+01,  8.850e+00,  3.450e+00,
-5.700e+00, -1.115e+01, -8.750e+00,  1.800e+00,  1.380e+01,
 1.775e+01,  1.390e+01,  6.200e+00, -4.350e+00, -1.390e+01,
-2.135e+01, -2.675e+01, -3.215e+01, -3.990e+01, -4.240e+01,
-3.780e+01, -3.510e+01, -3.460e+01, -3.255e+01, -3.355e+01,
-3.860e+01, -4.735e+01, -5.440e+01, -5.520e+01, -5.340e+01,
-5.080e+01, -4.525e+01, -3.815e+01, -3.070e+01, -2.670e+01,
-2.950e+01, -3.325e+01])
```

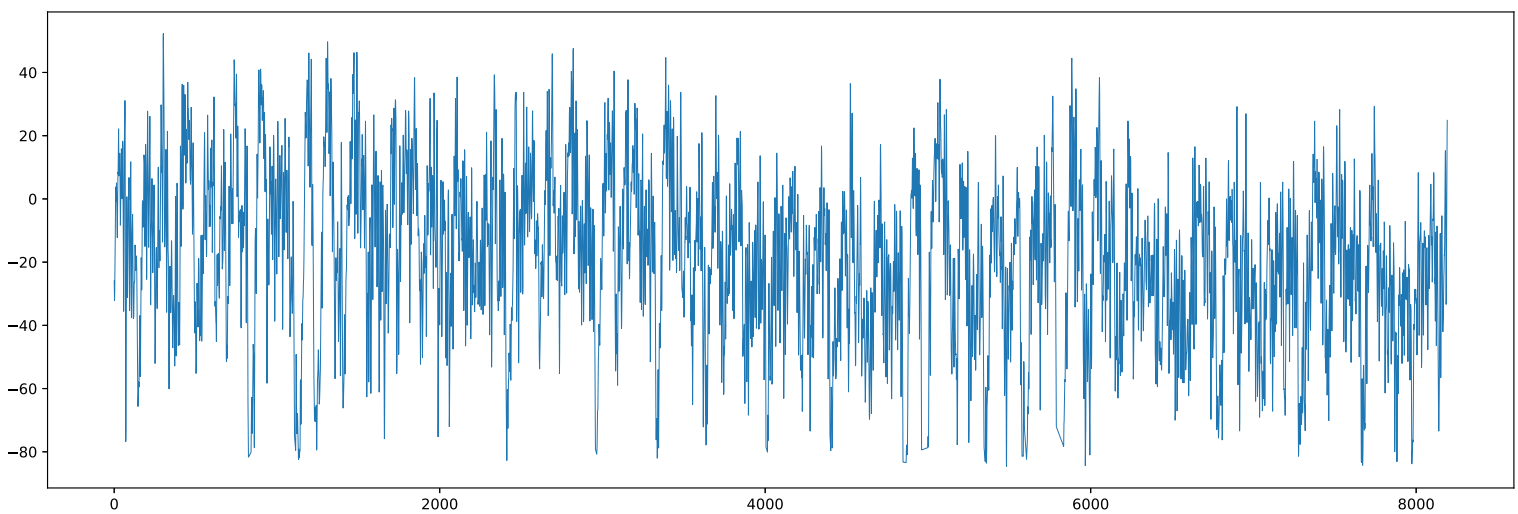
In [25]:

```
np.savetxt('data_eeg/eeg_512_2.csv', eeg_512, delimiter=',', fmt='%d')
```

· Let's display of eeg_8192 data

In [26]:

```
#Time-series graph display of EEG data that complements and corrects missing values
fig, ax = plt.subplots(figsize=(18,6))
plt.plot(eeg_8192, linewidth =0.5)
plt.savefig('data_eeg/spectral_analysis_3_fig_3_2.svg')
```



In [27]:

```
np.savetxt('data_eeg/eeg_8192_2.csv', eeg_8192, delimiter=',', fmt='%d')
```

· Set the parameters of the FFT function. Here, the sampling frequency is 128 Hz.

In [28]:

```
#Parameter set of FFT function
N=len(eeg_512) #Total number of data points
N2=len(eeg_8192) #Total number of data points
f_s = 128 #Sampling frequency
T = 1.0/f_s  #Data width (seconds)
s_rate = 1/f_s #Sampling rate
```

· Define a decibel conversion formula as a function

In [29]:

```
#dB conversion
def db(x, dBref):
    y = 20 * np.log10(x / dBref)      #Conversion formula
    return y                          #Returns a dB value
```

In [30]:

```
#Check the operation of the contents of the FFT function
1.0/(2.0*T), N/2
```

Out[30]:

```
(64.0, 256.0)
```

· Let's perform FFT of the eeg_512 (4 seconds) data with Scipy's fft () function.

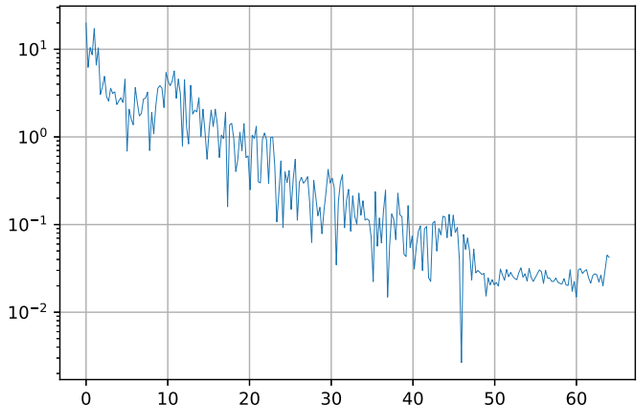
In [31]:

```
from scipy.fftpack import fft
#FFT transform with FFT function
xfft = np.linspace(0, int(1.0/(2.0*T)), int(N/2))
yfft=fft(eeg_512)
```

· Display the power spectrum with a simple FFT.

In [32]:

```
#Power spectrum display of each split EEG
plt.plot(xfft, 2.0/N*abs(yfft[0:int(N/2)]), linewidth =0.5)
plt.grid()
plt.yscale("log")
plt.savefig('data_eeg/spectral_analysis_3_fig_4.svg')
```

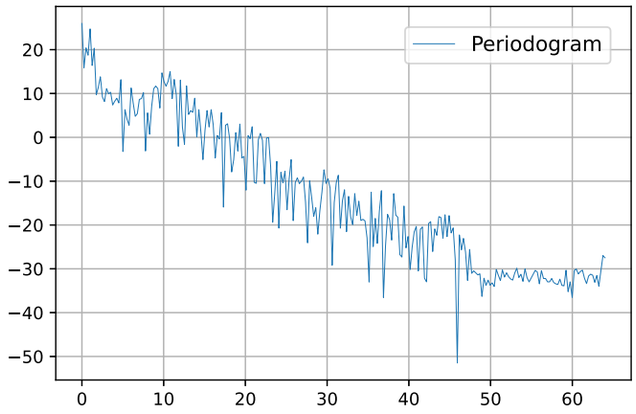


· Convert to decibels

In [33]:
```
dBref = 1
yfft_db = db(2.0/N*abs(yfft[0:int(N/2)]), dBref)
```

· Display SPD by simple FFT (periodogram) in decibel transform.

In [34]:
```
#Power spectrum display of each split EEG
plt.plot(xfft, yfft_db[0:int(N/2)], linewidth =0.5, label="Periodogram")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
#plt.yscale("log")
plt.grid()
plt.savefig('data_eeg/spectral_analysis_3_fig_4_1.svg')
```



· Create a Hann window using Scipy's Signal module.

In [35]:
```
#Prepare window function apply with scipy.
han = sig.hann(N)     #Creating the Hann window
```

In [36]:
```
acf_han = 1/(sum(han)/N) #Calculate Amplitude Correction Factor
print("acf_han:", acf_han)
```

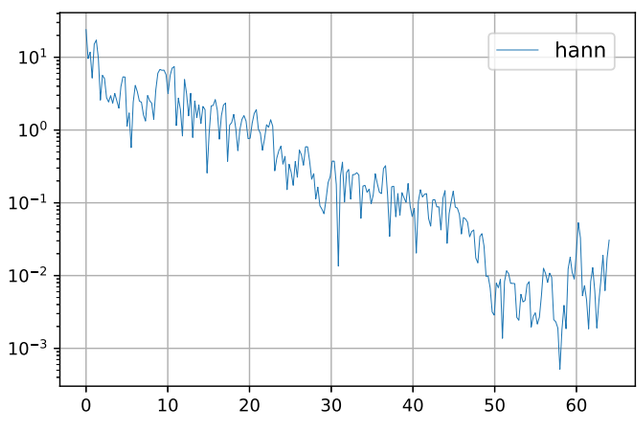
```
acf_han: 2.0039138943248536
```

· Adapt the Hann window to eeg_512 and perform FFT calculation.

In [37]:
```
#Window function correction
fft_data_acf_han = acf_han*np.abs(fft(eeg_512*han)/(N/2)) #data is time waveform, FFT, normalization, amplitude correction
```

· Display the PSD by the periodogram to which the Hann window function is applied.

In [38]:
```
#EEG power spectrum display
plt.plot(xfft, fft_data_acf_han[0:int(N/2)], linewidth =0.5, label="hann")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.yscale("log")
plt.grid()
plt.savefig('data_eeg/spectral_analysis_3_fig_5.svg')
```

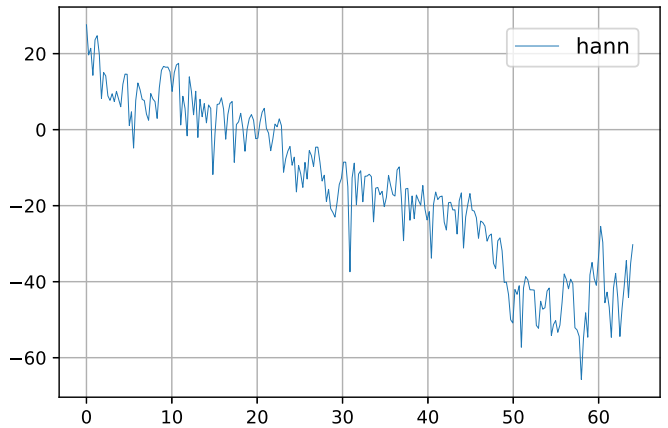


· Perform decibel conversion.

In [39]:
```
dBref = 1
yfft_acf_db_han = db(fft_data_acf_han[0:int(N/2)], dBref)
```

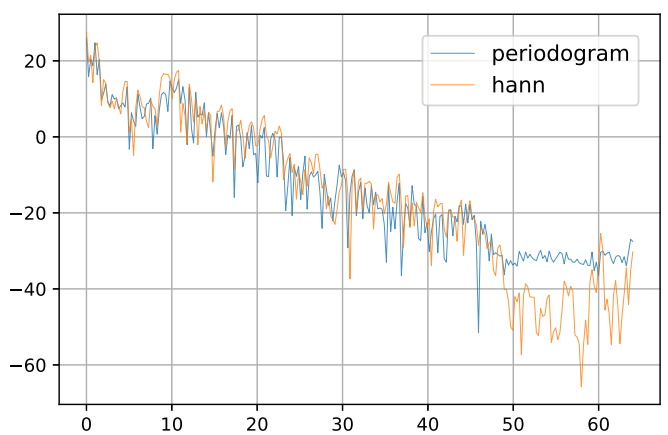
・Display a decibel PSD with a periodogram adapted to the Hann window function.

In [40]:
```
#Display SPD by simple FFT (periodogram) in decibel transform.
plt.plot(xfft, yfft_acf_db_han[0:int(N/2)], linewidth =0.5, label="hann")
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
#plt.yscale("log")
plt.grid()
plt.savefig('data_eeg/spectral_analysis_3_fig_6.svg')
```



・Compare PSD with simple periodogram and Hann window adaptation.

In [41]:
```
#EEG power spectrum display
plt.plot(xfft, yfft_db, linewidth =0.5, label="periodogram", alpha=0.8)
plt.plot(xfft, yfft_acf_db_han, linewidth =0.5, label="hann", alpha=0.8)
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
#plt.yscale("log")
plt.savefig('data_eeg/spectral_analysis_3_fig_7.svg')
```



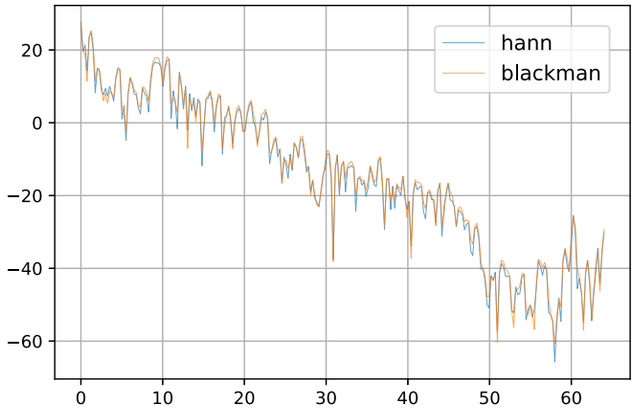
・Create Hann windows, Hamming windows, and Blackman windows.

In [42]:
```
#Prepare multiple window functions This time, with numpy.
window_n = np.hanning(N)
window_m = np.hamming(N)
window_b = np.blackman(N)

x = np.arange(0,N,1) #Horizontal axis
```

In [43]:
```
#Window function correction value
acf_bm=1/(sum(window_b)/N)
F_abs_amp_bm = acf_bm*np.abs(fft(eeg_512*window_b)/(N/2)) #Blackman window: OK if you multiply the value after FFT

dBref = 1
yfft_acf_db_bm = db(F_abs_amp_bm[0:int(N/2)], dBref)
```

・Display a graph comparing the PSD with the Blackman window function applied to the Hann window.

In [44]:
```
#Power spectrum display of each split EEG
plt.plot(xfft, yfft_acf_db_han, linewidth =0.5, label="hann", alpha=0.7)
plt.plot(xfft, yfft_acf_db_bm, linewidth =0.5, label="blackman", alpha=0.7)
plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)

plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=1, fontsize=12)
plt.grid()
#plt.yscale("log")
plt.savefig('data_eeg/spectral_analysis_3_fig_8.svg')
```



Practice of power spectrum of multiple analysis segments of EEG

・Let's allow you to select window functions. Here, let's choose the Hann window. The analysis data window consists of 512 samples, and the windows overlap by 50%.

In [45]:

```
dt = 1/128   #Sampling rate
split = 512/N2  #Overall data ratio of analysis frame data
overlap = 0.5 #Overlap rate

window = "hanning" #Window function selection: Hann, Hamming, Blackman
#窓の用意
if window == "hanning":
    window_select = "hanning"         # Hann window
    print("Select Hanning")
elif window == "hamming":
    window_select = "hamming"          # Hamming window
    print("Select Hamming")
elif window == "blackman":
    window_select = "blackman"         # Blackman window
    print("Select Blackman")
else:
    window_select = "hanning"          # Select Hann window
    print("Select Hanning")

#fq_ave, F_abs_amp_ave, time, freq, power = FFT_main(time, eeg, dt, split, overlap, window_select)
```

```
Select Hanning
```

・Define a function data_split () that allows the EEG record to be split and analyzed for each analysis segment.

In [46]:

```
#A function that splits data into FFT analysis widths and returns the split data as an array.

def data_split(_t, _x, _split, _overlap):

    split_data = []
    one_frame_cnt = int(len(_t)*_split) #Number of samples per frame
    overlap_cnt = int(one_frame_cnt*_overlap) #Number of overlapping samples
    print("Number of frame samples:  ", one_frame_cnt)
    print("Number of overlapping samples: ", overlap_cnt)

    start_S = 0
    end_S = start_S + one_frame_cnt

    while True:
        t_cnt = _t[start_S:end_S]
        x_cnt = _x[start_S:end_S]
        split_data.append([t_cnt, x_cnt])

        start_S = start_S + (one_frame_cnt - overlap_cnt)
        end_S = start_S + one_frame_cnt

        if end_S > len(_t):
            break

    return np.array(split_data)
```

・Define the FFT function.

In [47]:

```
#FFT function
def FFT(_data_input, _dt, _window_select):

    data_cnt_adj = len(_data_input[0])

    #Window function processing
    if _window_select == "hanning":
        window_select = np.hanning(data_cnt_adj)    #Hanning
    elif _window_select == "hamming":
        window_select = np.hamming(data_cnt_adj)    #Hamming
    elif _window_select == "blackman":
        window_select = np.blackman(data_cnt_adj)   #Blackman
    else:
        window_select  = np.hanning(data_cnt_adj)   #Hanning

    #Signal after window function
    x_windowed = _data_input[1]*window_select

    #FFT calculation
    F = np.fft.fft(x_windowed)
    F_abs = np.abs(F)
    F_abs_amp = F_abs / data_cnt_adj * 2
    fq = np.linspace(0, 1.0/dt, data_cnt_adj)

    #Window function correction
    acf=1/(sum(window_select)/data_cnt_adj)
    F_abs_amp = acf*F_abs_amp
```

```
    #Extract up to Nyquist constant
    fq_out = fq[:int(data_cnt_adj/2)+1]
    F_abs_amp_out = F_abs_amp[:int(data_cnt_adj/2)+1]

    return [fq_out, F_abs_amp_out]
```

- Define a function plot_TTF that plots FFT results.

In [48]:

```
#Graph display of FFT results (used to display average value)
def plot_FFT(_t, _x, _fq, _F_abs_amp):
    fig = plt.figure(figsize=(16, 6))
    ax2 = fig.add_subplot(121)
    plt.title("time")
    plt.xlabel("time [s]")
    plt.ylabel("amplitude"+"[V]")
    plt.plot(_t, _x, linewidth =0.5)

    ax2 = fig.add_subplot(122)
    plt.title("freq")
    plt.xlabel('freqency(Hz)')
    plt.ylabel("amplitude"+"[V/rtHz]")
    #plt.xscale("log")
    plt.yscale("log")
    plt.plot(_fq, _F_abs_amp, linewidth =0.5)
```

- Define a function plot_split_eeg () that displays split brain waves.

In [49]:

```
#Depicts split brain waves
def plot_split_eeg(_split_data, _FFT_result_list, _frame_cnt, _FFT_result_cnt):

    #Split EEG data
    fig3 = plt.figure(figsize=(24, 8))
    plt.title("time")
    plt.xlabel("time [s]")
    plt.ylabel("amplitude"+"[V]")
    loc = 0
    for _split_data_cnt, _FFT_result_cnt in zip(_split_data, _FFT_result_list):
        plt.xscale("linear")
        plt.yscale("linear")
        ax3 = fig3.add_subplot(_frame_cnt, 1, 1+loc)
        plt.plot(_split_data_cnt[0], _split_data_cnt[1])
        loc +=1
    plt.savefig('data_eeg/spectral_analysis_3_fig_9.svg')
```

- Define a function plot_FFT_all() that displays all the split brain waves together.

In [50]:

```
#dB conversion
def db(x, dBref):
```

```
def db(x, dBref):
    y = 20 * np.log10(x / dBref)      #Conversion formula
    return y                          #Returns a dB value

#Individual split EEG data
def plot_FFT_all(_split_data, _FFT_result_list, _frame_cnt, _FFT_result_cnt, _overlap_cnt, _dBref):

    #Spectrum of individual split data
    fig4 = plt.figure(figsize=(12, 4))
    ax4 = fig4.add_subplot(122)
    plt.title("freq")
    plt.xlabel('freqency(Hz)')
    plt.ylabel("amplitude"+"[V/rtHz]")
    #plt.xscale("log")
    plt.yscale("log")
    noc=1

    for _split_data_cnt, _FFT_result_cnt in zip(_split_data, _FFT_result_list):
        plt.plot(_FFT_result_cnt[0], _FFT_result_cnt[1], label=noc, linewidth =0.5)
        plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0, fontsize=8)
        noc +=1

    #DB-converted spectrum of individual split data
    fig5 = plt.figure(figsize=(12, 4))
    ax5 = fig5.add_subplot(122)
    plt.title("freq")
    plt.xlabel('freqency(Hz)')
    plt.ylabel("amplitude"+"[dB/Hz]")
    #plt.xscale("log")
    #plt.yscale("log")
    noc=1

    for _split_data_cnt, _FFT_result_cnt in zip(_split_data, _FFT_result_list):
        plt.plot(_FFT_result_cnt[0], db(_FFT_result_cnt[1], _dBref), label=noc, linewidth =0.5)

        plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0, fontsize=8)
        noc +=1
```

· Define a function set_3D_data that arranges data three-dimensionally.

In [51]:

```
#3D data placement
def set_3D_data(_split_data, _FFT_result_list, _frame_cnt, _FFT_result_cnt, _overlap_cnt):

    #Three-dimensional data: T = time, Y = frequency, z = power are stored in a list format array.
    #Data placement in a time axis array
    T_pre= np.arange(0, 64, 64/_frame_cnt)
    T=[]

    for i in T_pre:
        T.append([i] *(1+_overlap_cnt))

    print("Number of time array 2D data:", len(T))
    print("Number of time array 1D data:", len(T[0]))
    #print(T)

    #Arrangement of data in the frequency axis array
    Y_pre= _FFT_result_cnt[0].tolist()
    Y=[]

    for i in T_pre:
        Y.append(Y_pre)

    #print(len(Y_pre))
    print("Frequency array 2D number of data:", len(Y))
    print("Frequency array 1-dimensional number of data:", len(Y[0]))
    #print(Y)

    #Arrangement of data in the frequency power axis array
    Z=[]
    for split_data_cnt, FFT_result_cnt in zip(_split_data, _FFT_result_list):
        Z_pre=10*np.log10(FFT_result_cnt[1])
        Z.append(Z_pre.tolist())

    print("Frequency power array 2D number of data:", len(Z))
    print("Frequency power array 1D number of data:", len(Z[0]))
    #print(Z)

    #Convert dimensional data to numpy array
    T=np.array(T)
    Y=np.array(Y)
    Z=np.array(Z)

    return T, Y, Z
```

· Set the time axis array of 8,192 data for 64 seconds.

In [52]:

```
Time_adj= np.linspace(0, 4, N2)
```

· The SPD calculation process is performed for the data of eeg_8192 (64 seconds). Here it is divided into 31 segments.

In [53]:

```
#Overlapping division of EEG data.
split_data = data_split(Time_adj, eeg_8192, split, overlap)

frame_cnt =len(split_data)
print("Total number of frames: ", frame_cnt)

#Perform FFT calculation.
FFT_result_list = []

for split_data_cnt in split_data:
    FFT_result_cnt = FFT(split_data_cnt, dt, window)
    FFT_result_list.append(FFT_result_cnt)

one_frame_cnt = int(len(time_8192)*split)
overlap_cnt = int(one_frame_cnt*overlap) #Number of overlapping samples
```

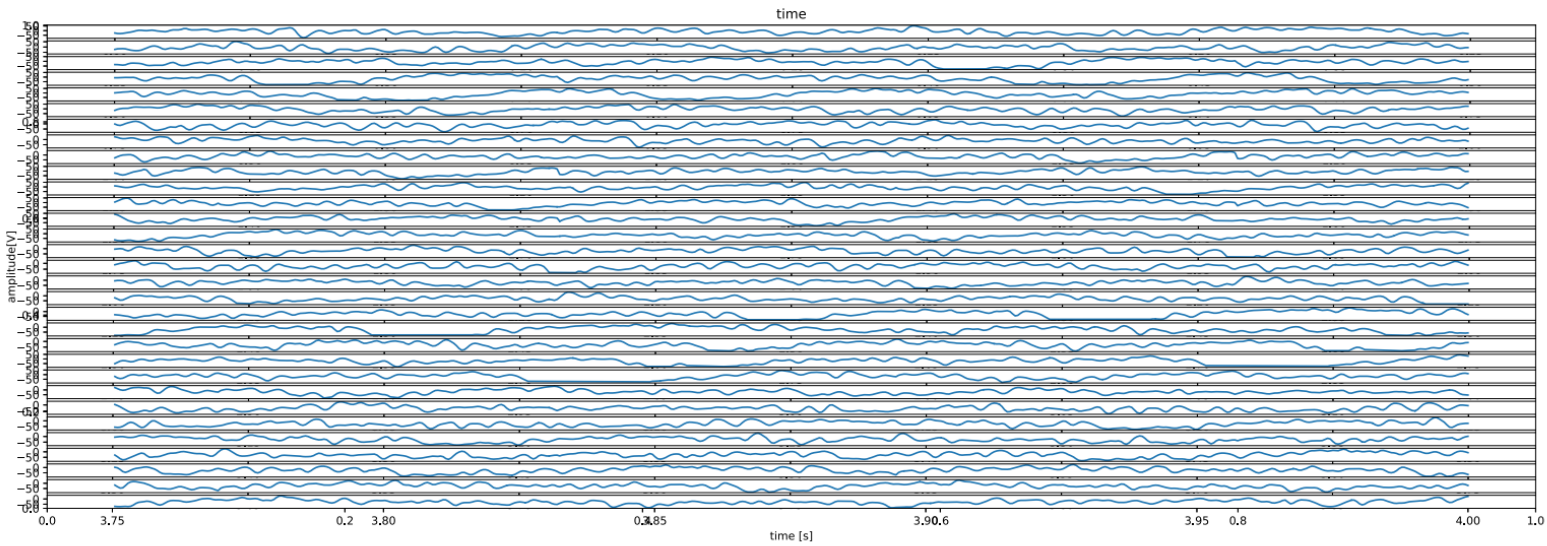
```
#print("FFT_result_list[0]: ", FFT_result_list[0])
#print("FFT_result_list[1]: ", FFT_result_list[1])

#Power averaging
fq_ave = FFT_result_list[0][0]
F_abs_amp_ave = np.zeros(len(fq_ave))

for i in range(len(FFT_result_list)):
    F_abs_amp_ave = F_abs_amp_ave + FFT_result_list[i][1]
F_abs_amp_ave = F_abs_amp_ave/(i+1)
```

```
Number of frame samples:   512
Number of overlapping samples:  256
Total number of frames:  31
```

· Display segmented EEG data.

In [54]:

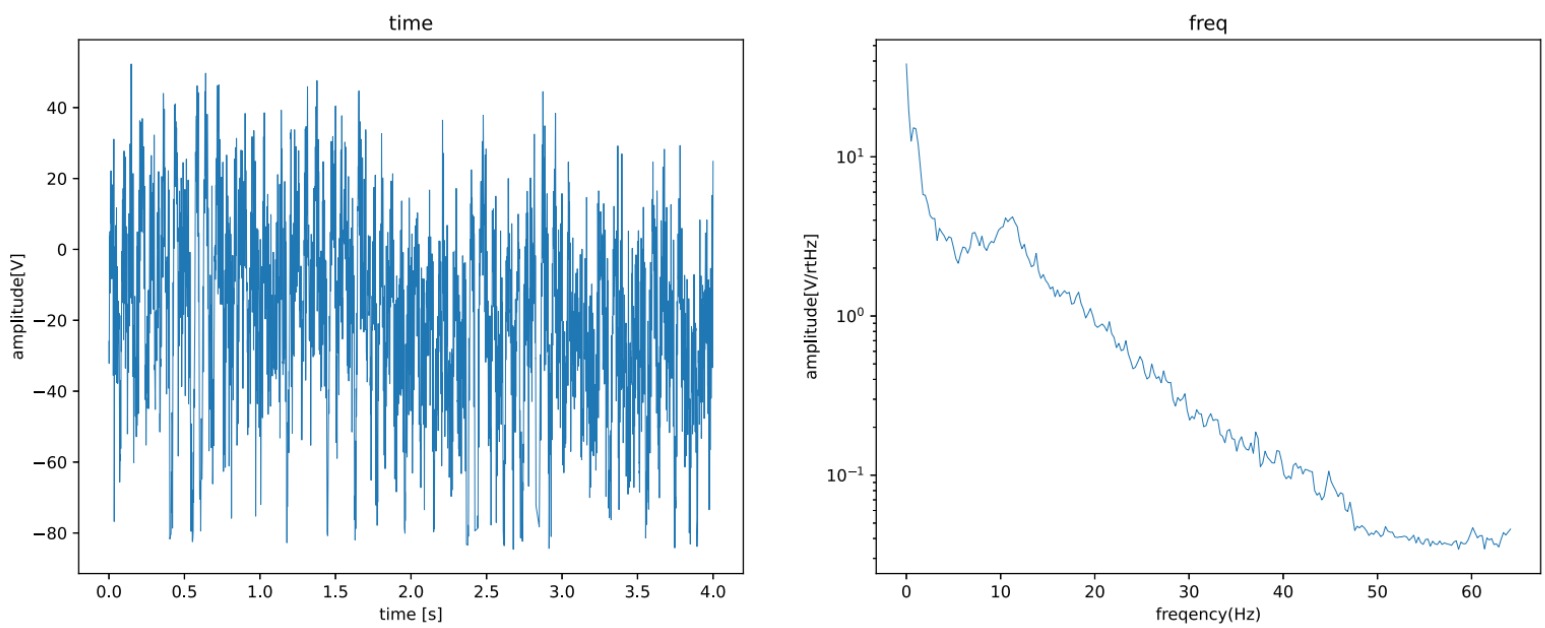
```
#Drawing of split brain waves
plot_split_eeg(split_data, FFT_result_list, frame_cnt, FFT_result_cnt)
```



· Shows the overall averaged amplitude and spectrum.

In [55]:

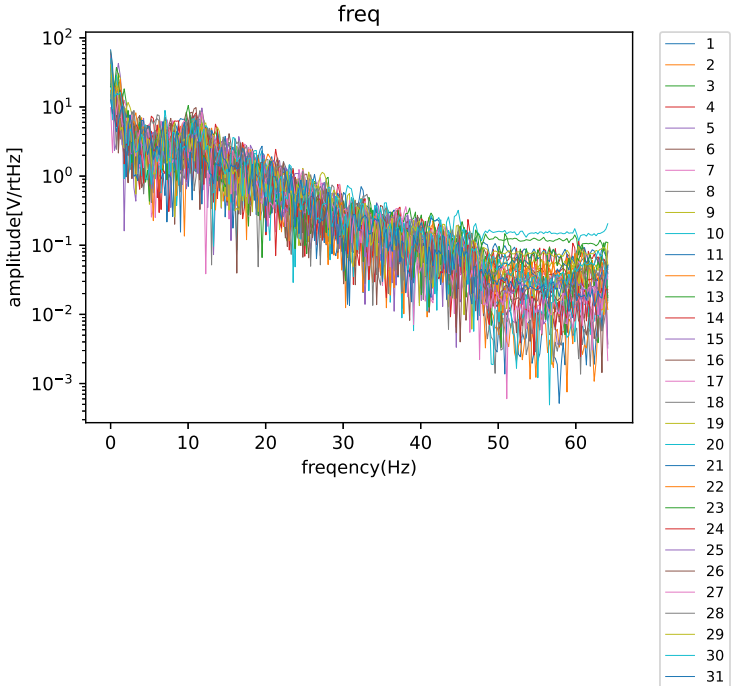
```
#Averaged spectrum
plot_FFT(Time_adj, eeg_8192, fq_ave, F_abs_amp_ave)
plt.savefig('data_eeg/spectral_analysis_3_fig_10.svg')
```

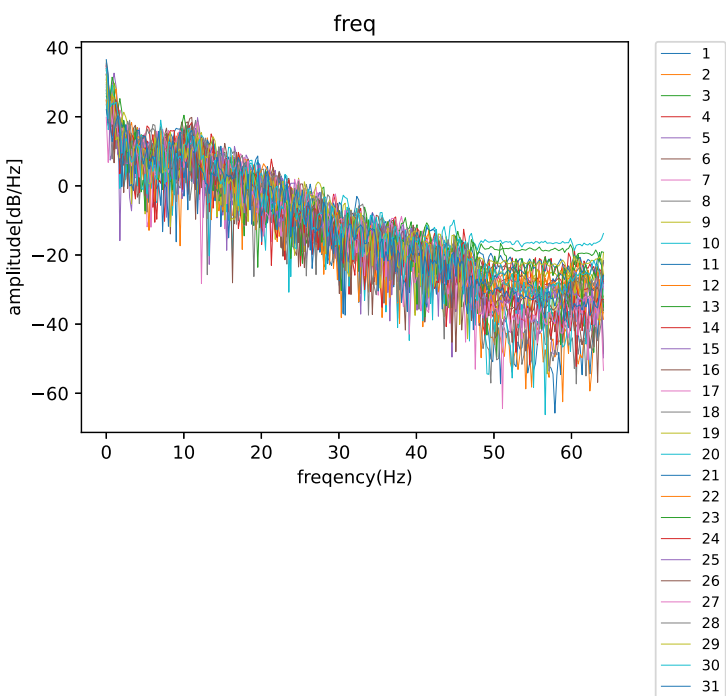


· The SPDs analyzed by segmenting are displayed together on the one graph window.

In [56]:

```
#Spectrum of individual split data
dBref=1
plot_FFT_all(split_data, FFT_result_list, frame_cnt, FFT_result_cnt, overlap_cnt, dBref)
plt.savefig('data_eeg/spectral_analysis_3_fig_11.svg')
```





Constructing a spectrogram

In [57]:

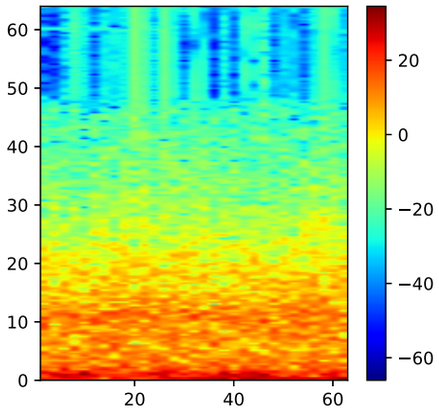
```
#Set of 3D data: 2D array of time, frequency, frequency power: The number of data in each dimension must match.
time2, freq, power = set_3D_data(split_data, FFT_result_list, frame_cnt, FFT_result_cnt, overlap_cnt)
```

```
Number of time array 2D data: 31
Number of time array 1D data: 257
Frequency array 2D number of data: 31
Frequency array 1-dimensional number of data: 257
Frequency power array 2D number of data: 31
Frequency power array 1D number of data: 257
```

- Spectrogram construction from a simple periodogram

In [58]:

```
nFFT = 512        # the length of the windowing segments
f_s =128  # the sampling frequency
noverlap=nFFT*0.5

# Pxx: the segments x freqs array of instantaneous power,
# freqs: frequency vector
# bins: the centers of the time bins in which the power is computed
# im: the matplotlib.image.AxesImage instance

window = np.blackman(nFFT)
plt.figure(figsize=(4, 4))
Pxx, freqs, bins, im = plt.specgram(eeg_8192, NFFT=nFFT, Fs=f_s, scale='dB', window=window, noverlap=noverlap, cmap='jet')
plt.colorbar()
plt.savefig('data_eeg/spectral_analysis_3_fig_12.svg')
```



· Introduce libtfr library, librosa library, etc. for more advanced spectrogram construction.

· Define parameters for spectrogram calculation using libtfr.

In [59]:
```
import libtfr
from librosa import display
from scipy.signal import chirp
import librosa
```

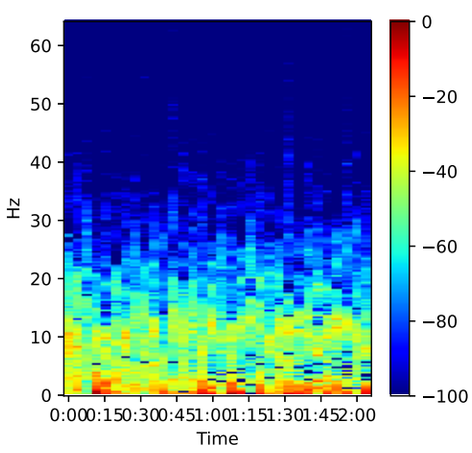
· Define parameters for spectrogram calculation using libtfr.

In [60]:
```
#Define parameters for spectrogram calculation using libtfr.
# libtfr

nfft = 512  # analysis segment
f_s = 128   # sampling frequency
shift = nfft/2  #overlap data points
K = 6          # number of tapers to use (default 6)
tm = 6.0    # time support of tapers (default 6.0)
flock = 1   # frequency locking parameter; power is not reassigned
                # more than this value (normalized frequency; default 0.01)
tlock = 5   # time locking parameter (in frames; default 5)

# load signal of dimension (npoints,)
S = libtfr.tfr_spec(eeg_8192, nfft, shift, f_s, K, tm, flock, tlock)
print(np.shape(S), np.max(S), np.min(S))

S2 = librosa.amplitude_to_db(S, ref=np.max, top_db =100)
print(np.shape(S2), np.max(S2), np.min(S2))
```

```
(257, 32) 1040081.6523023543 0.0
(257, 32) 0.0 -100.0
```

· SPD spectrogram construction by frequency-time reallocation method using libtfr's tfr_spec() function

In [61]:
```
fig, ax = plt.subplots(figsize=(4,4))
display.specshow(S2, x_axis='time', y_axis='hz', cmap='jet', sr=f_s, hop_length=nfft)
plt.colorbar()
plt.savefig('data_eeg/spectral_analysis_3_fig_14.svg')
```



In [62]:
```
list=[]
for i in range(0, 256):
    list.append(np.mean(S2[i]))
```

In [63]:
```
Fq= np.linspace(0, 64, 256)
```

· Create a spectrogram using libtfr's multi-taper method mfft_dpss().

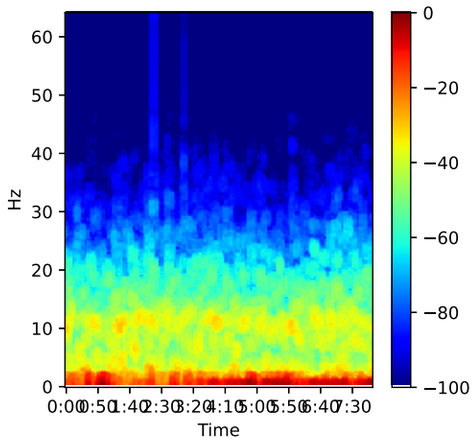
In [64]:
```
# generate a transform object with size 512 samples and 5 tapers for short-time analysis
D = libtfr.mfft_dpss(512, 12, 5)
# complex STFT with frame shift of 10 samples
Z = D.mtstft(eeg_8192, 64)
# spectrogram with frame shift of 10 samples
P = D.mtspec(eeg_8192, 64)
```

In [65]:
```
print(np.shape(Z), np.max(Z), np.min(Z))
print(np.shape(P), np.max(P), np.min(P))
```

```
(257, 121, 5) (485.5979889556745+41.1093200344951j) (-608.296678781169+0j)
(257, 121) 208056.85558300026 0.0003305215220193417
```

In [66]:

```
P = librosa.amplitude_to_db(P, ref=np.max, top_db =100)

fig, ax = plt.subplots(figsize=(4,4))
display.specshow(P, x_axis='time',  y_axis='hz', cmap='jet', sr=f_s, hop_length = nfft)
plt.colorbar()
plt.savefig('data_eeg/spectral_analysis_3_fig_18.svg')
```



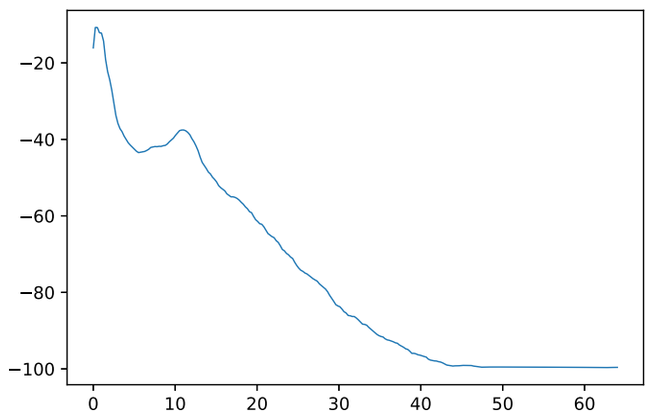
· Display the SPD of the first segment by the multitaper method.

In [67]:

```
list3=[]
for i in range(0, 256):
    list3.append(np.mean(P[i]))
```

In [68]:

```
plt.plot(Fq, list3, linewidth =0.8)
plt.savefig('data_eeg/spectral_analysis_3_fig_19.svg')
```



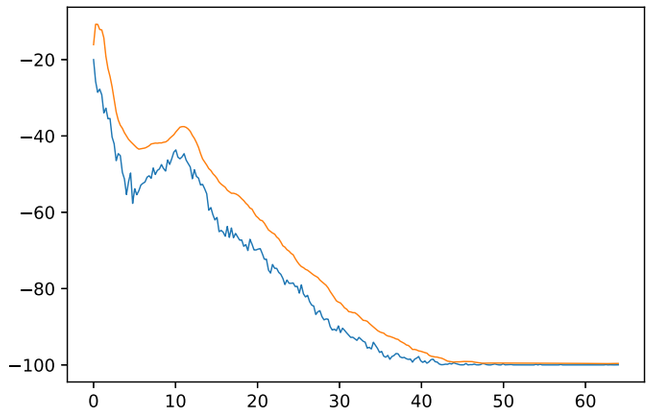
· For the PSD of the first segment, compare two of libtfr's two methods (time reallocation method and multitaper method).

In [69]:

```
fig = plt.figure()
plt.plot(Fq, list, linewidth =0.8)
#plt.plot(Fq, list2)
plt.plot(Fq, list3, linewidth =0.8)
plt.savefig('data_eeg/data_eeg/spectral_analysis_3_fig_20.svg')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-69-0f11b50bab4d> in <module>
      3 #plt.plot(Fq, list2)
      4 plt.plot(Fq, list3, linewidth =0.8)
----> 5 plt.savefig('data_eeg/data_eeg/spectral_analysis_3_fig_20.svg')

~/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/matplotlib/pyplot.py in savefig(*args, **kwargs)
    857 def savefig(*args, **kwargs):
    858     fig = gcf()
--> 859     res = fig.savefig(*args, **kwargs)
    860     fig.canvas.draw_idle()   # need this if 'transparent=True' to reset colors
    861     return res

~/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/matplotlib/figure.py in savefig(self, fname, transparent, **kwargs)
   2309                 patch.set_edgecolor('none')
   2310
-> 2311         self.canvas.print_figure(fname, **kwargs)
   2312
   2313         if transparent:

~/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/matplotlib/backend_bases.py in print_figure(self, filename, dpi, facecolor, edgecolo
r, orientation, format, bbox_inches, pad_inches, bbox_extra_artists, backend, **kwargs)
   2215                     orientation=orientation,
   2216                     bbox_inches_restore=_bbox_inches_restore,
-> 2217                     **kwargs)
   2218             finally:
   2219                 if bbox_inches and restore_bbox:

~/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/matplotlib/backends/backend_svg.py in print_svg(self, filename, *args, **kwargs)
   1315             __ DC_
   1316         """
-> 1317         with cbook.open_file_cm(filename, "w", encoding="utf-8") as fh:
   1318
   1319             filename = getattr(fh, 'name', '')
```

```
~/opt/anaconda3/envs/Python36_2/lib/python3.6/contextlib.py in __enter__(self)
     79     def __enter__(self):
     80         try:
---> 81             return next(self.gen)
     82         except StopIteration:
     83             raise RuntimeError("generator didn't yield") from None

~/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/matplotlib/cbook/__init__.py in open_file_cm(path_or_file, mode, encoding)
    421 def open_file_cm(path_or_file, mode="r", encoding=None):
    422     r"""Pass through file objects and context-manage path-likes."""
--> 423     fh, opened = to_filehandle(path_or_file, mode, True, encoding)
    424     if opened:
    425         with fh:

~/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/matplotlib/cbook/__init__.py in to_filehandle(fname, flag, return_opened, encoding)
    406             fh = bz2.BZ2File(fname, flag)
    407         else:
--> 408             fh = open(fname, flag, encoding=encoding)
    409         opened = True
    410     elif hasattr(fname, 'seek'):

FileNotFoundError: [Errno 2] No such file or directory: 'data_eeg/data_eeg/spectral_analysis_3_fig_20.svg'
```



Option Spectrogram

- Spectrogram construction with SPD from a simple periodogram

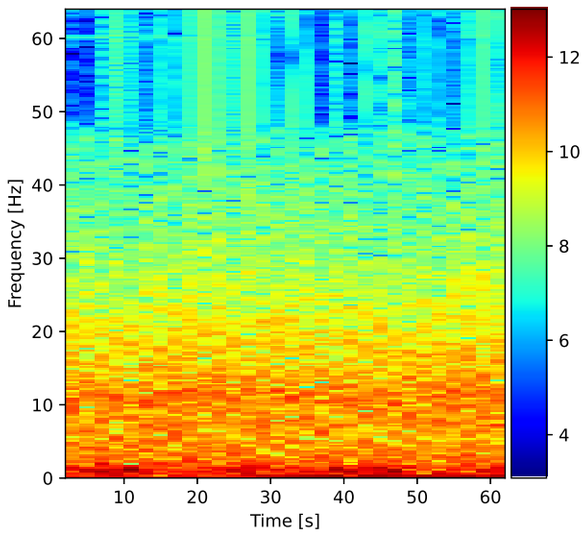
In [70]:
```
###option spectrogram
nFFT = 512        # the length of the windowing segments
f_s =128  # the sampling frequency
noverlap=nFFT*0.5

freqs, times, Sxx = sig.spectrogram(eeg_8192, fs=f_s, window='hanning', nperseg=512, noverlap=noverlap, detrend=False, scaling='spectrum')
```

In [71]:
```
###option spectrogram
len(Sxx), len(Sxx[0])
```

Out[71]: (257, 31)

In [72]:
```
###option spectrogram
from mpl_toolkits.axes_grid1 import make_axes_locatable
fig, ax = plt.subplots(figsize=(5, 6))
image = ax.pcolormesh(times, freqs, 10+ np.log10(Sxx), cmap='jet')
ax.axis("image")
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [s]')
divider = make_axes_locatable(ax)
ax_cb = divider.new_horizontal(size="8%", pad=0.05)
fig.add_axes(ax_cb)
plt.colorbar(image, cax=ax_cb)
plt.savefig('data_eeg/spectral_analysis_3_fig_21.svg')
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:4: MatplotlibDeprecationWarning: shading='flat' w
hen X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shadi
ng='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  after removing the cwd from sys.path.
```



- SPD spectrogram construction by frequency-time reallocation method using libtfr's tfr_spec() function

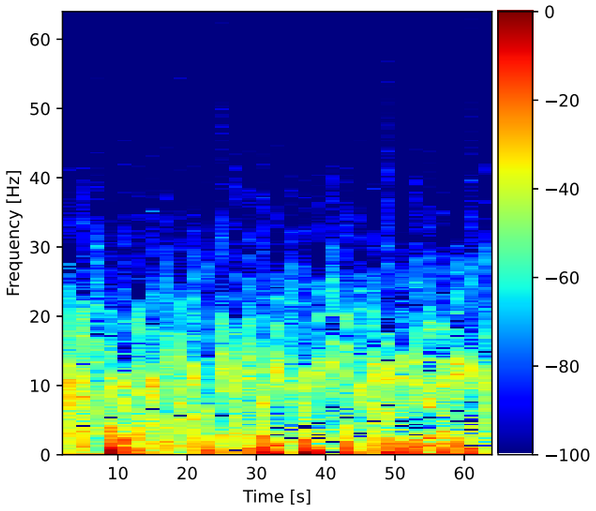
In [73]:
```
###option spectrogram
times_adj = np.append(times, 64)
```

In [74]:
```
###option spectrogram
times_adj
```

Out[74]:
```
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20., 22., 24., 26.,
       28., 30., 32., 34., 36., 38., 40., 42., 44., 46., 48., 50., 52.,
       54., 56., 58., 60., 62., 64.])
```

In [75]:
```
###option spectrogram
fig, ax = plt.subplots(figsize=(5, 6))
image = ax.pcolormesh(times_adj, freqs, S2, cmap='jet')
ax.axis("image")
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [s]')
divider = make_axes_locatable(ax)
ax_cb = divider.new_horizontal(size="8%", pad=0.05)
fig.add_axes(ax_cb)
plt.colorbar(image, cax=ax_cb)
plt.savefig('data_eeg/spectral_analysis_3_fig_22.svg')
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:3: MatplotlibDeprecationWarning: shading='flat' w
hen X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shadi
ng='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  This is separate from the ipykernel package so we can avoid doing imports until
```



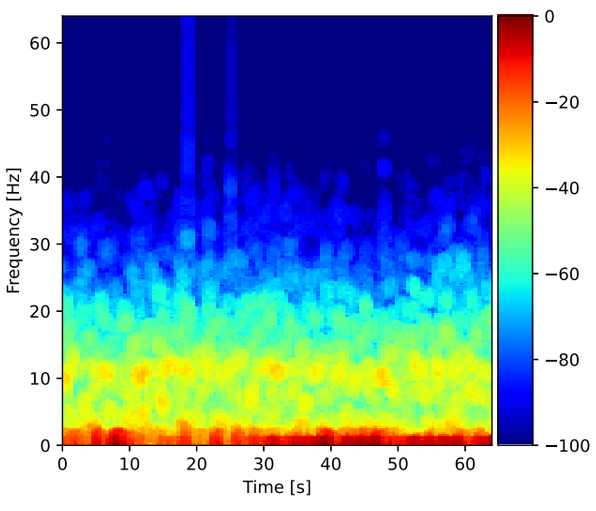
・Create a spectrogram using libtfr's multitaper method mfft_dpss().

In [76]:
```
###option spectrogram
times_adj2 = np.linspace(0, 64, 121)
len(times_adj2)
```

Out[76]:
```
121
```

In [77]:
```
###option spectrogram
fig, ax = plt.subplots(figsize=(5, 6))
image = ax.pcolormesh(times_adj2, freqs, P, cmap='jet')
ax.axis("image")
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [s]')
divider = make_axes_locatable(ax)
ax_cb = divider.new_horizontal(size="8%", pad=0.05)
fig.add_axes(ax_cb)
plt.colorbar(image, cax=ax_cb)
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [s]')
plt.savefig('data_eeg/spectral_analysis_3_fig_23.svg')
```

```
/Users/teijisw/opt/anaconda3/envs/Python36_2/lib/python3.6/site-packages/ipykernel_launcher.py:3: MatplotlibDeprecationWarning: shading='flat' w
hen X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shadi
ng='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  This is separate from the ipykernel package so we can avoid doing imports until
```



In [ ]: